Red Hat Fuse 7.2

Apache Camel コンポーネントリファレンス

Camel コンポーネントの設定リファレンス

Last Updated: 2023-04-29

Red Hat Fuse 7.2 Apache Camel コンポーネントリファレンス

Camel コンポーネントの設定リファレンス

法律上の通知

概要

Apache Camel には 100 を超えるコンポーネントがあり、コンポーネントはそれぞれ、細かな設定が可能です。このガイドでは、各コンポーネントの設定について説明します。

目次

第1章 コンポーネントの概要

この章では、Apache Camel で使用できるすべてのコンポーネントの概要を説明します。

1.1. コンテナーの種類

Red Hat Fuse には、さまざまなコンテナータイプがあり、Camel アプリケーションをデプロイできます。

- Spring Boot
- Apache Karaf
- JBoss Enterprise Application Platform (JBoss EAP)

さらに、Camel アプリケーションは **コンテナーレス** で実行できるので、Camel アプリケーションは特別なコンテナーなしで JVM 内で直接実行されます。

場合によっては、Fuse ではコンテナーごとにサポートされる Camel コンポーネントが異なります。これにはさまざまな理由がありますが、場合によっては、コンポーネントがすべてのコンテナータイプに適しているわけではありません。たとえば、**camel-ejb** コンポーネントは Java EE (つまり、JBoss EAP) 用に特別に設計されており、他のコンテナータイプではサポートできません。

1.2. サポートされるコンポーネント

以下のキーに注意してください。

記号	説明
✔	サポート対象
▮	サポートされないか、まだサポート対象ではない
非推奨	今後のリリースで削除される可能性があります

表1.1「Apache Camel コンポーネントサポートマトリックス」 では、各 Camel コンポーネントがサポートされるコンテナーについて、包括的に説明します。

表1.1 Apache Camel コンポーネントサポートマトリックス

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
activemq-camel	エンドポイント	✔	✔	✔	✔
camel-ahc	エンドポイント	✔	✔	✔	✔
camel-ahc-ws	エンドポイント	✔	✔	✔	✔
camel-ahc-wss	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-amqp	エンドポイント	✔	✔	✔	✔
camel-apns	エンドポイント	✔	✔	✔	✔
camel-asn1	データ形式	✔	✔	✔	✔
camel-asterisk	エンドポイント	✔	✔	✔	✔
camel-atmos	エンドポイント	✔	✔	▮	▮
camel-atmosphere-websocket	エンドポイント	✔	✔	✔	✔
camel-atom	エンドポイント	✔	✔	✔	✔
camel-atomix	エンドポイント	✔	✔	✔	✔
camel-avro	エンドポイント	✔	✔	✔	✔
camel-avro	データ形式	✔	✔	✔	✔
camel-aws	エンドポイント	✔	✔	✔	✔
camel-azure	エンドポイント	✔	✔	✔	✔
camel-bam	エンドポイント	**非推奨**	✔	✔	▮
camel-barcode	データ形式	✔	✔	✔	✔
camel-base64	データ形式	✔	✔	✔	✔
camel-bean	エンドポイント	✔	✔	✔	✔
camel-bean	言語	✔	✔	✔	✔
camel-bean-validator	エンドポイント	✔	✔	✔	✔
camel-beanio	データ形式	✔	✔	✔	✔
camel-beanstalk	エンドポイント	✔	✔	✔	▮
camel-binding	エンドポイント	**非推奨**	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-bindy	エンドポイント	✔	✔	✔	✔
camel-bindy	データ形式	✔	✔	✔	✔
camel-blueprint	エンドポイント	✔	▮	✔	▮
camel-bonita	エンドポイント	✔	▮	▮	▮
camel-boon	データ形式	✔	✔	✔	✔
camel-box	エンドポイント	✔	✔	✔	✔
camel-braintree	エンドポイント	✔	✔	✔	✔
camel-browse	エンドポイント	✔	✔	✔	✔
camel-cache	エンドポイント	**非推奨**	✔	✔	▮
camel-caffeine	エンドポイント	✔	✔	✔	✔
camel-castor	データ形式	**非推奨**	✔	✔	✔
camel-cdi	エンドポイント	✔	▮	**非推奨**	✔
camel-chronicle-engine	エンドポイント	✔	✔	✔	✔
camel-chunk	エンドポイント	✔	✔	✔	✔
camel-class	エンドポイント	✔	✔	✔	✔
camel-cm-sms	エンドポイント	✔	✔	✔	✔
camel-cmis	エンドポイント	✔	✔	✔	▮
camel-coap	エンドポイント	✔	✔	✔	✔
camel-cometd	エンドポイント	✔	✔	✔	✔
camel-constant	言語	✔	✔	✔	✔
camel-context	エンドポイント	**非推奨**	▮	▮	▮
camel-consul	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-controlbus	エンドポイント	✔	✔	✔	✔
camel-couchbase	エンドポイント	✔	✔	✔	✔
camel-couchdb	エンドポイント	✔	✔	✔	✔
camel-cql	エンドポイント	✔	✔	✔	✔
camel-crypto	エンドポイント	✔	✔	✔	✔
camel-crypto	データ形式	✔	✔	✔	✔
camel-crypto-cms	エンドポイント	✔	✔	✔	✔
camel-csv	データ形式	✔	✔	✔	✔
camel-cxf	エンドポイント	✔	✔	✔	✔
camel-cxf-transport	エンドポイント	✔	✔	✔	✔
camel-dataformat	エンドポイント	✔	✔	✔	✔
camel-dataset	エンドポイント	✔	✔	✔	✔
camel-digitalocean	エンドポイント	✔	✔	✔	✔
camel-direct	エンドポイント	✔	✔	✔	✔
camel-direct-vm	エンドポイント	✔	✔	✔	✔
camel-disruptor	エンドポイント	✔	✔	✔	✔
camel-dns	エンドポイント	✔	✔	✔	✔
camel-docker	エンドポイント	✔	✔	✔	✔
camel-dozer	エンドポイント	✔	✔	✔	✔
camel-drill	エンドポイント	✔	✔	✔	▮
camel-dropbox	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-eclipse		**非推奨**	▮	▮	▮
camel-ehcache	エンドポイント	✔	✔	✔	✔
camel-ejb	エンドポイント	✔	▮	▮	✔
camel-el	言語	**非推奨**	▮	▮	▮
camel-elasticsearch	エンドポイント	✔	✔	✔	✔
camel-elasticsearch5	エンドポイント	✔	✔	✔	✔
camel-elasticsearch-rest	エンドポイント	✔	✔	✔	▮
camel-elsql	エンドポイント	✔	✔	✔	✔
camel-etcd	エンドポイント	✔	✔	✔	✔
camel-eventadmin	エンドポイント	✔	▮	✔	▮
camel-exchangeProperty	言語	✔	✔	✔	✔
camel-exec	エンドポイント	✔	✔	✔	✔
camel-facebook	エンドポイント	✔	✔	✔	✔
camel-fhir	データ形式	✔	✔	✔	▮
camel-file	エンドポイント	✔	✔	✔	✔
camel-file	言語	✔	✔	✔	✔
camel-flatpack	エンドポイント	✔	✔	✔	✔
camel-flatpack	データ形式	✔	✔	✔	✔
camel-flink	エンドポイント	✔	✔	▮	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-fop	エンドポイント	✔	✔	✔	✔
camel-freemarker	エンドポイント	✔	✔	✔	✔
camel-ftp	エンドポイント	✔	✔	✔	✔
camel-gae	エンドポイント	**非推奨**	▮	▮	▮
camel-ganglia	エンドポイント	✔	✔	✔	▮
camel-geocoder	エンドポイント	✔	✔	✔	✔
camel-git	エンドポイント	✔	✔	✔	✔
camel-github	エンドポイント	✔	✔	✔	✔
camel-google-bigquery	エンドポイント	✔	✔	▮	✔
camel-google-calendar	エンドポイント	✔	✔	✔	✔
camel-google-drive	エンドポイント	✔	✔	✔	✔
camel-google-mail	エンドポイント	✔	✔	✔	✔
camel-google-pubsub	エンドポイント	✔	✔	✔	✔
camel-grape	エンドポイント	✔	✔	✔	▮
camel-groovy	言語	✔	✔	✔	✔
camel-groovy-dsl		**非推奨**	▮	▮	▮
camel-grpc	エンドポイント	✔	✔	✔	✔
camel-guava-eventbus	エンドポイント	✔	✔	✔	✔
camel-guice	エンドポイント	**非推奨**	✔	▮	▮
camel-gzip	データ形式	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-hawtdb	エンドポイント	**非推奨**	✔	✔	▮
camel-hazelcast	エンドポイント	✔	✔	✔	✔
camel-hbase	エンドポイント	✔	✔	▮	▮
camel-hdfs	エンドポイント	**非推奨**	✔	▮	▮
camel-hdfs2	エンドポイント	✔	✔	✔	✔
camel-header	言語	✔	✔	✔	✔
camel-headersmap		✔	✔	✔	✔
camel-hessian	データ形式	**非推奨**	✔	✔	✔
camel-hipchat	エンドポイント	✔	✔	✔	✔
camel-hl7	データ形式	✔	✔	✔	✔
camel-http	エンドポイント	**非推奨**	✔	✔	▮
camel-http4	エンドポイント	✔	✔	✔	✔
camel-hystrix	エンドポイント	✔	✔	✔	✔
camel-ibatis	エンドポイント	**非推奨**	▮	▮	▮
camel-ical	データ形式	✔	✔	✔	✔
camel-iec60870	エンドポイント	✔	✔	✔	✔
camel-ignite	エンドポイント	▮	▮	▮	▮
camel-imap	エンドポイント	✔	✔	✔	✔
camel-infinispan	エンドポイント	▮	▮	▮	▮
camel-influxdb	エンドポイント	✔	✔	✔	✔
camel-irc	エンドポイント	✔	✔	✔	✔
camel-ironmq	エンドポイント	▮	▮	▮	▮

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-jacksonxml	データ形式	✔	✔	✔	✔
camel-jasypt	エンドポイント	✔	✔	✔	✔
camel-javaspace	エンドポイント	**非推奨**	▮	▮	▮
camel-jaxb	データ形式	✔	✔	✔	✔
camel-jbpm	エンドポイント	▮	▮	▮	▮
camel-jcache	エンドポイント	✔	✔	✔	✔
camel-jcifs	エンドポイント	✔	▮	✔	▮
camel-jclouds	エンドポイント	✔	▮	✔	✔
camel-jcr	エンドポイント	✔	✔	✔	✔
camel-jdbc	エンドポイント	✔	✔	✔	✔
camel-jetty	エンドポイント	**非推奨**	**非推奨**	**非推奨**	▮
camel-jetty8	エンドポイント	▮	▮	▮	▮
camel-jetty9	エンドポイント	✔	✔	✔	▮
camel-jgroups	エンドポイント	✔	✔	✔	✔
camel-jibx	データ形式	✔	✔	✔	✔
camel-jing	エンドポイント	✔	✔	✔	✔
camel-jira	エンドポイント	✔	▮	▮	▮
camel-jms	エンドポイント	✔	✔	✔	✔
camel-jmx	エンドポイント	✔	✔	✔	✔
camel-jolt	エンドポイント	✔	✔	✔	✔
camel-josql	エンドポイント	**非推奨**	✔	✔	▮
camel-jpa	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-jsch	エンドポイント	✔	✔	✔	✔
camel-json-fastjson	データ形式	✔	✔	✔	✔
camel-json-gson	データ形式	✔	✔	✔	✔
camel-json-jackson	データ形式	✔	✔	✔	✔
camel-json-johnzon	データ形式	✔	✔	✔	✔
camel-json-validator	エンドポイント	✔	✔	✔	▮
camel-json-xstream	データ形式	✔	✔	✔	✔
camel-jsonpath	言語	✔	✔	✔	✔
camel-jt400	エンドポイント	✔	✔	✔	✔
camel-juel	エンドポイント	**非推奨**	✔	✔	▮
camel-jxpath	言語	**非推奨**	▮	▮	▮
camel-kafka	エンドポイント	✔	✔	✔	✔
camel-kestrel	エンドポイント	**非推奨**	✔	✔	▮
camel-krati	エンドポイント	**非推奨**	✔	✔	▮
camel-kubernetes	エンドポイント	✔	✔	✔	✔
camel-kura		✔	▮	✔	▮
camel-ldap	エンドポイント	✔	✔	✔	✔
camel-ldif	エンドポイント	✔	✔	✔	▮

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-leveldb	エンドポイント	✔	✔	✔	✔
camel-linkedin	エンドポイント	✔	✔	✔	✔
camel-log	エンドポイント	✔	✔	✔	✔
camel-lpr	エンドポイント	✔	✔	✔	✔
camel-lra		▮	▮	▮	▮
camel-lucene	エンドポイント	✔	✔	✔	✔
camel-lumberjack	エンドポイント	✔	✔	✔	✔
camel-lzf	データ形式	✔	✔	✔	✔
camel-master		✔	✔	✔	▮
camel-mail	エンドポイント	✔	✔	✔	✔
camel-metrics	エンドポイント	✔	✔	✔	✔
camel-milo	エンドポイント	✔	✔	✔	✔
camel-mime-multipart	データ形式	✔	✔	✔	✔
camel-mina	エンドポイント	**非推奨**	▮	✔	▮
camel-mina2	エンドポイント	✔	✔	✔	✔
camel-mllp	エンドポイント	✔	✔	✔	✔
camel-mock	エンドポイント	✔	✔	✔	✔
camel-mongodb	エンドポイント	✔	✔	✔	✔
camel-mongodb-gridfs	エンドポイント	✔	✔	✔	✔
camel-mongodb3	エンドポイント	✔	✔	✔	✔
camel-mqtt	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-msv	エンドポイント	✔	✔	✔	✔
camel-mustache	エンドポイント	✔	✔	✔	✔
camel-mvel	エンドポイント	✔	✔	✔	✔
camel-mvel	言語	✔	✔	✔	✔
camel-mybatis	エンドポイント	✔	✔	✔	✔
camel-nagios	エンドポイント	✔	✔	✔	▮
camel-nats	エンドポイント	✔	✔	✔	✔
camel-netty	エンドポイント	**非推奨**	✔	✔	▮
camel-netty-http	エンドポイント	**非推奨**	✔	✔	▮
camel-netty4	エンドポイント	✔	✔	✔	✔
camel-netty4-http	エンドポイント	✔	✔	✔	▮
camel-ognl	言語	✔	✔	✔	✔
camel-olingo2	エンドポイント	✔	✔	✔	✔
camel-olingo4	エンドポイント	✔	✔	✔	✔
camel-openshift	エンドポイント	**非推奨**	✔	✔	▮
camel-openstack	エンドポイント	✔	✔	✔	✔
camel-opentracing		✔	✔	✔	✔
camel-optaplanner	エンドポイント	✔	✔	✔	✔
camel-paho	エンドポイント	✔	✔	✔	✔
camel-paxlogging	エンドポイント	✔	▮	✔	▮
camel-pdf	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-pgevent	エンドポイント	✔	✔	✔	▮
camel-pgp	データ形式	✔	✔	✔	✔
camel-php	言語	**非推奨**	▮	▮	✔
camel-pop3	エンドポイント	✔	✔	✔	✔
camel-printer	エンドポイント	✔	✔	✔	✔
camel-properties	エンドポイント	✔	✔	✔	✔
camel-protobuf	データ形式	✔	✔	✔	✔
camel-pubnub	エンドポイント	✔	✔	✔	✔
camel-python	言語	**非推奨**	▮	▮	▮
camel-quartz	エンドポイント	**非推奨**	✔	✔	▮
camel-quartz2	エンドポイント	✔	✔	✔	✔
camel-quickfix	エンドポイント	✔	✔	✔	✔
camel-rabbitmq	エンドポイント	✔	✔	✔	✔
camel-reactive-streams	エンドポイント	✔	✔	✔	✔
camel-reactor		✔	✔	✔	✔
camel-ref	エンドポイント	✔	✔	✔	✔
camel-ref	言語	✔	✔	✔	✔
camel-rest	エンドポイント	✔	✔	✔	✔
camel-rest-api	エンドポイント	✔	✔	✔	✔
camel-rest-swagger	エンドポイント	✔	✔	✔	✔
camel-restlet	エンドポイント	✔	✔	✔	▮

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-ribbon		✔	✔	▮	✔
camel-rmi	エンドポイント	✔	✔	✔	✔
camel-routebox	エンドポイント	**非推奨**	✔	✔	▮
camel-rss	エンドポイント	✔	✔	✔	✔
camel-rss	データ形式	✔	✔	✔	✔
camel-ruby	言語	**非推奨**	▮	▮	▮
camel-rx	エンドポイント	**非推奨**	✔	✔	▮
camel-saga	エンドポイント	▮	▮	▮	▮
camel-salesforce	エンドポイント	✔	✔	✔	✔
camel-sap	エンドポイント	✔	✔	✔	✔
camel-sap-netweaver	エンドポイント	✔	✔	✔	✔
camel-saxon	エンドポイント	✔	✔	✔	✔
camel-scala	エンドポイント	**非推奨**	✔	✔	▮
camel-scheduler	エンドポイント	✔	✔	✔	✔
camel-schematron	エンドポイント	✔	✔	✔	✔
camel-scp	エンドポイント	✔	✔	✔	✔
camel-scr	エンドポイント	**非推奨**	✔	**非推奨**	▮
camel-script	エンドポイント	**非推奨**	**非推奨**	**非推奨**	**非推奨**
camel-seda	エンドポイント	✔	✔	✔	✔
camel-serialization	データ形式	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-servicenow	エンドポイント	✔	✔	✔	✔
camel-servlet	エンドポイント	✔	✔	✔	✔
camel-servletlistener	エンドポイント	**非推奨**	✔	✔	▮
camel-sftp	エンドポイント	✔	✔	✔	✔
camel-shiro	エンドポイント	✔	✔	✔	✔
camel-simple	言語	✔	✔	✔	✔
camel-sip	エンドポイント	✔	✔	✔	✔
camel-sjms	エンドポイント	✔	✔	✔	✔
camel-sjms2	エンドポイント	✔	✔	✔	✔
camel-slack	エンドポイント	✔	✔	✔	✔
camel-smpp	エンドポイント	✔	✔	✔	✔
camel-snakeyaml	エンドポイント	✔	✔	✔	✔
camel-snmp	エンドポイント	✔	✔	✔	✔
camel-soapjaxb	データ形式	✔	✔	✔	✔
camel-solr	エンドポイント	✔	✔	✔	▮
camel-spark	エンドポイント	✔	✔	▮	▮
camel-spark-rest	エンドポイント	✔	✔	▮	▮
camel-spel	言語	✔	✔	▮	✔
camel-splunk	エンドポイント	✔	✔	✔	✔
camel-spring	エンドポイント	✔	✔	✔	✔
camel-spring-batch	エンドポイント	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-spring-boot	エンドポイント	✔	✔	▮	▮
camel-spring-cloud		✔	✔	▮	▮
camel-spring-cloud-netflix		✔	✔	▮	▮
camel-spring-event	エンドポイント	✔	✔	▮	✔
camel-spring-integration	エンドポイント	✔	▮	▮	✔
camel-spring-javaconfig	エンドポイント	✔	✔	▮	✔
camel-spring-ldap	エンドポイント	✔	✔	✔	✔
camel-spring-redis	エンドポイント	✔	✔	✔	✔
camel-spring-security	エンドポイント	✔	✔	▮	✔
camel-spring-ws	エンドポイント	✔	✔	✔	✔
camel-sql	エンドポイント	✔	✔	✔	✔
camel-sql-stored	エンドポイント	✔	✔	✔	✔
camel-ssh	エンドポイント	✔	✔	✔	✔
camel-stax	エンドポイント	✔	✔	✔	✔
camel-stomp	エンドポイント	✔	✔	✔	✔
camel-stream	エンドポイント	✔	✔	✔	✔
camel-string	データ形式	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-string-template	エンドポイント	✔	✔	✔	✔
camel-stub	エンドポイント	✔	✔	✔	✔
camel-swagger	エンドポイント	**非推奨**	✔	**非推奨**	▮
camel-swagger-java	エンドポイント	✔	✔	✔	✔
camel-syslog	データ形式	✔	✔	✔	✔
camel-tagsoup	エンドポイント	✔	✔	✔	✔
camel-tarfile	データ形式	✔	✔	✔	✔
camel-telegram	エンドポイント	✔	✔	✔	✔
camel-thrift	エンドポイント	✔	✔	✔	✔
camel-thrift	データ形式	✔	✔	✔	✔
camel-tika	エンドポイント	✔	✔	✔	▮
camel-timer	エンドポイント	✔	✔	✔	✔
camel-tokenize	言語	✔	✔	✔	✔
camel-twilio	エンドポイント	✔	✔	✔	✔
camel-twitter	エンドポイント	✔	✔	✔	✔
camel-undertow	エンドポイント	✔	✔	✔	✔
camel-univocity-csv	データ形式	✔	✔	✔	✔
camel-univocity-fixed	データ形式	✔	✔	✔	✔
camel-univocity-tsv	データ形式	✔	✔	✔	✔
camel-urlrewrite	エンドポイント	**非推奨**	✔	✔	▮

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-validator	エンドポイント	✔	✔	✔	✔
camel-velocity	エンドポイント	✔	✔	✔	✔
camel-vertx	エンドポイント	✔	✔	✔	✔
camel-vm	エンドポイント	✔	✔	✔	✔
camel-weather	エンドポイント	✔	✔	✔	✔
camel-websocket	エンドポイント	✔	✔	✔	▮
camel-wordpress	エンドポイント	✔	✔	✔	▮
camel-xchange	エンドポイント	✔	✔	✔	▮
camel-xmlbeans	データ形式	**非推奨**	**非推奨**	**非推奨**	✔
camel-xmljson	データ形式	**非推奨**	**非推奨**	**非推奨**	**非推奨**
camel-xmlrpc	エンドポイント	▮	▮	▮	▮
camel-xmlrpc	データ形式	▮	▮	▮	▮
camel-xmlsecurity	エンドポイント	✔	✔	✔	✔
camel-xmpp	エンドポイント	✔	✔	✔	✔
camel-xpath	言語	✔	✔	✔	✔
camel-xquery	エンドポイント	✔	✔	✔	✔
camel-xquery	言語	✔	✔	✔	✔
camel-xslt	エンドポイント	✔	✔	✔	✔
camel-xstream	データ形式	✔	✔	✔	✔
camel-xtokenize	言語	✔	✔	✔	✔
camel-yaml-snakeyaml	データ形式	✔	✔	✔	✔

コンポーネント	タイプ	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-yammer	エンドポイント	✔	✔	✔	✔
camel-yql	エンドポイント	✔	✔	✔	▮
camel-zendesk	エンドポイント	✔	✔	▮	✔
camel-zip	データ形式	✔	✔	✔	✔
camel-zipfile	データ形式	✔	✔	✔	✔
camel-zipkin	エンドポイント	✔	✔	✔	✔
camel-zookeeper	エンドポイント	✔	✔	✔	✔
camel-zookeeper-master	エンドポイント	✔	✔	✔	✔

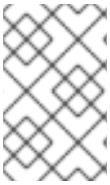
第2章 ACTIVEMQ

ACTIVEMQ コンポーネント

ActiveMQ コンポーネントを使用すると、メッセージを JMS キューまたはトピックに送信するか、Apache ActiveMQ を使用して JMS キューまたはトピックからメッセージを消費できます。

このコンポーネントは、169章*JMS コンポーネント* をベースにしており、Spring の JMS サポートを宣言型トランザクションに、また Spring の **JmsTemplate** を送信に、**MessageListenerContainer** を消費に使用します。169章*JMS コンポーネント* のコンポーネントのオプションはすべて、このコンポーネントに該当します。

このコンポーネントを使用するには、クラスパスに **activemq.jar** または **activemq-core.jar** があり、**camel-core.jar**、**camel-spring.jar**、**camel-jms.jar** などの Apache Camel 依存関係があることを確認してください。



トランザクションとキャッシング

JMS でトランザクションを使用している場合は、パフォーマンスに影響を与える可能性があるため、JMS ページの下の **トランザクションとキャッシュレベル** セクションを参照してください。

URI 形式

```
activemq:[queue:|topic:]destinationName
```

ここでは、**destinationName** は ActiveMQ キューまたはトピック名に置き換えます。デフォルトでは、**destinationName** はキュー名として解釈されます。たとえば、キューに接続するには、**FOO.BAR** を次のように使用します。

```
activemq:FOO.BAR
```

必要に応じて、オプションの **queue:** 接頭辞を含めることができます。

```
activemq:queue:FOO.BAR
```

トピックに接続するには、**topic:** 接頭辞を含める必要があります。たとえば、トピック **Stocks.Prices** に接続するには、次を使用します。

```
activemq:topic:Stocks.Prices
```

オプション

これらのオプションはすべてこのオプションに該当するので、169章*JMS コンポーネント* コンポーネントのオプションを参照してください。

CAMEL ON EAP デプロイメント

このコンポーネントは、Red Hat JBoss Enterprise Application Platform (JBoss EAP) コンテナー上で簡素化されたデプロイメントモデルを提供する Camel on EAP (Wildfly Camel) フレームワークによってサポートされます。

組み込みブローカーまたは外部ブローカーのいずれかで動作するように ActiveMQ Camel コンポーネントを設定できます。JBoss EAP コンテナーにブローカーを埋め込むには、EAP コンテナー設定ファイルで ActiveMQ リソースアダプターを設定します。詳細は、ActiveMQ リソースアダプターの設定 を参照してください。

接続ファクトリーの設定

次の テストケース は、ActiveMQ への接続に使用される brokerURL を指定しながら、**activeMQComponent ()** メソッド を使用して ActiveMQComponent を CamelContext に追加する方法を示しています。

```
camelContext.addComponent("activemq", activeMQComponent("vm://localhost?
broker.persistent=false"));
```

SPRING XML を使用した接続ファクトリーの設定

次のように、ActiveMQComponent で ActiveMQ ブローカー URL を設定できます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
  </camelContext>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="brokerURL" value="tcp://somehost:61616"/>
  </bean>

</beans>
```

接続プーリングの使用

Camel を使用して ActiveMQ ブローカーに送信する場合は、プールされた接続ファクトリーを使用して、JMS 接続、セッション、およびプロデューサーを効率的にプールするように処理することを推奨します。これは ActiveMQ Spring Support ページに記載されています。

Maven を使用して、Jencks AMQ プールを取得できます。

```
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
    <version>5.3.2</version>
  </dependency>
```

次に、**activemq** コンポーネントを次のように設定します。

```
    <bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
```

```
        <property name="brokerURL" value="tcp://localhost:61616" />
    </bean>

    <bean id="pooledConnectionFactory"
class="org.apache.activemq.pool.PooledConnectionFactory" init-method="start" destroy-
method="stop">
        <property name="maxConnections" value="8" />
        <property name="connectionFactory" ref="jmsConnectionFactory" />
    </bean>

    <bean id="jmsConfig" class="org.apache.camel.component.jms.JmsConfiguration">
        <property name="connectionFactory" ref="pooledConnectionFactory"/>
        <property name="concurrentConsumers" value="10"/>
    </bean>

    <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
        <property name="configuration" ref="jmsConfig"/>
    </bean>
```

注記

プールされた接続ファクトリーの **init** メソッドと **destroy** メソッドに注目してください。これは、接続プールが適切に開始および終了されるようにするために重要です。

PooledConnectionFactory は、同時に使用される最大 8 つの接続を含む接続プールを作成します。各接続は、多くのセッションで共有できます。接続ごとのセッションの最大数を設定するために使用できる **maxActive** という名前のオプションがあります。デフォルト値は **500** です。**ActiveMQ 5.7** 以降、このオプションはその目的をさらに反映するように名前が変更され、**maxActiveSessionPerConnection** という名前になりました。**concurrentConsumers** が **maxConnections** よりも高い値に設定されていることに注意してください。各コンシューマーがセッションを使用していて、セッションが同じ接続を共有できるので、これは問題ありません。この例では、同時に 8 * 500 = 4000 のアクティブなセッションを指定できます。

ルートでの MESSAGELISTENER POJO の呼び出し

ActiveMQ コンポーネントは、JMS MessageListener から Processor へのヘルパー Type Converter も提供します。これは、41章*Bean コンポーネント* コンポーネントは、任意のルート内で任意の JMS MessageListener Bean を直接呼び出すことができます。

たとえば、次のように JMS で MessageListener を作成できます。

```
public class MyListener implements MessageListener {
    public void onMessage(Message jmsMessage) {
        // ...
    }
}
```

次に、以下のようにルートで使用します。

```
from("file://foo/bar").
  bean(MyListener.class);
```

つまり、任意の Apache Camel コンポーネントを再利用して、それらを JMS **MessageListener** POJO に簡単に統合できます。

ACTIVEMQ 宛先オプションの使用

ActiveMQ 5.6 以降で利用可能

"destination." の接頭辞を使用して、エンドポイント uri で 宛先オプション を設定できます。たとえば、コンシューマーを排他的とマークし、そのプリフェッチサイズを 50 に設定するには、次のようにします。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file://src/test/data?noop=true"/>
    <to uri="activemq:queue:foo"/>
  </route>
  <route>
    <!-- use consumer.exclusive ActiveMQ destination option, notice we have to prefix with destination.
-->
    <from uri="activemq:foo?
destination.consumer.exclusive=true&amp;destination.consumer.prefetchSize=50"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

アドバイザリーメッセージの使用

ActiveMQ は、消費可能なトピックに配置される アドバイザリーメッセージ を生成できます。このようなメッセージは、遅いコンシューマーを検出した場合にアラートを送信したり、統計 (1 日に生成されるメッセージの数など) を作成したりするのに役立ちます。 次の Spring DSL の例は、トピックからメッセージを読み取る方法を示しています。

```
<route>
 <from uri="activemq:topic:ActiveMQ.Advisory.Connection?mapJmsMessage=false" />
 <convertBodyTo type="java.lang.String"/>
 <transform>
     <simple>${in.body}&#13;</simple>
 </transform>
 <to uri="file://data/activemq/?fileExist=Append&ileName=advisoryConnection-
${date:now:yyyyMMdd}.txt" />
</route>
```

キューでメッセージを消費すると、data/activemq フォルダーの下に次のファイルが表示されます。

advisoryConnection-20100312.txt advisoryProducer-20100312.txt

および含まれる文字列:

```
    ActiveMQMessage {commandId = 0, responseRequired = false, messageId = ID:dell-charles-
3258-1268399815140
    -1:0:0:0:221, originalDestination = null, originalTransactionId = null, producerId = ID:dell-charles-
    3258-1268399815140-1:0:0:0, destination = topic://ActiveMQ.Advisory.Connection, transactionId
= null,
    expiration = 0, timestamp = 0, arrival = 0, brokerInTime = 1268403383468, brokerOutTime =
1268403383468,
    correlationId = null, replyTo = null, persistent = false, type = Advisory, priority = 0, groupID = null,
    groupSequence = 0, targetConsumerId = null, compressed = false, userID = null, content = null,
```

```
    marshalledProperties = org.apache.activemq.util.ByteSequence@17e2705, dataStructure =
ConnectionInfo
    {commandId = 1, responseRequired = true, connectionId = ID:dell-charles-3258-1268399815140-
2:50,
    clientId = ID:dell-charles-3258-1268399815140-14:0, userName = , password = *****,
    brokerPath = null, brokerMasterConnector = false, manageable = true, clientMaster = true},
    redeliveryCounter = 0, size = 0, properties = {originBrokerName=master, originBrokerId=ID:dell-
charles-
    3258-1268399815140-0:0, originBrokerURL=vm://master}, readOnlyProperties = true,
readOnlyBody = true,
    droppable = false}
```

コンポーネント JAR の取得

以下の依存関係が必要です。

- **activemq-camel**

ActiveMQ は、ActiveMQ プロジェクト でリリースされた 169章*JMS コンポーネント* コンポーネントの拡張です。

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-camel</artifactId>
  <version>5.6.0</version>
</dependency>
```

第3章 AHC コンポーネント

Camel バージョン 2.8 以降で利用可能

ahc: コンポーネントは、外部 HTTP リソースを消費するための HTTP ベースのエンドポイントを提供します (HTTP を使用して外部サーバーを呼び出すクライアントとして)。
このコンポーネントは Async Http Client ライブラリーを使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ahc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

3.1. URI 形式

```
ahc:http://hostname[:port][/resourceUri][?options]
ahc:https://hostname[:port][/resourceUri][?options]
```

デフォルトでは、HTTP にはポート 80、HTTPS には 443 を使用します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

3.2. AHCENDPOINT オプション

AHC エンドポイントは、URI 構文を使用して設定されます。

```
ahc:httpUri
```

パスおよびクエリーパラメーターを使用します。

3.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpUri	**必須** 使用する URI (例:http://hostname:port/path)		URI

3.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeEndpoint (producer)	オプションが true の場合、Exchange.HTTP_URI ヘッダーは無視され、エンドポイントの URI を要求に使用します。また、throwExceptionOnFailure を false に設定して、AhcProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
bufferSize (producer)	Camel と AHC クライアントの間でデータを転送するときに使用される初期メモリー内バッファーサイズ。	4096	int
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを定義します。このパラメーターはデフォルトで false です)。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
headerFilterStrategy (producer)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に AhcOperationFailedException の出力を無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました (例: Jetty または Servlet Camel のコンポーネント)。プロデューサ側では、AhcOperationFailedException の代わりに、例外がデシリアライズされ、そのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
binding (advanced)	AHC と Camel 間のバインド方法を制御できるカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	カスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように AsyncHttpClient を設定するには。		AsyncHttpClientConfig

名前	説明	デフォルト	タイプ
clientConfigOptions (advanced)	マップのキー/値を使用して AsyncHttpClientConfig を設定します。		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
clientConfigRealmOptions (security)	マップのキー/値を使用して AsyncHttpClientConfig レルムを設定します。		Map
sslContextParameters (security)	レジストリー内の org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、設定されている SSLContextParameters をコンポーネントレベルでオーバーライドします。JSSE 設定ユーティリティーの使用を参照してください。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションを介して提供される SSL/TLS 設定オプションが上書きされることに注意してください。		SSLContextParameters

3.3. AHCCOMPONENT オプション

AHC コンポーネントは、以下に示す 8 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
client (advanced)	カスタム AsyncHttpClient を使用します。		AsyncHttpClient
binding (advanced)	AHC と Camel 間のバインド方法を制御できるカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	カスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように AsyncHttpClient を設定するには。		AsyncHttpClientConfig
sslContextParameters (security)	レジストリー内の org.apache.camel.util.jsse.SSLContextParameters への参照。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションを介して提供される SSL/TLS 設定オプションが上書きされることに注意してください。		SSLContextParameters

名前	説明	デフォルト	タイプ
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用するときに Java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AhcComponent に任意のオプションを設定すると、それらのオプションが作成中の **AhcEndpoints** に伝播されることに注意してください。ただし、**AhcEndpoint** はカスタムオプションを設定/オーバーライドすることもできます。エンドポイントに設定されたオプションは、常に **AhcComponent** のオプションよりも優先されます。

3.4. メッセージヘッダー

名前	タイプ	説明
Exchange.HTTP_URI	**String**	呼び出す URI。エンドポイントで直接設定された既存の URI をオーバーライドします。
Exchange.HTTP_PATH	**String**	リクエスト URI のパス。ヘッダーは、HTTP_URI でリクエスト URI を構築するために使用されます。パスが / で始まる場合には、http プロデューサーは Exchange.HTTP_BASE_URI ヘッダーまたは **exchange.getFromEndpoint ().getEndpointUri ();** に基づいて相対パスを見つけようとします。
Exchange.HTTP_QUERY	**String**	**Camel 2.11 以降:** URI パラメーター。エンドポイントに直接設定された既存の URI パラメーターをオーバーライドします。

名前	タイプ	説明
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_CHARACTER_ENCODING	**String**	文字エンコーディング。
Exchange.CONTENT_TYPE	**String**	HTTP コンテンツタイプ。**text/html** などのコンテンツタイプを提供するために、IN メッセージと OUT メッセージの両方に設定されます。
Exchange.CONTENT_ENCODING	**String**	HTTP コンテンツエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために、IN メッセージと OUT メッセージの両方に設定されます。

3.5. メッセージボディー

Camel は、外部サーバーからの HTTP レスポンスを OUT ボディに保存します。IN メッセージのすべてのヘッダーが OUT メッセージにコピーされるため、ヘッダーはルーティング中に保持されます。さらに、Camel は HTTP 応答ヘッダーも OUT メッセージヘッダーに追加します。

3.6. レスポンスコード

Camel は HTTP 応答コードに従って処理します。

- 応答コードは 100..299 の範囲で、Camel はそれを成功応答と見なします。
- 応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答と見なし、情報とともに **AhcOperationFailedException** を出力します。
- 応答コードが 400+ の場合に、Camel はこれを外部サーバーの障害と見なし、情報とともに **AhcOperationFailedException** を出力します。
  throwExceptionOnFailure

オプション **throwExceptionOnFailure** を **false** に設定して、失敗した応答コードに対して **AhcOperationFailedException** が出力されないようにすることができます。これにより、リモートサーバーからの応答を取得できます。

3.7. AHCOPERATIONFAILEDEXCEPTION

この例外には、次の情報が含まれています。

- HTTP ステータスコード
- HTTP ステータス行 (ステータスコードのテキスト)
- サーバーがリダイレクトを返した場合は、ロケーションをリダイレクトします
- サーバーがレスポンスとして本文を提供した場合、**java.lang.String** としてのレスポンス本文

3.8. GET または POST を使用した呼び出し

次のアルゴリズムを使用して、**GET** または **POST** HTTP メソッドを使用するかどうかを決定します。
1.ヘッダーで指定するメソッドを使用します。
2.ヘッダーにクエリー文字列が指定されている場合は **GET**。
3.エンドポイントがクエリー文字列で設定されている場合は **GET**。
4.送信するデータがある場合は **POST** (本文が null ではない)。
5.それ以外の場合は **GET**。

3.9. 呼び出す URI の設定

HTTP プロデューサーの URI は、エンドポイント URI から直接設定できます。以下のルートでは、Camel は HTTP を使用して外部サーバー **oldhost** を呼び出します。

```
from("direct:start")
        .to("ahc:http://oldhost");
```

同等の Spring の例:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="ahc:http://oldhost"/>
  </route>
</camelContext>
```

メッセージにキー **Exchange.HTTP_URI** を含むヘッダーを追加することで、HTTP エンドポイント URI をオーバーライドできます。

```
from("direct:start")
    .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
    .to("ahc:http://oldhost");
```

3.10. URI パラメーターの設定

ahc プロデューサーは、HTTP サーバーに送信される URI パラメーターをサポートしています。URI パラメーターは、エンドポイント URI に直接設定するか、メッセージのキー **Exchange.HTTP_QUERY** を含むヘッダーとして設定できます。

```
from("direct:start")
        .to("ahc:http://oldhost?order=123&detail=short");
```

または、ヘッダーで提供されるオプション:

```
from("direct:start")
            .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
        .to("ahc:http://oldhost");
```

3.11. HTTP プロデューサーに HTTP メソッドを設定する方法

HTTP コンポーネントは、メッセージヘッダーを設定することにより、HTTP 要求メソッドを設定する方法を提供します。以下に例を示します。

```
from("direct:start")
            .setHeader(Exchange.HTTP_METHOD, constant("POST"))
        .to("ahc:http://www.google.com")
            .to("mock:results");
```

同等の Spring の例:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelHttpMethod">
        <constant>POST</constant>
    </setHeader>
    <to uri="ahc:http://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

3.12. 文字セットの設定

POST を使用してデータを送信している場合は、**Exchange** プロパティーを使用して **charset** を設定できます。

```
exchange.setProperty(Exchange.CHARSET_NAME, "iso-8859-1");
```

3.12.1. エンドポイント URI からの URI パラメーター

このサンプルには、Web ブラウザーに入力したものとまったく同じ完全な URI エンドポイントがあります。もちろん、Web ブラウザーと同じように **&** 文字を区切り文字として使用して、複数の URI パラメーターを設定できます。Camel はここでトリックを行いません。

```
// we query for Camel at the Google page
template.sendBody("ahc:http://www.google.com/search?q=Camel", null);
```

3.12.2. メッセージからの URI パラメーター

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("ahc:http://www.google.com/search", null, headers);
```

上記のヘッダー値では、**?** を接頭辞として付けるべきでは **ない** ことに注意してください。**&** 文字を使用して、通常どおりパラメーターを区切ることができます。

3.12.3. 応答コードの取得

Exchange.HTTP_RESPONSE_CODE を使用して Out メッセージヘッダーから値を取得して、AHC コンポーネントから HTTP 応答コードを取得できます。

```
Exchange exchange = template.send("ahc:http://www.google.com/search", new Processor() {
        public void process(Exchange exchange) throws Exception {
            exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
        }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

3.13. ASYNCHTTPCLIENT の設定

AsyncHttpClient クライアントは、**AsyncHttpClientConfig** を使用してクライアントを設定します。のドキュメントを参照してください。詳細は 非同期 HTTP クライアント を参照してください。

Camel 2.8 では、設定は **AsyncHttpClientConfig.Builder** によって提供されるビルダーパターンの使用に制限されています。Camel 2.8 では、**AsyncHttpClientConfig は** getter/setter をサポートしていないため、Spring Bean スタイル (XML ファイルの <bean> タグなど) を使用して作成/設定するのは簡単ではありません。

以下の例は、ビルダーを使用して、**AhcComponent** で設定する **AsyncHttpClientConfig** を作成する方法を示しています。

Camel 2.9 では、AHC コンポーネントは Async HTTP ライブラリー 1.6.4 を使用します。今回の新しいバージョンでは、プレーン Bean スタイルの設定のサポートが追加されています。**AsyncHttpClientConfigBean** クラスは、**AsyncHttpClientConfig** で使用可能な設定オプションの getter と setter を提供します。**AsyncHttpClientConfigBean** のインスタンスは、AHC コンポーネントに直接渡すか、**clientConfig** URI パラメーターを使用してエンドポイント URI で参照できます。

また、Camel 2.9 では、設定オプションを URI で直接設定する機能も利用できます。clientConfig で始まる URI パラメーター。**AsyncHttpClientConfig** のさまざまな設定可能なプロパティーを設定するために使用できます。エンドポイント URI で指定されたプロパティーは、clientConfig URI パラメーターで参照される設定で指定されたプロパティーとマージされますが、"clientConfig." パラメーターを使用して設定されたプロパティーが優先されます。参照される **AsyncHttpClientConfig** インスタンスは、エンドポイントごとに常にコピーされるため、1 つのエンドポイントの設定は、以前に作成されたエンドポイントの設定とは無関係のままになります。次の例は、"clientConfig." タイプの URI を使用して AHC コンポーネントを設定する方法を示しています。URI パラメーターを入力します。

```
from("direct:start")
    .to("ahc:http://localhost:8080/foo?
clientConfig.maxRequestRetry=3&clientConfig.followRedirects=true")
```

3.14. SSL サポート (HTTPS)

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、AHC コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、AHC コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

AhcComponent component = context.getComponent("ahc", AhcComponent.class);
component.setSslContextParameters(scp));
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="ahc:https://localhost/foo?sslContextParameters=#sslContextParameters"/>
...
```

3.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Jetty
- HTTP

- HTTP4

第4章 AHC WEBSOCKET コンポーネント

Camel バージョン 2.14 以降で利用可能

ahc-ws コンポーネントは、Websocket を介して外部サーバーと通信するクライアントに、(外部サーバーへの Websocket 接続を開くクライアントとして) Websocket ベースのエンドポイントを提供します。
このコンポーネントは Async Http Client ライブラリーを使用する AHC コンポーネントを使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ahc-ws</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

4.1. URI 形式

```
ahc-ws://hostname[:port][/resourceUri][?options]
ahc-wss://hostname[:port][/resourceUri][?options]
```

デフォルトでは、ahc-ws にはポート 80 を使用し、ahc-wss には 443 を使用します。

4.2. AHC-WS オプション

AHC-WS コンポーネントは AHC コンポーネントに基づいているため、AHC コンポーネントのさまざまな設定オプションを使用できます。

AHC Websocket コンポーネントは、以下に示す 8 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
client (advanced)	カスタム AsyncHttpClient を使用します。		AsyncHttpClient
binding (advanced)	AHC と Camel 間のバインド方法を制御できるカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	カスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように AsyncHttpClient を設定するには。		AsyncHttpClientConfig
sslContextParameters (security)	レジストリー内の org.apache.camel.util.jsse.SSLContextParameters への参照。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションを介して提供される SSL/TLS 設定オプションが上書きされることに注意してください。		SSLContextParameters

名前	説明	デフォルト	タイプ
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用するときに Java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AHC Websocket エンドポイントは、URI 構文を使用して設定されます。

```
ahc-ws:httpUri
```

パスおよびクエリーパラメーターを使用します。

4.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpUri	**必須** 使用する URI (例:http://hostname:port/path)		URI

4.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeEndpoint (common)	オプションが true の場合、Exchange.HTTP_URI ヘッダーは無視され、エンドポイントの URI を要求に使用します。また、throwExceptionOnFailure を false に設定して、AhcProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
bufferSize (common)	Camel と AHC クライアントの間でデータを転送するときに使用される初期メモリー内バッファーサイズ。	4096	int
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
throwExceptionOnFailure (common)	リモートサーバーからの応答が失敗した場合に AhcOperationFailedException の出力を無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (common)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました (例: Jetty または Servlet Camel のコンポーネント)。プロデューサ側では、AhcOperationFailedException の代わりに、例外がデシリアライズされ、そのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendMessageOnError (consumer)	Web ソケットリスナーがエラーを受信した場合にメッセージを送信するかどうか。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを定義します。このパラメーターはデフォルトで false です)。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
useStreaming (producer)	ストリーミングを有効にして、データを複数のテキストフラグメントとして送信します。	false	boolean
binding (advanced)	AHC と Camel 間のバインド方法を制御できるカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	カスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように AsyncHttpClient を設定するには。		AsyncHttpClientConfig
clientConfigOptions (advanced)	マップのキー/値を使用して AsyncHttpClientConfig を設定します。		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
clientConfigRealmOptions (security)	マップのキー/値を使用して AsyncHttpClientConfig レルムを設定します。		Map
sslContextParameters (security)	レジストリー内の org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、設定されている SSLContextParameters をコンポーネントレベルでオーバーライドします。JSSE 設定ユーティリティーの使用を参照してください。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションを介して提供される SSL/TLS 設定オプションが上書きされることに注意してください。		SSLContextParameters

4.3. WEBSOCKET を介したデータの書き込みと読み取り

ahc-ws エンドポイントは、エンドポイントがそれぞれプロデューサーまたはコンシューマーとして設定されているかどうかに応じて、ソケットにデータを書き込むか、ソケットから読み取ることができます。

4.4. データの書き込みまたは読み取りのための URI の設定

以下のルートでは、Camel は指定された websocket 接続に書き込みます。

```
from("direct:start")
        .to("ahc-ws://targethost");
```

同等の Spring の例:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="ahc-ws://targethost"/>
  </route>
</camelContext>
```

以下のルートでは、Camel は指定された websocket 接続から読み取ります。

```
from("ahc-ws://targethost")
        .to("direct:next");
```

同等の Spring の例:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="ahc-ws://targethost"/>
    <to uri="direct:next"/>
  </route>
</camelContext>
```

4.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AHC
- Atmosphere-Websocket

第5章 AMQP コンポーネント

Camel バージョン 1.2 以降で利用可能

amqp: コンポーネントは、Qpid プロジェクトの JMS クライアント API を使用して AMQP 1.0 プロトコル をサポートします。AMQP 0.9 (特に RabbitMQ) を使用する場合は、 Camel RabbitMQ コンポーネントにも関心があるかもしれません。Camel 2.17.0 より前の AMQP コンポーネントは AMQP 0.9 以降をサポートしていましたが、Camel 2.17.0 以降は AMQP 1.0 のみをサポートしていることに注意してください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-amqp</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>
```

5.1. URI 形式

```
amqp:[queue:|topic:]destinationName[?options]
```

5.2. AMQP オプション

宛先名の後に、JMS コンポーネントのさまざまな設定オプションをすべて指定できます。

AMQP コンポーネントは、以下に示す 80 のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 JMS 設定を使用します。		JmsConfiguration
acceptMessagesWhileStopping (consumer)	コンシューマーが停止中にメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止するが、キューにメッセージが入れられている場合は、このオプションを有効にすることを検討してください。このオプションが false の場合は、JMS ルートを停止すると、メッセージが拒否される可能性があり、JMS ブローカは再配信を試行する必要がありますが、これも拒否される可能性があり、最終的にメッセージはJMS ブローカー上のデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることをお勧めします。	false	boolean

名前	説明	デフォルト	タイプ
allowReplyManagerQuick Stop (consumer)	JmsConfigurationisAcceptMessagesWhileStopping が有効で、org.apache.camel.CamelContext が現在停止している場合に、要求/応答メッセージングのリプライマネージャーで使用される DefaultMessageListenerContainer が、DefaultMessageListenerContainer.runningAllowed フラグを迅速に停止できるようにするかどうか。このクイック停止機能は、通常の JMS コンシューマーではデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
acknowledgementMode (consumer)	整数として定義された JMS 確認応答モード。確認モードにベンダー固有の拡張を設定できます。通常のモードでは、代わりに acknowledgementModeName を使用することをお勧めします。		int
eagerLoadingOfProperties (consumer)	メッセージが読み込まれるとすぐに JMS プロパティーの先行読み込みを有効にします。これは、JMS プロパティーが必要ない場合があるため一般的に非効率的ですが、基盤となる JMS プロバイダーと JMS プロパティーの使用に関する問題を早期に発見できる場合があります。	false	boolean
acknowledgementModeName (consumer)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	String
autoStartup (consumer)	コンシューマーコンテナーを自動起動するかどうかを指定します。	true	boolean
cacheLevel (consumer)	基礎となる JMS リソースの ID によってキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	基礎となる JMS リソースのキャッシュレベルを名前で設定します。可能な値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、および CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring のドキュメントとトランザクションキャッシュレベルを参照してください。	CACHE_AUTO	String

名前	説明	デフォルト	タイプ
replyToCacheLevelName (producer)	JMS を介して要求/応答を行うときに、応答コンシューマーのキャッシュレベルを名前で設定します。このオプションは、固定応答キュー (一時的ではない) を使用する場合にのみ適用されます。Camel はデフォルトで次を使用します: 排他的または replyToSelectorName と共有の CACHE_CONSUMER。そして、replyToSelectorName なしで共有するための CACHE_SESSION。IBM WebSphere などの一部の JMS ブローカーは、replyToCacheLevelName=CACHE_NONE を機能させるために設定する必要がある場合があります。注: 一時キューを使用する場合、CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などのより高い値を使用する必要があります。		String
clientId (common)	使用する JMS クライアント ID を設定します。この値を指定する場合は、一意である必要があり、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続的なトピックサブスクリプションの場合にのみ必要です。Apache ActiveMQ を使用している場合は、代わりに仮想トピックを使用することをお勧めします。		String
concurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーのデフォルト数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。	1	int
replyToConcurrent Consumers (producer)	JMS を介して要求/応答を行うときの同時コンシューマーのデフォルト数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。	1	int
connectionFactory (common)	使用する接続ファクトリー。コンポーネントまたはエンドポイントで接続ファクトリーを設定する必要があります。		ConnectionFactory
username (security)	ConnectionFactory で使用するユーザー名。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String

名前	説明	デフォルト	タイプ
password (security)	ConnectionFactory で使用するパスワード。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
deliveryPersistent (producer)	デフォルトで永続配信を使用するかどうかを指定します。	true	boolean
deliveryMode (producer)	使用する配信モードを指定します。可能な値は、javax.jms.DeliveryMode によって定義された値です。NON_PERSISTENT = 1 および PERSISTENT = 2。		Integer
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		String
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
errorHandler (advanced)	Message の処理中にキャッチされない例外が出力された場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ログレベルとスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングするよりも設定がはるかに簡単になります。		ErrorHandler
errorHandlerLoggingLevel (ロギング)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (ロギング)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean

名前	説明	デフォルト	タイプ
explicitQosEnabled (producer)	メッセージの送信時に、deliveryMode、priority、または timeToLive のサービス品質を使用する必要があるかどうかを設定します。このオプションは、Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で動作し、Camel In メッセージヘッダーから排他的に QoS プロパティーを読み取る preserveMessageQos オプションとは対照的です。	false	boolean
exposeListenerSession (consumer)	メッセージを消費するときにリスナーセッションを公開するかどうかを指定します。	false	boolean
idleTaskExecutionLimit (advanced)	実行中にメッセージを受信していない、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスクに受信を任せます (動的スケジューリングの場合。maxConcurrentConsumers 設定を参照してください)。Spring から入手できる追加のドキュメントがあります。	1	int
idleConsumerLimit (advanced)	常にアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
maxConcurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーの最大数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToMaxConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。		int
replyToMaxConcurrent Consumers (producer)	JMS を介した要求/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。		int
replyOnTimeoutToMax ConcurrentConsumers (producer)	JMS 経由の要求/応答を使用するときにタイムアウトが発生したときに、ルーティングを継続するための同時コンシューマーの最大数を指定します。	1	int

名前	説明	デフォルト	タイプ
maxMessagesPerTask (advanced)	タスクあたりのメッセージ数。-1 は無制限です。同時コンシューマーの範囲 (例: min max) を使用する場合、このオプションを使用して値を 100 などに設定し、必要な作業が少ない場合にコンシューマーが縮小する速度を制御できます。	-1	int
messageConverter (advanced)	カスタム Spring org.springframework.jms.support.converter.MessageConverter を使用して、javax.jms.Message との間でどのようにマッピングするかを制御できるようにします。		MessageConverter
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。	true	boolean
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合には、これらのメッセージはメッセージ ID を null に設定する必要があります。プロバイダーがヒントを無視する場合には、メッセージ ID は通常の一意の値に設定する必要があります	true	boolean
messageTimestampEnabled (advanced)	メッセージの送信時にデフォルトでタイムスタンプを有効にするかどうかを指定します。これは、JMS ブローカへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合には、これらのメッセージのタイムスタンプはゼロに設定する必要があります。プロバイダーがヒントを無視する場合には、タイムスタンプを通常の値に設定する必要があります	true	boolean
alwaysCopyMessage (producer)	true の場合、メッセージがプロデューサーに渡されて送信されると、Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、状況によってはメッセージのコピーが必要です (また、replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します)。	false	boolean
useMessageIDAsCorrelationID (advanced)	InOut メッセージの JMSCorrelationID として JMSMessageID を常に使用するかどうかを指定します。	false	boolean

名前	説明	デフォルト	タイプ
priority (producer)	1より大きい値は、送信時のメッセージの優先度を指定します (0 が最低の優先度で、9 が最高の優先度です)。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
pubSubNoLocal (advanced)	独自の接続によってパブリッシュされたメッセージの配信を禁止するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージ受信のタイムアウト (ミリ秒単位)。	1000	long
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	long
taskExecutor (consumer)	メッセージを消費するためのカスタムタスクエグゼキュータを指定できます。		TaskExecutor
timeToLive (producer)	メッセージの送信時に、メッセージの有効期限をミリ秒単位で指定します。	-1	long
取引済み (取引)	トランザクションモードを使用するかどうかを指定します	false	boolean
lazyCreateTransaction Manager (トランザクション)	true の場合、オプション transacted=true のときに transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (トランザクション)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (トランザクション)	使用するトランザクションの名前。		String
transactionTimeout (トランザクション)	トランザクションモードを使用している場合の、トランザクションのタイムアウト値 (秒単位)。	-1	int

名前	説明	デフォルト	タイプ
testConnectionOn Startup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel の起動時に、すべての JMS コンシューマーが JMS ブローカーへの有効な接続を持つことが保証されます。接続を許可できない場合、Camel は起動時に例外を出力します。これにより、接続に失敗した状態で Camel が開始されなくなります。JMS プロデューサーもテストされています。	false	boolean
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
forceSendOriginal Message (producer)	mapJmsMessage=false を使用すると、ルート中にヘッダーに触れると (get または set)、Camel は新しい JMS メッセージを作成して新しい JMS 宛先に送信します。Camel が受信した元の JMS メッセージを強制的に送信するには、このオプションを true に設定します。	false	boolean
requestTimeout (producer)	InOut Exchange パターン使用時の応答待ちタイムアウト (ミリ秒単位)。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めて、このエンドポイントで設定されたタイムアウト値をオーバーライドし、メッセージごとに個別のタイムアウト値を持つことができます。requestTimeoutCheckerInterval オプションも参照してください。	20000	long
requestTimeoutChecker Interval (advanced)	JMS を介してリクエスト/リプライを行うときに、Camel がタイムアウトになった Exchange をチェックする頻度を設定します。デフォルトでは、Camel は 1 秒に 1 回確認します。ただし、タイムアウトが発生したときに迅速に対応する必要がある場合は、この間隔を短くして、より頻繁にチェックすることができます。タイムアウトは、オプション requestTimeout によって決定されます。	1000	long

名前	説明	デフォルト	タイプ
transferExchange (advanced)	本文とヘッダーだけでなく、電信送金で交換を転送できます。次のフィールドが転送されます: In body、Out body、Fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、交換プロパティー、交換例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。プロデューサ側とコンシューマー側の両方でこのオプションを有効にする必要があるため、Camel はペイロードが Exchange であり、通常のペイロードではないことを認識します。	false	boolean
transferException (advanced)	有効で、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で失敗した場合、原因となった例外が javax.jms.ObjectMessage として応答で返されます。クライアントが Camel の場合、返された Exception は再出力されます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続的なキューを使用して堅牢なルーティングを有効にできます。transferExchange も有効にしている場合は、このオプションが優先されることに注意してください。キャッチされた例外はシリアライズ可能である必要があります。コンシューマー側の元の Exception は、プロデューサーに返されるときに org.apache.camel.RuntimeCamelException などの外部例外にラップできます。	false	boolean
transferFault (advanced)	これを有効にし、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で SOAP エラー (例外ではない) で失敗した場合には、MessageisFault() のエラーフラグが応答で、org.apache.camel.component.jms.JmsConstantsJMS_TRANSFER_FAULTJMS_TRANSFER_FAULT のキーを含んだ JMS ヘッダーとして送り返されます。クライアントが Camel の場合には、返される障害フラグはリンク org.apache.camel.MessagesetFault(boolean) に設定されます。cxf や spring-ws などの SOAP ベースなどの障害をサポートする Camel コンポーネントを使用する場合、これを有効できます。	false	boolean
jmsOperations (advanced)	org.springframework.jms.core.JmsOperations インターフェイスの独自の実装を使用できるようにします。Camel はデフォルトで JmsTemplate を使用します。テスト目的で使用できますが、Spring API ドキュメントに記載されているほどは使用されません。		JmsOperations

名前	説明	デフォルト	タイプ
destinationResolver (advanced)	独自のリゾルバーを使用できるようにするプラグ可能な org.springframework.jms.support.destination.DestinationResolver (たとえば、JNDI レジストリーで実際の宛先を検索するため)。		DestinationResolver
replyToType (producer)	JMS を介して要求/応答を行うときに、replyTo キューに使用する戦略の種類を明示的に指定できます。可能な値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、replyTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに専用キューを使用できます。詳細については、Camel JMS のドキュメントを参照してください。特に、クラスター化された環境で実行する場合の影響に関する注意事項と、共有応答キューは代替の一時および排他的キューよりもパフォーマンスが低いという事実を参照してください。		ReplyToType
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定ではなく、メッセージで指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。次の 3 つのヘッダーは、JMSPriority、JMSDeliveryMode、および JMSExpiration と見なされます。それらのすべてまたは一部のみを指定できます。指定されていない場合、Camel は代わりにエンドポイントからの値を使用するようにフォールバックします。したがって、このオプションを使用すると、ヘッダーはエンドポイントからの値をオーバーライドします。対照的に、explicitQosEnabled オプションは、エンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値は使用しません。	false	boolean
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得できますが、前のメッセージは (非同期ルーティングエンジンによって) 非同期に処理されます。これは、メッセージが 100% 厳密に順序どおりに処理されない可能性があることを意味します。無効になっている場合 (デフォルト)、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。transactioned が有効になっている場合、トランザクションは同期的に実行する必要があるため、asyncConsumer=true は非同期的に実行されないことに注意してください (Camel 3.0 は非同期トランザクションをサポートする場合があります)。	false	boolean

名前	説明	デフォルト	タイプ
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます (例: ファイアアンドフォーゲット)。このオプションを有効にすると、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用された実際の JMSMessageID で Camel Exchange が強化されます。	false	boolean
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean
defaultTaskExecutor Type (consumer)	コンシューマーエンドポイントとプロデューサエンドポイントの ReplyTo コンシューマーの両方に対して、DefaultMessageListenerContainer で使用するデフォルトの TaskExecutor タイプを指定します。可能な値: SimpleAsync (Spring の SimpleAsyncTaskExecutor を使用) または ThreadPool (Spring の ThreadPoolTaskExecutor を最適な値で使用 - キャッシュされたスレッドプールのようなもの)。設定されていない場合は、デフォルトで以前の動作になり、コンシューマーエンドポイントにはキャッシュされたスレッドプールが使用され、応答コンシューマーには SimpleAsync が使用されます。ThreadPool の使用は、同時コンシューマーが動的に増減するエラスティック設定でスレッドのゴミを減らすために推奨されます。		DefaultTaskExecutor Type
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy

名前	説明	デフォルト	タイプ
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値を持つ可能性がある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型の値を含む接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名でこれを行います。コンマで区切られた複数のヘッダー名を指定し、ワイルドカードマッチングの接尾辞として使用できます。		String
queueBrowseStrategy (advanced)	キューを参照するときにカスタム QueueBrowseStrategy を使用します。		QueueBrowseStrategy
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
waitForProvisionCorrelationToBeUpdated Counter (advanced)	JMS を介して要求/応答を行う場合、およびオプション useMessageIDAsCorrelationID が有効な場合に、暫定相関 ID が実際の相関 ID に更新されるのを待機する回数。	50	int
waitForProvisionCorrelationToBeUpdated ThreadSleepingTime (advanced)	暫定相関 ID が更新されるのを待機するたびにスリープする間隔 (ミリ単位)。	100	long
correlationProperty (producer)	JMSCorrelationID プロパティーの代わりに、この JMS プロパティーを使用して、InOut 交換パターン (要求 - 応答) でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用してメッセージと相関性のないシステムとメッセージを交換できます。JMSCorrelationID を使用すると、Camel によって使用または設定されません。ここで指定されたプロパティーの値は、同じ名前でメッセージのヘッダーに指定されていない場合に生成されます。		String
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常、subscriptionName 値と組み合わせて永続的なサブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean

名前	説明	デフォルト	タイプ
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常は subscriptionName 値と組み合わせて共有サブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。共有サブスクリプションも永続的である可能性があるため、このフラグを subscriptionDurable と組み合わせることもできます (多くの場合は組み合わせます)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 互換のメッセージブローカーが必要です。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有または永続的なサブスクリプションを持つトピック (pub-sub ドメイン) の場合に適用されます。サブスクリプション名は、このクライアントの JMS クライアント ID 内で一意である必要があります。デフォルトは、指定されたメッセージリスナーのクラス名です。注: 共有サブスクリプション (JMS 2.0 が必要) を除き、サブスクリプションごとに 1 つの同時コンシューマー (このメッセージリスナコンテナーのデフォルト) のみが許可されます。		String
streamMessageType Enabled (producer)	StreamMessage タイプを有効にするかどうかを設定します。ファイル、InputStream などのストリーミングの種類のメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、どの種類が使用されるかを制御します。デフォルトでは、BytesMessage が使用され、メッセージペイロード全体がメモリーに読み込まれます。このオプションを有効にすると、メッセージペイロードがチャンク単位でメモリーに読み込まれ、データがなくなるまで各チャンクが StreamMessage に書き込まれます。	false	boolean
formatDateHeadersTo Iso8601 (producer)	日付ヘッダーを ISO 8601 標準に従ってフォーマットするかどうかを設定します。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AMQP エンドポイントは、URI 構文を使用して設定されます。

```
amqp:destinationType:destinationName
```

パスおよびクエリーパラメーターを使用します。

5.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
destinationType	使用する宛先の種類	queue	String
destinationName	**必須** 宛先として使用するキューまたはトピックの名前		文字列

5.2.2. クエリーパラメーター (91 パラメーター)

名前	説明	デフォルト	タイプ
clientId (common)	使用する JMS クライアント ID を設定します。この値を指定する場合は、一意である必要があり、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続的なトピックサブスクリプションの場合にのみ必要です。Apache ActiveMQ を使用している場合は、代わりに仮想トピックを使用することをお勧めします。		String
connectionFactory (common)	使用する接続ファクトリー。コンポーネントまたはエンドポイントで接続ファクトリーを設定する必要があります。		ConnectionFactory

名前	説明	デフォルト	タイプ
disableReplyTo (common)	Camel がメッセージの JMSReplyTo ヘッダーを無視するかどうかを指定します。true の場合、Camel は JMSReplyTo ヘッダーで指定された宛先に返信を送り返しません。Camel にルートから消費させたいが、コード内の別のコンポーネントが応答メッセージを処理するため、Camel に自動的に応答メッセージを送り返したくない場合は、このオプションを使用できます。Camel を異なるメッセージブローカー間のプロキシーとして使用し、あるシステムから別のシステムにメッセージをルーティングする場合にも、このオプションを使用できます。	false	boolean
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		String
jmsMessageType (common)	JMS メッセージの送信に特定の javax.jms.Message 実装を強制的に使用できるようにします。可能な値は、Bytes、Map、Object、Stream、Text です。デフォルトでは、Camel は In body タイプから使用する JMS メッセージタイプを決定します。このオプションで指定できます。		JmsMessageType
testConnectionOnStartup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel の起動時に、すべての JMS コンシューマーが JMS ブローカーへの有効な接続を持つことが保証されます。接続を許可できない場合、Camel は起動時に例外を出力します。これにより、接続に失敗した状態で Camel が開始されなくなります。JMS プロデューサーもテストされています。	false	boolean
acknowledgementModeName (consumer)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	String

名前	説明	デフォルト	タイプ
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得できますが、前のメッセージは (非同期ルーティングエンジンによって) 非同期に処理されます。これは、メッセージが 100% 厳密に順序どおりに処理されない可能性があることを意味します。無効になっている場合 (デフォルト)、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。transactioned が有効になっている場合、トランザクションは同期的に実行する必要があるため、asyncConsumer=true は非同期的に実行されないことに注意してください (Camel 3.0 は非同期トランザクションをサポートする場合があります)。	false	boolean
autoStartup (consumer)	コンシューマーコンテナーを自動起動するかどうかを指定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cacheLevel (consumer)	基礎となる JMS リソースの ID によってキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	基礎となる JMS リソースのキャッシュレベルを名前で設定します。可能な値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、および CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring のドキュメントとトランザクションキャッシュレベルを参照してください。	CACHE_AUTO	String
concurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーのデフォルト数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。	1	int

名前	説明	デフォルト	タイプ
maxConcurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーの最大数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToMaxConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。		int
replyTo (consumer)	Message.getJMSReplyTo() の着信値をオーバーライドする明示的な ReplyTo 宛先を提供します。		String
replyToDeliveryPersistent (consumer)	返信に対してデフォルトで永続的な配信を使用するかどうかを指定します。	true	boolean
selector (consumer)	使用する JMS セレクターを設定します。		String
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常、subscriptionName 値と組み合わせて永続的なサブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有または永続的なサブスクリプションを持つトピック (pub-sub ドメイン) の場合に適用されます。サブスクリプション名は、このクライアントの JMS クライアント ID 内で一意である必要があります。デフォルトは、指定されたメッセージリスナーのクラス名です。注: 共有サブスクリプション (JMS 2.0 が必要) を除き、サブスクリプションごとに 1 つの同時コンシューマー (このメッセージリスナコンテナーのデフォルト) のみが許可されます。		String

名前	説明	デフォルト	タイプ
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常は subscriptionName 値と組み合わせて共有サブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。共有サブスクリプションも永続的である可能性があるため、このフラグを subscriptionDurable と組み合わせることもできます (多くの場合は組み合わせます)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 互換のメッセージブローカーが必要です。	false	boolean
acceptMessagesWhileStopping (consumer)	コンシューマーが停止中にメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止するが、キューにメッセージが入れられている場合は、このオプションを有効にすることを検討してください。このオプションが false の場合は、JMS ルートを停止すると、メッセージが拒否される可能性があり、JMS ブローカは再配信を試行する必要がありますが、これも拒否される可能性があり、最終的にメッセージはJMS ブローカー上のデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることをお勧めします。	false	boolean
allowReplyManagerQuickStop (consumer)	JmsConfigurationisAcceptMessagesWhileStopping が有効で、org.apache.camel.CamelContext が現在停止している場合に、要求/応答メッセージングのリプライマネージャーで使用される DefaultMessageListenerContainer が、DefaultMessageListenerContainer.runningAllowed フラグを迅速に停止できるようにするかどうか。このクイック停止機能は、通常の JMS コンシューマーではデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean

名前	説明	デフォルト	タイプ
consumerType (consumer)	使用するコンシューマータイプ。Simple、Default、または Custom のいずれかです。コンシューマータイプによって、使用する Spring JMS リスナーが決まります。デフォルトは org.springframework.jms.listener.DefaultMessageListenerContainer を使用し、Simple は org.springframework.jms.listener.SimpleMessageListenerContainer を使用します。Custom を指定した場合は、messageListenerContainerFactory オプションで定義された MessageListenerContainerFactory によって、使用する org.springframework.jms.listener.AbstractMessageListenerContainer が決まります。	デフォルト	ConsumerType
defaultTaskExecutorType (consumer)	コンシューマーエンドポイントとプロデューサエンドポイントの ReplyTo コンシューマーの両方に対して、DefaultMessageListenerContainer で使用するデフォルトの TaskExecutor タイプを指定します。可能な値: SimpleAsync (Spring の SimpleAsyncTaskExecutor を使用) または ThreadPool (Spring の ThreadPoolTaskExecutor を最適な値で使用 - キャッシュされたスレッドプールのようなもの)。設定されていない場合は、デフォルトで以前の動作になり、コンシューマーエンドポイントにはキャッシュされたスレッドプールが使用され、応答コンシューマーには SimpleAsync が使用されます。ThreadPool の使用は、同時コンシューマーが動的に増減するエラスティック設定でスレッドのゴミを減らすために推奨されます。		DefaultTaskExecutor Type
eagerLoadingOfProperties (consumer)	メッセージが読み込まれるとすぐに JMS プロパティーの先行読み込みを有効にします。これは、JMS プロパティーが必要ない場合があるため一般的に非効率的ですが、基盤となる JMS プロバイダーと JMS プロパティーの使用に関する問題を早期に発見できる場合があります。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
exposeListenerSession (consumer)	メッセージを消費するときにリスナーセッションを公開するかどうかを指定します。	false	boolean

名前	説明	デフォルト	タイプ
replyToSameDestination Allowed (consumer)	JMS コンシューマーが、コンシューマーが使用しているのと同じ宛先に応答メッセージを送信できるかどうか。これにより、同じメッセージを消費してそれ自体に送り返すことで、無限ループが回避されます。	false	boolean
taskExecutor (consumer)	メッセージを消費するためのカスタムタスクエグゼキュータを指定できます。		TaskExecutor
deliveryMode (producer)	使用する配信モードを指定します。可能な値は、javax.jms.DeliveryMode によって定義された値です。NON_PERSISTENT = 1 および PERSISTENT = 2。		Integer
deliveryPersistent (producer)	デフォルトで永続配信を使用するかどうかを指定します。	true	boolean
explicitQosEnabled (producer)	メッセージの送信時に、deliveryMode、priority、または timeToLive のサービス品質を使用する必要があるかどうかを設定します。このオプションは、Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で動作し、Camel In メッセージヘッダーから排他的に QoS プロパティーを読み取る preserveMessageQos オプションとは対照的です。	false	Boolean
formatDateHeadersToIso8601 (producer)	JMS 日付プロパティーを ISO 8601 標準に従ってフォーマットするかどうかを設定します。	false	boolean
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定ではなく、メッセージで指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。次の 3 つのヘッダーは、JMSPriority、JMSDeliveryMode、および JMSExpiration と見なされます。それらのすべてまたは一部のみを指定できます。指定されていない場合、Camel は代わりにエンドポイントからの値を使用するようにフォールバックします。したがって、このオプションを使用すると、ヘッダーはエンドポイントからの値をオーバーライドします。対照的に、explicitQosEnabled オプションは、エンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値は使用しません。	false	boolean

名前	説明	デフォルト	タイプ
priority (producer)	1より大きい値は、送信時のメッセージの優先度を指定します (0 が最低の優先度で、9 が最高の優先度です)。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
replyToConcurrentConsumers (producer)	JMS を介して要求/応答を行うときの同時コンシューマーのデフォルト数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。	1	int
replyToMaxConcurrent Consumers (producer)	JMS を介した要求/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。		int
replyToOnTimeoutMax ConcurrentConsumers (producer)	JMS 経由の要求/応答を使用するときにタイムアウトが発生したときに、ルーティングを継続するための同時コンシューマーの最大数を指定します。	1	int
replyToOverride (producer)	JMS メッセージで明示的な ReplyTo 宛先を提供します。これは、replyTo の設定をオーバーライドします。メッセージをリモート Queue に転送し、ReplyTo 宛先から応答メッセージを受け取る場合に便利です。		String
replyToType (producer)	JMS を介して要求/応答を行うときに、replyTo キューに使用する戦略の種類を明示的に指定できます。可能な値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、replyTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに専用キューを使用できます。詳細については、Camel JMS のドキュメントを参照してください。特に、クラスター化された環境で実行する場合の影響に関する注意事項と、共有応答キューは代替の一時および排他的キューよりもパフォーマンスが低いという事実を参照してください。		ReplyToType

名前	説明	デフォルト	タイプ
requestTimeout (producer)	InOut Exchange パターン使用時の応答待ちタイムアウト (ミリ秒単位)。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めて、このエンドポイントで設定されたタイムアウト値をオーバーライドし、メッセージごとに個別のタイムアウト値を持つことができます。requestTimeoutCheckerInterval オプションも参照してください。	20000	long
timeToLive (producer)	メッセージの送信時に、メッセージの有効期限をミリ秒単位で指定します。	-1	long
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値を持つ可能性がある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型の値を含む接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名でこれを行います。コンマで区切られた複数のヘッダー名を指定し、ワイルドカードマッチングの接尾辞として使用できます。		String
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
alwaysCopyMessage (producer)	true の場合、メッセージがプロデューサーに渡されて送信されると、Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、状況によってはメッセージのコピーが必要です (また、replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します)。	false	boolean
correlationProperty (producer)	InOut 交換パターンを使用する場合、JMSCorrelationID JMS プロパティーの代わりにこの JMS プロパティーを使用してメッセージを関連付けます。設定されたメッセージがこのプロパティーの値のみに関連付けられる場合、JMSCorrelationID プロパティーは無視され、Camel によって設定されません。		String

名前	説明	デフォルト	タイプ
disableTimeToLive (producer)	このオプションを使用して、有効期限を強制的に無効にします。たとえば、JMS を介して要求/応答を行う場合、Camel はデフォルトで、送信されるメッセージの存続時間として requestTimeout 値を使用します。問題は、送信側システムと受信側システムのクロックを同期させる必要があるため、同期していることです。これをアーカイブするのは必ずしも簡単ではありません。したがって、disableTimeToLive=true を使用して、送信されたメッセージに有効期限の値を設定しないようにすることができます。その後、メッセージは受信側システムで期限切れになりません。詳細については、以下の生存時間についてのセクションを参照してください。	false	boolean
forceSendOriginalMessage (producer)	mapJmsMessage=false を使用すると、ルート中にヘッダーに触れると (get または set)、Camel は新しい JMS メッセージを作成して新しい JMS 宛先に送信します。Camel が受信した元の JMS メッセージを強制的に送信するには、このオプションを true に設定します。	false	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます (例: ファイアアンドフォーゲット)。このオプションを有効にすると、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用された実際の JMSMessageID で Camel Exchange が強化されます。	false	boolean
replyToCacheLevelName (producer)	JMS を介して要求/応答を行うときに、応答コンシューマーのキャッシュレベルを名前で設定します。このオプションは、固定応答キュー (一時的ではない) を使用する場合にのみ適用されます。Camel はデフォルトで次を使用します: 排他的または replyToSelectorName と共有の CACHE_CONSUMER。そして、replyToSelectorName なしで共有するための CACHE_SESSION。IBM WebSphere などの一部の JMS ブローカーは、replyToCacheLevelName=CACHE_NONE を機能させるために設定する必要がある場合があります。注: 一時キューを使用する場合、CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などのより高い値を使用する必要があります。		String
replyToDestinationSelector Name (producer)	使用する固定名を使用して JMS セレクターを設定し、共有キューを使用している場合 (つまり、一時的な応答キューを使用していない場合) に、他の応答から自分の応答を除外できるようにします。		String

名前	説明	デフォルト	タイプ
streamMessageTypeEnabled (producer)	StreamMessage タイプを有効にするかどうかを設定します。ファイル、InputStream などのストリーミングの種類のメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、どの種類が使用されるかを制御します。デフォルトでは、BytesMessage が使用され、メッセージペイロード全体がメモリーに読み込まれます。このオプションを有効にすると、メッセージペイロードがチャンク単位でメモリーに読み込まれ、データがなくなるまで各チャンクが StreamMessage に書き込まれます。	false	boolean
allowSerializedHeaders (advanced)	シリアル化されたヘッダーを含めるかどうかを制御します。transferExchange が true の場合にのみ適用されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
destinationResolver (advanced)	独自のリゾルバーを使用できるようにするプラグ可能な org.springframework.jms.support.destination.DestinationResolver (たとえば、JNDI レジストリーで実際の宛先を検索するため)。		DestinationResolver

名前	説明	デフォルト	タイプ
errorHandler (advanced)	Message の処理中にキャッチされない例外が出力された場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ログレベルとスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングするよりも設定がはるかに簡単になります。		ErrorHandler
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
idleConsumerLimit (advanced)	常にアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
idleTaskExecutionLimit (advanced)	実行中にメッセージを受信していない、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスクに受信を任せます (動的スケジューリングの場合。maxConcurrentConsumers 設定を参照してください)。Spring から入手できる追加のドキュメントがあります。	1	int
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean

名前	説明	デフォルト	タイプ
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		文字列
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。	true	boolean
maxMessagesPerTask (advanced)	タスクあたりのメッセージ数。-1 は無制限です。同時コンシューマーの範囲 (例: min max) を使用する場合、このオプションを使用して値を 100 などに設定し、必要な作業が少ない場合にコンシューマーが縮小する速度を制御できます。	-1	int
messageConverter (advanced)	カスタム Spring org.springframework.jms.support.converter.MessageConverter を使用して、javax.jms.Message との間でどのようにマッピングするかを制御できるようにします。		MessageConverter
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合には、これらのメッセージはメッセージ ID を null に設定する必要があります。プロバイダーがヒントを無視する場合には、メッセージ ID は通常の一意の値に設定する必要があります	true	boolean

名前	説明	デフォルト	タイプ
messageListenerContainerFactory (advanced)	メッセージを消費するために使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定するために使用される MessageListenerContainerFactory のレジストリー ID。これを設定すると、consumerType が自動的に Custom に設定されます。		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	メッセージの送信時にデフォルトでタイムスタンプを有効にするかどうかを指定します。これは、JMS ブローカへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合には、これらのメッセージのタイムスタンプはゼロに設定する必要があります。プロバイダーがヒントを無視する場合には、タイムスタンプを通常の値に設定する必要があります	true	boolean
pubSubNoLocal (advanced)	独自の接続によってパブリッシュされたメッセージの配信を禁止するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージ受信のタイムアウト (ミリ秒単位)。	1000	long
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	long
requestTimeoutCheckerInterval (advanced)	JMS を介してリクエスト/リプライを行うときに、Camel がタイムアウトになった Exchange をチェックする頻度を設定します。デフォルトでは、Camel は 1 秒に 1 回確認します。ただし、タイムアウトが発生したときに迅速に対応する必要がある場合は、この間隔を短くして、より頻繁にチェックすることができます。タイムアウトは、オプション requestTimeout によって決定されます。	1000	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
transferException (advanced)	有効で、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で失敗した場合、原因となった例外が javax.jms.ObjectMessage として応答で返されます。クライアントが Camel の場合、返された Exception は再出力されます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続的なキューを使用して堅牢なルーティングを有効にできます。transferExchange も有効にしている場合は、このオプションが優先されることに注意してください。キャッチされた例外はシリアライズ可能である必要があります。コンシューマー側の元の Exception は、プロデューサーに返されるときに org.apache.camel.RuntimeCamelException などの外部例外にラップできます。	false	boolean
transferExchange (advanced)	本文とヘッダーだけでなく、電信送金で交換を転送できます。次のフィールドが転送されます: In body、Out body、Fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、交換プロパティー、交換例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。プロデューサ側とコンシューマー側の両方でこのオプションを有効にする必要があるため、Camel はペイロードが Exchange であり、通常のペイロードではないことを認識します。	false	boolean
transferFault (advanced)	これを有効にし、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で SOAP エラー (例外ではない) で失敗した場合には、MessageisFault() のエラーフラグが応答で、org.apache.camel.component.jms.JmsConstantsJMS_TRANSFER_FAULTJMS_TRANSFER_FAULT のキーを含んだ JMS ヘッダーとして送り返されます。クライアントが Camel の場合には、返される障害フラグはリンク org.apache.camel.MessagesetFault(boolean) に設定されます。cxf や spring-ws などの SOAP ベースなどの障害をサポートする Camel コンポーネントを使用する場合、これを有効できます。	false	boolean
useMessageIDAs CorrelationID (advanced)	InOut メッセージの JMSCorrelationID として JMSMessageID を常に使用するかどうかを指定します。	false	boolean

名前	説明	デフォルト	タイプ
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	JMS を介して要求/応答を行う場合、およびオプション useMessageIDAsCorrelationID が有効な場合に、暫定相関 ID が実際の相関 ID に更新されるのを待機する回数。	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleeping Time (advanced)	暫定相関 ID が更新されるのを待機するたびにスリープする間隔 (ミリ単位)。	100	long
errorHandlerLoggingLevel (logging)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean
password (security)	ConnectionFactory で使用するパスワード。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
username (security)	ConnectionFactory で使用するユーザー名。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
取引済み (取引)	トランザクションモードを使用するかどうかを指定します	false	boolean
lazyCreateTransactionManager (トランザクション)	true の場合、オプション transacted=true のときに transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (トランザクション)	使用する Spring トランザクションマネージャー。		PlatformTransactionManager
transactionName (トランザクション)	使用するトランザクションの名前。		String
transactionTimeout (トランザクション)	トランザクションモードを使用している場合の、トランザクションのタイムアウト値 (秒単位)。	-1	int

5.3. 用途

AMQP コンポーネントは JMS コンポーネントから継承されるため、前者の使用法は後者とほぼ同じです。

AMQP コンポーネントの使用

```
// Consuming from AMQP queue
from("amqp:queue:incoming").
  to(...);

// Sending message to the AMQP topic
from(...).
  to("amqp:topic:notify");
```

5.4. AMQP コンポーネントの設定

Camel 2.16.1 以降では、**AMQPComponent#amqp10Component (String connectionURI)** ファクトリーメソッドを使用して、事前設定されたトピックの接頭辞を付けて AMQP 1.0 コンポーネントを返すこともできます。

AMQP 1.0 コンポーネントの作成

```
 AMQPComponent amqp =
AMQPComponent.amqp10Component("amqp://guest:guest@localhost:5672");
```

Camel 2.17 以降、**AMQPComponent#amqp10Component (String connectionURI)** ファクトリーメソッドが非推奨になり、**AMQPComponent#amqpComponent (String connectionURI)** が後継となったことに注意してください。

AMQP 1.0 コンポーネントの作成

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672");

AMQPComponent authorizedAmqp = AMQPComponent.amqpComponent("amqp://localhost:5672",
"user", "password");
```

Camel 2.17 以降、AMQP コンポーネントを自動的に設定するために、**org.apache.camel.component.amqp.AMQPConnectionDetails** のインスタンスをレジストリーに追加することもできます。たとえば、Spring Boot の場合には、Bean を定義するだけです。

AMQP 接続の詳細の自動設定

```
@Bean
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}

@Bean
AMQPConnectionDetails securedAmqpConnection() {
  return new AMQPConnectionDetails("amqp://lcoalhost:5672", "username", "password");
}
```

同様に、Camel-CDI を使用する場合は、CDI プロデューサーメソッドも使用できます。

CDI の AMQP 接続の詳細の自動設定

```
@Produces
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}
```

Camel プロパティー を利用して、AMQP 接続の詳細を読み取ることもできます。ファクトリーメソッド **AMQPConnectionDetails.discoverAMQP()** は、以下のスニペットで示されているように、Kubernetes に似た規則で Camel プロパティーを読み取ろうとします。

AMQP 接続の詳細の自動設定

```
export AMQP_SERVICE_HOST = "mybroker.com"
export AMQP_SERVICE_PORT = "6666"
export AMQP_SERVICE_USERNAME = "username"
export AMQP_SERVICE_PASSWORD = "password"

...

@Bean
AMQPConnectionDetails amqpConnection() {
  return AMQPConnectionDetails.discoverAMQP();
}
```

AMQP 固有のオプションを有効にする

たとえば、**amqp.traceFrames** を有効にする必要がある場合は、次の例のように、オプションを URI に追加することで有効にできます。

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672?
amqp.traceFrames=true");
```

参考までに、QPID JMS クライアント設定 を参照してください。

5.5. トピックの使用

camel-amqp でトピックを使用するには、以下に示すように、**topic:**// をトピック接頭辞として使用するようにコンポーネントを設定する必要があります。

```
 <bean id="amqp" class="org.apache.camel.component.amqp.AmqpComponent">
  <property name="connectionFactory">
    <bean class="org.apache.qpid.jms.JmsConnectionFactory" factory-method="createFromURL">
      <property name="remoteURI" value="amqp://localhost:5672" />
      <property name="topicPrefix" value="topic://" /> <!-- only necessary when connecting to
ActiveMQ over AMQP 1.0 -->
    </bean>
  </property>
 </bean>
```

AMQPComponent#amqpComponent() メソッドと **AMQPConnectionDetails** の両方がトピック接頭辞を使用してコンポーネントを事前設定するので、明示的に設定する必要がないことに注意してください。

5.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第6章 APNS コンポーネント

Camel バージョン 2.8 以降で利用可能

apns コンポーネントは、iOS デバイスに通知を送信するために使用されます。apns コンポーネントは javapns ライブラリーを使用します。
このコンポーネントは、Apple Push Notification Servers (APNS) への通知送信と、サーバーからのフィードバックの消費をサポートしています。

Apple Push Notification Servers からのフィードバックストリームは、たまに消費することがベストプラクティスであるので、デフォルトでは、コンシューマーのポーリングは 3600 秒に設定されています。例: サーバーのフラッディングを避けるために 1 時間ごと。

フィードバックストリームは、アクティブではないデバイスに関する情報を提供します。モバイルアプリケーションが頻繁に使用されていない場合は、数時間ごとにこの情報を取得するだけで済みます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-apns</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

6.1. URI 形式

通知を送信するには、以下を実行します。

```
apns:notify[?options]
```

フィードバックを利用するには、以下を実行します。

```
apns:consumer[?options]
```

6.2. オプション

APNS コンポーネントは、次に示す 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
apnsService (common)	**必須** 使用する ApnsService。org.apache.camel.component.apns.factory.ApnsServiceFactory を使用して ApnsService を構築できます		ApnsService
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

APNS エンドポイントは、URI 構文を使用して設定されます。

```
apns:name
```

パスおよびクエリーパラメーターを使用します。

6.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	エンドポイントの名前		String

6.2.2. クエリーパラメーター (20 パラメーター)

名前	説明	デフォルト	タイプ
tokens (common)	通知するデバイスに関連するトークンを静的に宣言する場合は、このプロパティーを設定します。トークンはコンマで区切られます。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

6.2.3. コンポーネント

ApnsComponent は **com.notnoop.apns.ApnsService** で設定する必要があります。サービスは、**org.apache.camel.component.apns.factory.ApnsServiceFactory** を使用して作成および設定できます。例については、以下を参照してください。また、テストソースコード でも同様です。

6.2.3.1. SSL 設定

安全な接続を使用するには、コンポーネントの設定に使用される **org.apache.camel.component.apns.factory.ApnsServiceFactory** に **org.apache.camel.util.jsse.SSLContextParameters** のインスタンスを注入する必要があります。例については、テストリソースを参照してください。SSL の例

6.3. EXCHANGE データ形式

Camel がアクティブではないデバイスに対応するフィードバックデータを取得するタイミングで、InactiveDevice オブジェクトのリストを取得します。取得されたリストの各 InactiveDevice オブジェクトは In ボディーとして設定され、コンシューマーエンドポイントによって処理されます。

6.4. メッセージヘッダー

Camel Apns はこれらのヘッダーを使用します。

プロパティー	デフォルト	説明
CamelApnsTokens		デフォルトでは空です。
CamelApnsMessageType	**STRING, PAYLOAD, APNS_NOTIFICATION**	メッセージタイプに PAYLOAD を選択した場合には、メッセージは APNS ペイロードと見なされ、そのまま送信されます。STRING を選択した場合には、メッセージは APNS ペイロードとして変換されます。**Camel 2.16** 以降では、メッセージボディーを com.notnoop.apns.ApnsNotification タイプとして送信するために APNS_NOTIFICATION が使用されます。

6.5. APNSSERVICEFACTORY BUILDER CALLBACK

ApnsServiceFactory には、デフォルトの **ApnsServiceBuilder** インスタンスを設定 (または置換) するために使用できる空のコールバックメソッドが付属しています。メソッドのシグネチャーは次のようになります。

```
protected ApnsServiceBuilder configureServiceBuilder(ApnsServiceBuilder serviceBuilder);
```

そして、次のように使用できます。

```
ApnsServiceFactory proxiedApnsServiceFactory = new ApnsServiceFactory(){

  @Override
  protected ApnsServiceBuilder configureServiceBuilder(ApnsServiceBuilder serviceBuilder) {
    return serviceBuilder.withSocksProxy("my.proxy.com", 6666);
  }

};
```

6.6. サンプル

6.6.1. Camel Xml ルート

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

    <!-- Replace by desired values -->
```

```
    <bean id="apnsServiceFactory"
class="org.apache.camel.component.apns.factory.ApnsServiceFactory">

        <!-- Optional configuration of feedback host and port -->
        <!-- <property name="feedbackHost" value="localhost" /> -->
        <!-- <property name="feedbackPort" value="7843" /> -->

        <!-- Optional configuration of gateway host and port -->
        <!-- <property name="gatewayHost" value="localhost" /> -->
        <!-- <property name="gatewayPort" value="7654" /> -->

        <!-- Declaration of certificate used -->
              <!-- from Camel 2.11 onwards you can use prefix: classpath:, file: to refer to load the
certificate from classpath or file. Default it classpath -->
        <property name="certificatePath" value="certificate.p12" />
        <property name="certificatePassword" value="MyCertPassword" />

        <!-- Optional connection strategy - By Default: No need to configure -->
        <!-- Possible options: NON_BLOCKING, QUEUE, POOL or Nothing -->
        <!-- <property name="connectionStrategy" value="POOL" /> -->
        <!-- Optional pool size -->
        <!-- <property name="poolSize" value="15" /> -->

        <!-- Optional connection strategy - By Default: No need to configure -->
        <!-- Possible options: EVERY_HALF_HOUR, EVERY_NOTIFICATION or Nothing (Corresponds
to NEVER javapns option) -->
        <!-- <property name="reconnectionPolicy" value="EVERY_HALF_HOUR" /> -->
    </bean>

    <bean id="apnsService" factory-bean="apnsServiceFactory" factory-method="getApnsService" />

    <!-- Replace this declaration by wanted configuration -->
    <bean id="apns" class="org.apache.camel.component.apns.ApnsComponent">
        <property name="apnsService" ref="apnsService" />
    </bean>

    <camelContext id="camel-apns-test" xmlns="http://camel.apache.org/schema/spring">
            <route id="apns-test">
                <from uri="apns:consumer?initialDelay=10&amp;delay=3600&amp;timeUnit=SECONDS"
/>
                <to uri="log:org.apache.camel.component.apns?showAll=true&amp;multiline=true" />
                <to uri="mock:result" />
            </route>
    </camelContext>

</beans>
```

6.6.2. Camel Java ルート

camel コンテキストを作成し、apns コンポーネントをプログラムで宣言する

```
    protected CamelContext createCamelContext() throws Exception {
        CamelContext camelContext = super.createCamelContext();

        ApnsServiceFactory apnsServiceFactory = new ApnsServiceFactory();
```

```
        apnsServiceFactory.setCertificatePath("classpath:/certificate.p12");
        apnsServiceFactory.setCertificatePassword("MyCertPassword");

        ApnsService apnsService = apnsServiceFactory.getApnsService(camelContext);

        ApnsComponent apnsComponent = new ApnsComponent(apnsService);
        camelContext.addComponent("apns", apnsComponent);

        return camelContext;
    }
```

[[APNS-ApnsProducer-iOStargetdevicedynamicallyconfiguredviaheader:"CamelApnsTokens"]]
ApnsProducer - ヘッダー経由で動的に設定された iOS ターゲットデバイス: **" CamelApnsTokens "**

```
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            public void configure() throws Exception {
                from("direct:test")
                    .setHeader(ApnsConstants.HEADER_TOKENS, constant(IOS_DEVICE_TOKEN))
                    .to("apns:notify");
                }
        }
    }
```

ApnsProducer - URI を介して静的に設定された iOS ターゲットデバイス

```
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            public void configure() throws Exception {
                from("direct:test").
                to("apns:notify?tokens=" + IOS_DEVICE_TOKEN);
            }
        };
    }
```

ApnsConsumer

```
from("apns:consumer?initialDelay=10&delay=3600&timeUnit=SECONDS")
    .to("log:com.apache.camel.component.apns?showAll=true&multiline=true")
    .to("mock:result");
```

6.7. 関連項目

- コンポーネント
- エンドポイント * APNS の使用に関するブログ (フランス語)

第7章 ASN.1 FILE DATAFORMAT

Camel バージョン 2.20 以降で利用可能

ASN.1 データ形式 Data Format Intoduction to ASN.1(https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx) は、Bouncy に基づく Camel Frameworks のデータ形式の実装です。Castle の bcprov-jdk15on ライブラリーと jASN.1 の Java コンパイラーは、アプリケーションが複雑であろうと非常に単純であろうと、言語実装やこれらのデータの物理表現に関係なく、通信プロトコルによって送信されるデータを記述するために使用される正式な表記法に使用されます。メッセージはプレーン Java オブジェクトに非整列化 (単純な Java POJO への変換) できます。Camel のルーティングエンジンとデータ変換の助けを借りて、POJO を操作し、カスタマイズされた書式を適用し、他の Camel コンポーネントを呼び出してメッセージを変換し、上流のシステムに送信することができます。

7.1. ASN.1 データ形式オプション

ASN.1 ファイルのデータ形式は、以下に示す 3 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
usingIterator	**false**	**Boolean**	asn1 ファイルに複数のエントリーがある場合には、このオプションを true に設定すると、スプリッター EIP を使用して、ストリーミングモードで反復子を使用してデータを分割できます。
clazzName		**String**	アンマーシャリング時に使用するクラスの名前
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

7.2. UNMARSHAL

ASN.1 構造化メッセージを非整列化するには、3 つの異なる方法があります。(通常はバイナリーファイル)

この最初の例では、BER ファイルペイロードを OutputStream にアンマーシャリングし、モックエンドポイントに送信します。

```
from("direct:unmarshal").unmarshal(asn1).to("mock:unmarshal");
```

2 番目の例では、Split EIP を使用して BER ファイルペイロードをバイト配列に非整列化します。分割 EIP を適用する理由は、通常、各 BER ファイルまたは (ASN.1 構造化ファイル) には処理する複数のレコードが含まれており、分割 EIP はファイル内の各レコードを実際に ASN1Primitive のインスタンスであるバイト配列として取得するのに役立つためです (bcprov-jdk15on ライブラリーでの Bouncy Castle の ASN.1 サポート) 次に、(ASN1Primitive.fromByteArray) の public static メソッドを使用して、バイト配列を ASN1Primitive に変換できます。このような例では、**usingIterator=true** を設定する必要があることに注意してください。

```
from("direct:unmarshal").unmarshal(asn1).split(body(Iterator.class)).streaming().to("mock:unmarshal"
);
```

最後の例では、分割 EIP を使用して BER ファイルペイロードをプレーンな古い Java オブジェクトに非整列化します。分割 EIP を適用する理由は、前の例ですでに説明されています。その理由に注意して覚えておいてください。このような例では、クラスの完全修飾名または <YourObject>.class 参照をデータ形式で設定する必要もあります。ここで注意すべき重要なことは、オブジェクトは、ASN.1 構造の Java オブジェクト表現を生成する優れたツールである jasn1 コンパイラーによって生成されている必要があるということです。jasn1 コンパイラーの参照用法については、JASN.1 プロジェクトページ (https://www.openmuc.org/asn1/) を参照してください。また、maven の exec プラグインを使用してコンパイラーを呼び出す方法も参照してください。たとえば、このデータ形式の単体テストでは、サンプルの ASN.1 構造体 (TestSMSBerCdr.asn1) が **src/test/resources/asn1_structure** に追加されます。jasn1 コンパイラーが呼び出され、Java オブジェクトの表現 **が ${basedir}/target/generated/src/test/java** に生成されます。

```
from("direct:unmarshaldsl")
        .unmarshal()
        .asn1("org.apache.camel.dataformat.asn1.model.testsmscbercdr.SmsCdr")
        .split(body(Iterator.class)).streaming()
.to("mock:unmarshaldsl");
```

7.3. 依存関係

camel ルートで ASN.1 データ形式を使用するには、このデータ形式を実装する **camel-asn1** に依存関係を追加する必要があります。

Maven を使用する場合は、**pom.xml** に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-asn1</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第8章 アスタリスクコンポーネント

Camel バージョン 2.18 以降で利用可能

asterisk: コンポーネントを使用すると、asterisk-java を使用して、Asterisk PBX サーバー http://www.asterisk.org/ を簡単に操作できます。

このコンポーネントは、Asterisk Manager Interface とのインターフェイスに役立ちます

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-asterisk</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

8.1. URI 形式

```
asterisk:name[?options]
```

8.2. オプション

Asterisk コンポーネントにはオプションがありません。

Asterisk エンドポイントは、URI 構文を使用して設定されます。

```
asterisk:name
```

パスおよびクエリーパラメーターを使用します。

8.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** 論理名		String

8.2.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
hostname (common)	**必須** アスタリスクサーバーのホスト名		String
password (common)	**必須** ログインパスワード		String

名前	説明	デフォルト	タイプ
username (common)	**必須** ログインユーザー名		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
action (producer)	キューのステータス、sip ピア、または拡張状態の取得など、実行するアクション。		AsteriskAction
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

8.3. ACTION

サポートされているアクションは次のとおりです。

- QUEUE_STATUS、キューのステータス
- SIP_PEERS、SIP ピアの一覧表示
- EXTENSION_STATE、拡張ステータスをチェック

第9章 ATMOS コンポーネント

Camel バージョン 2.15 以降で利用可能

Camel-Atmos は、Atmos Client を使用して ViPR オブジェクトデータサービスを操作できるようにする Apache Camel コンポーネントです。

```
from("atmos:foo/get?remotePath=/path").to("mock:test");
```

9.1. オプション

Atmos コンポーネントは、次に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
fullTokenId (security)	Atmos クライアントに渡すトークン ID		String
secretKey (security)	Atmos クライアントに渡す秘密鍵		String
uri (advanced)	Atmos クライアントが接続するサーバーの URI		String
sslValidation (security)	Atmos クライアントが SSL 検証を実行する必要があるかどうか	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atmos エンドポイントは、URI 構文を使用して設定されます。

```
atmos:name/operation
```

パスおよびクエリーパラメーターを使用します。

9.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
name	Atmos 名		String
operation	実行するために **必要な** 操作		AtmosOperation

9.1.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
enableSslValidation (common)	Atmos SSL 検証	false	boolean
fullTokenId (common)	Atmos クライアントの fullTokenId		String
localPath (common)	ファイルを置くローカルパス		String
newRemotePath (common)	ファイル移動時の Atmos 上の新しいパス		String
query (common)	Atmos での検索クエリー		String
remotePath (common)	Atmos でファイルを配置する場所		String
secretKey (common)	Atmos 共有シークレット		String
uri (common)	Atomos サーバー uri		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

9.2. 依存関係

camel ルートで Atmos を使用するには、このデータ形式を実装する **camel-atmos** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atmos</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

9.3. 統合

atmos の統合を見ると、ScheduledPollConsumer の一種である GetConsumer という 1 つのタイプのコンシューマーがあります。

- **Get**

一方、生産者には 4 つのタイプがあります。

- **Get**
- **Del**
- **Move**
- **Put**

9.4. 例

これらの例は、テストから取得されます。

```
from("atmos:foo/get?remotePath=/path").to("mock:test");
```

ここでは、これはコンシューマーの例です。**remotePath** は、データが読み取られる場所からのパスを表し、camel exchange をプロデューサに渡します。その下で、このコンポーネントは、この操作と他のすべての操作に atmos クライアント API を使用します。

```
from("direct:start")
.to("atmos://get?remotePath=/dummy/dummy.txt")
.to("mock:result");
```

これがプロデューサーのサンプルです。**remotePath** は、ViPR オブジェクトデータサービスで操作が発生するパスを表します。プロデューサーでは、操作 (**Get**、**Del**、**Move**、**Put**) が ViPR オブジェクトデータサービスで実行され、結果が camel exchange のヘッダーに設定されます。

操作に関しては、次のヘッダーが camel exchange に設定されています

```
DOWNLOADED_FILE, DOWNLOADED_FILES, UPLOADED_FILE, UPLOADED_FILES,
FOUND_FILES, DELETED_PATH, MOVED_PATH;
```

9.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第10章 ATMOSPHERE WEBSOCKET コンポーネント

Camel バージョン 2.14 以降で利用可能

Attention-websocket: コンポーネントは、Websocket を介して外部クライアントと通信するサーブレットに Websocket ベースのエンドポイントを提供します (外部クライアントからの Websocket 接続を受け入れるサーブレットとして)。
このコンポーネントは、SERVLET コンポーネントを使用し、Atmosphere ライブラリーを使用して、さまざまなサーブレットコンテナー (Jetty、Tomcat など) で Websocket トランスポートをサポートします。

組み込みの Jetty サーバーを起動する Websocket コンポーネントとは異なり、このコンポーネントはコンテナーのサーブレットプロバイダーを使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atmosphere-websocket</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

10.1. ATMOSPHERE-WEBSOCKET オプション

Atmosphere Websocket コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
servletName (common)	使用するサーブレットのデフォルト名。デフォルト名は CamelServlet です。		文字列
httpRegistry (common)	カスタム org.apache.camel.component.servlet.HttpRegistry を使用します。		HttpRegistry
attachmentMultipart Binding (共通)	Camel エクスチェンジで multipart/form-data を添付として自動的にバインドするかどうか。オプション attachmentMultipartBinding=true と disableStreamCache=false は一緒に使用できません。AttachmentMultipartBinding を使用するには、disableStreamCache を削除します。サーブレットの使用時にこれを有効にするには、サーブレット固有の設定が必要になる場合があるため、これはデフォルトでオフになっています。	false	boolean
httpBinding (上級)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration を基本設定として使用するには、以下を行います。		HttpConfiguration

名前	説明	デフォルト	タイプ
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に Java シリアル化を許可するかどうか。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atmosphere Websocket エンドポイントは、URI 構文を使用して設定されます。

```
atmosphere-websocket:servicePath
```

パスおよびクエリーパラメーターを使用します。

10.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
servicePath	**必須** Websocket エンドポイントの名前		文字列

10.1.2. クエリーパラメーター(37 個のパラメーター):

名前	説明	デフォルト	タイプ
chunked (common)	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean

名前	説明	デフォルト	タイプ
disableStreamCache (common)	サーブレットからの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultHttpBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。サーブレットを使用してエンドポイントをブリッジ/プロキシーする場合、メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にしてパフォーマンスを向上させることを検討してください。http/http4 プロデューサーは、デフォルトでレスポンスボディーストリームをキャッシュします。このオプションを true に設定すると、プロデューサは応答本文ストリームをキャッシュせず、応答ストリームをそのままメッセージ本文として使用します。	false	boolean
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
sendToAll (common)	すべてに送信する (ブロードキャスト) か、単一の受信者に送信するか。	false	boolean
transferException (common)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
useStreaming (common)	ストリーミングを有効にして、データを複数のテキストフラグメントとして送信します。	false	boolean

名前	説明	デフォルト	タイプ
httpBinding (common)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
async (consumer)	非同期モードで動作するようにコンシューマーを設定します	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
httpMethodRestrict (consumer)	GET/POST/PUT など、HttpMethod が一致する場合にのみ消費を許可するために使用されます。複数のメソッドをコンマで区切って指定できます。		String
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、コンシューマーが URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse.		Integer
servletName (consumer)	使用するサーブレットの名前	Camel Servlet	String
attachmentMultipartBinding (consumer)	Camel エクスチェンジで multipart/form-data を添付として自動的にバインドするかどうか。オプション attachmentMultipartBinding=true と disableStreamCache=false は一緒に使用できません。AttachmentMultipartBinding を使用するには、disableStreamCache を削除します。サーブレットの使用時にこれを有効にするには、サーブレット固有の設定が必要になる場合があるため、これはデフォルトでオフになっています。	false	boolean
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 または存在しない場合に、HTTP リクエストにコンテンツがあるかどうかを先行チェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
optionsEnabled (consumer)	このサーブレットコンシューマーに対して HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは、OPTIONS はオフになっています。	false	boolean
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI を要求に使用します。オプション throwExceptionOnFailure を false に設定して、HttpProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは、connectionClose は false です。	false	boolean
copyHeaders (producer)	このオプションが true の場合、IN 交換ヘッダーは、コピー戦略に従って OUT 交換ヘッダーにコピーされます。これを false に設定すると、HTTP 応答からのヘッダーのみを含めることができます (IN ヘッダーは伝播されません)。	true	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。設定されている場合、HttpMethod ヘッダーはこのオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本文を読み取らず、入力ストリームをキャッシュしません。	false	boolean

名前	説明	デフォルト	タイプ
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在の Exchange Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受信した Host ヘッダーにアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで役立ちます。Host ヘッダーを使用するアプリケーションが、プロキシーされたサービスの正確な URL を生成できるようにします。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。	200-299	String
urlRewrite (producer)	**非推奨** カスタム org.apache.camel.component.http.UrlRewrite を参照して、エンドポイントをブリッジ/プロキシーするときに URL を書き換えることができます。詳細は、http://camel.apache.org/urlrewrite.html を参照してください。		UrlRewrite
mapHttpMessageBody (advanced)	このオプションが true の場合、交換の IN exchange ボディは HTTP ボディにマップされます。これを false に設定すると、HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、交換の IN exchange Form Encoded ボディは HTTP にマップされます。これを false に設定すると、HTTP Form Encoded ボディマッピングが回避されます。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、交換の IN exchange ヘッダーは HTTP ヘッダーにマップされます。これを false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		String
proxyHost (proxy)	使用するプロキシーホスト名		String
proxyPort (proxy)	使用するプロキシーポート		int
authHost (security)	NTML で使用する認証ホスト		文字列

10.2. URI 形式

```
atmosphere-websocket:///relative path[?options]
```

10.3. WEBSOCKET を介したデータの読み取りと書き込み

atmopshere-websocket エンドポイントは、エンドポイントがそれぞれプロデューサーまたはコンシューマーとして設定されているかどうかに応じて、ソケットにデータを書き込むか、ソケットから読み取ることができます。

10.4. データを読み書きするための URI の設定

以下のルートでは、Camel は指定された websocket 接続から読み取ります。

```
from("atmosphere-websocket:///servicepath")
        .to("direct:next");
```

同等の Spring の例:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="atmosphere-websocket:///servicepath"/>
    <to uri="direct:next"/>
  </route>
</camelContext>
```

以下のルートでは、Camel は指定された websocket 接続から読み取ります。

```
from("direct:next")
        .to("atmosphere-websocket:///servicepath");
```

同等の Spring の例:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:next"/>
```

```
    <to uri="atmosphere-websocket:///servicepath"/>
  </route>
</camelContext>
```

10.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- SERVLET
- AHC-WS * Websocket

第11章 ATOM コンポーネント

Camel バージョン 1.2 以降で利用可能

atom: コンポーネントは、Atom フィードのポーリングに使用されます。

デフォルトでは、Camel は 60 秒ごとにフィードをポーリングします。
注記: コンポーネントは現在、ポーリング (consuming) フィードのみをサポートしています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atom</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

11.1. URI 形式

```
atom://atomUri[?options]
```

atomUri は、ポーリングする Atom フィードへの URI です。

11.2. オプション

Atom コンポーネントにはオプションがありません。

Atom エンドポイントは、URI 構文を使用して設定されます。

```
atom:feedUri
```

パスおよびクエリーパラメーターを使用します。

11.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
feedUri	**必須** ポーリングするフィードへの URI。		String

11.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
feedHeader (consumer)	フィードオブジェクトをヘッダーとして追加するかどうかを設定します	true	boolean
filter (consumer)	エントリーのフィルタリングを使用するかどうかを設定します。	true	boolean
lastUpdate (consumer)	Atom フィードからのエントリーのフィルタリングに使用するタイムスタンプを設定します。このオプションは、splitEntries と組み合わせてのみ使用できます。		Date
password (consumer)	HTTP フィードからのポーリング時に Basic 認証に使用するパスワードを設定します		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sortEntries (consumer)	エントリーを公開日順にソートするかどうかを設定します。splitEntries = true の場合のみ機能します。	false	boolean
splitEntries (consumer)	エントリーを個別に送信するか、フィード全体を 1 つのメッセージとして送信するかを設定します	true	boolean
throttleEntries (consumer)	1 回のフィードポーリングで識別されたすべてのエントリーをすぐに配信するかどうかを設定します。true の場合、consumer.delay ごとに 1 つのエントリーのみが処理されます。splitEntries = true の場合にのみ適用されます。	true	boolean
username (consumer)	HTTP フィードからのポーリング時に Basic 認証に使用されるユーザー名を設定します		String

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

11.3. EXCHANGE データ形式

Camel は、返された **Exchange** の In ボディーにエントリーを設定します。 **splitEntries** フラグに応じて、Camel は 1 つの **Entry** または **List<Entry>** を返します。

オプション	値	動作
splitEntries	**true**	現在処理中のフィードから 1 つのエントリーのみが設定されます: **exchange.in.body (Entry)**
splitEntries	**false**	フィードからのエントリーの全リストが設定されます: **exchange.in.body (List<Entry>)**

Camel は、In ヘッダーに **Feed** オブジェクトを設定できます (これを無効にするには、**feedHeader** オプションを参照してください)。

11.4. メッセージヘッダー

Camel atom はこれらのヘッダーを使用します。

ヘッダー	説明
CamelAtomFeed	**org.apache.abdera.model.Feed** オブジェクトを使用すると、このヘッダーに設定されます。

11.5. サンプル

このサンプルでは、James Strachan のブログを調査します。

```
from("atom://http://macstrac.blogspot.com/feeds/posts/default").to("seda:feeds");
```

このサンプルでは、SEDA キューで気に入った良いブログのみをフィルタリングします。このサンプルは、コンテナーで実行したり、Spring を使用したりせずに、Camel をスタンドアロンでセットアップする方法も示しています。

11.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- RSS

第12章 ATOMIX マップコンポーネント

Camel バージョン 2.20 以降で利用可能

camel atomix-map コンポーネントを使用する と、Atomix の分散マップ コレクションを操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

12.1. URI 形式

```
atomix-map:mapName
```

12.2. オプション

Atomix Map コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有コンポーネントの設定		AtomixMapConfiguration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
nodes (common)	AtomixClient が接続する必要のあるノード		List
configurationUri (common)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Map エンドポイントは、URI 構文を使用して設定されます。

```
atomix-map:resourceName
```

パスおよびクエリーパラメーターを使用します。

12.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

12.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 uri。		String
defaultAction (common)	これがデフォルト動作です。	PUT	Action
key (common)	ヘッダーに何も設定されていない場合に使用するキー、または特定のキーのイベントをリッスンするキー。		Object
nodes (common)	クラスターを設定するノードのアドレス。		String
resultHeader (common)	結果を届けるヘッダー。		String
transport (common)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
ttl (common)	リソース ttl。		long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

12.3. ヘッダー

名前	タイプ	値	説明

名前	タイプ	値	説明
CamelAtomixResourceAction	AtomixMap.Action	• PUT • PUT_IF_ABSENT • GET • CLEAR • SIZE • CONTAINS_KEY • CONTAINS_VALUE • IS_EMPTY • ENTRY_SET • REMOVE • REPLACE • VALUES	実行する動作
CamelAtomixResourceKey	Object	-	操作するキー
CamelAtomixResourceValue	Object	-	Missing In Body が使用されている場合の値
CamelAtomixResourceOldValue	Object	-	古い値
CamelAtomixResourceTTL	String / long	-	エントリー TTL

名前	タイプ	値	説明
Camel Atomix ResourceReadConsistency	ReadConsistency	• ATOMIC • ATOMIC_LEASE • SEQUENTIAL • LOCAL	読み取りの一貫性レベル

12.4. ATOMIX クラスターに接続するためのコンポーネントの設定

参加する Atomix クラスターのノードは、以下の例のように、エンドポイントまたはコンポーネントレベル (推奨) で設定できます。

- **エンドポイント:**

```
<beans xmlns="...">
    <camelContext xmlns="http://camel.apache.org/schema/spring">
        <from uri="direct:start"/>
            <to uri="atomix-map:myMap?nodes=node-1.atomix.cluster:8700,node-
2.atomix.cluster:8700"/>
        </route>
    </camelContext>
</beans>
```

- **コンポーネント:**

```
<beans xmlns="...">
    <bean id="atomix-map"
class="org.apache.camel.component.atomix.client.map.AtomixMapComponent">
        <property name="nodes" value="nodes=node-1.atomix.cluster:8700,node-
2.atomix.cluster:8700"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/spring">
        <from uri="direct:start"/>
            <to uri="atomix-map:myMap"/>
        </route>
    </camelContext>
</beans>
```

12.5. 使用例:

- **TTL が 1秒の要素を PUT します。**

```
FluentProducerTemplate.on(context)
    .withHeader(AtomixClientConstants.RESOURCE_ACTION, AtomixMap.Action.PUT)
    .withHeader(AtomixClientConstants.RESOURCE_KEY, key)
    .withHeader(AtomixClientConstants.RESOURCE_TTL, "1s")
```

```
    .withBody(val)
    .to("direct:start")
    .send();
```

第13章 ATOMIX メッセージングコンポーネント

Camel バージョン 2.20 以降で利用可能

camel atomix-messaging コンポーネントを使用すると、Atomix の Group Messaging を操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

13.1. URI 形式

```
atomix-messaging:group
```

Atomix Messaging コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有コンポーネントの設定		AtomixMessaging 設定
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
nodes (common)	AtomixClient が接続する必要のあるノード		List
configurationUri (common)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix メッセージングエンドポイントは、URI 構文を使用して設定されます。

```
atomix-messaging:resourceName
```

パスおよびクエリーパラメーターを使用します。

13.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

13.1.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
atomix (common)	使用する Atomix インスタンス		Atomix
broadcastType (common)	ブロードキャストタイプ。	ALL	BroadcastType
channelName (common)	メッセージングチャネル名		String
configurationUri (common)	Atomix 設定 uri。		String
defaultAction (common)	これがデフォルト動作です。	DIRECT	Action
memberName (common)	Atomix グループのメンバー名		String
nodes (common)	クラスターを設定するノードのアドレス。		String
resultHeader (common)	結果を届けるヘッダー。		String
transport (common)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第14章 ATOMIX MULTIMAP COMPONENT

Camel バージョン 2.20 以降で利用可能

camel atomix-multimap コンポーネントを使用すると、Atomix の Distributed MultiMap コレクションを操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

14.1. URI 形式

```
atomix-multimap:multiMapName
```

Atomix MultiMap コンポーネントは、以下に示す 5個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (consumer)	共有コンポーネントの設定		AtomixMultiMapConfiguration
atomix (consumer)	共有 AtomixClient インスタンス		AtomixClient
nodes (consumer)	AtomixClient が接続する必要のあるノード		List
configurationUri (consumer)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix MultiMap エンドポイントは、URI 構文を使用して設定されます。

```
atomix-multimap:resourceName
```

パスおよびクエリーパラメーターを使用します。

14.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

14.1.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
atomix (consumer)	使用する Atomix インスタンス		Atomix
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
configurationUri (consumer)	Atomix 設定 uri。		String
defaultAction (consumer)	これがデフォルト動作です。	PUT	Action
key (consumer)	ヘッダーに何も設定されていない場合に使用するキー、または特定のキーのイベントをリッスンするキー。		Object
nodes (consumer)	クラスターを設定するノードのアドレス。		String
resultHeader (consumer)	結果を届けるヘッダー。		String
transport (consumer)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
ttl (consumer)	リソース ttl。		long

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第15章 ATOMIX キューコンポーネント

Camel バージョン 2.20 以降で利用可能

camel atomix-queue コンポーネントを使用すると、Atomix の 分散キュー コレクションを操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

15.1. URI 形式

```
atomix-queue:queueName
```

Atomix Queue コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有コンポーネントの設定		AtomixQueue Configuration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
nodes (common)	AtomixClient が接続する必要のあるノード		List
configurationUri (common)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix キューエンドポイントは、URI 構文を使用して設定されます。

```
atomix-queue:resourceName
```

パスおよびクエリーパラメーターを使用します。

15.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

15.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 uri。		String
defaultAction (common)	これがデフォルト動作です。	追加	Action
nodes (common)	クラスターを設定するノードのアドレス。		String
resultHeader (common)	結果を届けるヘッダー。		String
transport (common)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第16章 ATOMIX セットコンポーネント

Camel バージョン 2.20 以降で利用可能

camel atomix-set コンポーネントを使用すると、Atomix の 分散セット コレクションを操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

16.1. URI 形式

```
atomix-set:setName
```

Atomix Set コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有コンポーネントの設定		AtomixSetConfiguration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
nodes (common)	AtomixClient が接続する必要のあるノード		List
configurationUri (common)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Set エンドポイントは、URI 構文を使用して設定されます。

```
atomix-set:resourceName
```

パスおよびクエリーパラメーターを使用します。

16.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

16.1.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 uri。		String
defaultAction (common)	これがデフォルト動作です。	追加	Action
nodes (common)	クラスターを設定するノードのアドレス。		String
resultHeader (common)	結果を届けるヘッダー。		String
transport (common)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
ttl (common)	リソース ttl。		long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第17章 ATOMIX VALUE コンポーネント

Camel バージョン 2.20 以降で利用可能

camel atomix-value コンポーネントを使用すると、Atomix の Distributed Value を操作できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

17.1. URI 形式

```
atomix-value:valueName
```

Atomix Value コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有コンポーネントの設定		AtomixValue Configuration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
nodes (common)	AtomixClient が接続する必要のあるノード		List
configurationUri (common)	AtomixClient 設定へのパス		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Value エンドポイントは、URI 構文を使用して設定されます。

```
atomix-value:resourceName
```

パスおよびクエリーパラメーターを使用します。

17.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceName	**必須** 分散リソース名		String

17.1.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 uri。		String
defaultAction (common)	これがデフォルト動作です。	SET	Action
nodes (common)	クラスターを設定するノードのアドレス。		String
resultHeader (common)	結果を届けるヘッダー。		String
transport (common)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
ttl (common)	リソース ttl。		long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトのリソース設定。		Properties
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		Properties
ephemeral (advanced)	ローカルメンバーが PersistentMember としてグループに参加するかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取るため、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	読み取り一貫性レベル。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソース設定。		Map
resourceOptions (advanced)	ローカルリソースの設定		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第18章 AVRO コンポーネント

Camel バージョン 2.10 以降で利用可能

このコンポーネントは、avro のデータ形式を提供するので、Apache Avro のバイナリーデータ形式を使用したメッセージのシリアル化と逆シリアル化が可能になります。さらに、netty または http 経由で avro を使用するためのプロデューサーとコンシューマーのエンドポイントを提供することにより、Apache Avro の rpc のサポートを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-avro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

18.1. APACHE AVRO の概要

Avro を使用すると、json のような形式を使用してメッセージタイプとプロトコルを定義し、指定したタイプとメッセージの Java コードを生成できます。スキーマがどのように見えるかの例を以下に示します。

```
{"namespace": "org.apache.camel.avro.generated",
 "protocol": "KeyValueProtocol",

 "types": [
    {"name": "Key", "type": "record",
     "fields": [
         {"name": "key",   "type": "string"}
     ]
    },
    {"name": "Value", "type": "record",
     "fields": [
         {"name": "value",   "type": "string"}
     ]
    }
 ],

 "messages": {
    "put": {
       "request": [{"name": "key", "type": "Key"}, {"name": "value", "type": "Value"} ],
       "response": "null"
    },
    "get": {
       "request": [{"name": "key", "type": "Key"}],
       "response": "Value"
    }
 }
}
```

Maven、ant などを使用して、スキーマから簡単にクラスを生成できます。詳細には、Apache Avro のドキュメント を参照してください。

ただし、スキーマファーストのアプローチは強制されず、既存のクラスのスキーマを作成できます。**2.12 以降、** 既存のプロトコルインターフェイスを使用して RCP 呼び出しを行うことができます。プロトコル自体にはインターフェイスを使用し、パラメーターと結果の型には POJO Bean またはプリミティブ/文字列クラスを使用する必要があります。上記のスキーマに対応するクラスの例を次に示します。

```
package org.apache.camel.avro.reflection;

public interface KeyValueProtocol {
    void put(String key, Value value);
    Value get(String key);
}

class Value {
    private String value;
    public String getValue() { return value; }
    public void setValue(String value) { this.value = value; }
}
```

注: 既存のクラスは、データ形式ではなく、RPC (以下を参照) にのみ使用できます。

18.2. AVRO データ形式の使用

avro データ形式の使用は、ルートでマーシャリングまたはアンマーシャリングするクラスを指定するのと同様に簡単です。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:in"/>
        <marshal>
            <avro instanceClass="org.apache.camel.dataformat.avro.Message"/>
        </marshal>
        <to uri="log:out"/>
    </route>
</camelContext>
```

別の方法として、コンテキスト内でデータ形式を指定し、ルートから参照することもできます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
     <dataFormats>
        <avro id="avro" instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </dataFormats>
    <route>
        <from uri="direct:in"/>
        <marshal ref="avro"/>
        <to uri="log:out"/>
    </route>
</camelContext>
```

同じ方法で、avro データ形式を使用してアンマーシャリングできます。

18.3. CAMEL での AVRO RPC の使用

前述のように、Avro は http や netty などの複数のトランスポートを介した RPC サポートも提供します。Camel は、これら 2 つのトランスポートのコンシューマーとプロデューサーを提供します。

```
avro:[transport]:[host]:[port][?options]
```

現在サポートされているトランスポート値は http または netty です。

2.12 以降、 URI でメッセージ名を直接指定できます。

```
avro:[transport]:[host]:[port][/messageName][?options]
```

コンシューマーの場合、これにより、同じソケットに複数のルートを接続できます。正しいルートへのディスパッチは、avro コンポーネントによって自動的に行われます。messageName が指定されていないルート (存在する場合) がデフォルトとして使用されます。

avro ipc に camel プロデューサーを使用する場合、in メッセージボディーには、avro プロトコルで指定された操作のパラメーターを含める必要があります。応答は、out メッセージの本文に追加されます。

avro ipc に camel avro コンシューマーを使用する場合と同様に、リクエストパラメーターは、作成された exchange の in メッセージ本文内に配置され、exchange が処理されると、out メッセージの本文がレスポンスとして送信されます。.

注記: デフォルトでは、コンシューマーパラメーターは配列にラップされます。パラメーターが 1 つしかない場合は、**2.12 以降、singleParameter** URI オプションを使用して、配列をラップせずに in メッセージボディーで直接受け取ることができます。

18.4. AVRO RPC URI オプション

Avro コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 AvroConfiguration を使用してオプションを 1 回設定する場合		AvroConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Avro エンドポイントは、URI 構文を使用して設定されます。

```
avro:transport:host:port/messageName
```

パスおよびクエリーパラメーターを使用します。

18.4.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
transport	使用するために **必要な** トランスポート		AvroTransport
port	**必須** 使用するポート番号		int
host	使用する **必須** ホスト名		String
messageName	送信するメッセージの名前。		String

18.4.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
protocol (common)	使用する Avro プロトコル		Protocol
protocolClassName (common)	FQN クラス名で定義された使用する Avro プロトコル		String
protocolLocation (common)	Avro プロトコルのロケーション		String
reflectionProtocol (common)	指定されたプロトコルオブジェクトがリフレクションプロトコルの場合。protocolClassName ではプロトコルタイプが自動検出されるため、protocol パラメーターでのみ使用する必要があります。	false	boolean
singleParameter (common)	true の場合、consumer パラメーターは配列にラップされません。プロトコルがメッセージに複数のパラメーターを指定すると失敗します	false	boolean
uriAuthority (common)	使用する権限 (ユーザー名とパスワード)		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

18.5. AVRO RPC ヘッダー

名前	説明
CamelAvroMessageName	送信するメッセージの名前。コンシューマーでは、URI からのメッセージ名をオーバーライドします (存在する場合)

18.6. 例

http 経由で camel avro プロデューサーを使用する例:

```
        <route>
            <from uri="direct:start"/>
            <to uri="avro:http:localhost:{{avroport}}?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
            <to uri="log:avro"/>
        </route>
```

上記の例では、**CamelAvroMessageName** ヘッダーを入力する必要があります。**2.12 以降**、次の構文を使用して定数メッセージを呼び出すことができます。

```
        <route>
            <from uri="direct:start"/>
            <to uri="avro:http:localhost:{{avroport}}/put?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
            <to uri="log:avro"/>
        </route>
```

netty 経由で camel avro コンシューマーを使用してメッセージを消費する例:

```
        <route>
            <from uri="avro:netty:localhost:{{avroport}}?
```

```
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
        <choice>
          <when>
            <el>${in.headers.CamelAvroMessageName == 'put'}</el>
            <process ref="putProcessor"/>
          </when>
          <when>
            <el>${in.headers.CamelAvroMessageName == 'get'}</el>
            <process ref="getProcessor"/>
          </when>
        </choice>
      </route>
```

2.12 **以降**、同じタスクを実行するために 2 つの異なるルートを設定できます。

```
      <route>
        <from uri="avro:netty:localhost:{{avroport}}/put?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol">
        <process ref="putProcessor"/>
      </route>
      <route>
        <from uri="avro:netty:localhost:{{avroport}}/get?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol&singleParameter=true"/>
        <process ref="getProcessor"/>
      </route>
```

上記の例では、get はパラメーターを 1 つだけ受け取るため、**singleParameter** が使用され、**getProcessor は** 本文で直接 Value クラスを受け取りますが、 **putProcessor は** 配列の内容として文字列キーと値値が入力されたサイズ 2 の配列を受け取ります。

第19章 AVRO DATAFORMAT

Camel バージョン 2.14 以降で利用可能

このコンポーネントは、avro のデータ形式を提供するので、Apache Avro のバイナリーデータ形式を使用したメッセージのシリアル化と逆シリアル化が可能になります。さらに、netty または http 経由で avro を使用するためのプロデューサーとコンシューマーのエンドポイントを提供することにより、Apache Avro の rpc のサポートを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-avro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

19.1. APACHE AVRO の概要

Avro を使用すると、json のような形式を使用してメッセージタイプとプロトコルを定義し、指定したタイプとメッセージの Java コードを生成できます。スキーマがどのように見えるかの例を以下に示します。

```
{"namespace": "org.apache.camel.avro.generated",
 "protocol": "KeyValueProtocol",

 "types": [
    {"name": "Key", "type": "record",
     "fields": [
        {"name": "key",   "type": "string"}
     ]
    },
    {"name": "Value", "type": "record",
     "fields": [
        {"name": "value",   "type": "string"}
     ]
    }
 ],

 "messages": {
    "put": {
       "request": [{"name": "key", "type": "Key"}, {"name": "value", "type": "Value"} ],
       "response": "null"
    },
    "get": {
       "request": [{"name": "key", "type": "Key"}],
       "response": "Value"
    }
 }
}
```

Maven、ant などを使用して、スキーマから簡単にクラスを生成できます。詳細には、Apache Avro のドキュメント を参照してください。

ただし、スキーマファーストのアプローチは強制されず、既存のクラスのスキーマを作成できます。**2.12 以降、** 既存のプロトコルインターフェイスを使用して RCP 呼び出しを行うことができます。プロトコル自体にはインターフェイスを使用し、パラメーターと結果の型には POJO Bean またはプリミティブ/文字列クラスを使用する必要があります。上記のスキーマに対応するクラスの例を次に示します。

```
package org.apache.camel.avro.reflection;

public interface KeyValueProtocol {
    void put(String key, Value value);
    Value get(String key);
}

class Value {
    private String value;
    public String getValue() { return value; }
    public void setValue(String value) { this.value = value; }
}
```

注: 既存のクラスは、データ形式ではなく、RPC (以下を参照) にのみ使用できます。

19.2. AVRO データ形式の使用

avro データ形式の使用は、ルートでマーシャリングまたはアンマーシャリングするクラスを指定するのと同様に簡単です。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:in"/>
        <marshal>
            <avro instanceClass="org.apache.camel.dataformat.avro.Message"/>
        </marshal>
        <to uri="log:out"/>
    </route>
</camelContext>
```

別の方法として、コンテキスト内でデータ形式を指定し、ルートから参照することもできます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
     <dataFormats>
        <avro id="avro" instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </dataFormats>
    <route>
        <from uri="direct:in"/>
        <marshal ref="avro"/>
        <to uri="log:out"/>
    </route>
</camelContext>
```

同じ方法で、avro データ形式を使用してアンマーシャリングできます。

19.3. AVRO データ形式オプション

Avro データ形式は、以下にリストされている 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
instanceClassName		String	マーシャリングとアンマーシャリングに使用するクラス名
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

第20章 AWS CLOUDWATCH COMPONENT

Camel バージョン 2.11 以降で利用可能

CW コンポーネントを使用すると、メッセージを Amazon CloudWatch メトリクスに送信できます。Amazon API の実装は AWS SDK によって提供されます。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon CloudWatch を使用するためにサインアップしている必要がある。詳細については、Amazon CloudWatch を参照してください。

20.1. URI 形式

```
aws-cw://namespace[?options]
```

メトリクスが存在しない場合は作成されます。
URI には、 **?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

20.2. URI オプション

AWS CloudWatch コンポーネントは、以下に示す 5 のオプションをサポートします。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS CW のデフォルト設定		CwConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	CW クライアントが機能する必要があるリージョン。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS CloudWatch エンドポイントは、URI 構文を使用して設定されます。

```
aws-cw:namespace
```

パスおよびクエリーパラメーターを使用します。

20.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
namespace	**必須**。メトリクス namespace。		String

20.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
amazonCwClient (producer)	AmazonCloudWatch をクライアントとして使用します。		AmazonCloudWatch
name (producer)	メトリクス名		String
proxyHost (producer)	CW クライアントをインスタンス化する際にプロキシーホストを定義します。		String
proxyPort (producer)	CW クライアントをインスタンス化する際にプロキシーポートを定義します。		Integer
region (producer)	CW クライアントが機能する必要があるリージョン。		String
timestamp (producer)	メトリクスのタイムスタンプ。		Date
unit (producer)	メトリクスユニット		String
value (producer)	メトリクス値		double
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な CW コンポーネントオプション

Amazon の CloudWatch にアクセスするには、レジストリーに amazonCwClient を指定するか、accessKey と secretKey を指定する必要があります。

20.3. 使用方法

20.3.1. CW プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsCwMetricName	**String**	Amazon CW メトリクス名。
CamelAwsCwMetricValue	**double**	Amazon CW メトリクス値。
CamelAwsCwMetricUnit	**String**	Amazon CW メトリクスユニット。
CamelAwsCwMetricNamespace	**String**	Amazon CW メトリクス namespace。
CamelAwsCwMetricTimestamp	**日付**	Amazon CW メトリクスのタイムスタンプ。
CamelAwsCwMetricDimensionName	**String**	**Camel 2.12:** Amazon CW メトリクスディメンション名。
CamelAwsCwMetricDimensionValue	**String**	**Camel 2.12:** Amazon CW メトリクスディメンション値。

ヘッダー	タイプ	説明
CamelAwsCwMetricDimensions	**Map<String, String>**	Camel 2.12: ディメンション名とディメンション値のマップ。

20.3.2. Advanced AmazonCloudWatch configuration

AmazonCloudWatch インスタンス設定をさらに制御する必要がある場合は、独自のインスタンスを作成し、URI から参照できます。

```
from("direct:start")
.to("aws-cw://namepsace?amazonCwClient=#client");
```

#client は、レジストリー内の **AmazonCloudWatch** を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonCloudWatch client = new AmazonCloudWatchClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

20.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.10 以降) に置き換える必要があります。

20.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド
- AWS コンポーネント

第21章 AWS DYNAMODB COMPONENT

Camel バージョン 2.10 以降で利用可能

DynamoDB コンポーネントは、Amazon の DynamoDB サービスとの間でのデータの保存と取得をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon DynamoDB を使用するためにサインアップしている必要がある。詳細については、Amazon DynamoDB を参照してください。

21.1. URI 形式

```
aws-ddb://domainName[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

21.2. URI オプション

AWS DynamoDB コンポーネントは 5 のオプションをサポートします。これは以下に記載されています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS DDB のデフォルト設定		DdbConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	DDB クライアントが機能する必要があるリージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS DynamoDB エンドポイントは、URI 構文を使用して設定します。

```
aws-ddb:tableName
```

パスおよびクエリーパラメーターを使用します。

21.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
tableName	**必須**。現在作業中のテーブルの名前。		String

21.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
amazonDDBClient (producer)	AmazonDynamoDB をクライアントとして使用します		AmazonDynamoDB
consistentRead (producer)	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。	false	boolean
keyAttributeName (producer)	テーブルの作成時の属性名		String
keyAttributeType (producer)	テーブル作成時の属性タイプ		String
operation (producer)	実行する操作	PutItem	DdbOperations
proxyHost (producer)	DDB クライアントをインスタンス化する際にプロキシーホストを定義します		String
proxyPort (producer)	DDB クライアントをインスタンス化する際にプロキシーポートを定義します。		Integer
readCapacity (producer)	テーブルからリソースを読み取るために予約するプロビジョニングされたスループット		Long
region (producer)	DDB クライアントが機能する必要があるリージョン		String
writeCapacity (producer)	テーブルにリソースを書き込むために予約するプロビジョニングされたスループット。		Long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な DDB コンポーネントオプション

Amazon の DynamoDB にアクセスするには、レジストリーに amazonDDBClient を指定するか、accessKey と secretKey を指定する必要があります。

21.3. 使用方法

21.3.1. DDB プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbBatchItems	**Map<String, KeysAndAttributes>**	プライマリーキーによって取得するテーブル名と対応する項目のマップ。
CamelAwsDdbTableName	**String**	この操作のテーブル名。
CamelAwsDdbKey	**キー**	テーブル内の各項目を一意に識別するプライマリーキー。**Camel 2.16.0 以降、このヘッダーのタイプは Key ではなく Map<String、AttributeValue> です。**
CamelAwsDdbReturnValues	**String**	変更前または変更後の属性の名前および値のペアを取得する場合は、このパラメーターを使用します (NONE、ALL_OLD、UPDATED_OLD、ALL_NEW、UPDATED_NEW)。
CamelAwsDdbUpdateCondition	**Map<String, ExpectedAttributeValue>**	条件変更の属性を指定します。
CamelAwsDdbAttributeNames	**Collection<String>**	属性名が指定されていない場合、すべての属性が返されます。

ヘッダー	タイプ	説明
CamelAwsDdbConsistentRead	**Boolean**	true に設定すると、一貫性のある読み取りが発行されます。それ以外の場合は、最終的に一貫性が使用されます。
CamelAwsDdbIndexName	**String**	設定されている場合、クエリー操作のセカンダリーインデックスとして使用されます。
CamelAwsDdbItem	**Map<String, AttributeValue>**	アイテムの属性のマップ。アイテムを定義するプライマリーキー値を含める必要があります。
CamelAwsDdbExactCount	**Boolean**	true に設定すると、Amazon DynamoDB は、一致する項目とその属性のリストではなく、クエリーパラメーターに一致する項目の総数を返します。**Camel 2.16.0 から、このヘッダーはもう存在しません。**
CamelAwsDdbKeyConditions	**Map<String, Condition>**	From Camel 2.16.0.このヘッダーはクエリーの選択基準を指定し、2 つの古いヘッダー CamelAwsDdbHashKeyValue および CamelAwsDdbScanRangeKeyCondition をマージします。
CamelAwsDdbStartKey	**キー**	以前のクエリーを続行するアイテムのプライマリーキー。
CamelAwsDdbHashKeyValue	**AttributeValue**	複合プライマリーキーのハッシュコンポーネントの値。**Camel 2.16.0 から、このヘッダーはもう存在しません。**
CamelAwsDdbLimit	**Integer**	返すアイテムの最大数。

ヘッダー	タイプ	説明
CamelAwsDdbScanRangeKeyCondition	**状態**	クエリーに使用する属性値と比較演算子のコンテナー。**Camel 2.16.0 から、このヘッダーはもう存在しません。**
CamelAwsDdbScanIndexForward	**Boolean**	インデックスの順方向または逆方向のトラバーサルを指定します。
CamelAwsDdbScanFilter	**Map<String, Condition>**	スキャン結果を評価し、目的の値のみを返します。
CamelAwsDdbUpdateValues	**Map<String, AttributeValueUpdate>**	更新の新しい値とアクションへの属性名のマップ。

21.3.2. BatchGetItems 操作中に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbBatchResponse	**Map<String,BatchResponse>**	テーブル名およびテーブルの各項目属性。

ヘッダー	タイプ	説明
CamelAwsDdbUnprocessedKeys	**Map<String,KeysAndAttributes>**	テーブルのマップと、現在の応答で処理されなかった対応するキーが含まれます。

21.3.3. DeleteItem 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧。

21.3.4. DeleteTable 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbProvisionedThroughput		
ProvisionedThroughputDescription		このテーブルの ProvisionedThroughput プロパティーの値
CamelAwsDdbCreationDate	**日付**	このテーブルの DateTime の作成。

ヘッダー	タイプ	説明
CamelAwsDdbTableItemCount	**Long**	このテーブルのアイテム数。
CamelAwsDdbKeySchema	**KeySchema**	このテーブルのプライマリーキーを識別する KeySchema。**Camel 2.16.0 以降、このヘッダーのタイプは List<KeySchemaElement> であり、KeySchema ではありません。**
CamelAwsDdbTableName	**String**	テーブル名。
CamelAwsDdbTableSize	**Long**	テーブルサイズ（バイト単位）。
CamelAwsDdbTableStatus	**String**	テーブルのステータス： CREATING、UPDATING、DELETING、ACTIVE

21.3.5. DescribeTable 操作中に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbProvisionedThroughput	{{ProvisionedThroughputDescription}}	このテーブルの ProvisionedThroughput プロパティーの値

ヘッダー	タイプ	説明
CamelAwsDdbCreationDate	**日付**	このテーブルの DateTime の作成。
CamelAwsDdbTableItemCount	**Long**	このテーブルのアイテム数。
CamelAwsDdbKeySchema	{{KeySchema}}	このテーブルのプライマリーキーを識別する KeySchema。**Camel 2.16.0 以降、このヘッダーのタイプは List<KeySchemaElement> であり、KeySchema ではありません。**
CamelAwsDdbTableName	**String**	テーブル名。
CamelAwsDdbTableSize	**Long**	テーブルサイズ（バイト単位）。
CamelAwsDdbTableStatus	**String**	テーブルのステータス： CREATING、UPDATING、DELETING、ACTIVE
CamelAwsDdbReadCapacity	**Long**	このテーブルの ReadCapacityUnits プロパティー。
CamelAwsDdbWriteCapacity	**Long**	このテーブルの WriteCapacityUnits プロパティー。

21.3.6. GetItem 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧。

21.3.7. PutItem 操作中に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧。

21.3.8. Query 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	操作によって返される属性の一覧。
CamelAwsDdbLastEvaluatedKey	**キー**	前の結果セットを含む、クエリー操作が停止した項目のプライマリーキー。
CamelAwsDdbConsumedCapacity	**double**	操作中に消費された、テーブルのプロビジョニングされたスループットのキャパシティーユニットの数。

ヘッダー	タイプ	説明
CamelAwsDdbCount	**Integer**	応答のアイテム数。

21.3.9. Scan 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	操作によって返される属性の一覧。
CamelAwsDdbLastEvaluatedKey	**キー**	前の結果セットを含む、クエリー操作が停止した項目のプライマリーキー。
CamelAwsDdbConsumedCapacity	**double**	操作中に消費された、テーブルのプロビジョニングされたスループットのキャパシティーユニットの数。
CamelAwsDdbCount	**Integer**	応答のアイテム数。
CamelAwsDdbScannedCount	**Integer**	フィルターが適用される前の完全なスキャン内のアイテムの数。

21.3.10. UpdateItem 操作時に設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧。

21.3.11. 高度な AmazonDynamoDB 設定

AmazonDynamoDB インスタンス設定をさらに制御する必要がある場合は、独自のインスタンスを作成し、URI から参照できます。

```
from("direct:start")
.to("aws-ddb://domainName?amazonDDBClient=#client");
```

#client は、レジストリー内の **AmazonDynamoDB** を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonDynamoDB client = new AmazonDynamoDBClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

21.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.10 以降) に置き換える必要があります。

21.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- スタートガイド
- AWS コンポーネント

第22章 AWS DYNAMODB STREAMS COMPONENT

Camel バージョン 2.17 以降で利用可能

DynamoDB Stream コンポーネントは、Amazon DynamoDB Stream サービスからのメッセージの受信をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon DynamoDB ストリームを使用するためにサインアップしている必要がある。詳細については、AWS DynamoDB を参照してください。

22.1. URI 形式

```
aws-ddbstream://table-name[?options]
```

ストリームは、使用する前に作成する必要があります。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

22.2. URI オプション

AWS DynamoDB Streams コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS DDB ストリームのデフォルト設定		DdbStreamConfiguration
accessKey (consumer)	Amazon AWS Access Key		String
secretKey (consumer)	Amazon AWS Secret Key		String
region (consumer)	Amazon AWS リージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS DynamoDB Streams エンドポイントは、URI 構文を使用して設定されます。

```
aws-ddbstream:tableName
```

パスおよびクエリーパラメーターを使用します。

22.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
tableName	**必須** dynamodb テーブルの名前		String

22.2.2. クエリーパラメーター(28 パラメーター):

名前	説明	デフォルト	タイプ
amazonDynamoDbStreams Client (consumer)	このエンドポイントに対するすべての要求に使用する Amazon DynamoDB クライアント		AmazonDynamoDBStreams
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
iteratorType (consumer)	DynaboDB ストリーム内でレコードの取得を開始する場所を定義します。TRIM_HORIZON を使用すると、ストリームがリアルタイムに追いつく前に大幅な遅延が発生する可能性があることに注意してください。AT,AFTER_SEQUENCE_NUMBER を使用する場合は、sequenceNumberProvider を指定する必要があります。	LATEST	ShardIteratorType
maxResultsPerRequest (consumer)	各ポーリングでフェッチされる最大レコード数。		int
proxyHost (consumer)	DDBStreams クライアントをインスタンス化する際にプロキシーホストを定義します		String
proxyPort (consumer)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します		Integer
region (consumer)	DDBStreams クライアントが機能する必要があるリージョン		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

名前	説明	デフォルト	タイプ
sequenceNumberProvider (consumer)	2 つの ShardIteratorType.AT,AFTER_SEQUENCE_NUMBER イテレーター型のいずれかを使用する場合のシーケンス番号のプロバイダー。レジストリー参照またはリテラルシーケンス番号を指定できます。		SequenceNumberProvider
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long

名前	説明	デフォルト	タイプ
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必須の DynampDBStream コンポーネントオプション

レジストリーに amazonDynamoDbStreamsClient を提供し、プロキシーと関連する認証情報を設定する必要があります。

22.3. シーケンス番号

シーケンス番号としてリテラル文字列を指定するか、レジストリーに Bean を指定できます。Bean を使用する例としては、現在の位置を変更フィードに保存し、Camel の起動時に復元することが挙げられます。

describe-streams の結果で最大のシーケンス番号よりも大きいシーケンス番号を指定すると、AWS 呼び出しが HTTP 400 を返すことになるため、エラーになります。

22.4. バッチコンシューマー

このコンポーネントは、Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージの数を知ることができ、たとえば、Aggregator にこの数のメッセージを集約させることができます。

22.5. 使用方法

22.5.1. AmazonDynamoDBStreamsClient configuration

AmazonDynamoDBStreamsClient のインスタンスを作成し、それをレジストリーに BIND する必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
region.createClient(AmazonDynamoDBStreamsClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
DefaultAWSCredentialsProviderChain

registry.bind("kinesisClient", client);
```

22.5.2. AWS 認証情報の指定

新しい ClientConfiguration インスタンスを作成するときのデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することをお勧めしますが、createClient (...) を呼び出すときに別の AWSCredentialsProvider を指定できます。

22.6. ダウンタイムへの対処

22.6.1. 24 時間未満の AWS DynamoDB ストリームの停止

コンシューマーは最後に確認されたシーケンス番号 (CAMEL-9515 で実装) から再開されるため、停止に DynamoDB 自体が含まれていない限り、大量のイベントをすばやく連続して受信する必要があります。

22.6.2. AWS DynamoDB Streams が 24 時間以上停止する

AWS が 24 時間分の変更しか保持しないことを考えると、どのような軽減策が実施されていても、変更イベントを見逃すことになります。

22.7. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.7 以降) に置き換える必要があります。

22.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント
  +

第23章 AWS EC2 コンポーネント

Camel バージョン 2.16 以降で利用可能

EC2 コンポーネントは、AWS EC2 インスタンスの作成、実行、開始、停止、および終了をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon EC2 を使用するためにサインアップしている必要がある。詳細については、Amazon EC2 を参照してください。

23.1. URI 形式

```
aws-ec2:///label[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

23.2. URI オプション

AWS EC2 コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS EC2 のデフォルト設定		EC2 設定
region (producer)	EC2 クライアントが動作する必要があるリージョン		String
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS EC2 エンドポイントは、URI 構文を使用して設定されます。

```
aws-ec2:label
```

パスおよびクエリーパラメーターを使用します。

23.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
label	**必須** 論理名		String

23.2.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
accessKey (producer)	Amazon AWS Access Key		String
amazonEc2Client (producer)	既存の設定済み AmazonEC2Client をクライアントとして使用するには		AmazonEC2Client
operation (producer)	**必須** 実行する操作。createAndRunInstances、startInstances、stopInstances、terminateInstances、describeInstances、describeInstancesStatus、rebootInstances、monitorInstances、unmonitorInstances、createTags、または deleteTags のいずれかです。		EC2 オペレーション
proxyHost (producer)	EC2 クライアントをインスタンス化する際にプロキシーホストを定義します。		String
proxyPort (producer)	EC2 クライアントをインスタンス化する際にプロキシーポートを定義します。		Integer
region (producer)	EC2 クライアントが動作する必要があるリージョン		String
secretKey (producer)	Amazon AWS Secret Key		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必要な EC2 コンポーネントオプション

Amazon EC2 サービスにアクセスするには、レジストリーに amazonEc2Client を指定するか、accessKey と secretKey を指定する必要があります。

23.3. 使用方法

23.3.1. EC2 プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsEC2ImageId	**String**	AWS マーケットプレイスのイメージ ID
CamelAwsEC2InstanceType	com.amazonaws.services.ec2.model.InstanceType	作成して実行するインスタンスタイプ
CamelAwsEC2Operation	**String**	実行する操作
CamelAwsEC2InstanceMinCount	**Int**	実行したいインスタンスの最小数。
CamelAwsEC2InstanceMaxCount	**Int**	実行したいインスタンスの最大数。
CamelAwsEC2InstanceMonitoring	Boolean	実行中のインスタンスを監視するかどうかを定義します
CamelAwsEC2InstanceEbsOptimized	**Boolean**	作成中のインスタンスが EBS I/O 用に最適化されているかどうかを定義します。

ヘッダー	タイプ	説明
CamelAwsEC2InstanceSecurityGroups	コレクション	インスタンスに関連付けるセキュリティーグループ
CamelAwsEC2Instancesids	**コレクション**	開始、停止、記述、および終了操作を実行するインスタンス IDS のコレクション。
CamelAwsEC2InstancesTags	**コレクション**	EC2 リソースから追加または削除するタグのコレクション

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.16 以降) に置き換える必要があります。

23.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第24章 AWS KINESIS コンポーネント

Camel バージョン 2.17 以降で利用可能

Kinesis コンポーネントは、Amazon Kinesis サービスとのメッセージの送受信をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon Kinesis を使用するためにサインアップしている必要がある。詳細については、AWS Kinesis を参照してください

24.1. URI 形式

```
aws-kinesis://stream-name[?options]
```

ストリームは、使用する前に作成する必要があります。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

24.2. URI オプション

AWS DynamoDB コンポーネントは 5 個のオプションをサポートします。これは以下に記載されています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS S3 のデフォルト設定		KinesisConfiguration
accessKey (common)	Amazon AWS Access Key		String
secretKey (common)	Amazon AWS Secret Key		String
region (common)	Amazon AWS リージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Kinesis エンドポイントは、URI 構文を使用して設定します。

```
aws-kinesis:streamName
```

パスおよびクエリーパラメーターを使用します。

24.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
streamName	**必須** ストリームの名前。		String

24.2.2. クエリーパラメーター (30 パラメーター)

名前	説明	デフォルト	タイプ
amazonKinesisClient (common)	このエンドポイントに対するすべての要求に使用する Amazon Kinesis クライアント。		AmazonKinesis
proxyHost (common)	DDBStreams クライアントをインスタンス化する際にプロキシーホストを定義します		String
proxyPort (common)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します		Integer
region (common)	Kinesis クライアントが動作する必要があるリージョン		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
iteratorType (consumer)	Kinesis ストリーム内でレコードの取得を開始する場所を定義します	TRIM_HORIZON	ShardIteratorType
maxResultsPerRequest (consumer)	各ポーリングでフェッチされる最大レコード数。	1	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sequenceNumber (consumer)	ポーリングを開始するシーケンス番号。iteratorType が AFTER_SEQUENCE_NUMBER または AT_SEQUENCE_NUMBER に設定されている場合に必要です。		String

名前	説明	デフォルト	タイプ
shardClosed (consumer)	シャード (shard) が閉じられた場合の動作を定義します。設定可能な値は、ignore、silent、fail です。ignore の場合、メッセージはログに記録され、コンシューマーは最初から再起動します。silent の場合は、ログには記録されず、コンシューマーは最初から起動します。fail の場合は、ReachedClosedStateException が発生します。	ignore	KinesisShardClosed StrategyEnum
shardId (consumer)	Kinesis ストリームでどの shardId からレコードを取得するかを定義します。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

名前	説明	デフォルト	タイプ
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な Kinesis コンポーネントオプション

プロキシーと関連するクレデンシャル情報が設定された状態で、レジストリーに amazonKinesisClient を提供する必要があります。

24.3. バッチコンシューマー

このコンポーネントは、Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージの数を知ることができ、たとえば、Aggregator にこの数のメッセージを集約させることができます。

24.4. 使用方法

24.4.1. Kinesis コンシューマーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsKinesisSequenceNumber	**String**	このレコードのシーケンス番号。これは、サイズが API によって定義されていないため、文字列として表されます。数値型として使用する場合は、次を使用します
CamelAwsKinesisApproximateArrivalTimestamp	**String**	AWS がレコードの到着時間として割り当てた時間。
CamelAwsKinesisPartitionKey	**String**	データレコードが割り当てられているストリーム内のシャードを識別します。

24.4.2. AmazonKinesis の設定

AmazonKinesisClient のインスタンスを作成し、それをレジストリーに BIND する必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
region.createClient(AmazonKinesisClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
```

```
DefaultAWSCredentialsProviderChain

registry.bind("kinesisClient", client);
```

次に、**amazonKinesisClient** URI オプションで AmazonKinesisClient を参照する必要があります。

```
from("aws-kinesis://mykinesisstream?amazonKinesisClient=#kinesisClient")
  .to("log:out?showAll=true");
```

24.4.3. AWS 認証情報の指定

新しい ClientConfiguration インスタンスを作成するときのデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することをお勧めしますが、createClient (...) を呼び出すときに別の AWSCredentialsProvider を指定できます。

24.4.4. Kinesis プロデューサーが Kinesis に書き込むために使用するメッセージヘッダー。プロデューサーは、メッセージ本文が **ByteBuffer** であることを期待しています。

ヘッダー	タイプ	説明
CamelAwsKinesisPartitionKey	**String**	このレコードを保存するために Kinesis に渡す PartitionKey。
CamelAwsKinesisSequenceNumber	**String**	このレコードのシーケンス番号を示すオプションのパラメーター。

24.4.5. レコードの保存が成功したときに Kinesis プロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsKinesisSequenceNumber	**String**	Response Syntax で定義されているレコードのシーケンス番号

ヘッダー	タイプ	説明
CamelAwsKinesisShardId	**String**	レコードが保存された場所のシャード ID

24.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.17 以降) に置き換える必要があります。

24.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第25章 AWS KINESIS FIREHOSE コンポーネント

Camel バージョン 2.19 以降で利用可能

Kinesis Firehose コンポーネントは、Amazon Kinesis Firehose サービスへのメッセージの送信をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon Kinesis Firehose を使用するためにサインアップしている必要がある。詳細については、AWS Kinesis Firehose をご覧ください。

25.1. URI 形式

```
aws-kinesis-firehose://delivery-stream-name[?options]
```

ストリームは、使用する前に作成する必要があります。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

25.2. URI オプション

AWS Kinesis Firehose コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS Kinesis Firehose のデフォルト設定		KinesisFirehose Configuration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	Amazon AWS リージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Kinesis Firehose エンドポイントは、URI 構文を使用して設定されます。

```
aws-kinesis-firehose:streamName
```

パスおよびクエリーパラメーターを使用します。

25.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
streamName	**必須** ストリームの名前。		String

25.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
amazonKinesisFirehoseClient (producer)	このエンドポイントのすべてのリクエストに使用する Amazon Kinesis Firehose クライアント		AmazonKinesisFirehose
proxyHost (producer)	DDBStreams クライアントをインスタンス化する際にプロキシーホストを定義します		String
proxyPort (producer)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します		Integer
region (producer)	Kinesis クライアントが動作する必要があるリージョン		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な Kinesis Firehose コンポーネントのオプション

プロキシーと関連するクレデンシャル情報が設定された状態で、レジストリーに amazonKinesisClient を提供する必要があります。

25.3. 使用方法

25.3.1. Amazon Kinesis Firehose の設定

AmazonKinesisClient のインスタンスを作成し、それをレジストリーに BIND する必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
```

```
region.createClient(AmazonKinesisClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
DefaultAWSCredentialsProviderChain

registry.bind("kinesisFirehoseClient", client);
```

次に、**amazonKinesisFirehoseClient** URI オプションで AmazonKinesisFirehoseClient を参照する必要があります。

```
from("aws-kinesis-firehose://mykinesisdeliverystream?amazonKinesisFirehoseClient=#kinesisClient")
  .to("log:out?showAll=true");
```

25.3.2. AWS 認証情報の指定

新しい ClientConfiguration インスタンスを作成するときのデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することをお勧めしますが、createClient (...) を呼び出すときに別の AWSCredentialsProvider を指定できます。

25.3.3. レコードの保存が成功したときに Kinesis プロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsKinesisFirehoseRecordId	**String**	応答構文 で定義されているレコード ID

25.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は、Camel の実際のバージョン (2.19 以降) に置き換える必要があります。

25.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド
- AWS コンポーネント

第26章 AWS KMS コンポーネント

Camel バージョン 2.21 以降で利用可能

KMS コンポーネントは、AWS KMS インスタンスの作成、実行、開始、停止、および終了をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon KMS を使用するためにサインアップしている必要がある。詳細については、Amazon KMS を参照してください。

26.1. URI 形式

```
aws-kms://label[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

26.2. URI オプション

AWS KMS コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS MQ のデフォルト設定		KMSConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	MQ クライアントが動作する必要がある地域		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS KMS エンドポイントは、URI 構文を使用して設定されます。

```
aws-kms:label
```

パスおよびクエリーパラメーターを使用します。

26.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
label	**必須** 論理名		String

26.2.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
accessKey (producer)	Amazon AWS Access Key		String
kmsClient (producer)	既存の設定済み AWS KMS をクライアントとして使用する場合		AWSKMS
operation (producer)	**必須** 実行する操作		KMSOperations
proxyHost (producer)	KMS クライアントをインスタンス化するときにプロキシーホストを定義します。		String
proxyPort (producer)	KMS クライアントをインスタンス化するときにプロキシーポートを定義します。		Integer
region (producer)	KMS クライアントが動作する必要があるリージョン		String
secretKey (producer)	Amazon AWS Secret Key		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必須の KMS コンポーネントオプション

Amazon KMS サービスにアクセスするには、レジストリーに amazonKmsClient を指定するか、accessKey と secretKey を指定する必要があります。

26.3. 使用方法

26.3.1. MQ プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsKMSLimit	**Integer**	listKeys 操作の実行中に返されるキーの制限数
CamelAwsKMSOperation	**String**	実行する操作
CamelAwsKMSDescription	**String**	createKey 操作の実行中に使用するキーの説明
CamelAwsKMSKeyId	**String**	キー ID

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.16 以降) に置き換える必要があります。

26.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第27章 AWS LAMBDA コンポーネント

Camel バージョン 2.20 以降で利用可能

Lambda コンポーネントは、AWS Lambda 関数の作成、取得、一覧表示、削除、および呼び出しをサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon Lambda を使用するためにサインアップしている必要がある。詳細については、Amazon Lambda を参照してください。

Lambda 関数を作成するときは、少なくとも AWSLambdaBasicExecuteRole ポリシーがアタッチされた IAM ロールを指定する必要があります。

Warning

Lambda は地域サービスです。S3 バケットとは異なり、特定のリージョンで作成された Lambda 関数は他のリージョンでは使用できません。

27.1. URI 形式

```
aws-lambda://functionName[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

27.2. URI オプション

AWS Lambda コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS Lambda のデフォルト設定		LambdaConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	Amazon AWS リージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Lambda エンドポイントは、URI 構文を使用して設定されます。

```
aws-lambda:function
```

パスおよびクエリーパラメーターを使用します。

27.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
function	**必須** Lambda 関数の名前。		String

27.2.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
operation (producer)	**必須** 実行する操作。listFunctions、getFunction、createFunction、deleteFunction、または invokeFunction のいずれかです		LambdaOperations
region (producer)	Amazon AWS リージョン		String
awsLambdaClient (advanced)	既存の設定済みの AwsLambdaClient をクライアントとして使用する場合		AWSLambda
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
proxyHost (proxy)	Lambda クライアントをインスタンス化するときにプロキシーホストを定義します		String
proxyPort (proxy)	Lambda クライアントをインスタンス化するときにプロキシーポートを定義します		Integer
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な Lambda コンポーネントオプション

Amazon Lambda サービスにアクセスするには、レジストリーに awsLambdaClient を指定するか、accessKey と secretKey を指定する必要があります

27.3. 使用方法

27.3.1. Lambda プロデューサーによって評価されるメッセージヘッダー

操作	ヘッダー	タイプ	説明	必須
すべて	**CamelAwsLambdaOperation**	**String**	実行する操作。クエリーパラメーターとして渡されたオーバーライド操作	はい
createFunction	**CamelAwsLambdaS3Bucket**	**String**	デプロイパッケージを含む .zip ファイルが保存される Amazon S3 バケット名。このバケットは、Lambda 関数を作成しているのと同じ AWS リージョンに存在する必要があります。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaS3Key**	**String**	アップロードする Amazon S3 オブジェクト (デプロイパッケージ) のキー名。	いいえ
createFunction	**CamelAwsLambdaS3ObjectVersion**	String	アップロードする Amazon S3 オブジェクト (デプロイパッケージ) のバージョン。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaZipFile**	**String**	zip ファイル (デプロイメントパッケージ) のローカルパス。zip ファイルの内容をメッセージ本文に入れることもできます。	いいえ
createFunction	**CamelAwsLambdaRole**	**String**	Lambda が関数を実行して他のアマゾンウェブサービス (AWS) リソースにアクセスするときに引き受ける IAM ロールの Amazon リソースネーム (ARN)。	はい

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaRuntime**	String	アップロードする Lambda 関数のランタイム環境。(nodejs、nodejs4.3、nodejs6.10、java8、python2.7、python3.6、dotnetcore1.0、odejs4.3-edge)	はい
createFunction	**CamelAwsLambdaHandler**	**String**	実行を開始するために Lambda が呼び出すコード内の関数。Node.js の場合は、関数の module-name.export 値です。Java の場合は、package.class-name::handler または package.class-name にすることができます。	はい

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaDescription**	**String**	ユーザー提供の説明。	いいえ
createFunction	**CamelAwsLambdaTargetArn**	**String**	Amazon SQS キューまたは Amazon SNS トピックのターゲット ARN (Amazon リソースネーム) を含む親オブジェクト。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaMemorySize**	**Integer**	関数用に設定したメモリーサイズ (MB 単位)。64 MB の倍数である必要があります。	いいえ
createFunction	**CamelAwsLambdaKMSKeyArn**	**String**	関数の環境変数を暗号化するために使用される KMS キーの Amazon リソースネーム (ARN)。指定しない場合、AWS Lambda はデフォルトのサービスキーを使用します。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaPublish**	**Boolean**	このブール値パラメーターを使用して、AWS Lambda に Lambda 関数を作成し、バージョンをアトミック操作として発行するようにリクエストできます。	いいえ
createFunction	**CamelAwsLambdaTimeout**	**Integer**	Lambda が関数を終了する関数実行時間。デフォルトは 3 秒です。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaTracingConfig**	**String**	関数のトレース設定 (Active または PassThrough)。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaEnvironmentVariables**	**Map<String, String>**	環境の設定設定を表すキーと値のペア。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaEnvironmentTags**	**Map<String, String>**	新しい関数に割り当てられたタグ (キーと値のペア) のリスト。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaSecurityGroupIds**	**List<String>**	Lambda 関数が VPC 内のリソースにアクセスする場合、VPC 内の 1 つ以上のセキュリティーグループ ID のリスト。	いいえ

操作	ヘッダー	タイプ	説明	必須
createFunction	**CamelAwsLambdaSubnetIds**	**List<String>**	Lambda 関数が VPC 内のリソースにアクセスする場合、VPC 内の 1 つ以上のサブネット ID のリスト。	なし

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.16 以降) に置き換える必要があります。

27.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第28章 AWS MQ コンポーネント

Camel バージョン 2.21 以降で利用可能

EC2 コンポーネントは、AWS MQ インスタンスの作成、実行、開始、停止、および終了をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon MQ を使用するためにサインアップしている必要がある。詳細については、Amazon MQ を参照してください。

28.1. URI 形式

```
aws-mq://label[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

28.2. URI オプション

AWS MQ コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS MQ のデフォルト設定		MQConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	MQ クライアントが動作する必要がある地域		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS MQ エンドポイントは、URI 構文を使用して設定されます。

```
aws-mq:label
```

パスおよびクエリーパラメーターを使用します。

28.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
label	**必須** 論理名		String

28.2.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
accessKey (producer)	Amazon AWS Access Key		String
amazonMqClient (producer)	既存の設定済み AmazonMQClient をクライアントとして使用するには		AmazonMQ
operation (producer)	**必須** 実行する操作。It can be listBrokers,createBroker,deleteBroker		MQ 操作
proxyHost (producer)	MQ クライアントをインスタンス化する際にプロキシーホストを定義します。		String
proxyPort (producer)	MQ クライアントをインスタンス化する際にプロキシーポートを定義します。		Integer
region (producer)	MQ クライアントが動作する必要がある地域		String
secretKey (producer)	Amazon AWS Secret Key		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必要な EC2 コンポーネントオプション

Amazon EC2 サービスにアクセスするには、レジストリーに amazonEc2Client を指定するか、accessKey と secretKey を指定する必要があります。

28.3. 使用方法

28.3.1. MQ プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsMQMaxResults	**String**	listBrokers オペレーションから取得する必要がある結果の数
CamelAwsMQBrokerName	**String**	ブローカー名
CamelAwsMQOperation	**String**	実行する操作
CamelAwsMQBrokerId	**String**	ブローカー ID
CamelAwsMQBrokerDeploymentMode	**String**	createBroker 操作でのブローカーのデプロイメントモード

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.16 以降) に置き換える必要があります。

28.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド
- AWS コンポーネント

第29章 AWS S3 ストレージサービスコンポーネント

Camel バージョン 2.8 以降で利用可能

S3 コンポーネントは、Amazon の S3 サービスとの間でのオブジェクトの保存と取得をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon S3 を使用するためにサインアップしている必要がある。詳細については、Amazon S3 を参照してください。

29.1. URI 形式

```
aws-s3://bucketNameOrArn[?options]
```

バケットがまだ存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、バケット **helloBucket** からファイル **hello.txt** を読み取るには、次のスニペットを使用します。

```
from("aws-s3:helloBucket?accessKey=yourAccessKey&secretKey=yourSecretKey&prefix=hello.txt")
  .to("file:/var/downloaded");
```

29.2. URI オプション

AWS S3 Storage Service コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS S3 のデフォルト設定		S3 設定
accessKey (common)	Amazon AWS Access Key		String
secretKey (common)	Amazon AWS Secret Key		String
region (common)	バケットが配置されているリージョン。このオプションは、com.amazonaws.services.s3.model.CreateBucketRequest で使用されます。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS S3 Storage Service エンドポイントは、URI 構文を使用して設定されます。

```
aws-s3:bucketNameOrArn
```

パスおよびクエリーパラメーターを使用します。

29.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
bucketNameOrArn	**必須** バケット名または ARN		String

29.2.2. クエリーパラメーター(50 パラメーター):

名前	説明	デフォルト	タイプ
amazonS3Client (common)	link:registry.htmlRegistry の com.amazonaws.services.sqs.AmazonS3 を参照してください。		AmazonS3
pathStyleAccess (common)	S3 クライアントがパススタイルアクセスを使用するかどうか	false	boolean
policy (common)	com.amazonaws.services.s3.AmazonS3setBucketPolicy() メソッドに設定されるこのキューのポリシー。		文字列
proxyHost (common)	SQS クライアントをインスタンス化するときにプロキシーホストを定義します。		String
proxyPort (common)	クライアント定義内で使用されるプロキシーポートを指定します。		Integer
region (common)	S3 クライアントが機能する必要があるリージョン		String
useIAMCredentials (common)	S3 クライアントが EC2 インスタンスに認証情報をロードすることを期待するか、静的認証情報が渡されることを期待するかを設定します。	false	boolean
encryptionMaterials (common)	対称/非対称クライアントを使用する場合に使用する暗号化マテリアル		EncryptionMaterials
useEncryption (common)	暗号化を使用する必要があるかどうかを定義する	false	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
deleteAfterRead (consumer)	取得後に S3 からオブジェクトを削除します。削除は、エクスチェンジがコミットされた場合にのみ実行されます。ロールバックが発生すると、オブジェクトは削除されません。このオプションが false の場合、同じオブジェクトがポーリングで繰り返し取得されます。そのため、ルートで Idempotent Consumer EIP を使用して重複を除外する必要があります。リンク S3ConstantsBUCKET_NAME とリンク S3ConstantsKEY ヘッダー、またはリンク S3ConstantsKEY ヘッダーのみを使用してフィルタリングできます。	true	boolean
fileName (consumer)	指定のファイル名を持つバケットからオブジェクトを取得します。		String
includeBody (consumer)	true の場合、エクスチェンジ本文はファイルの内容へのストリームに設定されます。false の場合、ヘッダーには S3 オブジェクトのメタデータが設定されますが、ボディーは null になります。このオプションは、autocloseBody オプションと密接に関係します。includeBody を true に設定し、autocloseBody を false に設定した場合、S3Object ストリームを閉じるのは呼び出し側が判断します。autocloseBody を true に設定すると、S3Object ストリームが自動的に閉じられます。	true	boolean
maxConnections (consumer)	S3 クライアント設定の maxConnections パラメーターを設定します。	60	int
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の数を使用して無制限として無効にします。	10	int
prefix (consumer)	対象のオブジェクトのみを消費するために com.amazonaws.services.s3.model.ListObjectsRequest で使用される接頭辞。		String

名前	説明	デフォルト	タイプ
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
autocloseBody (consumer)	このオプションが true で、includeBody が true の場合、エクスチェンジの完了時に S3Object.close() メソッドが呼び出されます。このオプションは includeBody オプションと密接に関係しています。includeBody を true に設定し、autocloseBody を false に設定した場合、S3Object ストリームを閉じるのは呼び出し側が判断します。autocloseBody を true に設定すると、S3Object ストリームが自動的に閉じられます。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
deleteAfterWrite (producer)	S3 ファイルのアップロード後にファイルオブジェクトを削除します。	false	boolean
multiPartUpload (producer)	true の場合、Camel はマルチパート形式のファイルをアップロードし、パートサイズは partSize のオプションによって決定されます。	false	boolean
operation (producer)	ユーザーがアップロードだけをしたくない場合に行う操作		S3 オペレーション
partSize (producer)	マルチパートのアップロードで使用される partSize を設定します。デフォルトのサイズは 25M です。	26214400	long
serverSideEncryption (producer)	AWS が管理するキーを使用してオブジェクトを暗号化するときに、サーバー側の暗号化アルゴリズムを設定します。たとえば、AES256 を使用します。		String

名前	説明	デフォルト	タイプ
storageClass (producer)	com.amazonaws.services.s3.model.PutObjectRequest リクエストに設定するストレージクラス。		String
awsKMSKeyId (producer)	KMS が有効になっている場合に使用する KMS キーの ID を定義します。		String
useAwsKMS (producer)	KMS を使用する必要があるかどうかを定義します。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accelerateModeEnabled (advanced)	アクセラレートモードの有効化が true か false かを定義する	false	boolean
chunkedEncodingDisabled (advanced)	無効化されたチャンクエンコーディングが true か false かを定義します	false	boolean
dualstackEnabled (advanced)	Dualstack の有効化が true か false かを定義する	false	boolean
forceGlobalBucketAccess Enabled (advanced)	Force Global Bucket Access の有効化が true か false かを定義します	false	boolean
payloadSigningEnabled (advanced)	有効なペイロード署名が true か false かを定義する	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

名前	説明	デフォルト	タイプ
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必須の S3 コンポーネントオプション

Amazon の S3 にアクセスするには、レジストリーに amazonDDBClient を指定するか、accessKey と secretKey を指定する必要があります。

29.3. バッチコンシューマー

このコンポーネントは、Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージの数を知ることができ、たとえば、Aggregator にこの数のメッセージを集約させることができます。

29.4. 使用方法

29.4.1. S3 プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsS3BucketName	**String**	このオブジェクトが保存されるバケット名、または現在の操作に使用されるバケット名
CamelAwsS3BucketDestinationName	**String**	Camel 2.18: 現在の操作に使用されるバケット宛先名。
CamelAwsS3ContentLength	**Long**	このオブジェクトのコンテンツの長さ。
CamelAwsS3ContentType	**String**	このオブジェクトのコンテンツタイプ。
CamelAwsS3ContentControl	**String**	Camel 2.8.2: このオブジェクトのコンテンツコントロール。

ヘッダー	タイプ	説明
CamelAwsS3ContentDisposition	**String**	Camel 2.8.2: このオブジェクトのコンテンツ処理。
CamelAwsS3ContentEncoding	**String**	Camel 2.8.2: このオブジェクトのコンテンツエンコーディング。
CamelAwsS3ContentMD5	**String**	Camel 2.8.2: このオブジェクトの md5 チェックサム。
CamelAwsS3DestinationKey	**String**	Camel 2.18: 現在の操作に使用される宛先キー。
CamelAwsS3Key	**String**	このオブジェクトが格納されるキー、または現在の操作に使用されるキー
CamelAwsS3LastModified	**java.util.Date**	Camel 2.8.2: このオブジェクトの最終変更タイムスタンプ。
CamelAwsS3Operation	**String**	Camel 2.18 : 実行する操作。許可されている値は、copyObject、listBuckets、deleteBucket、downloadLink です。
CamelAwsS3StorageClass	**String**	Camel 2.8.4: このオブジェクトのストレージクラス。

ヘッダー	タイプ	説明
CamelAwsS3CannedAcl	**String**	Camel 2.11.0: オブジェクトに適用される既定の ACL。許可されている値については、**com.amazonaws.services.s3.model.CannedAccessControlList** を参照してください。
CamelAwsS3Acl	**com.amazonaws.services.s3.model.AccessControlList**	Camel 2.11.0: 適切に構築された Amazon S3 Access Control List オブジェクト。詳細については、**com.amazonaws.services.s3.model.AccessControlList** を参照してください。
CamelAwsS3Headers	**Map<String,String>**	Camel 2.15.0 : カスタム objectMetadata ヘッダーの取得または設定をサポート。
CamelAwsS3ServerSideEncryption	String	Camel 2.16: AWS が管理するキーを使用してオブジェクトを暗号化するときに、サーバー側の暗号化アルゴリズムを設定します。たとえば、AES256 を使用します。
CamelAwsS3VersionId	**String**	現在の操作から格納または返されるオブジェクトのバージョン ID

29.4.2. S3 プロデューサーによって設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsS3ETag	**String**	新しくアップロードされたオブジェクトの ETag 値。
CamelAwsS3VersionId	**String**	新しくアップロードされたオブジェクトの **オプション** のバージョン ID。

ヘッダー	タイプ	説明
CamelAwsS3DownloadLinkExpiration	**String**	URL ダウンロードリンクの有効期限 (ミリ秒)。リンクは CamelAwsS3DownloadLink レスポンスヘッダーに保存されます。

29.4.3. S3 コンシューマーによって設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsS3Key	**String**	このオブジェクトが格納されるキー。
CamelAwsS3BucketName	**String**	このオブジェクトが含まれるバケットの名前。
CamelAwsS3ETag	**String**	RFC 1864 に従って、関連付けられたオブジェクトの 16 進数でエンコードされた 128 ビット MD5 ダイジェスト。このデータは、呼び出し元によって受信されたデータが Amazon S3 によって送信されたデータと同じであることを確認するための整合性チェックとして使用されます。
CamelAwsS3LastModified	**日付**	Last-Modified ヘッダーの値。Amazon S3 が関連付けられたオブジェクトへの変更を最後に記録した日時を示します。
CamelAwsS3VersionId	**String**	関連する Amazon S3 オブジェクトのバージョン ID (利用可能な場合)。バージョン ID は、オブジェクトのバージョニングが有効になっている Amazon S3 バケットにオブジェクトがアップロードされた場合にのみ、オブジェクトに割り当てられます。
CamelAwsS3ContentType	**String**	関連付けられたオブジェクトに格納されているコンテンツのタイプを示す Content-Type HTTP ヘッダー。このヘッダーの値は、標準の MIME タイプです。

ヘッダー	タイプ	説明
CamelAwsS3ContentMD5	**String**	RFC 1864 に従って、関連付けられたオブジェクト (ヘッダーを含まないコンテンツ) の base64 でエンコードされた 128 ビット MD5 ダイジェスト。このデータは、Amazon S3 が受信したデータが発信者が送信したデータと同じであることを確認するためのメッセージ整合性チェックとして使用されます。
CamelAwsS3ContentLength	**Long**	関連付けられたオブジェクトのサイズをバイト単位で示す Content-Length HTTP ヘッダー。
CamelAwsS3ContentEncoding	**String**	オブジェクトに適用されたコンテンツエンコーディングと、Content-Type フィールドによって参照されるメディアタイプを取得するために適用する必要があるデコードメカニズムを指定する、**オプション** の Content-Encoding HTTP ヘッダー。
CamelAwsS3ContentDisposition	**String**	**オプション** の Content-Disposition HTTP ヘッダー。保存するオブジェクトの推奨ファイル名などの表示情報を指定します。
CamelAwsS3ContentControl	**String**	ユーザーが HTTP 要求/応答チェーンに沿ってキャッシュ動作を指定できるようにする、**オプション** の Cache-Control HTTP ヘッダー。
CamelAwsS3ServerSideEncryption	String	**Camel 2.16:** AWS が管理するキーを使用してオブジェクトを暗号化するときに、サーバー側の暗号化アルゴリズムを設定します。

29.4.4. 高度な AmazonS3 設定

Camel アプリケーションがファイアウォールの背後で実行されている場合、または **AmazonS3** インスタンス設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
```

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonS3 client = new AmazonS3Client(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

Camel aws-s3 コンポーネント設定で参照します。

```
from("aws-s3://MyBucket?amazonS3Client=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

29.4.5. S3 コンポーネントで KMS を使用する

AWS インフラストラクチャーを使用して AWS KMS を使用してデータを暗号化/復号化するには、次の例のように 2.21.x で導入されたオプションを使用できます。

```
from("file:tmp/test?fileName=test.txt")
    .setHeader(S3Constants.KEY, constant("testFile"))
    .to("aws-s3://mybucket?amazonS3Client=#client&useAwsKMS=true&awsKMSKeyId=3f0637ad-296a-3dfe-a796-e60654fb128c");
```

このようにして、KMS キー 3f0637ad-296a-3dfe-a796-e60654fb128c を使用してファイル test.txt を暗号化するよう S3 に依頼します。このファイルのダウンロードを要求すると、ダウンロードの直前に復号化が行われます。

29.4.6. s3 コンポーネントで useIAMCredentials を使用する

AWS IAM クレデンシャルを使用するには、Camel アプリケーションを起動する EC2 に、効果的に実行するためにアタッチされた適切なポリシーを含む IAM ロールが関連付けられていることを最初に確認する必要があります。この機能は、リモートインスタンスでのみ true に設定する必要があることに注意してください。さらに明確にするために、IAM は AWS 固有のコンポーネントであるため、静的認証情報をローカルで使用する必要がありますが、AWS 環境は管理が容易になるはずです。これを実装して理解したら、AWS 環境のクエリーパラメーター useIAMCredentials を true に設定できます。これをローカル環境とリモート環境に基づいて効果的にオンとオフを切り替えるには、システム環境変数でこのクエリーパラメーターを有効にすることを検討できます。たとえば、"isRemote" というシステム環境変数が true に設定されている場合、コードで "useIAMCredentials" クエリーパラメーターを "true" に設定できます (これを行うには他にも多くの方法があり、これは単純な例として機能するはずです)。静的認証情報の必要性が完全になくなるわけではありませんが、AWS 環境で IAM 認証情報を使用すると、リモート環境で更新する必要がなくなり、セキュリティーが大幅に向上します (IAM 認証情報は 6 時間ごとに自動的に更新され、ポリシーが更新されると更新されます)。更新しました)。これは AWS が認証情報を管理するために推奨する方法であるため、できるだけ頻繁に使用する必要があります。

29.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
```

```
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.8 以降) に置き換える必要があります。

29.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第30章 AWS SIMPLEDB コンポーネント

Camel バージョン 2.9 以降で利用可能

sdb コンポーネントは、Amazon の SDB サービスとの間でのデータの保存と取得をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon S3 を使用するためにサインアップしている必要がある。詳細については、Amazon SDB を参照してください。

30.1. URI 形式

```
aws-sdb://domainName[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

30.2. URI オプション

AWS SimpleDB コンポーネントにはオプションがありません。

AWS SimpleDB エンドポイントは、URI 構文を使用して設定されます。

```
aws-sdb:domainName
```

パスおよびクエリーパラメーターを使用します。

30.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
domainName	**必須** 現在使用しているドメインの名前。		String

30.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
accessKey (producer)	Amazon AWS Access Key		String
amazonSDBClient (producer)	AmazonSimpleDB をクライアントとして使用するには		AmazonSimpleDB
consistentRead (producer)	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。	false	boolean

名前	説明	デフォルト	タイプ
maxNumberOfDomains (producer)	返されるドメイン名の最大数。範囲は 1 から 100 です。		Integer
operation (producer)	実行する操作	PutAttributes	SdbOperations
proxyHost (producer)	DDB クライアントをインスタンス化する際にプロキシーホストを定義します。		String
proxyPort (producer)	SQS クライアントをインスタンス化するときにプロキシーポートを定義します。		Integer
region (producer)	DDB クライアントが機能する必要があるリージョン。		String
secretKey (producer)	Amazon AWS Secret Key		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必須の SDB コンポーネントオプション

Amazon の SDB にアクセスするには、レジストリーに amazonSDBClient を指定するか、accessKey と secretKey を指定する必要があります。

30.3. 使用方法

30.3.1. SDB プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSdbAttributes	**Collection<Attribute>**	作用する属性のリスト。
CamelAwsSdbAttributeNames	**Collection<String>**	取得する属性の名前。

ヘッダー	タイプ	説明
CamelAwsSdbConsistentRead	**Boolean**	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。
CamelAwsSdbDeletableItems	**Collection<DeletableItem>**	バッチで削除操作を実行するアイテムのリスト。
CamelAwsSdbDomainName	**String**	現在使用しているドメインの名前。
CamelAwsSdbItemName	**String**	このアイテムの一意のキー
CamelAwsSdbMaxNumberOfDomains	**Integer**	返されるドメイン名の最大数。範囲は 1* から 100 です。
CamelAwsSdbNextToken	**String**	ドメイン/項目名の次のリストの開始位置を指定する文字列。
CamelAwsSdbOperation	**String**	URI オプションからの操作をオーバーライドします。

ヘッダー	タイプ	説明
CamelAwsSdbReplaceableAttributes	**Collection<ReplaceableAttribute>**	アイテムに入れる属性のリスト。
CamelAwsSdbReplaceableItems	**Collection<ReplaceableItem>**	ドメインに入れるアイテムのリスト。
CamelAwsSdbSelectExpression	**String**	ドメインのクエリーに使用される式。
CamelAwsSdbUpdateCondition	**UpdateCondition**	指定された場合、指定された属性が更新/削除されるかどうかを決定する更新条件。

30.3.2. DomainMetadata 操作中に設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSdbTimestamp	**Integer**	メタデータが計算された日時 (エポック (UNIX) 秒単位)。
CamelAwsSdbItemCount	**Integer**	ドメイン内のすべてのアイテムの数。

ヘッダー	タイプ	説明
CamelAwsSdbAttributeNameCount	**Integer**	ドメイン内の一意の属性名の数。
CamelAwsSdbAttributeValueCount	**Integer**	ドメイン内のすべての属性の名前と値のペアの数。
CamelAwsSdbAttributeNameSize	**Long**	ドメイン内のすべての一意の属性名の合計サイズ (バイト単位)。
CamelAwsSdbAttributeValueSize	**Long**	ドメイン内のすべての属性値の合計サイズ (バイト単位)。
CamelAwsSdbItemNameSize	**Long**	ドメイン内のすべての項目名の合計サイズ (バイト単位)。

30.3.3. GetAttributes 操作中に設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSdbAttributes	**List<Attribute>**	操作によって返される属性の一覧。

30.3.4. ListDomains 操作中に設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSdbDomainNames	**List<String>**	式に一致するドメイン名のリスト。
CamelAwsSdbNextToken	**String**	指定された MaxNumberOfDomains よりも多くのドメインがまだ利用可能であることを示す不透明なトークン。

30.3.5. Select 操作中に設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSdbItems	**List<Item>**	select 式に一致するアイテムのリスト。
CamelAwsSdbNextToken	**String**	MaxNumberOfItems を超えるアイテムが一致したか、応答サイズが 1 メガバイトを超えたか、または実行時間が 5 秒を超えたかを示す不透明なトークン。

30.3.6. 高度な AmazonSimpleDB 設定

AmazonSimpleDB インスタンス設定をさらに制御する必要がある場合は、独自のインスタンスを作成し、URI から参照できます。

```
from("direct:start")
.to("aws-sdb://domainName?amazonSDBClient=#client");
```

#client は、レジストリー内の **AmazonSimpleDB** を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonSimpleDB client = new AmazonSimpleDBClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

30.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.8.4 以降) に置き換える必要があります。

30.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第31章 AWS シンプル電子メールサービスコンポーネント

Camel バージョン 2.9 以降で利用可能

ses コンポーネントは、Amazon の SES サービスを使用したメールの送信をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon SES を使用するためにサインアップしている必要がある。詳細については、Amazon SES を参照してください。

31.1. URI 形式

```
aws-ses://from[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

31.2. URI オプション

AWS Simple Email Service コンポーネントは、以下に示す 5個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS SES のデフォルト設定		SesConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	SES クライアントが機能する必要があるリージョン。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Email Service エンドポイントは、URI 構文を使用して設定されます。

```
aws-ses:from
```

パスおよびクエリーパラメーターを使用します。

31.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
from	**必須** 送信者の電子メールアドレス。		String

31.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
amazonSESClient (producer)	AmazonSimpleEmailService をクライアントとして使用するには		AmazonSimpleEmail Service
proxyHost (producer)	SNS クライアントをインスタンス化するときにプロキシーホストを定義します。		String
proxyPort (producer)	SES クライアントをインスタンス化するときにプロキシーポートを定義します。		Integer
region (producer)	SES クライアントが機能する必要があるリージョン。		String
replyToAddresses (producer)	メッセージの返信先電子メールアドレスのリスト。CamelAwsSesReplyToAddresses ヘッダーを使用してオーバーライドします。		List
returnPath (producer)	バウンス通知の転送先の電子メールアドレス。CamelAwsSesReturnPath ヘッダーを使用してオーバーライドします。		String
subject (producer)	メッセージヘッダー 'CamelAwsSesSubject' が存在しない場合に使用されるサブジェクト。		String
to (producer)	宛先メールアドレスのリスト。CamelAwsSesTo ヘッダーでオーバーライドできます。		List
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必須の SES コンポーネントオプション

Amazon の SES にアクセスするには、レジストリーに amazonSESClient を指定するか、accessKey と secretKey を指定する必要があります。

31.3. 使用方法

31.3.1. SES プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSesFrom	**String**	送信者の電子メールアドレス。
CamelAwsSesTo	**List<String>**	このメールの宛先。
CamelAwsSesSubject	**String**	メッセージの件名。
CamelAwsSesReplyToAddresses	**List<String>**	メッセージの返信先電子メールアドレス。
CamelAwsSesReturnPath	**String**	バウンス通知の転送先の電子メールアドレス。
CamelAwsSesHtmlEmail	**Boolean**	**Camel 2.12.3 以降** メールの内容が HTML の場合に表示するフラグ。

31.3.2. SES プロデューサによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSesMessageId	**String**	Amazon SES メッセージ ID。

31.3.3. 高度な AmazonSimpleEmailService 設定

AmazonSimpleEmailService インスタンスの設定をさらに制御する必要がある場合は、独自のインスタンスを作成して URI から参照できます。

```
from("direct:start")
.to("aws-ses://example@example.com?amazonSESClient=#client");
```

#client は、 レジストリー内の **AmazonSimpleEmailService** を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);
AmazonSimpleEmailService client = new AmazonSimpleEmailServiceClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

31.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.8.4 以降) に置き換える必要があります。

31.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド
- AWS コンポーネント

第32章 AWS シンプル通知システムコンポーネント

Camel バージョン 2.8 以降で利用可能

SNS コンポーネントを使用すると、メッセージを Amazon Simple Notification Topic に送信できます。Amazon API の実装は AWS SDK によって提供されます。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon Kinesis を使用するためにサインアップしている必要がある。詳細については、Amazon SNS を参照してください。

32.1. URI 形式

```
aws-sns://topicNameOrArn[?options]
```

トピックがまだ存在しない場合は作成されます。
URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

32.2. URI オプション

AWS Simple Notification System コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS SNS のデフォルト設定		SnsConfiguration
accessKey (producer)	Amazon AWS Access Key		String
secretKey (producer)	Amazon AWS Secret Key		String
region (producer)	SNS クライアントが機能する必要があるリージョン		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Notification System エンドポイントは、URI 構文を使用して設定されます。

```
aws-sns:topicNameOrArn
```

パスおよびクエリーパラメーターを使用します。

32.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
topicNameOrArn	**必須** トピック名または ARN		String

32.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
amazonSNSClient (producer)	AmazonSNS をクライアントとして使用します。		AmazonSNS
headerFilterStrategy (producer)	カスタムの HeaderFilterStrategy を使用して、ヘッダーから Camel または Camel からヘッダーにマッピングします。		HeaderFilterStrategy
messageStructure (producer)	json などの使用するメッセージ構造。		String
policy (producer)	このキューのポリシー		String
proxyHost (producer)	SNS クライアントをインスタンス化するときにプロキシーホストを定義します。		String
proxyPort (producer)	SNS クライアントをインスタンス化するときにプロキシーポートを定義します。		Integer
region (producer)	SNS クライアントが機能する必要があるリージョン		String
subject (producer)	メッセージヘッダー 'CamelAwsSnsSubject' が存在しない場合に使用されるサブジェクト。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必要な SNS コンポーネントオプション

Amazon の SNS にアクセスするには、レジストリーに amazonDDBClient を指定するか、accessKey と secretKey を指定する必要があります。

32.3. 使用方法

32.3.1. SNS プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSnsSubject	**String**	Amazon SNS メッセージの件名。設定されていない場合は、**SnsConfiguration** の件名が使用されます。

32.3.2. SNS プロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSnsMessageId	**String**	Amazon SNS メッセージ ID。

32.3.3. 高度な AmazonSNS 設定

AmazonSNS インスタンス設定をさらに制御する必要がある場合は、独自のインスタンスを作成し、URI から参照できます。

```
from("direct:start")
.to("aws-sns://MyTopic?amazonSNSClient=#client");
```

#client は、レジストリー内の **AmazonSNS** を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);
AmazonSNS client = new AmazonSNSClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

32.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.8 以降) に置き換える必要があります。

32.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第33章 AWS シンプルキューサービスコンポーネント

Camel バージョン 2.6 以降で利用可能

sqs コンポーネントは、Amazon の SQS サービス へのメッセージの送受信をサポートしています。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon SQS を使用するためにサインアップしている必要がある。詳細については、Amazon SQS を参照してください。

33.1. URI 形式

```
aws-sqs://queueNameOrArn[?options]
```

キューがまだ存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

33.2. URI オプション

AWS Simple Queue Service コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS SQS のデフォルト設定。		SqsConfiguration
accessKey (common)	Amazon AWS Access Key		String
secretKey (common)	Amazon AWS Secret Key		String
region (common)	サービス URL を作成するために queueOwnerAWSAccountId で使用できるキューリージョンを指定します。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Queue Service エンドポイントは、URI 構文を使用して設定されます。

```
aws-sqs:queueNameOrArn
```

パスおよびクエリーパラメーターを使用します。

33.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
queueNameOrArn	**必須** キュー名または ARN		文字列

33.2.2. クエリーパラメーター(46 個のパラメーター):

名前	説明	デフォルト	タイプ
amazonAWSHost (common)	Amazon AWS クラウドのホスト名。	amazonaws.com	String
amazonSQSClient (common)	AmazonSQS をクライアントとして使用します。		AmazonSQS
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、ヘッダーから Camel または Camel からヘッダーにマッピングします。		HeaderFilterStrategy
queueOwnerAWSAccountId (common)	異なるアカウント所有者でキューを接続する必要がある場合は、キュー所有者の aws アカウント ID を指定します。		String
region (common)	サービス URL を作成するために queueOwnerAWSAccountId で使用できるキューリージョンを指定します。		文字列
attributeNames (consumer)	消費時に受け取る属性名のリスト。複数の値はコンマで区切ることができます。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	複数のスレッドを使用して sqs キューをポーリングし、スループットを向上させることができます。	1	int
defaultVisibilityTimeout (consumer)	デフォルトの表示タイムアウト (秒単位)。		Integer

名前	説明	デフォルト	タイプ
deleteAfterRead (consumer)	メッセージが読まれた後、SQS からメッセージを削除します。	true	boolean
deleteIfFiltered (consumer)	エクスチェンジがフィルターを通過できなかった場合に、DeleteMessage を SQS キューに送信するかどうか。false でエクスチェンジがルートのアップストリームの Camel フィルターを通過しない場合は、DeleteMessage を送信しないでください。	true	boolean
extendMessageVisibility (consumer)	有効にすると、スケジュールされたバックグラウンドタスクにより、SQS でのメッセージの可視性が拡張され続けます。これは、メッセージの処理に時間がかかる場合に必要です。true に設定した場合は、defaultVisibilityTimeout を設定する必要があります。詳細については、Amazon ドキュメントを参照してください。	false	boolean
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の数を使用して無制限として無効にします。		int
messageAttributeNames (consumer)	消費時に受け取るメッセージ属性名のリスト。複数の値はコンマで区切ることができます。		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
visibilityTimeout (consumer)	受信したメッセージが、com.amazonaws.services.sqs.model.SetQueueAttributesRequest で設定する ReceiveMessage リクエストによって取得された後、後続の取得リクエストから非表示になる期間 (秒単位)。これは、defaultVisibilityTimeout とは異なる場合にのみ意味があります。キューの可視性タイムアウト属性を永続的に変更します。		Integer
waitTimeSeconds (consumer)	メッセージがキューに入れられて応答に含まれるまで、ReceiveMessage アクション呼び出しが待機する時間 (0 から 20) です。		Integer

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
delaySeconds (producer)	数秒間メッセージの送信を遅延します。		Integer
messageDeduplicationId Strategy (producer)	FIFO キューの場合のみ。メッセージに messageDeduplicationId を設定するストラテジー。useExchangeId または useContentBasedDeduplication のいずれかをオプションとして使用できます。useContentBasedDeduplication オプションでは、メッセージに messageDeduplicationId が設定されません。	useExchangeId	MessageDeduplicationId Strategy
messageGroupId Strategy (producer)	FIFO キューの場合のみ。メッセージに messageGroupId を設定するストラテジー。useConstant、useExchangeId、usePropertyValue のいずれかをオプションとして使用できます。usePropertyValue オプションでは、CamelAwsMessageGroupId プロパティーの値が使用されます。		MessageGroupIdStrategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

名前	説明	デフォルト	タイプ
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

名前	説明	デフォルト	タイプ
proxyHost (proxy)	SQS クライアントをインスタンス化するときにプロキシーホストを定義します。		String
proxyPort (proxy)	SQS クライアントをインスタンス化するときにプロキシーポートを定義します。		Integer
maximumMessageSize (キュー)	このキューの SQS メッセージに含めることができる maximumMessageSize (バイト単位)。		Integer
messageRetentionPeriod (queue)	このキューの SQS によってメッセージが保持される messageRetentionPeriod (秒単位)。		Integer
ポリシー (キュー)	このキューのポリシー		String
receiveMessageWaitTime Seconds (キュー)	要求で WaitTimeSeconds を指定しない場合は、キュー属性 ReceiveMessageWaitTimeSeconds を使用して待機時間を決定します。		Integer
redrivePolicy (キュー)	DeadLetter キューにメッセージを送信するポリシーを指定します。Amazon ドキュメントで詳細を参照してください。		String
accessKey (security)	Amazon AWS Access Key		String
secretKey (security)	Amazon AWS Secret Key		String

必須の SQS コンポーネントオプション

Amazon の SQS にアクセスするには、レジストリーに amazonDDBClient を指定するか、accessKey と secretKey を指定する必要があります。

33.3. バッチコンシューマー

このコンポーネントは、Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージの数を知ることができ、たとえば、Aggregator にこの数のメッセージを集約させることができます。

33.4. 使用方法

33.4.1. SQS プロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSqsMD5OfBody	**String**	Amazon SQS メッセージの MD5 チェックサム。
CamelAwsSqsMessageId	**String**	Amazon SQS メッセージ ID。
CamelAwsSqsDelaySeconds	**Integer**	**Camel 2.11** 以降、Amazon SQS メッセージが他のユーザーに表示される遅延秒数。

33.4.2. SQS コンシューマーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelAwsSqsMD5OfBody	**String**	Amazon SQS メッセージの MD5 チェックサム。
CamelAwsSqsMessageId	**String**	Amazon SQS メッセージ ID。
CamelAwsSqsReceiptHandle	**String**	Amazon SQS メッセージ受信ハンドル。
CamelAwsSqsAttributes	**Map<String, String>**	Amazon SQS メッセージ属性。

33.4.3. 高度な AmazonSQS 設定

Camel アプリケーションがファイアウォールの背後で実行されている場合、または AmazonSQS インスタンス設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");

ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonSQS client = new AmazonSQSClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

Camel aws-sqs コンポーネント設定で参照します。

```
from("aws-sqs://MyQueue?amazonSQSClient=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

33.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.6 以降) に置き換える必要があります。

33.6. JMS スタイルのセレクター

SQS ではセレクターを使用できませんが、キャメルフィルター EIP を使用して適切な **visibilityTimeout** を設定することで効果的にこれを実現できます。SQS がメッセージをディスパッチするとき、DeleteMessage が受信されない限り、可視性タイムアウトまで待機してから、別のコンシューマーにメッセージをディスパッチしようとします。デフォルトでは、ルートが失敗に終わっていない限り、Camel は常にルートの最後に DeleteMessage を送信します。適切なフィルタリングを実現し、ルートが正常に完了した場合でも DeleteMessage を送信しないようにするには、Filter を使用します。

```
from("aws-sqs://MyQueue?
amazonSQSClient=#client&defaultVisibilityTimeout=5000&deleteIfFiltered=false")
.filter("${header.login} == true")
.to("mock:result");
```

上記のコードでは、交換に適切なヘッダーがない場合、フィルターを通過せず、SQS キューからも削除されません。5000 ミリ秒後、メッセージは他のコンシューマーに表示されます。

33.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AWS コンポーネント

第34章 AWS SIMPLE WORKFLOW コンポーネント

Camel バージョン 2.13 以降で利用可能

Simple Workflow コンポーネントは、Amazon の Simple Workflow サービスからのワークフローの管理をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントを持っていて、Amazon Simple Workflow を使用するためにサインアップしている必要がある。詳細については、Amazon Simple Workflow を参照してください。

34.1. URI 形式

```
aws-swf://<workflow|activity>[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

34.2. URI オプション

AWS Simple Workflow コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	AWS SWF のデフォルト設定		SWFConfiguration
accessKey (common)	Amazon AWS Access Key。		String
secretKey (common)	Amazon AWS Secret Key。		String
region (common)	Amazon AWS リージョン。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Workflow エンドポイントは、URI 構文を使用して設定されます。

```
aws-swf:type
```

パスおよびクエリーパラメーターを使用します。

34.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
type	**必要な** アクティビティーまたはワークフロー		String

34.2.2. クエリーパラメーター (30 パラメーター)

名前	説明	デフォルト	タイプ
amazonSWClient (common)	指定された AmazonSimpleWorkflowClient をクライアントとして使用する場合		AmazonSimpleWorkflow Client
dataConverter (common)	データのシリアル化/逆シリアル化に使用する com.amazonaws.services.simpleworkflow.flow.DataConverter のインスタンス。		DataConverter
domainName (common)	使用するワークフロードメイン。		String
eventName (common)	使用するワークフローまたはアクティビティーイベント名。		String
region (common)	Amazon AWS リージョン。		String
version (common)	使用するワークフローまたはアクティビティーイベントのバージョン。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
clientConfiguration Parameters (advanced)	マップのキー/値を使用して ClientConfiguration を設定する場合。		Map
startWorkflowOptions Parameters (advanced)	マップのキー/値を使用して StartWorkflowOptions を設定する場合。		Map
sWClientParameters (advanced)	マップのキー/値を使用して AmazonSimpleWorkflowClient を設定する場合。		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
activityList (activity)	アクティビティーを使用するリスト名。		String
activitySchedulingOptions (activity)	アクティビティーのスケジュールオプション		アクティビティーのスケジュールオプション
activityThreadPoolSize (activity)	アクティビティーの作業プール内のスレッドの最大数。	100	int
activityTypeExecution Options (activity)	アクティビティー実行オプション		ActivityTypeExecution Options
activityTypeRegistration Options (activity)	アクティビティー登録オプション		ActivityTypeRegistrationOptions
childPolicy (workflow)	ワークフローを終了するときに子ワークフローで使用するポリシー。		String
executionStartToClose Timeout (workflow)	実行開始から終了までのタイムアウトを設定します。	3600	String
operation (workflow)	ワークフロー操作	開始	String
signalName (workflow)	ワークフローに送信するシグナルの名前。		String

名前	説明	デフォルト	タイプ
stateResultType (workflow)	ワークフロー状態が照会されたときの結果のタイプ。		String
taskStartToCloseTimeout (workflow)	タスクの開始から終了までのタイムアウトを設定します。	600	String
terminationDetails (workflow)	ワークフローを終了するための詳細。		String
terminationReason (workflow)	ワークフローを終了する理由。		String
workflowList (workflow)	ワークフローを使用するリスト名。		String
workflowTypeRegistration Options (workflow)	ワークフロー登録オプション		WorkflowTypeRegistrationOptions
accessKey (security)	Amazon AWS Access Key。		String
secretKey (security)	Amazon AWS Secret Key。		String

必要な SWF コンポーネントオプション

Amazon の Simple Workflow Service にアクセスするには、レジストリーに amazonSWClient を指定するか、accessKey と secretKey を指定する必要があります。

34.3. 使用方法

34.3.1. SWF ワークフロープロデューサーによって評価されるメッセージヘッダー

ワークフロープロデューサーを使用すると、ワークフローと対話できます。新しいワークフローの実行を開始したり、その状態をクエリーしたり、実行中のワークフローにシグナルを送信したり、ワークフローを終了してキャンセルしたりできます。

ヘッダー	タイプ	説明
CamelSWFOperation	**String**	ワークフローで実行する操作。サポートされている操作は次のとおりです。SIGNAL、CANCEL、TERMINATE、GET_STATE、START、DESCRIBE、GET_HISTORY。
CamelSWFWorkflowId	**String**	使用するワークフロー ID。
CamelAwsDdbKeyCamelSWFRunId	**String**	使用するワークフロー実行 ID。
CamelSWFStateResultType	**String**	ワークフロー状態が照会されたときの結果のタイプ。
CamelSWFEventName	**String**	使用するワークフローまたはアクティビティーイベント名。
CamelSWFVersion	**String**	使用するワークフローまたはアクティビティーイベントのバージョン。
CamelSWFReason	**String**	ワークフローを終了する理由。
CamelSWFDetails	**String**	ワークフローを終了するための詳細。
CamelSWFChildPolicy	**String**	ワークフローを終了するときに子ワークフローで使用するポリシー。

34.3.2. SWF ワークフロープロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelSWFWorkflowId	**String**	使用された、または新しく生成されたワークフロー ID。
CamelAwsDdbKey CamelSWFRunId	**String**	使用または生成されたワークフロー実行 ID。

34.3.3. SWF ワークフローコンシューマーによって設定されるメッセージヘッダー

ワークフローコンシューマーは、ワークフローロジックを表します。開始されると、ワークフロー決定タスクのポーリングが開始され、それらが処理されます。決定タスクの処理に加えて、ワークフローコンシューマールートはシグナル (ワークフロープロデューサーから送信) または状態クエリーも受信します。ワークフローコンシューマーの主な目的は、アクティビティープロデューサーを使用して実行するアクティビティータスクをスケジュールすることです。実際、アクティビティータスクは、ワークフローコンシューマーによって開始されたスレッドからのみスケジュールできます。

ヘッダー	タイプ	説明
CamelSWFAction	**String**	現在のイベントのタイプを示します: CamelSWFActionExecute、CamelSWFSignalReceivedAction、または CamelSWFGetStateAction。
CamelSWFWorkflowReplaying	**boolean**	現在の決定タスクがリプレイかどうかを示します。
CamelSWFWorkflowStartTime	**long**	この決定タスクの開始イベントの時刻。

34.3.4. SWF アクティビティープロデューサーによって設定されるメッセージヘッダー

アクティビティープロデューサーを使用すると、アクティビティータスクをスケジュールできます。アクティビティープロデューサーは、ワークフローコンシューマーによって開始されたスレッドからのみ使用できます。つまり、ワークフローコンシューマーによって開始された同期エクスチェンジを処理できます。

ヘッダー	タイプ	説明
CamelSWFEventName	**String**	スケジュールするアクティビティー名。
CamelSWFVersion	**String**	スケジュールするアクティビティーバージョン。

34.3.5. SWF アクティビティーコンシューマーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelSWFTaskToken	**String**	手動で完了したタスクの完了を報告するために必要なタスクトークン。

34.3.6. 高度な amazonSWClient 設定

AmazonSimpleWorkflowClient インスタンス設定をさらに制御する必要がある場合は、独自のインスタンスを作成し、URI から参照できます。

#client は、レジストリー内の AmazonSimpleWorkflowClient を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合:

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey", "mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonSimpleWorkflowClient client = new AmazonSimpleWorkflowClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

34.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
```

```
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は、Camel の実際のバージョン (2.13 以降) に置き換える必要があります。

34.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

AWS コンポーネント

第35章 AWS XRAY コンポーネント

Camel 2.21 以降で利用可能

camel-aws-xray コンポーネントは、AWS XRay を使用して受信および送信 Camel メッセージをトレースおよびタイミングするために使用されます。

イベント (サブセグメント) は、Camel との間で送受信される着信および発信メッセージに対してキャプチャーされます。

35.1. 依存関係

AWS XRay サポートを Camel に含めるには、Camel 関連の AWS XRay 関連クラスを含むアーカイブをプロジェクトに追加する必要があります。それに加えて、AWS XRay ライブラリーも利用できる必要があります。

AWS XRay と Camel の両方を含めるには、依存関係で次の Maven インポートを使用します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-bom</artifactId>
      <version>1.3.1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-aws-xray</artifactId>
    </dependency>

    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-core</artifactId>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-aws-sdk</artifactId>
    </dependency>
<dependencies>
```

35.2. 設定

AWS XRay トレーサーの設定プロパティーは次のとおりです。

オプション	デフォルト	説明
addExcludePatterns		パターンに一致する Camel メッセージのトレースを無効にする除外パターンを設定します。コンテンツは Set<String> で、キーは routeId に一致するパターンです。このパターンは Intercept のルールを使用します。
setTracingStrategy	NoopTracingStrategy	**BeanDefinition** や **ProcessDefinition** などの呼び出されたプロセッサー定義を追跡するために、カスタム Camel **InterceptStrategy** を提供できるようにします。**TraceAnnotatedTracingStrategy は、** クラスレベルで **@XRayTrace** アノテーションを含む **.bean (...)** または **.process (...)** を介して呼び出されたすべてのクラスを追跡します。

現在、Camel アプリケーションの分散トレースを提供するように AWS XRay トレーサーを設定できる方法は 1 つだけです。

35.2.1. Explicit

AWS XRay Tracer に関連付けられた特定の依存関係とともに、**camel-aws-xray** コンポーネントを POM に含めます。

AWS XRay サポートを明示的に設定するには、**XRayTracer** をインスタンス化し、camel コンテキストを初期化します。オプションで **Tracer** を指定することも、代わりに **Registry** または **ServiceLoader** を使用して暗黙的に検出することもできます。

```
XRayTracer xrayTracer = new XRayTracer();
// By default it uses a NoopTracingStrategy, but you can override it with a specific InterceptStrategy implementation.
xrayTracer.setTracingStrategy(...);
// And then initialize the context
xrayTracer.init(camelContext);
```

XML で XRayTracer を使用するには、AWS XRay トレーサー Bean を定義するだけです。Camel はそれを自動的に検出して使用します。

```
<bean id="tracingStrategy" class="..."/>
<bean id="aws-xray-tracer" class="org.apache.camel.component.aws.xray.XRayTracer" />
  <property name="tracer" ref="tracingStrategy"/>
</bean>
```

デフォルトの **NoopTracingStrategy** の場合、エクスチェンジの作成と削除のみが追跡されますが、特定の Bean または EIP パターンの呼び出しは追跡されません。

35.2.2. 包括的なルート実行の追跡

複数のルート間でのエクスチェンジの実行を追跡するために、エクスチェンジの作成時に一意のトレース ID が生成され、対応する値がまだ使用できない場合はヘッダーに格納されます。このトレース ID は、処理された交換の一貫したビューを維持するために、新しいエクスチェンジにコピーされます。

AWS XRay トレースはスレッドローカルベースで機能するため、現在のサブ/セグメントを新しいスレッドにコピーし、AWS XRay のドキュメント で説明されているように設定する必要があります。した

がって、Camel AWS XRay コンポーネントは、渡された AWS XRay **Entity** を新しいスレッドに設定するためにコンポーネントが使用する追加のヘッダーフィールドを提供し、実行されたどのルートとも無関係に見える新しいセグメントを公開するのではなく、ルートへの追跡データを保持します。

コンポーネントは、エクスチェンジのヘッダーにある次の定数を使用します。

ヘッダー	説明
Camel-AWS-XRay-Trace-ID	呼び出されたルートの包括的なビューを提供する AWS XRay **TraceID** オブジェクトへの参照が含まれています
Camel-AWS-XRay-Trace-Entity	新しいスレッドにコピーされる実際の AWS XRay **Segment** または **Subsegment** への参照が含まれています。このヘッダーは、新しいスレッドが生成され、実行されたタスクが新しい無関係なセグメントを作成する代わりに、実行されたルートの一部として公開される場合に設定する必要があります。

AWS XRay **Entity** (つまり、**Segment** と **Subsegment**) はシリアル化できないため、他の JVM プロセスに渡すべきではないことに注意してください。

35.3. 例

このプロジェクトに付随するテスト内で、AWS XRay トレースを設定する方法を示す例を見つけることができます。

第36章 WINDOWS AZURE サービスの CAMEL コンポーネント

Windows Azure サービス の Camel コンポーネントは、Camel から Azure サービスへの接続を提供します。

Azure Service	Camel コンポーネント	Camel バージョン	コンポーネントの説明
ストレージ BLOB サービス	Azure-Blob	2.9.0	BLOB の保存と取得をサポート
ストレージキューサービス	Azure-Queue	2.9.0	キューでのメッセージの保存と取得をサポート

第37章 AZURE STORAGE BLOB SERVICE コンポーネント

Camel バージョン 2.19 以降で利用可能

Azure Blob コンポーネントは、Azure Storage Blob サービスとの間での BLOB の保存と取得をサポートしています。

前提条件

有効な Windows Azure ストレージアカウントが必要です。詳細については、Azure ドキュメントポータル を参照してください。

37.1. URI 形式

```
azure-blob://accountName/containerName[/blobName][?options]
```

ほとんどの場合、blobName が必要であり、Blob がまだ存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、**camelazure** ストレージアカウントの **container1** にあるパブリックブロック blob **blockBlob** から BLOB コンテンツをダウンロードするには、次のスニペットを使用します。

```
from("azure-blob:/camelazure/container1/blockBlob").
to("file://blobdirectory");
```

37.2. URI オプション

Azure Storage Blob Service コンポーネントにはオプションがありません。

Azure Storage Blob Service エンドポイントは、URI 構文を使用して設定されます。

```
azure-blob:containerOrBlobUri
```

パスおよびクエリーパラメーターを使用します。

37.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
containerOrBlobUri	**必須** コンテナーまたは BLOB コンパクト Uri		String

37.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
azureBlobClient (common)	blob サービスクライアント		CloudBlob
blobOffset (common)	アップロードまたはダウンロード操作の BLOB オフセットを設定します。デフォルトは 0 です。	0	Long
blobType (common)	blob タイプを設定します。デフォルトは blockblob です	blockblob	BlobType
closeStreamAfterRead (common)	読み取り後にストリームを閉じるか、開いたままにします。デフォルトは true です。	true	boolean
credentials (common)	ストレージ認証情報を設定します。ほとんどの場合に必要です		StorageCredentials
dataLength (common)	ダウンロード操作またはページ blob アップロード操作のデータ長を設定する		Long
fileDir (common)	ダウンロードした blob が保存されるファイルディレクトリーを設定します		String
publicForRead (common)	ストレージリソースは、コンテンツを読み取るためにパブリックにすることができます。このプロパティーが有効になっている場合、認証情報を設定する必要はありません	false	boolean
streamReadSize (common)	blob コンテンツを読み取るときの最小読み取りサイズをバイト単位で設定します		int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
blobMetadata (producer)	blob メタデータを設定する		Map
blobPrefix (producer)	BLOB の一覧表示に使用できる接頭辞を設定します		String
closeStreamAfterWrite (producer)	書き込み後にストリームを閉じるか、開いたままにします。デフォルトは true です。	true	boolean
operation (producer)	プロデューサーへの Blob サービス操作のヒント	listBlobs	BlobServiceOperations
streamWriteSize (producer)	ブロックおよびページブロックを書き込むためのバッファーのサイズを設定します		int
useFlatListing (producer)	フラットまたは階層的な blob リストを使用するかどうかを指定します	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必須の Azure Storage Blob Service コンポーネントオプション

プライベート blob にアクセスする必要がある場合は、containerOrBlob 名と認証情報を提供する必要があります。

37.3. 使用方法

37.3.1. Azure Storage Blob Service プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明

37.3.2. Azure Storage Blob Service プロデューサーによって設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelFileName	**String**	ダウンロードした blob コンテンツのファイル名。

37.3.3. Azure Storage Blob Service プロデューサーコンシューマーによって設定されたメッセージヘッダー

ヘッダー	タイプ	説明
CamelFileName	**String**	ダウンロードした blob コンテンツのファイル名。

37.3.4. Azure Blob Service の操作

すべてのブロックタイプに共通の操作

操作	説明
getBlob	ブロブの内容を取得します。この操作の出力をブロブ範囲に制限できます。
deleteBlob	blob を削除します。
listBlobs	blob を一覧表示します。

ブロック blob 操作

操作	説明
updateBlockBlob	新しいブロック blob を作成するか、既存のブロック blob コンテンツを上書きするブロック blob コンテンツを置きます。
uploadBlobBlocks	最初に blob ブロックのシーケンスを生成し、次にそれらを blob にコミットすることにより、ブロック blob コンテンツをアップロードします。メッセージの CommitBlockListLater プロパティーを有効にすると、後で **commitBlobBlockList** 操作を使用してコミットを実行できます。後で個々のブロック blob を更新できます。
commitBlobBlockList	以前に blob サービスにアップロードしたブロックリストに一連の blob ブロックをコミットします (メッセージ CommitBlockListLater プロパティーを有効にして **updateBlockBlob** 操作を使用)。

操作	説明
getBlobBlockList	ブロック blob リストを取得します。

blob 操作を追加する

操作	説明
createAppendBlob	追加ブロックを作成します。デフォルトでは、ブロックがすでに存在する場合はリセットされません。メッセージの **AppendBlobCreated** プロパティーを有効にし、**updateAppendBlob** 操作を使用することで、代わりに追加 blob を作成できることに注意してください。
updateAppendBlob	新しいコンテンツを blob に追加します。この操作では、blob がまだ存在しない場合、およびメッセージの **AppendBlobCreated** プロパティーを有効にした場合にも blob が作成されます。

ページブロック操作

操作	説明
createPageBlob	ページブロックを作成します。デフォルトでは、ブロックがすでに存在する場合はリセットされません。メッセージの **PageBlobCreated** プロパティーを有効にし、**updatePageBlob** 操作を使用して、ページ blob を作成 (およびその内容を設定) することもできます。
updatePageBlob	ページブロックを作成し (メッセージの **PageBlobCreated** プロパティーを有効にし、同じ名前のブロックがすでに存在する場合を除く)、この blob のコンテンツを設定します。
resizePageBlob	ページ blob のサイズを変更します。
clearPageBlob	ページ blob をクリアします。
getPageBlobRanges	ページ blob のページ範囲を取得します。

37.3.5. Azure Blob クライアントの設定

Camel アプリケーションがファイアウォールの背後で実行されている場合、または Azure Blob Client 設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
StorageCredentials credentials = new StorageCredentialsAccountAndKey("camelazure", "thekey");

CloudBlob client = new CloudBlob("camelazure", credentials);

registry.bind("azureBlobClient", client);
```

Camel azure-blob コンポーネント設定でそれを参照します。

```
from("azure-blob:/camelazure/container1/blockBlob?azureBlobClient=#client")
.to("mock:result");
```

37.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.19.0 以降) に置き換える必要があります。

37.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Azure コンポーネント

第38章 AZURE STORAGE QUEUE SERVICE コンポーネント

Camel バージョン 2.19 以降で利用可能

Azure Queue コンポーネントは、Azure Storage Queue サービスとの間でのメッセージの保存と取得をサポートしています。

前提条件

有効な Windows Azure ストレージアカウントが必要です。詳細については、Azure ドキュメントポータル を参照してください。

38.1. URI 形式

```
azure-queue://accountName/queueName[?options]
```

キューがまだ存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、**camelazure** ストレージアカウントのキュー **messageQueue** からメッセージコンテンツを取得するには、次のスニペットを使用します。

```
from("azure-queue:/camelazure/messageQueue").
to("file://queuedirectory");
```

38.2. URI オプション

Azure Storage Queue Service コンポーネントにはオプションがありません。

Azure ストレージキューサービスエンドポイントは、URI 構文を使用して設定されます。

```
azure-queue:containerAndQueueUri
```

パスおよびクエリーパラメーターを使用します。

38.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
containerAndQueueUri	**必須** コンテナーキューのコンパクトな Uri		String

38.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
azureQueueClient (common)	キューサービスクライアント		CloudQueue
credentials (common)	ストレージ認証情報を設定します。ほとんどの場合に必要です		StorageCredentials
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
messageTimeToLive (producer)	メッセージの生存時間 (秒)		int
messageVisibilityDelay (producer)	メッセージの可視性の遅延 (秒)		int
operation (プロデューサー)	プロデューサーへのキューサービス操作のヒント。	listQueues	QueueServiceOperations
queuePrefix (producer)	キューの一覧表示に使用できる接頭辞を設定します		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

必須の Azure Storage Queue Service コンポーネントオプション

containerAndQueue URI と認証情報を提供する必要があります。

38.3. 使用方法

38.3.1. Azure Storage Queue Service プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明

38.3.2. Azure Storage Queue Service プロデューサーによって設定されたメッセージヘッダー

ヘッダー	タイプ	説明

38.3.3. Azure Storage Queue Service プロデューサーコンシューマーによって設定されたメッセージヘッダー

ヘッダー	タイプ	説明

38.3.4. Azure キューサービスの操作

操作	説明
listQueues	キューを一覧表示します。
createQueue	キューを作成します。
deleteQueue	キューの削除。
addMessage	メッセージをキューに追加します。
retrieveMessage	キューからメッセージを取得します。
peekMessage	たとえば、キュー内のメッセージを表示して、メッセージが正しいキューに到着したかどうかを判断します。
updateMessage	キュー内のメッセージを更新します。
deleteMessage	キュー内のメッセージを削除します。

38.3.5. Azure キュークライアントの設定

Camel アプリケーションがファイアウォールの背後で実行されている場合、または Azure Queue Client 設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
StorageCredentials credentials = new StorageCredentialsAccountAndKey("camelazure", "thekey");

CloudQueue client = new CloudQueue("camelazure", credentials);

registry.bind("azureQueueClient", client);
```

Camel azure-queue コンポーネント設定でそれを参照します。

```
from("azure-queue:/camelazure/messageQueue?azureQueueClient=#client")
.to("mock:result");
```

38.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-azure</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.19.0 以降) に置き換える必要があります。

38.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Azure コンポーネント

第39章 BARCODE DATAFORMAT

Camel バージョン 2.14 以降で利用可能

バーコードのデータ形式は zxing ライブラリー に基づいています。このコンポーネントの目的は、文字列 (マーシャル) からバーコードイメージを作成し、バーコードイメージから文字列 (アンマーシャル) を作成することです。zxing が提供するすべての機能を自由に使用できます。

39.1. 依存関係

camel ルートでバーコードデータ形式を使用するには、このデータ形式を実装する **camel-barcode** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-barcode</artifactId>
  <version>x.x.x</version>
</dependency>
```

39.2. バーコードオプション

バーコードデータ形式は、以下に示す 5 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
width		**Integer**	バーコードの幅
height		**Integer**	バーコードの高さ
imageType		**String**	png などのバーコードのイメージタイプ
barcodeFormat		**String**	QR コードなどのバーコード形式
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

39.3. JAVA DSL を使用

まず、バーコードデータフォーマットクラスを初期化する必要があります。デフォルトのコンストラクター、またはパラメーター化されたコンストラクターのいずれかを使用できます (JavaDoc を参照)。デフォルト値は次のとおりです。

パラメーター	デフォルト値
image type (BarcodeImageType)	PNG
width	100 px
height	100 px
encoding	UTF-8
バーコード形式 (BarcodeFormat)	QR-Code

```
// QR-Code default
DataFormat code = new BarcodeDataFormat();
```

zxing ヒントを使用する場合は、BarcodeDataFormat インスタンスの addToHintMap メソッドを使用できます。

```
code.addToHintMap(DecodeHintType.TRY_HARDER, Boolean.true);
```

考えられるヒントについては、xzing のドキュメントを参照してください。

39.3.1. マーシャリング

```
from("direct://code")
  .marshal(code)
  .to("file://barcode_out");
```

以下を使用して、テストクラスからルートを呼び出すことができます。

```
template.sendBody("direct://code", "This is a testmessage!");
```

barcode_out フォルダー内に次のイメージがあります。

39.3.2. アンマーシャリング

アンマーシャラーは汎用です。アンマーシャリングには、任意の BarcodeDataFormat インスタンスを使用できます。QR コード (生成) 用と PDF417 用の 2 つのインスタンスがある場合、どちらを使用するかは問題ではありません。

```
from("file://barcode_in?noop=true")
  .unmarshal(code) // for unmarshalling, the instance doesn't matter
  .to("mock:out");
```

上記の QR コードイメージを barcode_in フォルダーに貼り付けると、モック内に 「This is a testmessage!」 が表示されるはずです。バーコードデータ形式は、ヘッダー変数として見つけることができます。

名前	タイプ	説明
BarcodeFormat	String	com.google.zxing.BarcodeFormat の値。

第40章 BASE64 DATAFORMAT

Camel バージョン 2.11 以降で利用可能

Base64 データ形式は、base64 エンコードおよびデコードに使用されます。

40.1. オプション

Base64 データ形式は、以下に示す 4 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
lineLength	76	Integer	エンコードされたデータの最大行長を指定します。デフォルトでは、76 が使用されます。
lineSeparator		String	使用する行区切り。デフォルトでは改行文字 (CRLF) を使用します。
urlSafe	false	Boolean	'' と / を発行する代わりに、- と _ をそれぞれ発行します。urlSafe は、エンコード操作にのみ適用されます。デコードは両方のモードをシームレスに処理します。デフォルトでは false です。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

Spring DSL では、次のタグを使用してデータ形式を設定します。

```
<camelContext>
  <dataFormats>
    <!-- for a newline character (\n), use the HTML entity notation coupled with the ASCII code. -->
    <base64 lineSeparator="&#10;" id="base64withNewLine" />
    <base64 lineLength="64" id="base64withLineLength64" />
  </dataFormats>
  ...
</camelContext>
```

その後、参照して後で使用できます。

```
<route>
    <from uri="direct:startEncode" />
    <marshal ref="base64withLineLength64" />
    <to uri="mock:result" />
</route>
```

デフォルトのオプションを使用する場合、ほとんどの場合、データ形式を宣言する必要はありません。その場合、以下に示すようにデータ形式をインラインで宣言できます。

40.2. MARSHAL

この例では、ファイルの内容を base64 オブジェクトにマーシャリングします。

```
from("file://data.bin")
    .marshal().base64()
    .to("jms://myqueue");
```

Spring の DSL では:

```
<from uri="file://data.bin">
<marshal>
    <base64/>
</marshal>
<to uri="jms://myqueue"/>
```

40.3. UNMARSHAL

この例では、newOrder プロセッサーによって処理される前に、ペイロードを JMS キューから byte オブジェクトにアンマーシャリングします。

```
from("jms://queue/order")
    .unmarshal().base64()
    .process("newOrder");
```

Spring の DSL では:

```
<from uri="jms://queue/order">
<marshal>
    <base64/>
</marshal>
<to uri="bean:newOrder"/>
```

40.4. 依存関係

Camel ルートで Base64 を使用するには、このデータ形式を実装する **camel-base64** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけです。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-base64</artifactId>
  <version>x.x.x</version>  <!-- use the same version as your Camel core version -->
</dependency>
```

第41章 BEAN コンポーネント

Camel バージョン 1.0 以降で利用可能

bean: コンポーネントは Bean を Camel メッセージエクスチェンジにバインドします。

41.1. URI 形式

```
bean:beanName[?options]
```

beanID には、レジストリーで Bean を検索するために使用される任意の文字列を指定できます。

41.2. オプション

Bean コンポーネントにはオプションがありません。

Bean エンドポイントは、URI 構文を使用して設定されます。

```
bean:beanName
```

パスおよびクエリーパラメーターを使用します。

41.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
beanName	***必須***: 呼び出す Bean の名前を設定します。		String

41.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
method (producer)	Bean で呼び出すメソッドの名前を設定します。		String
cache (advanced)	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリー内の Bean がシングルトンスコープとして定義されている場合、キャッシュを有効にできます。	false	boolean

名前	説明	デフォルト	タイプ
multiParameterArray (advanced)	**非推奨** メッセージ本文から渡されるパラメーターの処理方法。true の場合、メッセージボディーはパラメーターの配列である必要があります。注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。非推奨の注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。	false	boolean
parameters (advanced)	Bean の追加プロパティーの設定に使用します		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

41.3. 使用

メッセージの消費に使用されるオブジェクトインスタンスは、レジストリーに明示的に登録する必要があります。たとえば、Spring を使用している場合は、Spring 設定 **spring.xml** で Bean を定義する必要があります。または、Spring を使用しない場合は、Bean を JNDI に登録します。

エラーフォーマットマクロ: スニペット: java.lang.IndexOutOfBoundsException: インデックス: 20、サイズ: 20

エンドポイントが登録されたら、エクスチェンジの処理に使用する Camel ルートを構築できます。

bean: エンドポイントは、ルートへの入力として定義できません。つまり、消費できません。一部のインバウンドメッセージエンドポイントから Bean エンドポイントに、出力としてのみルーティングできます。**direct:** または **queue:** エンドポイントを入力として使用することを検討してください。

ProxyHelper で **createProxy()** メソッドを使用して、BeanExchange を生成して任意のエンドポイントに送信するプロキシーを作成できます。

Spring DSL を使用した同じルートの場合:

```
<route>
  <from uri="direct:hello">
  <to uri="bean:bye"/>
</route>
```

41.4. エンドポイントとしての BEAN

Camel は、エンドポイントとしての Bean の呼び出しもサポートしています。ルートは以下のとおりです。

エクスチェンジが **myBean** にルーティングされると、Camel は Bean バインディングを使用して Bean を呼び出します。
Bean のソースは plain POJO です。

Camel は Bean バインディングを使用して **sayHello** メソッドを呼び出し、エクスチェンジの In ボディを **String** タイプに変換し、メソッドの出力をエクスチェンジの Out ボディに保存します。

41.5. JAVA DSL BEAN 構文

Java DSL には、Bean コンポーネントのシンタックスシュガーが付属しています。Bean を明示的にエンドポイント (つまり **to ("bean:beanName")**) として指定する代わりに、次の構文を使用できます。

```
// Send message to the bean endpoint
// and invoke method resolved using Bean Binding.
from("direct:start").beanRef("beanName");

// Send message to the bean endpoint
// and invoke given method.
from("direct:start").beanRef("beanName", "methodName");
```

Bean への参照の名前を渡す代わりに (Camel がレジストリーでそれを検索できるようにするため)、Bean 自体を指定できます。

```
// Send message to the given bean instance.
from("direct:start").bean(new ExampleBean());

// Explicit selection of bean method to be invoked.
from("direct:start").bean(new ExampleBean(), "methodName");

// Camel will create the instance of bean and cache it for you.
from("direct:start").bean(ExampleBean.class);
```

41.6. BEAN バインディング

呼び出される Bean メソッドの選択方法 (**method** パラメーターで明示的に指定されていない場合) と、メッセージからパラメーター値が構築される方法は、Camel 内のさまざまな Bean 統合メカニズム全体で使用される Bean バインディングメカニズムによりすべて定義されます。

41.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Class コンポーネント
- Bean バインディング
- Bean インテグレーション

第42章 BEANIO DATAFORMAT

Camel バージョン 2.10 以降で利用可能

BeanIO データ形式は、BeanIO を使用してフラットペイロード (XML、CSV、区切り、または固定長形式など) を処理します。

BeanIO は、フラット形式からオブジェクト (POJO) へのマッピングを定義する マッピング XML ファイルを使用して設定されます。このマッピングファイルは必須です。

42.1. オプション

BeanIO データ形式は、以下に示す 9 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
mapping		**String**	BeanIO マッピングファイル。デフォルトでは、クラスパスからロードされます。file:、http:、または classpath: の接頭辞を付けて、マッピングファイルのロード元を示すことができます。
streamName		**String**	使用するストリームの名前。
ignoreUnidentifiedRecords	**false**	**Boolean**	未確認のレコードを無視するかどうか。
ignoreUnexpectedRecords	**false**	**Boolean**	予期しないレコードを無視するかどうか。
ignoreInvalidRecords	**false**	**Boolean**	無効なレコードを無視するかどうか。
encoding		**String**	使用する文字セット。デフォルトでは、JVM プラットフォームのデフォルトの文字セットです。
beanReaderErrorHandlerType		**String**	解析中にカスタム org.apache.camel.dataformat.beanio.BeanIOErrorHandler をエラーハンドラーとして使用する場合。エラーハンドラーの完全修飾クラス名を設定します。カスタムエラーハンドラーを使用する場合、ignoreUnidentifiedRecords、ignoreUnexpectedRecords、ignoreInvalidRecords の各オプションが使用されない場合があることに注意してください。
unmarshalSingleObject	**false**	**Boolean**	このオプションは、オブジェクトのリストとしてアンマーシャリングするか、単一のオブジェクトのみとしてアンマーシャリングするかを制御します。前者はデフォルトのモードであり、後者は beanio が Camel メッセージを単一の POJO Bean にマップする特別なユースケースのみを対象としています。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

42.2. 使用方法

マッピングファイルの例はここにあります。

42.2.1. Java DSL の使用

BeanIODataFormat を使用するには、マッピングファイルとストリームの名前を使用してデータ形式を設定する必要があります。
Java DSL では、これは以下に示すように実行できます。streamName は employeeFile です。

次に、2 つのルートがあります。最初のルートは、CSV データを List<Employee> Java オブジェクトに変換するためのものです。次に、これを分割して、モックエンドポイント
行ごとにメッセージを受け取ります。

2 番目のルートは逆の操作で、List<Employee> を CSV データのストリームに変換します。

CSV データは、たとえば次のようになります。

42.2.2. XML DSL の使用

XML で BeanIO データ形式を使用するには、以下に示すように <beanio> XML タグを使用して設定する必要があります。ルートは上記の例と同様です。

42.3. 依存関係

Camel ルートで BeanIO を使用するには、このデータ形式を実装する **camel-beanio** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-beanio</artifactId>
  <version>2.10.0</version>
</dependency>
```

第43章 BEANSTALK コンポーネント

Camel バージョン 2.15 以降で利用可能

camel-beanstalk プロジェクトは、ジョブの取得と Beanstalk ジョブの後処理のための Camel コンポーネントを提供します。

Beanstalk プロトコル で、Beanstalk ジョブのライフサイクルの詳細な説明を見つけることができます。

43.1. 依存関係

Maven ユーザーは以下の依存関係を **pom.xml** に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-beanstalk</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.15.0 以降) に置き換える必要があります。

43.2. URI 形式

```
beanstalk://[host[:port]][/tube][?options]
```

Beanstalk のデフォルト (localhost と 11300) が使用されるように、**port** のいずれか、または **host** と **port**: の両方を省略できます。**tube** を省略すると、Beanstalk コンポーネントは default という名前のチューブを使用します。

リッスンしているときに、いくつかのチューブからのジョブを監視したい場合があります。プラス記号で区切るだけです。

```
beanstalk://localhost:11300/tube1+tube2
```

チューブ名は URL デコードされるため、チューブ名に + や ? などの特殊文字が含まれている場合は、それらを適切に URL エンコードするか、RAW 構文を使用する必要があります。詳細については、こちら を参照してください。

ちなみに、ジョブを Beanstalk に書き込む場合、複数のチューブを指定することはできません。

43.3. BEANSTALK オプション

Beanstalk コンポーネントは、以下にリストされている 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
connectionSettings Factory (common)	Custom ConnectionSettingsFactory.Beanstalkd への接続に使用する ConnectionSettingsFactory を指定します。特に、beanstalkd デーモンを使用しない単体テストに役立ちます (ConnectionSettings をモックできます)。		ConnectionSettings Factory
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Beanstalk エンドポイントは、URI 構文を使用して設定されます。

```
beanstalk:connectionSettings
```

パスおよびクエリーパラメーターを使用します。

43.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionSettings	接続設定 ホスト: ポート/チューブ		String

43.3.2. クエリーパラメーター(26 個のパラメーター):

名前	説明	デフォルト	タイプ
command (common)	put は、ジョブを Beanstalk に入れることを意味します。ジョブボディは Camel メッセージボディで指定します。ジョブ ID は、beanstalk.jobId メッセージヘッダーで返されます。メッセージヘッダー Beanstalk.jobId のジョブ ID を期待して、削除、解放、タッチ、または埋めます。オペレーションの結果は、beanstalk.result メッセージヘッダーで返されます。		BeanstalkCommand
jobDelay (common)	秒単位のジョブ遅延。	0	int
jobPriority (common)	仕事の優先順位。(0 が最高です。Beanstalk プロトコルを参照してください)	1000	long

名前	説明	デフォルト	タイプ
jobTimeToRun (common)	ジョブの実行時間 (秒単位)。(0 の場合、beanstalkd デーモンはそれを自動的に 1 に上げます。Beanstalk プロトコルを参照してください)	60	int
awaitJob (consumer)	Beanstalk からジョブを確認する前に、ジョブの完了を待機するかどうか	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
onFailure (consumer)	処理が失敗したときに使用するコマンド。		BeanstalkCommand
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
useBlockIO (consumer)	blockIO を使用するかどうか。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map

名前	説明	デフォルト	タイプ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

プロデューサーの動作は、ジョブの処理方法を指示する **command** パラメーターの影響を受けます。

コンシューマーは、ジョブを予約した直後にジョブを削除するか、Camel ルートが処理するまで待つことができます。最初のシナリオはメッセージキューに似ていますが、2 番目のシナリオはジョブキューに似ています。この動作は、**consumer.awaitJob** パラメーターによって制御されます。これは、デフォルトで **true** に等しくなります (Beanstalkd の性質に従います)。

同期の場合、コンシューマーはジョブが正常に完了すると **delete** を呼び出し、失敗すると **bury** を呼び出します。URI で **consumer.onFailure** パラメーターを指定することにより、失敗した場合に実行するコマンドを選択できます。**bury**、**delete**、または **release** の値を取ることができます。

JavaBeanstalkClient ライブラリーの同じパラメーターに対応するブール値パラメーター **consumer.useBlockIO** があります。デフォルトでは **true** です。

release を指定するときは注意してください。失敗したジョブはすぐに同じチューブで利用可能になり、コンシューマーはそれを再度取得しようとします。ただし、**jobDelay** を **release** して指定することはできます。

Beanstalk コンシューマーは、スケジュールされた ポーリングコンシューマー です。つまり、コンシューマーがポーリングする頻度など、設定できるオプションがさらにあります。詳細については、コンシューマーのポーリングを参照してください。

43.4. コンシューマーヘッダー

コンシューマーは、Exchange メッセージにいくつかのジョブヘッダーを格納します。

プロパティー	タイプ	説明
beanstalk.jobId	long	ジョブ ID:
beanstalk.tube	string	このジョブを含むチューブの名前

プロパティー	タイプ	説明
beanstalk.state	string	ready または delayed または reserved または burried (reserved でなければならない)
beanstalk.priority	long	プライオリティ値セット
beanstalk.age	int	このジョブを作成した put コマンドからの時間 (秒)
beanstalk.time-left	int	サーバーがこのジョブを準備完了キューに入れるまでの残り秒数
beanstalk.timeouts	int	予約中にこのジョブがタイムアウトになった回数
beanstalk.releases	int	クライアントがこのジョブを予約から解放した回数
beanstalk.buries	int	このジョブが葬られた回数
beanstalk.kicks	int	このジョブがキックされた回数

43.5. 例

この Camel コンポーネントを使用すると、処理するジョブを要求し、それらを Beanstalkd デーモンに提供できます。簡単なデモルートは次のようになります

```
from("beanstalk:testTube").
  log("Processing job #${property.beanstalk.jobId} with body ${in.body}").
  process(new Processor() {
    @Override
    public void process(Exchange exchange) {
      // try to make integer value out of body
      exchange.getIn().setBody( Integer.valueOf(exchange.getIn().getBody(classOf[String])) );
    }
  }).
  log("Parsed job #${property.beanstalk.jobId} to body ${in.body}");
```

```
from("timer:dig?period=30seconds").
  setBody(constant(10)).log("Kick ${in.body} buried/delayed tasks").
  to("beanstalk:testTube?command=kick");
```

最初のルートでは、チューブ testTube で新しいジョブをリッスンしています。それらが到着すると、メッセージ本文から整数値を解析しようとしています。成功した場合はログに記録し、交換が正常に完了すると、Camel コンポーネントはこのジョブを Beanstalk から自動的に **削除** します。反対に、ジョブデータを解析できない場合、エクスチェンジは失敗し、Camel コンポーネントはデフォルトでそれを **葬ります**。そのため、後で処理できるか、失敗したジョブを手動で検査することになります。

したがって、2 番目のルートは Beanstalk に定期的に 10 個のジョブを葬った状態や遅延状態から通常のキューに **追い出すよう** に要求します。

43.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第44章 BEAN バリデーターコンポーネント

Camel バージョン 2.3 以降で利用可能

バリデーターコンポーネントは、Java Bean Validation API (JSR 303) を使用してメッセージボディーの Bean 検証を実行します。Camel は Hibernate Validator のリファレンス実装を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bean-validator</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

44.1. URI 形式

```
bean-validator:label[?options]
```

または

```
bean-validator://label[?options]
```

label は、エンドポイントを記述する任意のテキスト値です。
URI には、?option=value&option=value&... の形式でクエリーオプションを追加できます。

44.2. URI オプション

Bean Validator コンポーネントにはオプションがありません。

Bean バリデーターエンドポイントは URI 構文を使用して設定されます。

```
bean-validator:label
```

パスおよびクエリーパラメーターを使用します。

44.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
label	**必須**: ラベルは、エンドポイントを記述する任意のテキスト値です。		String

44.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
constraintValidatorFactory (producer)	カスタムの ConstraintValidatorFactory を使用します		ConstraintValidator Factory
group (producer)	カスタム検証グループを使用します	javax.validation.groups.Default	String
messageInterpolator (producer)	カスタムの MessageInterpolator を使用します		MessageInterpolator
traversableResolver (producer)	カスタムの TraversableResolver を使用します		TraversableResolver
validationProviderResolver (producer)	カスタムの ValidationProviderResolver を使用します		ValidationProviderResolver
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

44.3. OSGI デプロイメント

OSGi 環境で Hibernate Validator を使用するには、**org.apache.camel.component.bean.validator.HibernateValidationProviderResolver** と同様に専用の **ValidationProviderResolver** 実装を使用します。以下のスニペットは、このアプローチを示しています。Camel 2.13.0 から **HibernateValidationProviderResolver** を使用できることに注意してください。

Using HibernateValidationProviderResolver

```
from("direct:test").
  to("bean-validator://ValidationProviderResolverTest?
validationProviderResolver=#myValidationProviderResolver");

...

<bean id="myValidationProviderResolver"
class="org.apache.camel.component.bean.validator.HibernateValidationProviderResolver"/>
```

カスタムの **ValidationProviderResolver** が定義されておらず、バリデーターコンポーネントが OSGi 環境にデプロイされている場合、**HibernateValidationProviderResolver** は自動的に使用されます。

44.4. 例

以下を持つ java Bean があると仮定します。アノテーション:

Car.java

```
public class Car {

    @NotNull
    private String manufacturer;

    @NotNull
    @Size(min = 5, max = 14, groups = OptionalChecks.class)
    private String licensePlate;

    // getter and setter
}
```

カスタムバリデーショングループのインターフェイス定義:

OptionalChecks.java

```
public interface OptionalChecks {
}
```

以下の Camel ルート。manufacturer および licensePlate 属性の **@NotNull** 制約のみが検証されます (Camel はデフォルトのグループ **javax.validation.groups.Default** を使用します)。

```
from("direct:start")
.to("bean-validator://x")
.to("mock:end")
```

OptionalChecks グループからの制約を確認する場合は、以下のようなルートを定義する必要があります。

```
from("direct:start")
.to("bean-validator://x?group=OptionalChecks")
.to("mock:end")
```

両方のグループからの制約を確認する場合は、最初に新しいインターフェイスを定義する必要があります。

AllChecks.java

```
@GroupSequence({Default.class, OptionalChecks.class})
public interface AllChecks {
}
```

ルート定義は以下のようになります。

```
from("direct:start")
.to("bean-validator://x?group=AllChecks")
.to("mock:end")
```

また、独自のメッセージインターポレーター、通過可能なリゾルバー、および制約バリデーターファクトリーを提供する必要がある場合は、次のようなルートを記述する必要があります。

```
<bean id="myMessageInterpolator" class="my.ConstraintValidatorFactory" />
<bean id="myTraversableResolver" class="my.TraversableResolver" />
<bean id="myConstraintValidatorFactory" class="my.ConstraintValidatorFactory" />

from("direct:start")
.to("bean-validator://x?group=AllChecks&messageInterpolator=#myMessageInterpolator
&traversableResolver=#myTraversableResolver&constraintValidatorFactory=#myConstraintValidatorFac
tory")
.to("mock:end")
```

また、制約を Java アノテーションではなく、XML として記述することも可能です。この場合、次のようなファイル **META-INF/validation.xml** を提供する必要があります。

validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<validation-config
    xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration">
    <default-provider>org.hibernate.validator.HibernateValidator</default-provider>
    <message-
interpolator>org.hibernate.validator.engine.ResourceBundleMessageInterpolator</message-
interpolator>
    <traversable-
resolver>org.hibernate.validator.engine.resolver.DefaultTraversableResolver</traversable-resolver>
    <constraint-validator-
factory>org.hibernate.validator.engine.ConstraintValidatorFactoryImpl</constraint-validator-factory>

    <constraint-mapping>/constraints-car.xml</constraint-mapping>
</validation-config>
```

constraints-car.xml ファイル

constraints-car.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<constraint-mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/mapping validation-mapping-1.0.xsd"
    xmlns="http://jboss.org/xml/ns/javax/validation/mapping">
    <default-package>org.apache.camel.component.bean.validator</default-package>

    <bean class="CarWithoutAnnotations" ignore-annotations="true">
        <field name="manufacturer">
            <constraint annotation="javax.validation.constraints.NotNull" />
        </field>

        <field name="licensePlate">
            <constraint annotation="javax.validation.constraints.NotNull" />

            <constraint annotation="javax.validation.constraints.Size">
                <groups>
                    <value>org.apache.camel.component.bean.validator.OptionalChecks</value>
                </groups>
                <element name="min">5</element>
```

```
            <element name="max">14</element>
        </constraint>
    </field>
  </bean>
</constraint-mappings>
```

OrderedChecks が https://github.com/apache/camel/blob/master/components/camel-bean-validator/src/test/java/org/apache/camel/component/bean/validator/OrderedChecks.java になるルート定義の例の XML 構文を次に示します。

ボディーには検証するクラスのインスタンスを含める必要があることに注意してください。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:start"/>
            <to uri="bean-validator://x?
group=org.apache.camel.component.bean.validator.OrderedChecks"/>
        </route>
    </camelContext>
</beans>
```

44.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第45章 BINDING コンポーネント (非推奨)

Camel バージョン 2.11 以降で利用可能

Camel の用語では、**バインディング** はコントラクトでエンドポイントをラップする方法です。データ形式、コンテンツエンリッチャー、検証ステップなど。バインディングは完全にオプションであり、任意の camel エンドポイントで使用することを選択できます。

バインディングは、Camel などのさまざまなテクノロジーにサービスコントラクトを追加する SwitchYard プロジェクト の作業によって引き起こされます。ただし、Camel を SCA でラップする SwitchYard アプローチではなく、**Camel Bindings** は、Camel フレームワーク自体内のコントラクトで Camel エンドポイントをラップする方法を提供します。そのため、どの Camel ルート内でも簡単に使用できます。

45.1. オプション

Binding コンポーネントにはオプションがありません。

Binding エンドポイントは、URI 構文を使用して設定されます。

```
binding:bindingName:delegateUri
```

パスおよびクエリーパラメーターを使用します。

45.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
bindingName	**必須** Camel レジストリーでルックアップするバインディングの名前。		String
delegateUri	デリゲートエンドポイントの **必須** Uri。		String

45.1.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

45.2. バインディングの使用

Binding は現在、コントラクトを定義する Bean です (ただし、バインディングを Camel DSL に追加できることが要望されます)。

バインドされたエンドポイント (バインディングでバインドされたエンドポイント) を定義するには、いくつかの方法があります。

45.3. バインディング URI の使用

任意のエンドポイント URI の前に **binding:nameOfBinding:** を付けることができます。**nameOfBinding** は、レジストリー内のバインディング Bean の名前です。

```
from("binding:jaxb:activemq:myQueue").to("binding:jaxb:activemq:anotherQueue")
```

ここでは、jaxb バインディングを使用しています。これは、たとえば、JAXB データ形式を使用してメッセージをマーシャリングおよびアンマーシャリングすることができます。

45.4. BINDINGCOMPONENT の使用

BindingComponent と呼ばれるコンポーネントがあり、依存性注入によってレジストリーで設定できます。これにより、すでに何らかのバインディングにバインドされているエンドポイントを作成できます。

たとえば、次のようなコードを使用してレジストリーに jsonmq という新しいコンポーネントを登録した場合

```
JacksonDataFormat format = new JacksonDataFormat(MyBean.class);
context.bind("jsonmq", new BindingComponent(new DataFormatBinding(format), "activemq:foo."));
```

その後、エンドポイントを他のエンドポイントであるかのように使用できます。

```
from("jsonmq:myQueue").to("jsonmq:anotherQueue")
```

これは、キュー foo.myQueue および foo.anotherQueue を使用し、指定された Jackson Data Format を使用してキューのオンとオフをマーシャリングします。

45.5. バインディングを使うタイミング

1 つのルートで 1 回だけエンドポイントを使用する場合。バインディングは、実際には raw エンドポイントを直接使用し、camel ルートで通常どおり明示的なマーシャリングと検証を使用するよりも複雑で、より多くの作業が必要になる場合があります。

ただし、バインドは、多くのルートを一緒に設定する場合に役立ちます。または、入力エンドポイントと出力エンドポイントが設定されたテンプレートとして単一のルートを使用します。バインディングは、コントラクトとエンドポイントをまとめるための優れた方法を提供します。

バインディングのもう 1 つの適切な使用例は、同じバインディングを使用する多くのエンドポイントを使用している場合です。常に特定のデータ形式や検証規則について言及する必要はなく、BindingComponent を使用して、選択したバインディングでエンドポイントをラップすることができます。

したがって、バインディングは実際には設定ツールです。意味のある場合にのみ使用してください。このような複雑さは、ルートやエンドポイントの数が多くない限り、その価値はないかもしれません。

第46章 BINDY DATAFORMAT

Camel バージョン 2.0 以降で利用可能

このコンポーネントの目的は、アノテーションでバインディングマッピングが定義された Java Bean との間で、非構造化データ (より正確には非 XML データ)
の解析/バインディングを可能にすることです。Bindy を使用すると、次のようなソースからデータをバインドできます。

- CSV レコード、
- 固定長レコード、
- FIX メッセージ、
- またはほとんどすべての非構造化データ

1 つまたは複数の Plain Old Java Object (POJO)。Bindy は、Java プロパティーの型に従ってデータを変換します。POJO は、場合によっては利用可能な 1 対多の関係と合わせてリンクできます。さらに、Date、Double、Float、Integer、Short、Long、BigDecimal などのデータ型の場合に、プロパティーのフォーマット中に適用するパターンを指定できます。

BigDecimal 数値の場合、精度と小数点またはグループ区切り記号も定義できます。

タイプ	フォーマットの種類	パターン例	リンク
Date	DateFormat	**dd-MM-yyyy**	http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html
Decimal*	DecimalFormat	**..##**	http://java.sun.com/j2se/1.5.0/docs/api/java/text/DecimalFormat.html

Decimal* = Double、Integer、Float、Short、Long

> *Format supported*

この最初のリリースでは、コンマ区切りの値フィールドとキーと値のペアフィールド (例: FIX メッセージ) のみがサポートされています。

camel-bindy を使用するには、最初にモデルをパッケージ (例: com.acme.model) で定義し、モデルクラス (例: Order、Client、Instrument など) ごとに必要なアノテーション (後述) をクラスまたはフィールドに追加する必要があります。

> *Multiple models*

複数のモデルを使用する場合は、予測できない結果を防ぐために、各モデルを独自のパッケージに配置する必要があります。

Camel 2.16 以降では、パッケージ名の代わりにクラス名を使用して bindy を設定するため、同じパッケージに複数のモデルを安全に含めることができるため、これは当てはまりません。

46.1. オプション

Bindy データ形式は、以下に示す 5 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
type		BindyType	csv、固定、またはキーと値のペアのモードを使用するかどうか。デフォルト値は、選択したデータ形式に応じて Csv または KeyValue のいずれかです。
classType		String	使用するモデルクラスの名前。
locale		String	米国の us など、使用する既定のロケールを設定します。JVM プラットフォームのデフォルトロケールを使用するには、default という名前を使用します。
unwrapSingleInstance	true	Boolean	アンマーシャリング時に、java.util.List にラップする代わりに、単一のインスタンスをアンラップして返す必要があります。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

46.2. アノテーション

作成されたアノテーションにより、モデルのさまざまな概念を次のように POJO にマップできます。

- レコードのタイプ (csv、キーと値のペア (FIX メッセージなど)、固定長など)、
- リンク (別のオブジェクトにあるオブジェクトをリンクする)、
- DataField とそのプロパティー (int、type、...)、
- KeyValuePairField (FIX 財務メッセージにあるようなキー = 値形式の場合)、
- セクション (ヘッダー、本文、およびフッターセクションを識別する)、
- OneToMany,
- BindyConverter (since 2.18.0),
- FormatFactory (2.18.0 以降)

このセクションでは、上記について説明します。

46.3. 1.CSVRECORD

CsvRecord アノテーションは、モデルのルートクラスを識別するために使用されます。レコード = CSV ファイルの 1 行を表し、複数の子モデルクラスにリンクできます。

アノテーション名	レコードの種類	レベル
CsvRecord	csv	Class

パラメーター名	type	Info
separator	string	必須 - ',' または ';' または 'anything' を設定できます。この値は正規表現として解釈されます。正規表現で特別な意味を持つ記号を使用する場合は、'|' の記号であれば、'|' のようにマスクする必要があります
skipFirstLine	boolean	オプション - デフォルト値 = false - CSV ファイルの最初の行をスキップできます
crlf	string	オプション - 可能な値 = WINDOWS、UNIX、MAC、またはカスタム。デフォルト値。WINDOWS - 使用するキャリッジリターン文字を定義できます。上記の 3 つ以外の値を指定すると、入力した値 (カスタム) が CRLF 文字として使用されます
generateHeaderColumns	boolean	オプション - デフォルト値 = false - CSV 生成のヘッダー列を生成するために使用します
autospanLine	boolean	**Camel 2.13/2.12.2:** オプション - デフォルト値 = false - 有効にすると、最後の列が自動的に行末に改行されます。たとえば、コメントなどの場合、これにより、行にすべての文字と区切り文字を含めることができます。
isOrdered	boolean	オプション - デフォルト値 = false - CSV の生成時にフィールドの順序を変更できます
quote	String	**Camel 2.8.3/2.9:** オプション - CSV の生成時にフィールドの引用符を指定できるようになりました。このアノテーションは、モデルのルートクラスに関連付けられており、一度宣言する必要があります。
引用	boolean	*Camel 2.11:*optional - デフォルト値 = false - CSV 生成時のマーシャリング時に値 (およびヘッダー) を引用符で囲む必要があるかどうかを示します。
endWithLineBreak	boolean	**Camel 2.21:** オプション - デフォルト値 = true - CSV 生成ファイルを改行で終了する必要があるかどうかを示します。

ケース 1: separator = ','

CSV レコード内のフィールド分離に使用されるセパレーターは ',' です。

```
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500,
USD,08-01-2009
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

case 2 : separator = ';'

前のケースと比較して、ここでのセパレーターは ',' の代わりに ';'を使用しています。

10; J; Pauline; M; XD12345678; Fortis Dynamic 15/15; 2500; USD; 08-01-2009

```
@CsvRecord( separator = ";" )
public Class Order {

}
```

case 3 : separator = '|'

前のケースと比較して、ここでのセパレーターは ';' の代わりに '|' を使用しています。

```
10| J| Pauline| M| XD12345678| Fortis Dynamic 15/15| 2500| USD|
08-01-2009
```

```
@CsvRecord( separator = "\\|" )
public Class Order {

}
```

ケース 4: separator = '\",\"'

Camel 2.8.2 以前に該当

CSV レコードの解析対象フィールドに ',' または ';' が含まれる場合。これはセパレーターとしても使用されるため、Camel バインドにこのケースの処理方法を伝える別のストラテジー
を見つける必要があります。データを含むフィールドをコンマで定義するには、単純引用符または二重引用符を区切り文字として使用します
(例: '10', 'Street 10, NY', 'USA' or "10", "Street 10, NY", "USA")。
注意: この場合、単純または二重引用符である行の最初と最後の文字は bindy によって削除されます。

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15"
2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "\",\"" )
public Class Order {

}
```

Camel 2.8.3/2.9 または never bindy から、レコードが一重引用符または二重引用符で囲まれているかどうかが自動的に検出され、CSV からオブジェクトにアンマーシャリングするときにそれらの引用符が自動的に削除されます。したがって、セパレーターに引用符を **含めず** に、単に以下を実行するだけで

す。

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15"
2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

Object から CSV にマーシャリングして引用符を使用する場合は、以下に示すように @CsvRecord の **quote** 属性を使用して、使用する引用符を指定する必要があることに注意してください。

```
@CsvRecord( separator = ",", quote = "\"" )
public Class Order {

}
```

case 5 : separator & skipfirstline

この機能は、クライアントがファイルの最初の行にデータフィールドの名前を入れる場合に役立ちます。

注文 ID、クライアント ID、名、姓、isin コード、商品名、数量、通貨、日付

解析プロセス中にこの最初の行をスキップする必要があることを bindy に通知するには、次の属性を使用します。

```
@CsvRecord(separator = ",", skipFirstLine = true)
public Class Order {

}
```

ケース 6: generateHeaderColumns

生成された CSV の最初の行に追加するには、次のようにアノテーションで属性 generateHeaderColumns を true に設定する必要があります。

```
@CsvRecord( generateHeaderColumns = true )
public Class Order {

}
```

その結果、アンマーシャリングプロセス中の Bindy は、次のような CSV を生成します。

注文 ID、クライアント ID、名、姓、isin コード、商品名、数量、通貨、日付

```
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500, USD,08-01-2009
```

case 7 : carriage return

camel-bindy が実行されるプラットフォームが Windows ではなく、Macintosh または Unix である場合には、次のように crlf プロパティーを変更できます。WINDOWS、UNIX、または MAC の 3 つの値を使用できます。

```
@CsvRecord(separator = ",", crlf="MAC")
public Class Order {

}
```

さらに、何らかの理由で別の行末文字を追加する必要がある場合は、crlf パラメーターを使用して指定できます。次の例では、コンマとそれに続く改行文字で行を終了できます。

```
@CsvRecord(separator = ",", crlf=",\n")
public Class Order {

}
```

ケース 8: isOrdered

モデルから CSV レコードを作成する際に従う順序が、解析時に使用される順序と異なる場合があります。次に、この場合、属性 isOrdered = true を使用して、DataField アノテーションの属性 position と組み合わせて指定できます。

```
@CsvRecord(isOrdered = true)
public Class Order {

   @DataField(pos = 1, position = 11)
   private int orderNr;

   @DataField(pos = 2, position = 10)
   private String clientNr;

}
```

備考 : pos はファイルの解析に使用され、stream は position を使用して CSV を生成します

46.4. 2.リンク

リンクアノテーションを使用すると、オブジェクトを相互にリンクできます。

アノテーション名	レコードの種類	レベル
リンク	all	クラスとプロパティー

パラメーター名	type	Info
linkType	LinkType	オプション - デフォルトの値は LinkType.oneToOne であるため、言及する義務はありません。

1 対 1 の関係のみが許可されます。

例: モデルクラス Client が Order クラスにリンクされている場合には、次のように Order クラスでアノテーション Link を使用します。

プロパティーリンク

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @Link
    private Client client;
}
```

クラス Client の場合:

クラスリンク

```
@Link
public class Client {

}
```

46.5. 3.DATAFIELD

DataField アノテーションは、フィールドのプロパティーを定義します。各データフィールドは、レコード内の位置、タイプ (string、int、date など)、およびパターン (任意) によって識別されます

アノテーション名	レコードの種類	レベル
DataField	all	プロパティー

パラメーター名	type	Info
pos	int	必須 - フィールドの **入力** 位置。1 から ... までの桁数 - position パラメーターを参照してください。
pattern	string	オプション - デフォルト値 = "" - Decimal、Date、
長さ	int	オプション - 固定長形式のフィールドの長さを表します

パラメーター名	type	Info
精度	int	オプション - 10 進数がフォーマット/解析されるときに使用される精度を表します
pattern	string	オプション - デフォルト値 = "" - Java フォーマッター (例では SimpleDateFormat) によってデータのフォーマット/検証に使用されます。パターンを使用する場合は、バインドデータ形式にロケールを設定することをお勧めします。us などの既知のロケールに設定するか、default を使用してプラットフォームのデフォルトロケールを使用します。デフォルトには Camel 2.14/2.13.3/2.12.5 が必要であることに注意してください。
position	int	オプション - 生成された CSV 内のフィールドの位置 (出力メッセージ) が、入力位置 (pos) と比較して異なる必要がある場合に使用します。pos パラメーターを参照してください。
必須	boolean	optional - default value = "false"
trim	boolean	optional - default value = "false"
defaultValue	string	**Camel 2.10:** オプション - デフォルト値 = "" - それぞれの CSV フィールドが空/利用できない場合のフィールドのデフォルト値を定義します
impliedDecimalSeparator	boolean	**Camel 2.11:** オプション - デフォルト値 = "false" - 指定された位置に暗黙の小数点があるかどうかを示します
lengthPos	int	**Camel 2.11** : オプション - このフィールドの固定長を定義する固定長レコード内のデータフィールドを識別するために使用できます
align	string	オプション - デフォルト値 = "R" - テキストを固定長フィールド内で右または左に揃えます。値 R または L を使用してください
delimiter	string	**Camel 2.11:** オプション - 固定長レコード内の可変長フィールドの終わりを区切るために使用できます

ケース 1: pos

このパラメーター/属性は、csv レコード内のフィールドの位置を表します

Position

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;
```

```
    @DataField(pos = 5)
    private String isinCode;

}
```

この例でわかるように、位置は 1 から始まりますが、クラス Order では続きは 5 になります。2 から 4 までの数字はクラス Client で定義されています (以下を参照)。

位置は別のモデルクラスで継続する

```
public class Client {

    @DataField(pos = 2)
    private String clientNr;

    @DataField(pos = 3)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

ケース 2: pattern

パターンを使用すると、データの形式を強化または検証できます

Pattern

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    // pattern used during parsing or when the date is created
    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 3: precision

精度は、数値の小数部分を定義する場合に役立ちます

精度

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @Link
    private Client client;

    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 4: 出力で位置が異なる

position 属性は、生成された CSV レコードにフィールドを配置する方法を bindy に通知します。デフォルトでは、使用される位置は属性 pos で定義された位置に対応します。位置が異なる場合 (つまり、マーシャリングとアンマーシャリングを比較する非対称プロセスがあることを意味します)、position を使用してこれを指定できます。

以下に例を示します。

出力位置が異なる

```
@CsvRecord(separator = ",", isOrdered = true)
public class Order {

    // Positions of the fields start from 1 and not from 0

    @DataField(pos = 1, position = 11)
    private int orderNr;

    @DataField(pos = 2, position = 10)
    private String clientNr;

    @DataField(pos = 3, position = 9)
    private String firstName;

    @DataField(pos = 4, position = 8)
    private String lastName;

    @DataField(pos = 5, position = 7)
    private String instrumentCode;
```

```
    @DataField(pos = 6, position = 6)
    private String instrumentNumber;
}
```

アノテーション @DataField のこの属性は、アノテーション @CsvRecord の属性 isOrdered = true と組み合わせて使用する必要があります。

ケース 5: 必須

フィールドが必須の場合は、required 属性を true に設定して使用します。

必須

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2, required = true)
    private String clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, required = true)
    private String lastName;
}
```

このフィールドがレコードに存在しない場合には、パーサーにより次の情報を含めてエラーが報告されます。

一部のフィールドが欠落しています (オプションまたは必須)、行:

case 6 : trim

フィールドの先頭および/または末尾にスペースがあり、処理する前に削除する必要がある場合は、trim 属性を true に設定してください。

Trim

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1, trim = true)
    private int orderNr;

    @DataField(pos = 2, trim = true)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

case 7 : defaultValue

フィールドが定義されていない場合は、defaultValue 属性で示される値が使用されます

デフォルト値

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, defaultValue = "Barin")
    private String lastName;
}
```

この属性は、オプションのフィールドにのみ適用されます。

46.6. 4.FIXEDLENGTHRECORD

FixedLengthRecord アノテーションは、モデルのルートクラスを識別するために使用されます。これは、レコード = フォーマットされた固定長のデータを含むファイル/メッセージの行を表し、複数の子モデルクラスにリンクできます。フィールドのデータは右または左に位置合わせできるため、この形式は少し特殊です。
データのサイズがフィールドの長さを完全に満たしていない場合は、埋め込み文字を追加できます。

アノテーション名	レコードの種類	レベル
FixedLengthRecord	固定:	クラス

パラメーター名	type	Info
crlf	string	オプション - 可能な値 = WINDOWS、UNIX、MAC、またはカスタム。デフォルト値。WINDOWS - 使用するキャリッジリターン文字を定義できます。上記の 3 つ以外の値を指定すると、入力した値 (カスタム) が CRLF 文字として使用されます。このオプションはマーシャリング中にのみ使用されますが、アンマーシャリングでは、eol がカスタマイズされていない限り、JDK が提供するシステムのデフォルトの行区切り文字が使用されます

パラメーター名	type	Info
eol	string	オプション - 空の文字列である default=""。アンマーシャリング中に各レコードの後に行末を考慮して処理するために使用される文字 (オプション - デフォルト = "" は、他の行区切り文字が提供されない限り、JDK が提供するデフォルトの行区切り文字を使用するのに役立ちます)。このオプションは、アンマーシャリング中にのみ使用されます。マーシャリングでは、他の値が指定されていない限り、システムのデフォルトで提供される行区切り文字が WINDOWS として使用されます。
paddingChar	char	mandatory - default value = ' '
長さ	int	必須 = 固定長レコードのサイズ
hasHeader	boolean	**Camel 2.11** - オプション - このタイプのレコードの前に、ファイル/ストリームの先頭にある単一のヘッダーレコードがあることを示します
hasFooter	boolean	**Camel 2.11** - オプション - このタイプのレコードの後に、ファイル/ストリームの最後に単一のフッターレコードが続く可能性があることを示します
skipHeader	boolean	**Camel 2.11** - オプション - ヘッダーレコードのマーシャリング/アンマーシャリングをスキップするようにデータ形式を設定します。プライマリーレコードでこのパラメーターを設定します (たとえば、ヘッダーやフッターではありません)。
skipFooter	boolean	**Camel 2.11** - オプション - フッターレコードのマーシャリング/アンマーシャリングをスキップするようにデータフォーマットを設定します。
isHeader	boolean	**Camel 2.11** - オプション - この FixedLengthRecord をヘッダーレコードとして識別します
isFooter	boolean	**Camel 2.11** - オプション - この FixedLengthRecords をフッターレコードとして識別します
ignoreTrailingChars	boolean	**Camel 2.11.1** - オプション - アンマーシャリング/解析時に、最後にマップされたフィールドを超える文字を無視できることを示します。 このアノテーションは、モデルのルートクラスに関連付けられており、一度宣言する必要があります。

hasHeader/hasFooter パラメーターは、isHeader/isFooter と相互に排他的です。レコードは、ヘッダー/フッターと、プライマリーの固定長レコードを両方に指定できません。

ケース 1: 単純な固定長レコード

この単純な例は、固定メッセージを解析/フォーマットするモデル設計方法を示しています

```
10A9PaulineMISINXD12345678BUYShare2500.45USD01-08-2009
```

Fixed-simple

```
@FixedLengthRecord(length=54, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;

    @DataField(pos = 3, length=2)
    private String clientNr;

    @DataField(pos = 5, length=7)
    private String firstName;

    @DataField(pos = 12, length=1, align="L")
    private String lastName;

    @DataField(pos = 13, length=4)
    private String instrumentCode;

    @DataField(pos = 17, length=10)
    private String instrumentNumber;

    @DataField(pos = 27, length=3)
    private String orderType;

    @DataField(pos = 30, length=5)
    private String instrumentType;

    @DataField(pos = 35, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 42, length=3)
    private String currency;

    @DataField(pos = 45, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 2: アラインメントとパディングのある固定長レコード

こちらの詳細な例は、フィールドの配置を定義する方法と、' ' here'' のパディング文字を割り当てる方法を示しています。

```
10A9 PaulineM ISINXD12345678BUYShare2500.45USD01-08-2009
```

Fixed-padding-align

```
@FixedLengthRecord(length=60, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;

    @DataField(pos = 3, length=2)
    private String clientNr;
```

```
    @DataField(pos = 5, length=9)
    private String firstName;

    @DataField(pos = 14, length=5, align="L")   // align text to the LEFT zone of the block
    private String lastName;

    @DataField(pos = 19, length=4)
    private String instrumentCode;

    @DataField(pos = 23, length=10)
    private String instrumentNumber;

    @DataField(pos = 33, length=3)
    private String orderType;

    @DataField(pos = 36, length=5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 48, length=3)
    private String currency;

    @DataField(pos = 51, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 3: フィールドパディング

' ' の代わりに '0' をパディングする数値形式があるため、レコードに定義されたデフォルトのパディングをフィールドに適用できない場合があります。この場合、モデルで属性 paddingField を使用してこの値を設定できます。

```
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-padding-field

```
@FixedLengthRecord(length = 65, paddingChar = ' ')
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    @DataField(pos = 5, length = 9)
    private String firstName;

    @DataField(pos = 14, length = 5, align = "L")
    private String lastName;

    @DataField(pos = 19, length = 4)
    private String instrumentCode;
```

```
    @DataField(pos = 23, length = 10)
    private String instrumentNumber;

    @DataField(pos = 33, length = 3)
    private String orderType;

    @DataField(pos = 36, length = 5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 53, length = 3)
    private String currency;

    @DataField(pos = 56, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 4: 区切り文字付きの固定長レコード

固定長レコードでは、レコード内のコンテンツが区切られている場合があります。次の例では、firstName フィールドと lastName フィールドが '^' 文字で区切られています。

```
10A9Pauline^M^ISINXD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-delimited

```
@FixedLengthRecord()
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;
```

```
    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;

    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

Camel 2.11 では、固定長レコードの pos 値は、正確な列番号の代わりに序数の連続した値を使用してオプションで定義できます。

ケース 5: レコード定義フィールド長の固定長レコード

固定長レコードには、同じレコード内の別のフィールドで必要とされる長さを定義するフィールドが含まれる場合があります。次の例では、instrumentNumber フィールド値の長さは、レコード内の instrumentNumberLen フィールドの値によって定義されます。

```
10A9Pauline^M^ISIN10XD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-delimited

```
@FixedLengthRecord()
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 2, align = "R", paddingChar = '0')
    private int instrumentNumberLen;

    @DataField(pos = 7, lengthPos=6)
    private String instrumentNumber;

    @DataField(pos = 8, length = 3)
    private String orderType;

    @DataField(pos = 9, length = 5)
    private String instrumentType;

    @DataField(pos = 10, precision = 2, length = 12, paddingChar = '0')
```

```
    private BigDecimal amount;

    @DataField(pos = 11, length = 3)
    private String currency;

    @DataField(pos = 12, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 6: ヘッダーとフッター付きの固定長レコード

Bindy は、モデルの一部として設定されている固定長のヘッダーおよびフッターレコードを検出します。ただし、アノテーション付きのクラスが、プライマリー@FixedLengthRecord クラスと同じパッケージに存在するか、設定されたスキャンパッケージの 1 つに含まれている必要があります。次のテキストは、ヘッダーレコードとフッターレコードで囲まれた 2 つの固定長レコードを示しています。

```
101-08-2009
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
10A9 RichN ISINXD12345678BUYShare000002700.45USD01-08-2009
9000000002
```

Fixed-header-and-footer-main-class

```
@FixedLengthRecord(hasHeader = true, hasFooter = true)
public class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, length = 9)
    private String firstName;

    @DataField(pos = 4, length = 5, align = "L")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;

    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;
```

```
    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}

@FixedLengthRecord(isHeader = true)
public  class OrderHeader {
    @DataField(pos = 1, length = 1)
    private int recordType = 1;

    @DataField(pos = 2, length = 10, pattern = "dd-MM-yyyy")
    private Date recordDate;
}

@FixedLengthRecord(isFooter = true)
public class OrderFooter {

    @DataField(pos = 1, length = 1)
    private int recordType = 9;

    @DataField(pos = 2, length = 9, align = "R", paddingChar = '0')
    private int numberOfRecordsInTheFile;
}
```

ケース 7: 固定長レコードの解析時にコンテンツをスキップする (Camel 2.11.1)

ターゲットユースケースに必要以上の情報を含む、固定長レコードを提供するシステムと統合するのが一般的です。この状況では、不要なフィールドの宣言と解析をスキップすると便利です。これに対応するために、次に宣言されたフィールドの pos 値が最後に解析されたフィールドのカーソル位置を超えている場合には、Bindy により、レコード内で次にマップされたフィールドに移動されます。対象のフィールドに (順序値ではなく) 絶対 pos 位置を使用すると、Bindy は 2 つのフィールド間のコンテンツをスキップします。

同様に、先のフィールドに対象となるコンテンツがない場合があります。この場合には、@FixedLengthRecord 宣言で **ignoreTrailingChars** プロパティーを設定して、最後にマップされたフィールド以降のすべての解析をスキップするように Bindy に指示できます。

```
@FixedLengthRecord(ignoreTrailingChars = true)
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    // any characters that appear beyond the last mapped field will be ignored

}
```

46.7. 5.メッセージ

Message アノテーションは、キーと値のペアフィールドを含むモデルのクラスを識別するために使用されます。この種の形式は、主に Financial Exchange Protocol Messages (FIX) で使用されます。いずれの場合も、このアノテーションは、データがキーによって識別される他の形式に使用できます。キーペ

アの値は、区切り記号で互いに区切られます。区切り文字には、タブ区切り記号 (Unicode 表現: \u0009) または見出しの開始 (Unicode 表現: \u0001) などの特殊文字を使用できます。

> *"FIX information"*

FIX の詳細については、Web サイト http://www.fixprotocol.org/ を参照してください。FIX メッセージを操作するには、モデルに、Order クラスである可能性があるルートメッセージクラスにリンクされた Header クラスと Trailer クラスが含まれている必要があります。これは必須ではありませんが、camel-bindy を、quickFix プロジェクト (http://www.quickfixj.org/) に基づく修正ゲートウェイである camel-fix と組み合わせて使用する場合に非常に役立ちます。

アノテーション名	レコードの種類	レベル
メッセージ	キーと値のペア。	Class

パラメーター名	type	Info
pairSeparator	string	必須 - '=' または ';' または 'anything'
keyValuePairSeparair	string	必須 - '\u0001、'\u0009'、'#' または 'anything'
crlf	string	オプション - 可能な値 = WINDOWS、UNIX、MAC、またはカスタム。デフォルト値 = WINDOWS - 使用する復帰文字を定義できます。上記の 3 つ以外の値を指定すると、入力した値 (カスタム) が CRLF 文字として使用されます
type	string	オプション - メッセージのタイプを定義します (FIX、EMX など)。
version	string	オプション - メッセージのバージョン (例: 4.1)
isOrdered	boolean	オプション - デフォルト値 = false - FIX メッセージの生成時にフィールドの順序を変更できます。このアノテーションは、モデルのメッセージクラスに関連付けられており、一度宣言する必要があります。

ケース 1: separator = 'u0001'

FIX メッセージでキーと値のペアフィールドを分離するために使用される区切り文字は、ASCII 01 文字または Unicode 形式の \u0001 です。Java ランタイムエラーを回避するには、この文字をもう一度エスケープする必要があります。以下は例です。

> 8=FIX.4.1 9=20 34=1 35=0 49=INVMGR 56=BRKR 1=BE.CHM.001 11=CHM0001-01
> 22=4 ...

およびアノテーションの使用方法

FIX - メッセージ

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

}
```

> *Look at test cases*

タブなどの ASCII 文字は、WIKI ページでは表示できません。そのため、camel-bindy のテストケースを見て、FIX メッセージ (src\test\data\fix\fix.txt) と Order、Trailer、Header クラス (src\test\java\) がどのように見えるかを正確に確認してください。
org\apache\camel\dataformat\bindy\model\fix\simple\Order.java)

46.8. 6.KEYVALUEPAIRFIELD

KeyValuePairField アノテーションは、キーと値のペアフィールドのプロパティーを定義します。各 KeyValuePairField は、タグ (= キー) とそれに関連付けられた値、タイプ (文字列、整数、日付など)、オプションのパターン、およびフィールドが必須かどうかによって識別されます

アノテーション名	レコードの種類	レベル
KeyValuePairField	キーと値のペア - FIX	プロパティー

パラメーター名	type	Info
tag	int	必須 - メッセージ内のフィールドを識別する数字 - 一意でなければなりません
pattern	string	オプション - デフォルト値 = "" - Decimal、Date などのフォーマットに使用されます
精度	int	オプション - 桁数 - 10 進数がフォーマット/解析されるときに使用される精度を表します
position	int	オプション - FIX メッセージ内のキー/タグの位置が異なる必要がある場合に使用する必要があります

パラメーター名	type	Info
必須	boolean	optional - default value = "false"
impliedDecimalSeparator	boolean	**Camel 2.11:** オプション - デフォルト値 = "false" - 指定された位置に暗黙の小数点があるかどうかを示します

case 1 : tag

このパラメーターは、メッセージ内のフィールドのキーを表します

FIX メッセージ - タグ

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String Account;

    @KeyValuePairField(tag = 11) // Order reference
    private String ClOrdId;

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String IDSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String SecurityId;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String Side;

    @KeyValuePairField(tag = 58) // Free text
    private String Text;
}
```

ケース 2: 出力での別の位置

FIX メッセージに入れるタグ/キーを事前定義された順序に従ってソートする必要がある場合は、アノテーション @KeyValuePairField の属性 position を使用します。

FIX メッセージ - タグ - 並べ替え

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version = "4.1",
isOrdered = true)
```

```
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
}
```

46.9. 7.セクション

固定長レコードの FIX メッセージでは、情報の表現にさまざまなセクション (ヘッダー、ボディ、およびセクション) があるのが一般的です。アノテーション @Section の目的は、モデルのどのクラスがヘッダー (= セクション 1)、ボディー (= セクション 2)、およびフッター (= セクション 3) を表しているかを bindy に通知することです。

このアノテーションには、1 つの属性/パラメーターのみが存在します。

アノテーション名	レコードの種類	レベル
セクション	FIX	Class

パラメーター名	type	Info
number	int	セクション位置を識別する桁数

ケース 1: セクション

ヘッダーセクションの定義

FIX メッセージ - セクション - ヘッダー

```
@Section(number = 1)
public class Header {

    @KeyValuePairField(tag = 8, position = 1) // Message Header
    private String beginString;

    @KeyValuePairField(tag = 9, position = 2) // Checksum
    private int bodyLength;
}
```

ボディーセクションの定義

FIX メッセージ - セクション - 本文

```
@Section(number = 2)
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version = "4.1",
isOrdered = true)
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
```

フッターセクションの定義:

FIX メッセージ - セクション - フッター

```
@Section(number = 3)
public class Trailer {

    @KeyValuePairField(tag = 10, position = 1)
    // CheckSum
    private int checkSum;

    public int getCheckSum() {
        return checkSum;
    }
```

46.10. 8.ONETOMANY

アノテーション @OneToMany の目的は、POJO クラスで定義された **List<?>** フィールドを操作できるようにすること、または反復グループを含むレコードから操作できるようにすることです。

Restrictions OneToMany

bindy の 1 対多では、階層の複数のレベルで定義された繰り返しを処理できないことに注意してください。

OneToMany の関係は、次の場合にのみ機能します。

- 反復グループ (= タグ/キーのグループ) を含む FIX メッセージの読み取り
- 反復データを含む CSV の生成

アノテーション名	レコードの種類	レベル
OneToMany	all	プロパティー

パラメーター名	type	Info
mappedTo	string	オプション - 文字列 - List<Type of the Class> の型に関連付けられたクラス名

ケース 1: 反復データで CSV を生成する

必要な CSV 出力は次のとおりです。

```
Claus,Ibsen,Camel in Action 1,2010,35
Claus,Ibsen,Camel in Action 2,2012,35
Claus,Ibsen,Camel in Action 3,2013,35
Claus,Ibsen,Camel in Action 4,2014,35
```

備考 : 反復データは本のタイトルとその発行日に関するもので、姓名と年齢が一般的です。

そして、これをモデル化するために使用されるクラス。Author クラスには、書籍リストが含まれています。

反復データを含む CSV を生成する

```
@CsvRecord(separator=",")
public class Author {

    @DataField(pos = 1)
    private String firstName;

    @DataField(pos = 2)
    private String lastName;

    @OneToMany
    private List<Book> books;

    @DataField(pos = 5)
    private String Age;
}

public class Book {

    @DataField(pos = 3)
    private String title;
```

```
    @DataField(pos = 4)
    private String year;
}
```

とてもシンプルですね!!!

ケース 2: タグ/キーのグループを含む FIX メッセージの読み取り

モデルで処理するメッセージは次のとおりです。

```
8=FIX 4.19=2034=135=049=INVMGR56=BRKR
1=BE.CHM.00111=CHM0001-0158=this is a camel - bindy test
22=448=BE000124567854=1
22=548=BE000987654354=2
22=648=BE000999999954=3
10=220
```

タグ 22、48、および 54 が繰り返されます

およびコード

タグ/キーのグループを含む FIX メッセージの読み取り

```
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11) // Order reference
    private String clOrdId;

    @KeyValuePairField(tag = 58) // Free text
    private String text;

    @OneToMany(mappedTo =
"org.apache.camel.dataformat.bindy.model.fix.complex.onetomany.Security")
    List<Security> securities;
}

public class Security {

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String idSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String securityCode;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String side;
}
```

46.11. 9.BINDYCONVERTER

アノテーション @BindyConverter の目的は、フィールドレベルで使用されるコンバーターを定義することです。提供されたクラスは、Format インターフェイスを実装する必要があります。

```
@FixedLengthRecord(length = 10, paddingChar = ' ')
public static class DataModel {
    @DataField(pos =  1, length = 10, trim = true)
    @BindyConverter(CustomConverter.class)
    public String field1;
}

public static class CustomConverter implements Format<String> {
    @Override
    public String format(String object) throws Exception {
        return (new StringBuilder(object)).reverse().toString();
    }

    @Override
    public String parse(String string) throws Exception {
        return (new StringBuilder(string)).reverse().toString();
    }
}
```

46.12. 10.FORMATFACTORIES

アノテーション @FormatFactories の目的は、一連のコンバーターをレコードレベルで定義することです。提供されるクラスは、FormatFactoryInterface インターフェイスを実装する必要があります。

```
@CsvRecord(separator = ",")
@FormatFactories({OrderNumberFormatFactory.class})
public static class Order {

    @DataField(pos = 1)
    private OrderNumber orderNr;

    @DataField(pos = 2)
    private String firstName;
}

public static class OrderNumber {
    private int orderNr;

    public static OrderNumber ofString(String orderNumber) {
        OrderNumber result = new OrderNumber();
        result.orderNr = Integer.valueOf(orderNumber);
        return result;
    }
}

public static class OrderNumberFormatFactory extends AbstractFormatFactory {

    {
        supportedClasses.add(OrderNumber.class);
```

```
    }

    @Override
    public Format<?> build(FormattingOptions formattingOptions) {
        return new Format<OrderNumber>() {
            @Override
            public String format(OrderNumber object) throws Exception {
                return String.valueOf(object.orderNr);
            }

            @Override
            public OrderNumber parse(String string) throws Exception {
                return OrderNumber.ofString(string);
            }
        };
    }
}
```

46.13. サポートされるデータタイプ

DefaultFormatFactory は、提供された FormattingOptions に基づいてインターフェイス FormatFactoryInterface のインスタンスを返して、次のデータ型のフォーマットを使用できるようにします。

- BigDecimal
- BigInteger
- Boolean
- Byte
- Character
- 日付
- double
- Enum
- Float
- Integer
- LocalDate (Java 8、2.18.0 以降)
- LocalDateTime (java 8, since 2.18.0)
- LocalTime (java 8, since 2.18.0)
- Long
- Short
- String

DefaultFormatFactory は、使用中のレジストリー (Spring または JNDI など) で FactoryRegistry のインスタンスを提供してオーバーライドできます。

46.14. JAVA DSL を使用

次のステップでは、このレコードタイプに関連付けられた DataFormat **bindy** クラスをインスタンス化し、Java パッケージ名をパラメーターとして提供します。

たとえば、次の例では、**com.acme.model** パッケージ名で設定されたクラス **BindyCsvDataFormat** (CSV レコードタイプに関連付けられたクラスに対応) を使用して、このパッケージで設定されたモデルオブジェクトを初期化します。

```
// Camel 2.15 or older (configure by package name)
DataFormat bindy = new BindyCsvDataFormat("com.acme.model");


// Camel 2.16 onwards (configure by class name)
DataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);
```

46.14.1. ロケールの設定

Bindy は、次のようなデータ形式でのロケールの設定をサポートしています。

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("us");
```

または、プラットフォームのデフォルトロケールを使用するには、ロケール名として default を使用します。これには Camel 2.14/2.13.3/2.12.5 が必要であることに注意してください。

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("default");
```

古いリリースの場合は、示されているように Java コードを使用して設定できます

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale(Locale.getDefault().getISO3Country());
```

46.14.2. アンマーシャリング

```
from("file://inbox")
  .unmarshal(bindy)
  .to("direct:handleOrders");
```

または、Spring XML ファイルなどのレジストリーで定義できるデータ形式への名前付き参照を使用できます。

```
from("file://inbox")
  .unmarshal("myBindyDataFormat")
  .to("direct:handleOrders");
```

Camel ルートは、受信トレイディレクトリー内のファイルをピックアップし、CSV レコードをモデルオブジェクトのコレクションにアンマーシャリングして、handleOrders によって参照されるルートにコレクション
を送信します。

返されるコレクションは、**List of Map** オブジェクト です。リスト内の各 Map には、CSV の各行からマーシャリングされたモデルオブジェクトが含まれています。理由は、**各行が複数のオブジェクトに対応している可能性** があるためです。1 行にオブジェクト 1 つだけが返されることを想定している場合には、混乱を招く可能性があります。

各オブジェクトは、そのクラス名を使用して取得できます。

```
List<Map<String, Object>> unmarshaledModels = (List<Map<String, Object>>)
exchange.getIn().getBody();

int modelCount = 0;
for (Map<String, Object> model : unmarshaledModels) {
  for (String className : model.keySet()) {
     Object obj = model.get(className);
     LOG.info("Count : " + modelCount + ", " + obj.toString());
  }
 modelCount++;
}

LOG.info("Total CSV records received by the csv bean : " + modelCount);
```

ルート内で処理するためにこのマップから単一の Order オブジェクトを抽出すると仮定する場合に、次のようにスプリッターとプロセッサーの組み合わせを使用できます。

```
from("file://inbox")
    .unmarshal(bindy)
    .split(body())
        .process(new Processor() {
            public void process(Exchange exchange) throws Exception {
                Message in = exchange.getIn();
                Map<String, Object> modelMap = (Map<String, Object>) in.getBody();
                in.setBody(modelMap.get(Order.class.getCanonicalName()));
            }
        })
        .to("direct:handleSingleOrder")
    .end();
```

Bindy は CHARSET_NAME プロパティーまたは CHARSET_NAME ヘッダーを Exchange インターフェイスで定義されているように使用して、アンマーシャリングのために受信した入力ストリームの文字

セット変換を行うことに注意してください。一部のプロデューサ (file-endpoint など) では、文字セットを定義できます。文字セット変換は、このプロデューサによってすでに実行されている可能性があります。アンマーシャルに送信する前に、エクスチェンジからこのプロパティーまたはヘッダーを削除する必要がある場合があります。削除しないと、変換が 2 回行われ、想定外の結果が生じる可能性があります。

```
from("file://inbox?charset=Cp922")
  .removeProperty(Exchange.CHARSET_NAME)
  .unmarshal("myBindyDataFormat")
  .to("direct:handleOrders");
```

46.14.3. マーシャリング

モデルオブジェクトのコレクションから CSV レコードを生成するには、次のルートを作成します。

```
from("direct:handleOrders")
   .marshal(bindy)
   .to("file://outbox")
```

46.15. SPRING XML の使用

これにより、お気に入りの DSL 言語として Spring を使用して、非常に簡単に camel-bindy に使用するルートを宣言できます。次の例は、最初のルートがファイルからレコードを取得し、コンテンツをアンマーシャリングしてモデルにバインドする 2 つのルートを示しています。結果は pojo に送信され (特別なことは何も行われません)、キューに入れられます。

2 番目のルートは、キューから pojo を抽出し、コンテンツをマーシャリングして、csv レコードを含むファイルを生成します。上記の例は、Camel 2.16 以降を使用するためのものです。

spring dsl

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring
      http://camel.apache.org/schema/spring/camel-spring.xsd">

   <!-- Queuing engine - ActiveMq - work locally in mode virtual memory -->
   <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
      <property name="brokerURL" value="vm://localhost:61616"/>
   </bean>

   <camelContext xmlns="http://camel.apache.org/schema/spring">
      <dataFormats>
        <bindy id="bindyDataformat" type="Csv" classType="org.apache.camel.bindy.model.Order"/>
      </dataFormats>

      <route>
         <from uri="file://src/data/csv/?noop=true" />
         <unmarshal ref="bindyDataformat" />
```

```
            <to uri="bean:csv" />
            <to uri="activemq:queue:in" />
        </route>

        <route>
            <from uri="activemq:queue:in" />
            <marshal ref="bindyDataformat" />
            <to uri="file://src/data/csv/out/" />
        </route>
    </camelContext>
</beans>
```

注記

モデルクラスが serializable を実装していることを確認してください。実装されていない場合には、キューマネージャーでエラーが発生します

46.16. 依存関係

camel ルートで Bindy を使用するには、このデータ形式を実装する **camel-bindy** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bindy</artifactId>
  <version>x.x.x</version>
</dependency>
```

第47章 CAMEL で OSGI ブループリントを使用する

ブループリント用のカスタム XML 名前空間が作成され、優れた XML ダイアレクトを利用できるようになりました。ブループリントのカスタム名前空間はまだ標準化されていないため、この名前空間は、Apache Karaf で使用される Apache Aries ブループリントの実装でのみ使用できます。

47.1. 概要

XML スキーマは Spring のものとほとんど同じであるため、ドキュメント全体で Spring XML を参照しているすべての xml スニペットは Blueprint ルートにも適用されます。

ブループリントを使用した非常に単純なルート定義を次に示します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route>
      <from uri="timer:test" />
      <to uri="log:test" />
    </route>
  </camelContext>

</blueprint>
```

この時点で、サポートされている xml 要素についていくつかの制限があります (Spring xml 構文と比較して)。

- beanPostProcessor は Spring に固有であり、許可されていません

ただし、アプリケーションを OSGi 環境にデプロイするときにブループリントを使用すると、いくつかの利点があります。

- 新しい camel バージョンにアップグレードする場合、名前空間を変更する必要はありません。バンドルによってインポートされた camel パッケージに基づいて正しいバージョンが選択されるためです。
- カスタム名前空間とバンドルに関する起動順序の問題はありません
- Blueprint プロパティープレースホルダーを使用できます

47.2. CAMEL-BLUEPRINT の使用

OSGi で camel-blueprint を活用するには、camel-core とその依存関係に加えて、Aries Blueprint バンドルと camel-blueprint バンドルのみが必要です。

Karaf を使用する場合は、必要なすべてのバンドルをインストールする camel-blueprint という名前の機能を使用できます。

第48章 BONITA コンポーネント

Camel バージョン 2.19 以降で利用可能

リモートの Bonita BPM プロセスエンジンとの通信に使用されます。

48.1. URI 形式

```
bonita:///[operation]?[options]
```

operation は、Bonita で実行する特定のアクションです。

48.2. 一般的なオプション

Bonita コンポーネントにはオプションがありません。

Bonita エンドポイントは、URI 構文を使用して設定されます。

```
bonita:operation
```

パスおよびクエリーパラメーターを使用します。

48.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	**必須** 必要な操作		BonitaOperation

48.2.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
hostname (consumer)	Bonita エンジンが実行されるホスト名	localhost	String
port (consumer)	Bonita エンジンをホストするサーバーのポート	8080	String

名前	説明	デフォルト	タイプ
processName (consumer)	操作に関係するプロセスの名前		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
password (security)	Bonita エンジンを認証するためのパスワード。		String
username (security)	Bonita エンジンを認証するためのユーザー名。		文字列

48.3. ボディーコンテンツ

startCase 操作の場合、入力変数はボディーメッセージから取得されます。これには Map<String,Serializable> が含まれている必要があります。

48.4. 例

次の例では、Bonita で新しいケースを開始します。

```
from("direct:start").to("bonita:startCase?
hostname=localhost&amp;port=8080&amp;processName=TestProcess&amp;username=install&amp;pa
ssword=install")
```

48.5. 依存関係

Camel ルートで Bonita を使用するには、コンポーネントを実装する **camel-bonita** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
```

```
  <artifactId>camel-bonita</artifactId>
  <version>x.x.x</version>
</dependency>
```

第49章 BOON DATAFORMAT

Camel バージョン 2.16 以降で利用可能

Boon は、Boon JSON マーシャリングライブラリーを使用して JSON ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを JSON ペイロードにマーシャリングするデータ形式です。Boon は、現在使用されている他の一般的なパーサーよりもシンプルで 高速なパーサー を目指しています。

49.1. オプション

Boon データ形式は、以下にリストされている 3 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
unmarshalTypeName		**String**	アンアームシャリング時に使用する Java 型のクラス名
useList	**false**	**Boolean**	Map の List または Pojo の List にアンマーシャリングします。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

49.2. JAVA DSL を使用

```
DataFormat boonDataFormat = new BoonDataFormat("com.acme.model.Person");

from("activemq:My.Queue")
  .unmarshal(boonDataFormat)
  .to("mqseries:Another.Queue");
```

49.3. ブループリント XML の使用

```
<bean id="boonDataFormat" class="org.apache.camel.component.boon.BoonDataFormat">
  <argument value="com.acme.model.Person"/>
</bean>

<camelContext id="camel" xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="activemq:My.Queue"/>
    <unmarshal ref="boonDataFormat"/>
    <to uri="mqseries:Another.Queue"/>
  </route>
</camelContext>
```

49.4. 依存関係

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-boon</artifactId>
  <version>x.x.x</version>
</dependency>
```

第50章 BOX コンポーネント

Camel バージョン 2.14 以降で利用可能

Box コンポーネントは、https://github.com/box/box-java-sdk を使用してアクセスできるすべての Box.com API へのアクセスを提供します。ファイルのアップロードとダウンロード、フォルダーの作成、編集、管理などのメッセージを作成できます。また、ユーザーアカウントの更新やエンタープライズアカウントの変更などをポーリングできる API もサポートしています。

Box.com では、すべてのクライアントアプリケーション認証に OAuth2.0 を使用する必要があります。アカウントで camel-box を使用するには、https://developer.box.com の Box.com 内で新しいアプリケーションを作成する必要があります。Box アプリケーションのクライアント ID とシークレットは、現在のユーザーを必要とする Box API へのアクセスを許可します。ユーザーアクセストークンは、エンドユーザーの API によって生成および管理されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-box</artifactId>
  <version>${camel-version}</version>
</dependency>
```

50.1. 接続認証の種類

Box コンポーネントは、3 種類の認証済み接続をサポートしています。

50.1.1. 標準認証

標準認証 は、**OAuth 2.0 three-legged 認証プロセス** を使用して、Box.com との接続を認証します。このタイプの認証により、Box **管理対象ユーザー** と **外部ユーザー** は、Box コンポーネントを介して Box コンテンツにアクセスし、編集し、保存することができます。

50.1.2. アプリケーションエンタープライズ認証

App Enterprise Authentication は、**OAuth 2.0 と JSON Web トークン (JWT)**を使用して、**Box アプリケーション** の **サービスアカウント** として接続を認証します。このタイプの認証により、サービスアカウントは、Box コンポーネントを介して **Box アプリケーション** の Box コンテンツにアクセスし、編集し、保存することができます。

50.1.3. アプリユーザー認証

アプリケーションユーザー認証 は、**OAuth 2.0 と JSON Web トークン (JWT)**を使用して、**Box アプリケーション** の **アプリユーザー** としての接続を認証します。このタイプの認証により、アプリユーザーは、Box コンポーネントを介して **Box アプリケーション** で Box コンテンツにアクセスし、編集し、保存することができます。

50.2. BOX のオプション

Box コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		BoxConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Box エンドポイントは、URI 構文を使用して設定されます。

```
box:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

50.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		BoxApiName
methodName	**必須**: 選択した操作に使用するサブ操作		String

50.2.2. クエリーパラメーター (20 パラメーター)

名前	説明	デフォルト	タイプ
clientId (common)	Box アプリケーションのクライアント ID		String
enterpriseId (common)	App Enterprise に使用するエンタープライズ ID。		String
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
userId (common)	アプリユーザーに使用するユーザー ID。		String

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpParams (advanced)	プロキシーホストなどの設定用のカスタム HTTP パラメーター		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accessTokenCache (security)	アクセストークンを保存および取得するためのカスタムアクセストークンキャッシュ。		IAccessTokenCache
clientSecret (security)	Box アプリケーションのクライアントシークレット		String
encryptionAlgorithm (security)	JWT の暗号化アルゴリズムのタイプ。サポートされているアルゴリズム: RSA_SHA_256 RSA_SHA_384 RSA_SHA_512	RSA_SHA_256	EncryptionAlgorithm
maxCacheEntries (security)	キャッシュ内のアクセストークンの最大数。	100	int
authenticationType (authentication)	接続の認証の種類。認証の種類: STANDARD_AUTHENTICATION - OAuth 2.0 (3-legged) SERVER_AUTHENTICATION - JSON Web トークンを使用した OAuth 2.0	APP_USER_AUTHENTICATION	String

名前	説明	デフォルト	タイプ
privateKeyFile (security)	JWT 署名を生成するための秘密鍵。		String
privateKeyPassword (security)	秘密鍵のパスワード。		String
publicKeyId (security)	JWT 署名を検証するための公開鍵の ID。		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
userName (security)	ボックスのユーザー名を指定する必要があります		String
userPassword (security)	Box ユーザーのパスワード。authSecureStorage が設定されていない場合は指定する必要があり、最初の呼び出しで null を返す		文字列

50.3. URI 形式

```
box:apiName/methodName
```

apiName は次のいずれかです。

- コラボレーション
- コメント
- event-logs
- files
- folders
- groups
- events
- search
- tasks
- users

50.4. プロデューサーエンドポイント:

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で示されます。エンドポイントに必須のオプションがない場合、[] オプションのセットの 1 つを提供する必要があります。プロデューサエンドポイントは、特別なオプション **inBody** を 使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelBox.<option>** の形式である必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelBox.option** ヘッダーをオーバーライドします。

エンドポイント URI またはメッセージヘッダーのオプション **defaultRequest** に値が指定されていない場合は、**null** と見なされます。**null** 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Box API エラーの場合、エンドポイントは **com.box.sdk.BoxAPIException** から生成された例外が原因で RuntimeCamelException を出力します。

50.4.1. エンドポイント接頭辞 collaborations

Box コラボレーションの詳細については、https://developer.box.com/reference#collaboration-object を参照してください。 次のエンドポイントは、次のように接頭辞 **collaborations** を使用して呼び出すことができます。

```
box:collaborations/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addFolderCollaboration	add	folderId、コラボレーター、ロール	com.box.sdk.BoxCollaboration
addFolderCollaborationByEmail	addByEmail	folderId、電子メール、ロール	com.box.sdk.BoxCollaboration
deleteCollaboration	delete	collaborationId	

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getFolderCollaborations	コラボレーション	folderId	java.util.Collection
getPendingCollaborations	pendingCollaborations		java.util.Collection
getCollaborationInfo	info	collaborationId	com.box.sdk.BoxCollaboration.Info
updateCollaborationInfo	updateInfo	collaborationId, info	com.box.sdk.BoxCollaboration

コラボレーション の URI オプション

名前	タイプ
collaborationId	String
コラボレーター	com.box.sdk.BoxCollaborator
role	com.box.sdk.BoxCollaboration.Role
folderId	String
email	String
info	com.box.sdk.BoxCollaboration.Info

50.4.2. エンドポイント接頭辞 comments

Box コメントの詳細については、https://developer.box.com/reference#comment-object を参照してください。 次のエンドポイントは、次のように接頭辞 **comments** を使用して呼び出すことができます。

```
box:comments/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addFileComment	add	fileId, message	com.box.sdk.BoxFile
changeCommentMessage	updateMessage	commentId, message	com.box.sdk.BoxComment
deleteComment	delete	commentId	
getCommentInfo	info	commentId	com.box.sdk.BoxComment.Info
getFileComments	コメント	fileId	java.util.List
replyToComment	応答	commentId, message	com.box.sdk.BoxComment

コラボレーション の URI オプション

名前	タイプ
commentId	String
fileId	String
message	String

50.4.3. エンドポイント接頭辞 events-logs

Box イベントログの詳細については、https://developer.box.com/reference#events を参照してください。次のエンドポイントは、次のように接頭辞 **events** で呼び出すことができます。

```
box:event-logs/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getEnterpriseEvents	events	position, after, before, [types]	java.util.List

イベントログ の URI オプション

名前	タイプ
position	String
after	Date
before	Date
types	com.box.sdk.BoxEvent.Types[]

50.4.4. エンドポイント接頭辞 files

Box ファイルの詳細については、https://developer.box.com/reference#file-object を参照してください。次のエンドポイントは、次のように接頭辞 **files** を使用して呼び出すことができます。

```
box:files/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
uploadFile	upload	parentFolderId, content, fileName, [created], [modified], [size], [listener]	com.box.sdk.BoxFile
downloadFile	ダウンロード	fileId, output, [rangeStart], [rangeEnd], [listener]	java.io.OutputStream
copyFile	copy	fileId, destinationFolderId, [newName]	com.box.sdk.BoxFile
moveFile	move	fileId, destinationFolderId, [newName]	com.box.sdk.BoxFile
renameFile	rename	fileId, newFileName	com.box.sdk.BoxFile
createFileSharedLink	リンク	fileId, access, [unshareDate], [permissions]	com.box.sdk.BoxSharedLink

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
deleteFile	delete	fileId	
uploadNewFileVersion	uploadVersion	fileId, fileContent, [modified], [fileSize], [listener]	com.box.boxsdk.BoxFile
promoteFileVersion	promoteVersion	fileId, version	com.box.sdk.BoxFileVersion
getFileVersions	versions	fileId	java.util.Collection
downloadPreviousFileVersions	downloadVersion	fileId, version, output, [listener]	java.io.OutputStream
deleteFileVersion	deleteVersion	fileId, version	
getFileInfo	info	fileId, fields	com.box.sdk.BoxFile.Info
updateFileInfo	updateInfo	fileId, info	com.box.sdk.BoxFile
createFileMetadata	createMetadata	fileId, metadata, [typeName]	com.box.sdk.Metadata

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getFileMetadata	metadata	fileId, [typeName]	com.box.sdk.Metadata
updateFileMetadata	updateMetadata	fileId, metadata	com.box.sdk.Metadata
deleteFileMetadata	deleteMetadata	fileId	
getDownloadUrl	url	fileId	java.net.URL
getPreviewLink	preview	fileId	java.net.URL
getFileThumbnail	thumbnail	fileId, fileType, minWidth, minHeight, maxWidth, maxHeight	byte[]

ファイル の URI オプション

名前	タイプ
parentFolderId	String
content	java.io.InputStream
fileName	String

名前	タイプ
created	Date
modified	Date
size	Long
listener	com.box.sdk.ProgressListener
output	java.io.OutputStream
rangeStart	Long
rangeEnd	Long
outputStreams	java.io.OutputStream[]
destinationFolderId	String
newName	String
fields	String[]
info	com.box.sdk.BoxFile.Info
fileSize	Long
version	Integer
access	com.box.sdk.BoxSharedLink.Access
unshareDate	Date
permissions	com.box.sdk.BoxSharedLink.Permissions

名前	タイプ
fileType	com.box.sdk.BoxFile.ThumbnailFileType
minWidth	Integer
minHeight	Integer
maxWidth	Integer
maxHeight	Integer
metadata	com.box.sdk.Metadata
typeName	String

50.4.5. エンドポイント接頭辞 folders

Box フォルダーの詳細については、https://developer.box.com/reference#folder-object を参照してください。次のエンドポイントは、次のように接頭辞 **folders** を使用して呼び出すことができます。

```
box:folders/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getRootFolder	root		com.box.sdk.BoxFolder
createFolder	create	parentFolderId, folderName	com.box.sdk.BoxFolder
createFolder	create	parentFolderId, path	com.box.sdk.BoxFolder

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
copyFolder	copy	folderId, destinationfolderId, [newName]	com.box.sdk.BoxFolder
moveFolder	move	folderId, destinationFolderId, newName	com.box.sdk.BoxFolder
renameFolder	rename	folderId, newFolderName	com.box.sdk.BoxFolder
createFolderSharedLink	リンク	folderId, access, [unsharedDate], [permissions]	java.util.List
deleteFolder	delete	folderId	
getFolder	folder	path	com.box.sdk.BoxFolder
getFolderInfo	info	folderId, fields	com.box.sdk.BoxFolder.Info
getFolderItems	items	folderId, offset, limit, fields	com.box.sdk.BoxFolder

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
updateFolderInfo	updateInfo	folderId, info	com.box.sdk.BoxFolder

folders の URI オプション

名前	タイプ
path	String[]
folderId	String
offset	Long
limit	Long
fields	String[]
parentFolderId	String
folderName	String
destinationFolderId	String
newName	String
newFolderName	String
info	String
access	com.box.sdk.BoxSharedLink.Access
unshareDate	Date

名前	タイプ
permissions	com.box.sdk.BoxSharedLink.Permissions

50.4.6. エンドポイント接頭辞 groups

Box グループの詳細については、https://developer.box.com/reference#group-object を参照してください。次のエンドポイントは、次のように接頭辞 **groups** を使用して呼び出すことができます。

```
box:groups/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
createGroup	create	name, [provenance, externalSyncIdentifier, description, invitabilityLevel, memberViewabilityLevel]	com.box.sdk.BoxGroup
addGroupMembership	createMembership	groupId, userId, role	com.box.sdk.BoxGroupMembership
deleteGroup	delete	groupId	
getAllGroups	groups		java.util.Collection
getGroupInfo	info	groupId	com.box.sdk.BoxGroup.Info

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
updateGroupInfo	updateInfo	groupId、groupInfo	com.box.sdk.BoxGroup
addGroupMembership	addMembership	groupId, userId, role	com.box.sdk.BoxGroupMembership
deleteGroupMembership	deleteMembership	groupMembershipId	
getGroupMemberships	memberships	groupId	java.uti.Collection
getGroupMembershipInfo	membershipInfo	groupMemebershipId	com.box.sdk.BoxGroup.Info
updateGroupMembershipInfo	updateMembershipInfo	groupMembershipId、info	com.box.sdk.BoxGroupMembership

groups の URI オプション

名前	タイプ
name	String
groupId	String
userId	String
role	com.box.sdk.BoxGroupMembership.Role

名前	タイプ
groupMembershipId	String
info	com.box.sdk.BoxGroupMembership.Info

50.4.7. エンドポイント接頭辞 search

Box 検索 API の詳細については、https://developer.box.com/reference#searching-for-content を参照してください。 次のエンドポイントは、次のように接頭辞 **search** で呼び出すことができます。

```
box:search/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
searchFolder	search	folderId, query	java.util.Collection

search の URI オプション

名前	タイプ
folderId	String
query	String

50.4.8. エンドポイント接頭辞 tasks

Box タスクの詳細については、https://developer.box.com/reference#task-object-1 を参照してください。次のエンドポイントは、次のように接頭辞 **tasks** を使用して呼び出すことができます。

```
box:tasks/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addFileTask	add	fileId, action, dueAt, [message]	com.box.sdk.BoxUser
deleteTask	delete	taskId	
getFileTasks	tasks	fileId	java.util.List
getTaskInfo	info	taskId	com.box.sdk.BoxTask.Info
updateTaskInfo	updateInfo	taskId, info	com.box.sdk.BoxTask
addAssignmentToTask	addAssignment	taskId, assignTo	com.box.sdk.BoxTask
deleteTaskAssignment	deleteAssignment	taskAssignmentId	
getTaskAssignments	assignments	taskId	java.util.List
getTaskAssignmentInfo	assignmentInfo	taskAssignmentId	com.box.sdk.BoxTaskAssignment.Info

tasks の URI オプション

名前	タイプ
fileId	String

名前	タイプ
action	com.box.sdk.BoxTask.Action
dueAt	Date
message	String
taskId	String
info	com.box.sdk.BoxTask.Info
assignTo	com.box.sdk.BoxUser
taskAssignmentId	String

50.4.9. エンドポイント接頭辞 users

Box ユーザーについては、https://developer.box.com/reference#user-object を参照してください。次のエンドポイントは、次のように接頭辞 **users** で呼び出すことができます。

```
box:users/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getCurrentUser	currentUser		com.box.sdk.BoxUser
getAllEnterpriseOrExternalUsers	users	filterTerm, [fields]	com.box.sdk.BoxUser
createAppUser	create	name, [params]	com.box.sdk.BoxUser
createEnterpriseUser	create	login, name, [params]	com.box.sdk.BoxUser

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
deleteUser	delete	userId, notifyUser, force	
getUserEmailAlias	emailAlias	userId	com.box.sdk.BoxUser
deleteUserEmailAlias	deleteEmailAlias	userId, emailAliasId	java.util.List
getUserInfo	info	userId	com.box.sdk.BoxUser.Info
updateUserInfo	updateInfo	userId, info	com.box.sdk.BoxUser
moveFolderToUser	-	userId, sourceUserId	com.box.sdk.BoxFolder.Info

users の URI オプション

名前	タイプ
defaultRequest	com.box.restclientv2.requestsbase.BoxDefaultRequestObject
emailAliasRequest	com.box.boxjavalibv2.requests.requestobjects.BoxEmailAliasRequestObject
emailId	String
filterTerm	String
folderId	String

名前	タイプ
simpleUserRequest	com.box.boxjavalibv2.requests.requestobjects.BoxSimpleUserRequestObject
userDeleteRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserDeleteRequestObject
userId	String
userRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserRequestObject
userUpdateLoginRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserUpdateLoginRequestObject

50.5. コンシューマーエンドポイント

Box イベントの詳細については、https://developer.box.com/reference#events を参照してください。次の例に示すように、コンシューマーエンドポイントはエンドポイント 接頭辞 **events** のみを使用できます。

```
box:events/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
events		[startingPosition]	com.box.sdk.BoxEvent

events の URI オプション

名前	タイプ
startingPosition	Long

50.6. メッセージヘッダー

CamelBox 接頭辞を使用するプロデューサーエンドポイントのメッセージヘッダーには、任意のオプションを指定できます。

50.7. メッセージボディー

すべての結果メッセージ本文は、Box Java SDK によって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

50.8. サンプル

次のルートは、新しいファイルをユーザーのルートフォルダーにアップロードします。

```
from("file:...")
    .to("box://files/upload/inBody=fileUploadRequest");
```

次のルートは、更新のためにユーザーのアカウントをポーリングします。

```
from("box://events/listen?startingPosition=-1")
    .to("bean:blah");
```

次のルートでは、動的ヘッダーオプションを持つプロデューサーを使用します。 **fileId** プロパティーには Box ファイル ID があり、**output** プロパティーにはファイルコンテンツの出力ストリームがあるため、次のように **CamelBox.fileId** ヘッダーと **CamelBox.output** ヘッダーにそれぞれ割り当てられます。

```
from("direct:foo")
    .setHeader("CamelBox.fileId", header("fileId"))
    .setHeader("CamelBox.output", header("output"))
    .to("box://files/download")
    .to("file://...");
```

第51章 BRAINTREE コンポーネント

Camel バージョン 2.17 以降で利用可能

Braintree コンポーネントは、Java SDK を介して Braintree Payments へのアクセスを提供します。

支払いを処理するには、すべてのクライアントアプリケーションに API 認証情報が必要です。アカウントで camel-braintree を使用するには、新しい Sandbox または Production アカウントを作成する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-braintree</artifactId>
  <version>${camel-version}</version>
</dependency>
```

51.1. BRAINTREE オプション

Braintree コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		BraintreeConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Braintree エンドポイントは、URI 構文を使用して設定されます。

```
braintree:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

51.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		BraintreeApiName
methodName	選択した操作に使用するサブ操作		String

51.1.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
environment (common)	環境 SANDBOX または PRODUCTION のいずれか		String
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
merchantId (common)	Braintree が提供するマーチャント ID。		String
privateKey (common)	Braintree が提供する秘密鍵。		String
publicKey (common)	Braintree が提供する公開鍵。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
accessToken (advanced)	マーチャントに代わってトランザクションを処理するために、マーチャントが別のマーチャントに付与するアクセストークン。環境、マーチャント ID、公開鍵、および秘密鍵フィールドの代わりに使用されます。		String
httpReadTimeout (advanced)	http 呼び出しの読み取りタイムアウトを設定します。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
httpLogLevel (logging)	http 呼び出しのロギングレベルを設定します。java.util.logging.Level を参照してください。		String
proxyHost (proxy)	プロキシーホスト		String
proxyPort (proxy)	プロキシーポート		Integer

51.2. URI 形式

```
braintree://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- アドオン
- address
- clientToken
- creditCardverification
- customer
- discount
- merchantAccount
- paymentmethod
- paymentmethodNonce
- plan
- settlementBatchSummary
- subscription
- transaction
- webhookNotification

51.3. BRAINTREECOMPONENT

Braintree コンポーネントは、以下のオプションで設定できます。これらのオプションは、タイプ **org.apache.camel.component.braintree.BraintreeConfiguration** のコンポーネントの Bean プロパティー **設定** を使用して提供できます。

オプション	タイプ	説明
environment	String	リクエストの送信先を指定する値 - sandbox または production
merchantId	String	マーチャントアカウント ID とは異なる、ゲートウェイアカウントの一意の識別子
publicKey	String	ユーザー固有の公開識別子
privateKey	String	共有してはならないユーザー固有の安全な識別子 - 私たちとさえも!
accessToken	String	Braintree Auth を使用してマーチャントに付与されるトークンで、別のマーチャントに代わってトランザクションを処理できるようになります。環境、merchantId、publicKey、および privateKey オプションの代わりに使用されます。

上記のすべてのオプションは、Braintree Payments によって提供されます。

51.4. プロデューサーエンドポイント:

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションはで示されます。エンドポイントに必須のオプションがない場合、オプションのセットの 1 つを提供する必要があります。プロデューサエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelBraintree.<option>** の形式である必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelBraintree.option** ヘッダーをオーバーライドします。

エンドポイントとオプションの詳細については、https://developers.braintreepayments.com/reference/overview で Braintree のリファレンスを参照してください。

51.4.1. エンドポイント接頭辞 addOn

次のエンドポイントは、次のように接頭辞 **addOn** を使用して呼び出すことができます。

```
braintree://addOn/endpoint
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Addon>

51.4.2. エンドポイント接頭辞 address

次のエンドポイントは、次のように接頭辞 **address** を使用して呼び出すことができます。

```
braintree://address/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
create		customerId, request	com.braintreegateway.Result<com.braintreegateway.Address>
delete		customerId, id	com.braintreegateway.Result<com.braintreegateway.Address>
find		customerId, id	com.braintreegateway.Address
update		customerId, id, request	com.braintreegateway.Result<com.braintreegateway.Address>

address の URI オプション

名前	タイプ
customerId	String
request	com.braintreegateway.AddressRequest
id	String

51.4.3. エンドポイント接頭辞 clientToken

次のエンドポイントは、次のように接頭辞 **clientToken** を使用して呼び出すことができます。

braintree://*clientToken/endpoint?[options]*

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
generate		request	String

clientToken の URI オプション

名前	タイプ
request	com.braintreegateway.ClientTokenrequest

51.4.4. エンドポイント接頭辞 creditCardVerification

次のエンドポイントは、次のように接頭辞 **creditCardverification** で呼び出すことができます。

braintree://*creditCardVerification/endpoint?[options]*

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
find		id	com.braintreegateway.CreditCardVerification
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.CreditCardVerification>

creditCardVerification の URI オプション

名前	タイプ
id	String
query	com.braintreegateway.CreditCardVerificationSearchRequest

51.4.5. エンドポイント接頭辞 customer

次のエンドポイントは、次のように接頭辞 **customer** で呼び出すことができます。

```
braintree://customer/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
all			
create		request	com.braintreegateway.Result<com.braintreegateway.Customer>
delete		id	com.braintreegateway.Result<com.braintreegateway.Customer>
find		id	com.braintreegateway.Customer
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.Customer>
update		id、request	com.braintreegateway.Result<com.braintreegateway.Customer>

customer の URI オプション

名前	タイプ
id	String
request	com.braintreegateway.CustomerRequest
query	com.braintreegateway.CustomerSearchRequest

51.4.6. エンドポイント接頭辞 discount

次のエンドポイントは、次のように接頭辞 **discount** で呼び出すことができます。

```
braintree://discount/endpoint
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Discount>

+

+

51.4.7. エンドポイント接頭辞 merchantAccount

次のエンドポイントは、次のように接頭辞 **MerchantAccount** で呼び出すことができます。

braintree:*//merchantAccount/endpoint?[options]*

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
create		request	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>
createForCurrency		currencyRequest	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>
find		id	com.braintreegateway.MerchantAccount
update		id、request	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>

MerchantAccount の URI オプション

名前	タイプ
id	String
request	com.braintreegateway.MerchantAccountRequest
currencyRequest	com.braintreegateway.MerchantAccountCreateForCurrencyRequest

51.4.8. エンドポイント接頭辞 paymentMethod

次のエンドポイントは、次のように **paymentMethod** 接頭辞を付けて呼び出すことができます。

```
braintree://paymentMethod/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
create		request	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>
delete		token, deleteRequest	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>
find		token	com.braintreegateway.PaymentMethod
update		token, request	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>

paymentMethod の URI オプション

名前	タイプ
token	String
request	com.braintreegateway.PaymentMethodRequest
deleteRequest	com.braintreegateway.PaymentMethodDeleteRequest

51.4.9. エンドポイント接頭辞 paymentMethodNonce

次のエンドポイントは、次のように接頭辞 **paymentMethodNonce** で呼び出すことができます。

```
braintree://paymentMethodNonce/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
create		paymentMethodToken	com.braintreegateway.Result<com.braintreegateway.PaymentMethodNonce>

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
find		paymentMethodNonce	com.braintreegateway.PaymentMethodNonce

paymentMethodNonce の URI オプション

名前	タイプ
paymentMethodToken	String
paymentMethodNonce	String

51.4.10. エンドポイント接頭辞 plan

次のエンドポイントは、次のように接頭辞 **plan** で呼び出すことができます。

```
braintree://plan/endpoint
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Plan>

51.4.11. エンドポイント接頭辞 settlementBatchSummary

次のエンドポイントは、次のように、**separationBatchSummary** 接頭辞を使用して呼び出すことができます。

```
braintree://settlementBatchSummary/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
generate		request	com.braintreegateway.Result<com.braintreegateway.SettlementBatchSummary>

決済 BatchSummary の URI オプション

名前	タイプ
settlementDate	Calendar
groupByCustomField	String

51.4.12. エンドポイント接頭辞 subscription

次のエンドポイントは、次のように接頭辞 **subscription** で呼び出すことができます。

```
braintree://subscription/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
cancel		id	com.braintreegateway.Result<com.braintreegateway.Subscription>
create		request	com.braintreegateway.Result<com.braintreegateway.Subscription>
delete		customerId, id	com.braintreegateway.Result<com.braintreegateway.Subscription>
find		id	com.braintreegateway.Subscription
retryCharge		subscriptionId, amount	com.braintreegateway.Result<com.braintreegateway.Transaction>
search		searchRequest	com.braintreegateway.ResourceCollection<com.braintreegateway.Subscription>
update		id、request	com.braintreegateway.Result<com.braintreegateway.Subscription>

subscription の URI オプション

名前	タイプ
id	String
request	com.braintreegateway.SubscriptionRequest
customerId	String
subscriptionId	String
amount	BigDecimal
searchRequest	com.braintreegateway.SubscriptionSearchRequest.

51.4.13. エンドポイント接頭辞 transaction

次のエンドポイントは、次のように接頭辞 **transaction** で呼び出すことができます。

```
braintree://transaction/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
cancelRelease		id	com.braintreegateway.Result<com.braintreegateway.Transaction>
cloneTransaction		id, cloneRequest	com.braintreegateway.Result<com.braintreegateway.Transaction>
credit		request	com.braintreegateway.Result<com.braintreegateway.Transaction>
find		id	com.braintreegateway.Transaction
holdInEscrow		id	com.braintreegateway.Result<com.braintreegateway.Transaction>

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
releaseFromEscrow		id	com.braintreegateway.Result<com.braintreegateway.Transaction>
refund		id, amount, refundRequest	com.braintreegateway.Result<com.braintreegateway.Transaction>
sale		request	com.braintreegateway.Result<com.braintreegateway.Transaction>
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.Transaction>
submitForPartialSettlement		id, amount	com.braintreegateway.Result<com.braintreegateway.Transaction>
submitForSettlement		id, amount, request	com.braintreegateway.Result<com.braintreegateway.Transaction>
voidTransaction		id	com.braintreegateway.Result<com.braintreegateway.Transaction>

transaction の URI オプション

名前	タイプ
id	String
request	com.braintreegateway.TransactionCloneRequest
cloneRequest	com.braintreegateway.TransactionCloneRequest
refundRequest	com.braintreegateway.TransactionRefundRequest
amount	BigDecimal
query	com.braintreegateway.TransactionSearchRequest

51.4.14. エンドポイント接頭辞 webhookNotification

次のエンドポイントは、次のように接頭辞 **webhookNotification** で呼び出すことができます。

```
braintree://webhookNotification/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
parse		signature, payload	com.braintreegateway.WebhookNotification
verify		challenge	String

webhookNotification の URI オプション

名前	タイプ
signature	String
payload	String
challenge	String

51.5. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Optionsと**consumer.**の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。デフォルトでは、配列またはコレクションを返す Consumer エンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。この動作を変更するには、プロパティー **consumer.splitResults=true** を使用して、リストまたは配列全体の単一のエクスチェンジを返します。

51.6. メッセージヘッダー

プロデューサーエンドポイントのメッセージヘッダーには、**CamelBraintree.**という接頭辞で任意の URI オプションを提供することができます。

51.7. メッセージボディー

すべての結果メッセージボディーは、Braintree Java SDK によって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

51.8. 例

ブループリント

```
<?xml version="1.0"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
          xsi:schemaLocation="
            http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0
http://aries.apache.org/schemas/blueprint-cm/blueprint-cm-1.0.0.xsd
            http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
            http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">


    <cm:property-placeholder id="placeholder" persistent-id="camel.braintree">
    </cm:property-placeholder>

    <bean id="braintree" class="org.apache.camel.component.braintree.BraintreeComponent">
        <property name="configuration">
            <bean class="org.apache.camel.component.braintree.BraintreeConfiguration">
                <property name="environment" value="${environment}"/>
                <property name="merchantId" value="${merchantId}"/>
                <property name="publicKey" value="${publicKey}"/>
                <property name="privateKey" value="${privateKey}"/>
            </bean>
        </property>
    </bean>

    <camelContext trace="true" xmlns="http://camel.apache.org/schema/blueprint" id="braintree-
example-context">
        <route id="braintree-example-route">
            <from uri="direct:generateClientToken"/>
            <to uri="braintree://clientToken/generate"/>
            <to uri="stream:out"/>
        </route>
    </camelContext>

</blueprint>
```

51.9. 関連項目

* Camel の設定 * コンポーネント * エンドポイント * はじめに

第52章 BROWSE コンポーネント

Camel バージョン 1.3 以降で利用可能

Browse コンポーネントは、テスト、視覚化ツール、またはデバッグに役立つシンプルな BrowsableEndpoint を提供します。エンドポイントに送信された交換はすべて閲覧可能です。

52.1. URI 形式

```
browse:someName[?options]
```

someName は、エンドポイントを一意に識別する任意の文字列にすることができます。

52.2. オプション

Browse コンポーネントにはオプションがありません。

Browse エンドポイントは、URI 構文を使用して設定されます。

```
browse:name
```

パスおよびクエリーパラメーターを使用します。

52.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	***必須*** エンドポイントを一意に識別する任意の文字列の名前		String

52.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

52.3. 例

以下のルートでは、通過する Exchange を参照できるように、**browse:** コンポーネントを挿入します。

```
from("activemq:order.in").to("browse:orderReceived").to("bean:processOrder");
```

受信した交換を Java コード内から検査できるようになりました。

```
private CamelContext context;

public void inspectRecievedOrders() {
    BrowsableEndpoint browse = context.getEndpoint("browse:orderReceived",
BrowsableEndpoint.class);
    List<Exchange> exchanges = browse.getExchanges();

    // then we can inspect the list of received exchanges from Java
    for (Exchange exchange : exchanges) {
        String payload = exchange.getIn().getBody();
        // do something with payload
    }
}
```

52.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第53章 EHCACHE コンポーネント (非推奨)

Camel バージョン 2.1 以降で利用可能

cache コンポーネントを使用すると、EHCache をキャッシュ実装として使用してキャッシュ操作を実行できます。キャッシュ自体はオンデマンドで作成されるか、その名前のキャッシュがすでに存在する場合は、元の設定で使用されます。

このコンポーネントは、プロデューサーおよびイベントベースのコンシューマーエンドポイントをサポートします。

キャッシュコンシューマーはイベントベースのコンシューマーであり、特定のキャッシュアクティビティーをリッスンして応答するために使用できます。既存のキャッシュから選択を実行する必要がある場合は、キャッシュコンポーネント用に定義されたプロセッサーを使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cache</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

53.1. URI 形式

```
cache://cacheName[?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=#beanRef&...**

53.2. オプション

EHCache コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
cacheManagerFactory (advanced)	指定された CacheManagerFactory を使用して CacheManager を作成します。デフォルトでは、DefaultCacheManagerFactory が使用されます。		CacheManagerFactory
configuration (common)	キャッシュ構成を設定します		CacheConfiguration
configurationFile (common)	クラスパスまたはファイルシステムからロードする ehcache.xml ファイルのロケーションを設定します。デフォルトでは、ファイルは classpath:ehcache.xml からロードされます	classpath:ehcache.xml	String

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

EHCache エンドポイントは、URI 構文を使用して設定されます。

```
cache:cacheName
```

パスおよびクエリーパラメーターを使用します。

53.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須** キャッシュの名前		String

53.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
diskExpiryThreadInterval Seconds (common)	ディスク期限切れスレッドの実行間隔の秒数。		long
diskPersistent (common)	アプリケーションの再起動間でディスクストアが保持されるかどうか。	false	boolean
diskStorePath (common)	**非推奨** このパラメーターは無視されます。CacheManager は、setter インジェクションを使用して設定します。		String
eternal (common)	要素が永続的かどうかを設定します。永続的である場合、タイムアウトは無視され、要素は期限切れになりません。	false	boolean
key (common)	使用するデフォルトのキー。メッセージヘッダーにキーが指定されている場合は、ヘッダーのキーが優先されます。		String
maxElementsInMemory (common)	メモリー内の定義済みキャッシュに格納できる要素の数。	1000	int

名前	説明	デフォルト	タイプ
memoryStoreEvictionPolicy (common)	メモリー内の要素が最大数に達したときに使用するエビクションストラテジー。ストラテジーは、削除する要素を定義します。LRU - 最近使用されないようにする LFU - 頻繁に使用されないようにする FIFO - 先入れ先出し	LFU	MemoryStoreEviction Policy
objectCache (common)	シリアル化できないオブジェクトをキャッシュに保存できるようにするかどうか。このオプションが有効になっている場合、ディスクへのオーバーフローも有効にできません。	false	boolean
operation (common)	使用するデフォルトのキャッシュ操作。メッセージヘッダー内の操作の場合、ヘッダーからの操作が優先されます。		String
overflowToDisk (common)	キャッシュがディスクにオーバーフローする可能性があるかどうかを指定します	true	boolean
timeToIdleSeconds (common)	要素の有効期限が切れるまでのアクセス間の最大時間	300	long
timeToLiveSeconds (common)	要素が作成されてから有効期限が切れるまでの最大時間。要素が永続的でない場合にのみ使用されます	300	long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cacheLoaderRegistry (advanced)	CacheLoaderRegistry を使用してキャッシュローダーを設定するには		CacheLoaderRegistry

名前	説明	デフォルト	タイプ
cacheManagerFactory (advanced)	カスタム CacheManagerFactory を使用して、このエンドポイントで使用される CacheManager を作成します。デフォルトでは、コンポーネントに設定された CacheManagerFactory が使用されます。		CacheManagerFactory
eventListenerRegistry (advanced)	CacheEventListenerRegistry を使用してイベントリスナーを設定するには		CacheEventListener Registry
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

53.3. キャッシュとのメッセージの送受信

53.3.1. Camel 2.7 までのメッセージヘッダー

ヘッダー	説明
CACHE_OPERATION	キャッシュに対して実行される操作。有効なオプションは以下のとおりです * GET * CHECK * ADD * UPDATE * DELETE * DELETEALL **GET** と **CHECK** には **Camel 2.3** 以降が必要です。
CACHE_KEY	メッセージをキャッシュに格納するために使用されるキャッシュキー。 CACHE_OPERATION が DELETEALL の場合、キャッシュキーはオプションです。

53.3.2. Message Headers Camel 2.8 以降

Camel 2.8 でのヘッダーの変更

ヘッダー名とサポートされる値が変更され、'CamelCache' の接頭辞が付けられ、大文字と小文字が混在して使用されるようになりました。これにより、識別しやすくなり、他のヘッダーと区別しやすくなります。CacheConstants 変数名は変更されず、値が変更されただけです。また、これらのヘッダーは、キャッシュ操作の実行後にエクスチェンジから削除されるようになりました。

ヘッダー	説明
CamelCacheOperation	キャッシュに対して実行される操作。有効なオプションは次のとおりです。 * CamelCacheGet * CamelCacheCheck * CamelCacheAdd * CamelCacheUpdate * CamelCacheDelete * CamelCacheDeleteAll
CamelCacheKey	メッセージをキャッシュに格納するために使用されるキャッシュキー。 CamelCacheOperation が CamelCacheDeleteAll の場合、キャッシュキーはオプションです。

CamelCacheAdd および **CamelCacheUpdate** 操作は、追加のヘッダーをサポートします。

ヘッダー	タイプ	説明
CamelCacheTimeToLive	**Integer**	Camel 2.11: 秒単位の動作時間。
CamelCacheTimeToIdle	**Integer**	Camel 2.11: アイドル状態になるまでの時間 (秒単位)。
CamelCacheEternal	**Boolean**	Camel 2.11: コンテンツが永続的かどうか。

53.3.3. Cache Producer

キャッシュへのデータの送信には、エクスチェンジのペイロードを、既存またはオンデマンドで作成されたキャッシュに格納するように指示する機能が含まれます。これを行うメカニズムには、

- 上記のメッセージエクスチェンジヘッダーを設定します。
- Message Exchange Body にキャッシュに送信されたメッセージが含まれていることを確認する

53.3.4. Cache Consumer

キャッシュからデータを受信するには、イベントリスナーを使用して既存のキャッシュまたはオンデマンドで作成されたキャッシュをリッスンし、キャッシュアクティビティーが発生したときに自動通知を受信する CacheConsumer の機能が必要です (つまり、CamelCacheGet/CamelCacheUpdate/CamelCacheDelete/CamelCacheDeleteAll)。このような活動が行われると

- メッセージエクスチェンジヘッダーを含む交換と、追加/更新されたばかりのペイロードを含むメッセージエクスチェンジボディーが配置され、送信されます。
- CamelCacheDeleteAll 操作の場合、Message Exchange Header CamelCacheKey と Message Exchange Body は取り込まれません。

53.3.5. キャッシュプロセッサー

キャッシュルックアップを実行し、ペイロードコンテンツを選択的に置き換える機能を備えた一連の優れたプロセッサーがあります。

- body
- token
- xpath レベル

53.4. キャッシュの使用例

53.4.1. 例 1: キャッシュの設定

```
from("cache://MyApplicationCache" +
          "?maxElementsInMemory=1000" +
          "&memoryStoreEvictionPolicy=" +
             "MemoryStoreEvictionPolicy.LFU" +
          "&overflowToDisk=true" +
          "&eternal=true" +
          "&timeToLiveSeconds=300" +
          "&timeToIdleSeconds=true" +
          "&diskPersistent=true" +
          "&diskExpiryThreadIntervalSeconds=300")
```

53.4.2. 例 2: キャッシュへのキーの追加

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
     from("direct:start")
     .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
     .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
     .to("cache://TestCache1")
   }
};
```

53.4.3. 例 2: キャッシュ内の既存のキーを更新する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
     from("direct:start")
     .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_UPDATE))
     .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
     .to("cache://TestCache1")
   }
};
```

53.4.4. 例 3: キャッシュ内の既存のキーを削除する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
     from("direct:start")
     .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_DELETE))
     .setHeader(CacheConstants.CACHE_KEY", constant("Ralph_Waldo_Emerson"))
     .to("cache://TestCache1")
   }
};
```

53.4.5. 例 4: キャッシュ内のすべての既存のキーを削除する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
     from("direct:start")
     .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_DELETEALL))
     .to("cache://TestCache1");
    }
};
```

53.4.6. 例 5: キャッシュに登録されている変更をプロセッサーおよび他のプロデューサーに通知する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
     from("cache://TestCache1")
     .process(new Processor() {
       public void process(Exchange exchange)
            throws Exception {
          String operation = (String)
exchange.getIn().getHeader(CacheConstants.CACHE_OPERATION);
          String key = (String) exchange.getIn().getHeader(CacheConstants.CACHE_KEY);
          Object body = exchange.getIn().getBody();
          // Do something
       }
     })
   }
};
```

53.4.7. 例 6: プロセッサーを使用してペイロードを選択的にキャッシュ値に置き換える

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    //Message Body Replacer
    from("cache://TestCache1")
    .filter(header(CacheConstants.CACHE_KEY).isEqualTo("greeting"))
    .process(new CacheBasedMessageBodyReplacer("cache://TestCache1","farewell"))
    .to("direct:next");

   //Message Token replacer
   from("cache://TestCache1")
   .filter(header(CacheConstants.CACHE_KEY).isEqualTo("quote"))
   .process(new CacheBasedTokenReplacer("cache://TestCache1","novel","#novel#"))
   .process(new CacheBasedTokenReplacer("cache://TestCache1","author","#author#"))
   .process(new CacheBasedTokenReplacer("cache://TestCache1","number","#number#"))
   .to("direct:next");

   //Message XPath replacer
   from("cache://TestCache1").
   .filter(header(CacheConstants.CACHE_KEY).isEqualTo("XML_FRAGMENT"))
   .process(new CacheBasedXPathReplacer("cache://TestCache1","book1","/books/book1"))
   .process (new CacheBasedXPathReplacer("cache://TestCache1","book2","/books/book2"))
```

```
    .to("direct:next");
  }
};
```

53.4.8. 例 7: キャッシュからエントリーを取得する

```
from("direct:start")
   // Prepare headers
   .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_GET))
   .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson")).
   .to("cache://TestCache1").
   // Check if entry was not found
   .choice().when(header(CacheConstants.CACHE_ELEMENT_WAS_FOUND).isNull()).
      // If not found, get the payload and put it to cache
      .to("cxf:bean:someHeavyweightOperation").
      .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
      .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
      .to("cache://TestCache1")
   .end()
   .to("direct:nextPhase");
```

53.4.9. 例 8: キャッシュ内のエントリーを確認する

注記: CHECK コマンドは、キャッシュ内のエントリーの存在をテストしますが、ボディーにはメッセージを配置しません。

```
from("direct:start")
   // Prepare headers
   .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_CHECK))
   .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson")).
   .to("cache://TestCache1").
   // Check if entry was not found
   .choice().when(header(CacheConstants.CACHE_ELEMENT_WAS_FOUND).isNull()).
      // If not found, get the payload and put it to cache
      .to("cxf:bean:someHeavyweightOperation").
      .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
      .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
      .to("cache://TestCache1")
   .end();
```

53.5. EHCACHE の管理

EHCache には、JMX からの独自の統計情報と管理があります。

Spring アプリケーションコンテキストで JMX を介してそれらを公開する方法のスニペットを次に示します。

```
<bean id="ehCacheManagementService" class="net.sf.ehcache.management.ManagementService"
init-method="init" lazy-init="false">
```

```
    <constructor-arg>
      <bean class="net.sf.ehcache.CacheManager" factory-method="getInstance"/>
    </constructor-arg>
    <constructor-arg>
      <bean class="org.springframework.jmx.support.JmxUtils" factory-method="locateMBeanServer"/>
    </constructor-arg>
    <constructor-arg value="true"/>
    <constructor-arg value="true"/>
    <constructor-arg value="true"/>
    <constructor-arg value="true"/>
  </bean>
```

もちろん、そのままの Java でも同じことができます。

```
ManagementService.registerMBeans(CacheManager.getInstance(), mbeanServer, true, true, true,
true);
```

この方法で、キャッシュヒット、ミス、メモリー内ヒット、ディスクヒット、サイズ統計を取得できます。その場で CacheConfiguration パラメーターを変更することもできます。

53.6. キャッシュレプリケーション CAMEL 2.8

Camel Cache コンポーネントは、RMI、JGroups、JMS、Cache Server などのいくつかの異なるレプリケーションメカニズムを使用して、サーバーノード間でキャッシュを分散できます。

それを動作させるには、次の 2 つの方法があります。

1.**ehcache.xml** を手動で設定できます

または

2.次の 3 つのオプションを設定できます。

- cacheManagerFactory
- eventListenerRegistry
- cacheLoaderRegistry

最初のオプションを使用して Camel Cache レプリケーションを設定するのは、すべてのキャッシュを個別に設定する必要があるため、少し大変です。そのため、キャッシュのすべての名前が不明な状況では、**ehcache.xml** を使用することはお勧めできません。

2 番目のオプションは、キャッシュごとにオプションを定義する必要がないため、多数の異なるキャッシュを使用する場合に適しています。これは、レプリケーションオプションが **CacheManager** ごとおよび **CacheEndpoint** ごとに設定されるためです。また、開発段階でキャッシュ名がわからない場合の唯一の方法です。

注記: Camel Cache のレプリケーションメカニズムをよりよく理解するには、EHCache のマニュアルを読むと役立つ場合があります。

53.6.1. 例: JMS キャッシュのレプリケーション

JMS レプリケーションは、最も強力で安全なレプリケーション方法です。Camel Cache レプリケーションと一緒に使用すると、かなり単純になります。例は 別のページ にあります。

第54章 CAFFEINE CACHE コンポーネント

Camel バージョン 2.20 以降で利用可能

caffeine-cache コンポーネントを使用すると、Caffeine の単純なキャッシュを使用してキャッシュ操作を実行できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-caffeine</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

54.1. URI 形式

```
caffeine-cache://cacheName[?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=#beanRef&...**

54.2. オプション

Caffeine Cache コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	グローバルコンポーネント設定を設定します		CaffeineConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Caffeine Cache エンドポイントは、URI 構文を使用して設定されます。

```
caffeine-cache:cacheName
```

パスおよびクエリーパラメーターを使用します。

54.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須** キャッシュ名		String

54.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
createCacheIfNotExist (common)	キャッシュが存在する場合、または事前設定できない場合にキャッシュを作成する必要があるかどうかを設定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
action (producer)	デフォルトのキャッシュアクションを設定します。メッセージヘッダーにアクションが設定されている場合は、ヘッダーからの操作が優先されます。		String
cache (producer)	デフォルトですでにインスタンス化されたキャッシュを使用するように設定するには		Cache
cacheLoader (producer)	LoadCache を使用する場合に CacheLoader を設定するには		CacheLoader
evictionType (producer)	このキャッシュの削除タイプを設定します	SIZE_BASED	EvictionType
expireAfterAccessTime (producer)	時間ベースのエビクションの場合は、アクセス時間後に有効期限を設定します (秒単位)。	300	int
expireAfterWriteTime (producer)	時間ベースのエビクションの場合に、アクセス書き込み後に有効期限を設定します (秒単位)	300	int
initialCapacity (producer)	キャッシュの初期容量を設定します	10000	int

名前	説明	デフォルト	タイプ
key (producer)	デフォルトのアクションキーを設定します。メッセージヘッダーにキーが設定されている場合は、ヘッダーのキーが優先されます。		Object
maximumSize (producer)	キャッシュの最大サイズを設定する	10000	int
removalListener (producer)	キャッシュに特定の削除リスナーを設定する		RemovalListener
statsCounter (producer)	キャッシュ統計用の特定の統計カウンターを設定します		StatsCounter
statsEnabled (producer)	キャッシュの統計を有効にするには	false	boolean
keyType (advanced)	キャッシュキータイプ、デフォルトの java.lang.Object	java.lang.Object	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
valueType (advanced)	キャッシュ値の型、デフォルトは java.lang.Object	java.lang.Object	文字列

第55章 CAFFEINE LOADCACHE コンポーネント

Camel バージョン 2.20 以降で利用可能

caffeine-loadcache コンポーネントを使用すると、Caffeine のロードキャッシュを使用してキャッシュ操作を実行できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-caffeine</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

55.1. URI 形式

```
caffeine-loadcache://cacheName[?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=#beanRef&...**

55.2. オプション

Caffeine LoadCache コンポーネントは、以下に示す 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	グローバルコンポーネント設定を設定します		CaffeineConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Caffeine LoadCache エンドポイントは、URI 構文を使用して設定されます。

```
caffeine-loadcache:cacheName
```

パスおよびクエリーパラメーターを使用します。

55.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須** キャッシュ名		String

55.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
createCacheIfNotExist (common)	キャッシュが存在する場合、または事前設定できない場合にキャッシュを作成する必要があるかどうかを設定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
action (producer)	デフォルトのキャッシュアクションを設定します。メッセージヘッダーにアクションが設定されている場合は、ヘッダーからの操作が優先されます。		String
cache (producer)	デフォルトですでにインスタンス化されたキャッシュを使用するように設定するには		Cache
cacheLoader (producer)	LoadCache を使用する場合に CacheLoader を設定するには		CacheLoader
evictionType (producer)	このキャッシュの削除タイプを設定します	SIZE_BASED	EvictionType
expireAfterAccessTime (producer)	時間ベースのエビクションの場合は、アクセス時間後に有効期限を設定します (秒単位)。	300	int
expireAfterWriteTime (producer)	時間ベースのエビクションの場合に、アクセス書き込み後に有効期限を設定します (秒単位)	300	int
initialCapacity (producer)	キャッシュの初期容量を設定します	10000	int

名前	説明	デフォルト	タイプ
key (producer)	デフォルトのアクションキーを設定します。メッセージヘッダーにキーが設定されている場合は、ヘッダーのキーが優先されます。		Object
maximumSize (producer)	キャッシュの最大サイズを設定する	10000	int
removalListener (producer)	キャッシュに特定の削除リスナーを設定する		RemovalListener
statsCounter (producer)	キャッシュ統計用の特定の統計カウンターを設定します		StatsCounter
statsEnabled (producer)	キャッシュの統計を有効にするには	false	boolean
keyType (advanced)	キャッシュキータイプ、デフォルトの java.lang.Object	java.lang.Object	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
valueType (advanced)	キャッシュ値の型、デフォルトは java.lang.Object	java.lang.Object	文字列

第56章 CASTOR DATAFORMAT (非推奨)

Camel バージョン 2.1 以降で利用可能

Castor は、Castor XML ライブラリー を使用して XML ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを XML ペイロードにマーシャリングするデータ形式です。

通常、Java DSL または Spring XML のいずれかを使用して Castor Data Format を操作できます。

56.1. JAVA DSL を使用

```
from("direct:order").
  marshal().castor().
  to("activemq:queue:order");
```

たとえば、次の例では、デフォルトの Castor データバインディング機能を使用する名前付きの Castor の DataFormat を使用しています。

```
CastorDataFormat castor = new CastorDataFormat ();

from("activemq:My.Queue").
  unmarshal(castor).
  to("mqseries:Another.Queue");
```

必要に応じて、Spring XML ファイルなどを介してレジストリーで定義できるデータ形式への名前付き参照を使用します。以下の例を示します。

```
from("activemq:My.Queue").
  unmarshal("mycastorType").
  to("mqseries:Another.Queue");
```

マッピングファイルを提供してデフォルトのマッピングスキーマをオーバーライドする場合は、次のように設定できます。

```
CastorDataFormat castor = new CastorDataFormat ();
castor.setMappingFile("mapping.xml");
```

また、Castor Marshaller と Unmarshaller をさらに制御したい場合は、以下のようにアクセスできます。

```
castor.getMarshaller();
castor.getUnmarshaller();
```

56.2. SPRING XML の使用

次の例は、Castor データ型を設定する Spring を使用してアンマーシャリングするために Castor を使用する方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
```

```
      <castor validation="true" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

この例は、データ型を 1 回だけ設定し、それを複数のルートで再利用する方法を示しています。<castor> 要素を <camelContext> に直接設定する必要があります。

```
<camelContext>
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <castor id="myCastor"/>
  </dataFormats>

  <route>
    <from uri="direct:start"/>
    <marshal ref="myCastor"/>
    <to uri="direct:marshalled"/>
  </route>
  <route>
    <from uri="direct:marshalled"/>
    <unmarshal ref="myCastor"/>
    <to uri="mock:result"/>
  </route>

</camelContext>
```

56.3. オプション

Castor データ形式は、以下に示す 9 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
mappingFile		**String**	クラスパスからロードする Castor マッピングファイルへのパス。
whitelistEnabled	**true**	**Boolean**	ホワイトリスト機能を有効にするかどうかを定義します
allowedUnmarshallObjects		**String**	非整列化を許可するオブジェクトを定義します。許可されたオブジェクトの FQN クラス名を指定でき、コンマを使用して複数のエントリーを区切ることができます。リンク org.apache.camel.util.EndpointHelpermatchPattern (String、String) で定義されたパターンに基づくワイルドカードと正規表現を使用することもできます。拒否されたオブジェクトは、許可されたオブジェクトよりも優先されます。

名前	デフォルト	Java タイプ	説明
deniedUnmarshallObjects		String	拒否されたオブジェクトをアンマーシャリングするように定義します。定義済みオブジェクトの FQN クラス名を指定でき、コンマを使用して複数のエントリーを区切ることができます。リンク org.apache.camel.util.EndpointHelpermatchPattern (String、String) で定義されたパターンに基づくワイルドカードと正規表現を使用することもできます。拒否されたオブジェクトは、許可されたオブジェクトよりも優先されます。
validation	true	Boolean	検証が有効か無効か。デフォルトでは true です。
encoding	UTF-8	String	オブジェクトを XML にマーシャリングするときに使用するエンコーディング。デフォルトでは UTF-8 です。
packages		String[]	追加のパッケージを Castor XmlContext に追加する
classes		String[]	クラス名を Castor XmlContext に追加する
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

56.4. 依存関係

camel ルートで Castor を使用するには、このデータ形式を実装する **camel-castor** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-castor</artifactId>
  <version>x.x.x</version>
</dependency>
```

第57章 CAMEL CDI

Camel CDI コンポーネントは、**convention-over-configuration** に基づく依存性注入フレームワークとして CDI を使用して、Apache Camel の自動設定を提供します。アプリケーションで使用可能な Camel ルートを自動検出し、**Endpoint**、**FluentProducerTemplate**、**ProducerTemplate**、**TypeConverter** などの一般的な Camel プリミティブの Bean を提供します。**@Consume**、**@Produce**、**@PropertyInject** などの Camel アノテーションを CDI Bean でシームレスに使用できるように、標準の Camel Bean 統合を実装します。さらに、Camel イベント (**RouteAddedEvent**、**CamelContextStartedEvent**、**ExchangeCompletedEvent** など) を CDI イベントとしてブリッジし、Camel ルートから/への CDI イベントを消費/生成するために使用できる CDI イベントエンドポイントを提供します。

Camel CDI コンポーネントは **Camel 2.10** で利用できますが、CDI プログラミングモデルにより適合するように **Camel 2.17** で書き直されました。したがって、Camel イベントから CDI イベントへのブリッジや CDI イベントエンドポイントなどの一部の機能は、Camel 2.17 以降にのみ適用されます。

Camel CDI アプリケーションをテストする方法の詳細については、Camel CDI テストを参照してください。

注意

camel-cdi は OSGi では推奨されておらず、サポートされていません。OSGi で Camel を使用する場合は、OSGi Blueprint を使用します。

57.1. 自動設定された CAMEL コンテキスト

Camel CDI は、**CamelContext** Bean を自動的にデプロイおよび設定します。その **CamelContext** Bean は、CDI コンテナーの初期化 (またはシャットダウン) 時に、自動的にインスタンス化、設定、および開始 (または停止) されます。アプリケーションに注入できます。たとえば、次のようになります。

```
@Inject
CamelContext context;
```

そのデフォルトの **CamelContext** Bean は、ビルトインの **@Default** 修飾子で修飾され、スコープが **@ApplicationScoped** であり、タイプ **DefaultCamelContext** です。

この Bean はプログラムでカスタマイズでき、他の Camel コンテキスト Bean もアプリケーションにデプロイできることに注意してください。

57.2. CAMEL ルートの自動検出

Camel CDI は、アプリケーション内のすべての **RoutesBuilder** Bean を自動的に収集し、インスタンス化して、CDI コンテナーの初期化時に **CamelContext** Bean インスタンスに追加します。たとえば、Camel ルートの追加は、クラスを宣言するのと同じくらい簡単です。

```
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
        from("jms:invoices").to("file:/invoices");
    }
}
```

必要な数の **RoutesBuilder** Bean を宣言できることに注意してください。さらに、**RouteContainer** Bean も自動的に収集され、インスタンス化され、コンテナーの初期化時に Camel CDI によって管理される **CamelContext** Bean インスタンスに追加されます。

Camel 2.19 以降で利用可能

状況によっては、**RouteBuilder** および **RouteContainer** Bean の自動設定を無効にする必要がある場合があります。これは、**CdiCamelConfiguration** イベントを監視することで実現できます。

```
static void configuration(@Observes CdiCamelConfiguration configuration) {
    configuration.autoConfigureRoutes(false);
}
```

同様に、設定された **CamelContext** Bean の自動起動を無効にすることができます。

```
static void configuration(@Observes CdiCamelConfiguration configuration) {
    configuration.autoStartContexts(false);
}
```

57.3. 自動設定された CAMEL プリミティブ

Camel CDI は、任意の CDI Bean に注入できる一般的な Camel プリミティブの Bean を提供します。

```
@Inject
@Uri("direct:inbound")
ProducerTemplate producerTemplate;

@Inject
@Uri("direct:inbound")
FluentProducerTemplate fluentProducerTemplate;

@Inject
MockEndpoint outbound; // URI defaults to the member name, i.e. mock:outbound

@Inject
@Uri("direct:inbound")
Endpoint endpoint;

@Inject
TypeConverter converter;
```

57.4. CAMEL コンテキスト設定

デフォルトの **CamelContext** Bean の名前を変更するだけの場合は、Camel CDI によって提供される **@ContextName** 修飾子を使用できます。

```
@ContextName("camel-context")
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
```

```
        from("jms:invoices").to("file:/invoices");
    }
}
```

それ以外の場合、さらにカスタマイズが必要な場合は、任意の **CamelContext** クラスを使用してカスタム Camel コンテキスト Bean を宣言できます。次に、**@PostConstruct** および **@PreDestroy** ライフサイクルコールバックを実行して、カスタマイズを行うことができます。

```
@ApplicationScoped
class CustomCamelContext extends DefaultCamelContext {

    @PostConstruct
    void customize() {
        // Set the Camel context name
        setName("custom");
        // Disable JMX
        disableJMX();
    }

    @PreDestroy
    void cleanUp() {
        // ...
    }
}
```

Producer メソッド と disposer メソッドを使用して、Camel コンテキスト Bean をカスタマイズすることもできます。次に例を示します。

```
class CamelContextFactory {

    @Produces
    @ApplicationScoped
    CamelContext customize() {
        DefaultCamelContext context = new DefaultCamelContext();
        context.setName("custom");
        return context;
    }

    void cleanUp(@Disposes CamelContext context) {
        // ...
    }
}
```

同様に、producer fields を使用できます。

```
@Produces
@ApplicationScoped
CamelContext context = new CustomCamelContext();

class CustomCamelContext extends DefaultCamelContext {

    CustomCamelContext() {
        setName("custom");
    }
}
```

このパターンは、たとえば、コンテナーが **setAutoStartup** メソッドを呼び出して初期化するときに Camel コンテキストルートが自動的に開始されるのを回避するために使用できます。

```
@ApplicationScoped
class ManualStartupCamelContext extends DefaultCamelContext {

    @PostConstruct
    void manual() {
        setAutoStartup(false);
    }
}
```

57.5. 複数の CAMEL コンテキスト

上で説明したように、実際にはアプリケーションで任意の数の **CamelContext** Bean を宣言できます。その場合、これらの **CamelContext** Bean で宣言された CDI 修飾子は、対応する Camel コンテキストへの Camel ルートおよび他の Camel プリミティブを BIND するために使用されます。例から、次の Bean が宣言された場合:

```
@ApplicationScoped
@ContextName("foo")
class FooCamelContext extends DefaultCamelContext {
}

@ApplicationScoped
@BarContextQualifier
class BarCamelContext extends DefaultCamelContext {
}

@ContextName("foo")
class RouteAddedToFooCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}

@BarContextQualifier
class RouteAddedToBarCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}

@ContextName("baz")
class RouteAddedToBazCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}
```

```
@MyOtherQualifier
class RouteNotAddedToAnyCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}
```

@ContextName で修飾された **RoutesBuilder** Bean は、Camel CDI によって対応する **CamelContext** Bean に自動的に追加されます。そのような **CamelContext** Bean が存在しない場合は、**RouteAddedToBazCamelContext** Bean の場合と同様に、自動的に作成されます。これは、Camel CDI によって提供される **@ContextName** 修飾子に対してのみ発生することに注意してください。したがって、ユーザー定義の **@MyOtherQualifier** 修飾子で修飾された **RouteNotAddedToAnyCamelContext** Bean は、どの Camel コンテキストにも追加されません。これは、たとえば、後でアプリケーションの実行中に追加する必要がある Camel ルートの場合に役立ちます。

注記

Camel バージョン 2.17.0 以降、Camel CDI はあらゆる種類の **CamelContext** Bean (例: **DefaultCamelContext**) を管理できます。以前のバージョンでは、タイプ **CdiCamelContext** の Bean しか管理できないため、拡張する必要があります。

CamelContext Bean で宣言された CDI 修飾子は、対応する Camel プリミティブを BIND するためにも使用されます。

```
@Inject
@ContextName("foo")
@Uri("direct:inbound")
ProducerTemplate producerTemplate;

@Inject
@ContextName("foo")
@Uri("direct:inbound")
FluentProducerTemplate fluentProducerTemplate;

@Inject
@BarContextQualifier
MockEndpoint outbound; // URI defaults to the member name, i.e. mock:outbound

@Inject
@ContextName("baz")
@Uri("direct:inbound")
Endpoint endpoint;
```

57.6. 設定プロパティー

Camel がプロパティープレースホルダーを解決するために使用する設定プロパティーのソースを設定するには、**@Named("properties")** で修飾された **PropertiesComponent** Bean を宣言できます。

```
@Produces
@ApplicationScoped
```

```
@Named("properties")
PropertiesComponent propertiesComponent() {
    Properties properties = new Properties();
    properties.put("property", "value");
    PropertiesComponent component = new PropertiesComponent();
    component.setInitialProperties(properties);
    component.setLocation("classpath:placeholder.properties");
    return component;
}
```

DeltaSpike 設定メカニズム を使用する場合は、次の **PropertiesComponent** Bean を宣言できます。

```
@Produces
@ApplicationScoped
@Named("properties")
PropertiesComponent properties(PropertiesParser parser) {
    PropertiesComponent component = new PropertiesComponent();
    component.setPropertiesParser(parser);
    return component;
}

// PropertiesParser bean that uses DeltaSpike to resolve properties
static class DeltaSpikeParser extends DefaultPropertiesParser {
    @Override
    public String parseProperty(String key, String value, Properties properties) {
        return ConfigResolver.getPropertyValue(key);
    }
}
```

DeltaSpike 設定メカニズムを使用した Camel CDI アプリケーションの実例については、**camel-example-cdi-properties の** 例を参照してください。

57.7. 自動設定型コンバーター

@Converter アノテーションが付けられた CDI Bean は、デプロイされた Camel コンテキストに自動的に登録されます。

```
@Converter
public class MyTypeConverter {

    @Converter
    public Output convert(Input input) {
        //...
    }
}
```

型コンバーター内で CDI インジェクションがサポートされていることに注意してください。

57.8. CAMEL BEAN の統合

57.8.1. Camel アノテーション

Camel Bean 統合 の一部として、Camel には、Camel CDI によってシームレスにサポートされる一連の アノテーション が付属しています。したがって、CDI Bean でこれらのアノテーションのいずれかを使用できます。

Camel アノテーション	CDI 相当
設定プロパティー	DeltaSpike 設定メカニズム を使用する場合: `@Inject` `@ConfigProperty(name = "key")` `String value;` 詳細については、設定プロパティーを参照してください。
プロデューサーテンプレートインジェクション (デフォルトの Camel コンテキスト)	`@Produce(uri = "mock:outbound")` `ProducerTemplate producer;` `@Produce(uri = "mock:outbound")` `FluentProducerTemplate producer;`	`@Inject` `@Uri("direct:outbound")` `ProducerTemplate producer;` `@Produce(uri = "direct:outbound")` `FluentProducerTemplate producer;`
エンドポイントインジェクション (デフォルトの Camel コンテキスト)	`@EndpointInject(uri = "direct:inbound")` `Endpoint endpoint;`	`@Inject` `@Uri("direct:inbound")` `Endpoint endpoint;`
エンドポイントインジェクション (名前による Camel コンテキスト)	`@EndpointInject(uri = "direct:inbound",` `context = "foo")` `Endpoint contextEndpoint;`	`@Inject` `@ContextName("foo")` `@Uri("direct:inbound")` `Endpoint contextEndpoint;`
Bean インジェクション (種類別)	`@BeanInject` `MyBean bean;`	`@Inject` `MyBean bean;`
Bean インジェクション (名前別)	`@BeanInject("foo")` `MyBean bean;`	`@Inject` `@Named("foo")` `MyBean bean;`

	Camel アノテーション	CDI 相当
POJO consuming	`@Consume(uri = "seda:inbound")` `void consume(@Body String body) {` `  //...` `}`	

57.8.2. Bean コンポーネント

Java Camel DSL などを使用して、Camel DSL から、タイプまたは名前で CDI Bean を参照できます。

```
class MyBean {
    //...
}

from("direct:inbound").bean(MyBean.class);
```

または、Java DSL から名前で CDI Bean を検索するには:

```
@Named("foo")
class MyNamedBean {
    //...
}

from("direct:inbound").bean("foo");
```

57.8.3. エンドポイント URI からの Bean の参照

URI 構文を使用してエンドポイントを設定する場合、**#** 表記を使用してレジストリー内の Bean を参照できます。 URI パラメーター値が **#** 記号で始まる場合、Camel CDI は指定されたタイプの Bean を名前で検索します。

```
from("jms:queue:{{destination}}?
transacted=true&transactionManager=#jtaTransactionManager").to("...");
```

次の CDI Bean を **@Named("jtaTransactionManager")** で修飾します。

```
@Produces
@Named("jtaTransactionManager")
PlatformTransactionManager createTransactionManager(TransactionManager transactionManager,
UserTransaction userTransaction) {
    JtaTransactionManager jtaTransactionManager = new JtaTransactionManager();
    jtaTransactionManager.setUserTransaction(userTransaction);
    jtaTransactionManager.setTransactionManager(transactionManager);
    jtaTransactionManager.afterPropertiesSet();
    return jtaTransactionManager;
}
```

57.9. CAMEL イベントから CDI イベントへ

Camel 2.17 以降で利用可能

Camel は、Camel コンテキスト、サービス、ルート、およびエクスチェンジイベントをリッスンするためにサブスクライブできる一連の 管理イベント を提供します。Camel CDI は、これらの Camel イベントを CDI observer メソッド を使用して監視できる CDI イベントにシームレスに変換します。

```
void onContextStarting(@Observes CamelContextStartingEvent event) {
    // Called before the default Camel context is about to start
}
```

Camel 2.18 の時点で、特定のルート (**RouteAddedEvent**、**RouteStartedEvent**、**RouteStoppedEvent**、および **RouteRemovedEvent**) のイベントを監視することが可能です。

```
from("...").routeId("foo").to("...");

void onRouteStarted(@Observes @Named("foo") RouteStartedEvent event) {
    // Called after the route "foo" has started
}
```

複数の Camel コンテキストが CDI コンテナーに存在する場合、**@ContextName** などの Camel コンテキスト Bean 修飾子を使用して、observer resolution で指定されている特定の Camel コンテキストに observer メソッドのソリューションを絞り込むことができます。次に例を示します。

```
void onRouteStarted(@Observes @ContextName("foo") RouteStartedEvent event) {
    // Called after the route 'event.getRoute()' for the Camel context 'foo' has started
}

void onContextStarted(@Observes @Manual CamelContextStartedEvent event) {
    // Called after the the Camel context qualified with '@Manual' has started
}
```

同様に、**@Default** 修飾子を使用して、複数のコンテキストが存在する場合、**デフォルトの** Camel コンテキストの Camel イベントを監視できます。

```
void onExchangeCompleted(@Observes @Default ExchangeCompletedEvent event) {
    // Called after the exchange 'event.getExchange()' processing has completed
}
```

その例では、修飾子が指定されていない場合、**@Any** 修飾子が暗黙的に想定されるため、すべての Camel コンテキストの対応するイベントが受信されます。

Camel イベントの CDI イベントへの変換のサポートは、デプロイメントで Camel イベントをリッスンする observer メソッドが検出され、Camel コンテキストごとに検出された場合にのみアクティブ化されることに注意してください。

57.10. CDI イベントエンドポイント

Camel 2.17 以降で利用可能

CDI イベントエンドポイントは、CDI イベントを Camel ルートにブリッジし、CDI イベントを Camel コンシューマー (または Camel プロデューサー) からシームレスに監視/消費 (または生成/起動) できるようにします。

Camel CDI が提供する **CdiEventEndpoint<T>** Bean を使用して、**イベントタイプ** が **T** である CDI イベントを監視/消費できます。次に例を示します。

```
@Inject
CdiEventEndpoint<String> cdiEventEndpoint;

from(cdiEventEndpoint).log("CDI event received: ${body}");
```

これは次のように記述することと同じです:

```
@Inject
@Uri("direct:event")
ProducerTemplate producer;

void observeCdiEvents(@Observes String event) {
    producer.sendBody(event);
}

from("direct:event").log("CDI event received: ${body}");
```

逆に、**CdiEventEndpoint<T>** Bean を使用して、**イベントタイプ** が **T** である CDI イベントを生成/起動できます。次に例を示します。

```
@Inject
CdiEventEndpoint<String> cdiEventEndpoint;

from("direct:event").to(cdiEventEndpoint).log("CDI event sent: ${body}");
```

これは次のように記述することと同じです:

```
@Inject
Event<String> event;

from("direct:event").process(new Processor() {
    @Override
    public void process(Exchange exchange) {
        event.fire(exchange.getBody(String.class));
    }
}).log("CDI event sent: ${body}");
```

または、Java 8 lambda 式を使用します。

```
@Inject
Event<String> event;

from("direct:event")
    .process(exchange -> event.fire(exchange.getIn().getBody(String.class)))
    .log("CDI event sent: ${body}");
```

特定の **CdiEventEndpoint<T>** インジェクションポイントの型変数 **T** (resp. 修飾子) は、パラメーター化された **イベント型** (resp. **イベント修飾子**) に自動的に変換されます。

```
@Inject
@FooQualifier
CdiEventEndpoint<List<String>> cdiEventEndpoint;

from("direct:event").to(cdiEventEndpoint);

void observeCdiEvents(@Observes @FooQualifier List<String> event) {
    logger.info("CDI event: {}", event);
}
```

複数の Camel コンテキストが CDI コンテナーに存在する場合、**@ContextName** などの Camel コンテキスト Bean 修飾子を使用して、**CdiEventEndpoint<T>** 注入ポイントを修飾できます。次に例を示します。

```
@Inject
@ContextName("foo")
CdiEventEndpoint<List<String>> cdiEventEndpoint;
// Only observes / consumes events having the @ContextName("foo") qualifier
from(cdiEventEndpoint).log("Camel context (foo) > CDI event received: ${body}");
// Produces / fires events with the @ContextName("foo") qualifier
from("...").to(cdiEventEndpoint);

void observeCdiEvents(@Observes @ContextName("foo") List<String> event) {
    logger.info("Camel context (foo) > CDI event: {}", event);
}
```

CDI イベントの Camel エンドポイントは、**イベントタイプ** と **イベント修飾子** の一意の組み合わせごとに observer メソッドを 動的に追加し、コンテナータイプセーフ オブザーバーソリューション にのみ依存することに注意してください。これにより、実装が可能な限り効率的になります。

さらに、CDI の **タイプセーフな** 性質と Camel コンポーネント モデルの **動的な** 性質の間のインピーダンスが非常に高いため、URI を介して CDI イベント Camel エンドポイントのインスタンスを作成することはできません。実際、CDI イベントコンポーネントの URI 形式は次のとおりです。

```
cdi-event://PayloadType<T1,...,Tn>[?qualifiers=QualifierType1[,...[,QualifierTypeN]...]]
```

権限 **PayloadType** (resp. **QualifierType**) は、URI エスケープされたペイロード (resp. qualifier) の raw タイプの完全修飾名であり、その後に、ペイロードのパラメーター化された型の山括弧で区切られた型パラメーターセクションが続きます。これは、**不親切な** URI につながります。

```
cdi-event://org.apache.camel.cdi.example.EventPayload%3Cjava.lang.Integer%3E?
qualifiers=org.apache.camel.cdi.example.FooQualifier%2Corg.apache.camel.cdi.example.BarQualifier
```

しかし、より根本的には、CDI コンテナーにはデプロイフェーズ中に Camel コンテキストモデルを検出する方法がないため、エンドポイントインスタンスとオブザーバーメソッド間の効率的なバインディングが妨げられます。

57.11. CAMEL XML 設定のインポート

Camel 2.18 から利用可能

CDI はタイプセーフな依存性注入メカニズムを優先しますが、既存の Camel XML 設定ファイルを Camel CDI アプリケーションに再利用すると便利な場合があります。 他の使用例では、Camel XML DSL に依存してその Camel コンテキストを設定すると便利な場合があります。

Camel CDI によって提供される **@ImportResource** アノテーションを任意の CDI Bean で使用できます。Camel CDI は、指定されたロケーションに Camel XML 設定を自動的にロードします。

```
@ImportResource("camel-context.xml")
class MyBean {
}
```

Camel CDI は、クラスパスから指定されたロケーションにリソースをロードします (他のプロトコルが将来追加される可能性があります)。

インポートされたリソースのすべての **CamelContext** 要素とその他の Camel **プリミティブは**、コンテナーのブートストラップ中に CDI Bean として自動的にデプロイされるため、Camel CDI によって提供される自動設定の恩恵を受け、実行時に注入できるようになります。そのような要素に明示的な **id** 属性が設定されている場合、対応する CDI Bean は **@Named** 修飾子で修飾されます。たとえば、次の Camel XML 設定が与えられます。

```
<camelContext id="foo">
   <endpoint id="bar" uri="seda:inbound">
      <property key="queue" value="#queue"/>
      <property key="concurrentConsumers" value="10"/>
   </endpoint>
<camelContext/>
```

対応する CDI Bean が自動的にデプロイされ、注入することができます。

```
@Inject
@ContextName("foo")
CamelContext context;

@Inject
@Named("bar")
Endpoint endpoint;
```

CamelContext Bean は、**@Named** 修飾子と **@ContextName** 修飾子の両方で自動的に修飾されることに注意してください。インポートされた **CamelContext** 要素に **id** 属性がない場合、対応する Bean は組み込みの **@Default** 修飾子でデプロイされます。

逆に、アプリケーションにデプロイされた CDI Bean は、通常は **ref** 属性を使用して、Camel XML 設定から参照できます。たとえば、次の Bean が宣言されているとします。

```
@Produces
@Named("baz")
Processor processor = exchange -> exchange.getIn().setHeader("qux", "quux");
```

その Bean への参照は、インポートされた Camel XML 設定で宣言できます。

```
<camelContext id="foo">
   <route>
      <from uri="..."/>
```

```
        <process ref="baz"/>
    </route>
<camelContext/>
```

57.12. トランザクションサポート

Camel 2.19 以降で利用可能

Camel CDI は、JTA を使用して Camel トランザクションクライアントをサポートします。

このサポートはオプションであるため、アプリケーションのクラスパスに JTA を含める必要があります。たとえば、Maven を使用するときに JTA を依存関係として明示的に追加します。

```
<dependency>
  <groupId>javax.transaction</groupId>
  <artifactId>javax.transaction-api</artifactId>
  <scope>runtime</scope>
</dependency>
```

アプリケーションを JTA 対応コンテナーにデプロイするか、スタンドアロンの JTA 実装を提供する必要があります。

注意

当面の間、トランザクションマネージャーは **java:/TransactionManager** キーで JNDI リソースとしてルックアップされることに注意してください。

より幅広いデプロイメントシナリオをサポートするために、より柔軟なストラテジーが将来追加される予定です。

57.12.1. トランザクションポリシー

Camel CDI は、通常サポートされる Camel **TransactedPolicy** の実装を CDI Bean として提供します。これらのポリシーは、処理された EIP を使用して名前で検索することができます。たとえば、次のようになります。

```
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
        from("activemq:queue:foo")
            .transacted("PROPAGATION_REQUIRED")
            .bean("transformer")
            .to("jpa:my.application.entity.Bar")
            .log("${body.id} inserted");
    }
}
```

これは次と同等です。

```
class MyRouteBean extends RouteBuilder {

    @Inject
    @Named("PROPAGATION_REQUIRED")
```

```
    Policy required;

    @Override
    public void configure() {
        from("activemq:queue:foo")
            .policy(required)
            .bean("transformer")
            .to("jpa:my.application.entity.Bar")
            .log("${body.id} inserted");
    }
}
```

サポートされているトランザクションポリシー名のリストは次のとおりです。

- **PROPAGATION_NEVER**,
- **PROPAGATION_NOT_SUPPORTED**,
- **PROPAGATION_SUPPORTS**,
- **PROPAGATION_REQUIRED**,
- **PROPAGATION_REQUIRES_NEW**,
- **PROPAGATION_NESTED**,
- **PROPAGATION_MANDATORY**.

57.12.2. トランザクションエラーハンドラー

Camel CDI は、再配信エラーハンドラーを拡張するトランザクションエラーハンドラーを提供し、例外が発生するたびにロールバックを強制し、再配信ごとに新しいトランザクションを作成します。

Camel CDI は、**transactionErrorHandler** ヘルパーメソッドを公開する **CdiRouteBuilder** クラスを提供して、設定への迅速なアクセスを可能にします。

```
class MyRouteBean extends CdiRouteBuilder {

    @Override
    public void configure() {
        errorHandler(transactionErrorHandler()
            .setTransactionPolicy("PROPAGATION_SUPPORTS")
            .maximumRedeliveries(5)
            .maximumRedeliveryDelay(5000)
            .collisionAvoidancePercent(10)
            .backOffMultiplier(1.5));
    }
}
```

57.13. 自動設定された OSGI 統合

Camel 2.17 以降で利用可能

Camel コンテキスト Bean は、Camel CDI によって自動的に適応されるため、OSGi サービスとして登録され、さまざまなリゾルバー (**ComponentResolver** や **DataFormatResolver** など) が OSGi レジス

トリーと統合されます。これは、Karaf Camel コマンドを使用して、Camel CDI によって自動設定された Camel コンテキストを操作できることを意味します。

```
karaf@root()> camel:context-list
 Context       Status            Total #       Failed #     Inflight #   Uptime
 -------       ------            -------       --------     -----------  ------
 camel-cdi     Started                 1              0              0   1 minute
```

Camel CDI OSGi 統合の実際の例については、**camel-example-cdi-osgi の** 例を参照してください。

57.14. レイジーインジェクション/プログラムによるルックアップ

CDI プログラムモデルは、アプリケーションの初期化時に発生する タイプセーフな解決 メカニズムを優先しますが、プログラムによるルックアップ メカニズムを使用して、後でアプリケーションの実行中に動的/レイジーインジェクションを実行することができます。

Camel CDI は、便宜上、Camel プリミティブの標準的な注入に使用できる CDI 修飾子に対応する注釈リテラルを提供します。これらの注釈リテラルは、レイジーインジェクション/プログラムによるルックアップを実行するための CDI エントリーポイントである **javax.enterprise.inject.Instance** インターフェイスと組み合わせて使用できます。

たとえば、**@Uri** 修飾子に提供されたアノテーションリテラルを使用して、**ProducerTemplate** Bean などの Camel プリミティブを遅延検索できます。

```
@Any
@Inject
Instance<ProducerTemplate> producers;

ProducerTemplate inbound = producers
    .select(Uri.Literal.of("direct:inbound"))
    .get();
```

または **エンドポイント** Bean の場合は、次のようになります。

```
@Any
@Inject
Instance<Endpoint> endpoints;

MockEndpoint outbound = endpoints
    .select(MockEndpoint.class, Uri.Literal.of("mock:outbound"))
    .get();
```

同様に、**@ContextName** 修飾子に提供されたアノテーションリテラルを使用して、**CamelContext** Bean を遅延検索できます。

```
@Any
@Inject
Instance<CamelContext> contexts;

CamelContext context = contexts
    .select(ContextName.Literal.of("foo"))
    .get();
```

Camel コンテキストタイプに基づいて選択を絞り込むこともできます。

```
@Any
@Inject
Instance<CamelContext> contexts;

// Refine the selection by type
Instance<DefaultCamelContext> context = contexts.select(DefaultCamelContext.class);

// Check if such a bean exists then retrieve a reference
if (!context.isUnsatisfied())
    context.get();
```

または、選択した Camel コンテキストを反復処理することもできます。たとえば、次のようになります。

```
@Any
@Inject
Instance<CamelContext> contexts;

for (CamelContext context : contexts)
    context.setUseBreadcrumb(true);
```

57.15. MAVEN ARCHETYPE

利用可能な Camel Maven archetypes の中で、提供されている **camel-archetype-cdi** を使用して Camel CDI Maven プロジェクトを生成できます。

```
mvn archetype:generate -DarchetypeGroupId=org.apache.camel.archetypes -
DarchetypeArtifactId=camel-archetype-cdi
```

57.16. サポートされるコンテナー

Camel CDI コンポーネントは、CDI 1.0、CDI 1.1、および CDI 1.2 準拠のランタイムと互換性があります。次のランタイムに対して正常にテストされています。

Container	Version	ランタイム
Weld SE	**1.1.28.Final**	CDI 1.0 / Java SE 7
OpenWebBeans	**1.2.7**	CDI 1.0 / Java SE 7
Weld SE	**2.4.2.Final**	CDI 1.2 / Java SE 7
OpenWebBeans	**1.7.2**	CDI 1.2 / Java SE 7
WildFly	**8.2.1.Final**	CDI 1.2 / Java EE 7
WildFly	**9.0.1.Final**	CDI 1.2 / Java EE 7
WildFly	**10.1.0.Final**	CDI 1.2 / Java EE 7

57.17. 例

次の例は、Camel プロジェクトの **examples** ディレクトリーにあります。

例	説明
camel-example-cdi	コンポーネント、エンドポイント、および Bean を設定するために CDI を使用して Camel を操作する方法を示します。
camel-example-cdi-kubernetes	Camel、CDI、および Kubernetes 間の統合を示します
camel-example-cdi-metrics	Camel、Dropwizard Metrics、および CDI の統合を示します。
camel-example-cdi-properties	設定プロパティーのための Camel、DeltaSpike、および CDI 間の統合を示します。
camel-example-cdi-osgi	PAX CDI を使用して OSGi コンテナー内で実行できる SJMS コンポーネントを使用する CDI アプリケーション
camel-example-cdi-rest-servlet	依存性注入フレームワークとして CDI を使用する Web アプリケーションで使用されている Camel REST DSL を示しています。
camel-example-cdi-test	Camel と CDI の統合の一部として提供されるテスト機能を示します。
camel-example-cdi-xml	Camel XML 設定ファイルを Camel CDI アプリケーションに使用する方法を示します。
camel-example-swagger-cdi	CDI で REST DSL と Swagger Java を使用する例
camel-example-widget-gadget-cdi	CDI 依存性インジェクションを使用して Java で実装された EIP ブックのウィジェットとガジェットのユースケース

57.18. 関連項目

- Camel CDI テスト
- CDI 仕様の Web サイト
- CDI エコシステム
- Weld ホームページ
- OpenWebBeans home page
- CDI と Camel をさらに活用 (Camel CDI セクションを参照)

57.19. {WILDFLY-CAMEL} での EAR デプロイメント用の CAMEL CDI

{wildfly-camel} での Camel CDI EAR デプロイメントは、標準の WAR または JAR デプロイメントと比較して、クラスおよびリソースのロード動作にいくつかの違いがあります。

{wildfly} は、EAR デプロイメント ClassLoader を使用して Weld をブートストラップします。{wildfly} は、単一の CDI 拡張機能のみが作成され、すべての EAR サブデプロイメントによって共有されることも義務付けています。

これにより、EAR デプロイメント ClassLoader を使用して自動設定 CDI Camel コンテキストが生成され、クラスとリソースが動的にロードされます。デフォルトでは、この ClassLoader は EAR サブデプロイメント内のリソースにアクセスできません。

EAR デプロイメントの場合、自動設定された CDI Camel コンテキストの使用を避け、**RouteBuilder** クラスに **@ContextName** のアノテーションを付けるか、**@ImportResource** アノテーションまたは CDI producer メソッドとフィールドを介して **CamelContext** を作成することをお勧めします。これは、{wildfly-camel} が Camel で使用する正しい ClassLoader を決定するのに役立ちます。

第58章 CHRONICLE エンジンコンポーネント

Camel バージョン 2.18 以降で利用可能

camel クロニクルエンジンコンポーネントを使用すると、OpenHFT の Chronicle-Engine のパワーを活用できます

58.1. URI 形式

```
chronicle-engine:addresses/path[?options]
```

58.2. URI オプション

Chronicle Engine コンポーネントにはオプションがありません。

Chronicle Engine エンドポイントは、URI 構文を使用して設定されます。

```
chronicle-engine:addresses/path
```

パスおよびクエリーパラメーターを使用します。

58.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
addresses	**必須** エンジンアドレス。複数のアドレスはコンマで区切ることができます。		String
path	**必須** エンジンパス		String

58.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
action (common)	実行するデフォルトのアクションで、有効な値は次のとおりです: - PUBLISH - PPUBLISH_AND_INDEX - PPUT - PGET_AND_PUT - PPUT_ALL - PPUT_IF_ABSENT - PGET - PGET_AND_REMOVE - PREMOVE - PIS_EMPTY - PSIZE		String
clusterName (common)	キューのクラスター名		String
filteredMapEvents (common)	ファイラーへの Map イベントタイプのコンマ区切りリスト。有効な値は、INSERT、UPDATE、REMOVE です。		String

名前	説明	デフォルト	タイプ
persistent (common)	データの永続性を有効/無効にする	true	boolean
subscribeMapEvents (common)	コンシューマーが Map イベントをサブスクライブする必要があるかどうかを設定します。デフォルトは true です。	true	boolean
subscribeTopicEvents (common)	コンシューマーが TopicEvents にサブスクライブする必要があるかどうかを設定します。デフォルトは false です。	false	boolean
subscribeTopologicalEvents (common)	コンシューマーが TopologicalEvents にサブスクライブする必要があるかどうかを設定します。デフォルトは false です。	false	boolean
wireType (common)	使用するワイヤータイプ。デフォルトはバイナリーワイヤーです。	BINARY	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第59章 CHUNK コンポーネント

Camel バージョン 2.15 以降で利用可能

chunk: コンポーネントを使用すると、Chunk テンプレートを使用してメッセージを処理できます。これは、テンプレーティングを使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
<groupId>org.apache.camel</groupId>
<artifactId>camel-chunk</artifactId>
<version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

59.1. URI 形式

```
chunk:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI です。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

59.2. オプション

Chunk コンポーネントにはオプションがありません。

Chunk エンドポイントは、URI 構文を使用して設定されます。

```
chunk:resourceUri
```

パスおよびクエリーパラメーターを使用します。

59.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

59.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding (producer)	本文のエンコーディングを定義する		String
extension (producer)	テンプレートのファイル拡張子を定義する		String
themeFolder (producer)	スキャンするテーマフォルダーを定義する		String
themeLayer (producer)	精緻化するテーマレイヤーを定義する		String
themeSubfolder (producer)	スキャンするテーマのサブフォルダーを定義します		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

Chunk コンポーネントは、拡張子が **.chtml** または _.cxml の **テーマ** フォルダーで特定のテンプレートを探します。_ 別のフォルダーまたは拡張子を指定する必要がある場合は、上記の特定のオプションを使用する必要があります。

59.3. CHUNK コンテキスト

Camel は Chunk コンテキスト (単なる **Map**) で交換情報を提供します。**Exchange** は次のように転送されます。

key	value
exchange	**Exchange** 自体。
exchange.properties	**Exchange** プロパティー。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト
request	IN メッセージ

key	value
body	In メッセージボディー
response	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。

59.4. 動的テンプレート

Camel は、テンプレートまたはテンプレートコンテンツ自体の異なるリソースのロケーションを定義できる 2 つのヘッダーを提供します。これらのヘッダーのいずれかが設定されている場合、Camel はエンドポイントで設定されたリソースに対してこれを使用します。これにより、実行時に動的なテンプレートを提供できます。

ヘッダー	タイプ	説明	サポートバージョン
ChunkConstants.CHUNK_RESOURCE_URI	String	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。	
ChunkConstants.CHUNK_TEMPLATE	String	設定されたエンドポイントの代わりに使用するテンプレート。	

59.5. サンプル

たとえば、次のようなものを使用できます。

```
from("activemq:My.Queue").
to("chunk:template");
```

Chunk テンプレートを使用して、InOut メッセージエクスチェンジ (**JMSReplyTo** ヘッダーがある場合) のメッセージに対するレスポンスを作成します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、次を使用できます。

```
from("activemq:My.Queue").
to("chunk:template").
to("activemq:Another.Queue");
```

コンポーネントがヘッダーを介して動的に使用する必要があるテンプレートを指定することが可能です。たとえば、次のようになります。

```
from("direct:in").
setHeader(ChunkConstants.CHUNK_RESOURCE_URI).constant("template").
to("chunk:dummy");
```

Chunk コンポーネントオプションの使用例:

```
from("direct:in").
to("chunk:file_example?themeFolder=template&themeSubfolder=subfolder&extension=chunk");
```

この例では、Chunk コンポーネントは、フォルダー **template/subfolder.** でファイル **file_example.chunk** を探します。

59.6. 電子メールのサンプル

このサンプルでは、注文確認電子メールに Chunk テンプレーティングを使用します。電子メールテンプレートは、次のように Chunk に配置されます。

```
Dear {$headers.lastName}, {$headers.firstName}

Thanks for the order of {$headers.item}.

Regards Camel Riders Bookstore
{$body}
```

59.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第60章 CLASS コンポーネント

Camel バージョン 2.4 以降で利用可能

class: コンポーネントは、Bean を Camel メッセージエクスチェンジにバインドします。 Bean コンポーネントと同じように機能しますが、レジストリーから Bean を検索する代わりに、クラス名に基づいて Bean を作成します。

60.1. URI 形式

```
class:className[?options]
```

className は、作成して Bean として使用する完全修飾クラス名です。

60.2. オプション

Class コンポーネントにはオプションがありません。

Class エンドポイントは、URI 構文を使用して設定されます。

```
class:beanName
```

パスおよびクエリーパラメーターを使用します。

60.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
beanName	**必須**: 呼び出す Bean の名前を設定します。		String

60.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
method (producer)	Bean で呼び出すメソッドの名前を設定します。		String
cache (advanced)	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリー内の Bean がシングルトンスコープとして定義されている場合、キャッシュを有効にできます。	false	boolean

名前	説明	デフォルト	タイプ
multiParameterArray (advanced)	**非推奨** メッセージ本文から渡されるパラメーターの処理方法。true の場合、メッセージボディーはパラメーターの配列である必要があります。注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。非推奨の注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。	false	boolean
parameters (advanced)	Bean の追加プロパティーの設定に使用します		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

60.3. 使用

class コンポーネントを単に Bean コンポーネントと同じように使用しますが、代わりに完全修飾クラス名を指定します。
たとえば、**MyFooBean** を使用するには、次のようにする必要があります。

```
    from("direct:start").to("class:org.apache.camel.component.bean.MyFooBean").to("mock:result");
```

hello など、**MyFooBean** で呼び出すメソッドを指定することもできます。

```
    from("direct:start").to("class:org.apache.camel.component.bean.MyFooBean?
method=hello").to("mock:result");
```

60.4. 作成したインスタンスにプロパティーを設定する

エンドポイント URI では、作成されたインスタンスに設定するプロパティーを指定できます (たとえば、**setPrefix** メソッドがある場合)。

```
// Camel 2.17 onwards
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?bean.prefix=Bye")
    .to("mock:result");

// Camel 2.16 and older
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?prefix=Bye")
    .to("mock:result");
```

また、**#** 構文を使用して、レジストリーで検索するプロパティーを参照することもできます。

```
// Camel 2.17 onwards
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?bean.cool=#foo")
    .to("mock:result");

// Camel 2.16 and older
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?cool=#foo")
    .to("mock:result");
```

ID **foo** でレジストリーから Bean を検索し、 **MyPrefixBean** クラスの作成されたインスタンスで **setCool** メソッドを呼び出します。

ヒント: **class** コンポーネントはほとんど同じように機能するため、Bean コンポーネントで詳細を参照してください。

60.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Bean
- Bean バインディング
- Bean インテグレーション

第61章 CMIS コンポーネント

Camel バージョン 2.11 以降で利用可能

cmis コンポーネントは Apache Chemistry クライアント API を使用し、CMIS 準拠のコンテンツリポジトリーにノードを追加したり、そこからノードを読み取ったりできるようにします。

61.1. URI 形式

```
cmis://cmisServerUrl[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

61.2. CMIS オプション

CMIS コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
sessionFacadeFactory (common)	カスタム CMISSessionFacadeFactory を使用して CMISSessionFacade インスタンスを作成する場合		CMISSessionFacade Factory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CMIS エンドポイントは、URI 構文を使用して設定されます。

```
cmis:cmsUrl
```

パスおよびクエリーパラメーターを使用します。

61.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cmsUrl	**必須** cmis リポジトリーへの URL		String

61.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
pageSize (common)	ページごとに取得するノードの数	100	int
readContent (common)	true に設定すると、プロパティーに加えてドキュメントノードのコンテンツが取得されます。	false	boolean
readCount (common)	読み取るノードの最大数		int
repositoryId (common)	使用するリポジトリーの ID。指定しない場合、最初に使用可能なリポジトリーが使用されます		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
query (consumer)	リポジトリーに対して実行する cmis クエリー。指定しない場合、コンシューマーはコンテンツツリーを再帰的に反復することにより、コンテンツリポジトリーからすべてのノードを取得します。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
queryMode (producer)	true の場合、メッセージボディーから cmis クエリーを実行して結果を返します。それ以外の場合は、cmis リポジトリーにノードを作成します	false	boolean
sessionFacadeFactory (advanced)	カスタム CMISSessionFacadeFactory を使用して CMISSessionFacade インスタンスを作成する場合		CMISSessionFacade Factory
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
password (security)	cmis リポジトリーのパスワード		String
username (security)	cmis リポジトリーのユーザー名		文字列

61.3. 使用方法

61.3.1. プロデューサーによって評価されるメッセージヘッダー

ヘッダー	デフォルト値	説明
CamelCMISFolderPath	/	実行中に使用する現在のフォルダー。指定しない場合は、ルートフォルダーが使用されます
CamelCMISRetrieveContent	**false**	**queryMode** では、このヘッダーはプロデューサーに強制的にドキュメントノードのコンテンツを取得させます。
CamelCMISReadSize	**0**	読み取るノードの最大数。
cmis:path	**null**	**CamelCMISFolderPath** が設定されていない場合、このcmisプロパティからノードのパスを見つけようとし、それが名前である場合
cmis:name	**null**	**CamelCMISFolderPath** が設定されていない場合、この cmis プロパティーからノードのパスを見つけようとします。
cmis:objectTypeId	**null**	ノードのタイプ
cmis:contentStreamMimeType	**null**	ドキュメントに設定する MIME タイプ

61.3.2. Producer 操作のクエリー中に設定されるメッセージヘッダー

ヘッダー	タイプ	説明
CamelCMISResultCount	**Integer**	クエリーから返されたノードの数。

メッセージ本文にはマップのリストが含まれます。マップの各エントリーは cmis プロパティーとその値です。**CamelCMISRetrieveContent** ヘッダーが true に設定されている場合、キー **CamelCMISContent** を持つマップ内の 1 つの追加エントリーには、ノードのドキュメントタイプの **InputStream** が含まれます。

61.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cmis</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version } は Camel の実際のバージョン (2.11 以降) に置き換える必要があります。

61.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第62章 CM SMS ゲートウェイコンポーネント

Camel バージョン 2.18 以降で利用可能

Camel-Cm-Sms は、CM SMS Gateway (https://www.cmtelecom.com) の Apache Camel コンポーネントです。

CM SMS API をキャメルコンポーネントとしてアプリケーションに統合できます。

有効なアカウントが必要です。詳細については、CM Telecom を参照してください。

```
cm-sms://sgw01.cm.nl/gateway.ashx?
defaultFrom=DefaultSender&defaultMaxNumberOfParts=8&productToken=xxxxx
```

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-cm-sms</artifactId>
 <version>x.x.x</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

62.1. オプション

CM SMS Gateway コンポーネントにはオプションがありません。

CM SMS ゲートウェイエンドポイントは、URI 構文を使用して設定されます。

```
cm-sms:host
```

パスおよびクエリーパラメーターを使用します。

62.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** SMS プロバイダー HOST とスキーム		String

62.1.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
defaultFrom (producer)	これは差出人の名前です。最大長は 11 文字です。		文字列

名前	説明	デフォルト	タイプ
defaultMaxNumberOfParts (producer)	マルチパートメッセージの場合は、最大数を強制します。メッセージは切り詰めることができます。技術的には、ゲートウェイは最初にメッセージが 160 文字を超えるかどうかをチェックします。160 文字を超える場合、メッセージはこれらのパラメーターによって制限された複数の 153 文字の部分に分割されます。	8	int
productToken (producer)	**必須** 使用する一意のトークン		文字列
testConnectionOnStartup (producer)	起動時に SMS ゲートウェイへの接続をテストするかどうか	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

62.2. 例

このプロジェクト を試して、camel-cm-sms をキャメルルートに統合する方法を確認できます。

第63章 COAP コンポーネント

Camel バージョン 2.16 以降で利用可能

Camel-CoAP は、マシン間の操作のための軽量な REST タイプのプロトコルである CoAP を操作できるようにする Apache Camel コンポーネントです。CoAP、Constrained Application Protocol は、制約のあるノードと制約のあるネットワークで使用するための特殊な Web 転送プロトコルであり、RFC 7252 に基づいています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-coap</artifactId>
 <version>x.x.x</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

63.1. オプション

CoAP コンポーネントにはオプションがありません。

CoAP エンドポイントは、URI 構文を使用して設定されます。

```
coap:uri
```

パスおよびクエリーパラメーターを使用します。

63.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
uri	CoAP エンドポイントの URI		URI

63.1.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
coapMethodRestrict (consumer)	CoAP コンシューマーが BIND するメソッドのコンマ区切りリスト。デフォルトは、すべてのメソッド (DELETE、GET、POST、PUT) に対して BIND です。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

63.2. メッセージヘッダー

名前	タイプ	説明
CamelCoapMethod	**String**	ターゲット CoAP サーバー URI を呼び出すときに CoAP プロデューサーが使用する要求メソッド。有効なオプションは、DELETE、GET、PING、POST、および PUT です。
CamelCoapResponseCode	**String**	外部サーバーから送信された CoAP 応答コード。各コードの意味の詳細については、RFC 7252 を参照してください。
CamelCoapUri	**String**	呼び出す CoAP サーバーの URI。エンドポイントで直接設定された既存の URI をオーバーライドします。

63.2.1. CoAP プロデューサーリクエストメソッドの設定

次のルールは、CoAP プロデューサーがターゲット URI を呼び出すために使用するリクエストメソッドを決定します。

1. **CamelCoapMethod** ヘッダーの値

2. ターゲット CoAP サーバー URI でクエリー文字列が提供されている場合は **GET**。
3. メッセージエクスチェンジボディーが null でない場合は **POST**。
4. それ以外の場合は **GET**。

第64章 CONSTANT 言語

Camel バージョン 1.5 以降で利用可能

定数式言語は、定数文字列を式の型として指定する方法にすぎません。

注記

これは、ルートの開始時に一度だけ設定される固定定数値であるため、ルーティング中に動的な値が必要な場合は、この値は使用しないでください。

64.1. CONSTANT オプション

Constant 言語は、以下に示す 1 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

64.2. 使用例

Spring DSL の setHeader 要素は、次のような定数式を利用できます。

```
<route>
  <from uri="seda:a"/>
  <setHeader headerName="theHeader">
    <constant>the value</constant>
  </setHeader>
  <to uri="mock:b"/>
</route>
```

この場合、seda:a エンドポイントからのメッセージには、定数値値に設定された theHeader ヘッダーがあります。

Java DSL を使用した同じ例:

```
from("seda:a")
  .setHeader("theHeader", constant("the value"))
  .to("mock:b");
```

64.3. 依存関係

Constant 言語は **camel-core** の一部です。

第65章 COMETD コンポーネント

Camel バージョン 2.0 以降で利用可能

cometd: コンポーネントは、cometd/bayeux プロトコル の jetty 実装を操作するためのトランスポートです。
このコンポーネントを dojo ツールキットライブラリーと組み合わせて使用すると、AJAX ベースのメカニズムを使用して Camel メッセージをブラウザーに直接プッシュできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cometd</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

65.1. URI 形式

```
cometd://host:port/channelName[?options]
```

channelName は、Camel エンドポイントがサブスクライブできるトピックを表します。

65.2. 例

```
cometd://localhost:8080/service/mychannel
cometds://localhost:8443/service/mychannel
```

ここで、**cometds:** は SSL 設定のエンドポイントを表します。

65.3. オプション

CometD コンポーネントは、以下に記載される 8 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
sslKeyPassword (security)	SSL を使用する場合のキーストアのパスワード。		String
sslPassword (security)	SSL 使用時のパスワード。		String
sslKeystore (security)	キーストアへのパス。		String
securityPolicy (security)	カスタム設定の SecurityPolicy を使用して承認を制御するには		SecurityPolicy

名前	説明	デフォルト	タイプ
extensions (common)	着信および発信要求を変更できるカスタム BayeuxServer.Extension のリストを使用するには。		List
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CometD エンドポイントは、URI 構文を使用して設定されます。

```
cometd:host:port/channelName
```

パスおよびクエリーパラメーターを使用します。

65.3.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** ホスト名		String
port	**必須** ホストのポート番号		int
channelName	**必須** channelName は、Camel エンドポイントがサブスクライブできるトピックを表します。		String

65.3.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
allowedOrigins (common)	crosssOriginFilterOn が true の場合、クロスをサポートするオリジンドメイン	*	String

名前	説明	デフォルト	タイプ
baseResource (common)	Web リソースまたはクラスパスのルートディレクトリー。コンポーネントがリソースをファイルシステムまたはクラスパスからロードするかどうかに応じて、protocol file: または classpath: を使用します。リソースが jar にパッケージ化されている OSGI デプロイメントには、クラスパスが必要です。		String
crossOriginFilterOn (common)	true の場合、サーバーはクロスドメインフィルタリングをサポートします。	false	boolean
filterPath (common)	crosssOriginFilterOn が true の場合、filterPath は CrossOriginFilter によって使用されます。		String
interval (common)	クライアント側のポーリングタイムアウト (ミリ秒)。クライアントが再接続の間に待機する時間		int
jsonCommented (common)	true の場合、サーバーはコメントでラップされた JSON を受け入れ、コメントでラップされた JSON を生成します。これは、Ajax ハイジャックに対する防御です。	true	boolean
logLevel (common)	ロギングレベル0=none、1=info、2=debug。	1	int
maxInterval (common)	クライアント側の最大ポーリングタイムアウト (ミリ秒)。この時間内に接続が受信されない場合、クライアントは削除されます。	30000	int
multiFrameInterval (common)	同じブラウザーから複数の接続が検出された場合の、クライアント側のポーリングタイムアウト。	1500	int
timeout (common)	サーバー側のポーリングタイムアウト (ミリ秒)。これは、サーバーが応答する前に再接続要求を保持する時間です。	240000	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
sessionHeadersEnabled (consumer)	着信要求の Camel メッセージを作成するときに、サーバーセッションヘッダーを Camel メッセージに含めるかどうか。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
disconnectLocalSession (producer)	メッセージをチャネルにパブリッシュした後、ローカルセッションを切断するかどうか。ローカルセッションはデフォルトで CometD によってスイープされないため、ローカルセッションを切断する必要があり、メモリーが不足する可能性があります。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

パラメーターを渡す方法の例を次に示します

ファイルの場合 (Web アプリケーションディレクトリーにある webapp リソースの場合 -→ cometd://localhost:8080?resourceBase=file./webapp
クラスパスの場合 (たとえば、Web リソースが webapp フォルダー内にパッケージ化されている場合 -→ cometd://localhost:8080?resourceBase=classpath:webapp

65.4. 認証

Camel 2.8 から利用可能

ここに記載されている ように、カスタム **SecurityPolicy** と **Extension's to the `CometdComponent** に設定して、認証を使用できるようにすることができます

65.5. COMETD コンポーネントの SSL の設定

65.5.1. JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Cometd コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例

は、Cometd コンポーネントでユーティリティーを使用する方法を示しています。CometdComponent で SSL を設定する必要があります。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
scp.setTrustManagers(tmp);

CometdComponent commetdComponent = getContext().getComponent("cometds",
CometdComponent.class);
commetdComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
   <camel:trustManagers>
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
  </camel:sslContextParameters>...

  <bean id="cometd" class="org.apache.camel.component.cometd.CometdComponent">
    <property name="sslContextParameters" ref="sslContextParameters"/>
  </bean>
...
  <to uri="cometds://127.0.0.1:443/service/test?baseResource=file:./target/test-
classes/webapp&timeout=240000&interval=0&maxInterval=30000&multiFrameInterval=1500&jsonCom
mented=true&logLevel=2"/>...
```

65.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド

第66章 CONSUL コンポーネント

Camel バージョン 2.18 以降で利用可能

Consul コンポーネントは、アプリケーションを Consul と統合するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-consul</artifactId>
  <version>${camel-version}</version>
</dependency>
```

66.1. URI 形式

```
consul://domain?[options]
```

URI には、次の形式でクエリーオプションを追加できます。

```
?option=value&option=value&...
```

66.2. オプション

Consul コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
url (common)	Consul エージェントの URL		String
datacenter (common)	データセンター		String
sslContextParameters (common)	org.apache.camel.util.jsse.SSLContextParameters インスタンスを使用した SSL 設定。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
aclToken (common)	Consul で使用する ACL トークンを設定します		String
userName (common)	Basic 認証に使用するユーザー名を設定します		String
password (common)	基本認証に使用するパスワードを設定します		String

名前	説明	デフォルト	タイプ
configuration (advanced)	エンドポイント間で共有される共通設定を設定します		ConsulConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Consul エンドポイントは、URI 構文を使用して設定されます。

```
consul:apiEndpoint
```

パスおよびクエリーパラメーターを使用します。

66.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
apiEndpoint	**必須** API エンドポイント		String

66.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

66.3. ヘッダー

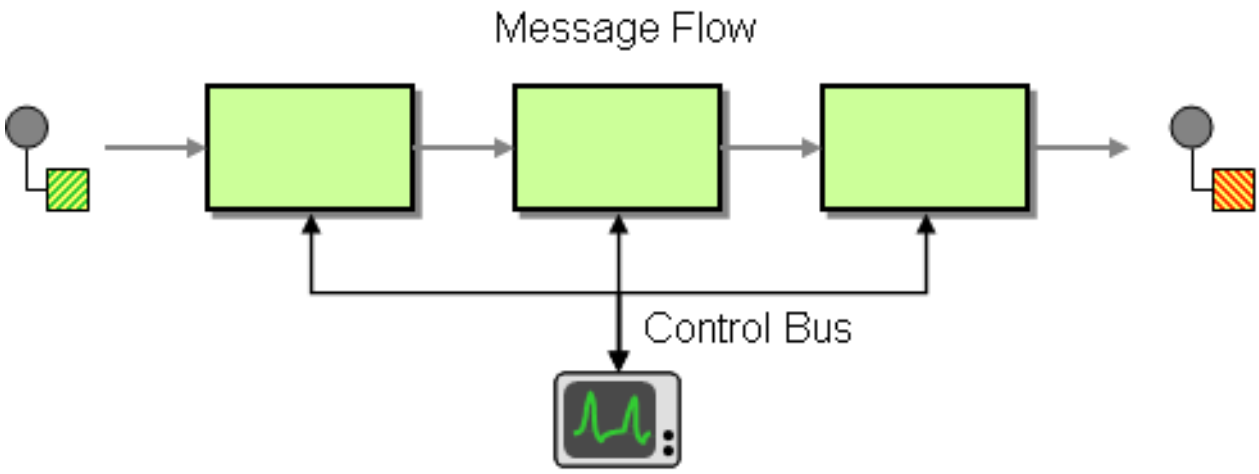
名前	タイプ	説明
Camel Consul Action	String	プロデューサーのアクション
Camel Consul Key	String	アクションが適用されるキー
Camel Consul EventId	String	イベント ID (コンシューマーのみ)
Camel Consul EventName	String	イベント名 (コンシューマーのみ)
Camel Consul EventLTime	Long	イベント LTime
Camel Consul NodeFilter	String	ノードフィルター
Camel Consul TagFilter	String	タグフィルター
Camel Consul SessionFilter	String	セッションフィルター

名前	タイプ	説明
Camel Consul Version	int	データのバージョン
Camel Consul Flags	Long	値に関連付けられたフラグ
Camel Consul CreateIndex	Long	エントリーがいつ作成されたかを表す内部インデックス値
Camel Consul LockIndex	Long	このキーがロックで正常に取得された回数
Camel Consul ModifyIndex	Long	このキーを変更した最後のインデックス
Camel Consul Options	Object	リクエストに関連付けられたオプション
Camel Consul Result	boolean	レスポンスに結果がある場合は true
Camel Consul Session	String	セッション ID
Camel Consul ValueAsString	boolean	Consul ie の KV エンドポイントから取得した値を文字列に変換します。

第67章 コントロールバスコンポーネント

Camel バージョン 2.11 以降で利用可能

EIP パターンからの 制御バス により、統合システムをフレームワーク内から監視および管理できます。



コントロールバスを使用して、エンタープライズ統合システムを管理します。コントロールバスは、アプリケーションデータで使用されるのと同じメッセージングメカニズムを使用しますが、メッセージフローに含まれるコンポーネントの管理に関連するデータを送信するために別のチャネルを使用します。

Camel では、JMX を使用して、または **CamelContext** から、または **org.apache.camel.api.management** パッケージから Java API を使用して、またはここに例があるイベント通知機能を使用して、管理および監視できます。

Camel 2.11 以降、それに応じて反応するコントロールバスエンドポイントにメッセージを送信できる新しい ControlBus コンポーネント を導入しました。

67.1. CONTROLBUS コンポーネント

Camel 2.11 から利用可能

controlbus: コンポーネントは、Control Bus EIP パターンに基づいて Camel アプリケーションを簡単に管理できるようにします。たとえば、エンドポイントにメッセージを送信することで、ルートのライフサイクルを制御したり、パフォーマンス統計を収集したりできます。

```
controlbus:command[?options]
```

command には、使用するコマンドのタイプを識別するための任意の文字列を指定できます。

67.2. コマンド

コマンド	説明
route	**routeId** と **action** パラメーターを使用してルートを制御します。

コマンド	説明
言語	メッセージボディーの評価に使用する 言語 を指定できます。評価の結果があれば、その結果がメッセージ本文に入れられます。

67.3. オプション

コントロールバスコンポーネントにはオプションがありません。

Control Bus エンドポイントは、URI 構文を使用して設定されます。

```
controlbus:command:language
```

パスおよびクエリーパラメーターを使用します。

67.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
command	**必要な** コマンドは、ルートまたは言語のいずれかです		String
言語	メッセージ本文の評価に使用する言語の名前を指定できます。評価の結果があれば、その結果がメッセージ本文に入れられます。		言語

67.3.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
action (producer)	開始、停止、またはステータスのいずれかのアクションを示す。ルートを開始または停止するか、ルートのステータスをメッセージ本文の出力として取得します。Camel 2.11.1 以降では、サスペンドとレジュームを使用して、ルートをサスペンドまたはレジュームできます。また、Camel 2.11.1 以降では、stats を使用して、パフォーマンスの統計を XML 形式で取得できます。routeId オプションを使用して、パフォーマンス統計を取得するルートを定義できます。routeId が定義されていない場合は、CamelContext 全体の統計を取得します。再起動アクションはルートを再起動します。		String

名前	説明	デフォルト	タイプ
async (producer)	コントロールバスタスクを非同期で実行するかどうか。重要: このオプションを有効にすると、タスクの結果は Exchange に設定されません。これは、タスクを同期的に実行する場合にのみ可能です。	false	boolean
loggingLevel (producer)	タスクが完了したとき、またはタスクの処理中に例外が発生した場合にログに使用されるログレベル。	INFO	LoggingLevel
restartDelay (producer)	ルートを再起動するときに使用するミリ単位の遅延。	1000	int
routeId (producer)	ID でルートを指定します。特別なキーワード current は、現在のルートを示します。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

67.4. ルートコマンドの使用

route コマンドを使用すると、特定のルートで一般的なタスクを非常に簡単に実行できます。たとえば、ルートを開始するには、このエンドポイントに空のメッセージを送信できます。

```
template.sendBody("controlbus:route?routeId=foo&action=start", null);
```

ルートのステータスを取得するには、次のようにします。

```
String status = template.requestBody("controlbus:route?routeId=foo&action=status", null,
String.class);
```

67.5. パフォーマンス統計の取得

Camel 2.11.1 以降で利用可能

これには、JMX を有効にする必要があります (デフォルトで)。そうすると、ルートごと、または CamelContext のパフォーマンス統計を取得できます。たとえば、foo という名前のルートの統計を取得するには、次のようにします。

```
String xml = template.requestBody("controlbus:route?routeId=foo&action=stats", null, String.class);
```

返される統計は XML 形式です。**ManagedRouteMBean** で **dumpRouteStatsAsXml** 操作を使用して JMX から取得できるデータと同じです。

CamelContext 全体の統計を取得するには、以下に示すように routeId パラメーターを省略します。

```
String xml = template.requestBody("controlbus:route?action=stats", null, String.class);
```

67.6. シンプルな言語を使用する

コントロールバスで Simple 言語を使用できます。たとえば、特定のルートを停止するには、次のメッセージを含む **"controlbus:language:simple"** エンドポイントにメッセージを送信できます。

```
template.sendBody("controlbus:language:simple", "${camelContext.stopRoute('myRoute')}");
```

これは無効な操作であるため、結果は返されません。ただし、ルートステータスが必要な場合は、次のことができます。

```
String status = template.requestBody("controlbus:language:simple",
"${camelContext.getRouteStatus('myRoute')}", String.class);
```

route コマンドを使用してルートのライフサイクルを制御する方が簡単です。**language** コマンドを使用すると、Groovy などのより強力な言語スクリプトを実行したり、Simple 言語を拡張したりできます。

たとえば、Camel 自体をシャットダウンするには、次のようにします。

```
template.sendBody("controlbus:language:simple?async=true", "${camelContext.stop()}");
```

async=true を使用して Camel を非同期的に停止します。そうしないと、制御バスコンポーネントに送信したメッセージを処理中に Camel を停止しようとすることになります。

ヒント

Groovy などの他の言語も使用できます。

第68章 COUCHBASE コンポーネント

Camel バージョン 2.19 以降で利用可能

couchbase: コンポーネントを使用すると、CouchBase インスタンスをメッセージのプロデューサーまたはコンシューマーとして扱うことができます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-couchbase</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

68.1. URI 形式

```
couchbase:url
```

68.2. オプション

Couchbase コンポーネントにはオプションがありません。

Couchbase エンドポイントは、URI 構文を使用して設定されます。

```
couchbase:protocol:hostname:port
```

パスおよびクエリーパラメーターを使用します。

68.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
protocol	**必須** 使用するプロトコル		String
hostname	**必須** 使用するホスト名		String
port	使用するポート番号	8091	int

68.2.2. クエリーパラメーター(47 個のパラメーター):

名前	説明	デフォルト	タイプ
bucket (common)	使用するバケット		String

名前	説明	デフォルト	タイプ
key (common)	使用する鍵		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerProcessedStrategy (consumer)	使用するコンシューマー処理済み戦略を定義する	none	String
descending (consumer)	この操作が降順かどうかを定義します	false	boolean
designDocumentName (consumer)	使用する設計ドキュメント名	beer	String
limit (consumer)	使用する出力制限	-1	int
rangeEndKey (consumer)	終了キーの範囲を定義する		String
rangeStartKey (consumer)	開始キーの範囲を定義する		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
skip (consumer)	使用するスキップを定義する	-1	int
viewName (consumer)	使用するビュー名	brewery_beers	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
autoStartIdForInserts (producer)	挿入操作を行っているときに自動開始 ID が必要かどうかを定義します	false	boolean
operation (producer)	やるべき操作	CCB_PUT	String
persistTo (producer)	データを永続化する場所	0	int
producerRetryAttempts (producer)	再試行回数を定義する	2	int
producerRetryPause (producer)	異なる試行間の再試行の一時停止を定義する	5000	int
replicaTo (producer)	データをレプリケートする場所	0	int
startingIdForInsertsFrom (producer)	挿入操作を行っている開始 ID を定義します		long
additionalHosts (advanced)	追加のホスト		String
maxReconnectDelay (advanced)	再接続中の最大遅延を定義する	30000	long
obsPollInterval (advanced)	監視ポーリング間隔を定義する	400	long
obsTimeout (advanced)	監視タイムアウトを定義する	-1	long

名前	説明	デフォルト	タイプ
opQueueMaxBlockTime (advanced)	操作がキューにブロックされている最大時間を定義する	10000	long
opTimeOut (advanced)	操作タイムアウトを定義する	2500	long
readBufferSize (advanced)	バッファーサイズを定義する	16384	int
shouldOptimize (advanced)	可能な場合は最適化を使用するかどうかを定義します	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
timeoutExceptionThreshold (advanced)	タイムアウト例外を出力するためのしきい値を定義します	998	int
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

名前	説明	デフォルト	タイプ
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
password (security)	使用するパスワード		String
username (security)	使用するユーザー名		文字列

第69章 COUCHDB COMPONENT

Camel バージョン 2.11 以降で利用可能

couchdb: コンポーネントを使用すると、CouchDB インスタンスをメッセージのプロデューサーまたはコンシューマーとして扱うことができます。軽量の LightCouch API を使用して、この camel コンポーネントには次の機能があります。

- コンシューマーとして、挿入、更新、および削除のために couch 変更セットを監視し、これらをメッセージとして camel ルートに公開します。
- プロデューサーとして、(DELETE 値を持つ CouchDbMethod を使用して) ドキュメントを couch に保存、更新、および Camel 2.18 から削除できます。
- 複数のインスタンスにわたる複数のデータベースなど、必要な数のエンドポイントをサポートできます。
- 削除のみ、挿入/更新のみ、またはすべて (デフォルト) に対してイベントをトリガーする機能。
- sequenceId、ドキュメントリビジョン、ドキュメント ID、および HTTP メソッドタイプに設定されたヘッダー。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-couchdb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

69.1. URI 形式

```
couchdb:http://hostname[:port]/database?[options]
```

ホスト名 は、実行中の couchdb インスタンスのホスト名です。ポートはオプションで、指定しない場合はデフォルトで 5984 になります。

69.2. オプション

CouchDB コンポーネントにはオプションがありません。

CouchDB エンドポイントは、URI 構文を使用して設定されます。

```
couchdb:protocol:hostname:port/database
```

パスおよびクエリーパラメーターを使用します。

69.2.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
protocol	**必須** データベースとの通信に使用するプロトコル。		String
hostname	**必須** 実行中の couchdb インスタンスのホスト名		String
port	実行中の couchdb インスタンスのポート番号	5984	int
database	**必須** 使用するデータベースの名前		String

69.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
createDatabase (common)	データベースがまだ存在しない場合は作成します	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
deletes (consumer)	ドキュメントの削除はイベントとして公開されます	true	boolean
heartbeat (consumer)	空のメッセージを送信してソケットを維持する頻度 (ミリ秒)	30000	long
since (consumer)	指定された更新シーケンスの直後に変更の追跡を開始します。デフォルトの null は、最新のシーケンスからモニタリングを開始します。		String
style (consumer)	changes 配列で返されるリビジョンの数を指定します。デフォルトの main_only は、現在の優勝リビジョンのみを返します。all_docs は、すべてのリーフリビジョンを返します (競合および削除された以前の競合を含む)。	main_only	String
updates (consumer)	ドキュメントの挿入/更新はイベントとして公開されます	true	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
password (security)	認証済みデータベースのパスワード		String
username (security)	認証済みデータベースの場合のユーザー名		文字列

69.3. ヘッダー

次のヘッダーは、メッセージの転送中にエクスチェンジに設定されます。

プロパティー	値
CouchDbDatabase	メッセージの送信元データベース
CouchDbSeq	update/delete メッセージの couchdb 変更セットのシーケンス番号
CouchDbId	couchdb のドキュメント ID
CouchDbRev	couchdb のドキュメントのリビジョン
CouchDbMethod	メソッド (削除/更新)

メッセージが受信されると、ヘッダーはコンシューマーによって設定されます。プロデューサーは、挿入/更新が行われると、ダウンストリームプロセッサーのヘッダーも設定します。プロデューサーの前に設定されたヘッダーはすべて無視されます。つまり、たとえば、CouchDbId をヘッダーとして設定すると、挿入の ID として使用されず、ドキュメントの ID が引き続き使用されます。

69.4. メッセージボディー

コンポーネントは、メッセージボディーを挿入するドキュメントとして使用します。ボディーが String のインスタンスである場合、挿入前に GSON オブジェクトにマーシャリングされます。これは、文字列が有効な JSON である必要があり、そうでない場合、挿入/更新が失敗することを意味します。ボディーが com.google.gson.JsonElement のインスタンスである場合は、そのまま挿入されます。そうしないと、プロデューサーはサポートされていないボディータイプの例外を出力します。

69.5. サンプル

たとえば、ポート 9999 でローカルに実行されている CouchDB インスタンスからすべての挿入、更新、および削除を使用する場合は、次のように使用できます。

```
from("couchdb:http://localhost:9999").process(someProcessor);
```

削除のみに関心がある場合は、次を使用できます

```
from("couchdb:http://localhost:9999?updates=false").process(someProcessor);
```

メッセージをドキュメントとして挿入する場合は、エクスチェンジのボディーが使用されます

```
from("someProducingEndpoint").process(someProcessor).to("couchdb:http://localhost:9999")
```

第70章 CASSANDRA CQL COMPONENT

Camel バージョン 2.15 以降で利用可能

Apache Cassandra は、コモディティハードウェアで大量のデータを処理するように設計されたオープンソースの NoSQL データベースです。Amazon の DynamoDB と同様に、Cassandra にはピアツーピアおよびマスターレスアーキテクチャーがあり、単一障害点を回避し、高可用性を保証します。Google の BigTable と同様に、Cassandra データは、Thrift RPC API または CQL と呼ばれる SQL に似た API を介してアクセスできる列ファミリーを使用して構造化されています。

このコンポーネントは、(Thrift API ではなく) CQL3 API を使用して Cassandra 2.0+ を統合することを目的としています。DataStax が提供する Cassandra Java Driver をベースにしています。

Maven ユーザーは、以下の依存関係を **pom.xml** に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cassandraql</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

70.1. URI 形式

エンドポイントは、Cassandra 接続を開始するか、既存のものを使用できます。

URI	説明
cql:localhost/keyspace	単一ホスト、デフォルトポート、通常はテスト用
cql:host1,host2/keyspace	マルチホスト、デフォルトポート
cql:host1,host2:9042/keyspace	マルチホスト、カスタムポート
cql:host1,host2	デフォルトのポートとキースペース
cql:bean:sessionRef	提供されたセッション参照
cql:bean:clusterRef/keyspace	提供されたクラスター参照

Cassandra 接続 (SSL オプション、プーリングオプション、負荷分散ポリシー、再試行ポリシー、再接続ポリシーなど) を微調整するには、独自のクラスターインスタンスを作成し、それを Camel エンドポイントに渡します。

70.2. CASSANDRA オプション

Cassandra CQL コンポーネントにはオプションがありません。

Cassandra CQL エンドポイントは、URI 構文を使用して設定されます。

```
cql:beanRef:hosts:port/keyspace
```

パスおよびクエリーパラメーターを使用します。

70.2.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
beanRef	beanRef は bean:id を使用して定義されます。		String
hosts	ホスト名 cassansdra サーバー。複数のホストはコンマで区切ることができます。		String
port	cassansdra サーバーのポート番号		Integer
keyspace	使用するキースペース		String

70.2.2. クエリーパラメーター (29 個のパラメーター):

名前	説明	デフォルト	タイプ
cluster (common)	Cluster インスタンスを使用するには (通常、このオプションは使用しません)		クラスター
clusterName (common)	Cluster name		String
consistencyLevel (common)	使用する一貫性レベル		ConsistencyLevel
cql (common)	実行する CQL クエリー。キー CamelCqlQuery を持つメッセージヘッダーでオーバーライドできます。		String
loadBalancingPolicy (common)	特定の LoadBalancingPolicy を使用するには。		String
password (common)	セッション認証のパスワード。		String
prepareStatements (Common)	PreparedStatements を使用するか、通常のステートメントを使用するか。	true	boolean
resultSetConversionStrategy (Common)	ResultSet をメッセージ本文 ALL、ONE、LIMIT_10、LIMIT_100... に変換するためのロジックを実装するカスタムクラスを使用するには		String

名前	説明	デフォルト	タイプ
session (Common)	Session インスタンスを使用するには (通常、このオプションは使用しません)。		Session
username (common)	セッション認証のユーザー名。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int

名前	説明	デフォルト	タイプ
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit

名前	説明	デフォルト	タイプ
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

70.3. メッセージ

70.3.1. 着信メッセージ

Camel Cassandra エンドポイントは、クエリーパラメーターとして CQL ステートメントにバインドされる一連の単純なオブジェクト (**Object** または **Object[]** または **Collection<Object>**) を想定しています。メッセージボディーが null または空の場合、CQL クエリーはパラメーターをバインドせずに実行されます。

ヘッダー:

- **CamelCqlQuery** (オプション、**String** または **RegularStatement**): プレーンな String として、または **QueryBuilder** を使用して構築された CQL クエリー。

70.3.2. 送信メッセージ

Camel Cassandra エンドポイントは、**resultSetConversionStrategy** に応じて、1 つまたは複数の Cassandra Row オブジェクトを生成します。

- **List<Row>** (**resultSetConversionStrategy** が **ALL** または **LIMIT_0-9+** の場合)
- **resultSetConversionStrategy** が **ONE** の場合 h Single` Row`
- その他 (**resultSetConversionStrategy** が **ResultSetConversionStrategy** のカスタム実装である場合)

70.4. リポジトリー

Cassandra を使用して、べき等および集約 EIP のメッセージキーまたはメッセージを格納できます。

Cassandra は、まだユースケースをキューに入れるための最適なツールではない可能性があります。Cassandra anti-patterns queues and queue like datasets を参照してください。これらのテーブルに LeveledCompaction と小さな GC 猶予設定を使用して、廃棄された行をすばやく削除できるようにすることをお勧めします。

70.5. 冪等リポジトリー

NamedCassandraIdempotentRepository は、メッセージキーを次のように Cassandra テーブルに格納します。

CAMEL_IDEMPOTENT.cql

```
CREATE TABLE CAMEL_IDEMPOTENT (
```

```
  NAME varchar,   -- Repository name
  KEY varchar,    -- Message key
  PRIMARY KEY (NAME, KEY)
) WITH compaction = {'class':'LeveledCompactionStrategy'}
  AND gc_grace_seconds = 86400;
```

このリポジトリーの実装では、軽量のトランザクション (Compare and Set とも呼ばれます) を使用し、Cassandra 2.0.7+ が必要です。

または、**CassandraIdempotentRepository** には **NAME** 列がなく、別のデータモデルを使用するように拡張できます。

オプション	デフォルト	説明
table	**CAMEL_IDEMPOTENT**	テーブル名
pkColumns	**NAME**,` KEY`	主キー列
name		リポジトリー名、**NAME** 列に使用される値
ttl		重要な生存期間
writeConsistencyLevel		キーの挿入/削除に使用される一貫性レベル: **ANY**、**ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM** ...
readConsistencyLevel		キーの読み取り/チェックに使用される一貫性レベル: **ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM** ...

70.6. 集計リポジトリー

NamedCassandraAggregationRepository は、次のように Cassandra テーブルに相関キーによる交換を格納します。

CAMEL_AGGREGATION.cql

```
CREATE TABLE CAMEL_AGGREGATION (
  NAME varchar,        -- Repository name
  KEY varchar,         -- Correlation id
  EXCHANGE_ID varchar, -- Exchange id
  EXCHANGE blob,       -- Serialized exchange
  PRIMARY KEY (NAME, KEY)
) WITH compaction = {'class':'LeveledCompactionStrategy'}
  AND gc_grace_seconds = 86400;
```

または、**CassandraAggregationRepository** には **NAME** 列がなく、別のデータモデルを使用するように拡張できます。

オプション	デフォルト	説明
table	**CAMEL_AGGREGATION**	テーブル名
pkColumns	**NAME,KEY**	主キー列
exchangeIdColumn	**EXCHANGE_ID**	交換 ID 列
exchangeColumn	**EXCHANGE**	交換内容欄
name		リポジトリー名、**NAME** 列に使用される値
ttl		生存時間の交換
writeConsistencyLevel		交換の挿入/削除に使用される一貫性レベル: **ANY**、**ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM** …
readConsistencyLevel		交換の読み取り/チェックに使用される整合性レベル: **ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM** …

第71章 CRYPTO (JCE) コンポーネント

Camel バージョン 2.3 以降で利用可能

Camel 暗号化エンドポイントと Java の暗号化拡張機能を使用すると、エクスチェンジ用のデジタル署名を簡単に作成できます。Camel は、取引所のワークフローの一部で取引所の署名を作成し、ワークフローの後半で署名を検証するために協調して使用される柔軟なエンドポイントのペアを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

71.1. はじめに

デジタル署名は、非対称暗号技術を使用してメッセージに署名します。(非常に) 高いレベルから、アルゴリズムは、1 つのキーで暗号化されたデータは、もう一方のキーでのみ復号化できるという特別なプロパティーを持つ補完的なキーのペアを使用します。一方の秘密鍵は厳重に保護され、メッセージの署名に使用され、もう一方の公開鍵は、署名されたメッセージの検証に関心のあるすべての人に共有されます。メッセージは、秘密鍵を使用してメッセージのダイジェストを暗号化することによって署名されます。この暗号化されたダイジェストは、メッセージとともに送信されます。一方、検証者はメッセージダイジェストを再計算し、公開鍵を使用して署名内のダイジェストを復号化します。両方のダイジェストが一致する場合、検証者は、秘密鍵の所有者だけが署名を作成した可能性があることを認識します。

Camel は、Java 暗号化拡張機能の署名サービスを使用して、エクスチェンジ署名の作成に必要なすべての重い暗号処理を行います。以下は、暗号化、メッセージダイジェスト、およびデジタル署名の仕組みと、それらを JCE で活用する方法を説明する優れたリソースです。

- Bruce Schneier の応用暗号
- David Hook による Java による暗号化の開始
- 洞察に満ちたウィキペディア Digital_signatures

71.2. URI 形式

前述のように、Camel は署名を作成および検証するための暗号化エンドポイントのペアを提供します

```
crypto:sign:name[?options]
crypto:verify:name[?options]
```

- **crypto:sign は** 署名を作成し、定数 **org.apache.camel.component.crypto.DigitalSignatureConstants.SIGNATURE** によってキー付けされたヘッダーに格納します。つまり、**"CamelDigitalSignature"**.
- **crypto:verify** は、このヘッダーの内容を読み取り、検証計算を行います。

署名と検証のプロセスが正しく機能するためには、共有するキーのペアが必要です。署名には **PrivateKey** が必要で、**PublicKey** (またはそれを含む **Certificate**) が検証されます。JCE を使用する

と、これらのキーペアを生成するのは非常に簡単ですが、通常は KeyStore を使用してキーを格納および共有するのが最も安全です。DSL は、キーの提供方法について非常に柔軟であり、多くのメカニズムを提供します。

通常、**crypto:sign** エンドポイントは 1 つのルートで定義され、補完的な **crypto:verify** は別のルートで定義されることに注意してください。署名と検証の両方を同じように設定する必要があることは言うまでもありません。

71.3. オプション

Crypto (JCE) コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 DigitalSignatureConfiguration を設定として使用するには		デジタル署名の設定
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Crypto (JCE) エンドポイントは、URI 構文を使用して設定されます。

```
crypto:cryptoOperation:name
```

パスおよびクエリーパラメーターを使用します。

71.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
cryptoOperation	**必須** エンドポイント URI の暗号スキームの後に指定された暗号操作を設定します。たとえば、crypto:sign 操作として署名を設定します。		CryptoOperation
name	**必須** この操作の論理名。		String

71.3.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
algorithm (producer)	署名者に使用するアルゴリズムの JCE 名を設定します。	SHA1WithDSA	String

名前	説明	デフォルト	タイプ
alias (producer)	エクスチェンジの署名と検証に使用される鍵とリンク java.security.cert.Certificate 証明書を KeyStore に照会するために使用される別名を設定します。この値は、実行時にメッセージヘッダーリンク org.apache.camel.component.crypto.DigitalSignature ConstantsKEYSTORE_ALIAS を介して提供できます。		文字列
certificateName (producer)	レジストリーで使用できる PrivateKey の参照名を設定します。		String
keystore (producer)	エクスチェンジの署名と検証に使用するキーと証明書を格納できる KeyStore を設定します。KeyStore は通常、Route 定義で提供されるか、メッセージヘッダーの CamelSignatureKeyStoreAlias を介して動的に指定されるエイリアスと共に使用されます。エイリアスが指定されておらず、キーストアに 1 つのエントリーしかない場合は、この 1 つのエントリーが使用されます。		KeyStore
keystoreName (producer)	レジストリーで使用できるキーストアの参照名を設定します。		String
privateKey (producer)	エクスチェンジの署名に使用する PrivateKey を設定します		PrivateKey
privateKeyName (producer)	レジストリーで使用できる PrivateKey の参照名を設定します。		String
provider (producer)	設定された署名アルゴリズムを提供するセキュリティープロバイダーの ID を設定します。		String
publicKeyName (producer)	コンテキストが変更されたときに解決する必要がある参照		String
secureRandomName (producer)	レジストリーで使用できる SecureRandom の参照名を設定します。		String
signatureHeaderName (producer)	base64 でエンコードされた署名を格納するために使用する必要があるメッセージヘッダーの名前を設定します。これはデフォルトで CamelDigitalSignature に設定されています		String
bufferSize (advanced)	Exchange ペイロードデータの読み込みに使用されるバッファーのサイズを設定します。	2048	Integer

名前	説明	デフォルト	タイプ
certificate (advanced)	ペイロードに基づいてエクスチェンジで署名を検証するために使用する証明書を設定します。		証明書
clearHeaders (advanced)	署名と検証後に署名固有のヘッダーをクリアするかどうかを決定します。デフォルトは true ですが、設定を解除するとキーやパスワードなどの重要な個人情報が漏洩する可能性があるため、極度の危険がある場合にのみ設定する必要があります。	true	boolean
keyStoreParameters (advanced)	指定された KeyStoreParameters に基づいて、エクスチェンジの署名と検証に使用するキーと証明書を格納できる KeyStore を設定します。KeyStore は通常、Route 定義で提供されるか、メッセージヘッダーの CamelSignatureKeyStoreAlias を介して動的に指定されるエイリアスと共に使用されます。エイリアスが指定されておらず、キーストアに 1 つのエントリーしかない場合は、この 1 つのエントリーが使用されます。		KeyStoreParameters
publicKey (advanced)	エクスチェンジで署名を検証するために使用する PublicKey を設定します。		PublicKey
secureRandom (advanced)	Signature サービスの初期化に使用する SecureRandom を設定します		SecureRandom
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
password (security)	KeyStore でエイリアス化された PrivateKey にアクセスするために使用するパスワードを設定します。		文字列

71.4. 使用

71.4.1. Raw keys

エクスチェンジに署名して検証する最も基本的な方法は、次のように KeyPair を使用することです。

キーへの参照を使用して、Spring XML 拡張機能 で同じことを実現できます。

71.4.2. キーストアとエイリアス。

JCE は、秘密鍵と証明書のペアを格納し、それらを暗号化し、パスワードで保護するための非常に用途の広いキーストアの概念を提供します。これらは、取得 API にエイリアスを適用することで取得できます。鍵と証明書を鍵ストアに入れる方法はいくつかありますが、ほとんどの場合、これは外部の keytool アプリケーションで行われます。これは、keytool を使用して、自己署名証明書と秘密鍵で KeyStore を作成する良い例です。

この例では、bob によってエイリアス化されたキーと証明書を持つキーストアを使用しています。キーストアとキーのパスワードは letmein です

以下は、Fluent ビルダーを介してキーストアを使用する方法を示しています。また、キーストアをロードして初期化する方法も示しています。

再び Spring では、ref を使用して実際のキーストアインスタンスを検索します。

71.4.3. JCE プロバイダーとアルゴリズムの変更

署名アルゴリズムまたはセキュリティープロバイダーの変更は、それらの名前を指定するだけの簡単な作業です。選択したアルゴリズムと互換性のあるキーも使用する必要があります。

または

71.4.4. 署名メッセージヘッダーの変更

署名を格納するために使用されるメッセージヘッダーを変更することが望ましい場合があります。次のように、ルート定義で別のヘッダー名を指定できます。

または

71.4.5. バッファーサイズの変更

バッファーのサイズを更新する必要がある場合...

または

71.4.6. キーを動的に提供します。

受信者リストまたは同様の EIP を使用する場合、エクスチェンジの受信者は動的に変化する可能性があります。すべての受信者に同じキーを使用することは、実現可能でも望ましくもない場合があります。署名キーを交換ごとに動的に指定できると便利です。エクスチェンジは、署名する前に、ターゲット受信者のキーで動的に強化できます。これを容易にするために、署名メカニズムでは、以下のメッセージヘッダーを介してキーを動的に提供できます。

- **Exchange.SIGNATURE_PRIVATE_KEY**, **"CamelSignaturePrivateKey"**
- **Exchange.SIGNATURE_PUBLIC_KEY_OR_CERT**, **"CamelSignaturePublicKeyOrCert"**

または

キーストアエイリアスを動的に指定することをお勧めします。この場合も、エイリアスはメッセージヘッダーで指定できます。

- **Exchange.KEYSTORE_ALIAS**, **"CamelSignatureKeyStoreAlias"**

または

ヘッダーは次のように設定されます

```
Exchange unsigned = getMandatoryEndpoint("direct:alias-sign").createExchange();
unsigned.getIn().setBody(payload);
unsigned.getIn().setHeader(DigitalSignatureConstants.KEYSTORE_ALIAS, "bob");
unsigned.getIn().setHeader(DigitalSignatureConstants.KEYSTORE_PASSWORD,
```

```
"letmein".toCharArray());
template.send("direct:alias-sign", unsigned);
Exchange signed = getMandatoryEndpoint("direct:alias-sign").createExchange();
signed.getIn().copyFrom(unsigned.getOut());
signed.getIn().setHeader(KEYSTORE_ALIAS, "bob");
template.send("direct:alias-verify", signed);
```

71.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第72章 暗号 CMS コンポーネント

Camel バージョン 2.20 以降で利用可能

Cryptographic Message Syntax (CMS) は、メッセージの署名と暗号化の確立された標準です。Apache Crypto CMS コンポーネントは、この標準の次の部分をサポートしています。CMS エンベロープデータインスタンスの作成、CMS エンベロープデータインスタンスの復号化、CMS 署名データインスタンスの作成、および CMS 署名データインスタンスの検証を行うことができます。

コンポーネントは、Bouncy Castle ライブラリー bcprov-jdk15on および bcpkix-jdk15on を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto-cms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントのエンドポイントを呼び出す前に、Bouncy Castle セキュリティープロバイダーをアプリケーションに登録することをお勧めします。

```
Security.addProvider(new BouncyCastleProvider());
```

Bouncy Castle セキュリティープロバイダーが登録されていない場合は、Crypto CMS コンポーネントがプロバイダーを登録します。

72.1. オプション

Crypto CMS コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
signedDataVerifier Configuration (advanced)	検証操作の uri パラメーターを決定する、共有の SignedDataVerifierConfiguration を設定します。		SignedDataVerifier Configuration
envelopedDataDecryptor Configuration (advanced)	暗号化解除操作の uri パラメーターを決定する共有 EnvelopedDataDecryptorConfiguration を設定する場合。		EnvelopedDataDecryptor Configuration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Crypto CMS エンドポイントは、URI 構文を使用して設定されます。

```
crypto-cms:cryptoOperation:name
```

パスおよびクエリーパラメーターを使用します。

72.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
cryptoOperation	**必須** エンドポイント URI の暗号スキームの後に指定された暗号操作を設定します。たとえば、crypto-cms:sign 操作として署名を設定します。可能な値: 署名、検証、暗号化、または復号化。		CryptoOperation
name	**必須** URI の名前部分をユーザーが選択して、camel コンテキスト内の異なる署名者/検証者/暗号化者/復号化者エンドポイントを区別できます。		String

72.1.2. クエリーパラメーター(15 個のパラメーター):

名前	説明	デフォルト	タイプ
keyStore (common)	操作に応じて、署名者の秘密鍵、検証者の公開鍵、暗号化者の公開鍵、復号化者の秘密鍵を含むキーストア。このパラメーターまたはパラメーター keyStoreParameters のいずれかを使用します。		KeyStore
keyStoreParameters (common)	操作に応じて、署名者の秘密鍵、検証者の公開鍵、暗号化者の公開鍵、復号化者の秘密鍵を含むキーストア。このパラメーターまたはパラメーター keystore のいずれかを使用します。		KeyStoreParameters
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
password (decrypt)	秘密鍵のパスワードを設定します。キーストア内のすべての秘密鍵のパスワードは同じであると想定されています。設定されていない場合、秘密鍵のパスワードは、KeyStoreParameters で指定されたキーストアパスワードによって指定されると想定されます。		Char[]
fromBase64 (decrypt_verify)	true の場合、CMS メッセージは Base 64 でエンコードされており、処理中にデコードする必要があります。デフォルト値は false です。	false	Boolean

名前	説明	デフォルト	タイプ
contentEncryptionAlgorithm (encrypt)	DESede/CBC/PKCS5Padding などの暗号化アルゴリズム。さらに可能な値: DESede/CBC/PKCS5Padding、AES/CBC/PKCS5Padding、Camellia/CBC/PKCS5Padding、CAST5/CBC/PKCS5Padding。		String
originatorInformation Provider (encrypt)	発信者情報のプロバイダー。https://tools.ietf.org/html/rfc5652section-6.1 を参照してください。デフォルト値は null です。		OriginatorInformation Provider
recipient (encrypt)	受信者情報: インターフェイス org.apache.camel.component.crypto.cms.api.TransRecipientInfo を実装する Bean への参照		List
secretKeyLength (encrypt)	コンテンツの暗号化に使用される秘密対称キーのキーの長さ。指定されたコンテンツ暗号化アルゴリズムが異なるサイズのキーを許可する場合にのみ使用されます。contentEncryptionAlgorithm=AES/CBC/PKCS5Padding または Camellia/CBC/PKCS5Padding の場合は 128。contentEncryptionAlgorithm=DESede/CBC/PKCS5Padding の場合、192、128。強力な暗号化が有効になっている場合、AES/CBC/PKCS5Padding と Camellia/CBC/PKCS5Padding では、キーの長さ 192 と 256 も可能です。		int
unprotectedAttributes GeneratorProvider (encrypt)	保護されていない属性のジェネレーターのプロバイダー。デフォルト値は null で、保護されていない属性がエンベロープデータオブジェクトに追加されないことを意味します。https://tools.ietf.org/html/rfc5652section-6.1 を参照してください。		AttributesGenerator プロバイダー
toBase64 (encrypt_sign)	Signed Data または Enveloped Data インスタンスが base 64 エンコードされるかどうかを示します。デフォルト値は false です。	false	Boolean
includeContent (sign)	署名されたコンテンツを Signed Data インスタンスに含める必要があるかどうかを示します。false の場合、分離された署名付きデータインスタンスがヘッダー CamelCryptoCmsSignedData に作成されます。	true	Boolean
signer (sign)	署名者情報: org.apache.camel.component.crypto.cms.api.SignerInfo を実装する Bean への参照		List

名前	説明	デフォルト	タイプ
signedDataHeaderBase64 (verify)	ヘッダー CamelCryptoCmsSignedData の値が base64 でエンコードされているかどうかを示します。デフォルト値は false です。デタッチされた署名にのみ関連します。デタッチされた署名の場合、ヘッダーには署名済みデータオブジェクトが含まれます。	false	Boolean
verifySignaturesOfAll Signers (verify)	true の場合、Signed Data オブジェクトに含まれるすべての署名者の署名が検証されます。false の場合、署名者情報が指定された証明書の 1 つと一致する 1 つの署名のみが検証されます。デフォルト値は true です。	true	Boolean

72.2. エンベロープデータ

通常、**crypto-cms:encypt** エンドポイントは 1 つのルートで定義され、補完的な **crypto-cms:decrypt** は別のルートで定義されることに注意してください。

次の例は、エンベロープデータメッセージを作成する方法と、エンベロープデータメッセージを復号化する方法を示しています。

Java DSL の基本的な例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private key if not specified in the crypto-cms:decrypt endpoint

DefaultKeyTransRecipientInfo recipient1 = new DefaultKeyTransRecipientInfo();
recipient1.setCertificateAlias("rsa"); // alias of the public key used for the encryption
recipient1.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); // register keystore in the registry
simpleReg.put("recipient1", recipient1); // register recipient info in the registry

from("direct:start")
    .to("crypto-cms:encrypt://testencrpyt?toBase64=true&recipient=#recipient1&contentEncryptionAlgorithm=DESede/CBC/PKCS5Padding&secretKeyLength=128")
    .to("crypto-cms:decrypt://testdecrypt?fromBase64=true&keyStoreParameters=#keyStoreParameters")
    .to("mock:result");
```

Spring XML の基本的な例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
  <bean id="recipient1"
      class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="certificateAlias" value="rsa" />
  </bean>
...
  <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:encrypt://testencrpyt?
toBase64=true&amp;recipient=#recipient1&amp;contentEncryptionAlgorithm=DESede/CBC/PKCS5Pad
ding&amp;secretKeyLength=128" />
      <to uri="crypto-cms:decrypt://testdecrypt?
fromBase64=true&amp;keyStoreParameters=#keyStoreParameters1" />
       <to uri="mock:result" />
  </route>
```

Java DSL の 2 つの受信者

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private
key if not specified in the crypto-cms:decrypt endpoint

DefaultKeyTransRecipientInfo recipient1 = new DefaultKeyTransRecipientInfo();
recipient1.setCertificateAlias("rsa"); // alias of the public key used for the encryption
recipient1.setKeyStoreParameters(keystore);

DefaultKeyTransRecipientInfo recipient2 = new DefaultKeyTransRecipientInfo();
recipient2.setCertificateAlias("dsa");
recipient2.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); // register keystore in the registry
simpleReg.put("recipient1", recipient1); // register recipient info in the registry

from("direct:start")
  .to("crypto-cms:encrypt://testencrpyt?
toBase64=true&recipient=#recipient1&recipient=#recipient2&contentEncryptionAlgorithm=DESede/CBC
PKCS5Padding&secretKeyLength=128")
  //the decryptor will automatically choose one of the two private keys depending which one is in the
decryptor keystore
  .to("crypto-cms:decrypt://testdecrypt?
fromBase64=true&keyStoreParameters=#keyStoreParameters")
  .to("mock:result");
```

Spring XML の 2 つの受信者

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
```

```
        password="some_password" type="JCEKS" />
    <bean id="recipient1"
        class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
        <property name="keyStoreParameters" ref="keyStoreParameters1" />
        <property name="certificateAlias" value="rsa" />
    </bean>
    <bean id="recipient2"
        class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
        <property name="keyStoreParameters" ref="keyStoreParameters1" />
        <property name="certificateAlias" value="dsa" />
    </bean>
...
    <route>
        <from uri="direct:start" />
        <to uri="crypto-cms:encrypt://testencrpyt?
toBase64=true&amp;recipient=#recipient1&amp;recipient=#recipient2&amp;contentEncryptionAlgorithm
=DESede/CBC/PKCS5Padding&amp;secretKeyLength=128" />
        <!-- the decryptor will automatically choose one of the two private keys depending which one is in
the decryptor keystore -->
        <to uri="crypto-cms:decrypt://testdecrypt?
fromBase64=true&amp;keyStoreParameters=#keyStoreParameters1" />
         <to uri="mock:result" />
    </route>
```

72.3. 署名付きデータ

通常、**crypto-cms:sign** エンドポイントは1つのルートで定義され、補完的な **crypto-cms:verify** は別のルートで定義されることに注意してください。

次の例は、Signed Data メッセージを作成する方法と、Signed Data メッセージを検証する方法を示しています。

Java DSL の基本的な例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private
key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType,
signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is included.
// If you want to add your own signed attributes or unsigned attributes, see methods
DefaultSignerInfo.setSignedAttributeGenerator and DefaultSignerInfo.setUnsignedAttributeGenerator.
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private key will
be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the signature
algorithm must fit to the signer private key.
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set then the
password of the KeyStoreParameters will be used for accessing the private key
```

```
signerInfo1.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry

from("direct:start")
    .to("crypto-cms:sign://testsign?signer=#signer1&includeContent=true&toBase64=true")
    .to("crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters&fromBase64=true"")
    .to("mock:result");
```

Spring XML の基本的な例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
   <bean id="signer1"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="rsa" />
      <property name="signatureAlgorithm" value="SHA256withRSA" />
      <property name="includeCertificates" value="true" />
      <!-- optional parameter 'password', if not set then the password of the KeyStoreParameters will
be used for accessing the private key -->
      <property name="password" value="private_key_pw" />
   </bean>
...
   <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:sign://testsign?
signer=#signer1&amp;includeContent=true&amp;toBase64=true" />
      <to uri="crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters1&amp;fromBase64=true" />
      <to uri="mock:result" />
   </route>
```

Java DSL での 2 人の署名者の例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private
key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType,
signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is included.
// If you want to add your own signed attributes or unsigned attributes, see methods
DefaultSignerInfo.setSignedAttributeGenerator and DefaultSignerInfo.setUnsignedAttributeGenerator.
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private key will
be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the signature
algorithm must fit to the signer private key.
```

```
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set then the
password of the KeyStoreParameters will be used for accessing the private key
signerInfo1.setKeyStoreParameters(keystore);

DefaultSignerInfo signerInfo2 = new DefaultSignerInfo();
signerInfo2.setIncludeCertificates(true);
signerInfo2.setSignatureAlgorithm("SHA256withDSA");
signerInfo2.setPrivateKeyAlias("dsa");
signerInfo2.setKeyStoreParameters(keystore);


simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry
simpleReg.put("signer2", signerInfo2); //register signer info in the registry

from("direct:start")
    .to("crypto-cms:sign://testsign?signer=#signer1&signer=#signer2&includeContent=true")
    .to("crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters")
    .to("mock:result");
```

Spring XML での 2 つの署名者の例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
    <bean id="signer1"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="rsa" />
      <property name="signatureAlgorithm" value="SHA256withRSA" />
      <property name="includeCertificates" value="true" />
      <!-- optional parameter 'password', if not set then the password of the KeyStoreParameters will
be used for accessing the private key -->
      <property name="password" value="private_key_pw" />
    </bean>
    <bean id="signer2"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="dsa" />
      <property name="signatureAlgorithm" value="SHA256withDSA" />
      <!-- optional parameter 'password', if not set then the password of the KeyStoreParameters will
be used for accessing the private key -->
      <property name="password" value="private_key_pw2" />
    </bean>
...
    <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:sign://testsign?
signer=#signer1&amp;signer=#signer2&amp;includeContent=true" />
      <to uri="crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters1" />
      <to uri="mock:result" />
    </route>
```

Java DSL でのデタッチされた署名の例

-

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType, signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is included.
// If you want to add your own signed attributes or unsigned attributes, see methods DefaultSignerInfo.setSignedAttributeGenerator and DefaultSignerInfo.setUnsignedAttributeGenerator.
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private key will be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the signature algorithm must fit to the signer private key.
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set then the password of the KeyStoreParameters will be used for accessing the private key
signerInfo1.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry

from("direct:start")
    //with the option includeContent=false the SignedData object without the signed text will be written into the header "CamelCryptoCmsSignedData"
   .to("crypto-cms:sign://testsign?signer=#signer1&includeContent=false&toBase64=true")
   //the verifier reads the Signed Data object form the header CamelCryptoCmsSignedData and assumes that the signed content is in the message body
   .to("crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters&signedDataHeaderBase64=true")
   .to("mock:result");
```

Spring XML のデタッチされた署名の例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
  <bean id="signer1"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="rsa" />
      <property name="signatureAlgorithm" value="SHA256withRSA" />
      <property name="includeCertificates" value="true" />
      <!-- optional parameter 'password', if not set then the password of the KeyStoreParameters will be used for accessing the private key -->
      <property name="password" value="private_key_pw" />
  </bean>
...
  <route>
      <from uri="direct:start" />
      <!-- with the option includeContent=false the SignedData object without the signed text will be written into the header "CamelCryptoCmsSignedData" -->
```

```
        <to uri="crypto-cms:sign://testsign?
signer=#signer1&amp;includeContent=false&amp;toBase64=true" />
        <!-- the verifier reads the Signed Data object form the header CamelCryptoCmsSignedData and
assumes that the signed content is in the message body -->
        <to uri="crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters1&amp;signedDataHeaderBase64=true" />
        <to uri="mock:result" />
    </route>
```

第73章 CRYPTO (JAVA 暗号化拡張) DATAFORMAT

Camel バージョン 2.3 以降で利用可能

Crypto Data Format は、Java Cryptographic Extension を Camel に統合し、Camel の使い慣れたマーシャルおよびアンマーシャルフォーマットメカニズムを使用して、メッセージのシンプルかつ柔軟な暗号化と復号化を可能にします。マーシャリングは暗号文への暗号化を意味し、アンマーシャリングは元の平文への復号化を意味すると想定しています。このデータ形式は、対称 (共有キー) 暗号化と復号化のみを実装します。

73.1. CRYPTODATAFORMAT オプション

Crypto (Java Cryptographic Extension) データ形式は、以下に示す 10 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
algorithm	**DES/CBC/PKCS5Padding**	**String**	使用される暗号アルゴリズムを示す JCE アルゴリズム名。デフォルトでは DES/CBC/PKCS5Padding です。
cryptoProvider		**String**	使用する JCE セキュリティープロバイダーの名前。
keyRef		**String**	使用するレジスタから参照する秘密鍵を参照します。
initVectorRef		**String**	Cipher の初期化に使用される初期化ベクトルを含むバイト配列を参照します。
algorithmParameterRef		**String**	Cipher の初期化に使用される JCE AlgorithmParameterSpec。指定された名前を java.security.spec.AlgorithmParameterSpec タイプとして使用してタイプを検索します。
buffersize		**Integer**	署名プロセスで使用されるバッファーのサイズ。
macAlgorithm	**HmacSHA1**	**String**	メッセージ認証アルゴリズムを示す JCE アルゴリズム名。
shouldAppendHMAC	**false**	**Boolean**	メッセージ認証コードを計算して暗号化されたデータに追加する必要があることを示すフラグ。
inline	**false**	**Boolean**	設定された IV を暗号化されたデータストリームにインライン化する必要があることを示すフラグ。デフォルトでは false です。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

73.2. 基本的な使用方法

交換を暗号化/復号化するために必要な最も基本的なものは、共有秘密鍵だけです。Crypto データ形式の 1 つ以上のインスタンスがこのキーで設定されている場合、その形式を使用して、あるルート (またはルートの一部) でペイロードを暗号化し、別のルートで復号化できます。たとえば、Java DSL を次のように使用します。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());

from("direct:basic-encryption")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

Spring では、データ形式が最初に設定され、次にルートで使用されます

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <crypto id="basic" algorithm="DES" keyRef="desKey" />
  </dataFormats>
    ...
  <route>
    <from uri="direct:basic-encryption" />
    <marshal ref="basic" />
    <to uri="mock:encrypted" />
    <unmarshal ref="basic" />
    <to uri="mock:unencrypted" />
  </route>
</camelContext>
```

73.3. 暗号化アルゴリズムの指定

アルゴリズムを変更するには、JCE アルゴリズム名を指定します。アルゴリズムを変更する場合は、互換性のあるキーを使用する必要があります。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);
cryptoFormat.setMacAlgorithm("HmacMD5");
```

```
from("direct:hmac-algorithm")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

Java 7 で使用可能なアルゴリズムのリストは、Java 暗号化アーキテクチャー標準アルゴリズム名のドキュメントから入手できます。

73.4. 初期化ベクトルの指定

一部の暗号化アルゴリズム、特にブロックアルゴリズムでは、初期化ベクトルと呼ばれるデータの初期ブロックを使用して設定する必要があります。JCE では、これは Cipher が初期化されるときに AlgorithmParameterSpec として渡されます。このようなベクトルを CryptoDataFormat で使用するには、必要なデータを含む byte[] で設定できます。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");
byte[] initializationVector = new byte[] {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding",
generator.generateKey());
cryptoFormat.setInitializationVector(initializationVector);

from("direct:init-vector")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または Spring を使用して、byte[] への参照を提供します

```
<crypto id="initvector" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey"
initVectorRef="initializationVector" />
```

暗号化フェーズと復号化フェーズの両方で同じベクトルが必要です。IV を秘密にしておく必要がないため、DataFormat を使用すると、IV を暗号化されたデータにインライン化し、その後復号化フェーズで読み取って Cipher を初期化できます。IV をインライン化するには、/oinline フラグを設定します。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");
byte[] initializationVector = new byte[] {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};
SecretKey key = generator.generateKey();

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding", key);
cryptoFormat.setInitializationVector(initializationVector);
cryptoFormat.setShouldInlineInitializationVector(true);
CryptoDataFormat decryptFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding", key);
decryptFormat.setShouldInlineInitializationVector(true);

from("direct:inline")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(decryptFormat)
    .to("mock:unencrypted");
```

または Spring 付き。

```
<crypto id="inline" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey"
initVectorRef="initializationVector"
  inline="true" />
<crypto id="inline-decrypt" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey" inline="true" />
```

初期化ベクトルの使用の詳細については、

- http://en.wikipedia.org/wiki/Initialization_vector
- http://www.herongyang.com/Cryptography/
- http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

73.5. ハッシュメッセージ認証コード (HMAC)

転送中の暗号化されたデータに対する攻撃を回避するために、CryptoDataFormat は、設定可能な MAC アルゴリズムに基づいて、暗号化されたエクスチェンジコンテンツのメッセージ認証コードを計算することもできます。計算された HMAC は、暗号化後にストリームに追加されます。復号化フェーズでストリームから分離されます。MAC は再計算され、送信されたバージョンに対して検証され、送信中に改ざんされていないことが保証されます。メッセージ認証コードの詳細については、http://en.wikipedia.org/wiki/HMAC を参照してください。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);

from("direct:hmac")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または Spring 付き。

```
<crypto id="hmac" algorithm="DES" keyRef="desKey" shouldAppendHMAC="true" />
```

デフォルトでは、HMAC は HmacSHA1 mac アルゴリズムを使用して計算されますが、これは別のアルゴリズム名を指定することで簡単に変更できます。設定されたセキュリティープロバイダーを通じて利用可能なアルゴリズムを確認する方法については、こちらを参照してください。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);
cryptoFormat.setMacAlgorithm("HmacMD5");

from("direct:hmac-algorithm")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または Spring 付き。

```
<crypto id="hmac-algorithm" algorithm="DES" keyRef="desKey" macAlgorithm="HmacMD5"
shouldAppendHMAC="true" />
```

73.6. キーを動的に提供する

受信者リストまたは同様の EIP を使用する場合、エクスチェンジの受信者は動的に変化する可能性があります。すべての受信者に同じキーを使用することは、実現可能または望ましくない場合があります。エクスチェンジごとにキーを動的に指定できると便利です。エクスチェンジは、データ形式によって処理される前に、ターゲット受信者のキーで動的に強化される可能性があります。これを容易にするために、DataFormat は以下のメッセージヘッダーを介してキーを動的に提供できるようにします。

- CryptoDataFormat.KEY "CamelCryptoKey"

```
CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", null);
/**
 * Note: the header containing the key should be cleared after
 * marshalling to stop it from leaking by accident and
 * potentially being compromised. The processor version below is
 * arguably better as the key is left in the header when you use
 * the DSL leaks the fact that camel encryption was used.
 */
from("direct:key-in-header-encrypt")
    .marshal(cryptoFormat)
    .removeHeader(CryptoDataFormat.KEY)
    .to("mock:encrypted");

from("direct:key-in-header-decrypt").unmarshal(cryptoFormat).process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().getHeaders().remove(CryptoDataFormat.KEY);
        exchange.getOut().copyFrom(exchange.getIn());
    }
}).to("mock:unencrypted");
```

または Spring 付き。

```
<crypto id="nokey" algorithm="DES" />
```

73.7. 依存関係

camel ルートで Crypto データ形式を使用するには、次の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

73.8. 関連項目

- データ形式
- 暗号 (デジタル署名)
- http://www.bouncycastle.org/java.html

第74章 CSV データ形式

Camel バージョン 1.3 以降で利用可能

CSV データ形式は、Apache Commons CSV を使用して、Excel によってエクスポート/インポートされたものなどの CSV ペイロード (コンマ区切り値) を処理します。

74.1. オプション

CSV データ形式は、次にリストされている 28 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
formatRef		**String**	使用する参照形式。他の形式オプションで更新されます。デフォルト値は CSVFormat.DEFAULT です。
formatName		**String**	使用するフォーマットの名前。デフォルト値は CSVFormat.DEFAULT です。
commentMarkerDisabled	**false**	**Boolean**	参照形式のコメントマーカーを無効にします。
commentMarker		**String**	参照形式のコメントマーカーを設定します。
delimiter		**String**	使用する区切り文字を設定します。デフォルト値は、, (コンマ) です。
escapeDisabled	**false**	**Boolean**	エスケープ文字の使用を無効にするために使用します
escape		**String**	使用するエスケープ文字を設定します
headerDisabled	**false**	**Boolean**	ヘッダーを無効にするために使用します
header		**List**	CSV ヘッダーを設定するには
allowMissingColumnNames	**false**	**Boolean**	欠落している列名を許可するかどうか。
ignoreEmptyLines	**false**	**Boolean**	空行を無視するかどうか。
ignoreSurroundingSpaces	**false**	**Boolean**	周囲のスペースを無視するかどうか
nullStringDisabled	**false**	**Boolean**	null 文字列を無効にするために使用

名前	デフォルト	Java タイプ	説明
nullString		**String**	空文字列を設定します
quoteDisabled	**false**	**Boolean**	引用符を無効にするために使用
quote		**String**	デフォルトで引用符を設定します
recordSeparatorDisabled		**String**	レコード区切りを無効にするために使用されます
recordSeparator		**String**	デフォルトでは改行文字 (CRLF) であるレコード区切り文字 (別名改行) を設定します
skipHeaderRecord	**false**	**Boolean**	出力でヘッダーレコードをスキップするかどうか
quoteMode		**String**	引用モードを設定します
ignoreHeaderCase	**false**	**Boolean**	ヘッダー名にアクセスするときに大文字と小文字を区別するかどうかを設定します
trim	**false**	**Boolean**	前後の空白を削除するかどうかを設定します
trailingDelimiter	**false**	**Boolean**	末尾の区切り文字を追加するかどうかを設定します
lazyLoad	**false**	**Boolean**	アンマーシャリングで、その場で行を読み取る反復子を生成するか、またはすべての行を一度に読み取る必要があるか。
useMaps	**false**	**Boolean**	アンマーシャリングで、リストではなく行の値のマップ (HashMap) を生成するかどうかヘッダー (定義または収集) が必要です。
useOrderedMaps	**false**	**Boolean**	アンマーシャリングで、リストの代わりに行の値の順序付きマップ (LinkedHashMap) を生成するかどうか。ヘッダー (定義または収集) が必要です。
recordConverterRef		**String**	使用するレジストリーから検索するカスタム CsvRecordConverter を参照します
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

74.2. CSV へのマップのマーシャリング

このコンポーネントを使用すると、Java マップ (またはマップで変換できるその他のメッセージタイプ) を CSV ペイロードにマーシャリングできます。

次のボディーを考慮します。

```
Map<String, Object> body = new LinkedHashMap<>();
body.put("foo", "abc");
body.put("bar", 123);
```

そしてこの Java ルート定義

```
from("direct:start")
    .marshal().csv()
    .to("mock:result");
```

またはこの XML ルート定義

```
<route>
   <from uri="direct:start" />
   <marshal>
      <csv />
   </marshal>
   <to uri="mock:result" />
</route>
```

そして、以下を生成します。

```
abc,123
```

74.3. CSV メッセージを JAVA リストにアンマーシャリングする

アンマーシャリングは、CSV メッセージを CSV ファイル行 (すべてのフィールド値を持つ別のリストを含む) を持つ Java リストに変換します。

例: 人の名前、IQ、現在の活動を含む CSV ファイルがあります。

```
Jack Dalton, 115, mad at Averell
Joe Dalton, 105, calming Joe
William Dalton, 105, keeping Joe from killing Averell
Averell Dalton, 80, playing with Rantanplan
Lucky Luke, 120, capturing the Daltons
```

CSV コンポーネントを使用して、このファイルを非整列化できるようになりました。

```
from("file:src/test/resources/?fileName=daltons.csv&noop=true")
    .unmarshal().csv()
    .to("mock:daltons");
```

結果のメッセージには、次のような **List<List<String>>** が含まれます。

```
List<List<String>> data = (List<List<String>>) exchange.getIn().getBody();
for (List<String> line : data) {
    LOG.debug(String.format("%s has an IQ of %s and is currently %s", line.get(0), line.get(1),
line.get(2)));
}
```

74.4. LIST<MAP> を CSV にマーシャリングする

Camel 2.1 以降で利用可能

複数行のデータを CSV 形式にマーシャリングする必要がある場合は、メッセージペイロードを **List<Map<String, Object>>** オブジェクトとして保存できるようになりました。このオブジェクトには、各行のマップがリストに含まれています。

74.5. CSV のファイルポーラー、アンマーシャリング

受信データを処理できる Bean が指定された場合...

MyCsvHandler.java

```
// Some comments here
public void doHandleCsvData(List<List<String>> csvData)
{
    // do magic here
}
```

i. ルートは次のようになります

```
<route>
        <!-- poll every 10 seconds -->
        <from uri="file:///some/path/to/pickup/csvfiles?delete=true&amp;consumer.delay=10000" />
        <unmarshal><csv /></unmarshal>
        <to uri="bean:myCsvHandler?method=doHandleCsvData" />
</route>
```

74.6. パイプを区切り文字としてマーシャリングする

次のボディーを考慮します。

```
Map<String, Object> body = new LinkedHashMap<>();
body.put("foo", "abc");
body.put("bar", 123);
```

そしてこの Java ルート定義

```
// Camel version < 2.15
CsvDataFormat oldCSV = new CsvDataFormat();
oldCSV.setDelimiter("|");
from("direct:start")
    .marshal(oldCSV)
    .to("mock:result")
```

```
// Camel version >= 2.15
from("direct:start")
    .marshal(new CsvDataFormat().setDelimiter(&#39;|&#39;))
    .to("mock:result")
```

またはこの XML ルート定義

```
<route>
  <from uri="direct:start" />
  <marshal>
    <csv delimiter="|" />
  </marshal>
  <to uri="mock:result" />
</route>
```

そして、以下を生成します。

```
abc|123
```

XML # DSL 内で autogenColumns、configRef、strategyRef 属性を使用します

Camel 2.9.2 / 2.10 以降で利用可能で、Camel 2.15 で削除される

CSV データ形式をカスタマイズして、独自の **CSVConfig** や **CSVStrategy** を利用できます。また、**autogenColumns** オプションのデフォルト値が true であることにも注意してください。次の例は、このカスタマイズを示しています。

```
<route>
  <from uri="direct:start" />
  <marshal>
    <!-- make use of a strategy other than the default one which is
'org.apache.commons.csv.CSVStrategy.DEFAULT_STRATEGY' -->
    <csv autogenColumns="false" delimiter="|" configRef="csvConfig" strategyRef="excelStrategy" />
  </marshal>
  <convertBodyTo type="java.lang.String" />
  <to uri="mock:result" />
</route>

<bean id="csvConfig" class="org.apache.commons.csv.writer.CSVConfig">
  <property name="fields">
    <list>
      <bean class="org.apache.commons.csv.writer.CSVField">
        <property name="name" value="orderId" />
      </bean>
      <bean class="org.apache.commons.csv.writer.CSVField">
        <property name="name" value="amount" />
      </bean>
    </list>
  </property>
</bean>

<bean id="excelStrategy"
class="org.springframework.beans.factory.config.FieldRetrievingFactoryBean">
```

```
  <property name="staticField" value="org.apache.commons.csv.CSVStrategy.EXCEL_STRATEGY"
/>
</bean>
```

74.7. アンマーシャリング中に SKIPFIRSTLINE オプションを使用する

Camel 2.10 以降で利用でき、Camel 2.15 で削除される。

CSV ヘッダーを含む最初の行をスキップするように CSV データ形式に指示できます。Spring/XML DSL の使用:

```
<route>
  <from uri="direct:start" />
  <unmarshal>
    <csv skipFirstLine="true" />
  </unmarshal>
  <to uri="bean:myCsvHandler?method=doHandleCsv" />
</route>
```

または Java DSL:

```
CsvDataFormat csv = new CsvDataFormat();
csv.setSkipFirstLine(true);

from("direct:start")
  .unmarshal(csv)
.to("bean:myCsvHandler?method=doHandleCsv");
```

74.8. パイプを区切り文字としてアンマーシャリングする

Spring/XML DSL の使用:

```
<route>
  <from uri="direct:start" />
  <unmarshal>
    <csv delimiter="|" />
  </unmarshal>
  <to uri="bean:myCsvHandler?method=doHandleCsv" />
</route>
```

または Java DSL:

```
CsvDataFormat csv = new CsvDataFormat();
CSVStrategy strategy = CSVStrategy.DEFAULT_STRATEGY;
strategy.setDelimiter('|');
csv.setStrategy(strategy);

from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

```
CsvDataFormat csv = new CsvDataFormat();
```

```
csv.setDelimiter("|");

from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

```
CsvDataFormat csv = new CsvDataFormat();
CSVConfig csvConfig = new CSVConfig();
csvConfig.setDelimiter(";");
csv.setConfig(csvConfig);

from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

CSVConfig の問題

以下のようになります。

```
CSVConfig csvConfig = new CSVConfig();
csvConfig.setDelimiter(';');
```

動作しません。区切り文字を文字列として設定する必要があります。

74.9. 依存関係

Camel ルートで CSV を使用するには、このデータ形式を実装する **camel-csv** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-csv</artifactId>
  <version>x.x.x</version>
</dependency>
```

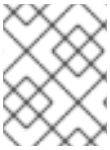
第75章 CXF

CXF コンポーネント

cxf: コンポーネントは、CXF でホストされている JAX-WS サービスに接続するための Apache CXF との統合を提供します。

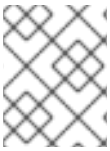
Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cxf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```



注記

CXF の依存関係について知りたい場合は、**WHICH-JARS** テキストファイルを参照してください。



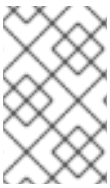
注記

ストリーミングモードで CXF を使用する場合 (DataFormat オプションを参照)、Stream caching についてもお読みください。

CAMEL ON EAP デプロイメント

このコンポーネントは、Red Hat JBoss Enterprise Application Platform (JBoss EAP) コンテナー上で簡素化されたデプロイメントモデルを提供する Camel on EAP (Wildfly Camel) フレームワークによってサポートされます。

CXF コンポーネントは、Apache CXF も使用する JBoss EAP **Web サービス** サブシステムと統合します。詳細については、JAX-WS を参照してください。



注記

現在、Camel on EAP サブシステムは CXF または Restlet コンシューマーをサポートしてい **ません**。ただし、**CamelProxy** を使用して、CXF コンシューマーの動作を模倣することは可能です。

URI 形式

```
cxf:bean:cxfEndpoint[?options]
```

cxfEndpoint は、Spring Bean レジストリー内の Bean を参照する Bean ID を表します。この URI 形式では、ほとんどのエンドポイントの詳細が Bean 定義で指定されます。

```
cxf://someAddress[?options]
```

someAddress は、CXF エンドポイントのアドレスを指定します。この URI 形式では、ほとんどのエンドポイントの詳細がオプションを使用して指定されます。

上記のどちらのスタイルでも、次のように URI にオプションを追加できます。

```
cxf:bean:cxfEndpoint?wsdlURL=wsdl/hello_world.wsdl&dataFormat=PAYLOAD
```

オプション

名前	必須	説明
wsdlURL	いいえ	WSDL のロケーション。デフォルトでは、WSDL はエンドポイントアドレスから取得されます。以下に例を示します。 **file://local/wsdl/hello.wsdl** or **wsdl/hello.wsdl**

serviceClass	○	SEI (Service Endpoint Interface) クラスの名前。このクラスには、JSR181 アノテーションを含めることができますが、必須ではありません。 **2.0 以降、** このオプションは POJO モードでのみ必要です。wsdlURL オプションが指定されている場合、PAYLOAD および MESSAGE モードでは serviceClass は必要ありません。wsdlURL オプションが serviceClass なしで使用される場合、serviceName および portName (Spring 設定のエンドポイント名) オプションを指定する **必要があり** ます。 2.0 以降、**\#** 表記を使用して、レジストリーから **serviceClass** オブジェクトインスタンスを参照できます。 非 Spring AOP プロキシーの **Object.getClass().getName()** メソッドに依存しているため、**参照されるオブジェクトをプロキシーにすることはできません (Spring AOP プロキシーは問題ありません)**。 **2.8 以降、** PAYLOAD および MESSAGE モードの wsdlURL および serviceClass オプションの両方を省略できます。それらを省略すると、任意の XML 要素を PAYLOAD モードで CxfPayload の本文に配置して、CXF ディスパッチモードを容易にすることができます。 例: **org.apache.camel.Hello**
serviceName	WSDL に複数の **serviceName が** 存在する場合のみ	このサービスが実装しているサービス名で、**wsdl:service@name** にマップされます。以下に例を示します。 **{http://org.apache.camel}ServiceName**

endpointName	**serviceName** の下に複数の **portName** が存在し、camel 2.2 以降の camel-cxf コンシューマーに必要な場合のみ	このサービスが実装しているポート名で、**wsdl:port@name** にマップされます。以下に例を示します。 **{http://org.apache.camel}PortName**
dataFormat	いいえ	CXF エンドポイントがサポートするメッセージデータ形式。可能な値は、**POJO (デフォルト)**、**PAYLOAD**、**MESSAGE** です。
relayHeaders	いいえ	このオプションについては、**relayHeadersオプションの説明** セクションを参照してください。CXF エンドポイントがルートに沿ってヘッダーをリレーする必要があります。現在、**dataFormat=POJO** の場合にのみ使用可能 **デフォルト**: **true例**: **true**、**false**
wrapped	いいえ	CXF エンドポイントプロデューサーが呼び出す操作の種類。設定可能な値は次のとおりです: **true**、**false (デフォルト)**。
wrappedStyle	いいえ	**2.5.0 以降** パラメーターが SOAP ボディーでどのように表現されるかを記述する WSDL スタイル。値が **false** の場合、CXF は document-literal のラップされていないスタイルを選択します。値が **true** の場合、CXF は document-literal のラップされたスタイルを選択します
setDefaultBus	いいえ	**非推奨:** このエンドポイントにデフォルトの CXF バスを使用するかどうかを指定します。設定可能な値は次のとおりです: **true**、**false (デフォルト)**。このオプションは非推奨であり、Camel 2.16 以降では **defaultBus** を使用する必要があります。

defaultBus	いいえ	**非推奨:** このエンドポイントにデフォルトの CXF バスを使用するかどうかを指定します。設定可能な値は次のとおりです: **true**、**false (デフォルト)**。このオプションは非推奨であり、Camel 2.16 以降では **defaultBus** を使用する必要があります。
bus	いいえ	レジストリーからバスオブジェクトを参照するには、**\#** 表記を使用します (例: **bus=\#busName**)。参照されるオブジェクトは **org.apache.cxf.Bus** のインスタンスでなければなりません。 デフォルトでは、CXF Bus Factory によって作成されたデフォルトバスを使用します。
cxfBinding	いいえ	**\#** 表記を使用して、レジストリーから CXF バインディングオブジェクトを参照します (例: **cxfBinding=\#bindingName)**。参照されるオブジェクトは **org.apache.camel.component.cxf.CxfBinding** のインスタンスでなければなりません。
headerFilterStrategy	いいえ	**\#** 表記を使用して、レジストリーから header filter strategy オブジェクトを参照します (例: **headerFilterStrategy=\#strategyName**)。参照されるオブジェクトは **org.apache.camel.spi.HeaderFilterStrategy** のインスタンスでなければなりません。
loggingFeatureEnabled	いいえ	2.3 の新機能であるこのオプションは、インバウンドおよびアウトバウンドの SOAP メッセージをログに書き込む CXF ログ機能を有効にします。設定可能な値は次のとおりです: **true**、**false (デフォルト)**。

defaultOperationName	いいえ	2.4 の新機能であるこのオプションは、リモートサービスを呼び出す **CxfProducer** によって使用されるデフォルトの **operationName** を設定します。以下に例を示します。 **defaultOperationName=greetMe**
defaultOperationNamespace	いいえ	2.4 の新機能であるこのオプションは、リモートサービスを呼び出す CxfProducer によって使用されるデフォルトの operationNamespace を設定します。以下に例を示します。 **defaultOperationNamespace=http://apache.org/hello_world_soap_http**
同期	いいえ	2.5 の新機能であるこのオプションにより、CXF エンドポイントは同期または非同期 API を使用して基礎となる作業を行うことを決定できます。デフォルト値は **false** です。これは、**camel-cxf** エンドポイントがデフォルトで非同期 API を使用しようとすることを意味します。
publishedEndpointUrl	いいえ	2.5 の新機能であるこのオプションは、サービスアドレス URL と **?wsdl** を使用してアクセスされる公開された WSDL に表示されるエンドポイント URL をオーバーライドします。以下に例を示します。 **publishedEndpointUrl=http://example.com/service**

properties.propName	いいえ	Camel 2.8: エンドポイント URI でカスタム CXF プロパティーを設定できます。たとえば、MTOM を有効にするには、**properties.mtom-enabled=true** を設定します。呼び出しの開始時に CXF がスレッドを切り替えないようにするには、**properties.org.apache.cxf.interceptor.OneWayProcessorInterceptor.USE_ORIGINAL_THREAD=true** を設定します。
allowStreaming	いいえ	2.8.2 の新機能。このオプションは、CXF コンポーネントが PAYLOAD モードで実行されている場合に (以下を参照してください)、着信メッセージを DOM 解析して DOM 要素にするか、場合によってはストリーミングを許可する javax.xml.transform.Source オブジェクトとしてペイロードを保持するかを制御します。
skipFaultLogging	いいえ	2.11 の新機能。このオプションは、PhaseInterceptorChain がキャッチした Fault のログ記録をスキップするかどうかを制御します。
cxfEndpointConfigurer	いいえ	Camel 2.11 の新機能。このオプションは、プログラムによる方法での CXF エンドポイントの設定をサポートする **org.apache.camel.component.cxf.CxfEndpointConfigurer which**の実装を適用できます。Camel 2.15.0 以降、
CxfEndpointConfigurer の configure{ServerClient} メソッドを実装することで、ユーザーは CXF サーバーとクライアントを設定できます。	**username**	いいえ
Camel 2.12.3 の新機能。このオプションは、CXF クライアントのユーザー名の基本認証情報を設定するために使用されます。	**password**	いいえ

Camel 2.12.3 の新機能。このオプションは、CXF クライアントのパスワードの基本認証情報を設定するために使用されます。	**continuationTimeout**	なし

serviceName と **portName** は QNames であるため、これらを指定する場合は、上記の例に示すように **{namespace}** を前に付けてください。

データ形式の説明

DataFormat	説明
POJO	POJO (plain old Java objects) は、ターゲットサーバーで呼び出されるメソッドへの Java パラメーターです。プロトコル JAX-WS ハンドラーと論理 JAX-WS ハンドラーの両方がサポートされています。
PAYLOAD	**PAYLOAD** は、CXF エンドポイントのメッセージ設定が適用された後のメッセージペイロード (**soap:body** の内容) です。Protocol JAX-WS ハンドラーのみがサポートされています。論理 JAX-WS ハンドラーはサポートされていません。
MESSAGE	**MESSAGE** は、トランスポート層から受信した raw メッセージです。Stream に触れたり変更したりすることは想定されていません。この種の DataFormat を使用している場合、CXF インターセプターの一部が削除されるため、camel-cxf コンシューマーと JAX-WS ハンドラーがサポートされていない後の SOAP ヘッダーは表示されません。
CXF_MESSAGE	Camel 2.8.2 の新機能。**CXF_MESSAGE** を使用すると、トランスポート層からのメッセージを生の SOAP メッセージに変換することで、CXF インターセプターの全機能を呼び出すことができます。

交換プロパティー **CamelCXFDataFormat** を取得することで、交換のデータ形式モードを判別できます。エクスチェンジキー定数は **org.apache.camel.component.cxf.CxfConstants.DATA_FORMAT_PROPERTY** で定義されています。

APACHE ARIES ブループリントを使用した CXF エンドポイントの設定。

Camel 2.8 以降、CXF エンドポイントに Aries ブループリント依存性注入を使用するためのサポートがあります。スキーマは Spring スキーマに非常に似ているため、移行はかなり透過的です。

以下に例を示します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
        xmlns:camel-cxf="http://camel.apache.org/schema/blueprint/cxf"
    xmlns:cxfcore="http://cxf.apache.org/blueprint/core"
        xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

      <camel-cxf:cxfEndpoint id="routerEndpoint"
                address="http://localhost:9001/router"
                serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
        <camel-cxf:properties>
          <entry key="dataFormat" value="MESSAGE"/>
        </camel-cxf:properties>
      </camel-cxf:cxfEndpoint>

      <camel-cxf:cxfEndpoint id="serviceEndpoint"
   address="http://localhost:9000/SoapContext/SoapPort"
                serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
    </camel-cxf:cxfEndpoint>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <route>
        <from uri="routerEndpoint"/>
        <to uri="log:request"/>
      </route>
    </camelContext>

</blueprint>
```

現在、エンドポイント要素は、サポートされている最初の CXF 名前空間ハンドラーです。

Spring と同じように Bean 参照を使用することもできます

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
        xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
        xmlns:cxf="http://cxf.apache.org/blueprint/core"
        xmlns:camel="http://camel.apache.org/schema/blueprint"
        xmlns:camelcxf="http://camel.apache.org/schema/blueprint/cxf"
        xsi:schemaLocation="
          http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
          http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd
          http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd
          ">

    <camelcxf:cxfEndpoint id="reportIncident"
                address="/camel-example-cxf-blueprint/webservices/incident"
                wsdlURL="META-INF/wsdl/report_incident.wsdl"
                serviceClass="org.apache.camel.example.reportincident.ReportIncidentEndpoint">
    </camelcxf:cxfEndpoint>

    <bean id="reportIncidentRoutes"
class="org.apache.camel.example.reportincident.ReportIncidentRoutes" />

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
```

```
        <routeBuilder ref="reportIncidentRoutes"/>
    </camelContext>

</blueprint>
```

MESSAGE モードで CXF の LOGGINGOUTINTERCEPTOR を有効にする方法

CXF の **LoggingOutInterceptor** は、ロギングシステム (**java.util.logging**) に送信されるアウトバウンドメッセージを出力します。**LoggingOutInterceptor** は **PRE_STREAM** フェーズにあるため (ただし、**PRE_STREAM** フェーズは **MESSAGE** モードでは削除されます)、**WRITE** フェーズ中に実行されるように **LoggingOutInterceptor** を設定する必要があります。以下に例を示します。

```
    <bean id="loggingOutInterceptor" class="org.apache.cxf.interceptor.LoggingOutInterceptor">
        <!--  it really should have been user-prestream but CXF does have such phase! -->
        <constructor-arg value="target/write"/>
    </bean>

<cxf:cxfEndpoint id="serviceEndpoint" address="http://localhost:9002/helloworld"
 serviceClass="org.apache.camel.component.cxf.HelloService">
 <cxf:outInterceptors>
     <ref bean="loggingOutInterceptor"/>
 </cxf:outInterceptors>
 <cxf:properties>
  <entry key="dataFormat" value="MESSAGE"/>
 </cxf:properties>
</cxf:cxfEndpoint>
```

RELAYHEADERS オプションの説明

JAXWS WSDL ファーストの開発者の観点から見ると、**帯域内** ヘッダーと **帯域外** ヘッダーがあります。

インバンド ヘッダーは、SOAP ヘッダーなどのエンドポイントの WSDL バインディングコントラクトの一部として明示的に定義されるヘッダーです。

アウトオブバンド ヘッダーは、ネットワーク経由でシリアル化されるヘッダーですが、明示的に WSDL バインディングコントラクトの一部ではありません。

ヘッダーの中継/フィルタリングは双方向です。

ルートに CXF エンドポイントがあり、開発者が SOAP ヘッダーなどのオンザワイヤヘッダーをルートに沿ってリレーして、たとえば別の JAXWS エンドポイントで消費する必要がある場合、**relayHeaders** を **true** に設定する必要があります。デフォルト値。

POJO モードでのみ使用可能

relayHeaders=true 設定は、ヘッダーをリレーする意図を表します。特定のヘッダーが中継されるかどうかの実際の決定は、**MessageHeadersRelay** インターフェイスを実装するプラグ可能なインスタンスに委譲されます。**MessageHeadersRelay** の具体的な実装を調べて、ヘッダーを中継する必要があるかどうかを判断します。既知の SOAP 名前空間にバインドする **SoapMessageHeadersRelay** の実装がすでにあります。現在、帯域外ヘッダーのみがフィルタリングされ、**relayHeaders=true** の場合、帯域内

ヘッダーは常に中継されます。ネームスペースがランタイムに不明なヘッダーがワイヤ上にある場合、フォールバック **DefaultMessageHeadersRelay** が使用されます。これにより、すべてのヘッダーの中継が単純に許可されます。

RelayHeaders=false 設定は、帯域内および帯域外のすべてのヘッダーがドロップされることを表明します。

独自の **MessageHeadersRelay** 実装をプラグインして、リレーのリストをオーバーライドまたは追加することができます。プリロードされたリレーインスタンスをオーバーライドするには、**MessageHeadersRelay** 実装がオーバーライドしようとしている名前空間と同じ名前空間を提供していることを確認してください。また、オーバーライドするリレーは、オーバーライドしようとしているすべての名前空間にサービスを提供する必要があることに注意してください。そうしないと、インスタンスマッピングをリレーする名前空間にあいまいさが生じるため、ルートの起動時に実行時例外が出力されます。

```
<cxf:cxfEndpoint ...>
  <cxf:properties>
    <entry key="org.apache.camel.cxf.message.headers.relays">
      <list>
        <ref bean="customHeadersRelay"/>
      </list>
    </entry>
  </cxf:properties>
 </cxf:cxfEndpoint>
 <bean id="customHeadersRelay"
class="org.apache.camel.component.cxf.soap.headers.CustomHeadersRelay"/>
```

ここでヘッダーをリレー/ドロップする方法を示すテストを見てください。

link:https://svn.apache.org/repos/asf/camel/branches/camel-1.x/components/camel-cxf/src/test/java/org/apache/camel/component/cxf/soap/headers/CxfMessageHeadersRelayTest.java[https://svn.apache.org/repos/asf/camel/branches/camel-1.x/components/camel-cxf/src/test/java/org/apache/camel/component/cxf/soap/headers/CxfMessageHeadersRelayTest.java]

リリース 2.0 以降の変更点

- **POJO** および **PAYLOAD** モードがサポートされています。**POJO** モードでは、CXF によってインバンドヘッダーが処理され、ヘッダーリストから削除されているため、フィルタリングに使用できるのはアウトオブバンドメッセージヘッダーのみです。インバンドヘッダーは、**POJO** モードで **MessageContentList** に組み込まれます。**camel-cxf** コンポーネントは、**MessageContentList** からインバンドヘッダーを削除しようとします。インバンドヘッダーのフィルタリングが必要な場合は、**PAYLOAD** モードを使用するか、(非常に簡単な) CXF インターセプター/JAXWS ハンドラーを CXF エンドポイントにプラグインしてください。

- メッセージヘッダーリレーメカニズムは **CxfHeaderFilterStrategy** にマージされました。**relayHeaders** オプション、そのセマンティクス、およびデフォルト値は同じままですが、**CxfHeaderFilterStrategy** のプロパティーです。以下はその設定例です。

  ```
  <bean id="dropAllMessageHeadersStrategy"
  class="org.apache.camel.component.cxf.common.header.CxfHeaderFilterStrategy">

    <!-- Set relayHeaders to false to drop all SOAP headers -->
  ```

```
    <property name="relayHeaders" value="false"/>

</bean>
```

その後、エンドポイントは **CxfHeaderFilterStrategy** を参照できます。

```
<route>
   <from uri="cxf:bean:routerNoRelayEndpoint?
headerFilterStrategy=#dropAllMessageHeadersStrategy"/>
   <to uri="cxf:bean:serviceNoRelayEndpoint?
headerFilterStrategy=#dropAllMessageHeadersStrategy"/>
</route>
```

- **MessageHeadersRelay** インターフェイスがわずかに変更され、名前が **MessageHeaderFilter** に変更されました。**CxfHeaderFilterStrategy** のプロパティーです。ユーザー定義のメッセージヘッダーフィルターを設定する例を次に示します。

```
<bean id="customMessageFilterStrategy"
class="org.apache.camel.component.cxf.common.header.CxfHeaderFilterStrategy">
   <property name="messageHeaderFilters">
      <list>
         <!--  SoapMessageHeaderFilter is the built in filter.  It can be removed by omitting it. -
->
         <bean
class="org.apache.camel.component.cxf.common.header.SoapMessageHeaderFilter"/>

         <!--  Add custom filter here -->
         <bean class="org.apache.camel.component.cxf.soap.headers.CustomHeaderFilter"/>
      </list>
   </property>
</bean>
```

- **RelayHeaders** 以外に、**CxfHeaderFilterStrategy** で設定できる新しいプロパティーがあります。

名前	説明	type	必須？	デフォルト値
relayHeaders	すべてのメッセージヘッダーは、メッセージヘッダーフィルターによって処理されます。	**boolean**	いいえ	**true** (1.6.1 動作)
relayAllMessageHeaders	すべてのメッセージヘッダーが伝達されます (メッセージヘッダーフィルターによる処理なし)。	**boolean**	いいえ	**false** (1.6.1 の動作)

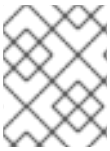
allowFilterNamespaceClash	アクティベーション名前空間で 2 つのフィルターが重複する場合、プロパティーはその処理方法を制御します。値が **true** の場合、最後のものが優先されます。値が **false** の場合、例外が出力されます	**boolean**	いいえ	**false** (1.6.1 の動作)

SPRING で CXF エンドポイントを設定する

以下に示す Spring 設定ファイルを使用して CXF エンドポイントを設定できます。また、エンドポイントを **camelContext** タグに埋め込むこともできます。サービスエンドポイントを呼び出すときに、**operationName** ヘッダーと **operationNamespace** ヘッダーを設定して、呼び出す操作を明示的に指定できます。

注記 Camel 2.x では、CXF エンドポイントのターゲット名前空間として **http://camel.apache.org/schema/cxf** を使用するように変更します。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://camel.apache.org/schema/cxf"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
       http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
">
 ...
```



注記

Apache Camel 2.x では、**http://activemq.apache.org/camel/schema/cxfEndpoint** 名前空間が **http://camel.apache.org/schema/cxf** に変更されました。

ルート Bean 要素で指定された JAX-WS **schemaLocation** 属性を必ず含めてください。これにより、CXF はファイルを検証できるようになります。これは必須です。**<cxf:cxfEndpoint/>** タグの末尾にある名前空間の宣言にも注意してください。これは、結合された **{namespace} localName** 構文が現在、このタグの属性値に対してサポートされていないため必要です。

cxf:cxfEndpoint 要素は、多くの追加属性をサポートしています。

名前	値

PortName	このサービスが実装しているエンドポイント名で、**wsdl:port@name** にマップされます。**ns:PORT_NAME** の形式で、**ns** はこのスコープで有効な名前空間接頭辞です。
serviceName	このサービスが実装しているサービス名で、**wsdl:service@name** にマップされます。**ns:SERVICE_NAME** の形式で、**ns** はこのスコープで有効な名前空間接頭辞です。
wsdlURL	WSDL のロケーション。クラスパス、ファイルシステム、またはリモートでホストできます。
bindingId	使用するサービスモデルの **bindingId**。
address	サービス公開アドレス。
bus	JAX-WS エンドポイントで使用されるバス名。
serviceClass	JSR181 アノテーションを持つかどうかにかかわらず、SEI (Service Endpoint Interface) クラスのクラス名。

また、多くの子要素もサポートしています。

名前	値
cxf:inInterceptors	このエンドポイントの着信インターセプター。**<bean>** または **<ref>** のリスト。
cxf:inFaultInterceptors	このエンドポイントの着信障害インターセプター。**<bean>** または **<ref>** のリスト。
cxf:outInterceptors	このエンドポイントの発信インターセプター。**<bean>** または **<ref>** のリスト。
cxf:outFaultInterceptors	このエンドポイントの送信障害インターセプター。**<bean>** または **<ref>** のリスト。
cxf:properties	JAX-WS エンドポイントに提供する必要があるプロパティーマップ。以下を参照してください。
cxf:handlers	JAX-WS エンドポイントに提供する必要がある JAX-WS ハンドラーリスト。以下を参照してください。
cxf:dataBinding	エンドポイントで使用する **DataBinding** を指定できます。これは、Spring **<bean class="MyDataBinding"/>** 構文を使用して提供できます。

cxf:binding	このエンドポイントが使用する **BindingFactory** を指定できます。これは、Spring **<bean class="MyBindingFactory"/>** 構文を使用して提供できます。
cxf:features	このエンドポイントのインターセプターを保持する機能。**<bean>** または **<ref>** のリスト
cxf:schemaLocations	エンドポイントが使用するスキーマのロケーション。**<schemaLocation>** のリスト
cxf:serviceFactory	このエンドポイントが使用するサービスファクトリー。これは、Spring **<bean class="MyServiceFactory"/>** 構文を使用して提供できます

インターセプター、プロパティー、およびハンドラーを提供する方法を示すより高度な例をここで見つけることができます: http://cwiki.apache.org/CXF20DOC/jax-ws-configuration.html

注記

次のように、CXF:properties を使用して、Spring 設定ファイルから CXF エンドポイントの **dataFormat** および **setDefaultBus** プロパティーを設定できます。

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/router"
    serviceClass="org.apache.camel.component.cxf.HelloService"
    endpointName="s:PortName"
    serviceName="s:ServiceName"
    xmlns:s="http://www.example.com/test">
    <cxf:properties>
      <entry key="dataFormat" value="MESSAGE"/>
      <entry key="setDefaultBus" value="true"/>
    </cxf:properties>
  </cxf:cxfEndpoint>
```

CAMEL-CXF コンポーネントで JAVA.UTIL.LOGGING の代わりに LOG4J を使用する方法

CXF のデフォルトのロガーは **java.util.logging** です。**log4j** に変更する場合は、次の手順を実行します。クラスパスに **META-INF/cxf/org.apache.cxf.logger** という名前のファイルを作成します。このファイルには、クラスの完全修飾名 **org.apache.cxf.common.logging.Log4jLogger** がコメントなしで 1 行に含まれている必要があります。

XML で CAMEL-CXF レスポンスメッセージを開始ドキュメントにする方法

PHP などの SOAP クライアントを使用している場合、CXF は XML 開始ドキュメント **<?xml version="1.0" encoding="utf-8"?>** を追加しないため、この種のエラーが発生します。

```
Error:sendSms: SoapFault exception: [Client] looks like we got no XML document in [...]
```

この問題を解決するには、StaxOutInterceptor に XML 開始ドキュメントを作成するよう指示するだけです。

```
public class WriteXmlDeclarationInterceptor extends AbstractPhaseInterceptor<SoapMessage> {
    public WriteXmlDeclarationInterceptor() {
        super(Phase.PRE_STREAM);
        addBefore(StaxOutInterceptor.class.getName());
    }

    public void handleMessage(SoapMessage message) throws Fault {
        message.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
    }

}
```

このようなカスタマーインターセプターを追加して、**camel-cxf** エンドポイントに設定できます。

```
<cxf:cxfEndpoint id="routerEndpoint"
address="http://localhost:${CXFTestSupport.port2}/CXFGreeterRouterTest/CamelContext/RouterPort"

  serviceClass="org.apache.hello_world_soap_http.GreeterImpl"
  skipFaultLogging="true">
    <cxf:outInterceptors>
        <!-- This interceptor will force the CXF server send the XML start document to client -->
        <bean class="org.apache.camel.component.cxf.WriteXmlDeclarationInterceptor"/>
    </cxf:outInterceptors>
    <cxf:properties>
        <!-- Set the publishedEndpointUrl which could override the service address from generated
WSDL as you want -->
        <entry key="publishedEndpointUrl" value="http://www.simple.com/services/test" />
    </cxf:properties>
</cxf:cxfEndpoint>
```

または、Camel 2.4 を使用している場合は、このようにメッセージヘッダーを追加します。

```
// set up the response context which force start document
Map<String, Object> map = new HashMap<String, Object>();
map.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
exchange.getOut().setHeader(Client.RESPONSE_CONTEXT, map);
```

POJO データ形式の CAMEL-CXF エンドポイントからのメッセージを使用する方法

camel-cxf エンドポイントコンシューマー **POJO** データ形式は cxf インボーカー に基づいているため、メッセージヘッダーには **CxfConstants.OPERATION_NAME** という名前のプロパティーがあり、メッセージ本文は SEI メソッドパラメーターのリストです。

```
public class PersonProcessor implements Processor {

    private static final transient Logger LOG = LoggerFactory.getLogger(PersonProcessor.class);

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        LOG.info("processing exchange in camel");
```

```
        BindingOperationInfo boi =
(BindingOperationInfo)exchange.getProperty(BindingOperationInfo.class.toString());
        if (boi != null) {
            LOG.info("boi.isUnwrapped" + boi.isUnwrapped());
        }
        // Get the parameters list which element is the holder.
        MessageContentsList msgList = (MessageContentsList)exchange.getIn().getBody();
        Holder<String> personId = (Holder<String>)msgList.get(0);
        Holder<String> ssn = (Holder<String>)msgList.get(1);
        Holder<String> name = (Holder<String>)msgList.get(2);

        if (personId.value == null || personId.value.length() == 0) {
            LOG.info("person id 123, so throwing exception");
            // Try to throw out the soap fault message
            org.apache.camel.wsdl_first.types.UnknownPersonFault personFault =
                new org.apache.camel.wsdl_first.types.UnknownPersonFault();
            personFault.setPersonId("");
            org.apache.camel.wsdl_first.UnknownPersonFault fault =
                new org.apache.camel.wsdl_first.UnknownPersonFault("Get the null value of person name",
personFault);
            // Since camel has its own exception handler framework, we can't throw the exception to
trigger it
            // We just set the fault message in the exchange for camel-cxf component handling and return
            exchange.getOut().setFault(true);
            exchange.getOut().setBody(fault);
            return;
        }

        name.value = "Bonjour";
        ssn.value = "123";
        LOG.info("setting Bonjour as the response");
        // Set the response message, first element is the return value of the operation,
        // the others are the holders of method parameters
        exchange.getOut().setBody(new Object[] {null, personId, ssn, name});
    }

}
```

POJO データ形式で CAMEL-CXF エンドポイントのメッセージを準備する方法

camel-cxf エンドポイントプロデューサーは、cxf client API に基づいています。まず、メッセージヘッダーでオペレーション名を指定し、次にメソッドパラメーターをリストに追加し、このパラメーターリストでメッセージを初期化する必要があります。レスポンスメッセージボディーは **messageContentsList** であり、そのリストから結果を取得できます。

メッセージヘッダーで操作名を指定しない場合、**CxfProducer** は **CxfEndpoint** の **defaultOperationName** を使用しようとします。**CxfEndpoint** に **defaultOperationName** が設定されていない場合、操作リストから最初の操作名が取得されます。

メッセージボディーからオブジェクト配列を取得する場合は、次のように **message.getbody (Object.class)** を使用してボディーを取得できます。

```
Exchange senderExchange = new DefaultExchange(context, ExchangePattern.InOut);
```

```
final List<String> params = new ArrayList<String>();
// Prepare the request message for the camel-cxf procedure
params.add(TEST_MESSAGE);
senderExchange.getIn().setBody(params);
senderExchange.getIn().setHeader(CxfConstants.OPERATION_NAME, ECHO_OPERATION);

Exchange exchange = template.send("direct:EndpointA", senderExchange);

org.apache.camel.Message out = exchange.getOut();
// The response message's body is an MessageContentsList which first element is the return value of
the operation,
// If there are some holder parameters, the holder parameter will be filled in the reset of List.
// The result will be extract from the MessageContentsList with the String class type
MessageContentsList result = (MessageContentsList)out.getBody();
LOG.info("Received output text: " + result.get(0));
Map<String, Object> responseContext = CastUtils.cast((Map<?, ?
>)out.getHeader(Client.RESPONSE_CONTEXT));
assertNotNull(responseContext);
assertEquals("We should get the response context here", "UTF-8",
responseContext.get(org.apache.cxf.message.Message.ENCODING));
assertEquals("Reply body on Camel is wrong", "echo " + TEST_MESSAGE, result.get(0));
```

PAYLOAD データ形式の CAMEL-CXF エンドポイントのメッセージを処理する方法

Apache Camel 2.0 の場合: **CxfMessage.getBody()** は、**org.apache.camel.component.cxf.CxfPayload** オブジェクトを返します。このオブジェクトには、SOAP メッセージヘッダーと Body 要素の getter があります。この変更により、ネイティブ CXF メッセージを Apache Camel メッセージから切り離すことができます。

```
protected RouteBuilder createRouteBuilder() {
    return new RouteBuilder() {
        public void configure() {
            from(SIMPLE_ENDPOINT_URI + "&dataFormat=PAYLOAD").to("log:info").process(new
Processor() {
                @SuppressWarnings("unchecked")
                public void process(final Exchange exchange) throws Exception {
                    CxfPayload<SoapHeader> requestPayload =
exchange.getIn().getBody(CxfPayload.class);
                    List<Source> inElements = requestPayload.getBodySources();
                    List<Source> outElements = new ArrayList<Source>();
                    // You can use a customer toStringConverter to turn a CxfPayLoad message into String
as you want
                    String request = exchange.getIn().getBody(String.class);
                    XmlConverter converter = new XmlConverter();
                    String documentString = ECHO_RESPONSE;

                    Element in = new XmlConverter().toDOMElement(inElements.get(0));
                    // Just check the element namespace
                    if (!in.getNamespaceURI().equals(ELEMENT_NAMESPACE)) {
                        throw new IllegalArgumentException("Wrong element namespace");
                    }
                    if (in.getLocalName().equals("echoBoolean")) {
                        documentString = ECHO_BOOLEAN_RESPONSE;
                        checkRequest("ECHO_BOOLEAN_REQUEST", request);
```

```
                } else {
                    documentString = ECHO_RESPONSE;
                    checkRequest("ECHO_REQUEST", request);
                }
                Document outDocument = converter.toDOMDocument(documentString);
                outElements.add(new DOMSource(outDocument.getDocumentElement()));
                // set the payload header with null
                CxfPayload<SoapHeader> responsePayload = new CxfPayload<SoapHeader>(null,
outElements, null);
                exchange.getOut().setBody(responsePayload);
            }
        });
    }
};
}
```

POJO モードで SOAP ヘッダーを取得および設定する方法

POJO は、camel-cxf エンドポイントが Camel エクスチェンジを生成または消費するときのデータ形式が **Java オブジェクトのリスト** であることを意味します。Apache Camel はこのモードでメッセージボディーを POJO として公開しますが、CXF コンポーネントは引き続き SOAP ヘッダーの読み取りと書き込みへのアクセスを提供します。ただし、CXF インターセプターはインバンド SOAP ヘッダーを処理後にヘッダーリストから削除するため、POJO モードではアウトオブバンド SOAP ヘッダーのみを使用できます。

次の例は、SOAP ヘッダーを取得/設定する方法を示しています。ある CXF エンドポイントから別のエンドポイントに転送するルートがあるとします。つまり、SOAP クライアント → Apache Camel → CXF サービスです。2 つのプロセッサーを接続して、(1) 要求が CXF サービスに送信される前と (2) 応答が SOAP クライアントに返される前に、SOAP ヘッダーを取得/挿入できます。この例のプロセッサー (1) と (2) は、InsertRequestOutHeaderProcessor と InsertResponseOutHeaderProcessor です。ルートは次のようになります。

```
<route>
    <from uri="cxf:bean:routerRelayEndpointWithInsertion"/>
    <process ref="InsertRequestOutHeaderProcessor" />
    <to uri="cxf:bean:serviceRelayEndpointWithInsertion"/>
    <process ref="InsertResponseOutHeaderProcessor" />
</route>
```

2.x では、SOAP ヘッダーは Apache Camel Message ヘッダーとの間で伝播されます。Apache Camel メッセージヘッダー名は、CXF (**org.apache.cxf.headers.Header.HEADER_LIST**) で定義されている定数である **org.apache.cxf.headers.Header.list** です。ヘッダー値は、CXF **SoapHeader** オブジェクト (**org.apache.cxf.binding.soap.SoapHeader**) の **List<>** です。次のスニペットは、**InsertResponseOutHeaderProcessor** (応答メッセージに新しい SOAP ヘッダーを挿入する) です。**InsertResponseOutHeaderProcessor** と **InsertRequestOutHeaderProcessor** の両方で SOAP ヘッダーにアクセスする方法は、実際には同じです。2 つのプロセッサーの唯一の違いは、挿入される SOAP ヘッダーの方向を設定することです。

```
public static class InsertResponseOutHeaderProcessor implements Processor {

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        // You should be able to get the header if exchange is routed from camel-cxf endpoint
        List<SoapHeader> soapHeaders = CastUtils.cast((List<?
>)exchange.getIn().getHeader(Header.HEADER_LIST));
```

```
        if (soapHeaders == null) {
            // we just create a new soap headers in case the header is null
            soapHeaders = new ArrayList<SoapHeader>();
        }

        // Insert a new header
        String xml = "<?xml version=\"1.0\" encoding=\"utf-8\"?><outofbandHeader "
            + "xmlns=\"http://cxf.apache.org/outofband/Header\" hdrAttribute=\"testHdrAttribute\" "
            + "xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\" soap:mustUnderstand=\"1\">"
            + "<name>New_testOobHeader</name><value>New_testOobHeaderValue</value>
</outofbandHeader>";
        SoapHeader newHeader = new SoapHeader(soapHeaders.get(0).getName(),
                    DOMUtils.readXml(new StringReader(xml)).getDocumentElement());
        // make sure direction is OUT since it is a response message.
        newHeader.setDirection(Direction.DIRECTION_OUT);
        //newHeader.setMustUnderstand(false);
        soapHeaders.add(newHeader);

    }

}
```

PAYLOAD モードで SOAP ヘッダーを取得および設定する方法

PAYLOAD モードで SOAP メッセージ (**CxfPayload** オブジェクト) にアクセスする方法はすでに示しました (「PAYLOAD データ形式の camel-cxf エンドポイントのメッセージを処理する方法」)。

CxfPayload オブジェクトを取得したら、DOM 要素 (SOAP ヘッダー) の **List** を返す **CxfPayload.getHeaders()** メソッドを呼び出すことができます。

```
from(getRouterEndpointURI()).process(new Processor() {
    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> payload = exchange.getIn().getBody(CxfPayload.class);
        List<Source> elements = payload.getBodySources();
        assertNotNull("We should get the elements here", elements);
        assertEquals("Get the wrong elements size", 1, elements.size());

        Element el = new XmlConverter().toDOMElement(elements.get(0));
        elements.set(0, new DOMSource(el));
        assertEquals("Get the wrong namespace URI", "http://camel.apache.org/pizza/types",
                el.getNamespaceURI());

        List<SoapHeader> headers = payload.getHeaders();
        assertNotNull("We should get the headers here", headers);
        assertEquals("Get the wrong headers size", headers.size(), 1);
        assertEquals("Get the wrong namespace URI",
                ((Element)(headers.get(0).getObject())).getNamespaceURI(),
                "http://camel.apache.org/pizza/types");
    }

})
.to(getServiceEndpointURI());
```

Camel 2.16.0 以降では、「POJO モードで SOAP ヘッダーを取得および設定する方法」 で説明したの

と同じ方法でSOAPヘッダを設定または取得することが可能です。**org.apache.cxf.headers.Header.list** ヘッダーを使用して、SOAP ヘッダーのリストを取得および設定できるようになりました。これは、ある Camel CXF エンドポイントから別のエンドポイント (SOAP クライアント → Camel → CXF サービス) に転送するルートがある場合、SOAP クライアントによって送信された SOAP ヘッダーも CXF サービスに転送されるようになったことを意味します。ヘッダーを転送したくない場合は、**org.apache.cxf.headers.Header.list** Camel ヘッダーからヘッダーを削除します。

SOAP ヘッダーは MESSAGE モードでは使用できません

SOAP 処理がスキップされるため、SOAP ヘッダーは **MESSAGE** モードでは使用できません。

APACHE CAMEL から SOAP 障害を出力する方法

CXF エンドポイントを使用して SOAP リクエストを使用している場合は、camel コンテキストから SOAP **Fault** を出力する必要がある場合があります。基本的に、これを行うには **throwFault** DSL を使用できます。**POJO**、**PAYLOAD**、および **MESSAGE** データ形式で機能します。SOAP 障害は次のように定義できます。

```
SOAP_FAULT = new SoapFault(EXCEPTION_MESSAGE, SoapFault.FAULT_CODE_CLIENT);
Element detail = SOAP_FAULT.getOrCreateDetail();
Document doc = detail.getOwnerDocument();
Text tn = doc.createTextNode(DETAIL_TEXT);
detail.appendChild(tn);
```

あとは好きなように投げてください

```
from(routerEndpointURI).setFaultBody(constant(SOAP_FAULT));
```

CXF エンドポイントが **MESSAGE** データ形式で動作している場合、メッセージボディーに SOAP 障害メッセージを設定し、メッセージヘッダーにレスポンスコードを設定できます。

```
from(routerEndpointURI).process(new Processor() {

    public void process(Exchange exchange) throws Exception {
        Message out = exchange.getOut();
        // Set the message body with the
        out.setBody(this.getClass().getResourceAsStream("SoapFaultMessage.xml"));
        // Set the response code here
        out.setHeader(org.apache.cxf.message.Message.RESPONSE_CODE, new Integer(500));
    }

});
```

同じことが POJO データ形式にも当てはまります。次のように、**Out** 本文に SOAP 障害を設定し、**Message.setFault (true)** を呼び出すことでそれがエラーであることを示すこともできます。

```
from("direct:start").onException(SoapFault.class).maximumRedeliveries(0).handled(true)
    .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            SoapFault fault = exchange
                .getProperty(Exchange.EXCEPTION_CAUGHT, SoapFault.class);
            exchange.getOut().setFault(true);
            exchange.getOut().setBody(fault);
```

```
        }

    }).end().to(serviceURI);
```

CXF エンドポイントのリクエストとレスポンスのコンテキストを伝播する方法

cxf client API は、要求と応答のコンテキストで操作を呼び出す方法を提供します。CXF エンドポイントプロデューサーを使用して外部 Web サービスを呼び出す場合は、次のコードを使用して、リクエストコンテキストを設定し、レスポンスコンテキストを取得できます。

```
        CxfExchange exchange = (CxfExchange)template.send(getJaxwsEndpointUri(), new
Processor() {
            public void process(final Exchange exchange) {
                final List<String> params = new ArrayList<String>();
                params.add(TEST_MESSAGE);
                // Set the request context to the inMessage
                Map<String, Object> requestContext = new HashMap<String, Object>();
                requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
JAXWS_SERVER_ADDRESS);
                exchange.getIn().setBody(params);
                exchange.getIn().setHeader(Client.REQUEST_CONTEXT , requestContext);
                exchange.getIn().setHeader(CxfConstants.OPERATION_NAME,
GREET_ME_OPERATION);
            }
        });
        org.apache.camel.Message out = exchange.getOut();
        // The output is an object array, the first element of the array is the return value
        Object\[\] output = out.getBody(Object\[\].class);
        LOG.info("Received output text: " + output\[0\]);
        // Get the response context form outMessage
        Map<String, Object> responseContext =
CastUtils.cast((Map)out.getHeader(Client.RESPONSE_CONTEXT));
        assertNotNull(responseContext);
        assertEquals("Get the wrong wsdl opertion name", "
{http://apache.org/hello_world_soap_http}greetMe",
                     responseContext.get("javax.xml.ws.wsdl.operation").toString());
```

アタッチメントサポート

POJO モード: SOAP with Attachment と MTOM の両方がサポートされています (MTOM を有効にするためのペイロードモードの例を参照してください)。 ただし、SOAP with Attachment はテストされていません。 添付は POJO にマーシャリングおよびアンマーシャリングされるため、通常、ユーザーは添付自体を処理する必要はありません。 2.1 以降、添付は Camel メッセージの添付に伝播されます。 そのため、Camel Message API で添付を取得することができます。

```
DataHandler Message.getAttachment(String id)
```

をクリックします。

ペイロードモード: MTOM は 2.1 以降でサポートされています。添付は、上記の Camel Message API によって取得できます。アタッチメント付きの SOAP は、このモードでは SOAP 処理がないため、サポートされていません。

MTOM を有効にするには、CXF エンドポイントプロパティー mtom_enabled を **true** に設定します。(Spring でしかできません。)

```
<cxf:cxfEndpoint id="routerEndpoint"
address="http://localhost:${CXFTestSupport.port1}/CxfMtomRouterPayloadModeTest/jaxws-
mtom/hello"
        wsdlURL="mtom.wsdl"
        serviceName="ns:HelloService"
        endpointName="ns:HelloPort"
        xmlns:ns="http://apache.org/camel/cxf/mtom_feature">

    <cxf:properties>
        <!--  enable mtom by setting this property to true -->
        <entry key="mtom-enabled" value="true"/>

        <!--  set the camel-cxf endpoint data fromat to PAYLOAD mode -->
        <entry key="dataFormat" value="PAYLOAD"/>
    </cxf:properties>
```

ペイロードモードで CXF エンドポイントに送信する添付付きの Camel メッセージを生成できます。

```
Exchange exchange = context.createProducerTemplate().send("direct:testEndpoint", new
Processor() {

    public void process(Exchange exchange) throws Exception {
        exchange.setPattern(ExchangePattern.InOut);
        List&lt;Source> elements = new ArrayList&lt;Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.REQ_MESSAGE)).getDocumentElement()));
        CxfPayload<SoapHeader> body = new CxfPayload<SoapHeader>(new ArrayList<SoapHeader>
(),
            elements, null);
        exchange.getIn().setBody(body);
        exchange.getIn().addAttachment(MtomTestHelper.REQ_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.REQ_PHOTO_DATA,
"application/octet-stream")));

        exchange.getIn().addAttachment(MtomTestHelper.REQ_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.requestJpeg, "image/jpeg")));

    }

});

// process response

CxfPayload<SoapHeader> out = exchange.getOut().getBody(CxfPayload.class);
Assert.assertEquals(1, out.getBody().size());

Map<String, String> ns = new HashMap<String, String>();
ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
ns.put("xop", MtomTestHelper.XOP_NS);

XPathUtils xu = new XPathUtils(ns);
Element oute = new XmlConverter().toDOMElement(out.getBody().get(0));
```

```
Element ele = (Element)xu.getValue("//ns:DetailResponse/ns:photo/xop:Include", oute,
                        XPathConstants.NODE);
String photoId = ele.getAttribute("href").substring(4); // skip "cid:"

ele = (Element)xu.getValue("//ns:DetailResponse/ns:image/xop:Include", oute,
                        XPathConstants.NODE);
String imageId = ele.getAttribute("href").substring(4); // skip "cid:"

DataHandler dr = exchange.getOut().getAttachment(photoId);
Assert.assertEquals("application/octet-stream", dr.getContentType());
MtomTestHelper.assertEquals(MtomTestHelper.RESP_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

dr = exchange.getOut().getAttachment(imageId);
Assert.assertEquals("image/jpeg", dr.getContentType());

BufferedImage image = ImageIO.read(dr.getInputStream());
Assert.assertEquals(560, image.getWidth());
Assert.assertEquals(300, image.getHeight());
```

ペイロードモードで CXF エンドポイントから受信した Camel メッセージを使用することもできます。

```
public static class MyProcessor implements Processor {

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> in = exchange.getIn().getBody(CxfPayload.class);

        // verify request
        assertEquals(1, in.getBody().size());

        Map<String, String> ns = new HashMap<String, String>();
        ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
        ns.put("xop", MtomTestHelper.XOP_NS);

        XPathUtils xu = new XPathUtils(ns);
        Element body = new XmlConverter().toDOMElement(in.getBody().get(0));
        Element ele = (Element)xu.getValue("//ns:Detail/ns:photo/xop:Include", body,
                                XPathConstants.NODE);
        String photoId = ele.getAttribute("href").substring(4); // skip "cid:"
        assertEquals(MtomTestHelper.REQ_PHOTO_CID, photoId);

        ele = (Element)xu.getValue("//ns:Detail/ns:image/xop:Include", body,
                                XPathConstants.NODE);
        String imageId = ele.getAttribute("href").substring(4); // skip "cid:"
        assertEquals(MtomTestHelper.REQ_IMAGE_CID, imageId);

        DataHandler dr = exchange.getIn().getAttachment(photoId);
        assertEquals("application/octet-stream", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.REQ_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

        dr = exchange.getIn().getAttachment(imageId);
        assertEquals("image/jpeg", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.requestJpeg,
IOUtils.readBytesFromStream(dr.getInputStream()));
```

```
        // create response
        List&lt;Source> elements = new ArrayList&lt;Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.RESP_MESSAGE)).getDocumentElement()));
        CxfPayload&lt;SoapHeader> sbody = new CxfPayload&lt;SoapHeader>(new
ArrayList&lt;SoapHeader>(),
            elements, null);
        exchange.getOut().setBody(sbody);
        exchange.getOut().addAttachment(MtomTestHelper.RESP_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.RESP_PHOTO_DATA,
"application/octet-stream")));

        exchange.getOut().addAttachment(MtomTestHelper.RESP_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.responseJpeg, "image/jpeg")));

    }
}
```

メッセージモード: メッセージをまったく処理しないため、添付はサポートされていません。

CXF_MESSAGE モード: MTOM がサポートされ、添付は上記の Camel Message API によって取得できます。マルチパート (つまり MTOM) メッセージを受信する場合、デフォルトの **SOAPMessage** から **String** へのコンバーターは、本文で完全なマルチパートペイロードを提供することに注意してください。**String** として SOAP XML だけが必要な場合は、 **message.getSOAPPart()** でメッセージボディーを設定でき、Camel convert が残りの作業を実行できます。

スタックトレース情報を伝播させる方法

Java の例外がサーバー側で発生したときに、例外のスタックトレースがフォールトメッセージにマーシャリングされてクライアントに返されるように、CXF エンドポイントを設定することができます。この機能を有効にするには、以下のように **cxfEndpoint** 要素で、**dataFormat** を **PAYLOAD** に設定し、**faultStackTraceEnabled** プロパティーを **true** に設定します。

```
<cxf:cxfEndpoint id="router" address="http://localhost:9002/TestMessage"
    wsdlURL="ship.wsdl"
    endpointName="s:TestSoapEndpoint"
    serviceName="s:TestService"
    xmlns:s="http://test">
  <cxf:properties>
    <!-- enable sending the stack trace back to client; the default value is false-->
    <entry key="faultStackTraceEnabled" value="true" /> <entry key="dataFormat" value="PAYLOAD"
/>
  </cxf:properties>
</cxf:cxfEndpoint>
```

セキュリティー上の理由から、スタックトレースには原因となる例外 (つまりスタックトレースの **Caused by** 以降の部分) は含まれません。スタックトレースに原因となる例外を含めたい場合は、以下のように **cxfEndpoint** 要素の **exceptionMessageCauseEnabled** プロパティーを **true** に設定します。

```
<cxf:cxfEndpoint id="router" address="http://localhost:9002/TestMessage"
    wsdlURL="ship.wsdl"
    endpointName="s:TestSoapEndpoint"
    serviceName="s:TestService"
    xmlns:s="http://test">
```

```
  <cxf:properties>
    <!-- enable to show the cause exception message and the default value is false -->
    <entry key="exceptionMessageCauseEnabled" value="true" />
    <!-- enable to send the stack trace back to client,  the default value is false-->
    <entry key="faultStackTraceEnabled" value="true" />
    <entry key="dataFormat" value="PAYLOAD" />
  </cxf:properties>
</cxf:cxfEndpoint>
```

警告

exceptionMessageCauseEnabled フラグは、テストおよび診断目的でのみ有効にしてください。サーバーにおいて例外の元の原因を隠すことで、敵対的なユーザーがサーバーを調査しにくくするのが、通常の実践的なやり方です。

PAYLOAD モードでのストリーミングのサポート

2.8.2 では、camel-cxf コンポーネントは、PAYLOAD モードの使用時に受信メッセージのストリーミングをサポートするようになりました。以前は、着信メッセージは完全に DOM 解析されていました。大きなメッセージの場合、これには時間がかかり、大量のメモリーが使用されます。2.8.2 以降、着信メッセージは、ルーティング中に javax.xml.transform.Source のままにすることができ、何もペイロードを変更しない場合は、ターゲットの宛先に直接ストリーミングできます。一般的な単純なプロキシーの使用例 (例: from ("cxf:...").to ("cxf:...")) では、これによりパフォーマンスが大幅に向上するだけでなく、メモリー要件が大幅に削減されます。

ただし、ストリーミングが適切でない、または望ましくない場合があります。ストリーミングの性質上、無効な着信 XML は、処理チェーンの後半までキャッチされない場合があります。また、特定のアクションでは、とにかくメッセージを DOM 解析する必要がある場合があります (WS-Security やメッセージトレースなど)。この場合、ストリーミングの利点は制限されます。この時点で、ストリーミングを制御する方法は 2 つあります。

- エンドポイントプロパティー: "allowStreaming=false" をエンドポイントプロパティーとして追加して、ストリーミングのオン/オフを切り替えることができます。
- Component プロパティー: CxfComponent オブジェクトには、そのコンポーネントから作成されたエンドポイントのデフォルトを設定できる allowStreaming プロパティーもあります。
- グローバルシステムプロパティー: org.apache.camel.component.cxf.streaming のシステムプロパティーを false に追加してオフにすることができます。これによりグローバルなデフォルトが設定されますが、上記のエンドポイントプロパティーを設定すると、そのエンドポイントのこの値がオーバーライドされます。

一般的な CXF ディスパッチモードの使用

2.8.0 から、camel-cxf コンポーネントは、任意の構造 (つまり、特定の XML スキーマにバインドされていない) のメッセージを転送できるジェネリック CXF ディスパッチモード をサポートしています。このモードを使用するには、CXF エンドポイントの wsdlURL および serviceClass 属性の指定を省略します。

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/SoapContext/SoapAnyPort">
```

```
  <cxf:properties>
    <entry key="dataFormat" value="PAYLOAD"/>
  </cxf:properties>
</cxf:cxfEndpoint>
```

デフォルトの CXF ディスパッチクライアントは、特定の SOAPAction ヘッダーを送信しないことに注意してください。したがって、ターゲットサービスが特定の SOAPAction 値を必要とする場合、キー SOAPAction (大文字と小文字を区別しない) を使用して Camel ヘッダーで提供されます。

75.1. {WILDFLY} の CXF コンシューマー

{wildfly} の camel-cxf コンシューマーの設定は、スタンドアロンの Camel の設定とは異なります。プロデューサーエンドポイントは通常どおりに機能します。

{wildfly} では、camel-cxf コンシューマーは、コンテナーによって提供されるデフォルトの Undertow HTTP サーバーを利用します。サーバーは undertow サブシステム設定内で定義されます。以下は、standalone.xml からのデフォルト設定の抜粋です。

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default" />
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true" />
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true" />
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content" />
            <filter-ref name="server-header" />
            <filter-ref name="x-powered-by-header" />
            <http-invoker security-realm="ApplicationRealm" />
        </host>
    </server>
</subsystem>
```

この例では、Undertow は、http および https ソケットバインディングで指定されたインターフェイス/ポートをリッスンするように設定されています。デフォルトでは、これは http の場合はポート 8080、https の場合は 8443 です。

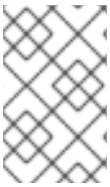
たとえば、異なるホストまたはポートの組み合わせを使用してエンドポイントコンシューマーを設定すると、サーバーログファイル内に警告が表示されます。たとえば、次のホストとポートの設定は無視されます。

```
<cxf:rsServer id="cxfRsConsumer"
              address="http://somehost:1234/path/to/resource"
              serviceClass="org.example.ServiceClass" />
```

```
<cxf:cxfEndpoint id="cxfWsConsumer"
                 address="http://somehost:1234/path/to/resource"
                 serviceClass="org.example.ServiceClass" />
```

```
[org.wildfly.extension.camel] (pool-2-thread-1) Ignoring configured host:
http://somehost:1234/path/to/resource
```

ただし、コンシューマーはデフォルトのホストとポート localhost:8080 または localhost:8443 で引き続き利用できます。



注記

camel-cxf コンシューマーを使用するアプリケーションは、WAR としてパッケージ化する **必要があります**。以前の {wildfly-camel} リリースでは、JAR などの他のタイプのアーカイブが許可されていましたが、これはサポートされなくなりました。

75.1.1. 代替ポートの設定

代替ポートを受け入れる場合は、{wildfly} サブシステム設定を介してこれらを設定する必要があります。これは、サーバーのドキュメントで説明されています。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html/configuration_guide/configuring_the_web_serv

75.1.2. SSL の設定

SSL を設定するには、{wildfly} SSL 設定ガイドを参照してください。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html-single/how_to_configure_server_security/#configure_one_way_and_two_way_ssl_tls_for_application

75.1.3. Elytron を使用したセキュリティーの設定

{wildfly-camel} は、Elytron セキュリティーフレームワークを使用して camel-cxf コンシューマーエンドポイントを保護することをサポートしています。

75.1.3.1. セキュリティードメインの設定

Elytron を使用して {wildfly-camel} アプリケーションを保護するには、WAR デプロイメントの **WEB-INF/jboss-web.xml** 内でアプリケーションセキュリティードメインを参照する必要があります。

```
<jboss-web>
  <security-domain>my-application-security-domain</security-domain>
</jboss-web>
```

<security-domain> 設定は、Undertow サブシステムによって定義された **<application-security-domain>** の名前を参照します。たとえば、Undertow サブシステム **<application-security-domain>** は、{wildfly} サーバーの **standalone.xml** 設定ファイル内で次のように設定されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:6.0">
    ...
    <application-security-domains>
        <application-security-domain name="my-application-security-domain" http-authentication-factory="application-http-authentication"/>
    </application-security-domains>
</subsystem>
```

<http-authentication-factory> application-http-authentication は、Elytron サブシステム内で定義されています。**application-http-authentication** は、デフォルトで、**standalone.xml** および **standalone-full.xml** サーバー設定ファイルの両方で使用できます。以下に例を示します。

```
<subsystem xmlns="urn:wildfly:elytron:1.2">
    ...
```

```
    <http>
        ...
        <http-authentication-factory name="application-http-authentication" http-server-mechanism-
factory="global" security-domain="ApplicationDomain">
            <mechanism-configuration>
                <mechanism mechanism-name="BASIC">
                    <mechanism-realm realm-name="Application Realm" />
                </mechanism>
                <mechanism mechanism-name="FORM" />
            </mechanism-configuration>
        </http-authentication-factory>
        <provider-http-server-mechanism-factory name="global" />
    </http>
    ...
</subsystem>
```

application-http-authentication という名前の **<http-authentication-factory>** は、**ApplicationDomain** と呼ばれる Elytron セキュリティードメインへの参照を保持します。

Elytron サブシステムの設定方法について詳しくは、Elytron のドキュメント を参照してください。

75.1.3.2. セキュリティー制約、認証方法、およびセキュリティーロールの設定

camel-cxf コンシューマーエンドポイントのセキュリティー制約、認証方法、およびセキュリティーロールは、WAR デプロイメント **WEB-INF/web.xml** 内で設定できます。たとえば、BASIC 認証を設定するには、次のようにします。

```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>/webservices/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>my-role</role-name>
    </auth-constraint>
  </security-constraint>
  <security-role>
    <description>The role that is required to log in to /webservices/*</description>
    <role-name>my-role</role-name>
  </security-role>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>my-realm</realm-name>
  </login-config>
</web-app>
```

サーブレット仕様で定義されている **<url-pattern>** は、Web アプリケーションのコンテキストパスに対して相対的であることに注意してください。アプリケーションが **my-app.war** としてパッケージ化されている場合、{wildfly} はコンテキストパス **/my-app** でアクセスできるようにし、**<url-patternpattern>** **/webservices/*** が **/my-app** への相対パスに適用されます。

たとえば、**http://my-server/my-app/webservices/my-endpoint** に対するリクエストは **/webservices/*** パターンに一致しますが、**http://my-server/webservices/my-endpoint** は一致しません。

{wildfly-camel} では、ベースパスがホスト Web アプリケーションコンテキストパスの外部にある camel-cxf エンドポイントコンシューマーを作成できるため、これは重要です。たとえば、**my-app.war** 内に **http://my-server/webservices/my-endpoint** の camel-cxf コンシューマーを作成することができます。

このようなアウトオブコンテキストエンドポイントのセキュリティー制約を定義するために、{wildfly-camel} はカスタム **の非標準の<url-pattern>** 規則をサポートします。この規則では、パターンの前に 3 つのスラッシュ /// を付けると、サーバーホスト名への絶対パスとして解釈されます。たとえば、**http://my-server/webservices/my-endpoint** を **my-app.war** 内でセキュリティー保護するには、次の設定を **web.xml** に追加します。

```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>///webservices/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>my-role</role-name>
    </auth-constraint>
  </security-constraint>
  <security-role>
    <description>The role that is required to log in to /webservices/*</description>
    <role-name>my-role</role-name>
  </security-role>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>my-realm</realm-name>
  </login-config>
</web-app>
```

第76章 CXF-RS コンポーネント

Camel バージョン 2.0 以降で利用可能

cxfrs: コンポーネントは、CXF でホストされている JAX-RS 1.1 および 2.0 サービスに接続するための Apache CXF との統合を提供します。

CXF をコンシューマーとして使用する場合、CXF Bean コンポーネント を使用すると、RESTful または SOAP Web サービスとしての処理からメッセージペイロードを受信する方法を考慮することができます。これには、多数のトランスポートを使用して Web サービスを利用できる可能性があります。Bean コンポーネントの設定も単純であり、Camel および CXF を使用して Web サービスを実装する最速の方法を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cxf</artifactId>
  <version>x.x.x</version>  <!-- use the same version as your Camel core version -->
</dependency>
```

76.1. URI 形式

```
cxfrs://address?options
```

address は CXF エンドポイントのアドレスを表します

```
cxfrs:bean:rsEndpoint
```

rsEndpoint は、CXFRS クライアントまたはサーバーを表す Spring Bean の名前を表します

上記のどちらのスタイルでも、次のように URI にオプションを追加できます。

```
cxfrs:bean:cxfEndpoint?resourceClasses=org.apache.camel.rs.Example
```

76.2. オプション

CXF-RS コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CXF-RS エンドポイントは、URI 構文を使用して設定されます。

```
cxfrs:beanId:address
```

パスおよびクエリーパラメーターを使用します。

76.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
beanId	既存の設定済み CxfRsEndpoint を検索する場合。bean: を接頭辞として使用する必要があります。		String
address	サービス公開アドレス。		文字列

76.2.2. クエリーパラメーター (29 個のパラメーター):

名前	説明	デフォルト	タイプ
features (common)	機能リストを CxfRs エンドポイントに設定します。		List
loggingFeatureEnabled (common)	このオプションは、インバウンドおよびアウトバウンドの REST メッセージをログに書き込む CXF ロギング機能を有効にします。	false	boolean
loggingSizeLimit (common)	ロギング機能が有効になっているときにロガーが出力するバイト数の合計サイズを制限します。		int
modelRef (common)	このオプションは、アノテーションのないリソースクラスに役立つモデルファイルを指定するために使用されます。このオプションを使用する場合、サービスクラスを省略して、ドキュメントのみのエンドポイントをエミュレートできます。		String

名前	説明	デフォルト	タイプ
providers (common)	カスタム JAX-RS プロバイダーリストを CxfRs エンドポイントに設定します。コンマで区切られたレジストリーで検索するプロバイダーのリストを含む文字列を指定できます。		String
resourceClasses (common)	REST サービスとしてエクスポートするリソースクラス。複数のクラスはコンマで区切ることができます。		List
schemaLocations (common)	着信 XML または JAXB 駆動型 JSON の検証に使用できるスキーマのロケーションを設定します。		List
skipFaultLogging (common)	このオプションは、PhaseInterceptorChain がキャッチした Fault のログ記録をスキップするかどうかを制御します。	false	boolean
bindingStyle (consumer)	リクエストとレスポンスが Camel との間でどのようにマッピングされるかを設定します。2 つの値が可能です: SimpleConsumer: このバインディングスタイルは、リクエストパラメーター、マルチパートなどを処理し、それらを IN ヘッダー、IN 添付、およびメッセージボディーにマップします。org.apache.cxf.message.MessageContentsList の低レベル処理を排除することを目的としています。また、レスポンスマッピングの柔軟性とシンプルさも向上します。一般コンシューマーのみご利用いただけます。デフォルト: デフォルトのスタイル。コンシューマーの場合、これにより MessageContentsList がルートに渡され、ルートで低レベルの処理が必要になります。これは従来のバインディングスタイルで、CXF スタックから入ってくる org.apache.cxf.message.MessageContentsList を IN メッセージボディーに単純にダンプします。ユーザーは、JAX-RS メソッド署名によって定義されたコントラクトに従ってそれを処理する責任があります。カスタム: バインディングオプションを介してカスタムバインディングを指定できます。	デフォルト	BindingStyle
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
publishedEndpointUrl (consumer)	このオプションは、リソースアドレス URL と _wadl でアクセスできる WADL から発行された endpointUrl をオーバーライドできます。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
hostnameVerifier (producer)	使用するホスト名ベリファイア。表記を使用して、レジストリーから HostnameVerifier を参照します。		HostnameVerifier
sslContextParameters (producer)	Camel SSL 設定リファレンス。表記を使用して、SSL コンテキストを参照します。		SSLContextParameters
throwExceptionOnFailure (producer)	このオプションは、CxfRsProducer に戻りコードを検査するように指示し、戻りコードが 207 より大きい場合は例外を生成します。	true	boolean
httpClientAPI (producer)	true の場合、CxfRsProducer は HttpClientAPI を使用してサービスを呼び出します。false の場合、CxfRsProducer は ProxyClientAPI を使用してサービスを呼び出します。	true	boolean
ignoreDeleteMethodMessageBody (producer)	このオプションは、HTTP API の使用時に DELETE メソッドのメッセージ本文を無視するように CxfRsProducer に指示するために使用されます。	false	boolean
maxClientCacheSize (producer)	このオプションを使用すると、キャッシュの最大サイズを設定できます。実装は、CXF クライアントまたは ClientFactoryBean を CxfProvider および CxfRsProvider にキャッシュします。	10	int
binding (advanced)	カスタム CxfBinding を使用して、Camel メッセージと CXF メッセージ間のバインディングを制御します。		CxfRsBinding

名前	説明	デフォルト	タイプ
bus (advanced)	カスタム設定の CXF バスを使用するには。		バス
continuationTimeout (advanced)	このオプションは、CXF サーバーが Jetty またはサーブレットトランスポートを使用している場合にデフォルトで CxfConsumer で使用できる CXF 継続タイムアウトを設定するために使用されます。	30000	long
cxfRsEndpointConfigurer (advanced)	このオプションは、プログラムによる方法での CXF エンドポイントの設定をサポートする org.apache.camel.component.cxf.jaxrs.CxfRsEndpointConfigurer の実装を適用できます。ユーザーは、CxfEndpointConfigurer の configureServer/Client メソッドを実装することで、CXF サーバーとクライアントを設定できます。		CxfRsEndpointConfigurer
defaultBus (advanced)	CXF エンドポイントが独自にバスを作成するときに、デフォルトのバスを設定します	false	boolean
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
performInvocation (advanced)	オプションが true の場合、Camel はリソースクラスインスタンスの呼び出しを実行し、さらに処理するために応答オブジェクトを交換に入れます。	false	boolean
propagateContexts (advanced)	このオプションが true の場合、JAXRS UriInfo、HttpHeaders、Request、および SecurityContext コンテキストは、型指定された Camel エクスチェンジプロパティーとしてカスタム CXFRS プロセッサーで使用できます。これらのコンテキストは、JAX-RS API を使用して現在のリクエストを分析するために使用できます。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

Spring 設定を使用して CXF REST エンドポイントを設定することもできます。CXF REST クライアントと CXF REST サーバーには多くの違いがあるため、それぞれに異なる設定を提供します。詳細については、スキーマファイル と CXF JAX-RS のドキュメント を参照してください。

76.3. CAMEL で REST エンドポイントを設定する方法

camel-cxf スキーマファイル には、REST エンドポイント定義の 2 つの要素があります。REST コンシューマーの場合は **cxf:rsServer**、REST プロデューサーの場合は **cxf:rsClient**。
Camel REST サービスのルート設定の例は、こちらにあります。

76.4. メッセージヘッダーから CXF プロデューサーアドレスを上書きする方法

camel-cxfrs プロデューサーは、"CamelDestinationOverrideUrl" のキーでメッセージを設定することにより、サービスアドレスのオーバーライドをサポートします。

```
// set up the service address from the message header to override the setting of CXF endpoint
 exchange.getIn().setHeader(Exchange.DESTINATION_OVERRIDE_URL,
constant(getServiceAddress()));
```

76.5. REST リクエストの使用 - シンプルなバインディングスタイル

Camel 2.11 から利用可能

Default のバインディングスタイルはかなり低レベルであり、ユーザーはルートに入ってくる **MessageContentsList** オブジェクトを手動で処理する必要があります。したがって、ルートロジックを JAX-RS 操作のメソッドシグネチャーおよびパラメーターインデックスと緊密に結合します。やや洗練されておらず、難しく、エラーが発生しやすいです。

対照的に、**SimpleConsumer** バインディングスタイルは、キャメルメッセージ内で **リクエストデータにアクセスしやすくする** ために、次のマッピングを実行します。

- JAX-RS パラメーター (@HeaderParam、@QueryParam など) は、IN メッセージヘッダーとして挿入されます。ヘッダー名は、アノテーションの値と一致します。
- リクエストエンティティー (POJO またはその他のタイプ) は、IN メッセージの本文になります。単一のエンティティーが JAX-RS メソッド署名で識別できない場合、元の **MessageContentsList** にフォールバックします。
- バイナリー **@Multipart** ボディパーツは IN メッセージの添付になり、**DataHandler**、**InputStream**、**DataSource**、および CXF の **Attachment** クラスをサポートします。
- 非バイナリー **@Multipart** ボディパーツは、IN メッセージヘッダーとしてマップされます。ヘッダー名はボディパーツ名と一致します。

さらに、次のルールが **Response マッピング** に適用されます。

- メッセージボディーのタイプが **javax.ws.rs.core.Response** (ユーザー作成のレスポンス) と異なる場合、新しい **Response** が作成され、メッセージボディーがエンティティーとして設定されます (null でない限り)。レスポンスステータスコードは、**Exchange.HTTP_RESPONSE_CODE** ヘッダーから取得されるか、存在しない場合はデフォルトで 200 OK になります。
- メッセージボディーのタイプが **javax.ws.rs.core.Response** と等しい場合、ユーザーがカスタムレスポンスを作成したことを意味し、それが尊重されて最終的なレスポンスになります。
- いずれの場合も、カスタムまたはデフォルトの **HeaderFilterStrategy** によって許可された Camel ヘッダーが HTTP 応答に追加されます。

76.5.1. Simple Binding Style の有効化

このバインディングスタイルは、コンシューマーエンドポイントの **bindingStyle** パラメーターを値 **SimpleConsumer** に設定することで有効化できます。

■

```
from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
  .to("log:TEST?showAll=true");
```

76.5.2. 異なるメソッドシグネチャーを使用したリクエストバインディングの例

以下は、Simple バインディングから期待される結果と一緒にメソッドシグネチャーのリストです。

public Response doAction(BusinessObject request);
リクエストペイロードは、元の MessageContentsList を置き換えて、IN メッセージボディーに配置されます。

public Response doAction(BusinessObject request, @HeaderParam("abcd") String abcd, @QueryParam("defg") String defg); 元の MessageContentsList を置き換えて、IN メッセージボディーに配置されたリクエストペイロード。両方のリクエストパラメーターは、abcd および defg という名前の IN メッセージヘッダーとしてマップされます。

public Response doAction(@HeaderParam("abcd") String abcd, @QueryParam("defg") String defg); 両方のリクエストパラメーターは、abcd および defg という名前の IN メッセージヘッダーとしてマップされます。元の MessageContentsList は、2 個のパラメーターしか含まれていませんが、保持されます。

public Response doAction(@Multipart(value="body1") BusinessObject request, @Multipart(value="body2") BusinessObject request2); 最初のパラメーターは body1 という名前のヘッダーとして転送され、2 番目のパラメーターはヘッダー body2 としてマップされます。元の MessageContentsList は IN メッセージボディーとして保持されます。

public Response doAction(InputStream abcd); InputStream は MessageContentsList からアンラップされ、IN メッセージボディーとして保存されます。

public Response doAction (DataHandler abcd); DataHandler は MessageContentsList からアンラップされ、IN メッセージボディーとして保存されます。

76.5.3. Simple Binding Style のその他の例

このメソッドで JAX-RS リソースクラスを指定すると、次のようになります。

```
  @POST @Path("/customers/{type}")
  public Response newCustomer(Customer customer, @PathParam("type") String type,
@QueryParam("active") @DefaultValue("true") boolean active) {
      return null;
  }
```

以下のルートでサービスを提供します。

```
  from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
     .recipientList(simple("direct:${header.operationName}"));

  from("direct:newCustomer")
     .log("Request: type=${header.type}, active=${header.active}, customerData=${body}");
```

XML ペイロードを含む次の HTTP リクエスト (Customer DTO が JAXB アノテーション付きである場合):

```
POST /customers/gold?active=true
```

```
Payload:
<Customer>
  <fullName>Raul Kripalani</fullName>
  <country>Spain</country>
  <project>Apache Camel</project>
</Customer>
```

メッセージを出力します:

```
Request: type=gold, active=true, customerData=<Customer.toString() representation>
```

リクエストの処理方法とレスポンスの書き込み方法に関するその他の例については、こちらを 参照してください。

76.6. REST リクエストの使用 - デフォルトのバインディングスタイル

CXF JAXRS フロントエンドは、JAX-RS (JSR-311) API を実装しているため、リソースクラスを REST サービスとしてエクスポートできます。また、CXF Invoker API を利用して、REST 要求を通常の Java オブジェクトメソッド呼び出しに変換します。
Camel Restlet コンポーネントとは異なり、エンドポイント内で URI テンプレートを指定する必要はありません。JSR-311 仕様に従って、CXF が REST 要求 URI からリソースクラスメソッドへのマッピングを処理します。Camel で行う必要があるのは、このメソッドリクエストを適切なプロセッサーまたはエンドポイントに委任することだけです。

これは CXFRS ルートの例です...

そして、エンドポイントの設定に使用される対応するリソースクラス...

情報:*リソースクラスに関する注意*

デフォルトでは、JAX-RS リソースクラスは**JAX-RS プロパティーの設定にのみ使用されます。エンドポイントへのメッセージのルーティング中にメソッドが実行されることは*ありません。代わりに、すべての処理を行うのはルートの責任です。

Camel 2.15 以降では、デフォルトモードの no-op サービス実装クラスとは対照的に、インターフェイスのみを提供するだけでも十分であることに注意してください。

Camel 2.15 から、**performInvocation** オプションが有効になっている場合、サービス実装が最初に呼び出され、レスポンスが Camel エクスチェンジに設定され、ルートの実行が通常どおり続行されます。これは、既存の JAX-RS 実装を Camel ルートに統合する場合や、カスタムプロセッサーで JAX-RS レスポンスを後処理する場合に役立ちます。

76.7. CAMEL-CXFRS プロデューサーを介して REST サービスを呼び出す方法

CXF JAXRS フロントエンドは、プロキシーベースのクライアント API を実装します。この API を使用すると、プロキシーを介してリモート REST サービスを呼び出すことができます。**camel-cxfrs** プロデューサーは、この プロキシー API に基づいています。
メッセージヘッダーで操作名を指定し、メッセージボディーでパラメーターを準備するだけで、camel-cxfrs プロデューサーが適切な REST リクエストを生成します。

以下に例を示します。

CXF JAXRS フロントエンドは、 http 中心のクライアント API も提供します。 この API を **camel-**

cxfrs プロデューサーから呼び出すこともできます。HTTP_PATH と HTTP_METHOD を指定し、URI オプション **httpClientAPI** を使用するか、メッセージヘッダー CxfConstants.CAMEL_CXF_RS_USING_HTTP_API を設定して、プロデューサーが http セントリッククライアント API を使用できるようにする必要があります。レスポンスオブジェクトを、メッセージヘッダー CxfConstants.CAMEL_CXF_RS_RESPONSE_CLASS で指定された型クラスに変換できます。

Camel 2.1 から、CXFRS http セントリッククライアントの cxfrs URI からクエリーパラメーターを指定することもサポートしています。

エラーフォーマットマクロ: スニペット: java.lang.IndexOutOfBoundsException: インデックス: 20、サイズ: 20

動的ルーティングをサポートするには、CxfConstants.CAMEL_CXF_RS_QUERY_MAP ヘッダーを使用して URI のクエリーパラメーターをオーバーライドし、そのパラメーターマップを設定します。

76.8. CXF のキャメルトランスポートとは

CXF では、アドレスを定義することで Web サービスを提供または使用します。アドレスの最初の部分は、使用するプロトコルを指定します。たとえば、エンドポイント設定の address="http://localhost:9000" は、localhost のポート 9000 で http プロトコルを使用してサービスが提供されることを意味します。Camel Transport を CXF に統合すると、新しいトランスポート camel が得られます。したがって、address="camel://direct:MyEndpointName" を指定して、CXF サービスアドレスを camel ダイレクトエンドポイントにバインドできます。

技術的に言えば、CXF のキャメルトランスポートは、Camel コアライブラリーを使用して CXF トランスポート API を実装するコンポーネントです。これにより、Camel のルーティングエンジンと統合パターンサポートを CXF サービスと一緒に簡単に使用できます。

76.9. CAMEL を CXF トランスポート層に統合する

Camel Transport を CXF バスに含めるには、CamelTransportFactory を使用します。これは、Spring だけでなく Java でも実行できます。

76.9.1. Spring に Camel トランスポートをセットアップする

特別なものを設定したい場合は、ApplicationContext で次のスニペットを使用できます。camel トランスポートのみをアクティブ化する場合は、アプリケーションコンテキストで何もする必要はありません。アプリに camel-cxf-transport jar (camel バージョンが 2.7.x 未満の場合は camel-cxf.jar) を含めるとすぐに、cxf は jar をスキャンして CamelTransportFactory をロードします。

```
<!-- you don't need to specify the CamelTransportFactory configuration as it is auto load by CXF bus
-->
<bean class="org.apache.camel.component.cxf.transport.CamelTransportFactory">
  <property name="bus" ref="cxf" />
  <property name="camelContext" ref="camelContext" />
  <!-- checkException new added in Camel 2.1 and Camel 1.6.2 -->
  <!-- If checkException is true , CamelDestination will check the outMessage's
     exception and set it into camel exchange. You can also override this value
     in CamelDestination's configuration. The default value is false.
     This option should be set true when you want to leverage the camel's error
     handler to deal with fault message -->
  <property name="checkException" value="true" />
  <property name="transportIds">
    <list>
      <value>http://cxf.apache.org/transports/camel</value>
```

```
    </list>
  </property>
</bean>
```

76.9.2. プログラムによる Camel Transport の統合

Camel トランスポートは、Camel コンテキストをトランスポートファクトリーに設定するために使用できる setContext メソッドを提供します。このファクトリーを有効にするには、ファクトリーを CXF バスに登録する必要があります。これが完全な例です。

```
import org.apache.cxf.Bus;
import org.apache.cxf.BusFactory;
import org.apache.cxf.transport.ConduitInitiatorManager;
import org.apache.cxf.transport.DestinationFactoryManager;
...

BusFactory bf = BusFactory.newInstance();
Bus bus = bf.createBus();
CamelTransportFactory camelTransportFactory = new CamelTransportFactory();
// set up the CamelContext which will be use by the CamelTransportFactory
camelTransportFactory.setCamelContext(context)
// if you are using CXF higher then 2.4.x the
camelTransportFactory.setBus(bus);

// if you are lower CXF, you need to register the ConduitInitiatorManager and
DestinationFactoryManager like below
// register the conduit initiator
ConduitInitiatorManager cim = bus.getExtension(ConduitInitiatorManager.class);
cim.registerConduitInitiator(CamelTransportFactory.TRANSPORT_ID, camelTransportFactory);
// register the destination factory
DestinationFactoryManager dfm = bus.getExtension(DestinationFactoryManager.class);
dfm.registerDestinationFactory(CamelTransportFactory.TRANSPORT_ID, camelTransportFactory);
// set or bus as the default bus for cxf
BusFactory.setDefaultBus(bus);
```

76.10. SPRING で宛先とコンジットを設定する

76.10.1. Namespace

Camel トランスポートエンドポイントの設定に使用される要素は、名前空間 **http://cxf.apache.org/transports/camel** で定義されています。通常、接頭辞 **camel** を使用して参照されます。Camel トランスポート設定要素を使用するには、以下に示す行をエンドポイントの設定ファイルの beans 要素に追加する必要があります。さらに、設定要素の名前空間を **xsi:schemaLocation** 属性に追加する必要があります。

設定名前空間の追加

```
<beans ...
      xmlns:camel="http://cxf.apache.org/transports/camel
      ...
      xsi:schemaLocation="...
```

```
                http://cxf.apache.org/transports/camel
                http://cxf.apache.org/transports/camel.xsd
               ...>
```

76.10.2. destination 要素

camel:destination 要素とその子要素を使用して、Camel トランスポートサーバーエンドポイントを設定します。**camel:destination** 要素は、エンドポイントに対応する WSDL port 要素を指定する単一の属性 **name** を取ります。**name** 属性の値は、**portQName**`.camel-destination` の形式を取ります。以下の例では、エンドポイントのターゲット名前空間が**http://widgets.widgetvendor.net**の場合、WSDLフラグメント**<port binding="widgetSOAPBinding" name="widgetSOAPPort">** によって指定されたエンドポイントの設定を追加するために使用する **camel:destination** 要素を示します。

camel:destination Element

```
...
  <camel:destination name="{http://widgets/widgetvendor.net}widgetSOAPPort.http-destination>
    <camelContext id="context" xmlns="http://activemq.apache.org/camel/schema/spring">
        <route>
          <from uri="direct:EndpointC" />
          <to uri="direct:EndpointD" />
        </route>
    </camelContext>
  </camel:destination>

  <!-- new added feature since Camel 2.11.x
  <camel:destination name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-destination"
camelContextId="context" />

...
```

Spring の **camel:destination** 要素には、設定情報を指定する多数の子要素があります。以下に説明します。

要素

説明

camel-spring:camelContext

camel 宛先で camel コンテキストを指定できます

camel:camelContextRef

camel 宛先に挿入する camel コンテキスト ID

76.10.3. conduit 要素

camel:conduit 要素とその子要素を使用して、Camel トランスポートクライアントを設定します。**camel:conduit** 要素は、エンドポイントに対応する WSDL ポート要素を指定する単一の属性 **name** を取ります。**name** 属性の値は、**portQName**`.camel-conduit` の形式を取ります。たとえば、以下のコードは、エンドポイントのターゲット名前空間が **http://widgets.widgetvendor.net** の場合、WSDL フラグメント **<port binding="widgetSOAPBinding" name="widgetSOAPPort">** によって指定されていたエンドポイントの設定を追加するために使用される **camel:conduit** 要素を示しています。

http-conf:conduit Element

```
...
  <camelContext id="conduit_context" xmlns="http://activemq.apache.org/camel/schema/spring">
      <route>
         <from uri="direct:EndpointA" />
         <to uri="direct:EndpointB" />
      </route>
   </camelContext>

  <camel:conduit name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-conduit">
    <camel:camelContextRef>conduit_context</camel:camelContextRef>
  </camel:conduit>

  <!-- new added feature since Camel 2.11.x
  <camel:conduit name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-conduit"
camelContextId="conduit_context" />


  <camel:conduit name="*.camel-conduit">
  <!-- you can also using the wild card to specify the camel-conduit that you want to configure -->
    ...
  </camel:conduit>
...
```

camel:conduit 要素には、設定情報を指定する多数の子要素があります。以下に説明します。

要素

説明

camel-spring:camelContext

camel コンジットで camel コンテキストを指定できます。

camel:camelContextRef

camel コンジットに挿入する camel コンテキスト ID

76.11. ブループリントで宛先とコンジットを設定する

Camel 2.11.x から、Camel Transport は Blueprint での設定をサポートします。

ブループリントを使用している場合は、名前空間 **http://cxf.apache.org/transports/camel/blueprint** を使用して、以下のようにスキーマをインポートする必要があります。

ブループリントの設定名前空間の追加

```
<beans ...
      xmlns:camel="http://cxf.apache.org/transports/camel/blueprint"
      ...
      xsi:schemaLocation="...
                          http://cxf.apache.org/transports/camel/blueprint
                          http://cxf.apache.org/schmemas/blueprint/camel.xsd
                         ...>
```

blueprint **camel:conduit camel:destination** には camelContextId 属性が 1 つしかないため、camel 宛先での camel コンテキストの指定はサポートされていません。

```
<camel:conduit id="*.camel-conduit" camelContextId="camel1" />
<camel:destination id="*.camel-destination" camelContextId="camel1" />
```

76.12. CXF のロードバランサーとして CAMEL を使用する例

この例では、CXF で camel ロードバランシング機能を使用する方法を示します。設定ファイルを CXF にロードし、アドレス camel://direct:EndpointA および camel://direct:EndpointB でエンドポイントを公開する必要があります。

76.13. CAMEL を CXF に接続するための完全なハウツーと例

Apache Camel を使用した CXF Web サービスの JMS トランスポートの改善

第77章 DATA FORMAT コンポーネント

Camel バージョン 2.12 以降で利用可能

dataformat: コンポーネントを使用すると、Data Format を Camel コンポーネントとして使用できます。

77.1. URI 形式

```
dataformat:name:(marshal|unmarshal)[?options]
```

name はデータ形式の名前です。次に、**marshal** または **unmarshal** のいずれかでなければならない操作が続きます。オプションは、使用中の データ形式 を設定するために使用されます。どのオプションがサポートされているかについては、データ形式のドキュメントを参照してください。

77.2. データ形式オプション

Data Format コンポーネントにはオプションがありません。

Data Format エンドポイントは、URI 構文を使用して設定されます。

```
dataformat:name:operation
```

パスおよびクエリーパラメーターを使用します。

77.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** データ形式の名前		String
operation	マーシャリングまたはアンマーシャリングのいずれかを使用するために **必要な** 操作		String

77.2.2. クエリーパラメーター(1 個のパラメーター):

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

77.3. サンプル

たとえば、JAXB データ形式 を使用するには、次のようにします。

```
from("activemq:My.Queue").
  to("dataformat:jaxb:unmarshal?contextPath=com.acme.model").
  to("mqseries:Another.Queue");
```

XML DSL では、次のようにします。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="dataformat:jaxb:unmarshal?contextPath=com.acme.model"/>
    <to uri="mqseries:Another.Queue"/>
  </route>
</camelContext>
```

第78章 DATASET コンポーネント

Camel バージョン 1.3 以降で利用可能

分散処理と非同期処理のテストは、非常に難しいことで知られています。Mock、Test、および DataSet エンドポイントは Camel テストフレームワークとうまく連携し、エンタープライズ統合パターン と Camel の幅広いコンポーネントを強力な Bean 統合と共に使用して、ユニットと統合のテストを簡素化します。

DataSet コンポーネントは、システムの負荷テストとソークテストを簡単に実行するメカニズムを提供します。メッセージのソースとして、またデータセットが受信されたことをアサートする方法として、DataSet インスタンス を作成できるようにすることで機能します。

Camel は、データセットを送信するときに スループットロガー を使用します。

78.1. URI 形式

```
dataset:name[?options]
```

name は、レジストリーで DataSet インスタンス を検索するために使用されます

Camel には、独自の DataSet を実装するためのベースとして使用できる **org.apache.camel.component.dataset.DataSet**、**org.apache.camel.component.dataset.DataSetSupport** クラスのサポート実装が付属しています。Camel には、テストに使用できるいくつかの実装も同梱されています: **org.apache.camel.component.dataset.SimpleDataSet**、**org.apache.camel.component.dataset.ListDataSet**、および **org.apache.camel.component.dataset.FileDataSet**。これらはすべて **DataSetSupport** を拡張します。

78.2. オプション

Dataset コンポーネントにはオプションがありません。

Dataset エンドポイントは、URI 構文を使用して設定されます。

```
dataset:name
```

パスおよびクエリーパラメーターを使用します。

78.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** レジストリーで検索する DataSet の名前。		DataSet

78.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
dataSetIndex (Common)	CamelDataSetIndex ヘッダーの動作を制御します。コンシューマーの場合: - off = ヘッダーは設定されません - strict/lenient = ヘッダーは設定されますプロデューサーの場合: - off = ヘッダー値は検証されず、存在しない場合は設定されません = strict =ヘッダー値が存在する必要があり、検証されます = lenient = ヘッダー値が存在する場合は検証され、存在しない場合は設定されます	lenient	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
initialDelay (consumer)	メッセージの送信を開始する前に待機する時間 (ミリ単位)。	1000	long
minRate (consumer)	DataSet に少なくともこの数のメッセージが含まれるまで待ちます	0	int
preloadSize (consumer)	ルートが初期化を完了する前にプリロード (送信) するメッセージの数を設定します	0	long
produceDelay (consumer)	メッセージがコンシューマーによって送信されるときに遅延を引き起こす遅延を指定できるようにします (遅い処理をシミュレートするため)	3	long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
assertPeriod (producer)	暫定的なアサーションがまだ有効であることを確認するために、モックエンドポイントが再アサートするまでの猶予期間を設定します。これは、たとえば、正確な数のメッセージが到着したことをアサートするために使用されます。たとえば、リンク expectedMessageCount (int) が 5 に設定されている場合、5 つ以上のメッセージが到着するとアサーションが満たされます。正確に 5 つのメッセージが到着するようにするには、それ以上メッセージが到着しないように少し待つ必要があります。これが、この setAssertPeriod (long) メソッドを使用できる目的です。デフォルトでは、この期間は無効になっています。	0	long
consumeDelay (producer)	メッセージがプロデューサによって消費されるときに遅延を引き起こす遅延を指定できるようにします (遅い処理をシミュレートするため)	0	long
expectedCount (producer)	このエンドポイントが受信するメッセージ交換の予想数を指定します。注意: 0 のメッセージを期待したい場合は、特別な注意が必要です。0 はテストの開始時に一致するため、アサート期間を設定して、テストをしばらく実行し、まだメッセージが到着していないことを確認する必要があります。;そのためにはリンク setAssertPeriod (long) を使用します。別の方法として、NotifyBuilder を使用し、モックで assertIsSatisfied () メソッドを呼び出す前に、NotifyBuilder を使用して、Camel がいくつかのメッセージのルーティングを完了したことを知ることができます。これにより、固定アサート期間を使用せずにテスト時間を短縮できます。正確に n 番目のメッセージがこのモックエンドポイントに到着することをアサートする場合は、詳細はリンク setAssertPeriod (long) メソッドも参照してください。	-1	int
reportGroup (producer)	サイズのグループに基づいてスループットログを有効にするために使用される数値。		int
resultMinimumWaitTime (producer)	ラッチが満たされるまでリンク assertIsSatisfied () が待機する最小予想時間 (ミリ秒単位) を設定します	0	long
resultWaitTime (producer)	ラッチが満たされるまでリンク assertIsSatisfied () が待機する最大時間 (ミリ秒単位) を設定します	0	long

名前	説明	デフォルト	タイプ
retainFirst (producer)	受信した Exchange の最初の n 番目の数だけを保持するように指定します。これは、ビッグデータでテストするときに使用され、このモックエンドポイントが受信するすべての Exchange のコピーを保存しないことでメモリー消費を削減します。重要: この制限を使用する場合、リンク getReceivedCounter() は受信した Exchange の実際の数を返します。たとえば、5000 の交換を受信し、最初の 10 の交換のみを保持するように設定した場合、リンク getReceivedCounter() は引き続き 5000 を返しますが、リンク getExchanges() およびリンク getReceivedExchanges() メソッドには最初の 10 の交換しかありません。このメソッドを使用する場合、他の期待値メソッドの一部はサポートされません。たとえば、リンク expectedBodiesReceived(Object...) は、受信した最初の数のボディに期待値を設定します。リンク setRetainFirst(int) メソッドとリンク setRetainLast(int) メソッドの両方を設定して、最初と最後の受信の両方を制限できます。	-1	int
retainLast (producer)	受信した Exchange の最後の n 番目の数だけを保持するように指定します。これは、ビッグデータでテストするときに使用され、このモックエンドポイントが受信するすべての Exchange のコピーを保存しないことでメモリー消費を削減します。重要: この制限を使用する場合、リンク getReceivedCounter() は受信した Exchange の実際の数を返します。たとえば、5000 の交換を受信し、最後の 20 の交換のみを保持するように設定した場合、リンク getReceivedCounter() は引き続き 5000 を返しますが、リンク getExchanges() およびリンク getReceivedExchanges() メソッドには最後の 20 の交換しかありません。このメソッドを使用する場合、他の期待値メソッドの一部はサポートされません。たとえば、リンク expectedBodiesReceived(Object...) は、受信した最初の数のボディに期待値を設定します。リンク setRetainFirst(int) メソッドとリンク setRetainLast(int) メソッドの両方を設定して、最初と最後の受信の両方を制限できます。	-1	int
sleepForEmptyTest (producer)	リンク expectedMessageCount (int) がゼロで呼び出されたときに、このエンドポイントが実際に空であることを確認するために待機するスリープを指定できるようにします	0	long
copyOnExchange (producer)	このモックエンドポイントで受信したときに受信 Exchange のディープコピーを作成するかどうかを設定します。デフォルトでは true です。	true	boolean

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

78.3. DATASET の設定

Camel は、DataSet インターフェイスを実装する Bean をレジストリーで検索します。したがって、独自の DataSet を次のように登録できます。

```
<bean id="myDataSet" class="com.mycompany.MyDataSet">
  <property name="size" value="100"/>
</bean>
```

78.4. 例

たとえば、一連のメッセージがキューに送信され、メッセージを失うことなくキューから消費されることをテストするには、次のようにします。

```
// send the dataset to a queue
from("dataset:foo").to("activemq:SomeQueue");

// now lets test that the messages are consumed correctly
from("activemq:SomeQueue").to("dataset:foo");
```

上記は、レジストリーを調べて、メッセージの作成に使用される **foo** DataSet インスタンスを見つけます。

次に、以下で説明するように **SimpleDataSet** を使用して DataSet 実装を作成し、データセットの大きさやメッセージの外観などを設定します。

78.5. DATASETSUPPORT (抽象クラス)

DataSetSupport 抽象クラスは、新しい DataSet の出発点として最適であり、派生クラスにいくつかの便利な機能を提供します。

78.5.1. DataSetSupport のプロパティー

プロパティー	タイプ	デフォルト	説明
defaultHeaders	**Map<String,Object>**	**null**	デフォルトのメッセージ本文を指定します。SimpleDataSet の場合、これは一定のペイロードです。ただし、メッセージごとにカスタムペイロードを作成する場合は、**DataSetSupport** の独自の派生を作成します。

プロパティー	タイプ	デフォルト	説明
outputTransformer	**org.apache.camel.Processor**	null	
size	**long**	**10**	送信/消費するメッセージの数を指定します。
reportCount	**long**	**-1**	進行状況を報告する前に受信するメッセージの数を指定します。大規模な負荷テストの進行状況を表示するのに役立ちます。< 0 の場合は **size**/5、0 の場合は **size**、それ以外の場合は **reportCount** 値に設定されます。

78.6. SIMPLEDATASET

SimpleDataSet は **DataSetSupport** を拡張し、デフォルトの本文を追加します。

78.6.1. SimpleDataSet の追加プロパティー

プロパティー	タイプ	デフォルト	説明
defaultBody	**Object**	**<hello>world!</hello>**	デフォルトのメッセージ本文を指定します。デフォルトでは、**SimpleDataSet** は交換ごとに同じ一定のペイロードを生成します。交換ごとにペイロードをカスタマイズする場合は、キャメル **プロセッサー** を作成し、**outputTransformer** プロパティーを設定して、それを使用するように **SimpleDataSet** を設定します。

78.7. LISTDATASET

Camel 2.17 以降で利用可能

List`DataSet` は **DataSetSupport** を拡張し、デフォルトボディーのリストを追加します。

78.7.1. ListDataSet の追加プロパティー

プロパティー	タイプ	デフォルト	説明
defaultBodies	**List<Object>**	**emptyLinkedList<Object>**	デフォルトのメッセージ本文を指定します。デフォルトでは、**ListDataSet** は **CamelDataSetIndex** を使用して **defaultBodies** のリストから定数ペイロードを選択します。ペイロードをカスタマイズする場合は、キャメル **プロセッサー** を作成し、**outputTransformer** プロパティーを設定して、それを使用するように **ListDataSet** を設定します。

プロパティー	タイプ	デフォルト	説明
size	**long**	defaultBodies リストのサイズ	送信/消費するメッセージの数を指定します。この値は、**defaultBodies** リストのサイズとは異なる場合があります。値が **defaultBodies** リストのサイズより小さい場合、一部のリスト要素は使用されません。値が **defaultBodies** リストのサイズより大きい場合、交換のペイロードは **CamelDataSetIndex** の係数と **defaultBodies** リストのサイズを使用して選択されます (つまり、**CamelDataSetIndex % defaultBodies.size()**)。

78.8. FILEDATASET

Camel 2.17 以降で利用可能

FileDataSet は **ListDataSet** を拡張し、ファイルから本文をロードするためのサポートを追加します。

78.8.1. FileDataSet の追加プロパティー

プロパティー	タイプ	デフォルト	説明
sourceFile	**File**	null	ペイロードのソースファイルを指定します
delimiter	**String**	\z	ファイルを複数のペイロードに分割するために **java.util.Scanner** によって使用される区切り文字パターンを指定します。

第79章 DIGITALOCEAN COMPONENT

Camel バージョン 2.19 以降で利用可能

DigitalOcean コンポーネントを使用すると、digitalocean-api-java (https://www.digitalocean.com/community/projects/api-client-in-java) をカプセル化することで、**Camel** を使用して DigitalOcean クラウド内のドロップレットとリソースを管理できます。DigitalOcean コントロールパネルで使い慣れたすべての機能は、この Camel コンポーネントからも利用できます。

79.1. 前提条件

有効な DigitalOcean アカウントと有効な OAuth トークンが必要です。アカウントの DigitalOcean コントロールパネルのアプリケーションと API(https://cloud.digitalocean.com/settings/applications) セクションにアクセスして、OAuth トークンを生成できます。

79.2. URI 形式

DigitalOcean コンポーネント は、次の URI 形式を使用します。

```
digitalocean://endpoint?[options]
```

ここで、**endpoint** は DigitalOcean リソースタイプです。

例 : ドロップレットを一覧表示するには:

```
digitalocean://droplets?operation=list&oAuthToken=XXXXXX&page=1&perPage=10
```

DigitalOcean コンポーネントはプロデューサーエンドポイントのみをサポートするため、ルートの開始時にこのコンポーネントを使用してチャネル内のメッセージをリッスンすることはできません。

79.3. オプション

DigitalOcean コンポーネントにはオプションがありません。

DigitalOcean エンドポイントは、URI 構文を使用して設定されます。

```
digitalocean:operation
```

パスおよびクエリーパラメーターを使用します。

79.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	指定されたリソースに対して実行する操作。		DigitalOceanOperations

79.3.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
page (producer)	ページネーションに使用します。ページ番号を強制します。	1	Integer
perPage (producer)	ページネーションに使用します。リクエストごとのアイテム数を設定します。1 ページあたりの結果の最大数は 200 です。	25	Integer
resource (producer)	**必須** 操作を実行する DigitalOcean リソースタイプ。		DigitalOceanResources
digitalOceanClient (advanced)	既存の設定済み DigitalOceanClient をクライアントとして使用するには		DigitalOceanClient
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
httpProxyHost (proxy)	必要に応じてプロキシーホストを設定する		String
httpProxyPassword (proxy)	必要に応じてプロキシーパスワードを設定します		String
httpProxyPort (proxy)	必要に応じてプロキシーポートを設定します		Integer
httpProxyUser (proxy)	必要に応じてプロキシーホストを設定します		String
oAuthToken (security)	DigitalOcean OAuth トークン		文字列

operation URI オプションまたは **CamelDigitalOceanOperation** メッセージヘッダーを使用して、エンドポイントごとに **operation** 値を指定する必要があります。

すべての **operation** 値は **DigitalOceanOperations** 列挙で定義されます。

コンポーネントによって使用されるすべての **ヘッダー** 名は、**DigitalOceanHeaders** 列挙で定義されます。

79.4. メッセージボディーの結果

返されるすべてのメッセージボディーは 、**digitalocean-api-java** ライブラリーによって提供されるオブジェクトを使用しています。

79.5. API レート制限

camel-digitalocean コンポーネントによってカプセル化された DigitalOcean REST API は、API レート制限の対象となります。API レート制限のドキュメント (https://developers.digitalocean.com/documentation/v2/#rate-limit) でメソッドごとの制限を確認できます。

79.6. アカウントエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **get** | get account info | | **com.myjeeva.digitalocean.pojo.Account** |

79.7. BLOCKSTORAGES エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the Block Storage volumes available on your account | | **List<com.myjeeva.digitalocean.pojo.Volume>** | | **get** | show information about a Block Storage volume| **CamelDigitalOceanId** Integer| **com.myjeeva.digitalocean.pojo.Volume** | | **get** | show information about a Block Storage volume by name| **CamelDigitalOceanName** String
`CamelDigitalOceanRegion` **String**| **com.myjeeva.digitalocean.pojo.Volume** | | **listSnapshots** | retrieve the snapshots that have been created from a volume | **CamelDigitalOceanId** Integer| **List<com.myjeeva.digitalocean.pojo.Snapshot>** | | **create** | create a new volume | **CamelDigitalOceanVolumeSizeGigabytes** Integer
`CamelDigitalOceanName` **String**
`CamelDigitalOceanDescription`* **String**
`CamelDigitalOceanRegion`* **String**| **com.myjeeva.digitalocean.pojo.Volume** | | **delete** | delete a Block Storage volume, destroying all data and removing it from your account| **CamelDigitalOceanId** Integer| **com.myjeeva.digitalocean.pojo.Delete**| | **delete** | delete a Block Storage volume by name| **CamelDigitalOceanName** String
`CamelDigitalOceanRegion` **String**| **com.myjeeva.digitalocean.pojo.Delete** | **attach** | attach a Block Storage volume to a Droplet| **CamelDigitalOceanId** Integer
`CamelDigitalOceanDropletId` **Integer**
`CamelDigitalOceanDropletRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | **attach** | attach a Block Storage volume to a Droplet by name| **CamelDigitalOceanName** **String**
`CamelDigitalOceanDropletId` **Integer**
`CamelDigitalOceanDropletRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | **detach** | detach a Block Storage volume from a Droplet| **CamelDigitalOceanId** Integer
`CamelDigitalOceanDropletId` **Integer**
`CamelDigitalOceanDropletRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | **attach** | detach a Block Storage volume from a Droplet by name| **CamelDigitalOceanName** **String**
`CamelDigitalOceanDropletId` **Integer**
`CamelDigitalOceanDropletRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | **resize** | resize a Block Storage volume | **CamelDigitalOceanVolumeSizeGigabytes** Integer
`CamelDigitalOceanRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | | **listActions** | retrieve all actions that have been executed on a volume | **CamelDigitalOceanId** Integer| **List<com.myjeeva.digitalocean.pojo.Action>** |

79.8. DROPLETS エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all Droplets in your account | | **List<com.myjeeva.digitalocean.pojo.Droplet>** | | **get** | show an individual droplet | **CamelDigitalOceanId** Integer| **com.myjeeva.digitalocean.pojo.Droplet** | | **create** | create a new Droplet | **CamelDigitalOceanName** String
`CamelDigitalOceanDropletImage` **String**
`CamelDigitalOceanRegion` **String**
`CamelDigitalOceanDropletSize` **String**
`CamelDigitalOceanDropletSSHKeys`* **List\<String\>**
`CamelDigitalOceanDropletEnableBackups`* **Boolean**
`CamelDigitalOceanDropletEnableIpv6`* **Boolean**
`CamelDigitalOceanDropletEnablePrivateNetworking`* **Boolean**
`CamelDigitalOceanDropletUserData`* **String**
`CamelDigitalOceanDropletVolumes`* **List\<String\>**
`CamelDigitalOceanDropletTags` **List\<String\>**|

com.myjeeva.digitalocean.pojo.Droplet | | **create** | create multiple Droplets | **CamelDigitalOceanNames List\<String\>**
`CamelDigitalOceanDropletImage` **String**
`CamelDigitalOceanRegion` **String**
`CamelDigitalOceanDropletSize` **String**
`CamelDigitalOceanDropletSSHKeys`* **List\<String\>**
`CamelDigitalOceanDropletEnableBackups`* **Boolean**
`CamelDigitalOceanDropletEnableIpv6`* **Boolean**
`CamelDigitalOceanDropletEnablePrivateNetworking`* **Boolean**
`CamelDigitalOceanDropletUserData`* **String**
`CamelDigitalOceanDropletVolumes`* **List\<String\>**
`CamelDigitalOceanDropletTags` **List\<String\>**| **com.myjeeva.digitalocean.pojo.Droplet** | | **delete** | delete a Droplet, | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Delete** | | **enableBackups** | enable backups on an existing Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **disableBackups** | disable backups on an existing Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **enableIpv6** | enable IPv6 networking on an existing Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **enablePrivateNetworking** | enable private networking on an existing Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **reboot** | reboot a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **powerCycle** | power cycle a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **shutdown** | shutdown a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **powerOff** | power off a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **powerOn** | power on a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **restore** | shutdown a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanImageId` **Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **passwordReset** | reset the password for a Droplet | **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **resize** | resize a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanDropletSize` **String**| **com.myjeeva.digitalocean.pojo.Action** | | **rebuild** | rebuild a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanImageId` **Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **rename** | rename a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Action** | | **changeKernel** | change the kernel of a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanKernelId` **Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **takeSnapshot** | snapshot a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanName`* **String**| **com.myjeeva.digitalocean.pojo.Action** | | **tag** | tag a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Response** | | **untag** | untag a Droplet | **CamelDigitalOceanId Integer**
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Response** | | **listKernels** | retrieve a list of all kernels available to a Droplet | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Kernel>** | | **listSnapshots** | retrieve the snapshots that have been created from a Droplet | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Snapshot>** | | **listBackups** | retrieve any backups associated with a Droplet | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Backup>** | | **listActions** | retrieve all actions that have been executed on a Droplet | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Action>** | | **listNeighbors** | retrieve a list of droplets that are running on the same physical server | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Droplet>** | | **listAllNeighbors** | retrieve a list of any droplets that are running on the same physical hardware | | **List<com.myjeeva.digitalocean.pojo.Droplet>** |

79.9. イメージエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list images available on your account | **CamelDigitalOceanType*** **DigitalOceanImageTypes** | **List<com.myjeeva.digitalocean.pojo.Image>** | | **ownList** | retrieve only the private images of a user | | **List<com.myjeeva.digitalocean.pojo.Image>** | | **listActions** | retrieve all actions that have been executed on a Image | **CamelDigitalOceanId Integer** | **List<com.myjeeva.digitalocean.pojo.Action>** | | **get** | retrieve information about an image (public or private) by id| **CamelDigitalOceanId Integer**| **com.myjeeva.digitalocean.pojo.Image** | | **get** | retrieve information about an public image by slug|

CamelDigitalOceanDropletImage String| **com.myjeeva.digitalocean.pojo.Image** | | **update** | update an image| **CamelDigitalOceanId** Integer
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Image** | | **delete** | delete an image| **CamelDigitalOceanId** Integer | **com.myjeeva.digitalocean.pojo.Delete** | | **transfer** | transfer an image to another region| **CamelDigitalOceanId** Integer
`CamelDigitalOceanRegion` **String**| **com.myjeeva.digitalocean.pojo.Action** | | **convert** | convert an image, for example, a backup to a snapshot| **CamelDigitalOceanId** Integer | **com.myjeeva.digitalocean.pojo.Action** |

79.10. スナップショットエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the snapshots available on your account | **CamelDigitalOceanType*** DigitalOceanSnapshotTypes | **List<com.myjeeva.digitalocean.pojo.Snapshot>** | | **get** | retrieve information about a snapshot| **CamelDigitalOceanId** Integer| **com.myjeeva.digitalocean.pojo.Snapshot** | | **delete** | delete an snapshot| **CamelDigitalOceanId** Integer | **com.myjeeva.digitalocean.pojo.Delete** |

79.11. キーエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the keys in your account | | **List<com.myjeeva.digitalocean.pojo.Key>** | | **get** | retrieve information about a key by id| **CamelDigitalOceanId** Integer| **com.myjeeva.digitalocean.pojo.Key** | | **get** | retrieve information about a key by fingerprint| **CamelDigitalOceanKeyFingerprint** String| **com.myjeeva.digitalocean.pojo.Key** | | **update** | update a key by id| **CamelDigitalOceanId** Integer
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Key** | | **update** | update a key by fingerprint| **CamelDigitalOceanKeyFingerprint** String
`CamelDigitalOceanName` **String**| **com.myjeeva.digitalocean.pojo.Key** | | **delete** | delete a key by id| **CamelDigitalOceanId** Integer | **com.myjeeva.digitalocean.pojo.Delete** | | **delete** | delete a key by fingerprint| **CamelDigitalOceanKeyFingerprint** String | **com.myjeeva.digitalocean.pojo.Delete** |

79.12. リージョンエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the regions that are available | | **List<com.myjeeva.digitalocean.pojo.Region>** |

79.13. サイズエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the sizes that are available | | **List<com.myjeeva.digitalocean.pojo.Size>** |

79.14. フローティング IP エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ----------- | | **list** | list all of the Floating IPs available on your account | | **List<com.myjeeva.digitalocean.pojo.FloatingIP>** | | **create** | create a new Floating IP assigned to a Droplet | **CamelDigitalOceanId** Integer | **List<com.myjeeva.digitalocean.pojo.FloatingIP>** | | **create** | create a new Floating IP assigned to a Region | **CamelDigitalOceanRegion** String | **List<com.myjeeva.digitalocean.pojo.FloatingIP>** | | **get** | retrieve information about a Floating IP| **CamelDigitalOceanFloatingIPAddress** String| **com.myjeeva.digitalocean.pojo.Key** | | **delete** | delete a Floating IP and remove it from your account| **CamelDigitalOceanFloatingIPAddress** String| **com.myjeeva.digitalocean.pojo.Delete** | | **assign** | assign a Floating IP to a Droplet| **CamelDigitalOceanFloatingIPAddress** String
`CamelDigitalOceanDropletId` **Integer**| **com.myjeeva.digitalocean.pojo.Action** | | **unassign** | unassign a Floating IP | **CamelDigitalOceanFloatingIPAddress** String |

com.myjeeva.digitalocean.pojo.Action | | **listActions** | retrieve all actions that have been executed on a Floating IP | **CamelDigitalOceanFloatingIPAddress** String | **List<com.myjeeva.digitalocean.pojo.Action>** |

79.15. タグのエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | ---------- | | **list** | list all of your tags | | **List<com.myjeeva.digitalocean.pojo.Tag>** | | **create** | create a Tag | **CamelDigitalOceanName** String | **com.myjeeva.digitalocean.pojo.Tag** | | **get** | retrieve an individual tag | **CamelDigitalOceanName** String | **com.myjeeva.digitalocean.pojo.Tag** | | **delete** | delete a tag | **CamelDigitalOceanName** String | **com.myjeeva.digitalocean.pojo.Delete** | | **update** | update a tag | **CamelDigitalOceanName** String
`CamelDigitalOceanNewName` **String**| **com.myjeeva.digitalocean.pojo.Tag** |

79.16. 例

アカウント情報を取得する

```
from("direct:getAccountInfo")
    .setHeader(DigitalOceanConstants.OPERATION, constant(DigitalOceanOperations.get))
    .to("digitalocean:account?oAuthToken=XXXXXX")
```

ドロップレットを作成する

```
from("direct:createDroplet")
    .setHeader(DigitalOceanConstants.OPERATION, constant("create"))
    .setHeader(DigitalOceanHeaders.NAME, constant("myDroplet"))
    .setHeader(DigitalOceanHeaders.REGION, constant("fra1"))
    .setHeader(DigitalOceanHeaders.DROPLET_IMAGE, constant("ubuntu-14-04-x64"))
    .setHeader(DigitalOceanHeaders.DROPLET_SIZE, constant("512mb"))
    .to("digitalocean:droplet?oAuthToken=XXXXXX")
```

すべてのドロップレットを一覧表示する

```
from("direct:getDroplets")
    .setHeader(DigitalOceanConstants.OPERATION, constant("list"))
    .to("digitalocean:droplets?oAuthToken=XXXXXX")
```

Retrieve information for the Droplet (dropletId = 34772987)

```
from("direct:getDroplet")
    .setHeader(DigitalOceanConstants.OPERATION, constant("get"))
    .setHeader(DigitalOceanConstants.ID, 34772987)
    .to("digitalocean:droplet?oAuthToken=XXXXXX")
```

ドロップレットのシャットダウン情報 (dropletId = 34772987)

```
from("direct:shutdown")
    .setHeader(DigitalOceanConstants.ID, 34772987)
    .to("digitalocean:droplet?operation=shutdown&oAuthToken=XXXXXX")
```

第80章 DIRECT コンポーネント

Camel バージョン 1.0 以降で利用可能

direct: コンポーネントプロデューサーがメッセージエクスチェンジを送信する際に、コンポーネントはコンシューマーを直接、同期呼び出しを提供します。
このエンドポイントは、**同じ** Camel コンテキストの既存ルートを接続するために使用できます。

ヒント

Asynchronous SEDA コンポーネントは、プロデューサーがメッセージエクスチェンジを送信するときに、コンシューマーの非同期呼び出しを提供します。

ヒント

他の camel コンテキストへの接続仮想マシン コンポーネントは、Camel コンテキストが同じ **JVM** で実行されている限り、Camel コンテキスト間の接続を提供します。

80.1. URI 形式

```
direct:someName[?options]
```

someName は、エンドポイントを一意に識別する任意の文字列にすることができます。

80.2. オプション

Direct コンポーネントは、以下に示す 3 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
block (producer)	アクティブなコンシューマーを持たない direct エンドポイントにメッセージを送信する場合、ブロックしてコンシューマーがアクティブになるのを待つようプロデューサーに指示できます。	true	boolean
timeout (producer)	ブロックが有効な場合に使用するタイムアウト値。	30000	long
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Direct エンドポイントは、URI 構文を使用して設定されます。

```
direct:name
```

パスおよびクエリーパラメーターを使用します。

80.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須**: direct エンドポイントの名前		String

80.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
block (producer)	アクティブなコンシューマーを持たない direct エンドポイントにメッセージを送信する場合、ブロックしてコンシューマーがアクティブになるのを待つようプロデューサーに指示できます。	true	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのない DIRECT エンドポイントに送信するときに、プロデューサーが例外を出力して失敗するかどうか。	false	boolean
timeout (producer)	ブロックが有効な場合に使用するタイムアウト値。	30000	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

80.3. サンプル

以下のルートでは、direct コンポーネントを使用して 2 つのルートをリンクします。

```
from("activemq:queue:order.in")
    .to("bean:orderServer?method=validate")
    .to("direct:processOrder");

from("direct:processOrder")
    .to("bean:orderService?method=process")
    .to("activemq:queue:order.out");
```

Spring DSL を使用した例:

```
<route>
 <from uri="activemq:queue:order.in"/>
 <to uri="bean:orderService?method=validate"/>
 <to uri="direct:processOrder"/>
</route>

<route>
 <from uri="direct:processOrder"/>
 <to uri="bean:orderService?method=process"/>
 <to uri="activemq:queue:order.out"/>
</route>
```

SEDA コンポーネントの例、どのように併用できるか参照してください。

80.4. 関連項目

- SEDA
- VM

第81章 ダイレクト仮想マシンコンポーネント

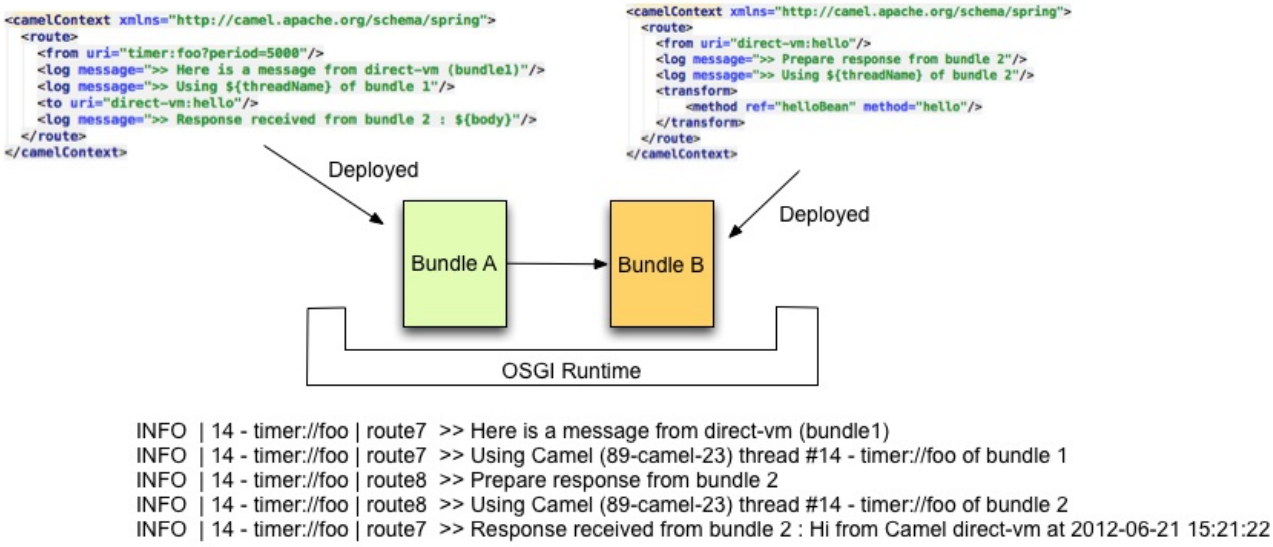
Camel バージョン 2.10 以降で利用可能

direct-vm: コンポーネントは、プロデューサがメッセージエクスチェンジを送信するときに、JVM 内のすべてのコンシューマーの直接の同期呼び出しを提供します。
このエンドポイントを使用して、同じ camel コンテキスト内の既存のルートを接続したり、**同じ** JVM 内の他の camel コンテキストから接続したりできます。

このコンポーネントは、Direct-VM が CamelContext インスタンス間の通信をサポートするという点で Direct コンポーネントとは異なります。そのため、このメカニズムを使用して Web アプリケーション間で通信できます (camel-core.jar がシステム/ブートクラスパスにある場合)。

実行時に、既存のコンシューマーを停止して新しいコンシューマーを開始することにより、新しいコンシューマーをスワップできます。
ただし、任意の時点で、特定のエンドポイントに対して最大で 1 つのアクティブなコンシューマーしか存在できません。

このコンポーネントを使用すると、後でわかるように、異なる OSGI バンドルにデプロイされたルートを接続することもできます。それらが異なるバンドルで実行されている場合でも、キャメルルートは使用します
同じスレッド。これは、Transactions - Tx を使用してアプリケーションを開発するために自動化されます。




```
INFO  | 14 - timer://foo | route7  >> Here is a message from direct-vm (bundle1)
INFO  | 14 - timer://foo | route7  >> Using Camel (89-camel-23) thread #14 - timer://foo of bundle 1
INFO  | 14 - timer://foo | route8  >> Prepare response from bundle 2
INFO  | 14 - timer://foo | route8  >> Using Camel (89-camel-23) thread #14 - timer://foo of bundle 2
INFO  | 14 - timer://foo | route7  >> Response received from bundle 2 : Hi from Camel direct-vm at 2012-06-21 15:21:22
```

81.1. URI 形式

```
direct-vm:someName
```

someName は、エンドポイントを一意に識別する任意の文字列にすることができます。

81.2. オプション

Direct 仮想マシンコンポーネントは、以下に示す 5 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
block (producer)	アクティブなコンシューマーを持たない direct エンドポイントにメッセージを送信する場合、ブロックしてコンシューマーがアクティブになるのを待つようプロデューサーに指示できます。	true	boolean
timeout (producer)	ブロックが有効な場合に使用するタイムアウト値。	30000	long
headerFilterStrategy (advanced)	プロデューサーエンドポイントにのみ適用される HeaderFilterStrategy を設定します (両方向: リクエストとレスポンス)。デフォルト値: None		HeaderFilterStrategy
propagateProperties (advanced)	プロデューサ側からコンシューマー側に、またはその逆にプロパティーを伝播するかどうか。デフォルト値: true	true	boolean
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Direct 仮想マシンエンドポイントは、URI 構文を使用して設定されます。

```
direct-vm:name
```

パスおよびクエリーパラメーターを使用します。

81.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** direct-vm エンドポイントの名前		String

81.2.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
block (producer)	アクティブなコンシューマーを持たない direct エンドポイントにメッセージを送信する場合、ブロックしてコンシューマーがアクティブになるのを待つようプロデューサーに指示できます。	true	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのない Direct 仮想マシンエンドポイントに送信するときに、プロデューサーが例外を出力して失敗するかどうか。	false	boolean
timeout (producer)	ブロックが有効な場合に使用するタイムアウト値。	30000	long
headerFilterStrategy (producer)	プロデューサーエンドポイントにのみ適用される HeaderFilterStrategy を設定します (両方向: リクエストとレスポンス)。デフォルト値: None		HeaderFilterStrategy
propagateProperties (advanced)	プロデューサ側からコンシューマー側に、またはその逆にプロパティーを伝播するかどうか。デフォルト値: true	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

81.3. サンプル

以下のルートでは、direct コンポーネントを使用して 2 つのルートをリンクします。

```
from("activemq:queue:order.in")
    .to("bean:orderServer?method=validate")
    .to("direct-vm:processOrder");
```

そして、別の OSGi バンドルなど、別の CamelContext で

```
from("direct-vm:processOrder")
    .to("bean:orderService?method=process")
    .to("activemq:queue:order.out");
```

Spring DSL を使用した例:

```
<route>
 <from uri="activemq:queue:order.in"/>
 <to uri="bean:orderService?method=validate"/>
 <to uri="direct-vm:processOrder"/>
</route>

<route>
 <from uri="direct-vm:processOrder"/>
 <to uri="bean:orderService?method=process"/>
 <to uri="activemq:queue:order.out"/>
</route>
```

81.4. 関連項目

- Direct
- SEDA
- VM

第82章 DISRUPTOR コンポーネント

Camel バージョン 2.12 以降で利用可能

disruptor: コンポーネントは、標準の SEDA コンポーネントと同様に非同期 SEDA の動作を提供しますが、標準の SEDA で使用される BlockingQueue の代わりに Disruptor を使用します。または、次のようになります。

interruptor-vm: エンドポイントはこのコンポーネントでサポートされており、標準の 仮想マシン に代わるものを提供します。SEDA コンポーネントと同様に、**ディスラプターのバッファー:** エンドポイントは **単一の** CamelContext 内でのみ表示され、永続性または回復のサポートは提供されません。また、**disruptor-vm:** エンドポイントのバッファーは、CamelContexts インスタンス間の通信をサポートするため、このメカニズムを使用して Web アプリケーション間で通信できます (**camel-disruptor.jar** が **システム/ブート** クラスパスにある場合)。

SEDA または仮想マシンコンポーネントよりも Disruptor コンポーネントを使用することを選択する主な利点は、プロデューサ間および/またはマルチキャストまたは同時コンシューマー間の競合が激しいユースケースでのパフォーマンスです。これらのケースでは、スループットの大幅な向上とレイテンシーの削減が観察されています。競合のないシナリオでのパフォーマンスは、SEDA および VM コンポーネントに匹敵します。

Disruptor は、SEDA および仮想マシンコンポーネントの動作とオプションを可能な限り模倣することを目的として実装されています。それらとの主な違いは次のとおりです。

- 使用されるバッファーのサイズは常に制限されています (デフォルトでは 1024 エクスチェンジ)。
- バッファーは常にバウンドしているため、Disruptor のデフォルトの動作は、例外を出力する代わりに、バッファーがいっぱいの間ブロックすることです。このデフォルトの動作は、コンポーネントで設定できます (オプションを参照)。
- Disruptor エンドポイントは BrowsableEndpoint インターフェイスを実装しません。そのため、現在 Disruptor にあるエクスチェンジは取得できず、エクスチェンジの量のみが取得されます。
- Disruptor は、そのコンシューマー (マルチキャストまたはその他) を静的に設定する必要があります。その場でコンシューマーを追加または削除するには、Disruptor で保留中のすべてのエクスチェンジを完全にフラッシュする必要があります。
- 再設定の結果: Disruptor 経由で送信されたデータは直接処理され、少なくとも 1 つのコンシューマーが存在する場合はなくなります。遅れて参加したユーザーは、参加した後に公開された新しいエクスチェンジのみを取得します。
- **pollTimeout** オプションは Disruptor コンポーネントではサポートされていません。
- プロデューサーが完全な Disruptor でブロックすると、スレッド割り込みに応答しません。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-disruptor</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

82.1. URI 形式

```
disruptor:someName[?options]
```

または

```
disruptor-vm:someName[?options]
```

someName は、現在の CamelContext 内 (またはコンテキスト全体) でエンドポイントを一意に識別する任意の文字列にすることができます。
disruptor-vm:)。
URI には、次の形式でクエリーオプションを追加できます。

```
?option=value&option=value&…
```

82.2. オプション

次のすべてのオプションは、**disruptor:** および **interruptor-vm:** コンポーネントの両方で有効です。

Disruptor コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
defaultConcurrent Consumers (consumer)	同時コンシューマーのデフォルト数を設定するには	1	int
defaultMultiple Consumers (consumer)	複数のコンシューマーのデフォルト値を設定するには	false	boolean
defaultProducerType (producer)	DisruptorProducerType のデフォルト値を設定するにはデフォルト値は Multi です。	Multi	DisruptorProducerType
defaultWaitStrategy (consumer)	DisruptorWaitStrategy のデフォルト値を設定するにはデフォルト値は Blocking です。	Blocking	DisruptorWaitStrategy
defaultBlockWhenFull (producer)	フル時のブロックのデフォルト値を設定するにはデフォルト値は true です。	true	boolean
queueSize (Common)	**非推奨** リングバッファーサイズを設定する場合		int
bufferSize (common)	リングバッファーサイズを設定する場合	1024	int

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Disruptor エンドポイントは、URI 構文を使用して設定されます。

```
disruptor:name
```

パスおよびクエリーパラメーターを使用します。

82.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** キューの名前		String

82.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
size (common)	Disruptors リングバッファーの最大容量は、実質的に 2 の累乗に増加します。注意: このオプションを使用する場合は、サイズを決定するキュー名で作成される最初のエンドポイントに注意してください。すべてのエンドポイントが同じサイズを使用するようにするには、すべてのエンドポイント、または作成される最初のエンドポイントでサイズオプションを設定します。	1024	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int

名前	説明	デフォルト	タイプ
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe メッセージングに Disruptor を使用できます。つまり、メッセージをキューに送信し、各コンシューマーにメッセージのコピーを受信させることができます。有効にすると、このオプションはすべてのコンシューマーエンドポイントで指定する必要があります。	false	boolean
waitStrategy (consumer)	コンシューマースレッドが新しいエクスチェンジの公開を待機するために使用するストラテジーを定義します。許可されるオプションは次のとおりです:Blocking、Sleeping、BusySpin、および Yielding。	Blocking	DisruptorWaitStrategy
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
blockWhenFull (producer)	フル Disruptorにメッセージを送信するスレッドが、リングバッファーの容量がなくなるまでブロックするかどうか。デフォルトでは、呼び出しスレッドはブロックされ、メッセージが受け入れられるまで待機します。このオプションを無効にすると、キューがいっぱいであることを示す例外が出力されます。	false	boolean
producerType (producer)	Disruptor で許可されるプロデューサーを定義します。使用できるオプションは次のとおりです。Multi は複数のプロデューサーを許可し、Single は特定の最適化を有効にします。これは、1 つの並行プロデューサー (1 つのスレッド上または同期されている) がアクティブな場合にのみ許可されます。	Multi	DisruptorProducerType
timeout (producer)	プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト (ミリ秒単位)。0 または負の値を使用して、タイムアウトを無効にすることができます。	30000	long

名前	説明	デフォルト	タイプ
waitForTaskToComplete (producer)	続行する前に呼び出し元が非同期タスクの完了を待つかどうかを指定するオプション。次の 3 つのオプションがサポートされています: Always、Never、または IfReplyExpected。最初の 2 つの値は一目瞭然です。最後の値 IfReplyExpected は、メッセージが Request Reply ベースの場合にのみ待機します。	IfReplyExpected	WaitForTaskToComplete
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

82.3. 待機戦略

待機戦略は、次のエクスチェンジが発行されるのを現在待機しているコンシューマースレッドによって実行される待機のタイプに影響します。次の戦略を選択できます。

名前	説明	アドバイス
Blocking	バリアで待機しているコンシューマーにロックと条件変数を使用するブロックストラテジー。	このストラテジーは、スループットと低レイテンシーが CPU リソースほど重要でない場合に使用できます。
スリープ状態	最初にスピンし、次に Thread.yield() を使用し、最終的にコンシューマーがバリアで待機している間に OS と JVM が許可する最小数のナノ秒を使用するスリープ戦略。	このストラテジーは、パフォーマンスと CPU リソースの間の適切な妥協点です。待機時間のスパイクは、静かな期間の後に発生する可能性があります。
BusySpin	バリアで待機しているコンシューマーにビジースピンループを使用するビジースピン戦略。	この戦略では、CPU リソースを使用して、遅延ジッターを引き起こす可能性のあるシステムコールを回避します。スレッドを特定の CPU コアにバインドできる場合に最適です。
Yielding	最初のスピン後にバリアで待機しているコンシューマーに Thread.yield() を使用する譲歩ストラテジー。	このストラテジーは、大幅なレイテンシースパイクを発生させることなく、パフォーマンスと CPU リソースの間の適切な妥協点です。

82.4. REQUEST REPLY (リクエストリプライ) の利用

Disruptor コンポーネントは、発信者が非同期ルートの完了を待機する Request Reply の使用をサポートしています。たとえば、以下のようになります。

```
from("mina:tcp://0.0.0.0:9876?textline=true&sync=true").to("disruptor:input");
from("disruptor:input").to("bean:processInput").to("bean:createResponse");
```

上記の経路では、受信リクエストを受け入れるポート 9876 に TCP リスナーがあります。リクエストは、**disruptor:input** バッファーにルーティングされます。Request Reply メッセージなので、レスポンスを待ちます。**interruptor:input** バッファーのコンシューマーが完了すると、レスポンスを元のメッ

セージレスポンスにコピーします。

82.5. 同時利用者

デフォルトでは、Disruptor エンドポイントは単一のコンシューマースレッドを使用しますが、コンシューマースレッドを同時に使用するように設定できます。したがって、スレッドプールの代わりに次を使用できます。

```
from("disruptor:stageName?concurrentConsumers=5").process(...)
```

2 つの違いについては、スレッドプールは負荷に応じて実行時に動的に増減できることに注意してください。一方、同時コンシューマーの数は常に固定されており、Disruptor によって内部的にサポートされているため、パフォーマンスが高くなります。

82.6. THREAD POOLS

次のような方法でスレッドプールを Disruptor エンドポイントに追加することに注意してください。

```
from("disruptor:stageName").thread(5).process(...)
```

Disruptor と組み合わせて使用する通常の BlockingQueue を追加することで、Disruptor を使用することによって達成されるパフォーマンスの向上の一部を効果的に無効にすることができます。代わりに、concurrentConsumers オプションを使用して、Disruptor エンドポイントでメッセージを処理するスレッドの数を直接設定することをお勧めします。

82.7. 例

以下のルートでは、Disruptor を使用してこの非同期キューにリクエストを送信し、別のスレッドでさらに処理するためにファイアアンドフォーゲットメッセージを送信し、このスレッドで一定の応答を元の呼び出し元に返すことができるようにします。

```
public void configure() throws Exception {
    from("direct:start")
        // send it to the disruptor that is async
        .to("disruptor:next")
        // return a constant response
        .transform(constant("OK"));

    from("disruptor:next").to("mock:result");
}
```

Hello World メッセージを送信し、応答が OK であることを期待します。

```
Object out = template.requestBody("direct:start", "Hello World");
assertEquals("OK", out);
```

Hello World メッセージは、さらに処理するために別のスレッドの Disruptor から消費されます。これは単体テストからのものであるため、単体テストでアサーションを実行できる mock エンドポイントに送信されます。

82.8. MULTIPLECONSUMERS の使用

この例では、2 つのコンシューマーを定義し、Spring Bean として登録しました。

```
<!-- define the consumers as spring beans -->
<bean id="consumer1" class="org.apache.camel.spring.example.FooEventConsumer"/>

<bean id="consumer2" class="org.apache.camel.spring.example.AnotherFooEventConsumer"/>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- define a shared endpoint which the consumers can refer to instead of using url -->
    <endpoint id="foo" uri="disruptor:foo?multipleConsumers=true"/>
</camelContext>
```

Disruptor foo エンドポイントで multipleConsumers=true を指定したので、これら 2 つ以上のコンシューマーに、一種の pub-sub スタイルメッセージングとしてメッセージの独自のコピーを受信させることができます。Bean は単体テストの一部であるため、メッセージをモックエンドポイントに送信するだけですが、@Consume を使用して Disruptor から消費する方法に注目してください。

```
public class FooEventConsumer {

    @EndpointInject(uri = "mock:result")
    private ProducerTemplate destination;

    @Consume(ref = "foo")
    public void doSomething(String body) {
        destination.sendBody("foo" + body);
    }

}
```

82.9. DISRUPTOR 情報の展開

必要に応じて、次の方法で JMX を使用せずにバッファーサイズなどの情報を取得できます。

```
DisruptorEndpoint disruptor = context.getEndpoint("disruptor:xxxx");
int size = disruptor.getBufferSize();
```

第83章 DNS コンポーネント

Camel バージョン 2.7 以降で利用可能

これは、DNSJava を使用して DNS クエリーを実行するための Camel の追加コンポーネントです。コンポーネントは、DNSJava の上の薄いレイヤーです。
このコンポーネントは、次の操作を提供します。

- ip、IP でドメインを解決する
- lookup、ドメインに関する情報を検索する
- dig、DNS クエリーを実行する

情報:*SUN JVM が必要です* DNSJava ライブラリーは、SUN JVM で実行する必要があります。Apache ServiceMix または Apache Karaf を使用する場合は、 **etc/jre.properties** ファイルを調整して、エクスポートされた Java プラットフォームパッケージのリストに **sun.net.spi.nameservice** を追加する必要があります。この変更を有効にするには、サーバーを再起動する必要があります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dns</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

83.1. URI 形式

DNS コンポーネントの URI スキームは次のとおりです。

```
dns://operation[?options]
```

このコンポーネントはプロデューサーのみをサポートします。

83.2. オプション

DNS コンポーネントにはオプションがありません。

DNS エンドポイントは、URI 構文を使用して設定されます。

```
dns:dnsType
```

パスおよびクエリーパラメーターを使用します。

83.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
dnsType	**必須** ルックアップのタイプ。		DnsType

83.2.2. クエリーパラメーター(1個のパラメーター):

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

83.3. ヘッダー

ヘッダー	タイプ	操作	説明
dns.domain	String	ip	ドメイン名。必須。
dns.name	String	lookup	検索する名前。必須。
dns.type		lookup、dig	ルックアップのタイプ。**org.xbill.dns.Type** の値と一致する必要があります。オプション。
dns.class		lookup、dig	ルックアップの DNS クラス。**org.xbill.dns.DClass** の値と一致する必要があります。オプション。
dns.query	String	dig	クエリー自体。必須。
dns.server	String	dig	特にクエリー用のサーバー。何も指定されていない場合は、OS によって指定されたデフォルトのものが使用されます。オプション。

83.4. 例

83.4.1. IP ルックアップ

```
<route id="IPCheck">
    <from uri="direct:start"/>
    <to uri="dns:ip"/>
</route>
```

これにより、ドメインの IP が検索されます。たとえば、www.example.com は 192.0.32.10 に解決されます。
検索する IP アドレスは、キー **"dns.domain"** を含むヘッダーで指定する必要があります。

83.4.2. DNS ルックアップ

```
<route id="IPCheck">
```

```
        <from uri="direct:start"/>
        <to uri="dns:lookup"/>
    </route>
```

これは、ドメインに関連付けられた一連の DNS レコードを返します。
検索する名前は、キー **"dns.name"** を使用してヘッダーに指定する必要があります。

83.4.3. DNS Dig

Dig は、DNS クエリーを実行する Unix コマンドラインユーティリティーです。

```
    <route id="IPCheck">
        <from uri="direct:start"/>
        <to uri="dns:dig"/>
    </route>
```

クエリーは、キー **"dns.query"** を使用してヘッダーに指定する必要があります。

83.5. DNS アクティベーションポリシー

DnsActivationPolicy を使用して、dns の状態に基づいてルートを動的に開始および停止できます。

異なるリージョンで実行されている同じコンポーネントのインスタンスがある場合、各リージョンでルートを設定して、dns がそのリージョンを指している場合にのみ有効化できます。

つまり、NYC にインスタンスがあり、SFO にインスタンスがあるとします。サービス CNAME service.example.com を nyc-service.example.com を指すように設定して、NYC インスタンスを起動し、SFO インスタンスを停止します。CNAME service.example.com を sfo-service.example.com を指すように変更すると、nyc インスタンスはそのルートを停止し、sfo はそのルートを起動します。これにより、実際のコンポーネントを再起動せずにリージョンを切り替えることができます。

```
 <bean id="dnsActivationPolicy"
class="org.apache.camel.component.dns.policy.DnsActivationPolicy">
  <property name="hostname" value="service.example.com" />
  <property name="resolvesTo" value="nyc-service.example.com" />
  <property name="ttl" value="60000" />
 </bean>

 <route id="routeId" autoStartup="false" routePolicyRef="dnsActivationPolicy">
 </route>
```

第84章 DOCKER コンポーネント

Camel バージョン 2.15 以降で利用可能

Docker と通信するための Camel コンポーネント。

Docker Camel コンポーネントは、Docker Remote API を介して docker-java を利用します。

84.1. URI 形式

```
docker://[operation]?[options]
```

operation は、Docker で実行する特定のアクションです。

84.2. 一般的なオプション

Docker コンポーネントは、以下に示す 2 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 docker 設定を使用するには		DockerConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Docker エンドポイントは、URI 構文を使用して設定されます。

```
docker:operation
```

パスおよびクエリーパラメーターを使用します。

84.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	**必須** 使用する操作		Docker 操作

84.2.2. クエリーパラメーター (20 パラメーター)

名前	説明	デフォルト	タイプ
email (Common)	ユーザーに関連付けられたメールアドレス。		String
host (Common)	**必要な** Docker ホスト	localhost	String
port (Common)	**必要な** Docker ポート	2375	Integer
requestTimeout (Common)	レスポンスのリクエストタイムアウト (秒単位)		Integer
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cmdExecFactory (advanced)	使用する DockerCmdExecFactory 実装の完全修飾クラス名	com.github.dockerjava.netty.NettyDockerCmdExecFactory	String
followRedirectFilter (advanced)	リダイレクトフィルターに従うかどうか	false	boolean
loggingFilter (advanced)	ロギングフィルターを使用するかどうか	false	boolean

名前	説明	デフォルト	タイプ
maxPerRouteConnections (advanced)	最大ルート接続	100	Integer
maxTotalConnections (advanced)	最大合計接続数	100	Integer
serverAddress (advanced)	docker レジストリーのサーバーアドレス。	https://index.docker.io/v1/	String
socket (advanced)	ソケット接続モード	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
certPath (security)	SSL 証明書チェーンを含むロケーション		String
password (security)	認証に使用するパスワード		String
secure (security)	HTTPS 通信を利用する	false	boolean
tlsVerify (security)	TLS を確認する	false	boolean
username (security)	認証するユーザー名		文字列

84.3. ヘッダーストラテジー

すべての URI オプションは、ヘッダープロパティーとして渡すことができます。メッセージヘッダーにある値は、URI パラメーターよりも優先されます。ヘッダープロパティーは、以下に示すように、**CamelDocker** で始まる URI オプションの形式を取ります

URI オプション	ヘッダープロパティー
containerId	CamelDockerContainerId

84.4. 例

次の例では、Docker からのイベントを使用します。

```
from("docker://events?host=192.168.59.103&port=2375").to("log:event");
```

次の例では、システム全体の情報について Docker にクエリーを実行します。

```
from("docker://info?host=192.168.59.103&port=2375").to("log:info");
```

84.5. 依存関係

Camel ルートで Docker を使用するには、コンポーネントを実装する **camel-docker** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-docker</artifactId>
  <version>x.x.x</version>
</dependency>
```

第85章 DOZER コンポーネント

Camel バージョン 2.15 以降で利用可能

dozer: コンポーネントは、**Camel 2.15.0** 以降の Dozer マッピングフレームワークを使用して Java Bean 間をマッピングする機能を提供します。 Camel は、タイプコンバーターとして Dozer マッピングをトリガーする機能もサポートしています。 Dozer エンドポイントと Dozer コンバーターの使用の主な違いは次のとおりです。

- エンドポイントごとに Dozer マッピング設定を管理する機能と、コンバーターレジストリーを介したグローバル設定を管理する機能。
- Dozer エンドポイントは、Camel データ形式を使用して入出力データをマーシャリング/アンマーシャリングするように設定して、単一の any-to-any 変換エンドポイントをサポートできます。
- Dozer コンポーネントを使用すると、Dozer をきめ細かく統合および拡張して、追加機能 (リテラル値のマッピング、マッピングの式の使用など) をサポートできます。

Dozer コンポーネントを使用するには、Maven ユーザーは **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dozer</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

85.1. URI 形式

Dozer コンポーネントはプロデューサーエンドポイントのみをサポートします。

```
dozer:endpointId[?options]
```

endpointId は、 Dozer エンドポイント設定を一意に識別するために使用される名前です。

Dozer エンドポイント URI の例:

```
from("direct:orderInput").
  to("dozer:transformOrder?mappingFile=orderMapping.xml&targetModel=example.XYZOrder").
  to("direct:orderOutput");
```

85.2. オプション

Dozer コンポーネントにはオプションがありません。

Dozer エンドポイントは、URI 構文を使用して設定されます。

```
dozer:name
```

パスおよびクエリーパラメーターを使用します。

85.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** 人が読める形式のマッピング名。		String

85.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
mappingConfiguration (producer)	Dozer マッピングの設定に使用する必要がある、Camel レジストリー内の DozerBeanMapperConfiguration Bean の名前。これは、Dozer の設定方法をきめ細かく制御するために使用できる mappingFile オプションの代替です。値に接頭辞を使用して、Bean が Camel レジストリーにあることを示すことを忘れないでください (例: myDozerConfig)。		DozerBeanMapper 設定
mappingFile (producer)	Dozer 設定ファイルのロケーション。デフォルトでは、ファイルはクラスパスからロードされますが、file:、classpath:、または http: を使用して、特定のロケーションから設定をロードできます。	dozerBeanMapping.xml	String
marshalId (producer)	マッピング出力を非 Java タイプにマーシャリングするために使用する Camel コンテキスト内で定義された dataFormat の ID。		String
sourceModel (producer)	マッピングで使用されるソースタイプの完全修飾クラス名。指定されている場合、マッピングへの入力は Dozer でマッピングされる前に指定された型に変換されます。		String
targetModel (producer)	**必須** マッピングで使用されるターゲットタイプの完全修飾クラス名。		String
unmarshalId (producer)	Java 以外の型からのマッピング入力をアンマーシャリングするために使用する Camel コンテキスト内で定義された dataFormat の ID。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

85.3. DOZER でのデータ形式の使用

Dozer は、Java 以外のソースとマッピングのターゲットをサポートしていないため、たとえば、XML ドキュメントを単独で Java オブジェクトにマッピングすることはできません。 幸いなことに、Camel

は Java と データフォーマット を使用したさまざまなフォーマットとの間のマーシャリングを幅広くサポートしています。 Dozer コンポーネントは、このサポートを利用して、入力および出力データを Dozer 経由で処理する前にデータ形式で渡す必要があることを指定できるようにします。 これは、Dozer への呼び出しの外でいつでも自分で行うことができますが、Dozer コンポーネントで直接サポートすることで、単一のエンドポイントを使用して Camel 内でエニーツーエニー変換を設定できます。

例として、Dozer コンポーネントを使用して、XML データ構造と JSON データ構造の間をマッピングしたいとします。 Camel コンテキストで次のデータ形式が定義されているとします。

```
<dataFormats>
  <json library="Jackson" id="myjson"/>
  <jaxb contextPath="org.example" id="myjaxb"/>
</dataFormats>
```

次に Dozer エンドポイントを設定して、JAXB データ形式を使用して入力 XML をアンマーシャリングし、Jackson を使用してマッピング出力をマーシャリングします。

```
<endpoint uri="dozer:xml2json?
marshalId=myjson&amp;unmarshalId=myjaxb&amp;targetModel=org.example.Order"/>
```

85.4. DOZER の設定

すべての Dozer エンドポイントには、ソースオブジェクトとターゲットオブジェクト間のマッピングを定義する Dozer マッピング設定ファイルが必要です。 エンドポイントで mappingFile または mappingConfiguration オプションが指定されていない場合、コンポーネントはデフォルトで META-INF/dozerBeanMapping.xml のロケーションになります。 単一のエンドポイントに複数のマッピング設定ファイルを提供するか、追加の設定オプション (イベントリスナー、カスタムコンバーターなど) を指定する必要がある場合は、**org.apache.camel.converter.dozer.DozerBeanMapperConfiguration** のインスタンスを使用できます。

```
<bean id="mapper" class="org.apache.camel.converter.dozer.DozerBeanMapperConfiguration">
  <property name="mappingFiles">
    <list>
      <value>mapping1.xml</value>
      <value>mapping2.xml</value>
    </list>
  </property>
</bean>
```

85.5. 拡張機能のマッピング

Dozer コンポーネントは、カスタムコンバーターとして Dozer マッピングフレームワークに多数の拡張機能を実装します。 これらのコンバーターは、Dozer 自体によって直接サポートされていないマッピング機能を実装します。

85.5.1. 変数のマッピング

変数マッピングを使用すると、ソースフィールドの値を使用する代わりに、Dozer 設定内の変数定義の値をターゲットフィールドにマッピングできます。 これは、ターゲットフィールドにリテラル値を割り当てることができる他のマッピングフレームワークでの定数マッピングと同等です。 変数マッピングを使用するには、マッピング設定内で変数を定義し、VariableMapper クラスから選択したターゲットフィールドにマップするだけです。

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <configuration>
    <variables>
      <variable name="CUST_ID">ACME-SALES</variable>
    </variables>
  </configuration>
  <mapping>
    <class-a>org.apache.camel.component.dozer.VariableMapper</class-a>
    <class-b>org.example.Order</class-b>
    <field custom-converter-id="_variableMapping" custom-converter-param="${CUST_ID}">
      <a>literal</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

85.5.2. カスタムマッピング

カスタムマッピングを使用すると、ソースフィールドをターゲットフィールドにマップする方法について独自のロジックを定義できます。 これらは Dozer のカスタマーコンバーターと機能が似ていますが、次の 2 つの顕著な違いがあります。

- カスタムマッピングを使用して、1 つのクラスに複数のコンバーターメソッドを含めることができます。
- カスタムマッピングを使用して Dozer 固有のインターフェイスを実装する必要はありません。

カスタムマッピングは、マッピング設定で組み込みの '_customMapping' コンバーターを使用して宣言されます。 このコンバーターのパラメーターの構文は次のとおりです。

```
[class-name][,method-name]
```

メソッド名はオプションです。Dozer コンポーネントは、マッピングに必要な入力と出力のタイプに一致するメソッドを検索します。 カスタムマッピングと設定の例を以下に示します。

```
public class CustomMapper {
    // All customer ids must be wrapped in "[ ]"
    public Object mapCustomer(String customerId) {
        return "[" + customerId + "]";
    }
}
```

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <mapping>
    <class-a>org.example.A</class-a>
    <class-b>org.example.B</class-b>
    <field custom-converter-id="_customMapping"
        custom-converter-param="org.example.CustomMapper,mapCustomer">
```

```
      <a>header.customerNum</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

85.5.3. 式のマッピング

式マッピングを使用すると、Camel の強力な 言語 機能を使用して式を評価し、結果をマッピングのターゲットフィールドに割り当てることができます。 Camel がサポートする任意の言語を式マッピングで使用できます。 式の基本的な例には、Camel メッセージヘッダーまたはエクスチェンジプロパティーをターゲットフィールドにマップする機能や、複数のソースフィールドをターゲットフィールドに連結する機能が含まれます。 マッピング式の構文は次のとおりです。

```
[language]:[expression]
```

メッセージヘッダーをターゲットフィールドにマッピングする例:

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <mapping>
    <class-a>org.apache.camel.component.dozer.ExpressionMapper</class-a>
    <class-b>org.example.B</class-b>
    <field custom-converter-id="_expressionMapping" custom-converter-
param="simple:\${header.customerNumber}">
      <a>expression</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

Dozer が EL を使用して定義された変数値を解決しようとするときのエラーを防ぐために、式内のすべてのプロパティーを \ でエスケープする必要があることに注意してください。

第86章 DRILL コンポーネント

Camel バージョン 2.19 以降で利用可能

drill: コンポーネントを使用する と、Apache Drill Cluster にクエリーを実行できます

Drill は、ビッグデータ探索用の Apache オープンソース SQL クエリーエンジンです。Drill は、業界標準のクエリー言語である ANSI SQL の親しみやすさとエコシステムを提供しながら、最新のビッグデータアプリケーションから得られる半構造化された急速に進化するデータの高性能分析をサポートするようにゼロから設計されています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-drill</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

86.1. URI 形式

```
drill://host[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

86.2. DRILL プロデューサー

プロデューサーは **CamelDrillQuery** ヘッダーを使用してクエリーを実行し、結果をボディーに入れます。

86.3. オプション

Drill コンポーネントにはオプションがありません。

Drill エンドポイントは、URI 構文を使用して設定されます。

```
drill:host
```

パスおよびクエリーパラメーターを使用します。

86.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** ZooKeeper ホスト名または IP アドレス。ホスト名または IP アドレスの代わりに local を使用して、ローカルの Drillbit に接続します		String

86.3.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
clusterId (producer)	クラスター ID https://drill.apache.org/docs/using-the-jdbc-driver/determining-the-cluster-id		文字列
directory (producer)	ZooKeeper のドリルディレクトリー		String
mode (producer)	接続モード: zk: Zookeeper drillbit: Drillbit 直接接続 https://drill.apache.org/docs/using-the-jdbc-driver/	ZK	DrillConnectionMode
port (producer)	ZooKeeper ポート番号		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

86.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第87章 DROPBOX コンポーネント

Camel バージョン 2.14 以降で利用可能

dropbox: コンポーネントを使用すると、Dropbox リモートフォルダーをメッセージのプロデューサーまたはコンシューマーとして扱うことができます。Dropbox Java Core API (このコンポーネントのリファレンスバージョンは 1.7.x) を使用すると、この camel コンポーネントには次の機能があります。

- コンシューマーとして、ファイルをダウンロードし、クエリーでファイルを検索します
- プロデューサーとして、ファイルのダウンロード、リモートディレクトリー間でのファイルの移動、ファイル/ディレクトリーの削除、ファイルのアップロード、およびクエリーによるファイルの検索

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dropbox</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

87.1. URI 形式

```
dropbox://[operation]?[options]
```

operation は、Dropbox リモートフォルダーで実行する特定のアクション (通常は CRUD アクション) です。

87.2. 操作

操作	説明
del	Dropbox 上のファイルまたはディレクトリーを削除します
get	Dropbox からファイルをダウンロードする
move	Dropbox のフォルダーからファイルを移動する
put	Dropbox にファイルをアップロードする
search	文字列クエリーに基づいて Dropbox のファイルを検索する

操作 には追加のオプションが必要です。特定の操作には必須のオプションもあります。

87.3. オプション

Dropbox API を使用するには、**accessToken** と **clientIdentifier** を取得する必要があります。
それらを取得する方法を説明している Dropbox のドキュメント を参照できます。

Dropbox コンポーネントにはオプションがありません。

Dropbox エンドポイントは、URI 構文を使用して設定されます。

```
dropbox:operation
```

パスおよびクエリーパラメーターを使用します。

87.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	**必須** Dropbox リモートフォルダーで実行する特定のアクション (通常は CRUD アクション)。		Dropbox の操作

87.3.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
accessToken (common)	**必須** 特定の Dropbox ユーザーに対して API リクエストを行うためのアクセストークン		String
client (common)	既存の DbxClient インスタンスを DropBox クライアントとして使用する場合。		DbxClientV2
clientIdentifier (common)	API リクエストを行うために登録されたアプリケーションの名前		String
localPath (common)	ローカルファイルシステムから Dropbox にアップロードするオプションのフォルダーまたはファイル。このオプションが設定されていない場合、メッセージボディーがアップロードするコンテンツとして使用されます。		String
newRemotePath (common)	宛先ファイルまたはフォルダー		String
query (common)	検索するサブストリングのスペース区切りのリスト。ファイルは、すべての部分文字列が含まれている場合にのみ一致します。このオプションが設定されていない場合、すべてのファイルが一致します。		String
remotePath (common)	移動する元のファイルまたはフォルダー		String

名前	説明	デフォルト	タイプ
uploadMode (common)	アップロードするモード。同じ名前のファイルが dropbox にすでに存在する場合、新しいファイルを追加する場合は名前が変更されます。dropbox に同じ名前のファイルがすでに存在する場合、これは上書きされます。		DropboxUploadMode
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

87.4. DEL 操作

Dropbox 上のファイルを削除します。

Camel プロデューサーとしてのみ機能します。

以下に、この操作のオプションを示します。

プロパティー	必須	説明
remotePath	**true**	Dropbox で削除するフォルダーまたはファイル

87.4.1. サンプル

```
from("direct:start")
  .to("dropbox://del?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("mock:result");
```

```
from("direct:start")
  .to("dropbox://del?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1/file1.tar.gz")
  .to("mock:result");
```

87.4.2. 結果メッセージのヘッダー

メッセージの結果には、次のヘッダーが設定されます。

プロパティー	値
DELETED_PATH	dropbox で削除されたパスの名前

87.4.3. 結果メッセージボディー

メッセージボディーの結果には、次のオブジェクトが設定されます。

オブジェクトタイプ	説明
String	dropbox で削除されたパスの名前

87.5. GET (ダウンロード) 操作

Dropbox からファイルをダウンロードします。

Camel プロデューサーまたは Camel コンシューマーとして機能します。

以下に、この操作のオプションを示します。

プロパティー	必須	説明
remotePath	**true**	Dropbox からダウンロードするフォルダーまたはファイル

87.5.1. サンプル

```
from("direct:start")
  .to("dropbox://get?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1/file1.tar.gz")
  .to("file:///home/kermit/?fileName=file1.tar.gz");

from("direct:start")
  .to("dropbox://get?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("mock:result");

from("dropbox://get?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("file:///home/kermit/");
```

87.5.2. 結果メッセージのヘッダー

メッセージの結果には、次のヘッダーが設定されます。

プロパティー	値
DOWNLOADED_FILE	単一ファイルのダウンロードの場合、ダウンロードされたリモートファイルのパス
DOWNLOADED_FILES	複数のファイルをダウンロードする場合、ダウンロードされたリモートファイルのパス

87.5.3. 結果メッセージボディー

メッセージボディーの結果には、次のオブジェクトが設定されます。

オブジェクトタイプ	説明
ByteArrayOutputStream	単一ファイルのダウンロードの場合、ダウンロードされたファイルを表すストリーム
Map<String, ByteArrayOutputStream>	複数のファイルをダウンロードする場合は、ダウンロードされたリモートファイルのパスをキーとして、ダウンロードされたファイルを表すストリームを値として持つマップ

87.6. MOVE 操作

Dropbox 上のファイルを 1 つのフォルダー間で別のフォルダーに移動します。

Camel プロデューサーとしてのみ機能します。

以下に、この操作のオプションを示します。

プロパティー	必須	説明
remotePath	**true**	移動する元のファイルまたはフォルダー
newRemotePath	**true**	宛先ファイルまたはフォルダー

87.6.1. サンプル

```
from("direct:start")
  .to("dropbox://move?
accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1&newRemotePath=/root/folder2")
  .to("mock:result");
```

87.6.2. 結果メッセージのヘッダー

メッセージの結果には、次のヘッダーが設定されます。

プロパティー	値
MOVED_PATH	ドロップボックスに移動したパスの名前

87.6.3. 結果メッセージボディー

メッセージボディーの結果には、次のオブジェクトが設定されます。

オブジェクトタイプ	説明
String	ドロップボックスに移動したパスの名前

87.7. PUT (アップロード) 操作

Dropbox にファイルをアップロードします。

キャメルプロデューサーとして活動。

以下に、この操作のオプションを示します。

プロパティー	必須	説明
uploadMode	**true**	add または force このオプションは、ドロップボックスにファイルを保存する方法を指定します。追加の場合、同じ名前のファイルがドロップボックスにすでに存在する場合、新しいファイルの名前が変更されます。強制の場合、ドロップボックスに同名のファイルがすでに存在する場合、上書きされます。
localPath	**false**	ローカルファイルシステムから Dropbox にアップロードするフォルダーまたはファイル。このオプションが設定されている場合、Camel メッセージ本文のコンテンツを含む単一のファイルとしてアップロードするよりも優先されます (メッセージ本文はバイト配列に変換されます)。
remotePath	**false**	Dropbox 上のフォルダーの宛先。プロパティーが設定されていない場合、コンポーネントはローカルパスと同じリモートパスにファイルをアップロードします。Windows を使用する場合、または絶対 localPath を使用しない場合、次のような例外が発生する可能性があります。 Caused by: java.lang.IllegalArgumentException: 'path': bad path: must start with "/": "C:/My/File" OR Caused by: java.lang.IllegalArgumentException: 'path': bad path: must start with "/": "MyFile"

87.7.1. サンプル

```
from("direct:start").to("dropbox://put?
```

```
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&localPath=/root/folder1")
  .to("mock:result");

from("direct:start").to("dropbox://put?
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&localPath=/root/folder1&remotePath=/root/f
older2")
  .to("mock:result");
```

メッセージボディーのコンテンツを含む単一のファイルをアップロードするには

```
from("direct:start")
   .setHeader(DropboxConstants.HEADER_PUT_FILE_NAME, constant("myfile.txt"))
   .to("dropbox://put?
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&remotePath=/root/folder2")
   .to("mock:result");
```

ファイルの名前は、優先順位に従ってヘッダー **DropboxConstants.HEADER_PUT_FILE_NAME** または **Exchange.FILE_NAME** に指定できます。ヘッダーが指定されていない場合、メッセージ ID (uuid) がファイル名として使用されます。

87.7.2. 結果メッセージのヘッダー

メッセージの結果には、次のヘッダーが設定されます。

プロパティー	値
UPLOADED_FILE	単一ファイルのアップロードの場合、アップロードされたリモートパスのパス
UPLOADED_FILES	複数のファイルをアップロードする場合は、アップロードされたリモートパスを含む文字列

87.7.3. 結果メッセージボディー

メッセージボディーの結果には、次のオブジェクトが設定されます。

オブジェクトタイプ	説明
String	単一ファイルのアップロードの場合、アップロード操作の結果、OK または KO
Map<String, DropboxResultCode>	複数のファイルをアップロードする場合は、アップロードされたリモートファイルのパスをキーとして、アップロード操作の結果を値として持つマップ、OK または KO

87.8. 検索操作

サブディレクトリーを含むリモート Dropbox フォルダー内を検索します。

Camel プロデューサーおよび Camel コンシューマーとして機能します。

以下に、この操作のオプションを示します。

プロパティー	必須	説明
remotePath	**true**	検索する Dropbox のフォルダー。
query	**true**	検索するサブストリングのスペース区切りのリスト。ファイルは、すべての部分文字列が含まれている場合にのみ一致します。このオプションが設定されていない場合、すべてのファイルが一致します。クエリーは、エンドポイント設定で、または Camel メッセージの **CamelDropboxQuery** ヘッダーとして提供する必要があります。

87.8.1. サンプル

```
from("dropbox://search?accessToken=XXX&clientIdentifier=XXX&remotePath=/XXX&query=XXX")
  .to("mock:result");

from("direct:start")
  .setHeader("CamelDropboxQuery", constant("XXX"))
  .to("dropbox://search?accessToken=XXX&clientIdentifier=XXX&remotePath=/XXX")
  .to("mock:result");
```

87.8.2. 結果メッセージのヘッダー

メッセージの結果には、次のヘッダーが設定されます。

プロパティー	値
FOUNDED_FILES	作成されたファイルパスのリスト

87.8.3. 結果メッセージボディー

メッセージボディーの結果には、次のオブジェクトが設定されます。

オブジェクトタイプ	説明
List<DbxEntry>	作成されたファイルパスのリスト。このオブジェクトの詳細については、Dropbox のドキュメントを参照してください。

第88章 EHCACHE コンポーネント

Camel バージョン 2.18 以降で利用可能

ehcache コンポーネントを使用すると、Ehcache 3 をキャッシュ実装として使用してキャッシュ操作を実行できます。

このコンポーネントは、プロデューサーおよびイベントベースのコンシューマーエンドポイントをサポートします。

キャッシュコンシューマーはイベントベースのコンシューマーであり、特定のキャッシュアクティビティーをリッスンして応答するために使用できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ehcache</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

88.1. URI 形式

```
ehcache://cacheName[?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=#beanRef&…**

88.2. オプション

Ehcache コンポーネントは、以下に示す 7個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	グローバルコンポーネント設定を設定します		EhcacheConfiguration
cacheManager (common)	キャッシュマネージャー		CacheManager
cacheManagerConfiguration (common)	キャッシュマネージャーの設定		設定
cacheConfiguration (common)	キャッシュの作成に使用されるデフォルトのキャッシュ設定。		CacheConfiguration<?,?>
cachesConfigurations (common)	キャッシュの作成に使用されるキャッシュ設定のマップ。		Map

名前	説明	デフォルト	タイプ
cacheConfigurationUri (common)	Ehcache XML 設定ファイルのロケーションを指す URI		String
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ehcache エンドポイントは、URI 構文を使用して設定されます。

```
ehcache:cacheName
```

パスおよびクエリーパラメーターを使用します。

88.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須** キャッシュ名		String

88.2.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
cacheManager (common)	キャッシュマネージャー		CacheManager
cacheManagerConfiguration (common)	キャッシュマネージャーの設定		設定
configurationUri (common)	Ehcache XML 設定ファイルのロケーションを指す URI		String
createCacheIfNotExist (common)	キャッシュが存在する場合、または事前設定できない場合にキャッシュを作成する必要があるかどうかを設定します。	true	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
eventFiring (consumer)	配信モードの設定 (同期、非同期)	ASYNCHRONOUS	EventFiring
eventOrdering (consumer)	配信モードを設定する (順序あり、順序なし)	ORDERED	EventOrdering
eventTypes (consumer)	リッスンするイベントのタイプを設定する	EVICTED,EXPIRED,REMOVED,CREATED,UPDATED	Set
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
action (producer)	デフォルトのキャッシュアクションを設定します。メッセージヘッダーにアクションが設定されている場合は、ヘッダーからの操作が優先されます。		String
key (producer)	デフォルトのアクションキーを設定します。メッセージヘッダーにキーが設定されている場合は、ヘッダーのキーが優先されます。		Object
configuration (advanced)	キャッシュの作成に使用されるデフォルトのキャッシュ設定。		CacheConfiguration<?,?>

名前	説明	デフォルト	タイプ
configurations (advanced)	キャッシュの作成に使用されるキャッシュ設定のマップ。		Map
keyType (advanced)	キャッシュキータイプ、デフォルトの java.lang.Object	java.lang.Object	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
valueType (advanced)	キャッシュ値の型、デフォルトは java.lang.Object	java.lang.Object	文字列

88.2.3. Message Headers Camel

ヘッダー	タイプ	説明
CamelEhcacheAction	**String**	キャッシュに対して実行される操作。有効なオプションは次のとおりです。 * CLEAR * PUT * PUT_ALL * PUT_IF_ABSENT * GET * GET_ALL * REMOVE * REMOVE_ALL * REPLACE
CamelEhcacheActionHasResult	Boolean	アクションに結果がある場合は true に設定します
CamelEhcacheActionSucceeded	**Boolean**	アクションが成功した場合は true に設定します
CamelEhcacheKey	Object	アクションに使用されるキャッシュキー
CamelEhcacheKeys	Set<Object>	以下で使用されるキーのリスト * PUT_ALL * GET_ALL * REMOVE_ALL

ヘッダー	タイプ	説明
CamelEhcacheValue	Object	キャッシュに入れる値または操作の結果
CamelEhcacheOldValue	Object	PUT_IF_ABSENT などのアクションのキーに関連付けられた古い値、または REPLACE などのアクションの比較に使用されるオブジェクト
CamelEhcacheEventType	EventType	受け取ったイベントの種類

88.3. EHCACHE ベースのべき等リポジトリーの例:

```
CacheManager manager = CacheManagerBuilder.newCacheManager(new
XmlConfiguration("ehcache.xml"));
EhcacheIdempotentRepository repo = new EhcacheIdempotentRepository(manager, "idempotent-
cache");

from("direct:in")
    .idempotentConsumer(header("messageId"), idempotentRepo)
    .to("mock:out");
```

88.4. EHCACHE ベースの集計リポジトリーの例:

```
public class EhcacheAggregationRepositoryRoutesTest extends CamelTestSupport {
    private static final String ENDPOINT_MOCK = "mock:result";
    private static final String ENDPOINT_DIRECT = "direct:one";
    private static final int[] VALUES = generateRandomArrayOfInt(10, 0, 30);
    private static final int SUM = IntStream.of(VALUES).reduce(0, (a, b) -> a + b);
    private static final String CORRELATOR = "CORRELATOR";

    @EndpointInject(uri = ENDPOINT_MOCK)
    private MockEndpoint mock;

    @Produce(uri = ENDPOINT_DIRECT)
    private ProducerTemplate producer;

    @Test
    public void checkAggregationFromOneRoute() throws Exception {
        mock.expectedMessageCount(VALUES.length);
        mock.expectedBodiesReceived(SUM);

        IntStream.of(VALUES).forEach(
```

```
            i -> producer.sendBodyAndHeader(i, CORRELATOR, CORRELATOR)
        );

        mock.assertIsSatisfied();
    }

    private Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
        if (oldExchange == null) {
            return newExchange;
        } else {
            Integer n = newExchange.getIn().getBody(Integer.class);
            Integer o = oldExchange.getIn().getBody(Integer.class);
            Integer v = (o == null ? 0 : o) + (n == null ? 0 : n);

            oldExchange.getIn().setBody(v, Integer.class);

            return oldExchange;
        }
    }

    @Override
    protected RoutesBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from(ENDPOINT_DIRECT)
                    .routeId("AggregatingRouteOne")
                    .aggregate(header(CORRELATOR))
                    .aggregationRepository(createAggregateRepository())
                    .aggregationStrategy(EhcacheAggregationRepositoryRoutesTest.this::aggregate)
                    .completionSize(VALUES.length)

.to("log:org.apache.camel.component.ehcache.processor.aggregate.level=INFO&showAll=true&mulltilin
e=true")
                        .to(ENDPOINT_MOCK);
            }
        };
    }

    protected EhcacheAggregationRepository createAggregateRepository() throws Exception {
        CacheManager cacheManager = CacheManagerBuilder.newCacheManager(new
XmlConfiguration("ehcache.xml"));
        cacheManager.init();

        EhcacheAggregationRepository repository = new EhcacheAggregationRepository();
        repository.setCacheManager(cacheManager);
        repository.setCacheName("aggregate");

        return repository;
    }
}
```

第89章 EJB コンポーネント

Camel バージョン 2.4 以降で利用可能

ejb: コンポーネント BIND EJB から Camel へのメッセージエクスチェンジ。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ejb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

89.1. URI 形式

```
ejb:ejbName[?options]
```

ejbName は、アプリケーションサーバー JNDI レジストリーで EJB をルックアップするために使用される任意の文字列です。

89.2. オプション

EJB コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
context (producer)	EJB のルックアップに使用する Context		コンテキスト
properties (producer)	コンテキストが設定されていない場合に javax.naming.Context を作成するためのプロパティー。		Properties
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

EJB エンドポイントは、URI 構文を使用して設定されます。

```
ejb:beanName
```

パスおよびクエリーパラメーターを使用します。

89.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
beanName	**必須**: 呼び出す Bean の名前を設定します。		String

89.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
method (producer)	Bean で呼び出すメソッドの名前を設定します。		String
cache (advanced)	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリー内の Bean がシングルトンスコープとして定義されている場合、キャッシュを有効にできます。	false	boolean
multiParameterArray (advanced)	**非推奨** メッセージボディーから渡されるパラメータをどのように扱うか。true はメッセージボディーがパラメータの配列であることを意味します。非推奨の注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。非推奨の注記: このオプションは Camel によって内部的に使用され、エンドユーザーが使用することを意図したものではありません。	false	boolean
parameters (advanced)	Bean の追加プロパティーの設定に使用します		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

89.3. BEAN バインディング

呼び出される Bean メソッドの選択方法 (**method** パラメーターで明示的に指定されていない場合) と、メッセージからパラメーター値が構築される方法は、Camel 内のさまざまな Bean 統合メカニズム全体で使用される Bean バインディングメカニズムによりすべて定義されます。

89.4. 例

次の例では、次のように定義された Greater EJB を使用します。

GreaterLocal.java

```
public interface GreaterLocal {
```

```
    String hello(String name);

    String bye(String name);

}
```

そして実装

GreaterImpl.java

```
@Stateless
public class GreaterImpl implements GreaterLocal {

    public String hello(String name) {
        return "Hello " + name;
    }

    public String bye(String name) {
        return "Bye " + name;
    }

}
```

89.4.1. Java DSL の使用

この例では、EJB で **hello** メソッドを呼び出します。この例は Apache OpenEJB を使用した単体テストに基づいているため、OpenEJB 設定を使用して EJB コンポーネントに **JndiContext** を設定する必要があります。

```
@Override
protected CamelContext createCamelContext() throws Exception {
    CamelContext answer = new DefaultCamelContext();

    // enlist EJB component using the JndiContext
    EjbComponent ejb = answer.getComponent("ejb", EjbComponent.class);
    ejb.setContext(createEjbContext());

    return answer;
}

private static Context createEjbContext() throws NamingException {
    // here we need to define our context factory to use OpenEJB for our testing
    Properties properties = new Properties();
    properties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"org.apache.openejb.client.LocalInitialContextFactory");

    return new InitialContext(properties);
}
```

これで、Camel ルートで EJB を使用する準備が整いました。

```
from("direct:start")
    // invoke the greeter EJB using the local interface and invoke the hello method
    .to("ejb:GreaterImplLocal?method=hello")
```

```
    .to("mock:result");
```

実際のアプリケーションサーバーで

実際のアプリケーションサーバーでは、ほとんどの場合、EJB コンポーネントに **JndiContext** をセットアップする必要はありません。これは、アプリケーションサーバーと同じ JVM にデフォルトの **JndiContext** を作成するためです。これにより、通常、JNDI レジストリーにアクセスして EJB をルックアップできます。ただし、リモート JVM などのアプリケーションサーバーにアクセスする必要がある場合は、事前にプロパティーを準備する必要があります。

89.4.2. Spring XML の使用

これは、代わりに Spring XML を使用した同じ例です。

これも単体テストに基づいているため、EJB コンポーネントをセットアップする必要があります。

```
<!-- setup Camel EJB component -->
<bean id="ejb" class="org.apache.camel.component.ejb.EjbComponent">
    <property name="properties" ref="jndiProperties"/>
</bean>

<!-- use OpenEJB context factory -->
<p:properties id="jndiProperties">
    <prop
key="java.naming.factory.initial">org.apache.openejb.client.LocalInitialContextFactory</prop>
</p:properties>
```

Camel ルートで EJB を使用する準備が整う前に:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:start"/>
        <to uri="ejb:GreaterImplLocal?method=hello"/>
        <to uri="mock:result"/>
    </route>
</camelContext>
```

89.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Bean
- Bean バインディング
- Bean インテグレーション

第90章 ELASTICSEARCH コンポーネント (非推奨)

Camel バージョン 2.11 以降で利用可能

ElasticSearch コンポーネントを使用すると、ElasticSearch サーバーとやり取りできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

90.1. URI 形式

```
elasticsearch://clusterName[?options]
```

90.2. エンドポイントオプション

Elasticsearch コンポーネントは、以下にリストされている 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
client (advanced)	エンドポイントごとにクライアントを作成する代わりに、既存の設定済み Elasticsearch クライアントを使用するには。		クライアント
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elasticsearch エンドポイントは、URI 構文を使用して設定されます。

```
elasticsearch:clusterName
```

パスおよびクエリーパラメーターを使用します。

90.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
clusterName	**必須** クラスターの名前、またはローカルモードに local を使用		String

90.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
clientTransportSniff (producer)	クライアントがクラスターの残りをスニッフィングできるかどうか (デフォルトは true)。この設定は client.transport.sniff 設定にマップされます。	true	Boolean
consistencyLevel (producer)	INDEX および BULK 操作で使用する書き込み整合性レベル (ONE、QUORUM、ALL、または DEFAULT のいずれか)	DEFAULT	WriteConsistencyLevel
data (producer)	ノードにデータ (シャード) の割り当てを許可するかどうか。この設定は node.data 設定にマップされます。		Boolean
indexName (producer)	動作させるインデックスの名前。		String
indexType (producer)	作用するインデックスのタイプ		String
ip (producer)	使用する TransportClient リモートホスト IP		String
operation (producer)	実行する操作		String
pathHome (producer)	ElasticSearch 設定の path.home プロパティー。有効なパスを指定する必要があります。そうしないと、デフォルトの $user.home/.elasticsearch が使用されます。	${user.home}/.elasticsearch	String
port (producer)	使用する TransportClient リモートポート (デフォルトは 9300)	9300	int
transportAddresses (producer)	使用する ip:port 形式のリモートトランスポートアドレスを含むコンマ区切りのリスト。transportAddresses が代わりに考慮されるようにするには、ip オプションと port オプションを空白のままにする必要があります。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

90.3. ローカルテスト

ローカル (JVM/クラスローダー) の ElasticSearch サーバーに対して実行する場合は、URI の clusterName 値を local に設定するだけです。詳細については、クライアントガイド を参照してください。

90.4. メッセージ操作

現在、次の ElasticSearch 操作がサポートされています。操作のキーと次のいずれかに設定された値を使用して、エンドポイント URI オプションまたはエクスチェンジヘッダーを設定するだけです。一部の操作では、他のパラメーターまたはメッセージ本文を設定する必要もあります。

operation	メッセージボディー	description
INDEX	Map、String、byte、または XContentBuilder コンテンツをインデックスに登録する	コンテンツをインデックスに追加し、ボディーでコンテンツの indexId を返します。**Camel 2.15 では**、メッセージヘッダーにキー "indexId" を設定することで、indexId を設定できます。
GET_BY_ID	取得するコンテンツのインデックス ID	指定されたインデックスを取得し、本文で GetResult オブジェクトを返します
DELETE	削除するコンテンツのインデックス ID	指定された indexId を削除し、本文で DeleteResult オブジェクトを返します

operation	メッセージボディー	description
BULK_INDEX	すでに受け入れられている任意のタイプの **リスト** または **コレクション** (XContentBuilder、Map、byte、String)	*Camel 2.14、* コンテンツをインデックスに追加し、ボディー内のインデックスに成功したドキュメントの ID のリストを返します
BULK	すでに受け入れられている任意のタイプの **リスト** または **コレクション** (XContentBuilder、Map、byte、String)	**Camel 2.15:** コンテンツをインデックスに追加し、本文で BulkResponse オブジェクトを返します
SEARCH	Map または SearchRequest オブジェクト	**Camel 2.15:** クエリー文字列のマップでコンテンツを検索します。

operation	メッセージボディー	description
MULTIGET	MultigetRequest.Item オブジェクトのリスト	**Camel 2.17:** MultigetRequest で指定されたインデックス、タイプなどを取得し、本文で MultigetResponse オブジェクトを返します。
MULTISEARCH	SearchRequest オブジェクトのリスト	**Camel 2.17:** MultiSearchRequest で指定されたパラメーターを検索し、ボディーで MultiSearchResponse オブジェクトを返します。
EXISTS	ヘッダーとしてのインデックス名	**Camel 2.17:** 本文に Boolean オブジェクトを返します。
UPDATE	更新する Map、String、byte、または XContentBuilder コンテンツ	**Camel 2.17:** コンテンツをインデックスに更新し、本文でコンテンツの indexId を返します。

90.5. インデックスの例

以下は単純な INDEX の例です

```
from("direct:index")
.to("elasticsearch://local?operation=INDEX&indexName=twitter&indexType=tweet");
```

```
<route>
   <from uri="direct:index" />
   <to uri="elasticsearch://local?operation=INDEX&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、Map を含む本文メッセージをルートに渡すだけで済みます。結果の本文には、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

90.6. 詳細については、これらのリソースを参照してください

ElasticSearch メインサイト

ElasticSearch Java API

90.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第91章 ELASTICSEARCH5 コンポーネント (非推奨)

Camel バージョン 2.19 以降で利用可能

ElasticSearch コンポーネントを使用すると、ElasticSearch 5.x API と連携できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch5</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

91.1. URI 形式

```
elasticsearch5://clusterName[?options]
```

91.2. エンドポイントオプション

Elasticsearch5 コンポーネントは、以下にリストされている 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
client (advanced)	エンドポイントごとにクライアントを作成する代わりに、既存の設定済み Elasticsearch クライアントを使用する場合。これにより、特定の設定でクライアントをカスタマイズできます。		TransportClient
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elasticsearch5 エンドポイントは、URI 構文を使用して設定されます。

```
elasticsearch5:clusterName
```

パスおよびクエリーパラメーターを使用します。

91.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
clusterName	**必須** クラスターの名前		String

91.2.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
clientTransportSniff (producer)	クライアントがクラスターの残りをスニッフィングできるかどうか。この設定は client.transport.sniff 設定にマップされます。	false	boolean
indexName (producer)	動作させるインデックスの名前。		String
indexType (producer)	作用するインデックスのタイプ		String
ip (producer)	使用する TransportClient リモートホスト IP		String
operation (producer)	実行する操作		ElasticsearchOperation
pingSchedule (producer)	クライアントがクラスターに ping を実行した時間 (単位)。	5s	String
pingTimeout (producer)	ノードから返ってくる ping レスポンスを待つ時間 (単位)。	5s	String
port (producer)	使用する TransportClient リモートポート (デフォルトは 9300)	9300	int
tcpCompress (producer)	すべてのノード間で圧縮 (LZF) が有効な場合は true。	false	boolean
tcpConnectTimeout (producer)	接続タイムアウトを待機する時間 (単位)。	30s	String
transportAddresses (producer)	使用する ip:port 形式のリモートトランスポートアドレスを含むコンマ区切りのリスト。transportAddresses が代わりに考慮されるようにするには、ip オプションと port オプションを空白のままにする必要があります。		String
waitForActiveShards (producer)	インデックスの作成は、シャードの書き込み整合性数が使用可能になるまで待機します	1	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
enableSSL (security)	SSL の有効化。クラスパスに XPack クライアント jar を要求します	false	boolean

名前	説明	デフォルト	タイプ
password (authentication)	クラスターに対して認証するためのパスワード。クラスパスに XPack クライアント jar を要求します		String
user (authentication)	クラスターに対して認証するためのユーザー。クラスターにアクセスするには transport_client ロールが必要です。クラスパスに XPack クライアント jar を要求します		文字列

91.3. メッセージ操作

現在、次の ElasticSearch 操作がサポートされています。操作のキーと次のいずれかに設定された値を使用して、エンドポイント URI オプションまたはエクスチェンジヘッダーを設定するだけです。一部の操作では、他のパラメーターまたはメッセージ本文を設定する必要もあります。

operation	メッセージボディー	description
INDEX	Map、String、byte、または XContentBuilder コンテンツをインデックスに登録する	コンテンツをインデックスに追加し、本文でコンテンツの indexId を返します。メッセージヘッダーにキー "indexId" を設定することで、indexId を設定できます。
GET_BY_ID	取得するコンテンツのインデックス ID	指定されたインデックスを取得し、本文で GetResult オブジェクトを返します
DELETE	削除するコンテンツのインデックス名とタイプ	指定された indexName と indexType を削除し、ボディーで DeleteResponse オブジェクトを返します

operation	メッセージボディー	description
DELETE_INDEX	削除するコンテンツのインデックス名	指定された indexName を削除し、本文で DeleteIndexResponse オブジェクトを返します
BULK_INDEX	すでに受け入れられている任意のタイプの **リスト** または **コレクション** (XContentBuilder、Map、byte、String)	コンテンツをインデックスに追加し、本文で正常にインデックス付けされたドキュメントの ID のリストを返します
BULK	すでに受け入れられている任意のタイプの **リスト** または **コレクション** (XContentBuilder、Map、byte、String)	コンテンツをインデックスに追加し、本文で BulkResponse オブジェクトを返します

operation	メッセージボディー	description
SEARCH	マップ、文字列、または SearchRequest オブジェクト	クエリー文字列のマップでコンテンツを検索します
MULTIGET	MultigetRequest.Item オブジェクトのリスト	MultigetRequest で指定されたインデックス、タイプなどを取得し、ボディーで MultigetResponse オブジェクトを返します
MULTISEARCH	SearchRequest オブジェクトのリスト	MultiSearchRequest で指定されたパラメーターを検索し、ボディーで MultiSearchResponse オブジェクトを返します
EXISTS	ヘッダーとしてのインデックス名	インデックスが存在するかどうかを確認し、ボディーにブール値のフラグを返します
UPDATE	更新する Map、String、byte、または XContentBuilder コンテンツ	コンテンツをインデックスに更新し、本文でコンテンツの indexId を返します。

91.4. インデックスの例

以下は単純な INDEX の例です

```
from("direct:index")
.to("elasticsearch5://elasticsearch?operation=INDEX&indexName=twitter&indexType=tweet");
```

```
<route>
  <from uri="direct:index" />
  <to uri="elasticsearch5://elasticsearch?operation=INDEX&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、Map を含む本文メッセージをルートに渡すだけで済みます。結果の本文には、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

91.5. 詳細については、これらのリソースを参照してください

Elastic メインサイト

ElasticSearch Java API

91.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第92章 ELASTICHSEARCH REST コンポーネント

Camel バージョン 2.21 以降で利用可能

ElasticSearch コンポーネントを使用すると、REST クライアントライブラリーを使用して ElasticSearch 6.x API とやり取りできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch-rest</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

92.1. URI 形式

```
elasticsearch-rest://clusterName[?options]
```

92.2. エンドポイントオプション

Elastichsearch Rest コンポーネントは、以下にリストされている 12 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
client (advanced)	エンドポイントごとにクライアントを作成する代わりに、既存の設定済み Elasticsearch クライアントを使用する場合。これにより、特定の設定でクライアントをカスタマイズできます。		RestClient
hostAddresses (advanced)	使用する ip:port 形式のリモートトランスポートアドレスを含むコンマ区切りのリスト。代わりに hostAddresses が考慮されるようにするには、ip オプションと port オプションを空白のままにする必要があります。		String
socketTimeout (advanced)	ソケットがタイムアウトする前に待機するミリ秒単位のタイムアウト。	30000	int
connectionTimeout (advanced)	接続がタイムアウトするまでのミリ秒単位の待機時間。	30000	int
user (advance)	基本認証ユーザー		文字列
password (producer)	認証用パスワード		文字列

名前	説明	デフォルト	タイプ
enableSSL (advanced)	SSL の有効化	false	Boolean
maxRetryTimeout (advanced)	再試行までの時間 (ミリ秒)	30000	int
enableSniffer (advanced)	実行中の Elasticsearch クラスターからのノードの自動検出を有効にする	false	Boolean
snifferInterval (advanced)	通常のスニファを連続して実行する間隔 (ミリ秒単位)。sniffOnFailure が無効になっている場合、または連続するスニファ実行の間に失敗がない場合に受け入れられます	300000	int
sniffAfterFailureDelay (advanced)	失敗後にスケジュールされたスニファ実行の遅延 (ミリ秒単位)	60000	int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elastichsearch Rest エンドポイントは、URI 構文を使用して設定されます。

```
elasticsearch-rest:clusterName
```

パスおよびクエリーパラメーターを使用します。

92.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
clusterName	**必須** クラスターの名前		String

92.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
connectionTimeout (producer)	接続がタイムアウトするまでのミリ秒単位の待機時間。	30000	int

名前	説明	デフォルト	タイプ
disconnect (producer)	プロデューサーの呼び出しが終了したら切断します	false	boolean
enableSSL (producer)	SSL の有効化	false	boolean
hostAddresses (producer)	**必須** 使用する ip:port 形式のリモートトランスポートアドレスを含むコンマ区切りのリスト。代わりに hostAddresses が考慮されるようにするには、ip オプションと port オプションを空白のままにする必要があります。		文字列
indexName (producer)	動作させるインデックスの名前。		String
indexType (producer)	作用するインデックスのタイプ		String
maxRetryTimeout (producer)	再試行までの時間 (ミリ秒)	30000	int
operation (producer)	実行する操作		ElasticsearchOperation
socketTimeout (producer)	ソケットがタイムアウトする前に待機するミリ秒単位のタイムアウト。	30000	int
waitForActiveShards (producer)	インデックスの作成は、シャードの書き込み整合性数が使用可能になるまで待機します	1	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

92.3. メッセージ操作

現在、次の ElasticSearch 操作がサポートされています。操作のキーと次のいずれかに設定された値を使用して、エンドポイント URI オプションまたはエクスチェンジヘッダーを設定するだけです。一部の操作では、他のパラメーターまたはメッセージ本文を設定する必要もあります。

operation	メッセージボディー	description
Index	**Map**、**String**、**byte**、**XContentBuilder** または **IndexRequest** コンテンツをインデックスに登録する	コンテンツをインデックスに追加し、本文でコンテンツの indexId を返します。メッセージヘッダーにキー "indexId" を設定することで、indexId を設定できます。
GetById	取得するコンテンツの **文字列** または **GetRequest** インデックス ID	指定されたインデックスを取得し、本文で GetResult オブジェクトを返します
Delete	削除するコンテンツの **文字列** または **DeleteRequest** インデックス名とタイプ	指定された indexName と indexType を削除し、ボディーで DeleteResponse オブジェクトを返します

operation	メッセージボディー	description
DeleteIndex	削除するインデックスの **文字列** または **DeleteRequest** インデックス名	指定された indexName を削除し、ボディーのステータスコードを返します。
BulkIndex	すでに受け入れられている任意のタイプの **List**、**BulkRequest**、または **Collection** (XContentBuilder、Map、byte、String)	コンテンツをインデックスに追加し、本文で正常にインデックス付けされたドキュメントの ID のリストを返します

operation	メッセージボディー	description
バルク	すでに受け入れられている任意のタイプの **List**、**BulkRequest**、または **Collection** (XContentBuilder、Map、byte、String)	コンテンツをインデックスに追加し、本文で BulkItemResponse オブジェクトを返します
Search	**Map**、**String** または **SearchRequest**	クエリー文字列のマップでコンテンツを検索します
Exists	ヘッダーとしてのインデックス名 (indexName)	インデックスが存在するかどうかを確認し、ボディーにブール値のフラグを返します

operation	メッセージボディー	description
Update	更新する Map、UpdateRequest、String、byte[] または XContentBuilder コンテンツ	コンテンツをインデックスに更新し、本文でコンテンツの indexId を返します。
Ping	なし	リモート Elasticsearch クラスターに ping を実行し、ping が成功した場合は true、それ以外の場合は false を返します。

92.4. コンポーネントを設定して基本認証を有効にする

Elasticsearch コンポーネントを使用するには、最小構成で設定する必要があります。

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
camelContext.addComponent("elasticsearch-rest", elasticsearchComponent);
```

Elasticsearch を使用した基本認証、または Elasticsearch クラスターの前でリバース HTTP プロキシーを使用する場合は、以下の例のようにコンポーネントで基本認証と SSL をセットアップするだけです。

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
elasticsearchComponent.setUser("elkuser");
elasticsearchComponent.setPassword("secure!!");
elasticsearchComponent.setEnableSSL(true);

camelContext.addComponent("elasticsearch-rest", elasticsearchComponent);
```

92.5. インデックスの例

以下は単純な INDEX の例です

```
from("direct:index")
  .to("elasticsearch-rest://elasticsearch?operation=Index&indexName=twitter&indexType=tweet");
```

```
<route>
    <from uri="direct:index" />
    <to uri="elasticsearch-rest://elasticsearch?
operation=Index&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、Map を含む本文メッセージをルートに渡すだけで済みます。結果の本文には、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

92.6. 検索例

特定のフィールドと値を検索するには、検索操作を使用します。クエリーの JSON 文字列またはマップを渡します

```
from("direct:search")
  .to("elasticsearch-rest://elasticsearch?operation=Search&indexName=twitter&indexType=tweet");
```

```
<route>
    <from uri="direct:search" />
    <to uri="eelasticsearch-rest://elasticsearch?
operation=Search&indexName=twitter&indexType=tweet"/>
</route>
```

```
String query = "{\"query\":{\"match\":{\"content\":\"new release of ApacheCamel\"}}}";
SearchHits response = template.requestBody("direct:search", query, SearchHits.class);
```

マップを使用して特定のフィールドを検索します。

```
Map<String, Object> actualQuery = new HashMap<>();
actualQuery.put("content", "new release of ApacheCamel");

Map<String, Object> match = new HashMap<>();
match.put("match", actualQuery);

Map<String, Object> query = new HashMap<>();
query.put("query", match);
SearchHits response = template.requestBody("direct:search", query, SearchHits.class);
```

第93章 ELSQL コンポーネント

Camel バージョン 2.16 以降で利用可能

elsql: コンポーネントは、ElSql を使用して SQL クエリーを定義する既存の SQL コンポーネント の拡張です。

このコンポーネントは、実際の SQL 処理のために舞台裏で **spring-jdbc** を使用します。

このコンポーネントは、Transactional Client として使用できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elsql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

SQL コンポーネントは、次のエンドポイント URI 表記を使用します。

```
sql:elSqlName:resourceUri[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

SQL クエリーへのパラメーターは、elsql マッピングファイル内の名前付きパラメーターであり、指定された優先順位で Camel メッセージから対応するキーにマップされます。

1. **Camel 2.16.1:** Simple 式の場合はメッセージボディーから。
2. `java.util.Map` の場合はメッセージ本文から 3.メッセージヘッダーから

名前付きパラメーターを解決できない場合は、例外が出力されます。

93.1. オプション

ElSQL コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
databaseVendor (Common)	ベンダー固有の com.opengamma.elsql.ElSqlConfig を使用する場合		ElSqlDatabaseVendor
dataSource (Common)	データベースとの通信に使用する DataSource を設定します。		DataSource
elSqlConfig (advanced)	特定の設定済み ElSqlConfig を使用する場合。代わりに、databaseVendor オプションを使用することをお勧めします。		ElSqlConfig

名前	説明	デフォルト	タイプ
resourceUri (common)	使用する elsql SQL ステートメントを含むリソースファイル。複数のリソースをコンマで区切って指定できます。リソースはデフォルトでクラスパスにロードされます。file: を前に付けて、ファイルシステムからロードできます。このオプションはコンポーネントで設定でき、エンドポイントでこれを設定する必要がないことに注意してください。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

EISQL エンドポイントは、URI 構文を使用して設定されます。

```
elsql:elsqlName:resourceUri
```

パスおよびクエリーパラメーターを使用します。

93.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
elsqlName	**必須** 使用する elsql の名前 (elsql ファイルでは NAMED です)		String
resourceUri	使用する elsql SQL ステートメントを含むリソースファイル。複数のリソースをコンマで区切って指定できます。リソースはデフォルトでクラスパスにロードされます。file: を前に付けて、ファイルシステムからロードできます。このオプションはコンポーネントで設定でき、エンドポイントでこれを設定する必要がないことに注意してください。		String

93.1.2. クエリーパラメーター(47 個のパラメーター):

名前	説明	デフォルト	タイプ
allowNamedParameters (common)	クエリーで名前付きパラメーターの使用を許可するかどうか。	true	boolean

名前	説明	デフォルト	タイプ
databaseVendor (Common)	ベンダー固有の com.opengamma.elsql.ElSqlConfig を使用する場合		ElSqlDatabaseVendor
dataSource (Common)	データベースとの通信に使用する DataSource を設定します。		DataSource
dataSourceRef (common)	**非推奨** データベースとの通信に使用するために、レジストリーから参照する DataSource への参照を設定します。		String
outputClass (common)	outputType=SelectOne の場合、変換として使用する完全なパッケージとクラス名を指定します。		String
outputHeader (common)	メッセージ本文ではなく、ヘッダーにクエリー結果を格納します。デフォルトでは、outputHeader == null で、クエリー結果はメッセージ本文に格納され、メッセージ本文の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を格納するヘッダーの名前として使用され、元のメッセージ本文は保持されます。		String
outputType (common)	以下の方法で、コンシューマーまたはプロデューサーの出力を SelectList にマップのリストとして、または SelectOne を単一の Java オブジェクトとして作成します。a) クエリーに単一の列しかない場合、その JDBC 列オブジェクトが返されます。(SELECT COUNT() FROM PROJECT などは Long オブジェクトを返します。b) クエリーに複数の列がある場合、その結果のマップを返します。c) outputClass が設定されている場合、クエリーを変換します。列名に一致するすべてのセッターを呼び出すことにより、Java Bean オブジェクトになります。クラスには、インスタンスを作成するためのデフォルトのコンストラクターがあると想定されます。d) クエリーの結果が複数の行になった場合、一意でない結果が出力されます。exception.StreamList は、Iterator を使用してクエリーの結果をストリーミングします。これは、ストリーミング方式で ResultSet を処理するために、ストリーミングモードでスプリッタ EIP とともに使用できます。	SelectList	SqlOutputType
separator (common)	パラメーター値がメッセージ本文から取得されるときに使用するセパレーター (ボディーが文字列型の場合)、プレースホルダーに挿入されます。名前付きパラメーターを使用する場合は、代わりに Map 型が使用されることに注意してください。デフォルト値は、コンマです。	,	char

名前	説明	デフォルト	タイプ
breakBatchOnConsumeFail (consumer)	onConsume が失敗した場合にバッチを中断するかどうかを設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
expectedUpdateCount (consumer)	onConsume を使用するときに検証する予想更新カウントを設定します。	-1	int
maxMessagesPerPoll (consumer)	ポーリングするメッセージの最大数を設定します		int
onConsume (consumer)	各行を処理した後、エクスチェンジが正常に処理された場合、たとえば行を処理済みとしてマークするために、このクエリーを実行できます。クエリーにはパラメーターを含めることができます。		String
onConsumeBatchComplete (consumer)	バッチ全体を処理した後、このクエリーを実行して行などを一括更新できます。クエリーにパラメーターを含めることはできません。		String
onConsumeFailed (consumer)	各行を処理した後、エクスチェンジが失敗した場合、たとえば、行を失敗としてマークするために、このクエリーを実行できます。クエリーにはパラメーターを含めることができます。		String
routeEmptyResultSet (consumer)	空の結果セットを次のホップに送信できるようにするかどうかを設定します。デフォルトは false です。したがって、空の結果セットは除外されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
transacted (consumer)	トランザクションを有効または無効にします。有効にすると、交換の処理が失敗した場合に、コンシューマーはそれ以上の交換の処理を中断して、先行してロールバックを実行させます。	false	boolean

名前	説明	デフォルト	タイプ
useIterator (consumer)	結果セットをルートに配信する方法を設定します。リストまたは個々のオブジェクトとして配信を示します。デフォルトは真です。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processingStrategy (consumer)	コンシューマーが行/バッチを処理したときに、カスタム org.apache.camel.component.sql.SqlProcessingStrategy を使用してクエリーを実行するプラグインを許可します。		SqlProcessingStrategy
batch (producer)	バッチモードを有効または無効にします	false	boolean
noop (producer)	設定されている場合、SQL クエリーの結果を無視し、既存の IN メッセージを OUT メッセージとして使用して処理を続行します。	false	boolean
useMessageBodyForSql (producer)	メッセージ本文を SQL として使用し、次にパラメーターのヘッダーを使用するかどうか。このオプションを有効にすると、URI の SQL は使用されません。	false	boolean
alwaysPopulateStatement (producer)	有効にすると、org.apache.camel.component.sql.SqlPrepareStatementStrategy の populateStatement メソッドが常に呼び出されます。また、準備する必要のあるパラメーターがない場合も同様です。これが false の場合、populateStatement は、1 つ以上の予期されるパラメーターが設定される場合にのみ呼び出されます。たとえば、これにより、パラメーターのない SQL クエリーのメッセージ本文/ヘッダーの読み取りが回避されます。	false	boolean

名前	説明	デフォルト	タイプ
parametersCount (プロデューサー)	0 より大きい値を設定すると、Camel は JDBC メタデータ API を介してクエリーを実行する代わりに、このパラメーターのカウント値を使用して置き換えます。これは、JDBC ベンダーが正しいパラメーター数を返すことができず、ユーザーが代わりにオーバーライドできる場合に役立ちます。		int
elSqlConfig (advanced)	特定の設定済み ElSqlConfig を使用する場合。代わりに、databaseVendor オプションを使用することをお勧めします。		ElSqlConfig
placeholder (advanced)	SQL クエリーで置換される文字を指定します。これは単純な String.replaceAll() 操作であり、SQL 解析が含まれていないことに注意してください (引用符で囲まれた文字列も変更されます)。	#	String
prepareStatementStrategy (advanced)	プラグインでカスタム org.apache.camel.component.sql.SqlPrepareStatementStrategy を使用して、クエリーと準備済みステートメントの準備を制御できるようにします。		SqlPrepareStatement Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
templateOptions (advanced)	マップからのキー/値を使用して Spring JdbcTemplate を設定します		Map
usePlaceholder (advanced)	SQL クエリーでプレースホルダーを使用し、すべてのプレースホルダー文字を符号に置き換えるかどうかを設定します。	true	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

名前	説明	デフォルト	タイプ
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

93.2. クエリーの結果

select 操作の場合、結果は JdbcTemplate.queryForList() メソッドによって返される **List<Map<String, Object>>** タイプのインスタンスです。**更新** 操作の場合、結果は更新された行の数であり、**Integer** として返されます。

デフォルトでは、結果はメッセージボディーに配置されます。 outputHeader パラメーターが設定されている場合、結果はヘッダーに配置されます。 これは、完全なメッセージエンリッチメントパターンを使用してヘッダーを追加する代わりの方法であり、シーケンスまたはその他の小さな値をヘッダーにクエリーするための簡潔な構文を提供します。 outputHeader と outputType を一緒に使用すると便利です。

93.3. ヘッダーの値

update 操作を実行すると、SQL コンポーネントは更新カウントを次のメッセージヘッダーに格納します。

ヘッダー	説明
CamelSqlUpdateCount	**update** 操作によって更新された行の数を **Integer** オブジェクトとして返します。
CamelSqlRowCount	**select** 操作によって返される行の数を **Integer** オブジェクトで返します。

93.3.1. 例

以下の特定のルートでは、projects テーブルからすべてのプロジェクトを取得します。SQL クエリーには、:#lic と :#min という 2 つの名前付きパラメーターがあることに注意してください。

Camel は、メッセージ本文またはメッセージヘッダーからこれらのパラメーターを検索します。上記の例では、2 つのヘッダーに定数値を設定していることに注意してください。
名前付きパラメーターの場合:

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
  .setHeader("min", constant(123))
  .to("elsql:projects:com/foo/orders.elsql")
```

そして elsql マッピングファイル

```
@NAME(projects)
  SELECT *
  FROM projects
  WHERE license = :lic AND id > :min
  ORDER BY id
```

ただし、メッセージボディーが **java.util.Map** の場合、名前付きパラメーターは本文から取得されます。

```
from("direct:projects")
  .to("elsql:projects:com/foo/orders.elsql")
```

Camel 2.16.1 以降では、Simple 式も使用できます。これにより、メッセージボディーで OGNL のような表記を使用できます。ここでは、**getLicense** および **getMinimum** メソッドがあることを前提としています。

```
@NAME(projects)
  SELECT *
  FROM projects
  WHERE license = :${body.license} AND id > :${body.minimum}
  ORDER BY id
```

93.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- SQL コンポーネント
- MyBatis
- JDBC

第94章 ETCD コンポーネント

Camel バージョン 2.18 以降で利用可能

camel etcd コンポーネントを使用すると、分散型の信頼できるキー値ストアである Etcd を操作できます。

94.1. URI 形式

```
etcd:namespace/path[?options]
```

94.2. URI オプション

etcd コンポーネントは、次に示す 7 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
uris (Common)	クライアントが接続する URI を設定します。		String
sslContextParameters (common)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
userName (common)	基本認証に使用するユーザー名。		String
password (common)	基本認証に使用するパスワード。		String
configuration (advanced)	エンドポイント間で共有される共通設定を設定します		EtcdConfiguration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

etcd エンドポイントは、URI 構文を使用して設定されます。

```
etcd:namespace/path
```

パスおよびクエリーパラメーターを使用します。

94.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
namespace	**必須** 使用する API 名前空間		EtcdNamespace
path	エンドポイントが参照するパス		String

94.2.2. クエリーパラメーター (29 個のパラメーター):

名前	説明	デフォルト	タイプ
recursive (Common)	アクションを再帰的に適用すること。	false	boolean
servicePath (Common)	サービス検出のために探すパス	/services/	String
timeout (common)	アクションが完了するまでにかかる最大時間を設定します。		Long
uris (Common)	クライアントが接続する URI を設定します。	http://localhost:2379,http://localhost:4001	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyExchangeOnTimeout (consumer)	キーの監視がタイムアウトした場合に空のメッセージを送信します。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
fromIndex (consumer)	監視するインデックス	0	Long
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
timeToLive (producer)	キーの寿命をミリ秒単位で設定します。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long

名前	説明	デフォルト	タイプ
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
password (security)	基本認証に使用するパスワード。		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
userName (security)	基本認証に使用するユーザー名。		String

第95章 OSGI EVENTADMIN コンポーネント

Camel バージョン 2.6 以降で利用可能

OSGi 環境で **eventadmin** コンポーネントを使用して、OSGi EventAdmin イベントを受け取り、処理することができます。

95.1. 依存関係

Maven ユーザーは以下の依存関係を **pom.xml** に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-eventadmin</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.6.0 以降) に置き換える必要があります。

95.2. URI 形式

```
eventadmin:topic[?options]
```

ここで、**topic** もリッスンするトピックの名前です。

95.3. URI オプション

OSGi EventAdmin コンポーネントは、以下にリストされている 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
bundleContext (Common)	OSGi BundleContext は Camel によって自動的に注入されます		BundleContext
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OSGi EventAdmin エンドポイントは、URI 構文を使用して設定されます。

```
eventadmin:topic
```

パスおよびクエリーパラメーターを使用します。

95.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
topic	リッスンまたは送信するトピックの名前		String

95.3.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
send (Common)	送信または同期配信のどちらを使用するか。デフォルト false (非同期配信)	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

95.4. メッセージヘッダー

名前	タイプ	メッセージ
説明		

95.5. メッセージボディー

in メッセージボディーは、受信したイベントに設定されます。

95.6. 使用例

```
<route>
    <from uri="eventadmin:*"/>
    <to uri="stream:out"/>
</route>
```

第96章 EXEC コンポーネント

Camel バージョン 2.3 以降で利用可能

exec コンポーネントを使用して、システムコマンドを実行できます。

96.1. 依存関係

Maven ユーザーは以下の依存関係を **pom.xml** に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-exec</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.3.0 以降) に置き換える必要があります。

96.2. URI 形式

```
exec://executable[?options]
```

ここで、**executable** は、実行されるシステムコマンドの名前またはファイルパスです。実行可能ファイル名を使用する場合 (例: **exec:java**)、実行可能ファイルはシステムパスにある必要があります。

96.3. URI オプション

Exec コンポーネントにはオプションがありません。

Exec エンドポイントは、URI 構文を使用して設定されます。

```
exec:executable
```

パスおよびクエリーパラメーターを使用します。

96.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
executable	**必須** 実行する実行ファイルを設定します。実行可能ファイルを空または null にすることはできません。		String

96.3.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
args (producer)	引数は、空白で区切られた 1 つまたは複数のトークンです。		String
binding (producer)	レジストリー内の org.apache.commons.exec.ExecBinding への参照。		ExecBinding
commandExecutor (producer)	コマンドの実行をカスタマイズするレジストリー内の org.apache.commons.exec.ExecCommandExecutor への参照。デフォルトのコマンドエグゼキューターは commons-exec ライブラリーを利用し、実行されたすべてのコマンドにシャットダウンフックを追加します。		ExecCommandExecutor
outFile (producer)	実行可能ファイルによって作成され、その出力と見なされるファイルの名前。outFile が設定されていない場合は、代わりに実行可能ファイルの標準出力 (stdout) が使用されます。		String
timeout (producer)	実行可能ファイルを終了するまでのミリ秒単位のタイムアウト。タイムアウト内に実行が完了しなかった場合、コンポーネントは終了要求を送信します。		long
useStderrOnEmptyStdout (producer)	stdout が空の場合、このコンポーネントが Camel メッセージボディーに stderr を入力することを示すブール値。この動作はデフォルトで無効 (false) です。	false	boolean
workingDir (producer)	コマンドを実行するディレクトリー。null の場合、現在のプロセスの作業ディレクトリーが使用されます。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

96.4. メッセージヘッダー

サポートされているヘッダーは **org.apache.camel.component.exec.ExecBinding** で定義されています。

名前	タイプ	メッセージ	説明
ExecBinding.EXEC_COMMAND_EXECUTABLE	**String**	**in**	実行されるシステムコマンドの名前。URI の **executable** をオーバーライドします。
ExecBinding.EXEC_COMMAND_ARGS	**java.util.List<String>**	**in**	実行されたプロセスに渡すコマンドライン引数。引数は文字どおりに使用されます - 引用符は適用されません。URI 内の既存の **args** をオーバーライドします。
ExecBinding.EXEC_COMMAND_ARGS	**String**	**in**	Camel 2.5: 各引数が空白で区切られた単一の文字列としての実行可能ファイルの引数 (URI オプションの **args** を参照)。引数は文字どおりに使用され、引用符は適用されません。URI 内の既存の **args** をオーバーライドします。
ExecBinding.EXEC_COMMAND_OUT_FILE	**String**	**in**	実行可能ファイルによって作成され、その出力と見なされるファイルの名前。URI 内の既存の **outFile** をオーバーライドします。
ExecBinding.EXEC_COMMAND_TIMEOUT	**long**	**in**	実行可能ファイルを終了するまでのミリ秒単位のタイムアウト。URI の既存の **timeout** をオーバーライドします。
ExecBinding.EXEC_COMMAND_WORKING_DIR	**String**	**in**	コマンドを実行するディレクトリー。URI 内の既存の **workingDir** をオーバーライドします。

名前	タイプ	メッセージ	説明
ExecBinding.EXEC_EXIT_VALUE	**int**	**out**	このヘッダーの値は、実行可能ファイルの **終了値** です。ゼロ以外の終了値は通常、異常終了を示します。終了値は OS に依存することに注意してください。
ExecBinding.EXEC_STDERR	**java.io.InputStream**	**out**	このヘッダーの値は、実行可能ファイルの標準エラーストリーム (stderr) を指します。stderr が書き込まれない場合、値は **null** です。
ExecBinding.EXEC_USE_STDERR_ON_EMPTY_STDOUT	**boolean**	**in**	**stdout** が空の場合、このコンポーネントが Camel メッセージボディーに **stderr** を設定することを示します。この動作はデフォルトで無効 (**false**) です。

96.5. メッセージボディー

Exec コンポーネントが **java.io.InputStream** に変換可能な **in** メッセージボディーを受信した場合、stdin を介して実行可能ファイルに入力を供給するために使用されます。実行後、メッセージボディーは実行の結果、つまり、stdout、stderr、終了値、および出力ファイルを含む **org.apache.camel.components.exec.ExecResult** インスタンスです。このコンポーネントは、便宜上、次の **ExecResult**型コンバーター をサポートしています。

From	終了
ExecResult	**java.io.InputStream**
ExecResult	**String**
ExecResult	**byte []**
ExecResult	**org.w3c.dom.Document**

out ファイルが (**outFile** を介してエンドポイントで、または **ExecBinding.EXEC_COMMAND_OUT_FILE** を介してメッセージヘッダーで) 指定されている場合、コンバーターは out ファイルの内容を返します。out ファイルが使用されていない場合、このコンポーネ

ントはプロセスの stdout をターゲットタイプに変換します。詳細については、以下の 使用例 を参照してください。

96.6. 使用例

96.6.1. ワードカウントの実行 (Linux)

以下の例では、**wc** (単語カウント、Linux) を実行して、ファイル **/usr/share/dict/words** 内の単語をカウントします。単語数 (出力) は、**wc** の標準出力ストリームに書き込まれます。

```
from("direct:exec")
.to("exec:wc?args=--words /usr/share/dict/words")
.process(new Processor() {
    public void process(Exchange exchange) throws Exception {
      // By default, the body is ExecResult instance
      assertIsInstanceOf(ExecResult.class, exchange.getIn().getBody());
      // Use the Camel Exec String type converter to convert the ExecResult to String
      // In this case, the stdout is considered as output
      String wordCountOutput = exchange.getIn().getBody(String.class);
      // do something with the word count
    }
});
```

96.6.2. java の実行

次の例では、**java** がシステムパスにある場合、**-server** と **-version** の 2 つの引数を指定して **java** を実行します。

```
from("direct:exec")
.to("exec:java?args=-server -version")
```

以下の例では、3 つの引数 **-server**、**-version**、およびシステムプロパティー **user.name** を使用して **c:\temp** で **java** を実行します。

```
from("direct:exec")
.to("exec:c:/program files/jdk/bin/java?args=-server -version -
Duser.name=Camel&workingDir=c:/temp")
```

96.6.3. Ant スクリプトの実行

次の例では、**ant.bat** がシステムパスにあり、**CamelExecBuildFile.xml** が現在のディレクトリーにある場合に、ビルドファイル **CamelExecBuildFile.xml** を使用して Apache Ant (Windows のみ) を実行します。

```
from("direct:exec")
.to("exec:ant.bat?args=-f CamelExecBuildFile.xml")
```

次の例では、**ant.bat** コマンドは **-l** を使用してその出力を **CamelExecOutFile.txt** にリダイレクトします。ファイル **CamelExecOutFile.txt** は **outFile=CamelExecOutFile.txt** で出力ファイルとして使用されます。この例では、**ant.bat** がシステムパスにあり、**CamelExecBuildFile.xml** が現在のディレクトリーにあると想定しています。

```
from("direct:exec")
.to("exec:ant.bat?args=-f CamelExecBuildFile.xml -l
CamelExecOutFile.txt&outFile=CamelExecOutFile.txt")
.process(new Processor() {
    public void process(Exchange exchange) throws Exception {
       InputStream outFile = exchange.getIn().getBody(InputStream.class);
       assertIsInstanceOf(InputStream.class, outFile);
       // do something with the out file here
    }
 });
```

96.6.4. echo の実行 (Windows)

echo や **dir** などのコマンドは、オペレーティングシステムのコマンドインタープリターでのみ実行できます。この例は、Windows でこのようなコマンド (**echo**) を実行する方法を示しています。

```
from("direct:exec").to("exec:cmd?args=/C echo echoString")
```

96.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第97章 FACEBOOK コンポーネント

Camel バージョン 2.14 以降で利用可能

Facebook コンポーネントは、Facebook4J を使用してアクセス可能なすべての Facebook API へのアクセスを提供します。メッセージを生成して、投稿、いいね、コメント、写真、アルバム、ビデオ、写真、チェックイン、ロケーション、リンクなどを取得、追加、および削除できます。また、投稿、ユーザー、チェックイン、グループ、ロケーションなどのポーリングを可能にする API もサポートしています。

Facebook では、すべてのクライアントアプリケーション認証に OAuth を使用する必要があります。アカウントで camel-facebook を使用するには、https://developers.facebook.com/apps で Facebook 内に新しいアプリケーションを作成し、アプリケーションにアカウントへのアクセスを許可する必要があります。Facebook アプリケーションの ID とシークレットは、現在のユーザーを必要としない Facebook API へのアクセスを許可します。ログインユーザーが必要な API には、ユーザーアクセストークンが必要です。ユーザーアクセストークンの取得の詳細については、https://developers.facebook.com/docs/facebook-login/access-tokens/ を参照してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-facebook</artifactId>
  <version>${camel-version}</version>
</dependency>
```

97.1. URI 形式

```
facebook://[endpoint]?[options]
```

97.2. FACEBOOKCOMPONENT

Facebook コンポーネントは、以下の必須の Facebook アカウント設定で構成できます。値は、タイプ **org.apache.camel.component.facebook.config.FacebookConfiguration** の Bean プロパティー **設定** を使用してコンポーネントに提供できます。**oAuthAccessToken** オプションは省略できますが、アプリケーション API へのアクセスのみが許可されます。

Facebook コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有設定を使用するには		FacebookConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Facebook エンドポイントは、URI 構文を使用して設定されます。

```
facebook:methodName
```

パスおよびクエリーパラメーターを使用します。

97.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
methodName	**必須** 実行する操作		String

97.2.2. クエリーパラメーター(102 個のパラメーター):

名前	説明	デフォルト	タイプ
achievementURL (common)	実績の一意の URL		URL
albumId (common)	アルバム ID		String
albumUpdate (common)	作成または更新する Facebook アルバム		AlbumUpdate
appId (common)	Facebook アプリケーションの ID		String
center (common)	ロケーションの緯度と経度		GeoLocation
checkinId (common)	チェックイン ID		String
checkinUpdate (common)	**非推奨** 作成されるチェックイン。非推奨、代わりにロケーションを添付して投稿を作成		CheckinUpdate
clientURL (common)	Facebook4J API クライアント URL		String
clientVersion (common)	Facebook4J クライアント API バージョン		String
commentId (common)	コメント ID		String
commentUpdate (common)	作成または更新する facebook コメント		CommentUpdate

名前	説明	デフォルト	タイプ
debugEnabled (common)	deubg 出力を有効にします。組み込みロガーでのみ有効	false	Boolean
description (common)	説明テキスト		String
distance (common)	メートル単位の距離		Integer
domainId (common)	ドメイン ID		String
domainName (common)	ドメイン名		String
domainNames (common)	ドメイン名		List
eventId (common)	イベント ID		String
eventUpdate (common)	作成または更新するイベント		EventUpdate
friendId (common)	フレンド ID		String
friendlistId (common)	フレンドリスト ID		String
friendlistName (common)	フレンドリストの名前		String
friendUserId (common)	友達のユーザー ID		String
groupId (common)	グループ ID		String
gzipEnabled (common)	Facebook GZIP エンコーディングを使用する	true	Boolean
httpConnectionTimeout (common)	ミリ秒単位の Http 接続タイムアウト。	20000	Integer

名前	説明	デフォルト	タイプ
httpDefaultMaxPerRoute (common)	ルートあたりの HTTP 最大接続数	2	Integer
httpMaxTotalConnections (common)	HTTP の最大合計接続数	20	Integer
httpReadTimeout (common)	HTTP 読み取りタイムアウト (ミリ秒)	120000	Integer
httpRetryCount (common)	HTTP 再試行回数	0	Integer
httpRetryIntervalSeconds (common)	HTTP 再試行間隔 (秒)	5	Integer
httpStreamingReadTimeout (common)	HTTP ストリーミング読み取りタイムアウト (ミリ秒)	40000	Integer
ids (common)	ユーザーの ID		List
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
includeRead (common)	未読の通知に加えて、ユーザーがすでに読んだ通知を有効にします		Boolean
isHidden (common)	hidden かどうか		Boolean
jsonStoreEnabled (common)	true に設定すると、raw JSON フォームが DataObjectFactory に格納されます	false	Boolean
link (common)	リンク URL		URL
linkId (common)	リンク ID		String
locale (common)	希望する FQL ロケール		Locale
mbeanEnabled (common)	true に設定すると、Facebook4J mbean が登録されます	false	Boolean

名前	説明	デフォルト	タイプ
message (common)	メッセージテキスト		String
messageId (common)	メッセージ ID		String
metric (common)	メトリクス名		String
milestoneId (common)	マイルストーン ID		String
name (common)	テストユーザー名。名姓の形式にする必要があります		String
noteId (common)	ノート ID		String
notificationId (common)	通知 ID		String
objectId (common)	インサイトオブジェクト ID		String
offerId (common)	オファー ID		String
optionDescription (common)	質問の回答オプションの説明		String
pageId (common)	ページ ID		String
permissionName (common)	パーミッション名		String
permissions (common)	perm1、perm2、... の形式でユーザー権限をテストします。		String
photoId (common)	写真付き身分証明書		String
pictureId (common)	写真 ID		Integer
pictureId2 (common)	写真 2 ID		Integer
pictureSize (common)	写真サイズ		PictureSize

名前	説明	デフォルト	タイプ
placeId (common)	場所 ID		String
postId (common)	投稿 ID		String
postUpdate (common)	作成または更新する投稿		PostUpdate
prettyDebugEnabled (common)	true に設定されている場合、JSON デバッグ出力を整形します	false	Boolean
queries (common)	FQL クエリー		Map
query (common)	検索エンドポイントの FQL クエリーまたは検索語		String
questionId (common)	質問 ID		String
reading (common)	オプションの読み取りパラメーター。読み取りオプション (reading) を参照してください。		読み取り
readingOptions (common)	マップからのキーと値のペアを使用して読み取りを設定する場合。		Map
restBaseURL (common)	API ベース URL	https://graph.facebook.com/	String
scoreValue (common)	値を持つ数値スコア		Integer
size (common)	写真のサイズ (large,、normal、small、square のいずれか)		PictureSize
source (common)	java.io.File または java.io.Inputstream のメディアコンテンツ		メディア
subject (common)	Subject のメモ		String
tabId (common)	タブ ID		String
tagUpdate (common)	写真タグ情報		TagUpdate

名前	説明	デフォルト	タイプ
testUser1 (common)	テストユーザー 1		TestUser
testUser2 (common)	テストユーザー 2		TestUser
testUserId (common)	テストユーザーの ID		String
title (common)	タイトルテキスト		String
toUserId (common)	タグ付けするユーザーの ID		String
toUserIds (common)	タグ付けするユーザーの ID		List
userId (common)	Facebook のユーザー ID		String
userId1 (common)	ユーザー 1 の ID		String
userId2 (common)	ユーザー 2 の ID		String
userIds (common)	イベントに招待するユーザーの ID		List
userLocale (common)	テストユーザーロケール		String
useSSL (common)	SSL の使用	true	Boolean
videoBaseURL (common)	ビデオ API ベース URL	https://graph-video.facebook.com/	String
videoId (common)	ビデオ ID		String

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
httpProxyHost (proxy)	HTTP プロキシーサーバーのホスト名		String
httpProxyPassword (proxy)	HTTP プロキシーサーバーのパスワード		String
httpProxyPort (proxy)	HTTP プロキシーサーバーポート		Integer
httpProxyUser (proxy)	HTTP プロキシーサーバーのユーザー名		String
oAuthAccessToken (security)	ユーザーアクセストークン		String
oAuthAccessTokenURL (security)	OAuth アクセストークン URL	https://graph.facebook.com/oauth/access_token	String

名前	説明	デフォルト	タイプ
oAuthAppId (security)	アプリケーション ID		String
oAuthAppSecret (security)	アプリケーションシークレット		String
oAuthAuthorizationURL (security)	OAuth 認可 URL	https://www.facebook.com/dialog/oauth	String
oAuthPermissions (security)	デフォルトの OAuth 権限。コンマ区切りの権限名。詳細は、https://developers.facebook.com/docs/reference/login/permissions を参照してください。		文字列

97.3. プロデューサーエンドポイント:

プロデューサーエンドポイントは、以下の表のエンドポイント名とオプションを使用できます。**getCheckin** と **searchCheckin** の間にあいまいさがあるため、**checkin** を除いて、エンドポイントは **get** または **search** 接頭辞なしで短い名前を使用することもできます。必須ではないエンドポイントオプションはで示されます。

プロデューサーエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。たとえば、次のルートの facebook エンドポイントは、受信メッセージボディーのユーザー ID 値のアクティビティーを取得します。

```
from("direct:test").to("facebook://activities?inBody=userId")...
```

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelFacebook.https://cwiki.apache.org/confluence/pages/createpage.action?spaceKey=CAMEL&title=option&linkCreation=true&fromPageId=34020899[option]** 形式である必要があります。たとえば、前のルートの **userId** オプション値は、代わりにメッセージヘッダー **CamelFacebook.userId** で提供できます。inBody オプションはメッセージヘッダーをオーバーライドすることに注意してください。たとえば、エンドポイントオプション **inBody=user** は **CamelFacebook.userId** ヘッダーをオーバーライドします。

文字列を返すエンドポイントは、作成または変更されたエンティティーの ID を返します。たとえば、**addAlbuPhoto は** 新しいアルバム ID を返します。ブール値を返すエンドポイントは、成功した場合は true を返し、それ以外の場合は false を返します。Facebook API エラーの場合、エンドポイントは facebook4j.FacebookException が原因で RuntimeCamelException を出力します。

97.4. コンシューマーエンドポイント

reading#reading パラメーターを受け取るプロデューサーエンドポイントはどれも、コンシューマーエ

ンドポイントとして使用できます。ポーリングコンシューマーは、**since** および **until** フィールドを使用して、ポーリング間隔内に応答を取得します。他の読み取りフィールドに加えて、最初のポーリングのエンドポイントで **初期** 値を指定できます。

エンドポイントが単一のルート交換を通じて List (または **facebook4j.ResponseList**) を返すのではなく、camel-facebook は返されたオブジェクトごとに 1 つのルート交換を作成します。例として、**facebook://home** の結果が 5 つの投稿になる場合、ルートは 5 回 (投稿ごとに 1 回) 実行されます。

97.5. 読み取りオプション

タイプ **facebook4j.Reading** の **読み取り** オプションは、読み取りパラメーターのサポートを追加します。これにより、特定のフィールドを選択したり、結果の数を制限したりできます。詳細については、Graph API#reading - Facebook Developers を参照してください。

また、コンシューマーエンドポイントが Facebook データをポーリングして、複数のポーリング間でメッセージが重複して送信されないようにするためにも使用されます。

読み取りオプションは、**facebook4j.Reading** 型の参照または値であるか、または**CamelFacebook.** 接頭辞を持つエンドポイントURIまたはエクスチェンジヘッダーのいずれかに以下の読み取りオプションを使用して指定することができます。

97.6. メッセージヘッダー

URI オプション #urioptions のいずれかを、**CamelFacebook** 接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーで指定できます。

97.7. メッセージボディー

すべての結果メッセージ本文は、Facebook4J API によって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

配列、または **facebook4j.ResponseList**、または **java.util.List** を返すエンドポイントの場合、コンシューマーエンドポイントはリスト内のすべての要素を個別のメッセージにマップします。

97.8. ユースケース

Facebook プロファイル内に投稿を作成するには、このプロデューサーに facebook4j.PostUpdate ボディーを送信します。

```
from("direct:foo")
    .to("facebook://postFeed/inBody=postUpdate);
```

5 秒ごとにポーリングするには (consumer の接頭辞を追加して polling consumer オプションを設定できます)、ホームフィードのすべてのステータスを次のようにします。

```
from("facebook://home?consumer.delay=5000")
    .to("bean:blah");
```

ヘッダーから動的オプションを使用してプロデューサーを使用して検索します。

バーヘッダーには、公開投稿で実行する Facebook 検索文字列があるため、この値を CamelFacebook.query ヘッダーに割り当てる必要があります。

```
from("direct:foo")
    .setHeader("CamelFacebook.query", header("bar"))
    .to("facebook://posts");
```

第98章 FHIR JSON データ形式

Camel バージョン 2.21 以降で利用可能Camel バージョン 2.21 以降で利用可能 Camel バージョン 2.21 以降で利用可能

FHIR-JSON データ形式は、HAPI-FHIR の JSON パーサーを利用して、JSON 形式との間で HAPI-FHIR の **IBaseResource** との間で解析を行います。

98.1. FHIR JSON 形式のオプション

FHIR JSon データ形式は、以下に示す 2 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
fhirVersion	**DSTU3**	**String**	使用する FHIR のバージョン。設定可能な値は、DSTU2、DSTU2_HL7ORG、DSTU2_1、DSTU3、R4 です
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

第99章 FHIR XML DATAFORMAT

Camel バージョン 2.21 以降で利用可能 Camel バージョン 2.21 以降で利用可能

FHIR-XML データフォーマットは、HAPI-FHIR の XML パーサーを利用して、HAPI-FHIR の **IBaseResource** から/へ XML フォーマットから/へ解析を行います。

99.1. FHIR XML 形式のオプション

FHIR XML データ形式は、以下に示す 2 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
fhirVersion	**DSTU3**	**String**	使用する FHIR のバージョン。設定可能な値は、DSTU2、DSTU2_HL7ORG、DSTU2_1、DSTU3、R4 です
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

第100章 FILE コンポーネント

Camel バージョン 1.0 以降で利用可能

File コンポーネントはファイルシステムへのアクセスを提供します。これにより、ファイルを他の Camel コンポーネントで処理したり、他のコンポーネントからのメッセージをディスクに保存したりできます。

100.1. URI 形式

```
file:directoryName[?options]
```

または

```
file://directoryName[?options]
```

directoryName は基礎となるファイルディレクトリーを表します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

ディレクトリーのみ

Camel は、開始ディレクトリーで設定されたエンドポイントのみをサポートします。そのため、**directoryName** はディレクトリーである必要があります。
1 つのファイルのみ使用する場合は、**fileName** オプションを使用できます (例: **fileName=thefilename**)。
また、開始ディレクトリーに $\{ } プレースホルダーを使用した動的な式を含めることはできません。ここでも、**fileName** オプションを使用してファイル名の動的部分を指定します。

警告

別のアプリケーションによって書き込まれている最中のファイルの読み取りは回避してください。JDK File IO API は、別のアプリケーションがファイルに対して書き込み/コピーを実行している最中かどうかの検出に少し制限があることに注意してください。実装は、OS プラットフォームによっても異なる場合があります。これにより、別のプロセスでロックされていないと Camel が判断して消費を開始する可能性があります。したがって、お使いの環境に何が適しているかを独自に調査する必要があります。これを支援するために、Camel では、使用できるさまざまな **readLock** オプションと **doneFileName** オプションを提供しています。**Consuming files from folders where others drop files directly**セクションも参照してください。

100.2. URI オプション

ファイルコンポーネントにはオプションがありません。

File エンドポイントは、URI 構文を使用して設定されます。

```
file:directoryName
```

パスおよびクエリーパラメーターを使用します。

100.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
directoryName	**必須**: 開始ディレクトリー		ファイル

100.2.2. クエリーパラメーター (81 パラメーター)

名前	説明	デフォルト	タイプ
charset (Common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。コンシューマーでこれを使用して、ファイルのエンコーディングを指定できます。これにより、Camel は、ファイルコンテンツがアクセスされている場合にファイルコンテンツをロードする必要がある charset を知ることができます。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む charset を指定できます。ファイルを書き込むとき、Camel はメッセージの内容をメモリーに読み込んで、データを設定された charset に変換できるようにする必要があることに注意してください。つまり、メッセージが大きい場合は、これを使用しないでください。		String
doneFileName (Common)	Producer: 指定された場合、元のファイルが書き込まれると、Camel は 2 番目の完了ファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的プレースホルダーを使用することもできます。完了ファイルは、常に元のファイルと同じフォルダーに書き込まれます。Consumer: 指定すると、Camel は完了ファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的なプレースホルダーを使用できます。完了ファイルは、常に元のファイルと同じフォルダーにあると想定されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		String

名前	説明	デフォルト	タイプ
fileName (Common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込むファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。(注: ヘッダー自体を式にすることもできます)。式オプションは String タイプと Expression タイプの両方をサポートします。式が String タイプである場合、これは常にファイル言語を使用して評価されます。式が Expression タイプである場合、指定された Expression タイプが使用されます。これにより、たとえば OGNL 式を使用できます。コンシューマーの場合、これを使用してファイル名をフィルターリングできるため、たとえば、ファイル言語構文 mydata-$date:now:yyyyMMdd.txt を使用して今日のファイルを消費できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は一度だけ使用されるヘッダーであり、CamelFileName を一時的に保存して後で復元する必要がなくなるため、簡単になります。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
delete (consumer)	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて move failure 式を設定します。たとえば、ファイルを .error サブディレクトリーに移動するには、.error を使用します。注: ファイルを失敗したロケーションに移動すると、Camel はエラーを処理し、ファイルを再度取得しません。		String
noop (consumer)	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel は idempotent=true も設定し、同じファイルを繰り返し消費しないようにします。	false	boolean

名前	説明	デフォルト	タイプ
preMove (consumer)	処理前に移動する場合にファイル名を動的に設定するために使用される式 (File 言語など)。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		String
preSort (consumer)	pre-sort が有効になっている場合、コンシューマーはポーリング中に、ファイルシステムから取得されたファイル名とディレクトリー名を並べ替えます。ソートされた順序でファイルを操作する必要がある場合に、これを行うことができます。pre-sort は、コンシューマーがフィルターリングを開始する前に実行され、Camel によって処理されるファイルを受け入れます。このオプション default=false で無効になっています。	false	boolean
recursive (consumer)	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist に似ていますが、これは再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
extendedAttributes (consumer)	対象のファイル属性を定義します。posix:permissions,posix:owner,basic:lastAccessTime と同様に、posix:、basic:lastAccessTime などの基本的なワイルドカードをサポートします。		String
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。in-progress リポジトリーは、現在進行中のファイルが消費されていることを示すために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>

名前	説明	デフォルト	タイプ
localWorkDirectory (consumer)	使用する場合、ローカルの作業ディレクトリーを使用して、リモートファイルのコンテンツをローカルファイルに直接保存し、コンテンツがメモリーに読み込まれないようにできます。これは、非常に大きなリモートファイルを使用している場合に、メモリーを節約するために役立ちます。		String
onCompletionException Handler (consumer)	カスタム org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを実行する完了プロセスのファイル中に出力される例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視します。		ExceptionHandler
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。つまり、ポーリングにより情報を収集しているときにエラーが発生し (例えばファイルネットワークへのアクセスに失敗した)、Camel はそれにアクセスしてファイルをスキャンできません。デフォルトの実装では、WARN レベルで原因となった例外をログに記録し、無視します。		PollingConsumerPoll Strategy
probeContentType (consumer)	コンテンツタイプのプローブを有効にするかどうか。有効な場合、コンシューマーはリンク FilesprobeContentType(java.nio.file.Path) を使用してファイルのコンテンツタイプを判断し、それをキーリンク ExchangeFILE_CONTENT_TYPE を持つヘッダーとしてメッセージに格納します。	false	boolean
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。特別な準備完了ファイルが存在するなど、ファイルを使用する前に特別な条件を満たす必要がある場合にも使用できます。このオプションを設定すると、readLock オプションは適用されません。		GenericFileProcess s Strategy<T>

名前	説明	デフォルト	タイプ
startingDirectoryMustExist (consumer)	開始ディレクトリーの存在が必要かどうか。autoCreate オプションがデフォルトで有効になっていることに注意してください。これは、開始ディレクトリーが存在しない場合、通常は自動作成されることを意味します。autoCreate を無効にして有効にすると、開始ディレクトリーの存在が必要なことを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。Override: これがデフォルトで、既存のファイルを置き換えます。append: 既存ファイルにコンテンツを追加します。fail: GenericFileOperationException を出力し、既存ファイルがあることを示します。ignore: 問題を警告なしで無視して既存のファイルは上書きしませんが、問題は発生していないと想定します。move: オプションを設定するには、moveExisting オプションも使用する必要があります。オプション eagerDeleteTargetFile を使用して、ファイルを移動する際に既存ファイルが存在する場合に何をすべきか制御でき、そうでない場合は移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用できます。これにより、存在チェックを実行せずに、一時的なファイル名から実際のファイル名への変更を試みることができます。このチェックは、一部のファイルシステム、特に FTP サーバーでは高速になる場合があります。	オーバーライド	GenericFileExist
flatten (producer)	flatten は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に使用できますが、たとえばファイルを別のディレクトリーに書き込む場合、ファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が先頭パスから削除されます。	false	boolean

名前	説明	デフォルト	タイプ
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式 (File 言語など)。ファイルをバックアップサブディレクトリーに移動するには、backup と入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントでは file:parent がサポートされていないことに注意してください。FTP コンポーネントは、既存のファイルを現在のディレクトリーをベースとする相対ディレクトリーにしか移動できないためです。		String
tempFileName (producer)	tempPrefix オプションと同じですが、ファイル言語を使用するため、一時ファイル名の命名をより細かく制御できます。		String
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込み、書き込みが完了した後に、その名前を実際の名前に変更するために使用されます。書き込み中のファイルを識別し、(排他的読み取りロックを使用せずに) コンシューマーが進行中のファイルを読み取らないようにするために使用できます。大きなファイルをアップロードするときに FTP でよく使用されます。		String
allowNullBody (producer)	ファイルの書き込み中に null の本文を許可するかどうかを指定するために使用されます。true に設定すると空のファイルが作成され、false に設定して null の本文をファイルコンポーネントに送信しようとすると、Cannot write null body to file.という GenericFileWriteException が出力されます。fileExist オプションを Override に設定するとファイルは切り捨てられ、append に設定するとファイルは変更されません。	false	boolean
chmod (producer)	プロデューサーによって送信されるファイルパーミッションを指定します。chmod の値は 000 から 777 の間の値である必要があります。0755 のように先頭に数字がある場合は無視します。		String
chmodDirectory (producer)	不足しているディレクトリーをプロデューサーが作成するときに使用するディレクトリーパーミッションを指定します。chmod 値は 000 から 777 の間である必要があります。0755 のように先頭に数字がある場合は無視します。		String

名前	説明	デフォルト	タイプ
eagerDeleteTargetFile (producer)	既存のターゲットファイルを先行して削除するかどうか。このオプションは、fileExists=Override および tempFileName オプションを使用している場合にのみ適用されます。これを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除することを無効化 (false に設定) できます。たとえば、大きなファイルを書き込んで、一時ファイルの書き込み中にターゲットファイルを存在させたい場合があります。これにより、一時ファイルの名前がターゲットファイル名に変更される直前まで、ターゲットファイルは削除されません。このオプションは、fileExist=Move が有効で、既存のファイルが存在する場合に、既存のファイルを削除するかどうかを制御するためにも使用されます。このオプション copyAndDeleteOnRenameFails が false の場合、既存のファイルが存在する場合は例外が出力されます。true の場合、移動操作の前に既存のファイルが削除されます。	true	boolean
forceWrites (producer)	ファイルシステムへの書き込みを強制的に同期するかどうか。たとえばログや監査ログへの書き込みなど、このレベルの保証が必要ない場合はオフにすることでパフォーマンスが向上します。	true	boolean
keepLastModified (producer)	ソースファイル (存在する場合) からの最終変更のタイムスタンプを保持します。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには、java.util.Date またはタイムスタンプ付きの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプが設定されます。注記: このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
autoCreate (advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	書き込みバッファーのサイズ (バイト単位)。	131072	int
copyAndDeleteOnRenameFail (advanced)	ファイルの名前を直接変更できなかった場合に、フォールバックしてファイルのコピーと削除を行うかどうか。このオプションは FTP コンポーネントでは使用できません。	true	boolean

名前	説明	デフォルト	タイプ
renameUsingCopy (advanced)	コピーおよび削除ストラテジーを使用して名前変更操作を実行します。これは主に、通常の名前変更操作が信頼できない環境で使用されます (たとえば、異なるファイルシステムまたはネットワーク間)。このオプションは、遅延が追加された後に限り、コピーおよび削除ストラテジーに自動的にフォールバックする copyAndDeleteOnRenameFail パラメーターよりも優先されます。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。コンマ区切り形式で複数の除外を指定できます。		String
antFilterCaseSensitive (filter)	ant フィルターに大文字と小文字を区別するフラグを設定します	true	boolean
antInclude (filter)	Ant スタイルフィルターの組み込み。コンマ区切り形式で複数の組み込みを指定できます。		String
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が eager かどうかを制御できます。eager の場合、ファイルのスキャン中に制限されます。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートしてからポーリングを制限できます。ソートのためにすべてのファイルの詳細がメモリー内にあるため、メモリー使用量が大きくなることに注意してください。	true	boolean
exclude (filter)	ファイル名が正規表現パターンに一致する場合にファイルを除外するために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
filter (filter)	org.apache.camel.component.file.GenericFileFilter クラスとしてのプラグ可能なフィルター。フィルターがその accept () メソッドで false を返す場合、ファイルをスキップします。		GenericFileFilter<T>

名前	説明	デフォルト	タイプ
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルターリングします。たとえば、現在の日付でフィルターリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		String
filterFile (filter)	Simple 言語に基づいてファイルをフィルターリングします。たとえば、ファイルサイズでフィルターリングするには、$file:size 5000 を使用できます。		String
idempotent (filter)	Camel が既に処理されたファイルをスキップできるように、Idempotent Consumer EIP パターンを使用するオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	Boolean
idempotentKey (filter)	カスタムのべき等性キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用できます。たとえば、ファイル名とファイルサイズを使用するには idempotentKey=$file:name-$file:size となります。		String
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、プラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository はデフォルトで MemoryMessageIdRepository を使用します。		String>
include (filter)	ファイル名が正規表現パターンに一致する場合にファイルを含めるために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
maxDepth (filter)	ディレクトリーを再帰的に処理する際にトラバースする最大深度。	2147483647	int

名前	説明	デフォルト	タイプ
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されていません。たとえば制限を 1000 などに設定して、数千のファイルがあるサーバーの起動を回避できます。無効にするには、0 または負の値を設定します。注記: このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 個のファイルがある場合に maxMessagesPerPoll=500 を使用すると、最初の 500 個のファイルのみ選択され、ソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 はベースディレクトリーを意味します。minDepth=2 は最初のサブディレクトリーを意味します。		int
move (filter)	処理後に移動する場合にファイル名を動的に設定するために使用される式 (Simple 言語など)。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		String
exclusiveReadLockStrategy (lock)	org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装としてのプラグ可能な読み取りロック。		GenericFileExclusiveReadLockStrategy<T>

名前	説明	デフォルト	タイプ
readLock (lock)	ファイルに排他的な読み取りロックがある (つまり、ファイルが進行中または書き込み中ではない) 場合にのみファイルをポーリングするために、コンシューマーが使用します。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none: 読み取りロックは使用されていません。markerFile: Camel はマーカーファイル (fileName.camelLock) を作成してロックを保持します。このオプションは、FTP コンポーネント changed では使用できません。changed は、ファイルの長さ/変更のタイムスタンプを使用して、ファイルが現在コピーされているかどうかを検出します。この判断には 1 秒以上かかるため、このオプションは他のオプションほど速くファイルを消費できませんが、JDK IO API はファイルが別のプロセスで使用中かどうか判断できないため、信頼性が高くなります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。ファイルシステムが分散ファイルロックをサポートしていない限り、マウント/共有によりリモートファイルシステムにアクセスする場合、このアプローチは避ける必要があります。rename: 排他的な読み取りロックを取得できるかどうかのテストとしてファイル名の変更を試みるために使用します。idempotent: (ファイルコンポーネントのみ) 読み取りロックとして idempotentRepository を使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-changed: (ファイルコンポーネントのみ) idempotentRepository および changed を結合された read-lock として使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-rename: (ファイルコンポーネントのみ) idempotentRepository および rename を結合された読み取りロックとして使用するためのものです。これにより、べき等性リポジトリーの実装がサポートしている場合、クラスターリングをサポートする読み取りロックを使用できます。注記: さまざまな読み取りロックは、異なるノード上の同時コンシューマーが共有ファイルシステム上の同じファイルを求めて競合するクラスターモードでの動作にすべて適しているわけではありません。アトミックに近い操作を使用して空のマーカーファイルを作成する markerFile ですが、クラスターでの動作は保証されていません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等性リ	none	String

名前	説明	デフォルト	タイプ
	ポジトリーが Hazelcast コンポーネントや Infinispan などのクラスターリングに対応している場合、べき等性等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合、読み取りロックの間隔 (ミリ単位)。この間隔は、読み取りロックを取得する試行間のスリープに使用されます。たとえば、changed 読み取りロックを使用する場合、遅い書き込みに対応するために間隔を長く設定できます。デフォルトは 1 秒ですが、プロデューサーによるファイルの書き込みが非常に遅い場合は短すぎる可能性があります。注記: FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	1000	long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされなかった場合 (JVM クラッシュなど)、マーカーファイルを使用した読み取りロックが、ファイルシステムに残っている可能性のある孤立した読み取りロックファイルを起動時に削除する必要があるかどうか。このオプションを false にすると、孤立したロックファイルがあると Camel はそのファイルを取得しようとしなくなります。これは、別のノードが同じ共有ディレクトリーから同時にファイルを読み取っているが原因である可能性もあります。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルを変更できます。たとえば、ログを記録しないように OFF に設定できます。このオプションを適用できる readLock タイプは、changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename のみです。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	changed、rename、exclusive の読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、他のプロセスが同じファイルを取得するのを防ぐために、マーカーファイルも使用されます。このオプションを false に設定すると、この動作をオフにできます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みたくない場合などです。	true	boolean

名前	説明	デフォルト	タイプ
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションは、読み取りロックを取得しようとする前に、ファイルが経過しなければならない最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルに 5 分以上の経過を要求します。これにより、指定された期間以上のファイルの取得を試みるため、changed 読み取りロックが高速化されます。	0	long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小ファイル長を設定できます。デフォルトで Camel はファイルにデータが含まれていると想定するため、デフォルト値は 1 です。このオプションをゼロに設定すると、長さがゼロのファイルを使用できます。	1	long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に成功し、コミットが行われるときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトはファイルは削除されないため、競合状態が発生せず、別のアクティブなノードがファイルを取得しようとする可能性があります。代わりにべき等性リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする場合があります。これにより、競合状態の問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に失敗し、ロールバックが発生するときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが (ファイルがコミットされたかのように) 確認されます。	true	boolean

名前	説明	デフォルト	タイプ
readLockTimeout (lock)	読み取りロックでサポートされている場合、読み取りロックのオプションのタイムアウト (ミリ秒単位)。読み取りロックを許可できず、タイムアウトがトリガーされた場合、Camel はファイルをスキップします。次のポーリングで、Camel はファイルを再試行します。このときに、読み取りロックが許可される可能性があります。無期限を指定するには、0 以下の値を使用します。現在、fileLock、changed、および rename がタイムアウトに対応しています。注記: FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	10000	long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。デフォルト値は 500 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。デフォルト値は 1000 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。このオプションを使用すると、複数のコンシューマー間でスレッドプールを共有できます。		ScheduledExecutor Service
scheduler (scheduler)	カスタムの org.apache.camel.spi.ScheduledPollConsumerScheduler をプラグインして、ポーリングコンシューマーの実行時に起動するスケジューラーとして使用できるようにします。デフォルトの実装では ScheduledExecutorService を使用し、CRON 式をサポートする Quartz2 および Spring ベースのものがあります。注記: カスタムスケジューラーを使用している場合、initialDelay、useFixedDelay、timeUnit、および scheduledExecutorService のオプションが使用されていない可能性があります。Quartz2 スケジューラーの使用を参照するにはテキスト Quartz2 を使用し、Spring ベースを使用するにはテキスト spring を使用し、レジストリー内の ID でカスタムスケジューラーを参照するにはテキスト myScheduler を使用します。例については、Quartz2 ページを参照してください。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルします (ランダムな順序でのソート)	false	boolean

名前	説明	デフォルト	タイプ
sortBy (sort)	File 言語を使用したビルトインソート。ネストされたソートをサポートしているため、ファイル名でのソートと、2 つ目のグループとして変更日でソートできます。		String
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>

ヒント

ファイルプロデューサーのデフォルト動作 デフォルトでは、同じ名前の既存ファイルが存在する場合は、既存ファイルをオーバーライドします。

100.3. 移動および削除操作

移動または削除操作は、ルーティングが完了した後 (ポストコマンド) に実行されます。そのため、**Exchange** の処理中、ファイルはまだ受信トレイフォルダーにあります。

これを例で説明しましょう。

```
from("file://inbox?move=.done").to("bean:handleOrder");
```

ファイルが **inbox** フォルダーにドロップされると、ファイルコンシューマーはこれに気づき、**handleOrder** Bean にルーティングされる新しい **FileExchange** を作成します。次に、Bean は **File** オブジェクトを処理します。この時点で、ファイルはまだ **inbox** フォルダーにあります。Bean が完了してルートが完了すると、ファイルコンシューマーは移動操作を実行し、ファイルを **.done** サブフォルダーに移動します。

move および **preMove** オプションは、ディレクトリー名と見なされます (ただし、File 言語 や Simple などの式を使用する場合は、式の評価結果が使用されるファイル名になります。

```
move=../backup/copy-of-${file:name}
```

次に、使用する File 言語 で、使用するファイル名を返します)。これは、相対または絶対パスのいずれかです。相対の場合、ディレクトリーは、ファイルが消費されたフォルダー内のサブフォルダーとして作成されます。

デフォルトでは、Camel は消費されたファイルを、ファイルが消費されたディレクトリーに相対的な **.camel** サブフォルダーに移動します。

処理後にファイルを削除する場合、ルートは次のようになります。

```
from("file://inobox?delete=true").to("bean:handleOrder");
```

処理 **前** にファイルを移動する **事前** 移動操作が導入されました。これにより、処理される前にこのサブフォルダーに移動されるときに、どのファイルがスキャンされたかをマークできます。

```
from("file://inbox?preMove=inprogress").to("bean:handleOrder");
```

pre の動きと通常の動きを組み合わせることができます。

```
from("file://inbox?preMove=inprogress&move=.done").to("bean:handleOrder");
```

したがって、この状況では、ファイルは処理中は **inprogress** フォルダーにあり、処理後は **.done** フォルダーに移動されます。

100.4. MOVE および PREMOVE オプションのきめ細かな制御

move および **preMove** オプションは式ベースであるため、ディレクトリーおよび名前パターンの高度な設定を行う File 言語の機能をフルに活用できます。
実際、Camel は、入力したディレクトリー名を File 言語 式に内部的に変換します。したがって、**move=.done** と入力すると、Camel はこれを **${file:parent}/.done/${``file:onlyname**} に変換します。これは、オプション値に $\{ } を指定していないことを Camel が検出した場合にのみ行われます。したがって、$\{ } を入力すると Camel はそれを変換し **ない** ため、フルパワーを使用できます。

そのため、ファイルを今日の日付をパターンとしてバックアップフォルダーに移動する場合は、次のようにします。

```
move=backup/${date:now:yyyyMMdd}/${file:name}
```

100.5. MOVEFAILED について

moveFailed オプションを使用すると、正常に処理 **できなかった** ファイルを別のロケーション (選択したエラーフォルダーなど) に移動できます。たとえば、エラーフォルダー内のファイルをタイムスタンプ付きで移動するには、**moveFailed=/error/${file:name.noext}-${date:now:yyyyMMddHHmmssSSS}.${``file:ext**} を使用できます。

ファイル言語 でさらに例を参照してください

100.6. メッセージヘッダー

このコンポーネントでは、次のヘッダーがサポートされています。

100.6.1. ファイルプロデューサーのみ

ヘッダー	説明
CamelFileName	書き込むファイルの名前を指定します (エンドポイントディレクトリーに相対的)。この名前は **String** に指定できます。File 言語 または Simple 言語式を含む **文字列**、または式オブジェクト。**null** の場合、Camel はメッセージの一意の ID に基づいてファイル名を自動生成します。
CamelFileNameProduced	書き込まれた出力ファイルの実際の絶対ファイルパス (パス + 名前)。このヘッダーは Camel によって設定され、その目的は、書き込まれたファイルの名前をエンドユーザーに提供することです。

ヘッダー	説明
CamelOverruleFileName	Camel 2.11: **CamelFileName** ヘッダーを上書きするために使用され、代わりに値を使用します (ただし、プロデューサーはファイルの書き込み後にこのヘッダーを削除するため、一度だけです)。値は文字列のみです。オプション **fileName** が設定されている場合、これはまだ評価されていることに注意してください。

100.6.2. ファイルコンシューマーのみ

ヘッダー	説明
CamelFileName	エンドポイントで設定された開始ディレクトリーからのオフセットを含む相対ファイルパスとしての使用済みファイルの名前。
CamelFileNameOnly	ファイル名のみ (先行パスを含まない名前)。
CamelFileAbsolute	消費されたファイルが絶対パスを示すかどうかを指定する **boolean** オプション。通常、相対パスの場合は **false** にする必要があります。通常、絶対パスは使用されませんが、ファイルを絶対パスに移動できるように移動オプションを追加しました。しかし、他の場所でも使用できます。
CamelFileAbsolutePath	ファイルへの絶対パス。相対ファイルの場合、このパスは代わりに相対パスを保持します。
CamelFilePath	ファイルパス。相対ファイルの場合、これは開始ディレクトリー + 相対ファイル名です。絶対ファイルの場合、これは絶対パスです。
CamelFileRelativePath	相対パス。
CamelFileParent	親パス。
CamelFileLength	ファイルサイズを含む **long** 値。

ヘッダー	説明
CamelFileLastModified	ファイルの最終変更タイムスタンプを含む **Long** 値。Camel 2.10.3 **以前** では、タイプは **Date** です。

100.7. バッチコンシューマー

このコンポーネントは、Batch Consumer を実装します。

100.8. EXCHANGE プロパティー (ファイルコンシューマーのみ)

ファイルコンシューマーは **BatchConsumer** を実装するため、ポーリングするファイルのバッチ処理をサポートします。バッチ処理とは、Camel が次の追加プロパティーを Exchange に追加することを意味します。これにより、ポーリングされたファイルの数、現在のインデックス、およびバッチがすでに完了しているかどうかがわかります。

プロパティー	説明
CamelBatchSize	このバッチでポーリングされたファイルの総数。
CamelBatchIndex	バッチの現在のインデックス。0 から始まります。
CamelBatchComplete	バッチ内の最後の Exchange を示す **boolean** 値。最後のエントリーにのみ **true** に当てはまります。

これにより、たとえば、このバッチに存在するファイルの数を知ることができ、たとえば、Aggregator2 にこの数のファイルを集約させることができます。

100.9. 文字セットの使用

Camel 2.9.3 以降で利用可能
charset オプションを使用すると、コンシューマーエンドポイントとプロデューサーエンドポイントの両方でファイルのエンコードを設定できます。たとえば、utf-8 ファイルを読み込んで、ファイルを iso-8859-1 に変換する場合は、次のようにします。

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

ルートで **convertBodyTo** を使用することもできます。以下の例では、まだ utf-8 形式の入力ファイルがありますが、ファイルの内容を iso-8859-1 形式のバイト配列に変換します。そして、Bean にデータを処理させます。現在の文字セットを使用して送信トレイフォルダーにコンテンツを書き込む前。

```
from("file:inbox?charset=utf-8")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

コンシューマーエンドポイントで文字セットを省略した場合、Camel はファイルの文字セットを認識せず、デフォルトで UTF-8 を使用します。ただし、キー **org.apache.camel.default.charset** を使用して、JVM システムプロパティーをオーバーライドし、別のデフォルトエンコーディングを使用するように設定できます。

以下の例では、ファイルが UTF-8 エンコーディングでない場合、これが問題になる可能性があります。これは、ファイルを読み取るためのデフォルトのエンコーディングです。
この例では、ファイルを書き込むときに、コンテンツはすでにバイト配列に変換されているため、コンテンツをそのまま (さらにエンコーディングせずに) 直接書き込みます。

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

キー **Exchange.CHARSET_NAME** を使用してエクスチェンジのプロパティーを設定することにより、ファイルの書き込み時に動的なエンコーディングをオーバーライドして制御することもできます。たとえば、以下のルートでは、メッセージヘッダーの値を使用してプロパティーを設定します。

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .setProperty(Exchange.CHARSET_NAME, header("someCharsetHeader"))
  .to("file:outbox");
```

より単純にすることをお勧めします。同じエンコーディングのファイルをピックアップし、特定のエンコーディングでファイルを書き込みたい場合は、エンドポイントで **charset** オプションを使用することをお勧めします。

エンドポイントで **charset** オプションを明示的に設定した場合は、**Exchange.CHARSET_NAME** プロパティーに関係なく、その設定が使用されることに注意してください。

いくつかの問題がある場合は、**org.apache.camel.component.file** で DEBUG ロギングを有効にし、特定の文字セットを使用してファイルを読み書きするときに Camel ログを有効にすることができます。たとえば、以下のルートでは次のログが記録されます。

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

そしてログ:

```
DEBUG GenericFileConverter          - Read file /Users/davsclaus/workspace/camel/camel-
core/target/charset/input/input.txt with charset utf-8
DEBUG FileOperations                - Using Reader to write file: target/charset/output.txt with charset:
iso-8859-1
```

100.10. フォルダーとファイル名に関するよくある問題

Camel がファイルを生成する (ファイルを書き込む) 場合、選択したファイル名を設定する方法に影響するいくつかの問題があります。デフォルトでは、Camel はメッセージ ID をファイル名として使用します。メッセージ ID は通常、一意に生成された ID であるため、**ID-MACHINENAME-2443-1211718892437-1-0** のようなファイル名になります。そのようなファイル名が望ましくない場合は、**CamelFileName** メッセージヘッダーにファイル名を指定する必要があります。定数 **Exchange.FILE_NAME** も使用できます。

以下のサンプルコードは、メッセージ ID をファイル名として使用してファイルを生成します。

```
from("direct:report").to("file:target/reports");
```

report.txt をファイル名として使用するには、次の手順を実行する必要があります。

```
from("direct:report").setHeader(Exchange.FILE_NAME, constant("report.txt")).to( "file:target/reports");
```

i. 上記と同じですが、**CamelFileName** を使用します:

```
from("direct:report").setHeader("CamelFileName", constant("report.txt")).to( "file:target/reports");
```

そして、**fileName** URI オプションを使用してエンドポイントにファイル名を設定する構文。

```
from("direct:report").to("file:target/reports/?fileName=report.txt");
```

100.11. ファイル名式

ファイル名は、**expression** オプションを使用するか、**CamelFileName** ヘッダーの文字列ベースの File 言語 式として設定できます。構文とサンプルについては、File 言語 を参照してください。

100.12. 他のユーザーがファイルを直接ドロップしたフォルダーからファイルを消費する

他のアプリケーションがファイルを直接書き込むフォルダーからファイルを使用する場合は注意してください。さまざまな readLock オプションを見て、ユースケースに適したオプションを確認してください。ただし、最善の方法は、別のフォルダーに書き込み、書き込み後にファイルをドロップフォルダーに移動することです。ただし、ファイルをドロップフォルダーに直接書き込む場合、変更されたオプションは、ファイルサイズ/変更が一定期間にわたって変更されたかどうかを確認するためにファイル変更アルゴリズムを使用するため、ファイルが現在書き込み/コピーされているかどうかをより適切に検出できます。他の readLock オプションは Java File API に依存していますが、残念ながら、これは必ずしもこれを検出するのに適しているとは限りません。また、doneFileName オプションを確認することもできます。これは、マーカーファイル (完了ファイル) を使用して、ファイルが完了し、使用する準備が整ったときに通知します。

100.13. 完了ファイルの使用

Camel 2.6 以降で利用可能

以下のセクション **書き込み完了ファイル** も参照してください。

完了ファイルが存在する場合にのみファイルを使用する場合は、エンドポイントで **doneFileName** オプションを使用できます。

```
from("file:bar?doneFileName=done");
```

完了した **ファイル** がターゲットファイルと同じディレクトリーに存在する場合、bar フォルダーのファイルのみを消費します。Camel は、ファイルの消費が完了すると、**完了した** ファイルを自動的に削除します。Camel 2.9.3 以降では、**noop=true** が設定されている場合、Camel は **完了したファイルを** 自動的に削除しません。

ただし、ターゲットファイルごとに 1 つの **完了ファイル** を作成する方が一般的です。これは、1:1 の相関があることを意味します。これを行うには、**doneFileName** オプションで動的プレースホルダーを使用する必要があります。現在、Camel は次の 2 つの動的トークンをサポートしています: **file:name** と **file:name.noext** は $\{ } で囲む必要があります。コンシューマーは、**done ファイル** 名の静的部分のみを接頭辞または接尾辞 (両方ではない) としてサポートします。

```
from("file:bar?doneFileName=${file:name}.done");
```

この例では、**ファイル名** が .done の完了ファイルが存在する場合にのみ、ファイルがポーリングされます。以下に例を示します。

- **hello.txt** - 使用するファイルです
- **hello.txt.done** - 関連する完了ファイルです

次のように、done ファイルの接頭辞を使用することもできます。

```
from("file:bar?doneFileName=ready-${file:name}");
```

- **hello.txt** - 使用するファイルです
- **ready-hello.txt** - 関連する完了ファイルです

100.14. 完了ファイルの書き込み

Camel 2.6 以降で利用可能

ファイルを書き終わったら、ファイルが完成して書き終わったことを他の人に示すために、一種のマーカーとして追加の **完了 ファイル** を書きたいと思うかもしれません。これを行うには、ファイルプロデューサーエンドポイントで **doneFileName** オプションを使用できます。

```
.to("file:bar?doneFileName=done");
```

ターゲットファイルと同じディレクトリーに **done** という名前のファイルを作成するだけです。

ただし、ターゲットファイルごとに 1 つの完了ファイルを作成する方が一般的です。これは、1:1 の相関があることを意味します。これを行うには、**doneFileName** オプションで動的プレースホルダーを使用する必要があります。現在、Camel は次の 2 つの動的トークンをサポートしています: **file:name** と **file:name.noext** は $\{ } で囲む必要があります。

```
.to("file:bar?doneFileName=done-${file:name}");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーにある **foo.txt** の場合、**done-foo.txt** という名前のファイルを作成します。

```
.to("file:bar?doneFileName=${file:name}.done");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーにある **foo.txt** の場合、**foo.txt.done** という名前のファイルを作成します。

```
.to("file:bar?doneFileName=${file:name.noext}.done");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーにある **foo.txt** の場合、**foo.done** という名前のファイルが作成されます。

100.15. サンプル

#=== ディレクトリーから読み取り、別のディレクトリーに書き込む

```
from("file://inputdir/?delete=true").to("file://outputdir")
```

100.15.1. オーバーライド動的名を使用して、ディレクトリーから読み取り、別のディレクトリーに書き込みます

```
from("file://inputdir/?delete=true").to("file://outputdir?overruleFile=copy-of-${file:name}")
```

ディレクトリーをリッスンし、そこにドロップされた各ファイルのメッセージを作成します。内容を **outputdir** にコピーし、**inputdir** 内のファイルを削除します。

100.15.2. ディレクトリーから再帰的に読み取り、別のディレクトリーに書き込む

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

ディレクトリーをリッスンし、そこにドロップされた各ファイルのメッセージを作成します。内容を **outputdir** にコピーし、**inputdir** 内のファイルを削除します。サブディレクトリーに再帰的にスキャンします。サブディレクトリーを含めて、**outputdir** 内の **inputdir** と同じディレクトリー構造にファイルを配置します。

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

次の出力レイアウトになります。

```
outputdir/foo.txt
outputdir/sub/bar.txt
```

100.16. フラット化の使用

ファイルを同じディレクトリーの outputdir ディレクトリーに保存する場合に、ソースディレクトリーのレイアウトを無視して (パスをフラット化するなど)、ファイルプロデューサー側で **flatten=true** オプションを追加するだけです。

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir?flatten=true")
```

次の出力レイアウトになります。

```
outputdir/foo.txt
outputdir/bar.txt
```

100.17. ディレクトリーからの読み取りとデフォルトの移動操作

Camel はデフォルトで、処理されたファイルをファイルが消費されたディレクトリーの **.camel** サブディレクトリーに移動します。

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

次のようにレイアウトに影響します。
前

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

after

```
inputdir/.camel/foo.txt
inputdir/sub/.camel/bar.txt
outputdir/foo.txt
outputdir/sub/bar.txt
```

100.18. ディレクトリーから読み取り、JAVA でメッセージを処理する

```
from("file://inputdir/").process(new Processor() {
  public void process(Exchange exchange) throws Exception {
    Object body = exchange.getIn().getBody();
    // do some business logic with the input body
  }
});
```

本文は、**inputdir** ディレクトリーにドロップされたばかりのファイルを指す **File** オブジェクトになります。

100.19. ファイルへの書き込み

もちろん、Camel はファイルを書き込むこともできます。つまり、ファイルを生成します。以下のサンプルでは、ディレクトリーに書き込まれる前に処理する SEDA キューに関するいくつかのレポートを受け取ります。

100.19.1. Exchange.FILE_NAME を使用してサブディレクトリーに書き込む

単一のルートを使用して、任意の数のサブディレクトリーにファイルを書き込むことができます。そのようなルート設定がある場合:

```
<route>
  <from uri="bean:myBean"/>
  <to uri="file:/rootDirectory"/>
</route>
```

myBean でヘッダー **Exchange.FILE_NAME** を次のような値に設定できます。

```
Exchange.FILE_NAME = hello.txt => /rootDirectory/hello.txt
Exchange.FILE_NAME = foo/bye.txt => /rootDirectory/foo/bye.txt
```

これにより、単一のルートでファイルを複数の宛先に書き込むことができます。

100.19.2. 最終宛先に相対的な一時ディレクトリーを介してファイルを書き込む

宛先ディレクトリーからの相対ディレクトリーにファイルを一時的に書き込む必要がある場合があります。このような状況は通常、フィルタリング機能が制限された外部プロセスが、書き込み先のディレクトリーから読み取っているときに発生します。以下の例では、ファイルは **/var/myapp/filesInProgress** ディレクトリーに書き込まれ、データ転送が完了すると、原子的に /var/myapp/finalDirectory ディレクトリーに移動されます。

```
from("direct:start").
  to("file:///var/myapp/finalDirectory?tempPrefix=/../filesInProgress/");
```

100.20. ファイル名に式を使用する

このサンプルでは、今日の日付をサブフォルダー名として使用して、消費されたファイルをバックアップフォルダーに移動します。

```
from("file://inbox?move=backup/${date:now:yyyyMMdd}/${file:name}").to("...");
```

その他のサンプルについては、File 言語 を参照してください。

100.21. 同じファイルを複数回読み取ることを避ける (べき等コンシューマー)

Camel は Idempotent Consumer をコンポーネント内で直接サポートしているため、すでに処理されたファイルはスキップされます。この機能は、**idempotent=true** オプションを設定することで有効にできます。

```
from("file://inbox?idempotent=true").to("...");
```

Camel は絶対ファイル名を冪等キーとして使用して、重複ファイルを検出します。**Camel 2.11** 以降では、idempotentKey オプションで式を使用して、このキーをカスタマイズできます。たとえば、名前とファイルサイズの両方をキーとして使用するには

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentKey=${file:name}-${file:size}"/>
  <to uri="bean:processInbox"/>
</route>
```

デフォルトでは、Camel は消費されたファイルを追跡するためにインメモリーベースのストアを使用し、最大 1000 エントリーを保持する最も使用頻度の低いキャッシュを使用します。値に **#** 記号を使用して **idempotentRepository** オプションを使用して、指定された **id** を持つレジストリー内の Bean を参照していることを示すことにより、このストアの独自の実装をプラグインできます。

```
<!-- define our store as a plain spring bean -->
<bean id="myStore" class="com.mycompany.MyIdempotentStore"/>
```

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#myStore"/>
  <to uri="bean:processInbox"/>
</route>
```

以前に消費されたためにファイルをスキップした場合、Camel は **DEBUG** レベルでログを記録します。

```
DEBUG FileConsumer is idempotent and the file has been consumed before. Will skip this file:
target\idempotent\report.txt
```

100.22. ファイルベースの冪等リポジトリーの使用

このセクションでは、デフォルトとして使用されるメモリー内ベースの代わりに、ファイルベースのべき等リポジトリー **org.apache.camel.processor.idempotent.FileIdempotentRepository** を使用します。
このリポジトリーは、ファイルリポジトリーの読み取りを回避するために、第 1 レベルのキャッシュを使用します。ファイルリポジトリーのみを使用して、第 1 レベルのキャッシュのコンテンツを格納します。これにより、リポジトリーはサーバーの再起動後も存続できます。起動時にファイルのコンテンツを第 1 レベルのキャッシュにロードします。ファイル内の別々の行にキーを格納するため、ファイル構造は非常に単純です。デフォルトでは、ファイルストアのサイズ制限は 1MB です。ファイルが大きくなると、Camel はファイルストアを切り詰め、第 1 レベルのキャッシュを新しい空のファイルにフラッシュしてコンテンツを再構築します。

ファイルべき等リポジトリーを作成する Spring XML を使用してリポジトリーを設定し、**#** 記号を使用して **idempotentRepository** でリポジトリーを使用するようにファイルコンシューマーを定義して、レジストリールックアップを示します。

100.23. JPA ベースのべき等リポジトリーの使用

このセクションでは、デフォルトとして使用されるメモリー内ベースの代わりに、JPA ベースのべき等リポジトリーを使用します。

まず、クラス **org.apache.camel.processor.idempotent.jpa.MessageProcessed** をモデルとして使用する必要がある **META-INF/persistence.xml** に persistence-unit が必要です。

```
<persistence-unit name="idempotentDb" transaction-type="RESOURCE_LOCAL">
  <class>org.apache.camel.processor.idempotent.jpa.MessageProcessed</class>

  <properties>
    <property name="openjpa.ConnectionURL" value="jdbc:derby:target/idempotentTest;create=true"/>
    <property name="openjpa.ConnectionDriverName"
value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema"/>
    <property name="openjpa.Log" value="DefaultLevel=WARN, Tool=INFO"/>
    <property name="openjpa.Multithreaded" value="true"/>
  </properties>
</persistence-unit>
```

次に、Spring XML ファイルにも JPA べき等リポジトリーを作成できます。

```
<!-- we define our jpa based idempotent repository we want to use in the file consumer -->
<bean id="jpaStore" class="org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository">
```

```
    <!-- Here we refer to the entityManagerFactory -->
    <constructor-arg index="0" ref="entityManagerFactory"/>
    <!-- This 2nd parameter is the name  (= a category name).
         You can have different repositories with different names -->
    <constructor-arg index="1" value="FileConsumer"/>
</bean>
```

はい、**#** 構文オプションを使用して **idempotentRepository** を使用して、ファイルコンシューマーエンドポイントで **jpaStore** Bean を参照する必要があります。

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#jpaStore"/>
  <to uri="bean:processInbox"/>
</route>
```

100.24. ORG.APACHE.CAMEL.COMPONENT.FILE.GENERICFILEFILTER を使用するフィルター

Camel は、プラグイン可能なフィルタリング戦略をサポートしています。次に、そのようなフィルターを使用してエンドポイントを設定し、処理中の特定のファイルをスキップできます。

サンプルでは、ファイル名が **skip** で始まるファイルをスキップする独自のフィルターを作成しました。

そして、**フィルター** 属性を使用してルートを設定し、Spring XML ファイルで定義したフィルターを (**#** 表記を使用して) 参照できます。

```
<!-- define our filter as a plain spring bean -->
<bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="file://inbox?filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

100.25. ANT パスマッチャーを使用したフィルタリング

ANT パスマッチャーは、**camel-Spring** jar ですぐに利用できる状態で出荷されています。したがって、Maven を使用している場合は **camel-spring** に依存する必要があります。
その理由は、Spring の AntPathMatcher を利用して実際のマッチングを行うためです。

ファイルパスは、次のルールに一致します。

- **?** 1 文字に一致
- * 0 個以上の文字に一致
- ** パス内の 0 個以上のディレクトリーに一致

ヒント

Camel 2.10 以降の新しいオプション antInclude および **antExclude** オプションが追加され、フィルターを定義しなくても ANT スタイルの include/exclude を簡単に指定できるようになりました。詳細については、上記の URI オプションを参照してください。

以下のサンプルは、その使用方法を示しています。

100.25.1. コンパレータを使用した並べ替え

Camel は、プラグイン可能な並べ替え戦略をサポートしています。この戦略は、Java の **java.util.Comparator** でビルドを使用することです。次に、このようなコンパレーターを使用してエンドポイントを設定し、処理する前に Camel にファイルをソートさせることができます。

サンプルでは、ファイル名でソートする独自のコンパレータを作成しました。

次に、**sorter** オプションを使用してルートを設定し、Spring XML ファイルで定義したソーター (**mySorter**) を参照できます。

```
<!-- define our sorter as a plain spring bean -->
<bean id="mySorter" class="com.mycompany.MyFileSorter"/>

<route>
  <from uri="file://inbox?sorter=#mySorter"/>
  <to uri="bean:processInbox"/>
</route>
```

ヒント

URI オプションは、# 構文を使用して Bean を参照できます。上記の Spring DSL アプリケーションでは、id の前に **#** を付けることで、レジストリー内の Bean を参照できます。したがって、**sorter=#mySorter** と記述すると、Camel はレジストリーで ID が **mySorter** の Bean を探すように指示されます。

100.25.2. sortBy を使用した並べ替え

Camel は、プラグイン可能な並べ替え戦略をサポートしています。このストラテジーは、File 言語 を使用して並べ替えを設定することです。**sortBy** オプションは次のように設定されます。

```
sortBy=group 1;group 2;group 3;...
```

各グループはセミコロンで区切ります。単純な状況では、1 つのグループのみを使用するため、単純な例は次のようになります。

```
sortBy=file:name
```

これはファイル名でソートされます。グループの先頭に **reverse:** を付けることで順序を逆にすることができるため、ソートは Z..A: になります。

```
sortBy=reverse:file:name
```

File 言語 の全機能を使用できるので、他のパラメーターの一部を使用できるので、ファイルサイズで並べ替える場合は、次のようにします。

```
sortBy=file:length
```

文字列の比較に **ignoreCase:** を使用して、大文字と小文字を区別しないように設定できます。そのため、ファイル名の並べ替えを使用したいが大文字と小文字を区別したくない場合は、次のようにします。

```
sortBy=ignoreCase:file:name
```

ignore case と reverse を組み合わせることができますが、reverse を最初に指定する必要があります。

```
sortBy=reverse:ignoreCase:file:name
```

以下のサンプルでは、最後に変更されたファイルで並べ替えたいので、次のようにします。

```
sortBy=file:modified
```

次に、2 番目のオプションとして名前でグループ化し、同じ変更を含むファイルが名前でソートされるようにします。

```
sortBy=file:modified;file:name
```

ここで問題が発生しました。それを見つけることができますか?ファイルの変更されたタイムスタンプはミリ秒単位なので細かすぎますが、日付のみで並べ替えてから名前でサブグループ化したい場合はどうすればよいでしょうか?
File 言語 の実際の機能で、パターンをサポートする date コマンドを使用できます。したがって、これは次のように解決できます。

```
sortBy=date:file:yyyyMMdd;file:name
```

ええ、それは非常に強力です。ちなみに、グループごとにリバースを使用することもできるので、ファイル名を逆にすることができます。

```
sortBy=date:file:yyyyMMdd;reverse:file:name
```

100.26. GENERICFILEPROCESSSTRATEGY の使用

オプション **processStrategy** を使用して、独自の **begin**、**commit**、および **rollback** ロジックを実装できるカスタム **GenericFileProcessStrategy** を使用できます。
たとえば、システムが、使用する必要があるフォルダーにファイルを書き込むと仮定します。ただし、別の **準備完了** ファイルが同様に書き込まれる前に、ファイルの使用を開始しないでください。

したがって、独自の **GenericFileProcessStrategy** を実装することで、これを次のように実装できます。

- **begin ()** メソッドでは、特別な **準備完了** ファイルが存在するかどうかをテストできます。begin メソッドは **ブール値** を返し、ファイルを使用できるかどうかを示します。
- **abort()** メソッド (Camel 2.10) では、**begin** オペレーションが **false** を返した場合に、リソースのクリーンアップなどの特別なロジックを実行できます。
- **commit ()** メソッドでは、実際のファイルを移動し、**準備完了** ファイルを削除することもできます。

100.27. フィルターの使用

filter オプションを使用すると、**org.apache.camel.component.file.GenericFileFilter** インターフェイスを実装することにより、Java コードでカスタムフィルターを実装できます。このインターフェイスには、ブール値を返す **accept** メソッドがあります。ファイルを含めるには **true** を返し、ファイルをスキップするには **false** を返します。Camel 2.10 以降では、ファイルがディレクトリーであるかどうかにかかわらず、**GenericFile** に **isDirectory** メソッドがあります。これにより、不要なディレクトリーをフィルタリングして、不要なディレクトリーをたどることを回避できます。

たとえば、名前が **"skip"** で始まるディレクトリーをスキップするには、次のように実装できます。

100.28. CONSUMER.BRIDGEERRORHANDLER の使用

Camel 2.10 以降で利用可能

Camel エラーハンドラーを使用してファイルコンシューマーで発生した例外を処理する場合は、以下に示すように **consumer.bridgeErrorHandler** オプションを有効にできます。

```
// to handle any IOException being thrown
onException(IOException.class)
    .handled(true)
    .log("IOException occurred due: ${exception.message}")
    .transform().simple("Error ${exception.message}")
    .to("mock:error");

// this is the file route that pickup files, notice how we bridge the consumer to use the Camel routing error handler
// the exclusiveReadLockStrategy is only configured because this is from an unit test, so we use that to simulate exceptions
from("file:target/nospace?consumer.bridgeErrorHandler=true")
    .convertBodyTo(String.class)
    .to("mock:result");
```

したがって、このオプションを有効にするだけで、ルート内のエラーハンドラーがそこから取得します。

重要

consumer.bridgeErrorHandler を使用する場合の重要事項 consumer.bridgeErrorHandler を使用する場合、インターセプター、OnCompletions は適用さ **れません**。Exchange は Camel エラーハンドラーによって直接処理され、インターセプターや onCompletion などの前のアクションがアクションを実行することを許可しません。

100.29. デバッグロギング

このコンポーネントには、問題が発生した場合に役立つログレベル **TRACE** があります。

100.30. 関連項目

- File 言語
- FTP
- Polling Consumer

第101章 FILE 言語

Camel バージョン 1.1 以降で利用可能

情報:*ファイル言語は Simple 言語に統合されました* Camel 2.2 以降、ファイル言語は Simple 言語に統合されました。これは、すべてのファイル構文を Simple 言語内で直接使用できることを意味します。

File Expression Language は Simple 言語の拡張機能であり、File 関連の機能が追加されています。これらの機能は、ファイルパスと名前を操作する一般的なユースケースに関連しています。目標は、式をコンシューマーとプロデューサーの両方に動的ファイルパターンを設定するためのファイルおよび FTP コンポーネントと合わせて使用することです。

101.1. FILE 言語オプション

File 言語は、以下に示す 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
resultType		**String**	結果の型 (出力からの型) のクラス名を設定します
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

101.2. 構文

この言語は Simple 言語の **拡張** であるため、Simple 構文も適用されます。したがって、以下の表には追加のみがリストされています。
Simple 言語とは対照的に、ファイル言語は 定数式も サポートしているため、固定のファイル名を入力できます。

すべての File トークンは、**java.io.File** オブジェクトのメソッドと同じ式名を使用します。たとえば、**file:absolute** は **java.io.File.getAbsolute ()** メソッドを参照します。すべての式が現在の Exchange でサポートされているわけではないことに注意してください。たとえば、FTP コンポーネントはいくつかのオプションをサポートしていますが、ファイルコンポーネントはそれらすべてをサポートしています。

式	タイプ	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
file:name	String	はい	いいえ	はい	いいえ	ファイル名を指します (開始ディレクトリーを起点にした相対パスにあります。以下の注記を参照してください)。
file:name.ext	String	はい	いいえ	はい	いいえ	**Camel 2.3:** ファイル拡張子のみを参照

式	タイプ	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
file:name.ext.single	String	はい	いいえ	はい	いいえ	**Camel 2.14.4/2.15.3:** ファイル拡張子を指します。ファイル拡張子に複数のドットがある場合には、この式は削除され、最後の部分のみが返されます。
file:name.noext	String	はい	いいえ	はい	いいえ	拡張子のないファイル名を指します (開始ディレクトリーを起点にした相対パスにあります。以下の注を参照してください)
file:name.noext.single	String	はい	いいえ	はい	いいえ	**Camel 2.14.4/2.15.3:** 拡張子のないファイル名を指します (開始ディレクトリーを起点にした相対パスにあります。以下の注を参照してください) ファイル拡張子に複数のドットがある場合、この変数は最後の部分のみを取り除き、他の部分を返します。
file:onlyname	String	はい	いいえ	はい	いいえ	先行パスなしでファイル名のみを指します。
file:onlyname.noext	String	はい	いいえ	はい	いいえ	拡張子や先行パスのないファイル名のみを指します。
file:onlyname.noext.single	String	はい	いいえ	はい	いいえ	*Camel 2.14.4/2.15.3:*拡張子や先行パスのないファイル名のみを指します。ファイル拡張子に複数のドットがある場合、この変数は最後の部分のみを取り除き、他の部分を返します。
file:ext	String	はい	いいえ	はい	いいえ	ファイル拡張子のみを指します

式	タイプ	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
file:parent	String	はい	いいえ	はい	いいえ	ファイルの親を参照します
file:path	String	はい	いいえ	はい	いいえ	ファイルパスを参照します
file:absolute	Boolean	はい	いいえ	いいえ	いいえ	ファイルが絶対パスまたは相対パスと見なされるかどうかを示します。
file:absolute.path	String	はい	いいえ	いいえ	いいえ	絶対ファイルパスを指します。
file:length	Long	はい	いいえ	はい	いいえ	Long 型として返されるファイルの長さを参照します
file:size	Long	はい	いいえ	はい	いいえ	**Camel 2.5:** Long 型として返されるファイルの長さを参照します
file:modified	日付	はい	いいえ	はい	いいえ	日付型として返された最終変更ファイルを指します。
date:_command:pattern_	String	はい	はい	はい	はい	**java.text.SimpleDateFormat** パターンを使用した日付フォーマット用。Simple 言語の **拡張** です。追加のコマンドは、ファイルの最終変更タイムスタンプ向けの **file** (コンシューマーのみ) です。注意: Simple 言語のすべてのコマンドも使用できます。

101.3. FILE トークンの例

101.3.1. 相対パス

相対 ディレクトリー (**.\filelanguage\test**) にファイル **hello.txt** の **java.io.File** ハンドルがあります。そして、この開始ディレクトリー **.\filelanguage** を使用するようにエンドポイントを設定します。File トークンは次のように返されます。

式	戻り値
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt
file:parent	filelanguage\test
file:path	filelanguage\test\hello.txt
file:absolute	false
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

101.3.2. 絶対パス

絶対 ディレクトリー (**\workspace\camel\camel-core\target\filelanguage\test**) にファイル **hello.txt** の **java.io.File** ハンドルがあります。そして、絶対開始ディレクトリー **\workspace\camel\camel-core\target\filelanguage** を使用するように out エンドポイントを設定します。File トークンは次のように返されます。

式	戻り値
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt

式	戻り値
file:parent	\workspace\camel\camel-core\target\filelanguage\test
file:path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt
file:absolute	true
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

101.4. サンプル

myfile.txt などの固定 定数 式を入力できます。

```
fileName="myfile.txt"
```

ファイルコンシューマーを使用してファイルを読み取り、読み取ったファイルを移動して、現在の日付をサブフォルダーとして持つバックアップフォルダーに移動するとします。これは、次のような式を使用してアーカイブできます。

```
fileName="backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

相対フォルダー名もサポートされているため、バックアップフォルダーがシブリングフォルダーであると仮定すると、次のように .. を追加できます。

```
fileName="../backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

これは Simple 言語の拡張であるため、この言語の利点をすべて活用できるので、このユースケースでは、in.header.type を動的式のパラメーターとして使用します。

```
fileName="../backup/${date:now:yyyyMMdd}/type-${in.header.type}/backup-of-${file:name.noext}.bak"
```

式で使用するカスタムの日付がある場合、Camel はメッセージヘッダーからの日付の取得をサポートします

```
fileName="orders/order-${in.header.customerId}-${date:in.header.orderDate:yyyyMMdd}.xml"
```

最後に、Bean 式を使用して、使用する文字列出力 (または文字列に変換可能) を生成する POJO クラスを呼び出すこともできます。

```
fileName="uniquefile-${bean:myguidgenerator.generateid}.txt"
```

そしてもちろん、これらすべてを 1 つの式に結合して、ファイル言語、Simple および Bean 言語を 1 つの結合式で使用できます。これは、これらの一般的なファイルパスパターンに対して非常に強力です。

101.5. SPRING PROPERTYPLACEHOLDERCONFIGURER を FILE コンポーネントと一緒に使用する

Camel では、Simple 言語から ファイル言語 を直接使用できます。これにより、以下に示すように、ファイル拡張子に基づいてルーティングできる Spring XML でコンテンツベースのルーターを簡単に実行できます。

```
<from uri="file://input/orders"/>
  <choice>
    <when>
      <simple>${file:ext} == 'txt'</simple>
      <to uri="bean:orderService?method=handleTextFiles"/>
    </when>
    <when>
      <simple>${file:ext} == 'xml'</simple>
      <to uri="bean:orderService?method=handleXmlFiles"/>
    </when>
    <otherwise>
      <to uri="bean:orderService?method=handleOtherFiles"/>
    </otherwise>
  </choice>
```

File エンドポイントで **fileName** オプションを使用して、 ファイル言語 で動的ファイル名を設定する場合は、
別の構文 (Camel 2.5 以降で使用可能) を使用して、Springs **PropertyPlaceholderConfigurer** との衝突を回避します。

bundle-context.xml

```
<bean id="propertyPlaceholder"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:bundle-context.cfg" />
</bean>

<bean id="sampleRoute" class="SampleRoute">
    <property name="fromEndpoint" value="${fromEndpoint}" />
    <property name="toEndpoint" value="${toEndpoint}" />
</bean>
```

bundle-context.cfg

```
fromEndpoint=activemq:queue:test
toEndpoint=file://fileRoute/out?fileName=test-$simple{date:now:yyyyMMdd}.txt
```

上記の **toEndpoint** で $simple\{ } 構文を使用する方法に注意してください。
これを行わないと、衝突が発生し、Spring は次のような例外を出力します。

```
org.springframework.beans.factory.BeanDefinitionStoreException:
Invalid bean definition with name 'sampleRoute' defined in class path resource [bundle-context.xml]:
Could not resolve placeholder 'date:now:yyyyMMdd'
```

101.6. 依存関係

File 言語は **camel-core** の一部です。

第102章 FLATPACK コンポーネント

Camel バージョン 1.4 以降で利用可能

Flatpack コンポーネントは、FlatPack ライブラリー を介した固定幅および区切りファイルの解析をサポートしています。
注意: このコンポーネントは、フラットパックファイルからオブジェクトモデルへの消費のみをサポートします。(まだ) オブジェクトモデルから flatpack 形式に書き込むことはできません。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

102.1. URI 形式

```
flatpack:[delim|fixed]:flatPackConfig.pzmap.xml[?options]
```

または、設定ファイルのない区切りファイルハンドラーの場合は、単に使用します

```
flatpack:someName[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

102.2. URI オプション

Flatpack コンポーネントにはオプションがありません。

Flatpack エンドポイントは、URI 構文を使用して設定されます。

```
flatpack:type:resourceUri
```

パスおよびクエリーパラメーターを使用します。

102.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
type	固定文字と区切り文字のどちらを使用するか	delim	FlatpackType
resourceUri	クラスパスまたはファイルシステムからフラットパックマッピングファイルをロードするために **必要な** URL		String

102.2.2. クエリーパラメーター(25 個のパラメーター):

名前	説明	デフォルト	タイプ
allowShortLines (common)	行を予想より短くすることができ、余分な文字は無視されます	false	boolean
delimiter (common)	区切りファイルのデフォルトの区切り文字。	,	char
ignoreExtraColumns (common)	行が予想よりも長くなる可能性があり、余分な文字は無視されます	false	boolean
ignoreFirstRecord (common)	区切りファイル (列ヘッダー) の最初の行を無視するかどうか。	true	boolean
splitRows (common)	解析後に各行を個別のエクスチェンジとして送信するようにコンポーネントを設定します	true	boolean
textQualifier (common)	区切りファイルのテキスト修飾子。		char
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

102.3. 例

- **flatpack:fixed:foo.pzmap.xml** は、**foo.pzmap.xml** ファイル設定を使用して固定幅のエンドポイントを作成します。
- **flatpack:delim:bar.pzmap.xml** は、**bar.pzmap.xml** ファイル設定を使用して、区切られたエンドポイントを作成します。
- **flatpack:foo** は、ファイル設定のない **foo** という区切りのエンドポイントを作成します。

102.4. メッセージヘッダー

Camel は、IN メッセージに次のヘッダーを保存します。

ヘッダー	説明
camelFlatpackCounter	現在の行インデックス。**splitRows=false** の場合、カウンターは行の総数です。

102.5. メッセージボディー

コンポーネントは、IN メッセージ内のデータを、**java.util.Map** または **java.util.List** のコンバーターを持つ **org.apache.camel.component.flatpack.DataSetList** オブジェクトとして配信します。
一度に 1 つの行を処理する場合 (**splitRows=true**)、通常は **Map** が必要です。コンテンツ全体 (**splitRows=false**) には **List** を使用します。リスト内の各要素は **Map** です。
各 **Map** には、列名のキーとそれに対応する値が含まれています。

たとえば、以下のサンプルから名を取得するには:

```
Map row = exchange.getIn().getBody(Map.class);
String firstName = row.get("FIRSTNAME");
```

ただし、常に **List** として取得することもできます (**splitRows=true** の場合でも)。同じ例:

```
List data = exchange.getIn().getBody(List.class);
Map row = (Map)data.get(0);
String firstName = row.get("FIRSTNAME");
```

102.6. ヘッダーおよびトレーラーレコード

Flatpack のヘッダーとトレーラーの概念がサポートされています。ただし、固定レコード ID を使用する **必要があります**。

- ヘッダーレコードの **header** (小文字にする必要があります)
- トレーラーレコードの **trailer** (小文字にする必要があります)

以下の例は、ヘッダーとトレーラーがあるという事実を示しています。不要な場合は、一方または両方を省略できます。

```
<RECORD id="header" startPosition="1" endPosition="3" indicator="HBT">
    <COLUMN name="INDICATOR" length="3"/>
    <COLUMN name="DATE" length="8"/>
</RECORD>

<COLUMN name="FIRSTNAME" length="35" />
<COLUMN name="LASTNAME" length="35" />
<COLUMN name="ADDRESS" length="100" />
<COLUMN name="CITY" length="100" />
<COLUMN name="STATE" length="2" />
<COLUMN name="ZIP" length="5" />

<RECORD id="trailer" startPosition="1" endPosition="3" indicator="FBT">
    <COLUMN name="INDICATOR" length="3"/>
    <COLUMN name="STATUS" length="7"/>
</RECORD>
```

102.7. エンドポイントの使用

一般的な使用例は、別のルートでさらに処理するために、このエンドポイントにファイルを送信することです。以下に例を示します。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="file://someDirectory"/>
    <to uri="flatpack:foo"/>
  </route>

  <route>
    <from uri="flatpack:foo"/>
```

```
    ...
  </route>
</camelContext>
```

簡単な Bean 統合のために、作成された各メッセージのペイロードを **Map** に変換することもできます

102.8. FLATPACK DATAFORMAT

Flatpack コンポーネントには Flatpack データ形式が付属しており、これを使用して、固定幅または区切りテキストメッセージを **Map** として行の **List** にフォーマットすることができます。

- marshal = **List<Map<String, Object>>** から **OutputStream** へ (**String** に変換可能)
- unmarshal = **java.io.InputStream** (**File** や **String** など) から **org.apache.camel.component.flatpack.DataSetList** インスタンスとしての **java.util.List** へ。操作の結果には、すべてのデータが含まれます。各行を 1 つずつ処理する必要がある場合は、Splitter を使用してエクスチェンジを分割できます。

注意: Flatpack ライブラリーは現在、マーシャル操作のヘッダーとトレーラーをサポートしていません。

102.9. オプション

データ形式には次のオプションがあります。

オプション	デフォルト	説明
定義	**null**	flatpack pzmap 設定ファイル。単純な状況では省略できますが、pzmap を使用することをお勧めします。
固定:	**false**	区切りまたは固定。
ignoreFirstRecord	**true**	区切りファイル (列ヘッダー) の最初の行を無視するかどうか。
textQualifier	**"**	テキストが **"** などの文字で修飾されている場合。
delimiter	**,**	区切り文字 (**;** ,など)
parserFactory	**null**	デフォルトの Flatpack パーサーファクトリーを使用します。
allowShortLines	**false**	**Camel 2.9.7 および 2.10.5 以降**: 行を予想より短くすることができ、余分な文字は無視されます。

オプション	デフォルト	説明
ignoreExtraColumns	**false**	Camel 2.9.7 および 2.10.5 以降: 行が予想よりも長くなっても、余分な文字は無視されます。

102.10. 使用方法

データ形式を使用するには、単にインスタンスをインスタンス化し、ルートビルダーで整列化または非整列化操作を呼び出します。

```
FlatpackDataFormat fp = new FlatpackDataFormat();
fp.setDefinition(new ClassPathResource("INVENTORY-Delimited.pzmap.xml"));
...
from("file:order/in").unmarshal(df).to("seda:queue:neworder");
```

上記のサンプルは、**order/in** フォルダーからファイルを読み取り、ファイルの構造を設定する Flatpack 設定ファイル **INVENTORY-Delimited.pzmap.xml** を使用して入力を非整列化します。結果は、SEDA キューに格納する **DataSetList** オブジェクトです。

```
FlatpackDataFormat df = new FlatpackDataFormat();
df.setDefinition(new ClassPathResource("PEOPLE-FixedLength.pzmap.xml"));
df.setFixed(true);
df.setIgnoreFirstRecord(false);

from("seda:people").marshal(df).convertBodyTo(String.class).to("jms:queue:people");
```

上記のコードでは、Object 表現からのデータを、行の **List** として **Maps** としてマーシャリングします。**Map** としての行には、キーとしての列名と、対応する値が含まれます。この構造は、プロセッサーなどから Java コードで作成できます。Flatpack 形式に従ってデータをマーシャリングし、結果を **String** オブジェクトとして変換して JMS キューに格納します。

102.11. 依存関係

camel ルートで Flatpack を使用するには、このデータ形式を実装する **camel-flatpack** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
</dependency>
```

102.12. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント
- エンドポイント
- スタートガイド

第103章 FLATPACK DATAFORMAT

Camel バージョン 2.1 以降で利用可能

Flatpack コンポーネントには Flatpack データ形式が付属しており、これを使用して、固定幅または区切りテキストメッセージを **Map** として行の **List** にフォーマットすることができます。

- marshal = **List<Map<String, Object>>** から **OutputStream** へ (**String** に変換可能)
- unmarshal = **java.io.InputStream** (**File** や **String** など) から **org.apache.camel.component.flatpack.DataSetList** インスタンスとしての **java.util.List** へ。操作の結果には、すべてのデータが含まれます。各行を 1 つずつ処理する必要がある場合は、Splitter を使用してエクスチェンジを分割できます。

注意: Flatpack ライブラリーは現在、マーシャル操作のヘッダーとトレーラーをサポートしていません。

103.1. オプション

Flatpack データ形式は、以下に示す 9 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
定義		**String**	flatpack pzmap 設定ファイル。単純な状況では省略できますが、pzmap を使用することをお勧めします。
固定:	**false**	**Boolean**	区切りまたは固定。デフォルトでは false = 区切られています
ignoreFirstRecord	**true**	**Boolean**	区切りファイル (列ヘッダー) の最初の行を無視するかどうか。デフォルトでは true です。
textQualifier		**String**	テキストが文字で修飾されている場合。デフォルトで引用符を使用します。
delimiter	,	**String**	区切り文字 (; など)
allowShortLines	**false**	**Boolean**	行を予想より短くすることができ、余分な文字は無視されます
ignoreExtraColumns	**false**	**Boolean**	予想よりも長い行を許可し、余分な文字を無視します。
parserFactoryRef		**String**	レジストリーを検索するためのカスタムパーサーファクトリーへの参照

名前	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

103.2. 使用方法

データ形式を使用するには、単にインスタンスをインスタンス化し、ルートビルダーで整列化または非整列化操作を呼び出します。

```
FlatpackDataFormat fp = new FlatpackDataFormat();
fp.setDefinition(new ClassPathResource("INVENTORY-Delimited.pzmap.xml"));
...
from("file:order/in").unmarshal(df).to("seda:queue:neworder");
```

上記のサンプルは、**order/in** フォルダーからファイルを読み取り、ファイルの構造を設定する Flatpack 設定ファイル **INVENTORY-Delimited.pzmap.xml** を使用して入力を非整列化します。結果は、SEDA キューに格納する **DataSetList** オブジェクトです。

```
FlatpackDataFormat df = new FlatpackDataFormat();
df.setDefinition(new ClassPathResource("PEOPLE-FixedLength.pzmap.xml"));
df.setFixed(true);
df.setIgnoreFirstRecord(false);

from("seda:people").marshal(df).convertBodyTo(String.class).to("jms:queue:people");
```

上記のコードでは、Object 表現からのデータを、行の **List** として **Maps** としてマーシャリングします。**Map** としての行には、キーとしての列名と、対応する値が含まれます。この構造は、プロセッサーなどから Java コードで作成できます。Flatpack 形式に従ってデータをマーシャリングし、結果を **String** オブジェクトとして変換して JMS キューに格納します。

103.3. 依存関係

camel ルートで Flatpack を使用するには、このデータ形式を実装する **camel-flatpack** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
</dependency>
```

第104章 APACHE FLINK コンポーネント

Camel バージョン 2.18 以降で利用可能

このドキュメントページでは、Apache Camel の Apache Flink コンポーネントについて説明します。**camel-flink** コンポーネントは、Camel コネクターと Flink タスクの間のブリッジを提供します。この Camel Flink コネクターは、さまざまなトランスポートからメッセージをルーティングし、実行する flink タスクを動的に選択し、着信メッセージをタスクの入力データとして使用し、最終的に結果を Camel パイプラインに戻す方法を提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flink</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

104.1. URI 形式

現在、Flink コンポーネントはプロデューサーのみをサポートしています。DataSet、DataStream ジョブを作成できます。

```
flink:dataset?dataset=#myDataSet&dataSetCallback=#dataSetCallback
flink:datastream?datastream=#myDataStream&dataStreamCallback=#dataStreamCallback
```

FlinkEndpoint オプション

Apache Flink エンドポイントは、URI 構文を使用して設定されます。

```
flink:endpointType
```

パスおよびクエリーパラメーターを使用します。

104.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
endpointType	**必須** エンドポイントのタイプ (データセット、データストリーム)。		EndpointType

104.1.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
collect (producer)	結果を収集またはカウントする必要があるかどうかを示します。	true	boolean

名前	説明	デフォルト	タイプ
dataSet (producer)	計算対象の DataSet。		DataSet
dataSetCallback (producer)	DataSet に対してアクションを実行する関数。		DataSetCallback
dataStream (producer)	計算対象の DataStream。		DataStream
dataStreamCallback (producer)	DataStream に対してアクションを実行する関数。		DataStreamCallback
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

104.2. FLINKCOMPONENT オプション

Apache Flink コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
dataSet (producer)	計算対象の DataSet。		DataSet
dataStream (producer)	計算対象の DataStream。		DataStream
dataSetCallback (producer)	DataSet に対してアクションを実行する関数。		DataSetCallback
dataStreamCallback (producer)	DataStream に対してアクションを実行する関数。		DataStreamCallback
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

104.3. FLINK DATASET コールバック

```
@Bean
public DataSetCallback<Long> dataSetCallback() {
    return new DataSetCallback<Long>() {
```

```
        public Long onDataSet(DataSet dataSet, Object... objects) {
            try {
                dataSet.print();
                return new Long(0);
            } catch (Exception e) {
                return new Long(-1);
            }
        }
    };
}
```

104.4. FLINK DATASTREAM CALLBACK

```
@Bean
public VoidDataStreamCallback dataStreamCallback() {
    return new VoidDataStreamCallback() {
        @Override
        public void doOnDataStream(DataStream dataStream, Object... objects) throws Exception {
            dataStream.flatMap(new Splitter()).print();

            environment.execute("data stream test");
        }
    };
}
```

104.5. CAMEL-FLINK プロデューサー呼び出し

```
CamelContext camelContext = new SpringCamelContext(context);

String pattern = "foo";

try {
    ProducerTemplate template = camelContext.createProducerTemplate();
    camelContext.start();
    Long count = template.requestBody("flink:dataSet?
dataSet=#myDataSet&dataSetCallback=#countLinesContaining", pattern, Long.class);
    } finally {
        camelContext.stop();
    }
```

104.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第105章 FOP コンポーネント

Camel バージョン 2.10 以降で利用可能

FOP コンポーネントを使用すると、Apache FOP を使用してメッセージをさまざまな出力形式にレンダリングできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-fop</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

105.1. URI 形式

```
fop://outputFormat?[options]
```

105.2. 出力形式

主な出力形式は PDF ですが、他の出力 形式 もサポートされています。

name	output Format	description
PDF	application/pdf	ポータブルドキュメントフォーマット
PS	application/postscript	Adobe Postscript
PCL	application/x-pcl	プリンター制御言語
PNG	image/png	PNG イメージ
JPEG	image/jpeg	JPEG イメージ
SVG	image/svg+xml	スケーラブルなベクターグラフィックス

name	outputFormat	description
XML	application/X-fop-areatree	エリアツリーの表現
MIF	application/mif	FrameMaker の MIF
RTF	application/rtf	リッチテキスト形式
TXT	text/plain	テキスト

有効な出力形式の完全なリストは、ここ にあります。

105.3. エンドポイントオプション

FOP コンポーネントにはオプションがありません。

FOP エンドポイントは、URI 構文を使用して設定されます。

```
fop:outputType
```

パスおよびクエリーパラメーターを使用します。

105.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
outputType	**必須** 主な出力形式は PDF ですが、他の出力形式もサポートされています。		FopOutputType

105.3.2. クエリーパラメーター(3 個のパラメーター):

名前	説明	デフォルト	タイプ
fopFactory (producer)	org.apache.fop.apps.FopFactory のカスタム設定または実装の使用を許可します。		FopFactory

名前	説明	デフォルト	タイプ
userConfigURL (producer)	クラスパスまたはファイルシステムからロードできる設定ファイルのロケーション。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

次の 構造 を持つ設定ファイルのロケーション。**Camel 2.12** 以降、ファイルはデフォルトでクラスパスからロードされます。**file:** または **classpath:** を接頭辞として使用して、ファイルまたはクラスパスからリソースをロードできます。以前のリリースでは、ファイルは常にファイルシステムからロードされていました。

fopFactory

org.apache.fop.apps.FopFactory のカスタム設定または実装の使用を許可します。

105.4. メッセージ操作

name	デフォルト値	description
CamelFop.Output.Format		そのメッセージの出力形式をオーバーライドします
CamelFop.Encrypt.userPassword		PDF ユーザーパスワード
CamelFop.Encrypt.ownerPassword		PDF 所有者パスワード
CamelFop.Encrypt.allowPrint	true	PDF の印刷を許可します

name	デフォルト値	description
CamelFop.Encrypt.allowCopyContent	true	PDF のコンテンツをコピーできます
CamelFop.Encrypt.allowEditContent	true	PDF のコンテンツを編集できます
CamelFop.Encrypt.allowEditAnnotations	true	PDF の注釈を編集できます
CamelFop.Render.producer	Apache FOP	ドキュメントを生成するシステム/ソフトウェアのメタデータ要素
CamelFop.Render.creator		ドキュメントを作成したユーザーのメタデータ要素
CamelFop.Render.creationDate		Creation Date
CamelFop.Render.author		ドキュメントのコンテンツの作成者
CamelFop.Render.title		ドキュメントのタイトル

name	デフォルト値	description
CamelFop.Render.subject		文書の件名
CamelFop.Render.keywords		このドキュメントに適用可能なキーワードのセット

105.5. 例

以下は、xml データと xslt テンプレートから PDF をレンダリングし、PDF ファイルをターゲットフォルダーに保存するルートの例です。

```
from("file:source/data/xml")
    .to("xslt:xslt/template.xsl")
    .to("fop:application/pdf")
    .to("file:target/data");
```

詳細については、これらのリソースを参照してください...

105.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第106章 FREEMARKER コンポーネント

Camel バージョン 2.10 以降で利用可能

freemarker: コンポーネントを使用すると、FreeMarker テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-freemarker</artifactId>
  <version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

106.1. URI 形式

```
freemarker:templateName[?options]
```

templateName は呼び出すテンプレートのクラスパスローカル URI です。またはリモートテンプレートの完全な URL (例: file://folder/myfile.ftl)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

106.2. オプション

Freemarker コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	既存の freemarker.template.Configuration インスタンスを設定として使用する場合。		設定
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Freemarker エンドポイントは、URI 構文を使用して設定されます。

```
freemarker:resourceUri
```

パスおよびクエリーパラメーターを使用します。

106.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

106.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
configuration (producer)	使用する Freemarker 設定を設定します		設定
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding (producer)	テンプレートファイルの読み込みに使用するエンコーディングを設定します。		String
templateUpdateDelay (producer)	読み込まれたテンプレートリソースがキャッシュに残る秒数。		int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

106.3. ヘッダー

FreeMarker の評価中に設定されたヘッダーはメッセージに返され、ヘッダーとして追加されます。これにより、FreeMarker コンポーネントが Message に値を返すメカニズムが提供されます。

例: FreeMarker テンプレートで **fruit** のヘッダー値を設定します。

```
${request.setHeader('fruit', 'Apple')}
```

ヘッダーの **fruit は、message.out.headers** からアクセスできるようになりました。

106.4. FREEMARKER コンテキスト

Camel は FreeMarker コンテキスト (単なる **Map**) で交換情報を提供します。**Exchange** は次のように転送されます。

key	value
exchange	**Exchange** 自体。
exchange.properties	**Exchange** プロパティー。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト
request	IN メッセージ
body	In メッセージボディー
response	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。

Camel 2.14 から、このようにキー "**CamelFreemarkerDataModel**" を使用して、メッセージヘッダーにカスタム FreeMarker コンテキストをセットアップできます。

```
Map<String, Object> variableMap = new HashMap<String, Object>();
variableMap.put("headers", headersMap);
variableMap.put("body", "Monday");
variableMap.put("exchange", exchange);
exchange.getIn().setHeader("CamelFreemarkerDataModel", variableMap);
```

106.5. ホットリロード

FreeMarker テンプレートリソースは、デフォルトでは、ファイルリソースとクラスパスリソース (拡張 jar) の両方でホットリロード **できません**。**contentCache=false** を設定すると、Camel はリソースをキャッシュしないため、ホットリロードが有効になります。このシナリオは、開発で使用できます。

106.6. 動的テンプレート

Camel は、テンプレートまたはテンプレートコンテンツ自体の異なるリソースのロケーションを定義できる 2 つのヘッダーを提供します。これらのヘッダーのいずれかが設定されている場合、Camel はエンドポイントで設定されたリソースに対してこれを使用します。これにより、実行時に動的なテンプレートを提供できます。

ヘッダー	タイプ	説明	サポートバージョン
FreemarkerConstants.FREEMARKER_RESOURCE	org.springframework.core.io.Resource	テンプレートリソース	⇐ 2.1

ヘッダー	タイプ	説明	サポートバージョン
FreemarkerConstants.FREEMARKER_RESOURCE_URI	String	設定されたエンドポイントの代わりに使用するテンプレートリソースのURI。	>= 2.1
FreemarkerConstants.FREEMARKER_TEMPLATE	String	設定されたエンドポイントの代わりに使用するテンプレート。	>= 2.1

106.7. サンプル

たとえば、次のようなものを使用できます。

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl");
```

FreeMarker テンプレートを使用して、InOut メッセージ交換 (**JMSReplyTo** ヘッダーがある場合) のメッセージに対する応答を作成します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、次を使用できます。

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl").
  to("activemq:Another.Queue");
```

また、**.ftl** テンプレートをホットリロードする必要がある開発用途などで、コンテンツキャッシュを無効にするには:

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl?contentCache=false").
  to("activemq:Another.Queue");
```

そしてファイルベースのリソース:

```
from("activemq:My.Queue").
  to("freemarker:file://myfolder/MyResponse.ftl?contentCache=false").
  to("activemq:Another.Queue");
```

Camel 2.1 では、コンポーネントがヘッダーを介して動的に使用するテンプレートを指定できます。たとえば、次のようになります。

```
from("direct:in").

setHeader(FreemarkerConstants.FREEMARKER_RESOURCE_URI).constant("path/to/my/template.ftl").
  to("freemarker:dummy");
```

106.8. 電子メールのサンプル

このサンプルでは、注文確認メールに FreeMarker テンプレートを使用します。電子メールテンプレートは、FreeMarker で次のようにレイアウトされます。

```
Dear ${headers.lastName}, ${headers.firstName}

Thanks for the order of ${headers.item}.

Regards Camel Riders Bookstore
${body}
```

そして Java コード:

106.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第107章 FTP コンポーネント

Camel バージョン 1.1 以降で利用可能

このコンポーネントは、FTP および SFTP プロトコルを介したリモートファイルシステムへのアクセスを提供します。

リモート FTP サーバーから使用する場合は、ファイルの使用に関する詳細について、さらに下の **ファイルを使用する場合のデフォルト** というタイトルのセクションを必ずお読みください。

絶対パスはサポートされて **いません**。Camel 2.16 は、**directoryname** から先頭のスラッシュをすべて削除することにより、絶対パスを相対パスに変換します。ログに WARN メッセージが出力されます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>See the documentation of the Apache Commons
  <!-- use the same version as your Camel core version -->
</dependency>
```

107.1. URI 形式

```
ftp://[username@]hostname[:port]/directoryname[?options]
sftp://[username@]hostname[:port]/directoryname[?options]
ftps://[username@]hostname[:port]/directoryname[?options]
```

directoryname は、基礎となるディレクトリーを表します。ディレクトリー名は相対パスです。絶対パスはサポートされて **いません**。相対パスには、/inbox/us などのネストされたフォルダーを含めることができます。

Camel 2.16 より前のバージョンの Camel の場合、このコンポーネントは **autoCreate** オプション (ファイルコンポーネントがサポートする) をサポートしていないため、directoryName はすでに存在している **必要があります**。その理由は、FTP 管理者 (FTP サーバー) のタスクが適切にユーザーアカウントを設定し、適切なファイルアクセス許可を持つホームディレクトリーなどを設定するためです。

Camel 2.16 では、**autoCreate** オプションがサポートされています。コンシューマーが開始すると、ポーリングがスケジュールされる前に、エンドポイント用に設定されたディレクトリーを作成するために追加の FTP 操作が実行されます。**autoCreate** のデフォルト値は **true** です。

ユーザー名 が指定されていない場合、パスワードを使用せずに **匿名** ログインが試行されます。
ポート 番号が指定されていない場合、Camel はプロトコル (ftp = 21、sftp = 22、ftps = 2222) に従ってデフォルト値を提供します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

このコンポーネントは、実際の FTP 作業に 2 つの異なるライブラリーを使用します。FTP と FTPS は Apache Commons Net を使用し、SFTP は JCraft JSCH を使用します。

FTPS コンポーネントは、Camel 2.2 以降でのみ使用できます。
FTPS (FTP セキュアとも呼ばれる) は、Transport Layer Security (TLS) および Secure Sockets Layer (SSL) 暗号化プロトコルのサポートを追加する FTP の拡張機能です。

107.2. URI オプション

以下のオプションは、FTP コンポーネント専用です。

FTP コンポーネントにはオプションがありません。

MLLP エンドポイントは、URI 構文を使用して設定されます。

```
ftp:host:port/directoryName
```

パスおよびクエリーパラメーターを使用します。

107.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** FTP サーバーのホスト名		String
port	FTP サーバーのポート		int
directoryName	開始ディレクトリー		文字列

107.2.2. クエリーパラメーター (108 パラメーター)

名前	説明	デフォルト	タイプ
binary (Common)	ファイル転送モードを BINARY または ASCII で指定します。デフォルトは ASCII (false) です。	false	boolean
charset (Common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。コンシューマーでこれを使用して、ファイルのエンコーディングを指定できます。これにより、Camel は、ファイルコンテンツがアクセスされている場合にファイルコンテンツをロードする必要がある charset を知ることができます。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む charset を指定できます。ファイルを書き込むとき、Camel はメッセージの内容をメモリーに読み込んで、データを設定された charset に変換できるようにする必要があることに注意してください。つまり、メッセージが大きい場合は、これを使用しないでください。		String
disconnect (Common)	使用直後にリモート FTP サーバーから切断するかどうか。切断は、FTP サーバーへの現在の接続のみを切断します。停止したいコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean

名前	説明	デフォルト	タイプ
doneFileName (Common)	Producer: 指定された場合、元のファイルが書き込まれると、Camel は 2 番目の完了ファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的プレースホルダーを使用することもできます。完了ファイルは、常に元のファイルと同じフォルダーに書き込まれます。Consumer: 指定すると、Camel は完了ファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的なプレースホルダーを使用できます。完了ファイルは、常に元のファイルと同じフォルダーにあると想定されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		String
fileName (Common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込むファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。(注: ヘッダー自体を式にすることもできます)。式オプションは String タイプと Expression タイプの両方をサポートします。式が String タイプである場合、これは常にファイル言語を使用して評価されます。式が Expression タイプである場合、指定された Expression タイプが使用されます。これにより、たとえば OGNL 式を使用できます。コンシューマーの場合、これを使用してファイル名をフィルターリングできるため、たとえば、ファイル言語構文 mydata-$date:now:yyyyMMdd.txt を使用して今日のファイルを消費できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は一度だけ使用されるヘッダーであり、CamelFileName を一時的に保存して後で復元する必要がなくなるため、簡単になります。		String
passiveMode (Common)	パッシブモード接続の設定デフォルトはアクティブモード接続です。	false	boolean
separator (common)	使用するパス区切りを設定します。UNIX = UNIX スタイルのパス区切りを使用 Windows = Windows スタイルのパス区切りを使用 Auto = (デフォルト) ファイル名に既存のパス区切りを使用します	UNIX	PathSeparator

名前	説明	デフォルト	タイプ
transferLoggingInterval Seconds (Common)	進行中のアップロードおよびダウンロード操作の進行状況をログに記録するときに使用する間隔を秒単位で設定します。これは、操作に時間がかかる場合に進行状況を記録するために使用されます。	5	int
transferLoggingLevel (Common)	アップロードおよびダウンロード操作の進行状況をログに記録するときに使用するログレベルを設定します。	DEBUG	LoggingLevel
transferLoggingVerbose (Common)	がアップロードおよびダウンロード操作の進行状況の詳細な (詳細な) ログを実行するかどうかを設定します。	false	boolean
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用して、ファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示することをサポートしていない可能性があるため、オプションが false の場合、camel-ftp は古い方法を使用してディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。このオプションは、readLock=changed にも影響を与え、ファイル情報を更新するための高速チェックを実行するかどうかを制御します。これは、FTP サーバーに多くのファイルがある場合にプロセスを高速化するために使用できます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delete (consumer)	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて move failure 式を設定します。たとえば、ファイルを .error サブディレクトリーに移動するには、.error を使用します。注: ファイルを失敗したロケーションに移動すると、Camel はエラーを処理し、ファイルを再度取得しません。		String

名前	説明	デフォルト	タイプ
noop (consumer)	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel は idempotent=true も設定し、同じファイルを繰り返し消費しないようにします。	false	boolean
preMove (consumer)	処理前に移動する場合にファイル名を動的に設定するために使用される式 (File 言語など)。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		String
preSort (consumer)	pre-sort が有効になっている場合、コンシューマーはポーリング中に、ファイルシステムから取得されたファイル名とディレクトリー名を並べ替えます。ソートされた順序でファイルを操作する必要がある場合に、これを行うことができます。pre-sort は、コンシューマーがフィルターリングを開始する前に実行され、Camel によって処理されるファイルを受け入れます。このオプション default=false で無効になっています。	false	boolean
recursive (consumer)	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
resumeDownload (consumer)	ダウンロードの再開を有効にするかどうかを設定します。これは、FTP サーバーでサポートされている必要があります (ほとんどすべての FTP サーバーがサポートしています)。さらに、オプション localWorkDirectory を設定して、ダウンロードしたファイルがローカルディレクトリーに保存されるようにし、オプションバイナリーを有効にする必要があります。これは、ダウンロードの再開をサポートするために必要です。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streamDownload (consumer)	ローカル作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取られるときにルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーにロードされます。	false	boolean

名前	説明	デフォルト	タイプ
directoryMustExist (consumer)	startingDirectoryMustExist に似ていますが、これは再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードする必要があるかどうか。このオプションが false に設定されている場合、メッセージ本文は null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルに関する詳細を含む Camel Exchange を引き続きトリガーします。ファイルがダウンロードされないだけです。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handleDirectoryParserAbsoluteResult (consumer)	ディレクトリーパーサーの結果が絶対パスである場合に、コンシューマーがパス内のサブフォルダーとファイルを処理する方法を設定できます。この理由は、一部の FTP サーバーが絶対パスでファイル名を返す場合があるためです。その場合、FTP コンポーネントは返されたパスを相対パスに変換することでこれを処理します。	false	boolean
ignoreFileNotFoundOrPermissionError (consumer)	(ディレクトリー内のファイルを一覧表示しようとするとき、またはファイルをダウンロードするとき)、存在しない場合、またはアクセス許可エラーが原因である場合に無視するかどうか。デフォルトでは、ディレクトリーまたはファイルが存在しないか、権限が不十分な場合、例外が出力されます。このオプションを true に設定すると、代わりにそれを無視できます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。in-progress リポジトリーは、現在進行中のファイルが消費されていることを示すために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>

名前	説明	デフォルト	タイプ
localWorkDirectory (consumer)	使用する場合、ローカルの作業ディレクトリーを使用して、リモートファイルのコンテンツをローカルファイルに直接保存し、コンテンツがメモリーに読み込まれないようにできます。これは、非常に大きなリモートファイルを使用している場合に、メモリーを節約するために役立ちます。		String
onCompletionException Handler (consumer)	カスタム org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを実行する完了プロセスのファイル中に出力される例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視します。		ExceptionHandler
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。特別な準備完了ファイルが存在するなど、ファイルを使用する前に特別な条件を満たす必要がある場合にも使用できます。このオプションを設定すると、readLock オプションは適用されません。		GenericFileProcess s Strategy<T>
receiveBufferSize (consumer)	受信 (ダウンロード) バッファーサイズ FTPClient のみが使用	32768	int
startingDirectory MustExist (consumer)	開始ディレクトリーの存在が必要かどうか。autoCreate オプションがデフォルトで有効になっていることに注意してください。これは、開始ディレクトリーが存在しない場合、通常は自動作成されることを意味します。autoCreate を無効にして有効にすると、開始ディレクトリーの存在が必要なことを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean

名前	説明	デフォルト	タイプ
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。場合によっては、特定のファイルをダウンロードする必要があり、LIST コマンドの使用が許可されていない場合があるため、このオプションを false に設定できます。このオプションを使用する場合、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、権限などのメタデータ情報が含まれないことに注意してください。これらの情報は、LIST コマンドを使用している場合にのみ取得できるためです。	true	boolean
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。Override: これがデフォルトで、既存のファイルを置き換えます。append: 既存ファイルにコンテンツを追加します。fail: GenericFileOperationException を出力し、既存ファイルがあることを示します。ignore: 問題を警告なしで無視して既存のファイルは上書きしませんが、問題は発生していないと想定します。move: オプションを設定するには、moveExisting オプションも使用する必要があります。オプション eagerDeleteTargetFile を使用して、ファイルを移動する際に既存ファイルが存在する場合に何をすべきか制御でき、そうでない場合は移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用できます。これにより、存在チェックを実行せずに、一時的なファイル名から実際のファイル名への変更を試みることができます。このチェックは、一部のファイルシステム、特に FTP サーバーでは高速になる場合があります。	オーバーライド	GenericFileExist
flatten (producer)	flatten は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に使用できますが、たとえばファイルを別のディレクトリーに書き込む場合、ファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が先頭パスから削除されます。	false	boolean

名前	説明	デフォルト	タイプ
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式 (File 言語など)。ファイルをバックアップサブディレクトリーに移動するには、backup と入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントでは file:parent がサポートされていないことに注意してください。FTP コンポーネントは、既存のファイルを現在のディレクトリーをベースとする相対ディレクトリーにしか移動できないためです。		String
tempFileName (producer)	tempPrefix オプションと同じですが、ファイル言語を使用するため、一時ファイル名の命名をより細かく制御できます。		String
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込み、書き込みが完了した後に、その名前を実際の名前に変更するために使用されます。書き込み中のファイルを識別し、(排他的読み取りロックを使用せずに) コンシューマーが進行中のファイルを読み取らないようにするために使用できます。大きなファイルをアップロードするときに FTP でよく使用されます。		String
allowNullBody (producer)	ファイルの書き込み中に null の本文を許可するかどうかを指定するために使用されます。true に設定すると空のファイルが作成され、false に設定して null の本文をファイルコンポーネントに送信しようとすると、Cannot write null body to file.という GenericFileWriteException が出力されます。fileExist オプションを Override に設定するとファイルは切り捨てられ、append に設定するとファイルは変更されません。	false	boolean
chmod (producer)	保存されたファイルに chmod を設定できます。たとえば、chmod=640 です。		String
disconnectOnBatchComplete (producer)	バッチアップロードが完了した直後にリモート FTP サーバーから切断するかどうか。disconnectOnBatchComplete は、FTP サーバーへの現在の接続のみを切断します。	false	boolean

名前	説明	デフォルト	タイプ
eagerDeleteTargetFile (producer)	既存のターゲットファイルを先行して削除するかどうか。このオプションは、fileExists=Override および tempFileName オプションを使用している場合にのみ適用されます。これを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除することを無効化 (false に設定) できます。たとえば、大きなファイルを書き込んで、一時ファイルの書き込み中にターゲットファイルを存在させたい場合があります。これにより、一時ファイルの名前がターゲットファイル名に変更される直前まで、ターゲットファイルは削除されません。このオプションは、fileExist=Move が有効で、既存のファイルが存在する場合に、既存のファイルを削除するかどうかを制御するためにも使用されます。このオプション copyAndDeleteOnRenameFails が false の場合、既存のファイルが存在する場合は例外が出力されます。true の場合、移動操作の前に既存のファイルが削除されます。	true	boolean
keepLastModified (producer)	ソースファイル (存在する場合) からの最終変更のタイムスタンプを保持します。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには、java.util.Date またはタイムスタンプ付きの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプが設定されます。注記: このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	ファイルを FTP サーバーにアップロードする前に書き込み前チェックとして noop コマンドを送信するかどうか。接続の検証がまだ有効であるため、これはデフォルトで有効になっています。これにより、サイレントに再接続してファイルをアップロードできるようになります。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
activePortRange (advanced)	アクティブモードでクライアント側のポート範囲を設定します。構文は minPort-maxPort です。両方のポート番号が含まれます。たとえば、すべての 1xxxx ポートを含めるには 10000-19999 です。		String
autoCreate (advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean

名前	説明	デフォルト	タイプ
bufferSize (advanced)	書き込みバッファーのサイズ (バイト単位)。	131072	int
connectTimeout (advanced)	接続が確立されるのを待つための接続タイムアウトを設定します。FTPClient と JSCH の両方で使用されます	10000	int
ftpClient (advanced)	FTPClient のカスタムインスタンスを使用します		FTPClient
ftpClientConfig (advanced)	FTPClientConfig のカスタムインスタンスを使用して、エンドポイントが使用する FTP クライアントを設定するには。		FTPClientConfig
ftpClientConfigParameters (advanced)	FtpClientConfig. に追加パラメーターを提供するために FtpComponent によって使用されます		Map
ftpClientParameters (advanced)	FTPClient に追加のパラメーターを提供するために FtpComponent によって使用されます		Map
maximumReconnectAttempts (advanced)	Camel がリモート FTP サーバーへの接続を試行するときに実行する再接続の最大試行回数を指定します。この動作を無効にするには、0 を使用します。		int
reconnectDelay (advanced)	ミリ秒単位の遅延 Camel は、再接続試行を実行する前に待機します。		long
siteCommand (advanced)	ログインの成功後に実行されるオプションのサイトコマンドを設定します。複数のサイトコマンドは、改行文字を使用して区切ることができます。		String
soTimeout (advanced)	SO タイムアウトを設定します。FTPClient によってのみ使用されます。	300000	int
stepwise (advanced)	ファイルをダウンロードするとき、またはファイルをディレクトリーにアップロードするときに、ファイル構造をトラバースしながらディレクトリーを段階的に変更するかどうかを設定します。たとえば、セキュリティー上の理由で FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
throwExceptionOnConnect Failed (advanced)	接続が失敗した (使い果たされた) 場合に例外を出力する必要があります。デフォルトでは、例外は出力されず、WARN がログに記録されます。これを使用して、例外の出力を有効にし、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから出力された例外を処理できます。	false	boolean
timeout (advanced)	応答を待つためのデータタイムアウトを設定します。FTPClient だけが使用します	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。コンマ区切り形式で複数の除外を指定できます。		String
antFilterCaseSensitive (filter)	ant フィルターに大文字と小文字を区別するフラグを設定します	true	boolean
antInclude (filter)	Ant スタイルフィルターの組み込み。コンマ区切り形式で複数の組み込みを指定できます。		String
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が eager かどうかを制御できます。eager の場合、ファイルのスキャン中に制限されます。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートしてからポーリングを制限できます。ソートのためにすべてのファイルの詳細がメモリー内にあるため、メモリー使用量が大きくなることに注意してください。	true	boolean
exclude (filter)	ファイル名が正規表現パターンに一致する場合にファイルを除外するために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
filter (filter)	org.apache.camel.component.file.GenericFileFilter クラスとしてのプラグ可能なフィルター。フィルターがその accept () メソッドで false を返す場合、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルターリングします。たとえば、現在の日付でフィルターリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		String

名前	説明	デフォルト	タイプ
filterFile (filter)	Simple 言語に基づいてファイルをフィルターリングします。たとえば、ファイルサイズでフィルターリングするには、$file:size 5000 を使用できます。		String
idempotent (filter)	Camel が既に処理されたファイルをスキップできるように、Idempotent Consumer EIP パターンを使用するオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	Boolean
idempotentKey (filter)	カスタムのべき等性キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用できます。たとえば、ファイル名とファイルサイズを使用するには idempotentKey=$file:name-$file:size となります。		String
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、プラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository はデフォルトで MemoryMessageIdRepository を使用します。		String>
include (filter)	ファイル名が正規表現パターンに一致する場合にファイルを含めるために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
maxDepth (filter)	ディレクトリーを再帰的に処理する際にトラバースする最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されていません。たとえば制限を 1000 などに設定して、数千のファイルがあるサーバーの起動を回避できます。無効にするには、0 または負の値を設定します。注記: このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 個のファイルがある場合に maxMessagesPerPoll=500 を使用すると、最初の 500 個のファイルのみ選択され、ソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int

名前	説明	デフォルト	タイプ
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 はベースディレクトリーを意味します。minDepth=2 は最初のサブディレクトリーを意味します。		int
move (filter)	処理後に移動する場合にファイル名を動的に設定するために使用される式 (Simple 言語など)。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		String
exclusiveReadLockStrategy (lock)	org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装としてのプラグ可能な読み取りロック。		GenericFileExclusiveReadLockStrategy<T>

名前	説明	デフォルト	タイプ
readLock (lock)	ファイルに排他的な読み取りロックがある (つまり、ファイルが進行中または書き込み中ではない) 場合にのみファイルをポーリングするために、コンシューマーが使用します。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none: 読み取りロックは使用されていません。markerFile: Camel はマーカーファイル (fileName.camelLock) を作成してロックを保持します。このオプションは、FTP コンポーネント changed では使用できません。changed は、ファイルの長さ/変更のタイムスタンプを使用して、ファイルが現在コピーされているかどうかを検出します。この判断には 1 秒以上かかるため、このオプションは他のオプションほど速くファイルを消費できませんが、JDK IO API はファイルが別のプロセスで使用中かどうか判断できないため、信頼性が高くなります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。ファイルシステムが分散ファイルロックをサポートしていない限り、マウント/共有によりリモートファイルシステムにアクセスする場合、このアプローチは避ける必要があります。rename: 排他的な読み取りロックを取得できるかどうかのテストとしてファイル名の変更を試みるために使用します。idempotent: (ファイルコンポーネントのみ) 読み取りロックとして idempotentRepository を使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-changed: (ファイルコンポーネントのみ) idempotentRepository および changed を結合された read-lock として使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-rename: (ファイルコンポーネントのみ) idempotentRepository および rename を結合された読み取りロックとして使用するためのものです。これにより、べき等性リポジトリーの実装がサポートしている場合、クラスターリングをサポートする読み取りロックを使用できます。注記: さまざまな読み取りロックは、異なるノード上の同時コンシューマーが共有ファイルシステム上の同じファイルを求めて競合するクラスターモードでの動作にすべて適しているわけではありません。アトミックに近い操作を使用して空のマーカーファイルを作成する markerFile ですが、クラスターでの動作は保証されていません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等性リ	none	String

名前	説明	デフォルト	タイプ
	ポジトリーが Hazelcast コンポーネントや Infinispan などのクラスターリングに対応している場合、べき等性等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合、読み取りロックの間隔 (ミリ単位)。この間隔は、読み取りロックを取得する試行間のスリープに使用されます。たとえば、changed 読み取りロックを使用する場合、遅い書き込みに対応するために間隔を長く設定できます。デフォルトは 1 秒ですが、プロデューサーによるファイルの書き込みが非常に遅い場合は短すぎる可能性があります。注記: FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	1000	long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされなかった場合 (JVM クラッシュなど)、マーカーファイルを使用した読み取りロックが、ファイルシステムに残っている可能性のある孤立した読み取りロックファイルを起動時に削除する必要があるかどうか。このオプションを false にすると、孤立したロックファイルがあると Camel はそのファイルを取得しようとしなくなります。これは、別のノードが同じ共有ディレクトリーから同時にファイルを読み取っているが原因である可能性もあります。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルを変更できます。たとえば、ログを記録しないように OFF に設定できます。このオプションを適用できる readLock タイプは、changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename のみです。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	changed、rename、exclusive の読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、他のプロセスが同じファイルを取得するのを防ぐために、マーカーファイルも使用されます。このオプションを false に設定すると、この動作をオフにできます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みたくない場合などです。	true	boolean

名前	説明	デフォルト	タイプ
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションは、読み取りロックを取得しようとする前に、ファイルが経過しなければならない最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルに 5 分以上の経過を要求します。これにより、指定された期間以上のファイルの取得を試みるため、changed 読み取りロックが高速化されます。	0	long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小ファイル長を設定できます。デフォルトで Camel はファイルにデータが含まれていると想定するため、デフォルト値は 1 です。このオプションをゼロに設定すると、長さがゼロのファイルを使用できます。	1	long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に成功し、コミットが行われるときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトはファイルは削除されないため、競合状態が発生せず、別のアクティブなノードがファイルを取得しようとする可能性があります。代わりにべき等性リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする場合があります。これにより、競合状態の問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に失敗し、ロールバックが発生するときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが (ファイルがコミットされたかのように) 確認されます。	true	boolean

名前	説明	デフォルト	タイプ
readLockTimeout (lock)	読み取りロックでサポートされている場合、読み取りロックのオプションのタイムアウト (ミリ秒単位)。読み取りロックを許可できず、タイムアウトがトリガーされた場合、Camel はファイルをスキップします。次のポーリングで、Camel はファイルを再試行します。このときに、読み取りロックが許可される可能性があります。無期限を指定するには、0 以下の値を使用します。現在、fileLock、changed、および rename がタイムアウトに対応しています。注記: FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	10000	long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルします (ランダムな順序でのソート)	false	boolean
sortBy (sort)	File 言語を使用したビルトインソート。ネストされたソートをサポートしているため、ファイル名でのソートと、2 つ目のグループとして変更日でソートできます。		String
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
アカウント (security)	ログインに使用するアカウント		String
password (security)	ログインに使用するパスワード		String
username (security)	ログインに使用するユーザー名		文字列

107.3. FTPS コンポーネントのデフォルトの信頼ストア

ftpClient. を使用する場合。トラストストアはすべての証明書を受け入れます。トラスト選択証明書のみが必要な場合は、**ftpClient.trustStore.xxx** オプションを使用するか、カスタム **ftpClient** を設定して、トラストストアを設定する必要があります。

sslContextParameters を使用する場合、トラストストアは提供された SSLContextParameters インスタンスの設定によって管理されます。

ftpClient を使用して、URI から直接 **ftpClient** および **ftpClientConfig** の追加オプションを設定できます。または **ftpClientConfig.** 接頭辞。

たとえば、**FTPClient** の **setDataTimeout** を 30 秒に設定するには、次のようにします。

```
from("ftp://foo@myserver?password=secret&ftpClient.dataTimeout=30000").to("bean:foo");
```

たとえば、日付形式やタイムゾーンを設定するために、両方の接頭辞を組み合わせて使用することができます。

```
from("ftp://foo@myserver?
password=secret&ftpClient.dataTimeout=30000&ftpClientConfig.serverLanguageCode=fr").to("bean:foo");
```

これらのオプションはいくつでも使用できます。

可能なオプションと詳細については、Apache Commons FTP FTPClientConfig のドキュメントを参照してください。 Apache Commons FTP FTPClient についても同様です。

URL に多くの長い設定を含めるのが気に入らない場合は、キャメルがレジストリーで検索できるようにすることで、使用する **ftpClient** または **ftpClientConfig** を参照できます。

以下に例を示します。

```
<bean id="myConfig" class="org.apache.commons.net.ftp.FTPClientConfig">
    <property name="lenientFutureDates" value="true"/>
    <property name="serverLanguageCode" value="fr"/>
</bean>
```

そして、URL で # 表記を使用すると、Camel がこの Bean をルックアップします。

```
from("ftp://foo@myserver?password=secret&ftpClientConfig=#myConfig").to("bean:foo");
```

107.4. 例

ftp://someone@someftpserver.com/public/upload/images/holiday2008?password=secret&binary=true

ftp://someoneelse@someotherftpserver.co.uk:12049/reports/2008/password=secret&binary=false
ftp://publicftpserver.com/download

107.5. 並行処理性

FTP コンシューマーは同時実行をサポートしていません

FTP コンシューマー (同じエンドポイントを持つ) は同時実行をサポートしません (バッキング FTP クライアントはスレッドセーフではありません)。
複数の FTP コンシューマーを使用して、異なるエンドポイントからポーリングできます。同時コンシューマーをサポートしないのは、単一のエンドポイントのみです。

FTP プロデューサーにはこの問題は **なく**、並行性がサポートされています。

107.6. 補足情報

このコンポーネントは、File コンポーネントの拡張です。そのため、ファイルコンポーネントページには、より多くのサンプルと詳細があります。

107.7. ファイルを使用するときのデフォルト

FTP コンシューマーは、デフォルトで、消費されたファイルをリモート FTP サーバーにそのまま残します。ファイルを削除したり、別のロケーションに移動したりする場合は、明示的に設定する必要があります。たとえば、**delete=true** を使用してファイルを削除したり、**move=.done** を使用してファイルを非表示の done サブディレクトリーに移動したりできます。

デフォルトでファイルを **.camel** サブディレクトリーに移動するため、通常のファイルコンシューマーは異なります。Camel が FTP コンシューマーに対してデフォルトでこれを行わ **ない** 理由は、ファイルを移動または削除できる権限がデフォルトで不足している可能性があるためです。

107.7.1. limitations

オプション **readLock** を使用して、Camel が現在書き込み中のファイルを消費し **ない** ようにすることができます。ただし、このオプションは、ユーザーが書き込みアクセス権を持っている必要があるため、デフォルトではオフになっています。読み取りロックの詳細については、File2 のオプションテーブルを参照してください。
現在 FTP 経由で書き込まれているファイルを消費しないようにするための解決策は他にもあります。たとえば、一時的な宛先に書き込み、書き込み後にファイルを移動できます。

move または **preMove** オプションを使用してファイルを移動する場合、ファイルは FTP_ROOT フォルダーに制限されます。これにより、FTP 領域外にファイルを移動できなくなります。ファイルを別の領域に移動する場合は、ソフトリンクを使用してファイルをソフトリンクフォルダーに移動できます。

107.8. メッセージヘッダー

次のメッセージヘッダーを使用して、コンポーネントの動作に影響を与えることができます。

ヘッダー	説明
CamelFileName	エンドポイントへの送信時に出力メッセージに使用される出力ファイル名 (エンドポイントディレクトリーに関連する) を指定します。これが存在せず、式もない場合は、代わりに生成されたメッセージ ID がファイル名として使用されます。
CamelFileNameProduced	書き込まれた出力ファイルの実際のファイルパス (パス + 名前)。このヘッダーは Camel によって設定され、その目的は、書き込まれたファイルの名前をエンドユーザーに提供することです。

ヘッダー	説明
CamelFileIndex	このバッチで消費されているファイルの総数に対する現在のインデックス。
CamelFileSize	このバッチで消費されているファイルの総数。
CamelFileHost	リモートホスト名。
CamelFileLocalWorkPath	ローカル作業ディレクトリーが使用されている場合は、ローカル作業ファイルへのパス。

さらに、FTP/FTPS のコンシューマーとプロデューサーは、次のヘッダーを使用してキャメル **Message** を強化します。

ヘッダー	説明
CamelFtpReplyCode	Camel 2.11.1: FTP クライアントの応答コード (型は整数)
CamelFtpReplyString	Camel 2.11.1: FTP クライアントの応答文字列

107.9. タイムアウトについて

ライブラリーの 2 つのセット (上を参照) には、タイムアウトを設定するための異なる API があります。どちらにも **connectTimeout** オプションを使用して、ミリ秒単位でタイムアウトを設定し、ネットワーク接続を確立できます。個々の **soTimeout** は、FTP/FTPS で設定することもできます。これは、**ftpClient.soTimeout** の使用に対応します。SFTP は自動的に **connectTimeout** を **soTimeout** として使用することに注意してください。**timeout** オプションは、**ftpClient.dataTimeout** 値に対応するデータタイムアウトとして FTP/FTSP にのみ適用されます。すべてのタイムアウト値はミリ単位です。

107.10. ローカル作業ディレクトリーの使用

Camel は、リモート FTP サーバーからの消費と、ローカルの作業ディレクトリーへのファイルの直接ダウンロードをサポートしています。これにより、**FileOutputStream** を使用してローカルファイルに直接ストリーミングされるため、リモートファイルのコンテンツ全体がメモリーに読み込まれるのを回避できます。

Camel はリモートファイルと同じ名前のローカルファイルに保存しますが、ファイルのダウンロード中は拡張子 **.inprogress** が付きます。その後、ファイルの名前が変更され、**.inprogress** 接尾辞が削除されます。最後に、Exchange が完了すると、ローカルファイルが削除されます。

したがって、リモート FTP サーバーからファイルをダウンロードしてファイルとして保存する場合は、次のようなファイルエンドポイントにルーティングする必要があります。

```
from("ftp://someone@someserver.com?password=secret&localWorkDirectory=/tmp").to("file://inbox");
```

ヒント

上記のルートは、ファイルの内容全体をメモリーに読み込まないようにするため、非常に効率的です。リモートファイルをローカルファイルストリームに直接ダウンロードします。次に、**java.io.File** ハンドルが Exchange 本文として使用されます。ファイルプロデューサーはこの事実を利用して、作業ファイルの **java.io.File** ハンドルを直接操作し、ターゲットファイル名に対して **java.io.File.rename** を実行できます。Camel はそれがローカルの作業ファイルであることを認識しているため、作業ファイルはとにかく削除することを意図しているため、最適化してファイルのコピーの代わりに名前の変更を使用できます。

107.11. ディレクトリーを段階的に変更する

Camel FTP は、ファイルの消費時 (ダウンロードなど) またはファイルの生成時 (アップロードなど) にディレクトリーをトラバースするという点で、2 つのモードで動作できます。

- stepwise
- 段階的ではない

状況とセキュリティーの問題に応じて、どちらかを選択することをお勧めします。ステップワイズを使用する場合にのみファイルをダウンロードできる Camel エンドユーザーもいれば、そうでない場合にのみダウンロードできる Camel エンドユーザーもいます。少なくとも、選択する選択肢があります (Camel 2.6 以降)。

Camel 2.0 - 2.5 にはモードが 1 つしかなく、それは次のとおりです。

- 2.5 より前のバージョンは段階的ではありません
- 2.5 は段階的です

Camel 2.6 以降では、動作を制御するために使用できる **stepwise** オプションがあります。

ディレクトリーの段階的な変更は、ほとんどの場合、ユーザーがそのホームディレクトリーに限定されていて、ホームディレクトリーが **"/"** として報告されている場合にのみ機能することに注意してください。

それらの 2 つの違いは、例を使用して最もよく説明されています。ファイルをトラバースしてダウンロードする必要があるリモート FTP サーバーに次のディレクトリー構造があるとします。

```
/
/one
/one/two
/one/two/sub-a
/one/two/sub-b
```

そして、サブ a (a.txt) とサブ b (b.txt) フォルダーのそれぞれにファイルがあることを確認します。

107.11.1. stepwise=true の使用 (デフォルトモード)

```
TYPE A
200 Type set to A
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
```

```
CWD two
250 CWD successful. "/one/two" is current directory.
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,17,94
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,95
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,96
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
PORT 127,0,0,1,17,97
200 Port command successful
RETR foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,98
200 Port command successful
RETR a.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
```

```
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,99
200 Port command successful
RETR b.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
QUIT
221 Goodbye
disconnected.
```

stepwise が有効になるとわかるように、CD xxx を使用してディレクトリー構造をトラバースします。

107.11.2. stepwise=false の使用

```
230 Logged on
TYPE A
200 Type set to A
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,4,122
200 Port command successful
LIST one/two
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,123
200 Port command successful
LIST one/two/sub-a
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,124
200 Port command successful
LIST one/two/sub-b
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,125
200 Port command successful
RETR one/two/foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,126
200 Port command successful
RETR one/two/sub-a/a.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,127
```

```
200 Port command successful
RETR one/two/sub-b/b.txt
150 Opening data channel for file transfer.
226 Transfer OK
QUIT
221 Goodbye
disconnected.
```

stepwise を使用しない場合にわかるように、CD 操作はまったく呼び出されません。

107.12. サンプル

以下のサンプルでは、すべてのレポートを 1 時間 (60 分) ごとに FTP サーバーからバイナリーコンテンツとしてダウンロードし、ローカルファイルシステムにファイルとして保存するように Camel をセットアップします。

そして、Spring DSL を使用したルート:

```
  <route>
    <from uri="ftp://scott@localhost/public/reports?
password=tiger&amp;binary=true&amp;delay=60000"/>
    <to uri="file://target/test-reports"/>
  </route>
```

107.12.1. リモート FTPS サーバー (暗黙的 SSL) とクライアント認証の使用

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&securityProtocol=SSL&isImplicit=true
      &ftpClient.keyStore.file=./src/test/resources/server.jks
      &ftpClient.keyStore.password=password&ftpClient.keyStore.keyPassword=password")
  .to("bean:foo");
```

107.12.2. リモート FTPS サーバー (明示的な TLS) とカスタム信頼ストア設定の使用

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&ftpClient.trustStore.file=./src/test/resources/server.jks&ftpClient.trustStore.password=
password")
  .to("bean:foo");
```

107.13. ORG.APACHE.CAMEL.COMPONENT.FILE.GENERICFILEFILTER を使用するフィルター

Camel は、プラグイン可能なフィルタリング戦略をサポートしています。この戦略は、Java で **org.apache.camel.component.file.GenericFileFilter** のビルドを使用することです。次に、そのようなフィルターを使用してエンドポイントを設定し、処理される前に特定のフィルターをスキップできます。

サンプルでは、ファイル名が report で始まるファイルのみを受け入れる独自のフィルターを作成しました。

そして、**フィルター** 属性を使用してルートを設定し、Spring XML ファイルで定義したフィルターを (**#** 表記を使用して) 参照できます。

```
 <!-- define our sorter as a plain spring bean -->
 <bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="ftp://someuser@someftpserver.com?password=secret&amp;filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

107.14. ANT パスマッチャーを使用したフィルタリング

ANT パスマッチャーは、**camel-spring** jar ですぐに使用できるフィルターです。したがって、Maven を使用している場合は **camel-spring** に依存する必要があります。
その理由は、Spring の AntPathMatcher を利用して実際のマッチングを行うためです。

ファイルパスは、次のルールに一致します。

- **?** 1 文字に一致
- * 0 個以上の文字に一致
- ** パス内の 0 個以上のディレクトリーに一致

以下のサンプルは、その使用方法を示しています。

107.15. SFTP でプロキシーを使用する

HTTP プロキシーを使用してリモートホストに接続するには、次の方法でルートを設定できます。

```
<!-- define our sorter as a plain spring bean -->
<bean id="proxy" class="com.jcraft.jsch.ProxyHTTP">
  <constructor-arg value="localhost"/>
  <constructor-arg value="7777"/>
</bean>

<route>
  <from uri="sftp://localhost:9999/root?username=admin&password=admin&proxy=#proxy"/>
  <to uri="bean:processFile"/>
</route>
```

必要に応じて、ユーザー名とパスワードをプロキシーに割り当てることもできます。すべてのオプションについては、**com.jcraft.jsch.Proxy** のドキュメントを参照してください。

107.16. 優先 SFTP 認証方式の設定

sftp コンポーネントで使用する認証方法のリストを明示的に指定する場合は、**preferredAuthentications** オプションを使用します。たとえば、Camel にプライベート/パブリック SSH キーで認証を試みさせ、公開キーが利用できない場合にユーザー/パスワード認証にフォールバックさせたい場合は、次のルート設定を使用します。

```
from("sftp://localhost:9999/root?
username=admin&password=admin&preferredAuthentications=publickey,password").
  to("bean:processFile");
```

107.17. 固定名を使用して単一のファイルを使用する

単一のファイルをダウンロードする必要があり、ファイル名がわかっている場合は、**fileName=myFileName.txt** を使用して、ダウンロードするファイルの名前を Camel に伝えることができます。デフォルトでは、コンシューマーは引き続き FTP LIST コマンドを実行してディレクトリーのリストを作成し、次に **fileName** オプションに基づいてこれらのファイルをフィルタリングします。ただし、このユースケースでは、**useList=false** を設定してディレクトリーリストをオフにすることが望ましい場合があります。たとえば、FTP サーバーへのログインに使用されるユーザーアカウントには、FTP LIST コマンドを実行する権限がない場合があります。したがって、**useList=false** でこれをオフにしてから、ダウンロードするファイルの固定名を **fileName=myFileName.txt** で指定すると、FTP コンシューマーは引き続きファイルをダウンロードできます。何らかの理由でファイルが存在しない場合、Camel はデフォルトで例外を出力します。これをオフにして、**ignoreFileNotFoundOrPermissionError=true** を設定することでこれを無視できます。

たとえば、単一のファイルを取得し、使用後に削除する Camel ルートを作成するには、次のようにします。

```
from("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true&fileName
=report.txt&delete=true")
  .to("activemq:queue:report");
```

上記で説明したすべてのオプションを使用したことに注意してください。

これは **ConsumerTemplate** でも使用できます。たとえば、単一のファイル (存在する場合) をダウンロードし、ファイルの内容を文字列型として取得するには、次のようにします。

```
String data = template.retrieveBodyNoWait("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true&fileName
=report.txt&delete=true", String.class);
```

107.18. デバッグロギング

このコンポーネントには、問題が発生した場合に役立つログレベル **TRACE** があります。

107.19. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- File2

第108章 FTPS コンポーネント

Camel バージョン 2.2 以降で利用可能

このコンポーネントは、FTP および SFTP プロトコルを介したリモートファイルシステムへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細については、FTP コンポーネント を参照してください。

108.1. URI オプション

以下のオプションは、FTPS コンポーネント専用です。

FTSP コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

FTPS エンドポイントは、URI 構文を使用して設定されます。

```
ftps:host:port/directoryName
```

パスおよびクエリーパラメーターを使用します。

108.1.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** FTP サーバーのホスト名		String
port	FTP サーバーのポート		int

名前	説明	デフォルト	タイプ
directoryName	開始ディレクトリー		文字列

108.1.2. クエリーパラメーター (116 パラメーター)

名前	説明	デフォルト	タイプ
binary (Common)	ファイル転送モードを BINARY または ASCII で指定します。デフォルトは ASCII (false) です。	false	boolean
charset (Common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。コンシューマーでこれを使用して、ファイルのエンコーディングを指定できます。これにより、Camel は、ファイルコンテンツがアクセスされている場合にファイルコンテンツをロードする必要がある charset を知ることができます。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む charset を指定できます。ファイルを書き込むとき、Camel はメッセージの内容をメモリーに読み込んで、データを設定された charset に変換できるようにする必要があることに注意してください。つまり、メッセージが大きい場合は、これを使用しないでください。		String
disconnect (Common)	使用直後にリモート FTP サーバーから切断するかどうか。切断は、FTP サーバーへの現在の接続のみを切断します。停止したいコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
doneFileName (Common)	Producer: 指定された場合、元のファイルが書き込まれると、Camel は 2 番目の完了ファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的プレースホルダーを使用することもできます。完了ファイルは、常に元のファイルと同じフォルダーに書き込まれます。Consumer: 指定すると、Camel は完了ファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的なプレースホルダーを使用できます。完了ファイルは、常に元のファイルと同じフォルダーにあると想定されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		String

名前	説明	デフォルト	タイプ
fileName (Common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込むファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。(注: ヘッダー自体を式にすることもできます)。式オプションは String タイプと Expression タイプの両方をサポートします。式が String タイプである場合、これは常にファイル言語を使用して評価されます。式が Expression タイプである場合、指定された Expression タイプが使用されます。これにより、たとえば OGNL 式を使用できます。コンシューマーの場合、これを使用してファイル名をフィルターリングできるため、たとえば、ファイル言語構文 mydata-$date:now:yyyyMMdd.txt を使用して今日のファイルを消費できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は一度だけ使用されるヘッダーであり、CamelFileName を一時的に保存して後で復元する必要がなくなるため、簡単になります。		String
passiveMode (Common)	パッシブモード接続の設定デフォルトはアクティブモード接続です。	false	boolean
separator (common)	使用するパス区切りを設定します。UNIX = UNIX スタイルのパス区切りを使用 Windows = Windows スタイルのパス区切りを使用 Auto = (デフォルト) ファイル名に既存のパス区切りを使用します	UNIX	PathSeparator
transferLoggingInterval Seconds (Common)	進行中のアップロードおよびダウンロード操作の進行状況をログに記録するときに使用する間隔を秒単位で設定します。これは、操作に時間がかかる場合に進行状況を記録するために使用されます。	5	int
transferLoggingLevel (Common)	アップロードおよびダウンロード操作の進行状況をログに記録するときに使用するログレベルを設定します。	DEBUG	LoggingLevel
transferLoggingVerbose (Common)	がアップロードおよびダウンロード操作の進行状況の詳細な (詳細な) ログを実行するかどうかを設定します。	false	boolean

名前	説明	デフォルト	タイプ
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用して、ファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示することをサポートしていない可能性があるため、オプションが false の場合、camel-ftp は古い方法を使用してディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。このオプションは、readLock=changed にも影響を与え、ファイル情報を更新するための高速チェックを実行するかどうかを制御します。これは、FTP サーバーに多くのファイルがある場合にプロセスを高速化するために使用できます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delete (consumer)	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて move failure 式を設定します。たとえば、ファイルを .error サブディレクトリーに移動するには、.error を使用します。注: ファイルを失敗したロケーションに移動すると、Camel はエラーを処理し、ファイルを再度取得しません。		String
noop (consumer)	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel は idempotent=true も設定し、同じファイルを繰り返し消費しないようにします。	false	boolean
preMove (consumer)	処理前に移動する場合にファイル名を動的に設定するために使用される式 (File 言語など)。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		String

名前	説明	デフォルト	タイプ
preSort (consumer)	pre-sort が有効になっている場合、コンシューマーはポーリング中に、ファイルシステムから取得されたファイル名とディレクトリー名を並べ替えます。ソートされた順序でファイルを操作する必要がある場合に、これを行うことができます。pre-sort は、コンシューマーがフィルターリングを開始する前に実行され、Camel によって処理されるファイルを受け入れます。このオプション default=false で無効になっています。	false	boolean
recursive (consumer)	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
resumeDownload (consumer)	ダウンロードの再開を有効にするかどうかを設定します。これは、FTP サーバーでサポートされている必要があります (ほとんどすべての FTP サーバーがサポートしています)。さらに、オプション localWorkDirectory を設定して、ダウンロードしたファイルがローカルディレクトリーに保存されるようにし、オプションバイナリーを有効にする必要があります。これは、ダウンロードの再開をサポートするために必要です。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streamDownload (consumer)	ローカル作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取られるときにルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーにロードされます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist に似ていますが、これは再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードする必要があるかどうか。このオプションが false に設定されている場合、メッセージ本文は null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルに関する詳細を含む Camel Exchange を引き続きトリガーします。ファイルがダウンロードされないだけです。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handleDirectoryParserAbsoluteResult (consumer)	ディレクトリーパーサーの結果が絶対パスである場合に、コンシューマーがパス内のサブフォルダーとファイルを処理する方法を設定できます。この理由は、一部の FTP サーバーが絶対パスでファイル名を返す場合があるためです。その場合、FTP コンポーネントは返されたパスを相対パスに変換することでこれを処理します。	false	boolean
ignoreFileNotFoundOrPermissionError (consumer)	(ディレクトリー内のファイルを一覧表示しようとするとき、またはファイルをダウンロードするとき)、存在しない場合、またはアクセス許可エラーが原因である場合に無視するかどうか。デフォルトでは、ディレクトリーまたはファイルが存在しないか、権限が不十分な場合、例外が出力されます。このオプションを true に設定すると、代わりにそれを無視できます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。in-progress リポジトリーは、現在進行中のファイルが消費されていることを示すために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>
localWorkDirectory (consumer)	使用する場合、ローカルの作業ディレクトリーを使用して、リモートファイルのコンテンツをローカルファイルに直接保存し、コンテンツがメモリーに読み込まれないようにできます。これは、非常に大きなリモートファイルを使用している場合に、メモリーを節約するために役立ちます。		String
onCompletionException Handler (consumer)	カスタム org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを実行する完了プロセスのファイル中に出力される例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視します。		ExceptionHandler

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。特別な準備完了ファイルが存在するなど、ファイルを使用する前に特別な条件を満たす必要がある場合にも使用できます。このオプションを設定すると、readLock オプションは適用されません。		GenericFileProcess Strategy<T>
receiveBufferSize (consumer)	受信 (ダウンロード) バッファーサイズ FTPClient のみが使用	32768	int
startingDirectoryMustExist (consumer)	開始ディレクトリーの存在が必要かどうか。autoCreate オプションがデフォルトで有効になっていることに注意してください。これは、開始ディレクトリーが存在しない場合、通常は自動作成されることを意味します。autoCreate を無効にして有効にすると、開始ディレクトリーの存在が必要なことを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。場合によっては、特定のファイルをダウンロードする必要があり、LIST コマンドの使用が許可されていない場合があるため、このオプションを false に設定できます。このオプションを使用する場合、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、権限などのメタデータ情報が含まれないことに注意してください。これらの情報は、LIST コマンドを使用している場合にのみ取得できるためです。	true	boolean

名前	説明	デフォルト	タイプ
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。Override: これがデフォルトで、既存のファイルを置き換えます。append: 既存ファイルにコンテンツを追加します。fail: GenericFileOperationException を出力し、既存ファイルがあることを示します。ignore: 問題を警告なしで無視して既存のファイルは上書きしませんが、問題は発生していないと想定します。move: オプションを設定するには、moveExisting オプションも使用する必要があります。オプション eagerDeleteTargetFile を使用して、ファイルを移動する際に既存ファイルが存在する場合に何をすべきか制御でき、そうでない場合は移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用できます。これにより、存在チェックを実行せずに、一時的なファイル名から実際のファイル名への変更を試みることができます。このチェックは、一部のファイルシステム、特に FTP サーバーでは高速になる場合があります。	オーバーライド	GenericFileExist
flatten (producer)	flatten は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に使用できますが、たとえばファイルを別のディレクトリーに書き込む場合、ファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が先頭パスから削除されます。	false	boolean
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式 (File 言語など)。ファイルをバックアップサブディレクトリーに移動するには、backup と入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントでは file:parent がサポートされていないことに注意してください。FTP コンポーネントは、既存のファイルを現在のディレクトリーをベースとする相対ディレクトリーにしか移動できないためです。		String
tempFileName (producer)	tempPrefix オプションと同じですが、ファイル言語を使用するため、一時ファイル名の命名をより細かく制御できます。		String

名前	説明	デフォルト	タイプ
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込み、書き込みが完了した後に、その名前を実際の名前に変更するために使用されます。書き込み中のファイルを識別し、(排他的読み取りロックを使用せずに) コンシューマーが進行中のファイルを読み取らないようにするために使用できます。大きなファイルをアップロードするときに FTP でよく使用されます。		String
allowNullBody (producer)	ファイルの書き込み中に null の本文を許可するかどうかを指定するために使用されます。true に設定すると空のファイルが作成され、false に設定して null の本文をファイルコンポーネントに送信しようとすると、Cannot write null body to file.という GenericFileWriteException が出力されます。fileExist オプションを Override に設定するとファイルは切り捨てられ、append に設定するとファイルは変更されません。	false	boolean
chmod (producer)	保存されたファイルに chmod を設定できます。たとえば、chmod=640 です。		String
disconnectOnBatchComplete (producer)	バッチアップロードが完了した直後にリモート FTP サーバーから切断するかどうか。disconnectOnBatchComplete は、FTP サーバーへの現在の接続のみを切断します。	false	boolean
eagerDeleteTargetFile (producer)	既存のターゲットファイルを先行して削除するかどうか。このオプションは、fileExists=Override および tempFileName オプションを使用している場合にのみ適用されます。これを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除することを無効化 (false に設定) できます。たとえば、大きなファイルを書き込んで、一時ファイルの書き込み中にターゲットファイルを存在させたい場合があります。これにより、一時ファイルの名前がターゲットファイル名に変更される直前まで、ターゲットファイルは削除されません。このオプションは、fileExist=Move が有効で、既存のファイルが存在する場合に、既存のファイルを削除するかどうかを制御するためにも使用されます。このオプション copyAndDeleteOnRenameFails が false の場合、既存のファイルが存在する場合は例外が出力されます。true の場合、移動操作の前に既存のファイルが削除されます。	true	boolean

名前	説明	デフォルト	タイプ
keepLastModified (producer)	ソースファイル (存在する場合) からの最終変更のタイムスタンプを保持します。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには、java.util.Date またはタイムスタンプ付きの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプが設定されます。注記: このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	ファイルを FTP サーバーにアップロードする前に書き込み前チェックとして noop コマンドを送信するかどうか。接続の検証がまだ有効であるため、これはデフォルトで有効になっています。これにより、サイレントに再接続してファイルをアップロードできるようになります。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
activePortRange (advanced)	アクティブモードでクライアント側のポート範囲を設定します。構文は minPort-maxPort です。両方のポート番号が含まれます。たとえば、すべての 1xxxx ポートを含めるには 10000-19999 です。		String
autoCreate (advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	書き込みバッファーのサイズ (バイト単位)。	131072	int
connectTimeout (advanced)	接続が確立されるのを待つための接続タイムアウトを設定します。FTPClient と JSCH の両方で使用されます	10000	int
ftpClient (advanced)	FTPClient のカスタムインスタンスを使用します		FTPClient
ftpClientConfig (advanced)	FTPClientConfig のカスタムインスタンスを使用して、エンドポイントが使用する FTP クライアントを設定するには。		FTPClientConfig

名前	説明	デフォルト	タイプ
ftpClientConfigParameters (advanced)	FtpClientConfig. に追加パラメーターを提供するために FtpComponent によって使用されます		Map
ftpClientParameters (advanced)	FTPClient に追加のパラメーターを提供するために FtpComponent によって使用されます		Map
maximumReconnectAttempts (advanced)	Camel がリモート FTP サーバーへの接続を試行するときに実行する再接続の最大試行回数を指定します。この動作を無効にするには、0 を使用します。		int
reconnectDelay (advanced)	ミリ秒単位の遅延 Camel は、再接続試行を実行する前に待機します。		long
siteCommand (advanced)	ログインの成功後に実行されるオプションのサイトコマンドを設定します。複数のサイトコマンドは、改行文字を使用して区切ることができます。		String
soTimeout (advanced)	SO タイムアウトを設定します。FTPClient によってのみ使用されます。	300000	int
stepwise (advanced)	ファイルをダウンロードするとき、またはファイルをディレクトリーにアップロードするときに、ファイル構造をトラバースしながらディレクトリーを段階的に変更するかどうかを設定します。たとえば、セキュリティー上の理由で FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
throwExceptionOnConnect Failed (advanced)	接続が失敗した (使い果たされた) 場合に例外を出力する必要があります。デフォルトでは、例外は出力されず、WARN がログに記録されます。これを使用して、例外の出力を有効にし、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから出力された例外を処理できます。	false	boolean
timeout (advanced)	応答を待つためのデータタイムアウトを設定します。FTPClient だけが使用します	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。コンマ区切り形式で複数の除外を指定できます。		String

名前	説明	デフォルト	タイプ
antFilterCaseSensitive (filter)	ant フィルターに大文字と小文字を区別するフラグを設定します	true	boolean
antInclude (filter)	Ant スタイルフィルターの組み込み。コンマ区切り形式で複数の組み込みを指定できます。		String
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が eager かどうかを制御できます。eager の場合、ファイルのスキャン中に制限されます。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートしてからポーリングを制限できます。ソートのためにすべてのファイルの詳細がメモリー内にあるため、メモリー使用量が大きくなることに注意してください。	true	boolean
exclude (filter)	ファイル名が正規表現パターンに一致する場合にファイルを除外するために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
filter (filter)	org.apache.camel.component.file.GenericFileFilter クラスとしてのプラグ可能なフィルター。フィルターがその accept () メソッドで false を返す場合、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルターリングします。たとえば、現在の日付でフィルターリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		String
filterFile (filter)	Simple 言語に基づいてファイルをフィルターリングします。たとえば、ファイルサイズでフィルターリングするには、$file:size 5000 を使用できます。		String
idempotent (filter)	Camel が既に処理されたファイルをスキップできるように、Idempotent Consumer EIP パターンを使用するオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	Boolean

名前	説明	デフォルト	タイプ
idempotentKey (filter)	カスタムのべき等性キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用できます。たとえば、ファイル名とファイルサイズを使用するには idempotentKey=$file:name-$file:size となります。		String
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、プラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository はデフォルトで MemoryMessageIdRepository を使用します。		String>
include (filter)	ファイル名が正規表現パターンに一致する場合にファイルを含めるために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
maxDepth (filter)	ディレクトリーを再帰的に処理する際にトラバースする最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されていません。たとえば制限を 1000 などに設定して、数千のファイルがあるサーバーの起動を回避できます。無効にするには、0 または負の値を設定します。注記: このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 個のファイルがある場合に maxMessagesPerPoll=500 を使用すると、最初の 500 個のファイルのみ選択され、ソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 はベースディレクトリーを意味します。minDepth=2 は最初のサブディレクトリーを意味します。		int
move (filter)	処理後に移動する場合にファイル名を動的に設定するために使用される式 (Simple 言語など)。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		String

名前	説明	デフォルト	タイプ
exclusiveReadLockStrategy (lock)	org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装としてのプラグ可能な読み取りロック。		GenericFileExclusiveReadLockStrategy<T>

名前	説明	デフォルト	タイプ
readLock (lock)	ファイルに排他的な読み取りロックがある (つまり、ファイルが進行中または書き込み中ではない) 場合にのみファイルをポーリングするために、コンシューマーが使用します。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none: 読み取りロックは使用されていません。markerFile: Camel はマーカーファイル (fileName.camelLock) を作成してロックを保持します。このオプションは、FTP コンポーネント changed では使用できません。changed は、ファイルの長さ/変更のタイムスタンプを使用して、ファイルが現在コピーされているかどうかを検出します。この判断には 1 秒以上かかるため、このオプションは他のオプションほど速くファイルを消費できませんが、JDK IO API はファイルが別のプロセスで使用中かどうか判断できないため、信頼性が高くなります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。ファイルシステムが分散ファイルロックをサポートしていない限り、マウント/共有によりリモートファイルシステムにアクセスする場合、このアプローチは避ける必要があります。rename: 排他的な読み取りロックを取得できるかどうかのテストとしてファイル名の変更を試みるために使用します。idempotent: (ファイルコンポーネントのみ) 読み取りロックとして idempotentRepository を使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-changed: (ファイルコンポーネントのみ) idempotentRepository および changed を結合された read-lock として使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-rename: (ファイルコンポーネントのみ) idempotentRepository および rename を結合された読み取りロックとして使用するためのものです。これにより、べき等性リポジトリーの実装がサポートしている場合、クラスターリングをサポートする読み取りロックを使用できます。注記: さまざまな読み取りロックは、異なるノード上の同時コンシューマーが共有ファイルシステム上の同じファイルを求めて競合するクラスターモードでの動作にすべて適しているわけではありません。アトミックに近い操作を使用して空のマーカーファイルを作成する markerFile ですが、クラスターでの動作は保証されていません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等性リ	none	String

名前	説明	デフォルト	タイプ
	ポジトリーが Hazelcast コンポーネントや Infinispan などのクラスターリングに対応している場合、べき等性等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合、読み取りロックの間隔 (ミリ単位)。この間隔は、読み取りロックを取得する試行間のスリープに使用されます。たとえば、changed 読み取りロックを使用する場合、遅い書き込みに対応するために間隔を長く設定できます。デフォルトは 1 秒ですが、プロデューサーによるファイルの書き込みが非常に遅い場合は短すぎる可能性があります。注記: FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	1000	long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされなかった場合 (JVM クラッシュなど)、マーカーファイルを使用した読み取りロックが、ファイルシステムに残っている可能性のある孤立した読み取りロックファイルを起動時に削除する必要があるかどうか。このオプションを false にすると、孤立したロックファイルがあると Camel はそのファイルを取得しようとしなくなります。これは、別のノードが同じ共有ディレクトリーから同時にファイルを読み取っているが原因である可能性もあります。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルを変更できます。たとえば、ログを記録しないように OFF に設定できます。このオプションを適用できる readLock タイプは、changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename のみです。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	changed、rename、exclusive の読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、他のプロセスが同じファイルを取得するのを防ぐために、マーカーファイルも使用されます。このオプションを false に設定すると、この動作をオフにできます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みたくない場合などです。	true	boolean

名前	説明	デフォルト	タイプ
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションは、読み取りロックを取得しようとする前に、ファイルが経過しなければならない最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルに 5 分以上の経過を要求します。これにより、指定された期間以上のファイルの取得を試みるため、changed 読み取りロックが高速化されます。	0	long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小ファイル長を設定できます。デフォルトで Camel はファイルにデータが含まれていると想定するため、デフォルト値は 1 です。このオプションをゼロに設定すると、長さがゼロのファイルを使用できます。	1	long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に成功し、コミットが行われるときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトはファイルは削除されないため、競合状態が発生せず、別のアクティブなノードがファイルを取得しようとする可能性があります。代わりにべき等性リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする場合があります。これにより、競合状態の問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に失敗し、ロールバックが発生するときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが (ファイルがコミットされたかのように) 確認されます。	true	boolean

名前	説明	デフォルト	タイプ
readLockTimeout (lock)	読み取りロックでサポートされている場合、読み取りロックのオプションのタイムアウト (ミリ秒単位)。読み取りロックを許可できず、タイムアウトがトリガーされた場合、Camel はファイルをスキップします。次のポーリングで、Camel はファイルを再試行します。このときに、読み取りロックが許可される可能性があります。無期限を指定するには、0 以下の値を使用します。現在、fileLock、changed、および rename がタイムアウトに対応しています。注記: FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	10000	long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

名前	説明	デフォルト	タイプ
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルします (ランダムな順序でのソート)	false	boolean
sortBy (sort)	File 言語を使用したビルトインソート。ネストされたソートをサポートしているため、ファイル名でのソートと、2 つ目のグループとして変更日でソートできます。		String
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
アカウント (security)	ログインに使用するアカウント		String
disableSecureDataChannelDefaults (security)	このオプションを使用して、安全なデータチャネルを使用する場合のデフォルトオプションを無効にします。これにより、使用する execPbsz および execProt 設定を完全に制御できます。デフォルトは false です。	false	boolean
execPbsz (security)	セキュアなデータチャネルを使用する場合、実行保護バッファーサイズを設定できます。		Long

名前	説明	デフォルト	タイプ
execProt (security)	実行保護レベル PROT コマンド。C - クリア S - 安全 (SSL プロトコルのみ) E - 社外秘 (SSL プロトコルのみ) P - プライベート		String
ftpClientKeyStore Parameters (security)	キーストアのパラメーターを設定する		Map
ftpClientTrustStore Parameters (security)	トラストストアパラメーターを設定する		Map
isImplicit (security)	セキュリティーモード (Implicit/Explicit) を設定します。true - 暗黙モード/False - 明示モード	false	boolean
password (security)	ログインに使用するパスワード		String
securityProtocol (security)	基礎となるセキュリティープロトコルを設定します。	TLS	String
sslContextParameters (security)	リンク FtpsEndpointftpClientKeyStoreParameters、リンク ftpClientTrustStoreParameters、およびリンク FtpsConfigurationgetSecurityProtocol() の設定をオーバーライドする JSSE 設定を取得します。		SSLContextParameters
username (security)	ログインに使用するユーザー名		文字列

第109章 GANGLIA コンポーネント

Camel バージョン 2.15 以降で利用可能

値 (メッセージボディー) をメトリクスとして Ganglia モニタリングシステムに送信するメカニズムを提供します。 gmetric4j ライブラリーを使用します。 標準の Ganglia および JMXetric と組み合わせて使用し、OS、JVM、およびビジネスプロセスからのメトリックを単一のプラットフォームでモニタリングできます。

JVM を実行するマシンで、Ganglia gmond エージェントを実行する必要があります。 gmond は、Ganglia インフラストラクチャーにハートビートを送信します。camel-ganglia は現在、ハートビート自体を送信できません。

ほとんどの Linux システム (Debian、Ubuntu、Fedora、および EPEL を使用する RHEL/CentOS) では、Ganglia エージェントパッケージをインストールするだけで、マルチキャスト設定を使用して自動的に実行されます。 必要に応じて、通常の UDP ユニキャストを使用するように設定できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ganglia</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

109.1. URI 形式

```
ganglia:address:port[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

109.2. GANGLIA コンポーネントとエンドポイント URI オプション

Ganglia コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有設定を使用するには		GangliaConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ganglia エンドポイントは、URI 構文を使用して設定されます。

```
ganglia:host:port
```

パスおよびクエリーパラメーターを使用します。

109.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	Ganglia サーバーのホスト名	239.2.11.71	String
port	Ganglia サーバーのポート	8649	int

109.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
dmax (producer)	有効期限が切れた場合に、Ganglia がメトリック値をパージするまでの最小時間 (秒単位)。0 に設定すると、値は gmond エージェントが再起動するまで無期限に Ganglia に残ります。	0	int
groupName (producer)	メトリックが属するグループ。	java	String
metricName (producer)	メトリックに使用する名前。	メトリクス	String
mode (producer)	MULTICAST または UNICAST を使用して UDP メトリックパケットを送信する	MULTICAST	UDPAddressingMode
prefix (producer)	この文字列とアンダースコアをメトリック名の前に付けます。		String
slope (producer)	スロープ	BOTH	GMetricSlope
spoofHostname (producer)	なりすまし情報 IP: ホスト名		String
tmax (producer)	値が現行と見なされる最大時間 (秒単位)。この後、Ganglia は値が期限切れになったと見なします。	60	int
ttl (producer)	マルチキャストを使用する場合は、パケットの TTL を設定します	5	int

名前	説明	デフォルト	タイプ
type (producer)	値のタイプ	STRING	GMetricType
units (producer)	ウィジェット、リットル、バイトなど、メトリックを修飾する任意の測定単位。k (キロ) や m (ミリ) などの接頭辞を含めないでください。後で他のツールが単位をスケーリングする可能性があります。値はスケーリングされていない必要があります。		String
wireFormat31x (producer)	Ganglia 3.1.0 以降のバージョンのワイヤー形式を使用してください。Ganglia 3.0.x 以前を使用するには、これを false に設定します。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

109.3. メッセージボディー

ボディーの値 (文字列や数値型など) は、Ganglia システムに送信されます。

109.4. 戻り値レスポンス

Ganglia は、単方向 UDP またはマルチキャストを使用してメトリクスを送信します。 メッセージボディーへのレスポンスまたは変更はありません。

109.5. 例

109.5.1. 文字列メトリクスの送信

メッセージボディーは文字列に変換され、メトリック値として送信されます。 数値メトリクスとは異なり、文字列値はグラフ化できませんが、Ganglia ではレポートに使用できます。 すべての Ganglia ホストページの上部にある os_version 文字列は、文字列メトリックの例です。

```
from("direct:string.for.ganglia")
    .setHeader(GangliaConstants.METRIC_NAME, simple("my_string_metric"))
    .setHeader(GangliaConstants.METRIC_TYPE, GMetricType.STRING)
    .to("direct:ganglia.tx");

from("direct:ganglia.tx")
    .to("ganglia:239.2.11.71:8649?mode=MULTICAST&prefix=test");
```

109.5.2. 数値メトリクスの送信

```
from("direct:value.for.ganglia")
    .setHeader(GangliaConstants.METRIC_NAME, simple("widgets_in_stock"))
```

```
    .setHeader(GangliaConstants.METRIC_TYPE, GMetricType.UINT32)
    .setHeader(GangliaConstants.METRIC_UNITS, simple("widgets"))
    .to("direct:ganglia.tx");

from("direct:ganglia.tx")
    .to("ganglia:239.2.11.71:8649?mode=MULTICAST&prefix=test");
```

第110章 GEOCODER コンポーネント

Camel バージョン 2.12 以降で利用可能

geocoder: コンポーネントは、指定された住所のジオコード (緯度と経度) の検索、または逆引きに使用されます。このコンポーネントは、Google Geocoder ライブラリーの Java API を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-geocoder</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

110.1. URI 形式

```
geocoder:address:name[?options]
geocoder:latlng:latitude,longitude[?options]
```

110.2. オプション

Geocoder コンポーネントにはオプションがありません。

Geocoder エンドポイントは、URI 構文を使用して設定されます。

```
geocoder:address:latlng
```

パスおよびクエリーパラメーターを使用します。

110.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
address	アドレスの接頭辞を付ける必要がある地理的なアドレス:		String
latlng	latlng を前に付ける必要がある地理的な緯度と経度:		文字列

110.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
clientId (producer)	このクライアント ID で Google プレミアムを使用するには		文字列

名前	説明	デフォルト	タイプ
clientKey (producer)	このクライアントキーで Google プレミアムを使用するには		文字列
headersOnly (producer)	Exchange をヘッダーで強化するだけで、ボディーはそのままにするかどうか。	false	boolean
language (producer)	使用する言語。	en	文字列
httpClientConfigurer (advanced)	認証メカニズムなどを設定するために、プロデューサーまたはコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定戦略を登録します		HttpClientConfigurer
httpConnectionManager (advanced)	カスタム HttpConnectionManager を使用して接続を管理する場合		HttpConnectionManager
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
proxyAuthDomain (プロキシー)	プロキシー NTML 認証用のドメイン		String
proxyAuthHost (プロキシー)	プロキシー NTML 認証用のオプションのホスト		String
proxyAuthMethod (proxy)	Basic、Digest、または NTLM のいずれかのプロキシーの認証方法。		String
proxyAuthPassword (proxy)	プロキシー認証のパスワード		String
proxyAuthUsername (proxy)	プロキシー認証用のユーザー名		String
proxyHost (proxy)	プロキシーホスト名		String
proxyPort (proxy)	プロキシーポート番号		Integer

110.3. EXCHANGE データ形式

Camel はボディを **com.google.code.geocoder.model.GeocodeResponse** 型として配信します。また、住所が **"current"** の場合、レスポンスは現在のロケーションを JSON で表現した文字列型になります。

オプション **headersOnly** が **true** に設定されている場合、メッセージボディーはそのまま残り、ヘッダーのみが Exchange に追加されます。

110.4. メッセージヘッダー

ヘッダー	説明
CamelGeoCoderStatus	必須。geocoder ライブラリーからのステータスコード。ステータスが **GeocoderStatus.OK** の場合、追加のヘッダーが強化されます
CamelGeoCoderAddress	フォーマットされた住所
CamelGeoCoderLat	ロケーションの緯度。
CamelGeoCoderLng	ロケーションの経度。
CamelGeoCoderLatlng	ロケーションの緯度と経度。コンマ区切り。
CamelGeoCoderCity	市の長い名前。
CamelGeoCoderRegionCode	地域コード。
CamelGeoCoderRegionName	地域名。
CamelGeoCoderCountryLong	国の長い名前。
CamelGeoCoderCountryShort	国の略称。

利用可能なデータと使用中のモード (アドレスと latlng) によっては、すべてのヘッダーが提供されるわけではないことに注意してください。

110.5. サンプル

以下の例では、フランスのパリの緯度と経度を取得しています。

```
from("direct:start")
  .to("geocoder:address:Paris, France")
```

CamelGeoCoderAddress を含むヘッダーを指定すると、エンドポイントの設定がオーバーライドされるため、デンマークのコペンハーゲンのロケーションを取得するために、次のようなヘッダーを含むメッセージを送信できます。

```
template.sendBodyAndHeader("direct:start", "Hello", GeoCoderConstants.ADDRESS, "Copenhagen, Denmark");
```

緯度と経度の住所を取得するには、次のようにします。

```
  from("direct:start")
    .to("geocoder:latlng:40.714224,-73.961452")
    .log("Location ${header.CamelGeocoderAddress} is at lat/lng: ${header.CamelGeocoderLatlng}
and in country ${header.CamelGeoCoderCountryShort}")
```

どちらがログに記録されますか

```
Location 285 Bedford Avenue, Brooklyn, NY 11211, USA is at lat/lng: 40.71412890,-73.96140740
and in country US
```

現在のロケーションを取得するには、次のようにアドレスとして current を使用できます。

```
  from("direct:start")
    .to("geocoder:address:current")
```

第111章 GIT コンポーネント

Camel バージョン 2.16 以降で利用可能

git: コンポーネントを使用すると、一般的な Git リポジトリーを操作できます。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-git</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

URI 形式

```
git://localRepositoryPath[?options]
```

111.1. URI オプション

プロデューサーは、特定のリポジトリーでの操作を許可します。
コンシューマーは、特定のリポジトリーでコミット、タグ、およびブランチを使用できます。

Git コンポーネントにはオプションがありません。

Git エンドポイントは、URI 構文を使用して設定されます。

```
git:localPath
```

パスおよびクエリーパラメーターを使用します。

111.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
localPath	**必須** ローカルリポジトリーパス		String

111.1.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
branchName (Common)	作業するブランチ名		String
password (common)	リモートリポジトリーのパスワード		String

名前	説明	デフォルト	タイプ
remoteName (Common)	pull などの特定の操作で使用するリモートリポジトリー名		String
remotePath (common)	リモートリポジトリーパス		String
tagName (Common)	作業するタグ名		String
username (common)	リモートリポジトリーのユーザー名		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
type (consumer)	コンシューマータイプ		GitType
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
allowEmpty (producer)	空の git コミットを管理するためのフラグ	true	boolean
operation (producer)	リポジトリーで行う操作		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

111.2. メッセージヘッダー

名前	デフォルト値	タイプ	コンテキスト	説明
Camel Git 操作	**null**	String	Producer	エンドポイントオプションとして指定されていない場合に、リポジトリーで実行する操作
Camel GitFilename	**null**	String	Producer	追加操作のファイル名
Camel GitCommitMessage	**null**	String	Producer	コミット操作に関連するコミットメッセージ
Camel GitCommitUsername	**null**	String	Producer	コミット操作でのコミットユーザー名
Camel GitCommitEmail	**null**	String	Producer	コミット操作でのコミット電子メール
Camel GitCommitId	**null**	String	Producer	コミット ID
Camel GitAllowEmpty	**null**	Boolean	Producer	空の git コミットを管理するためのフラグ

111.3. プロデューサーの例

以下は、ファイル test.java をローカルリポジトリーに追加し、マスターブランチで特定のメッセージを使用してコミットし、リモートリポジトリーにプッシュするプロデューサーのルート例です。

```
from("direct:start")
        .setHeader(GitConstants.GIT_FILE_NAME, constant("test.java"))
        .to("git:///tmp/testRepo?operation=add")
        .setHeader(GitConstants.GIT_COMMIT_MESSAGE, constant("first commit"))
        .to("git:///tmp/testRepo?operation=commit")
        .to("git:///tmp/testRepo?
operation=push&remotePath=https://foo.com/test/test.git&username=xxx&password=xxx")
```

111.4. コンシューマーの例

以下は、コミットを消費するコンシューマーのルートの例です。

```
from("git:///tmp/testRepo?type=commit")
                        .to(....)
```

第112章 GITHUB コンポーネント

Camel バージョン 2.15 以降で利用可能

GitHub コンポーネントは、egit-github をカプセル化することで GitHub API と対話します。現在、新しいプルリクエスト、プルリクエストのコメント、タグ、およびコミットのポーリングを提供しています。 また、プルリクエストにコメントを付けたり、プルリクエストを完全に閉じたりすることもできます。

このエンドポイントは Webhook ではなく、単純なポーリングに依存しています。理由は次のとおりです。

- 信頼性安定性への懸念
- 通常、ポーリングするペイロードの種類は大きくありません (さらに、ページングは API で利用できます)。
- Webhook が失敗するような一般公開されていない場所で実行されているアプリをサポートする必要性

GitHub API はかなり広範囲に及ぶことに注意してください。 したがって、このコンポーネントを簡単に拡張して、追加の相互作用を提供できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-github</artifactId>
  <version>${camel-version}</version>
</dependency>
```

112.1. URI 形式

```
github://endpoint[?options]
```

112.2. 必須オプション:

これらはエンドポイントから直接設定できることに注意してください。

GitHub コンポーネントにはオプションがありません。

GitHub エンドポイントは、URI 構文を使用して設定されます。

```
github:type/branchName
```

パスおよびクエリーパラメーターを使用します。

112.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
type	**必須** 実行する git 操作		GitHubType
branchName	ブランチ名		String

112.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
oauthToken (common)	GitHub OAuth トークン、ユーザー名とパスワードが提供されない限り必須		String
password (common)	oauthToken が提供されていない場合は必須の GitHub パスワード		String
repoName (common)	**必要な** GitHub リポジトリー名		String
repoOwner (common)	**必要な** GitHub リポジトリーの所有者 (組織)		String
username (common)	GitHub ユーザー名。oauthToken が提供されていない場合は必須		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
encoding (producer)	git commit ファイルを取得するときに指定されたエンコーディングを使用する場合		String

名前	説明	デフォルト	タイプ
state (producer)	git commit ステータス状態を設定する場合		String
targetUrl (producer)	git commit ステータスのターゲット URL を設定する場合		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

112.3. コンシューマーエンドポイント

エンドポイント	コンテキスト	ボディタイプ
pullRequest	ポーリング	org.eclipse.egit.github.core.PullRequest
pullRequestComment	ポーリング	org.eclipse.egit.github.core.Comment (一般的なプルリクエストのディスカッションに関するコメント) または org.eclipse.egit.github.core.CommitComment (プルリクエストの差分に関するインラインコメント)
tag	ポーリング	org.eclipse.egit.github.core.RepositoryTag
commit	ポーリング	org.eclipse.egit.github.core.RepositoryCommit

112.4. プロデューサーエンドポイント:

エンドポイント	ボディー	メッセージヘッダー
pullRequestComment	String (コメントテキスト)	- GitHubPullRequest (integer) (必須): プルリクエスト番号。 - GitHubInResponseTo (integer): プルリクエストの差分に関する別のインラインコメントに応答する場合に必要です。省略した場合は、プルリクエストのディスカッションに関する一般的なコメントと見なされます。
closePullRequest	none	- GitHubPullRequest (integer) (必須): プルリクエスト番号。
createIssue (Camel 2.18 から)	String (issue ボディーテキスト)	- GitHubIssueTitle (String) (必須): Issue のタイトル。

第113章 GZIP DATAFORMAT

Camel バージョン 2.0 以降で利用可能

GZip データ形式は、メッセージの圧縮および解凍形式です。Zip DataFormat で使用されるものと同じ deflate アルゴリズムを使用しますが、追加のヘッダーがいくつか提供されます。この形式は、一般的な **gzip/gunzip** ツールによって生成されます。GZip 圧縮を使用してマーシャリングされたメッセージは、エンドポイントで消費される直前に GZip 解凍を使用してアンマーシャリングできます。圧縮機能は、大きな XML およびテキストベースのペイロードを処理する場合、または **gzip** ツールを使用して以前に圧縮されたメッセージを読み取る場合に非常に役立ちます。

113.1. オプション

GZip データ形式は、以下にリストされている 1 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

113.2. MARSHAL

この例では、通常のテキスト/XML ペイロードを gzip 圧縮形式を使用する圧縮ペイロードにマーシャリングし、MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().gzip().to("activemq:queue:MY_QUEUE");
```

113.3. UNMARSHAL

この例では、MY_QUEUE という ActiveMQ キューから gzipped されたペイロードを元の形式にアンマーシャリングして **UnGZippedMessageProcessor** に転送し、処理します。

```
from("activemq:queue:MY_QUEUE").unmarshal().gzip().process(new
UnGZippedMessageProcessor());
```

113.4. 依存関係

このデータ形式は **camel-core** で提供されるため、追加の依存関係は必要ありません。

第114章 GOOGLE BIGQUERY コンポーネント

Camel バージョン 2.20 以降で利用可能

114.1. コンポーネントの説明

Google Bigquery コンポーネントは、Google Client Services API を介して Cloud BigQuery インフラストラクチャー へのアクセスを提供します。

現在の実装では gRPC を使用していません。

現在の実装は、BigQuery のクエリーをサポートしていません。つまり、プロデューサーのみです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-bigquery</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

114.2. AUTHENTICATION-CONFIGURATION

Google BigQuery コンポーネント認証は、GCP サービスアカウントでの使用を対象としています。詳細については、Google Cloud Platform Auth ガイド を参照してください。

Google のセキュリティー認証情報は、次の 2 個のオプションのいずれかを使用して明示的に設定できます。

- サービスアカウントのメールとサービスアカウントキー (PEM 形式)
- GCP 認証情報ファイルのロケーション

両方が設定されている場合、サービスアカウントの電子メール/キーが優先されます。

または暗黙的に、接続ファクトリーが Application Default Credentials にフォールバックします。

OBS!デフォルトの認証情報ファイルのロケーションは、GOOGLE_APPLICATION_CREDENTIALS 環境変数を介して設定できます。

サービスアカウントのメールとサービスアカウントキーは、GCP JSON 認証情報ファイルでそれぞれ client_email と private_key として見つかります。

114.3. URI 形式

```
google-bigquery://project-id:datasetId[:tableId]?[options]
```

114.4. オプション

Google BigQuery コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
projectId (producer)	Google Cloud Project Id		String
datasetId (producer)	BigQuery Dataset Id		String
connectionFactory (producer)	Bigquery Service への接続を取得するための ConnectionFactory。指定されていない場合は、デフォルトのものが使用されます		GoogleBigQueryConnectionFactory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google BigQuery エンドポイントは、URI 構文を使用して設定されます。

```
google-bigquery:projectId:datasetId:tableName
```

パスおよびクエリーパラメーターを使用します。

114.4.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
projectId	**必須** Google Cloud プロジェクト ID		String
datasetId	**必須** BigQuery データセット ID		String
tableId	BigQuery テーブル ID		String

114.4.2. クエリーパラメーター(3 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionFactory (producer)	Bigquery Service への接続を取得するための ConnectionFactory。指定されていない場合は、デフォルトが使用されます。		GoogleBigQueryConnectionFactory
useAsInsertId (producer)	挿入 ID として使用するフィールド名		String

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

114.5. メッセージヘッダー

名前	タイプ	説明
CamelGoogleBigQuery.TableSuffix	**String**	データを挿入するときに使用するテーブル 接尾辞
CamelGoogleBigQuery.InsertId	**String**	データの挿入時に使用する InsertId
CamelGoogleBigQuery.PartitionDecorator	**String**	データの挿入時に使用するパーティションを示すパーティションデコレーター
CamelGoogleBigQuery.TableId	**String**	データが送信されるテーブル ID。指定すると、エンドポイント設定がオーバーライドされます

114.6. プロデューサーエンドポイント

プロデューサーエンドポイントは、BigQuery の個別のエクスチェンジとグループ化されたエクスチェンジを同様に受け入れて配信できます。グループ化された取引所には、**Exchange.GROUPED_EXCHANGE** プロパティーセットがあります。

Goole BigQuery プロデューサーは、異なるテーブル 接尾辞またはパーティションデコレータが指定されていない限り、グループ化されたエクスチェンジを 1 回の API 呼び出しで送信します。指定された場合は、データが正しい接尾辞またはパーティションデコレータで書き込まれるように分割されます。

Google BigQuery エンドポイントは、ペイロードがマップまたはマップのリストであると想定しています。マップを含むペイロードは単一の行を挿入し、マップのリストを含むペイロードはリスト内の各エントリーの行を挿入します。

114.7. テンプレートテーブル

参照: https://cloud.google.com/bigquery/streaming-data-into-bigquery#template-tables

テンプレート化されたテーブルは、**GoogleBigQueryConstants.TABLE_SUFFIX** ヘッダーを使用して指定できます。

つまり、次のルートはテーブルを作成し、1日ごとに分割されたレコードを挿入します。

```
from("direct:start")
.header(GoogleBigQueryConstants.TABLE_SUFFIX, "_${date:now:yyyyMMdd}")
.to("google-bigquery:sampleDataset:sampleTable")
```

このユースケースではパーティショニングを使用することをお勧めします。

114.8. パーティション設定

参照: https://cloud.google.com/bigquery/docs/creating-partitioned-tables

テーブルの作成時にパーティショニングを指定すると、設定されたデータは自動的に個別のテーブルにパーティショニングされます。データを挿入するときに、エクスチェンジで **GoogleBigQueryConstants.PARTITION_DECORATOR** ヘッダーを設定することにより、特定のパーティションを指定できます。

114.9. データの整合性の確保

参照: https://cloud.google.com/bigquery/streaming-data-into-bigquery#dataconsistency

挿入 ID は、ヘッダー **GoogleBigQueryConstants.INSERT_ID** を使用するか、クエリーパラメーター **useAsInsertId** を指定することによって、エクスチェンジに設定できます。挿入される行ごとに挿入 ID を指定する必要があるため、ペイロードがリストの場合はエクスチェンジヘッダーを使用できません。ペイロードがリストの場合、**GoogleBigQueryConstants.INSERT_ID** は無視されます。その場合、クエリーパラメーター **useAsInsertId** を使用します。

第115章 GOOGLE CALENDAR コンポーネント

Camel バージョン 2.15 以降で利用可能

Google Calendar コンポーネントは、Google Calendar Web API を介して Google カレンダー へのアクセスを提供します。

Google カレンダーは、OAuth 2.0 プロトコル を使用して Google アカウントを認証し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、アカウントを作成して OAuth 認証情報を生成する 必要があります。認証情報は、clientId、clientSecret、および refreshToken で設定されます。長期間有効な refreshToken を生成するための便利なリソースは、OAuth プレイグラウンド です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-google-calendar</artifactId>
    <version>2.15.0</version>
</dependency>
```

115.1. 1.GOOGLE カレンダーのオプション

Google カレンダーコンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		GoogleCalendar 設定
clientFactory (advanced)	クライアントを作成するためのファクトリーとして GoogleCalendarClientFactory を使用します。デフォルトで BatchGoogleCalendarClientFactory を使用します		GoogleCalendarClient ファクトリー
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google カレンダーエンドポイントは、URI 構文を使用して設定されます。

```
google-calendar:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

115.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		GoogleCalendarApiName
methodName	**必須**: 選択した操作に使用するサブ操作		String

115.1.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
accessToken (common)	OAuth 2 アクセストークン通常、これは 1 時間後に期限切れになるため、長期間の使用には refreshToken をお勧めします。		String
applicationName (Common)	Google カレンダーアプリケーション名。例は camel-google-calendar/1.0 です		String
clientId (common)	カレンダーアプリケーションのクライアント ID		String
clientSecret (Common)	カレンダーアプリケーションのクライアントシークレット		String
emailAddress (Common)	Google サービスアカウントの emailAddress。		String
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
p12FileName (Common)	Google サービスアカウントで使用する秘密鍵を含む p12 ファイルの名前。		String
refreshToken (Common)	OAuth 2 トークンの更新これにより、Google カレンダーコンポーネントは、現在の有効期限が切れるたびに新しい accessToken を取得できます。アプリケーションが長期間使用する場合は必要となります。		String
scopes (Common)	カレンダーアプリケーションがユーザーアカウントに対して持つ権限のレベルを指定します。複数のスコープをコンマで区切ることができます。詳細については、https://developers.google.com/google-apps/calendar/auth を参照してください。	https://www.googleapis.com/auth/calendar	String

名前	説明	デフォルト	タイプ
user (Common)	アプリケーションがサービスアカウントフローで偽装しようとしているユーザーのメールアドレス		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

115.2. URI 形式

GoogleCalendar コンポーネントは、次の URI 形式を使用します。

```
google-calendar://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- acl
- calendars
- channels
- colors
- events
- freebusy
- list
- settings

115.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションはで示されます。エンドポイントに必須のオプションがない場合、オプションのセットの 1 つを提供する必要があります。プロデューサーエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelGoogleCalendar.<option>** の形式である必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelGoogleCalendar.option** ヘッダーをオーバーライドします。

115.4. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Options と **consumer.** の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。

115.5. メッセージヘッダー

CamelGoogleCalendar. 接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

115.6. メッセージボディー

すべての結果メッセージボディーは、GoogleCalendarComponent によって使用される基礎となる API によって提供されるオブジェクトを利用します。プロデューサエンドポイントは、**inBody** エンドポイント URI パラメーターで受信メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントの場合、コンシューマーエンドポイントはすべての要素を個別のメッセージにマップします。

第116章 GOOGLE ドライブコンポーネント

Camel バージョン 2.14 以降で利用可能

Google ドライブコンポーネントは 、Google ドライブ Web API を介して Google ドライブファイルストレージサービス へのアクセスを提供します。

Google ドライブは、OAuth 2.0 プロトコル を使用して Google アカウントを認証し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、アカウントを作成して OAuth 認証情報を生成する 必要があります。認証情報は、clientId、clientSecret、および refreshToken で設定されます。長期間有効な refreshToken を生成するための便利なリソースは、OAuth プレイグラウンド です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-google-drive</artifactId>
        <version>2.14-SNAPSHOT</version>
</dependency>
```

116.1. URI 形式

GoogleDrive コンポーネントは、次の URI 形式を使用します。

```
google-drive://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- drive-about
- drive-apps
- drive-changes
- drive-channels
- drive-children
- drive-comments
- drive-files
- drive-parents
- drive-permissions
- drive-properties
- drive-realtime
- drive-replies
- drive-revisions

116.2. GOOGLEDRIVECOMPONENT

Google ドライブコンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		GoogleDrive 設定
clientFactory (advanced)	クライアントを作成するためのファクトリーとして GoogleCalendarClientFactory を使用します。デフォルトで BatchGoogleDriveClientFactory を使用します		GoogleDriveClient ファクトリー
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google ドライブエンドポイントは、URI 構文を使用して設定されます。

```
google-drive:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

116.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		GoogleDriveApiName
methodName	**必須**: 選択した操作に使用するサブ操作		String

116.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
accessToken (common)	OAuth 2 アクセストークン通常、これは 1 時間後に期限切れになるため、長期間の使用には refreshToken をお勧めします。		String
applicationName (Common)	Google ドライブアプリケーション名。例は camel-google-drive/1.0 です		String

名前	説明	デフォルト	タイプ
clientFactory (Common)	クライアントを作成するためのファクトリーとして GoogleCalendarClientFactory を使用します。デフォルトで BatchGoogleDriveClientFactory を使用します		GoogleDriveClient ファクトリー
clientId (common)	ドライブアプリケーションのクライアント ID		String
clientSecret (Common)	ドライブアプリケーションのクライアントシークレット		String
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
refreshToken (Common)	OAuth 2 トークンの更新これにより、Google カレンダーコンポーネントは、現在の有効期限が切れるたびに新しい accessToken を取得できます。アプリケーションが長期間使用する場合は必要となります。		String
scopes (Common)	ドライブアプリケーションがユーザーアカウントに対して持つ権限のレベルを指定します。詳細については、https://developers.google.com/drive/web/scopes を参照してください。		List
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

116.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションはで示されます。エンドポイントに必須のオプションがない場合、オプションのセットの 1 つを提供する必要があります。プロデューサーエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelGoogleDrive.<option>** の形式にする必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は**CamelGoogleDrive.option** ヘッダーをオーバーライドします。

エンドポイントとオプションの詳細については、https: //developers.google.com/drive/v2/reference/ の API ドキュメントを参照してください。

116.4. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Options と **consumer.** の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。

116.5. メッセージヘッダー

CamelGoogleDrive. 接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

116.6. メッセージボディー

すべての結果メッセージ本文は、GoogleDriveComponent によって使用される基礎となる API によって提供されるオブジェクトを利用します。プロデューサエンドポイントは、**inBody** エンドポイント URI パラメーターで受信メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントの場合、コンシューマーエンドポイントはすべての要素を個別のメッセージにマップします。

第117章 GOOGLE メールコンポーネント

Camel バージョン 2.15 以降で利用可能

Google Mail コンポーネントは、Google Mail Web API を介して Gmail へのアクセスを提供します。

Google Mail は、OAuth 2.0 プロトコル を使用して Google アカウントを認証し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、アカウントを作成して OAuth 認証情報を生成する 必要があります。認証情報は、clientId、clientSecret、および refreshToken で設定されます。長期間有効な refreshToken を生成するための便利なリソースは、OAuth プレイグラウンド です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-google-mail</artifactId>
    <version>2.15-SNAPSHOT</version>
</dependency>
```

117.1. URI 形式

GoogleMail コンポーネントは、次の URI 形式を使用します。

```
google-mail://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- attachments
- drafts
- history
- labels
- messages
- threads
- users

117.2. GOOGLEMAILCOMPONENT

Google Mail コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		GoogleMailConfiguration

名前	説明	デフォルト	タイプ
clientFactory (advanced)	クライアントを作成するためのファクトリーとして GoogleCalendarClientFactory を使用します。デフォルトで BatchGoogleMailClientFactory を使用します		GoogleMailClientFactory
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google Mail エンドポイントは、URI 構文を使用して設定されます。

```
google-mail:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

117.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		GoogleMailApiName
methodName	**必須**: 選択した操作に使用するサブ操作		文字列

117.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
accessToken (common)	OAuth 2 アクセストークン通常、これは 1 時間後に期限切れになるため、長期間の使用には refreshToken をお勧めします。		String
applicationName (Common)	Google メールアプリケーション名。例は camel-google-mail/1.0 です		String
clientId (common)	メールアプリケーションのクライアント ID		String
clientSecret (Common)	メールアプリケーションのクライアントシークレット		String

名前	説明	デフォルト	タイプ
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
refreshToken (Common)	OAuth 2 トークンの更新これにより、Google カレンダーコンポーネントは、現在の有効期限が切れるたびに新しい accessToken を取得できます。アプリケーションが長期間使用する場合は必要となります。		String
scopes (Common)	メールアプリケーションがユーザーアカウントに対して持つ権限のレベルを指定します。詳細は、https://developers.google.com/gmail/api/auth/scopes を参照してください。		List
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

117.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションはで示されます。エンドポイントに必須のオプションがない場合、オプションのセットの 1 つを提供する必要があります。プロデューサーエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelGoogleMail.<option>** の形式にする必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelGoogleMail.option** ヘッダーをオーバーライドします。

エンドポイントとオプションの詳細については、https://developers.google.com/gmail/api/v1/reference/ の API ドキュメントを参照してください。

117.4. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Options と **consumer.** の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。

117.5. メッセージヘッダー

CamelGoogleMail. 接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

117.6. メッセージボディー

すべての結果メッセージボディーは、GoogleMailComponent によって使用される基礎となる API によって提供されるオブジェクトを利用します。プロデューサエンドポイントは、**inBody** エンドポイント URI パラメーターで受信メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントの場合、コンシューマーエンドポイントはすべての要素を個別のメッセージにマップします。

第118章 GOOGLE PUBSUB コンポーネント

Camel バージョン 2.19 以降で利用可能

Google Pubsub コンポーネントは、Google Client Services API を介して Cloud Pub/Sub インフラストラクチャー へのアクセスを提供します。

現在の実装では gRPC を使用していません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-pubsub</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

118.1. URI 形式

GoogleMail コンポーネントは、次の URI 形式を使用します。

```
google-pubsub://project-id:destinationName?[options]
```

宛先名は、トピック名またはサブスクリプション名のいずれかです。

118.2. オプション

Google Pubsub コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionFactory (common)	使用する接続ファクトリーを設定します: 接続認証情報を明示的に管理する機能を提供します: - キーファイルへのパス - サービスアカウントキー/電子メールのペア		GooglePubsubConnection ファクトリー
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google Pubsub エンドポイントは、URI 構文を使用して設定されます。

```
google-pubsub:projectId:destinationName
```

パスおよびクエリーパラメーターを使用します。

118.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
projectId	**必須** プロジェクト ID		String
destinationName	**必須** 宛先名		String

118.2.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
ackMode (Common)	AUTO = エクスチェンジは完了時に確認/拒否されます。NONE = ダウンストリームプロセスは明示的に ack/nack する必要があります	AUTO	AckMode
concurrentConsumers (Common)	サブスクリプションから消費する並列ストリームの数	1	Integer
connectionFactory (common)	PubSub サービスへの接続を取得するための ConnectionFactory。指定されていない場合は、デフォルトが使用されます。		GooglePubsubConnection ファクトリー
loggerId (Common)	親ルートとの一致が必要な場合に使用するロガー ID		String
maxMessagesPerPoll (Common)	1 回の API 呼び出しでサーバーから受信するメッセージの最大数	1	Integer
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

118.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、PubSub の個別のエクスチェンジとグループ化されたエクスチェンジを同様に受け入れて配信できます。グループ化された取引所には、**Exchange.GROUPED_EXCHANGE** プロパティーセットがあります。

Google PubSub は、ペイロードが byte[] 配列であることを想定しており、プロデューサーエンドポイントは次を送信します。

- UTF-8 としてエンコードされた byte[] としての String ボディー
- byte[] ボディそのまま
- それ以外はすべて byte[] 配列にシリアル化される

メッセージヘッダー **GooglePubsubConstants.ATTRIBUTES** として設定されたマップは、PubSub 属性として送信されます。エクスチェンジが PubSub に配信されると、PubSub メッセージ ID がヘッダー **GooglePubsubConstants.MESSAGE_ID** に割り当てられます。

118.4. コンシューマーエンドポイント

サブスクリプションの設定オプションとして設定された期間内にメッセージが確認されない場合、Google PubSub はメッセージを再配信します。

エクスチェンジ処理が完了すると、コンポーネントはメッセージを確認します。

ルートが例外を出力した場合、エクスチェンジは失敗としてマークされ、コンポーネントはメッセージに NACK を送信します。メッセージはすぐに再配信されます。

メッセージを確認/拒否するために、コンポーネントはヘッダー **GooglePubsubConstants.ACK_ID** として保存されている確認 ID を使用します。ヘッダーが削除または改ざんされた場合、ack は失敗し、ack の期限後にメッセージが再配信されます。

118.5. メッセージヘッダー

コンシューマーエンドポイントによって設定されるヘッダー:

- GooglePubsubConstants.MESSAGE_ID
- GooglePubsubConstants.ATTRIBUTES
- GooglePubsubConstants.PUBLISH_TIME
- GooglePubsubConstants.ACK_ID

118.6. メッセージボディー

コンシューマーエンドポイントは、メッセージの内容を byte[] として返します。これは、基になるシステムが送信するのとまったく同じです。コンテンツを変換/アンマーシャリングするのはルート次第です。

118.7. AUTHENTICATION-CONFIGURATION

Google Pubsub コンポーネント認証は、GCP サービスアカウントでの使用を対象としています。詳細については、Google Cloud Platform Auth ガイド を参照してください。

Google のセキュリティー認証情報は、次の 2 個のオプションのいずれかを使用して明示的に設定できます。

- サービスアカウントのメールとサービスアカウントキー (PEM 形式)
- GCP 認証情報ファイルのロケーション

両方が設定されている場合、サービスアカウントの電子メール/キーが優先されます。

または暗黙的に、接続ファクトリーが Application Default Credentials にフォールバックします。

OBS!デフォルトの認証情報ファイルのロケーションは、GOOGLE_APPLICATION_CREDENTIALS 環境変数を介して設定できます。

サービスアカウントのメールとサービスアカウントキーは、GCP JSON 認証情報ファイルでそれぞれ client_email と private_key として見つかります。

118.8. ロールバックと再配信

Google PubSub のロールバックは、確認応答期限 (Google PubSub が確認応答を受け取る予定の期間) の考え方に依存しています。受信確認が受信されていない場合、メッセージは再配信されます。

Google は、メッセージの期限を延長するための API を提供しています。

詳細については、Google PubSub ドキュメント を参照してください

したがって、ロールバックは事実上、値がゼロの期限延長 API 呼び出しです。つまり、期限に達したので、メッセージを次のコンシューマーに再配信できます。

メッセージヘッダー **GooglePubsubConstants.ACK_DEADLINE** を秒単位の値に設定して、ロールバックの確認期限を明示的に設定することで、メッセージの再配信を遅らせることができます。

第119章 GROOVY 言語

Camel バージョン 1.3 以降で利用可能

Camel は、他のスクリプト言語の中でも Groovy をサポートし、式または述語を DSL または Xml 設定で使用できるようにします。

Groovy 式を使用するには、次の Java コードを使用します

```
... groovy("someGroovyExpression") ...
```

たとえば、**groovy** 関数を使用して、メッセージフィルター で述語を作成したり、受信者リストの式として使用したりできます。

119.1. GROOVY オプション

Groovy 言語は、以下に示す 1 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

119.2. GROOVY SHELL のカスタマイズ

Groovy 式でカスタム **GroovyShell** インスタンスを使用する必要がある場合があります。カスタム **GroovyShell** を指定するには、**org.apache.camel.language.groovy.GroovyShellFactory** SPI インターフェイスの実装を Camel レジストリーに追加します。たとえば、次の Bean を Spring コンテキストに追加した後...

```
public class CustomGroovyShellFactory implements GroovyShellFactory {

  public GroovyShell createGroovyShell(Exchange exchange) {
    ImportCustomizer importCustomizer = new ImportCustomizer();
    importCustomizer.addStaticStars("com.example.Utils");
    CompilerConfiguration configuration = new CompilerConfiguration();
    configuration.addCompilationCustomizers(importCustomizer);
    return new GroovyShell(configuration);
  }

}
```

...Camel は、デフォルトのインスタンスではなく、カスタムの GroovyShell インスタンス (カスタムの静的インポートを含む) を使用します。

119.3. 例

```
// lets route if a line item is over $100
from("queue:foo").filter(groovy("request.lineItems.any { i -> i.value > 100 }")).to("queue:bar")
```

そして Spring DSL:

```
<route>
  <from uri="queue:foo"/>
  <filter>
    <groovy>request.lineItems.any { i -> i.value > 100 }</groovy>
    <to uri="queue:bar"/>
  </filter>
</route>
```

119.4. SCRIPTCONTEXT

JSR-223 スクリプト言語の ScriptContext は、すべて **ENGINE_SCOPE** に設定された次の属性で事前設定されています。

属性	タイプ	値
context	**org.apache.camel.CamelContext**	Camel コンテキスト (groovy では使用できません)
camelContext	**org.apache.camel.CamelContext**	Camel コンテキスト
exchange	**org.apache.camel.Exchange**	現在のエクスチェンジ
request	**org.apache.camel.Message**	メッセージ (IN メッセージ)
response	**org.apache.camel.Message**	**非推奨**: OUT メッセージ。デフォルトで null の場合は OUT メッセージ。代わりに IN メッセージを使用してください。

属性	タイプ	値
properties	**org.apache.camel.builder.script.PropertiesFunction**	Camel 2.9: スクリプトから CamelsProperties コンポーネントを簡単に使用できるように、**resolve** メソッドを備えた関数。例については、以下を参照してください。

明示的な DSL をサポートする言語のリストについては、スクリプト言語を参照してください。

119.5. SCRIPTINGENGINE への追加の引数

Camel 2.8 から利用可能

CamelScriptArguments キーを持つ Camel メッセージのヘッダーを使用して、 **ScriptingEngine** に追加の引数を提供できます。
次の例を参照してください。

119.6. プロパティー機能の使用

Camel 2.9 以降で利用可能

スクリプトから Properties コンポーネントを使用してプロパティープレースホルダーを検索する必要がある場合は、少し面倒です。たとえば、ヘッダー名 myHeader をプロパティープレースホルダーの値で設定するには、そのキーは foo という名前のヘッダーで提供されます。

```
.setHeader("myHeader").groovy(""context.resolvePropertyPlaceholders( + '{{' +
request.headers.get(&#39;foo&#39;) + '}}' + ")")
```

Camel 2.9 以降では、プロパティー関数を使用できるようになり、同じ例がより単純になりました。

```
.setHeader("myHeader").groovy("properties.resolve(request.headers.get(&#39;foo&#39;))")
```

119.7. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").groovy("resource:classpath:mygroovy.groovy")
```

119.8. 複数ステートメントのスクリプトから結果を取得する方法

Camel 2.14 から利用可能

scripteengine の evale メソッドは、複数のステートメントスクリプトを実行する場合、Null を返すだけです。Camel は、値セットから結果のキーを使用して、スクリプトの結果の値を検索するようになりました。複数のステートメントスクリプトがある場合は、結果変数の値をスクリプトの戻り値として設定する必要があります。

```
bar = "baz";
# some other statements ...
# camel take the result value as the script evaluation result
result = body * 2 + 1
```

119.9. 依存関係

camel ルートでスクリプト言語を使用するには、**camel-groovy** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-groovy</artifactId>
  <version>x.x.x</version>
</dependency>
```

第120章 GRPC COMPONENT

Camel バージョン 2.19 以降で利用可能

gRPC コンポーネントを使用すると、HTTP/2 トランスポートで プロトコルバッファー (protobuf) エクスチェンジフォーマットを使用して、リモートプロシージャコール (RPC) サービスを呼び出したり、公開したりできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-grpc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

120.1. URI 形式

```
grpc://service[?options]
```

120.2. エンドポイントオプション

gRPC コンポーネントにはオプションがありません。

gRPC エンドポイントは、URI 構文を使用して設定されます。

```
grpc:host:port/service
```

パスおよびクエリーパラメーターを使用します。

120.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** gRPC サーバーのホスト名。これは、コンシューマーの場合は localhost または 0.0.0.0、プロデューサーを使用する場合はリモートサーバーのホスト名です。		String
port	**必須** gRPC ローカルまたはリモートサーバーポート		int
service	**必須** プロトコルバッファー記述子ファイルからの完全修飾サービス名 (パッケージドットサービス定義名)		String

120.2.2. クエリーパラメーター(25 個のパラメーター):

名前	説明	デフォルト	タイプ
flowControlWindow (common)	HTTP/2 フロー制御ウィンドウサイズ (MiB)	1048576	int
maxMessageSize (common)	送受信できるメッセージの最大サイズ (MiB)	4194304	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerStrategy (consumer)	このオプションは、ストリーミングモードでサービスリクエストとレスポンスを処理するためのトップレベルのストラテジーを指定します。集約ストラテジーが選択されている場合、すべてのリクエストがリストに蓄積されてからフローに転送され、蓄積されたレスポンスが送信者に送信されます。伝播ストラテジーが選択されている場合、リクエストはストリームに送信され、レスポンスはすぐに送信者に返されます。	PROPAGATION	GrpcConsumerStrategy
forwardOnCompleted (consumer)	onCompleted イベントを Camel ルートにプッシュするかどうかを決定します。	false	boolean
forwardOnError (consumer)	onError イベントを Camel ルートにプッシュするかどうかを決定します。例外はメッセージボディーとして設定されます。	false	boolean
maxConcurrentCallsPerConnection (consumer)	着信サーバー接続ごとに許可される同時呼び出しの最大数	2147483647	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
method (producer)	gRPC メソッド名		String
producerStrategy (producer)	リモート gRPC サーバーとの通信に使用されるモード。SIMPLE モードでは、単一のエクスチェンジがリモートプロシージャコールに変換されます。STREAMING モードでは、すべてのエクスチェンジが同じリクエスト内で送信されます (受信者 gRPC サービスの入力と出力はストリームタイプである必要があります)。	SIMPLE	GrpcProducerStrategy
streamRepliesTo (producer)	STREAMING クライアントモードを使用する場合、レスポンスを転送するエンドポイントを示します。		String
userAgent (producer)	サーバーに渡されたユーザーエージェントヘッダー		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
authenticationType (security)	SSL/TLS ネゴシエーションに先立つ認証方式の種類	NONE	GrpcAuthType
jwtAlgorithm (security)	JSON Web Token 署名アルゴリズム	HMAC256	JwtAlgorithm
jwtIssuer (security)	JSON Web トークン発行者		String
jwtSecret (security)	JSON Web トークンシークレット		String
jwtSubject (security)	JSON Web Token サブジェクト		String
keyCertChainResource (security)	PEM 形式の X.509 証明書チェーンファイルリソースリンク		String
keyPassword (security)	PKCS8 秘密鍵ファイルのパスワード		文字列
keyResource (security)	PEM 形式の PKCS8 秘密鍵ファイルリソースリンク		文字列
negotiationType (security)	HTTP/2 通信に使用されるセキュリティーネゴシエーションタイプを識別します	PLAINTEXT	NegotiationType

名前	説明	デフォルト	タイプ
serviceAccountResource (security)	Google Cloud SDK でサポートされている JSON 形式のリソースリンクのサービスアカウントキーファイル		String
trustCertCollectionResource (security)	リモートエンドポイントの証明書を検証するための PEM 形式の信頼できる証明書コレクションファイルリソース		文字列

120.3. トランスポートセキュリティーと認証のサポート (CAMEL 2.20 から利用可能)

次の 認証 メカニズムは gRPC に組み込まれており、このコンポーネントで使用できます。

- **SSL/TLS:** gRPC には SSL/TLS 統合があり、SSL/TLS を使用してサーバーを認証し、クライアントとサーバー間でエクスチェンジされるすべてのデータを暗号化することを促進します。クライアントが相互認証用の証明書を提供するためのオプションのメカニズムを使用できます。
- **Google によるトークンベースの認証:** gRPC は、メタデータベースの認証情報を要求と応答に添付するための一般的なメカニズムを提供します。gRPC を介して Google API にアクセスしながらアクセストークンを取得するための追加のサポートが提供されます。一般に、このメカニズムは、チャネルで SSL/TLS と同様に使用する必要があります。

これらの機能を有効にするには、次のコンポーネントプロパティーの組み合わせを設定する必要があります。

Num.	オプション	パラメーター	値	必須/オプション
1	**SSL/TLS**	negotiationType	TLS	必須
		keyCertChainResource		必須
		keyResource		必須
		keyPassword		オプション
		trustCertCollectionResource		オプション
2	**Google API によるトークンベースの認証**	authenticationType	GOOGLE	必須
		negotiationType	TLS	必須

Num.	オプション	パラメーター	値	必須/オプション
		serviceAccountResource		必須
3	**カスタム JSON Web トークン実装認証**	authenticationType	JWT	必須
		negotiationType	NONE または TLS	オプション。TLS/SSL はこのタイプをチェックしませんが、強くお勧めします。
		jwtAlgorithm	HMAC256(デフォルト) または (HMAC384, HMAC512)	オプション
		jwtSecret		必須
		jwtIssuer		オプション
		jwtSubject		オプション

OpenSSL を使用した TLS は、gRPC over TLS コンポーネントを使用するための現在推奨されるアプローチです。ALPN に JDK を使用すると、通常ははるかに遅くなり、HTTP2 に必要な暗号をサポートしない場合があります。この機能はコンポーネントには実装されていません。

120.4. GRPC プロデューサーリソースタイプのマッピング

以下の表は、入力および出力パラメーターのタイプ (単純またはストリーム) と呼び出しスタイル (同期または非同期) に応じた、メッセージボディー内のオブジェクトのタイプを示しています。非同期スタイルの入力ストリームパラメーターを使用したプロシージャの呼び出しは許可されていないことに注意してください。

呼び出しスタイル	リクエストのタイプ	レスポンスのタイプ	リクエストボディータイプ	結果ボディーのタイプ
同期	simple	simple	Object	Object
同期	simple	stream	Object	List<Object>
同期	stream	simple	not allowed	not allowed
同期	stream	stream	not allowed	not allowed
非同期	simple	simple	Object	List<Object>

呼び出しスタイル	リクエストのタイプ	レスポンスのタイプ	リクエストボディータイプ	結果ボディーのタイプ
非同期	simple	stream	Object	List<Object>
非同期	stream	simple	Object または List<Object>	List<Object>
非同期	stream	stream	Object または List<Object>	List<Object>

120.5. GRPC コンシューマーヘッダー (コンシューマーの呼び出し後にインストールされます)

ヘッダー名	説明	使用できる値
CamelGrpcMethodName	コンシューマーサービスが扱うメソッド名	
CamelGrpcEventType	送信されたリクエストから受信したイベントタイプ	onNext、onCompleted、または onError
CamelGrpcUserAgent	提供されている場合、指定されたエージェントは gRPC ライブラリーのユーザーエージェント情報を先頭に追加します	

120.6. 例

以下は、ホストとポートのパラメーターを使用した単純な同期メソッドの呼び出しです。

```
from("direct:grpc-sync")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=sendPing&synchronous=true");
```

```
<route>
    <from uri="direct:grpc-sync" />
    <to uri="grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=sendPing&synchronous=true"/>
</route>
```

非同期メソッド呼び出し

```
from("direct:grpc-async")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=pingAsyncResponse");
```

伝播コンシューマーストラテジーを使用した gRPC サービスコンシューマー

```
from("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
consumerStrategy=PROPAGATION")
.to("direct:grpc-service");
```

ストリーミングプロデューサーストラテジーを使用した gRPC サービスプロデューサー (ストリームモードを入力および出力として使用するサービスが必要です)

```
from("direct:grpc-request-stream")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=PingAsyncAsync&producerStrategy=STREAMING&streamRepliesTo=direct:grpc-response-
stream");

from("direct:grpc-response-stream")
.log("Response received: ${body}");
```

gRPC サービスコンシューマー TLS/SLL セキュリティーネゴシエーションの有効化

```
from("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
consumerStrategy=PROPAGATION&negotiationType=TLS&keyCertChainResource=file:src/test/resour
ces/certs/server.pem&keyResource=file:src/test/resources/certs/server.key&trustCertCollectionResource
=file:src/test/resources/certs/ca.pem")
.to("direct:tls-enable")
```

カスタム JSON Web トークン実装認証を使用した gRPC サービスプロデューサー

```
from("direct:grpc-jwt")
.to("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
method=pingSyncSync&synchronous=true&authenticationType=JWT&jwtSecret=supersecuredsecret"
);
```

120.7. 設定

カスタムプロジェクトの .proto (プロトコルバッファー定義) ファイルから Java ソースファイルを生成するために Protocol Buffer Compiler (protoc) ツールを呼び出す Maven Protocol Buffers Plugin を使用することをお勧めします。このプラグインは、プロシージャのリクエストとレスポンスのクラス、それらのビルダー、および gRPC プロシージャのスタブクラスも生成します。

次の手順が必要です。

プロジェクト pom.xml の **<build>** タグ内にオペレーティングシステムと CPU アーキテクチャー検出拡張機能を挿入するか、手動で ${os.detected.classifier} パラメーターを設定します。

```
<extensions>
  <extension>
    <groupId>kr.motd.maven</groupId>
    <artifactId>os-maven-plugin</artifactId>
    <version>1.4.1.Final</version>
  </extension>
</extensions>
```

プロジェクト pom.xml の gRPC および protobuf Java コードジェネレータープラグイン **<plugins>** タグを挿入します。

```
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <version>0.5.0</version>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:${protobuf-
version}:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc-
version}:exe:${os.detected.classifier}</pluginArtifact>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>compile-custom</goal>
        <goal>test-compile</goal>
        <goal>test-compile-custom</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

120.8. 詳細については、これらのリソースを参照してください

gRPC プロジェクトサイト

Maven プロトコルバッファープラグイン

120.9. 関連項目

- スタートガイド
- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- プロトコルバッファーのデータ形式

第121章 GUAVA EVENTBUS コンポーネント

Camel バージョン 2.10 以降で利用可能

Google Guava EventBus を使用すると、コンポーネントを相互に明示的に登録する必要なく (したがって、相互に認識しなくても)、コンポーネント間のパブリッシュ/サブスクライブスタイルの通信が可能になります。**guava-eventbus:** コンポーネントは、Camel と Google Guava EventBus インフラストラクチャー間の統合ブリッジを提供します。後者のコンポーネントを使用すると、Guava **EventBus** とエクスチェンジされるメッセージを透過的に Camel ルートに転送できます。EventBus コンポーネントを使用すると、Camel エクスチェンジのボディーを Guava **EventBus** にルーティングすることもできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-guava-eventbus</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

121.1. URI 形式

```
guava-eventbus:busName[?options]
```

busName は、 Camel レジストリーにある **com.google.common.eventbus.EventBus** インスタンスの名前を表します。

121.2. オプション

Guava EventBus コンポーネントは、以下に示す 3 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
eventBus (Common)	指定された Guava EventBus インスタンスを使用するには		EventBus
listenerInterface (common)	Subscribe アノテーションでマークされたメソッドを含むインターフェイス。EventBus リスナーとして登録できるように、インターフェイスを介して動的プロキシーが作成されます。マルチイベントリスナーを作成する場合や、DeadEvent を適切に処理する場合に特に役立ちます。このオプションは、eventClass オプションと一緒には使用できません。		Class<?>
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Guava EventBus エンドポイントは、URI 構文を使用して設定されます。

```
guava-eventbus:eventBusRef
```

パスおよびクエリーパラメーターを使用します。

121.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
eventBusRef	指定された名前でレジストリーから Guava EventBus を検索するには		String

121.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
eventClass (Common)	ルートのコンシューマー側で使用された場合、EventBus から受信したイベントをクラスのインスタンスと eventClass のスーパークラスにフィルタリングします。このオプションの Null 値は、java.lang.Object に設定するのと同じです。つまり、コンシューマーは、イベントバスに着信するすべてのメッセージをキャプチャーします。このオプションは、listenerInterface オプションと一緒には使用できません。		Class<?>
listenerInterface (common)	Subscribe アノテーションでマークされたメソッドを含むインターフェイス。EventBus リスナーとして登録できるように、インターフェイスを介して動的プロキシーが作成されます。マルチイベントリスナーを作成する場合や、DeadEvent を適切に処理する場合に特に役立ちます。このオプションは、eventClass オプションと一緒には使用できません。		Class<?>
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

121.3. 使用方法

ルートのコンシューマー側で **guava-eventbus** コンポーネントを使用すると、Guava **EventBus** に送信されたメッセージがキャプチャーされ、Camel ルートに転送されます。Guava EventBus コンシューマーは、入力メッセージを 非同期的に 処理します。

```
SimpleRegistry registry = new SimpleRegistry();
EventBus eventBus = new EventBus();
registry.put("busName", eventBus);
CamelContext camel = new DefaultCamelContext(registry);

from("guava-eventbus:busName").to("seda:queue");

eventBus.post("Send me to the SEDA queue.");
```

ルートのプロデューサー側で **guava-eventbus** コンポーネントを使用すると、Camel 交換の本文が Guava **EventBus** インスタンスに転送されます。

```
SimpleRegistry registry = new SimpleRegistry();
EventBus eventBus = new EventBus();
registry.put("busName", eventBus);
CamelContext camel = new DefaultCamelContext(registry);

from("direct:start").to("guava-eventbus:busName");

ProducerTemplate producerTemplate = camel.createProducerTemplate();
producer.sendBody("direct:start", "Send me to the Guava EventBus.");

eventBus.register(new Object(){
  @Subscribe
  public void messageHander(String message) {
    System.out.println("Message received from the Camel: " + message);
  }
});
```

121.4. DEADEVENT に関する考慮事項

Guava EventBus の設計による制限により、**@Subscribe** メソッドでアノテーションが付けられたクラスを作成しないと、リスナーが受信するイベントクラスを指定できないことに注意してください。この制限は、指定された **eventClass** オプションを持つエンドポイントが実際にすべての可能なイベント (**java.lang.Object**) をリッスンし、実行時にプログラムによって適切なメッセージをフィルタリングすることを意味します。以下は、Camel コードベースからの適切な抜粋を示しています。

```
@Subscribe
public void eventReceived(Object event) {
  if (eventClass == null || eventClass.isAssignableFrom(event.getClass())) {
    doEventReceived(event);
...
```

このアプローチの欠点は、Camel で使用される **EventBus** インスタンスが **com.google.common.eventbus.DeadEvent** 通知を生成しないことです。Camel が正確に指定されたイベントのみをリッスンする (したがって、**DeadEvent** サポートを有効にする) 場合は、**listenerInterface** エンドポイントオプションを使用します。Camel は、後者のオプションで指定したインターフェイス上に動的プロキシーを作成し、インターフェイスハンドラーメソッドによって指定されたメッセージのみをリッスンします。**SpecificEvent** インスタンスのみを処理する単一のメソッドを持つリスナーインターフェイスの例を以下に示します。

```
package com.example;

public interface CustomListener {

  @Subscribe
  void eventReceived(SpecificEvent event);

}
```

上記のリスナーは、次のようにエンドポイント定義で使用できます。

```
from("guava-eventbus:busName?listenerInterface=com.example.CustomListener").to("seda:queue");
```

121.5. 複数のタイプのイベントの消費

Guava EventBus コンシューマーによって消費される複数のタイプのイベントを定義するには、**listenerInterface** エンドポイントオプションを使用します。これは、リスナーインターフェイスが **@Subscribe** アノテーションでマークされた複数のメソッドを提供できるためです。

```
package com.example;

public interface MultipleEventsListener {

  @Subscribe
  void someEventReceived(SomeEvent event);

  @Subscribe
  void anotherEventReceived(AnotherEvent event);

}
```

上記のリスナーは、次のようにエンドポイント定義で使用できます。

```
from("guava-eventbus:busName?
listenerInterface=com.example.MultipleEventsListener").to("seda:queue");
```

121.6. HAWTDB

Camel 2.3 の時点で利用可能

HawtDB は非常に軽量で組み込み可能なキー値データベースです。Camel と一緒に、Aggregator などのさまざまな Camel 機能の永続的なサポートを提供できます。

非推奨

HawtDB プロジェクトは非推奨になり、軽量で埋め込み可能なキー値データベースとして leveldb に置き換えられています。leveldb を簡単に使用できるようにするための leveldbjni プロジェクトがあります。Apache ActiveMQ プロジェクトは、今後 kahadb を置き換えるために、主要なファイルベースのメッセージストアとして leveldb を使用することを計画しています。

これの代わりに使用することをお勧めする camel-leveldb コンポーネントがあります。

HawtDB 1.4 以前の問題

HawtDB 1.4 以前にはバグがあり、ファイルストアが未使用のスペースを解放しません。これは、ファイルが増え続けることを意味します。これは、Camel 2.5 以降に同梱されている HawtDB 1.5 で修正されています。

それが提供する現在の機能:

- HawtDBAggregationRepository

121.6.1. HawtDBAggregationRepository の使用

HawtDBAggregationRepository は **AggregationRepository** であり、その場で集約されたメッセージを永続化します。デフォルトのアグリゲーターはメモリー内のみの **AggregationRepository** を使用するため、これによりメッセージが失われないことが保証されます。

以下のオプションがあります。

オプション	タイプ	説明
repositoryName	String	必須のリポジトリー名。複数のリポジトリーに共有 **HawtDBFile** を使用できるようにします。
persistentFileName	String	永続ストレージのファイル名。起動時にファイルが存在しない場合は、新しいファイルが作成されます。
bufferSize	int	ファイルストアにマップされるメモリーセグメントバッファーのサイズ。デフォルトでは 8MB です。値はバイト単位です。

オプション	タイプ	説明
sync	boolean	**HawtDBFile** が書き込み時に同期するかどうか。デフォルトは **true** です。書き込み時に同期することにより、すべての書き込みがディスクにスプールされるのを常に待機するため、更新が失われることはありません。このオプションを無効にすると、HawtDB は多数の書き込みをまとめたときに自動同期します。
pageSize	short	メモリーページのサイズ。デフォルトでは 512 バイトです。値はバイト単位です。
hawtDBFile	HawtDBFile	既存の設定済み **org.apache.camel.component.hawtdb.HawtDBFile** インスタンスを使用します。
returnOldExchange	boolean	古い既存の Exchange が存在する場合、get 操作でそれを返すかどうか。デフォルトでは、集約時に古いエクスチェンジは必要ないため、最適化するにはこのオプションは **false** です。
useRecovery	boolean	リカバリーが有効かどうか。このオプションはデフォルトで **true** です。Camel Aggregator の自動回復を有効にすると、集約されたエクスチェンジに失敗し、それらが再送信されます。
recoveryInterval	long	回復が有効になっている場合、x 回ごとにバックグラウンドタスクが実行され、失敗した交換をスキャンして回復し、再送信します。デフォルトでは、この間隔は 5000 ミリ秒です。
maximumRedeliveries	int	回復されたエクスチェンジの再配信試行の最大回数を制限できます。有効にすると、すべての再配信の試行が失敗した場合、エクスチェンジはデッドレターチャネルに移動されます。デフォルトでは、このオプションは無効となっています。このオプションを使用する場合は、**deadLetterUri** オプションも指定する必要があります。
deadLetterUri	String	枯渇した回復済みエクスチェンジが移動されるデッドレターチャネルのエンドポイント uri。このオプションを使用する場合、**maximumRedeliveries** オプションも指定する必要があります。
optimisticLocking	**false**	Camel 2.12: 複数の Camel アプリケーションが同じ HawtDB ベースの集計リポジトリーを共有するクラスター化された環境で必要になることが多い楽観的ロックを有効にします。

repositoryName オプションを指定する必要があります。次に、**persistentFileName** または **hawtDBFile** を指定する必要があります。

121.6.2. 持続時に保持されるもの

HawtDBAggregationRepository は、**Serializable** な互換性のあるデータ型のみを保持します。データ型がそのような型でない場合は削除され、**WARN** がログに記録されます。また、**Message** ボディーと **Message** ヘッダーのみを保持します。**Exchange** プロパティーは保持され **ません**。

121.6.3. 復元

HawtDBAggregationRepository はデフォルトで、失敗したエクスチェンジを回復します。これは、永続ストア内の失敗したエクスチェンジをスキャンするバックグラウンドタスクを持つことによって行われます。**checkInterval** オプションを使用して、このタスクの実行頻度を設定できます。回復はトランザクションとして機能するため、Camel は失敗したエクスチェンジを回復して再配信しようとします。回復されたことが判明したエクスチェンジは、永続ストアから復元され、再送信され、再度送信されます。

次のヘッダーは、エクスチェンジが回復または再配信されるときに設定されます。

ヘッダー	タイプ	説明
Exchange.REDELIVERED	Boolean	エクスチェンジが再配信されていることを示すために true に設定されます。
Exchange.REDELIVERY_COUNTER	Integer	1 から始まる再配信の試行。

エクスチェンジが正常に処理された場合にのみ、**AggregationRepository** で **confirm** メソッドが呼び出されたときに完了としてマークされます。これは、同じエクスチェンジが再び失敗した場合、成功するまで再試行されることを意味します。

オプション **maximumRedeliveries** を使用して、復元された特定の Exchange の再配信試行の最大回数を制限できます。**maximumRedeliveries** に達したときに Camel がエクスチェンジの送信先を認識できるように、**deadLetterUri** オプションも設定する必要があります。

このテスト など、camel-hawtdb の単体テストでいくつかの例を確認できます。

121.6.3.1. Java DSL での HawtDBAggregationRepository の使用

この例では、集計されたメッセージを **target/data/hawtdb.dat** ファイルに保存します。

121.6.3.2. Spring XML での HawtDBAggregationRepository の使用

同じ例ですが、代わりに Spring XML を使用しています。

121.6.4. 依存関係

camel ルートで HawtDB を使用するには、**camel-hawtdb** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hawtdb</artifactId>
```

```
  <version>2.3.0</version>
</dependency>
```

121.6.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- アグリゲーター
- コンポーネント

第122章 HAZELCAST コンポーネント

Camel バージョン 2.7 以降で利用可能

hazelcast- コンポーネントを使用すると、Hazelcast 分散データグリッド/キャッシュを操作できます。Hazelcast はインメモリーデータグリッドであり、Java で完全に記述されています (単一の jar)。マップ、マルチマップ (同じキー、n 値)、キュー、リスト、原子番号など、さまざまなデータストアの優れたパレットを提供します。Hazelcast を使用する主な理由は、単純なクラスターサポートです。ネットワークでマルチキャストを有効にしている場合は、追加の設定なしで数百のノードを持つクラスターを実行できます。Hazelcast は、ノード間の n 個のコピー (デフォルトは 1)、キャッシュの永続性、ネットワーク設定 (必要な場合)、ニアキャッシュ、enviction などの追加機能を追加するように簡単に設定できます。詳細については、http://www.hazelcast.com/docs.jsp にある Hazelcast のドキュメントを参照してください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hazelcast</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

122.1. HAZELCAST コンポーネント

各コンポーネントの使用法については、以下を参照してください: * map * multimap * queue * topic * list * seda * set * atomic number * cluster support (instance) * replicatedmap * ringbuffer

122.2. HAZELCAST 参照の使用

122.2.1. その名前別

```
<bean id="hazelcastLifecycle" class="com.hazelcast.core.LifecycleService"
      factory-bean="hazelcastInstance" factory-method="getLifecycleService"
      destroy-method="shutdown" />

<bean id="config" class="com.hazelcast.config.Config">
  <constructor-arg type="java.lang.String" value="HZ.INSTANCE" />
</bean>

<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast" factory-
method="newHazelcastInstance">
  <constructor-arg type="com.hazelcast.config.Config" ref="config"/>
</bean>
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route id="testHazelcastInstanceBeanRefPut">
    <from uri="direct:testHazelcastInstanceBeanRefPut"/>
    <setHeader headerName="CamelHazelcastOperationType">
      <constant>put</constant>
    </setHeader>
    <to uri="hazelcast-map:testmap?hazelcastInstanceName=HZ.INSTANCE"/>
  </route>
```

```
    <route id="testHazelcastInstanceBeanRefGet">
      <from uri="direct:testHazelcastInstanceBeanRefGet" />
      <setHeader headerName="CamelHazelcastOperationType">
        <constant>get</constant>
      </setHeader>
      <to uri="hazelcast-map:testmap?hazelcastInstanceName=HZ.INSTANCE"/>
      <to uri="seda:out" />
    </route>
</camelContext>
```

122.2.2. インスタンス別

```
<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast"
    factory-method="newHazelcastInstance" />
<bean id="hazelcastLifecycle" class="com.hazelcast.core.LifecycleService"
    factory-bean="hazelcastInstance" factory-method="getLifecycleService"
    destroy-method="shutdown" />

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route id="testHazelcastInstanceBeanRefPut">
      <from uri="direct:testHazelcastInstanceBeanRefPut"/>
      <setHeader headerName="CamelHazelcastOperationType">
        <constant>put</constant>
      </setHeader>
      <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
    </route>

    <route id="testHazelcastInstanceBeanRefGet">
      <from uri="direct:testHazelcastInstanceBeanRefGet" />
      <setHeader headerName="CamelHazelcastOperationType">
        <constant>get</constant>
      </setHeader>
      <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
      <to uri="seda:out" />
    </route>
</camelContext>
```

122.3. HAZELCAST インスタンスを OSGI サービスとして公開する

OSGI コンテナーで操作していて、同じコンテナー内のすべてのバンドルで hazelcast の 1 つのインスタンスを使用する場合。キャッシュを使用してインスタンスを OSGI サービスおよびバンドルとして公開できます。必要なのは、hazelcast エンドポイントでサービスを参照することです。

122.3.1. バンドル A インスタンスを作成し、OSGI サービスとして公開します

```
<bean id="config" class="com.hazelcast.config.FileSystemXmlConfig">
    <argument type="java.lang.String" value="${hazelcast.config}"/>
</bean>

<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast" factory-
method="newHazelcastInstance">
    <argument type="com.hazelcast.config.Config" ref="config"/>
```

```
</bean>

<!-- publishing the hazelcastInstance as a service -->
<service ref="hazelcastInstance" interface="com.hazelcast.core.HazelcastInstance" />
```

122.3.2. バンドル B はインスタンスを使用します

```
<!-- referencing the hazelcastInstance as a service -->
<reference ref="hazelcastInstance" interface="com.hazelcast.core.HazelcastInstance" />

<camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route id="testHazelcastInstanceBeanRefPut">
        <from uri="direct:testHazelcastInstanceBeanRefPut"/>
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>put</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
    </route>

    <route id="testHazelcastInstanceBeanRefGet">
        <from uri="direct:testHazelcastInstanceBeanRefGet" />
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>get</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
        <to uri="seda:out" />
    </route>
</camelContext>
```

第123章 HAZELCAST ATOMIC NUMBER コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast atomic number コンポーネントは、Hazelcast atomic number にアクセスできる Camel Hazelcast コンポーネントの 1 つです。atomic number は、単純にグリッド全体の数 (長い) を提供するオブジェクトです。

このエンドポイントのコンシューマーはありません!

123.1. オプション

Hazelcast Atomic Number コンポーネントは、以下に示す 3 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Atomic Number エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-atomicvalue:cacheName
```

パスおよびクエリーパラメーターを使用します。

123.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

123.1.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
defaultOperation (producer)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation
hazelcastInstance (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

123.2. ATOMIC NUMBER PRODUCER - TO("HAZELCAST-ATOMICVALUE:FOO")

このプロデューサの操作は次のとおりです。

リクエストメッセージのヘッダー変数:

名前	タイプ	説明
CamelHazelcastOperationType	**String**	有効な値は次のとおりです: setvalue、get、increase、decrew、destroy

123.2.1. セット のサンプル:

Java DSL の場合

```
from("direct:set")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.SET_VALUE))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:set" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>setvalue</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

メッセージボディー内に設定する値を指定します (ここでは値は 10 です):
template.sendBody("direct:set", 10);

123.2.2. get のサンプル:

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:get" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>get</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

long body = template.requestBody("direct:get", null, Long.class); で数値を取得できます。

123.2.3. increment のサンプル:

Java DSL の場合

```
from("direct:increment")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.INCREMENT))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:increment" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>increment</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

実際の値 (インクリメント後) は、メッセージボディー内に提供されます。

123.2.4. decrement のサンプル:

Java DSL の場合

```
from("direct:decrement")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DECREMENT))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:decrement" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>decrement</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

実際の値 (デクリメント後) は、メッセージボディー内に提供されます。

123.2.5. destroy のサンプル

Java DSL の場合

```
from("direct:destroy")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DESTROY))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL:

```
<route>
```

```
    <from uri="direct:destroy" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
      <constant>destroy</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

第124章 HAZELCAST INSTANCE コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast instance コンポーネントは、キャメル Hazelcast コンポーネントの 1 つで、クラスター内のキャッシュインスタンスの参加/脱退イベントを使用できます。Hazelcast は 1 つのサーバーノードでは意味がありますが、クラスター化された環境では非常に強力です。

このエンドポイントはプロデューサーを提供しません!

124.1. オプション

Hazelcast Instance コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast インスタンスエンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-instance:cacheName
```

パスおよびクエリーパラメーターを使用します。

124.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	***必須***: キャッシュの名前		String

124.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
defaultOperation (consumer)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation
hazelcastInstance (consumer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (consumer)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

124.2. インスタンスコンシューマー - FROM ("HAZELCAST-INSTANCE:FOO")

> The instance consumer fires if a new cache instance will join or leave the cluster.

サンプルは次のとおりです。

```
fromF("hazelcast-%sfoo", HazelcastConstants.INSTANCE_PREFIX)
.log("instance...")
.choice()
    .when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
        .to("mock:added")
    .otherwise()
        .log("...removed")
        .to("mock:removed");
```

各イベントは、メッセージヘッダー内に次の情報を提供します。

レスポンスメッセージ内のヘッダー変数:

名前	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間 (ミリ秒)
CamelHazelcastListenerType	**String**	ここでコンシューマーが設定するマップ instancelistener
CamelHazelcastListenerAction	**String**	イベントのタイプ - ここで **追加** または **削除** されます。
CamelHazelcastInstanceHost	**String**	インスタンスのホスト名
CamelHazelcastInstancePort	**Integer**	インスタンスのポート番号

第125章 HAZELCAST LIST コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast List コンポーネントは、Hazelcast 分散リストにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

125.1. オプション

Hazelcast List コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast List エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-list:cacheName
```

パスおよびクエリーパラメーターを使用します。

125.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

125.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

125.2. リストプロデューサー – TO (“HAZELCAST-LIST:FOO”)

リストプロデューサーは 7 つの操作を提供します: * add * addAll * set * get * removevalue * removeAll * clear

125.2.1. add のサンプル:

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

125.2.2. get のサンプル:

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX)
.to("seda:out");
```

125.2.3. setvalue のサンプル:

```
from("direct:set")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.SET_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

125.2.4. removevalue のサンプル:

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

CamelHazelcastObjectIndex ヘッダーはインデックス作成の目的で使用されることに注意してください。

125.3. コンシューマーのリスト – FROM (“HAZELCAST-LIST:FOO”)

リストコンシューマーは 2 つの操作を提供します: * 追加 * 削除

```
fromF("hazelcast-%smm", HazelcastConstants.LIST_PREFIX)
    .log("object...")
    .choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
            .log("...added")
                    .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))

            .log("...removed")
                    .to("mock:removed")
                .otherwise()
                    .log("fail!");
```

第126章 HAZELCAST マップコンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast Map コンポーネントは、Hazelcast 分散マップにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

126.1. オプション

Hazelcast Map コンポーネントは、以下に示す 3個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Map エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-map:cacheName
```

パスおよびクエリーパラメーターを使用します。

126.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

126.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

126.2. マップキャッシュプロデューサー - TO ("HAZELCAST-MAP:FOO")

マップに値を格納する場合は、マップキャッシュプロデューサーを使用できます。

マップキャッシュプロデューサーは、**CamelHazelcastOperationType** ヘッダーで指定された次の操作を提供します。

- put
- putIfAbsent
- get
- getAll
- keySet
- containsKey
- containsValue
- delete
- update
- query
- clear

- evict
- evictAll

すべての操作は、hazelcast.operation.type ヘッダー変数内で提供されます。Java DSL では、**org.apache.camel.component.hazelcast.HazelcastOperation** の定数を使用できます。

リクエストメッセージのヘッダー変数:

名前	タイプ	説明
CamelHazelcastOperationType	**String**	すでに説明したとおりです。
CamelHazelcastObjectId	**String**	キャッシュ内でオブジェクトを保存/検索するためのオブジェクト ID (クエリー操作には必要ありません)

put および putIfAbsent 操作は、エビクションメカニズムを提供します。

名前	タイプ	説明
CamelHazelcastObjectTtlValue	**Integer**	TTL の値。
CamelHazelcastObjectTtlUnit	**java.util.concurrent.TimeUnit**	時間単位の値 (DAYS / HOURS / MINUTES / ....

サンプルは次の方法で呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|update|delete|query|evict]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

126.2.1. put のサンプル:

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:put" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>put</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

エビクションを使用した **put** のサンプル:

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.setHeader(HazelcastConstants.TTL_VALUE, constant(Long.valueOf(1)))
.setHeader(HazelcastConstants.TTL_UNIT, constant(TimeUnit.MINUTES))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:put" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>put</constant>
    </setHeader>
    <setHeader headerName="HazelcastConstants.TTL_VALUE">
        <simple resultType="java.lang.Long">1</simple>
    </setHeader>
    <setHeader headerName="HazelcastConstants.TTL_UNIT">
        <simple resultType="java.util.concurrent.TimeUnit">TimeUnit.MINUTES</simple>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

126.2.2. get のサンプル:

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.to("seda:out");
```

Spring DSL:

```
<route>
    <from uri="direct:get" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>get</constant>
```

```
    </setHeader>
    <to uri="hazelcast-map:foo" />
    <to uri="seda:out" />
</route>
```

126.2.3. update のサンプル:

Java DSL の場合

```
from("direct:update")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.UPDATE))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:update" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>update</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

126.2.4. delete のサンプル:

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:delete" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>delete</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

126.2.5. query のサンプル

Java DSL の場合

```
from("direct:query")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.QUERY))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.to("seda:out");
```

Spring DSL:

```
<route>
    <from uri="direct:query" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>query</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
    <to uri="seda:out" />
</route>
```

クエリー操作の場合、Hazelcast は、分散マップをクエリーするための SQL のような構文を提供します。

```
String q1 = "bar > 1000";
template.sendBodyAndHeader("direct:query", null, HazelcastConstants.QUERY, q1);
```

126.3. マップキャッシュコンシューマー - FROM ("HAZELCAST-MAP:FOO")

Hazelcast は、データグリッドでイベントリスナーを提供します。キャッシュが操作される場合に通知を受け取りたい場合は、マップコンシューマーを使用できます。4 つのイベントがあります: **put**、**update**、**delete**、および **envict** です。イベントタイプは、**hazelcast.listener.action** ヘッダー変数に格納されます。マップコンシューマーは、これらの変数内にいくつかの追加情報を提供します。

レスポンスメッセージ内のヘッダー変数:

名前	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間 (ミリ秒)
CamelHazelcastListenerType	**String**	マップのコンシューマーがここで設定する cachelistener
CamelHazelcastListenerAction	**String**	イベントのタイプ - ここでは、**追加**、**更新**、**削除**、**削除** を行います。
CamelHazelcastObjectId	**String**	オブジェクトの oid

名前	タイプ	説明
CamelHazelcastCacheName	**String**	キャッシュの名前 - 例: foo
CamelHazelcastCacheType	**String**	キャッシュのタイプ - here map

オブジェクト値は、メッセージボディー内の **put** および **update** アクション内に格納されます。

サンプルは次のとおりです。

```
fromF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.log("object...")
.choice()
    .when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
        .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVICTED))

        .log("...envicted")
        .to("mock:envicted")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.UPDATED))
        .log("...updated")
        .to("mock:updated")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))

        .log("...removed")
        .to("mock:removed")
    .otherwise()
        .log("fail!");
```

第127章 HAZELCAST MULTIMAP コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast Multimap コンポーネントは、Hazelcast 分散マルチマップにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

127.1. オプション

Hazelcast Multimap コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Multimap エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-multimap:cacheName
```

パスおよびクエリーパラメーターを使用します。

127.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

127.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

127.2. MULTIMAP CACHE PRODUCER - TO("HAZELCAST-MULTIMAP:FOO")

multimap は、n 個の値を 1 つのキーに格納できるキャッシュです。multimap プロデューサーは、4 つの操作 (put、get、removevalue、delete) を提供します。

リクエストメッセージのヘッダー変数:

名前	タイプ	説明
CamelHazelcastOperationType	**String**	有効な値は次のとおりです: put、get、removevalue、delete **Camel 2.16 以降:** clear。
CamelHazelcastObjectId	**String**	キャッシュ内でオブジェクトを保存/検索するためのオブジェクト ID

127.2.1. put のサンプル:

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.to(String.format("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX));
```

Spring DSL:

```
<route>
    <from uri="direct:put" />
    <log message="put.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>put</constant>
    </setHeader>
    <to uri="hazelcast-multimap:foo" />
</route>
```

127.2.2. removevalue のサンプル:

Java DSL の場合

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:removevalue" />
    <log message="removevalue..."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>removevalue</constant>
    </setHeader>
    <to uri="hazelcast-multimap:foo" />
</route>
```

値を削除するには、メッセージボディー内で削除する値を指定する必要があります。マルチマップオブジェクト \{**key: "4711" values: { "my-foo", "my-bar"**}} がある場合、メッセージボディーの中に my-foo を入れて my-foo 値を削除する必要があります。

127.2.3. get のサンプル:

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.to("seda:out");
```

Spring DSL:

```
<route>
    <from uri="direct:get" />
```

```
    <log message="get.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>get</constant>
    </setHeader>
    <to uri="hazelcast-multimap:foo" />
    <to uri="seda:out" />
</route>
```

127.2.4. delete のサンプル:

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:delete" />
    <log message="delete.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>delete</constant>
    </setHeader>
    <to uri="hazelcast-multimap:foo" />
</route>
```

以下を使用して、テストクラスでそれらを呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|removevalue|delete]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

127.3. MULTIMAP CACHE CONSUMER - FROM("HAZELCAST-MULTIMAP:FOO")

マルチマップキャッシュの場合、このコンポーネントはマップキャッシュコンシューマーと同じリスナー/変数を提供します (更新および環境リスナーを除く)。唯一の違いは、URI 内の **マルチマップ** 接頭辞です。以下にサンプルを示します。

```
fromF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.log("object...")
.choice()
    .when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
            .to("mock:added")

//.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVICTED))

        //        .log("...envicted")
        //        .to("mock:envicted")
```

```
.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))

        .log("...removed")
        .to("mock:removed")
    .otherwise()
        .log("fail!");
```

レスポンスメッセージ内のヘッダー変数:

名前	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間 (ミリ秒)
CamelHazelcastListenerType	**String**	マップのコンシューマーがここで設定する cachelistener
CamelHazelcastListenerAction	**String**	イベントのタイプ - ここで **追加** および **削除** されます (そしてすぐに **取り除かれます**)
CamelHazelcastObjectId	**String**	オブジェクトの oid
CamelHazelcastCacheName	**String**	キャッシュの名前 - 例: foo
CamelHazelcastCacheType	**String**	キャッシュのタイプ - ここでは multimap

第128章 HAZELCAST QUEUE コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast Queue コンポーネントは、Hazelcast 分散キューにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

128.1. オプション

Hazelcast Queue コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast キューエンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-queue:cacheName
```

パスおよびクエリーパラメーターを使用します。

128.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

128.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

128.2. キュープロデューサー – TO (“HAZELCAST-QUEUE:FOO”)

キュープロデューサーは 10 の操作を提供します。* Add * put * poll * peek * offer * remove value * remaining capacity * remove all * remove if * drain to * take * retain all

128.2.1. add のサンプル:

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.2. put のサンプル:

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.3. poll のサンプル:

```
from("direct:poll")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.POLL))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.4. peek のサンプル:

```
from("direct:peek")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PEEK))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.5. offer のためのサンプル:

```
from("direct:offer")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.OFFER))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.6. removevalue のサンプル:

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.7. remaining capacity のサンプル:

```
from("direct:remaining-capacity").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMAINING_CAPACITY)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.8. remove all のサンプル:

```
from("direct:removeAll").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMOVE_ALL)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.9. remove if のサンプル:

```
from("direct:removeIf").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMOVE_IF)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.10. drain to のサンプル:

```
from("direct:drainTo").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.DRAIN_TO)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.11. take のサンプル:

```
from("direct:take").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.TAKE)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.12. retain all のサンプル:

-

```
from("direct:retainAll").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.RETAIN_ALL)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.3. キューコンシューマー – FROM(“HAZELCAST-QUEUE:FOO”)

キューコンシューマーには、次の 2 つの異なるモードがあります。

- Poll
- Listen

Poll モードのサンプル

```
fromF("hazelcast-%sfoo?queueConsumerMode=Poll",
HazelcastConstants.QUEUE_PREFIX)).to("mock:result");
```

このようにして、コンシューマーはキューをポーリングし、タイムアウト後にキューの先頭または null を返します。

代わりに Listen モードでは、コンシューマーはキューのイベントをリッスンします。

Listen モードのキューコンシューマーは、次の 2 つの操作を提供します: * 追加 * 削除

Listen モードのサンプル

```
fromF("hazelcast-%smm", HazelcastConstants.QUEUE_PREFIX)
   .log("object...")
   .choice()
    .when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
            .log("...added")
        .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))

        .log("...removed")
        .to("mock:removed")
    .otherwise()
        .log("fail!");
```

第129章 HAZELCAST REPLICATED MAP コンポーネント

Camel バージョン 2.16 以降で利用可能

Hazelcast instance コンポーネントは、キャメル Hazelcast コンポーネントの 1 つで、クラスター内のキャッシュインスタンスの参加/脱退イベントを使用できます。レプリケートされたマップは、データパーティションのない、一貫性の低い分散キー値データ構造です。

129.1. オプション

Hazelcast Replicated Map コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Replicated Map エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-replicatedmap:cacheName
```

パスおよびクエリーパラメーターを使用します。

129.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

129.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

129.2. レプリケートされたマップキャッシュプロデューサー

レプリケートマッププロデューサーは 4 つの操作を提供します: * put * get * delete * clear

リクエストメッセージのヘッダー変数:

名前	タイプ	説明
CamelHazelcastOperationType	**String**	有効な値: put、get、removevalue、delete
CamelHazelcastObjectId	**String**	キャッシュ内でオブジェクトを保存/検索するためのオブジェクト ID

129.2.1. put のサンプル:

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.to(String.format("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX));
```

Spring DSL:

```
<route>
    <from uri="direct:put" />
    <log message="put.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>put</constant>
    </setHeader>
    <to uri="hazelcast-replicatedmap:foo" />
</route>
```

129.2.2. get のサンプル:

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX)
.to("seda:out");
```

Spring DSL:

```
<route>
    <from uri="direct:get" />
    <log message="get.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>get</constant>
    </setHeader>
    <to uri="hazelcast-replicatedmap:foo" />
    <to uri="seda:out" />
</route>
```

129.2.3. delete のサンプル:

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX);
```

Spring DSL:

```
<route>
    <from uri="direct:delete" />
    <log message="delete.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
```

```
        <constant>delete</constant>
    </setHeader>
    <to uri="hazelcast-replicatedmap:foo" />
</route>
```

以下を使用して、テストクラスでそれらを呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|delete|clear]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

129.3. レプリケートされたマップキャッシュコンシューマー

マルチマップキャッシュの場合、このコンポーネントはマップキャッシュコンシューマーと同じリスナー/変数を提供します (更新および環境リスナーを除く)。唯一の違いは、URI 内の **multimap** 接頭辞です。以下にサンプルを示します。

```
fromF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.log("object...")
.choice()
    .when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
            .to("mock:added")

//.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVICTED))

        //      .log("...envicted")
        //      .to("mock:envicted")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))

            .log("...removed")
            .to("mock:removed")
        .otherwise()
            .log("fail!");
```

レスポンスメッセージ内のヘッダー変数:

名前	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間 (ミリ秒)
CamelHazelcastListenerType	**String**	マップのコンシューマーがここで設定する cachelistener

名前	タイプ	説明
CamelHazelcastListenerAction	**String**	イベントのタイプ - ここで **追加** および **削除** されます (そしてすぐに **取り除かれます**)
CamelHazelcastObjectId	**String**	オブジェクトの oid
CamelHazelcastCacheName	**String**	キャッシュの名前 - 例: foo
CamelHazelcastCacheType	**String**	キャッシュのタイプ - ここでは replicadmap

第130章 HAZELCAST RINGBUFFER コンポーネント

Camel バージョン 2.16 以降で利用可能

Camel 2.16 から利用可能

Hazelcast ringbuffer コンポーネントは、Hazelcast ringbuffer にアクセスできる Camel Hazelcast コンポーネントの 1 つです。Ringbuffer は、データがリング状の構造に格納される分散データ構造です。これは、一定の容量を持つ円形配列と考えることができます。

130.1. オプション

Hazelcast Ringbuffer コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Ringbuffer エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-ringbuffer:cacheName
```

パスおよびクエリーパラメーターを使用します。

130.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

130.1.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
defaultOperation (producer)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation
hazelcastInstance (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

130.2. RINGBUFFER キャッシュプロデューサー

ringbuffer プロデューサーは 5 つの操作を提供します: * add * readonceHead * readonceTail * remainingCapacity * capacity

リクエストメッセージのヘッダー変数:

名前	タイプ	説明
CamelHazelcastOperationType	**String**	有効な値: put、get、removevalue、delete
CamelHazelcastObjectId	**String**	キャッシュ内でオブジェクトを保存/検索するためのオブジェクト ID

130.2.1. put のサンプル:

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.to(String.format("hazelcast-%sbar", HazelcastConstants.RINGBUFFER_PREFIX));
```

Spring DSL:

```
<route>
    <from uri="direct:put" />
    <log message="put.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" -->
    <setHeader headerName="hazelcast.operation.type">
        <constant>add</constant>
    </setHeader>
    <to uri="hazelcast-ringbuffer:foo" />
</route>
```

130.2.2. readonce from head のサンプル:

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.READ_ONCE_HEAD))
.toF("hazelcast-%sbar", HazelcastConstants.RINGBUFFER_PREFIX)
.to("seda:out");
```

第131章 HAZELCAST SEDA コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast SEDA コンポーネントは、Hazelcast BlockingQueue へのアクセスを可能にする Camel Hazelcast コンポーネントの 1 つです。SEDA コンポーネントは、提供されている残りのコンポーネントとは異なります。コアの SEDA コンポーネントと同様に、非同期 SEDA アーキテクチャーをサポートするために作業キューを実装します。

131.1. オプション

Hazelcast SEDA コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast SEDA エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-seda:cacheName
```

パスおよびクエリーパラメーターを使用します。

131.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

131.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

131.2. SEDA プロデューサー – TO (“HAZELCAST-SEDA:FOO”)

SEDA プロデューサは操作を提供しません。指定したキューにのみデータを送信します。

Java DSL:

```
from("direct:foo")
.to("hazelcast-seda:foo");
```

Spring DSL :

```
<route>
  <from uri="direct:start" />
  <to uri="hazelcast-seda:foo" />
</route>
```

131.3. SEDA コンシューマー – FROM(“HAZELCAST-SEDA:FOO”)

SEDA コンシューマーは操作を提供しません。指定したキューからのみデータを取得します。

Java DSL:

```
from("hazelcast-seda:foo")
.to("mock:result");
```

Spring DSL:

```
<route>
  <from uri="hazelcast-seda:foo" />
  <to uri="mock:result" />
</route>
```

第132章 HAZELCAST SET コンポーネント

Camel バージョン 2.7 以降で利用可能

Hazelcast Set コンポーネントは、Hazelcast 分散セットにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

132.1. オプション

Hazelcast Set コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Set エンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-set:cacheName
```

パスおよびクエリーパラメーターを使用します。

132.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

132.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

第133章 HAZELCAST TOPIC コンポーネント

Camel バージョン 2.15 以降で利用可能

Hazelcast Topic コンポーネントは、Hazelcast 分散トピックにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

133.1. オプション

Hazelcast Topic コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	hazelcast モードは、どの種類のインスタンスを使用する必要があるかを示しています。モードを指定しないと、ノードモードがデフォルトになります。	node	String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast トピックエンドポイントは、URI 構文を使用して設定されます。

```
hazelcast-topic:cacheName
```

パスおよびクエリーパラメーターを使用します。

133.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

133.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultOperation (Common)	操作ヘッダーが提供されていない場合に、使用するデフォルトの操作を指定します。		HazelcastOperation

名前	説明	デフォルト	タイプ
hazelcastInstance (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (Common)	hazelcast エンドポイントに使用できる hazelcast インスタンスの参照名。インスタンス参照を指定しない場合、camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		String
reliable (Common)	エンドポイントが信頼できる Topic 構造体を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	Poll モードでの Queue コンシューマーのポーリングタイムアウトを定義します。	10000	long
poolSize (consumer)	Queue Consumer Executor のプールサイズの定義	1	int
queueConsumerMode (consumer)	Queue Consumer モードの定義：Listen または Poll	Listen	HazelcastQueueConsumer モード
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
concurrentConsumers (seda)	SEDA キューからの同時コンシューマーのポーリングを使用します。	1	int

名前	説明	デフォルト	タイプ
onErrorDelay (seda)	エラーが発生した後、コンシューマーがポーリングを継続するまでの時間（ミリ秒単位）。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
transacted (seda)	true に設定すると、コンシューマーはトランザクションモードで実行し、処理の完了時に発生するトランザクションがコミットされた場合にのみ seda キュー内のメッセージが削除されます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディにシリアライズ可能なオブジェクトが含まれていない場合、それらはスキップされます。	false	boolean

133.2. トピックプロデューサー – TO (“HAZELCAST-TOPIC:FOO”)

トピックプロデューサーが提供する操作は 1 つだけです (パブリッシュ)。

133.2.1. publish のサンプル:

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUBLISH))
.toF("hazelcast-%sbar", HazelcastConstants.PUBLISH_OPERATION);
```

133.3. トピックコンシューマー – FROM(“HAZELCAST-TOPIC:FOO”)

トピックコンシューマーは、1 つの操作 (受信) のみを提供します。このコンポーネントは、トピックに関して予想されるように複数の消費をサポートすることになっているため、同じ hazelcast トピックで必要な数のコンシューマーを自由に使用できます。

```
fromF("hazelcast-%sfoo", HazelcastConstants.TOPIC_PREFIX)
  .choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.RECEIVED))

      .log("...message received")
    .otherwise()
      .log("...this should never have happened")
```

第134章 HBASE コンポーネント

Camel バージョン 2.10 以降で利用可能

このコンポーネントは、Apache HBase のべき等のリポジトリー、プロデューサー、およびコンシューマーを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hbase</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

134.1. APACHE HBASE の概要

HBase は、Google の Bigtable: A Distributed Storage System for Structured Data をモデルにした、オープンソースで分散型のバージョン管理された列指向のストアです。ビッグデータへのランダムなリアルタイムの読み取り/書き込みアクセスが必要な場合は、HBase を使用できます。詳細については、Apache HBase を参照してください。

134.2. CAMEL と HBASE

camel ルート内でデータストアを使用する場合、camel メッセージをデータストアに保存する方法を指定するという課題が常にあります。ドキュメントベースのストアでは、メッセージボディーをドキュメントに直接マップできるため、作業がより簡単になります。リレーショナルデータベースでは、ORM ソリューションを使用して、プロパティーを列などにマップできます。列ベースのストアでは、その種のマッピングを実行する標準的な方法がないため、より困難です。

HBase はさらに 2 つの課題を追加します。

- HBase は列をファミリーにグループ化するため、名前付け規則を使用してプロパティーを列にマッピングするだけでは十分ではありません。
- HBase には型の概念がありません。つまり、すべてを byte[] として格納し、byte[] が文字列、数値、シリアル化された Java オブジェクト、または単にバイナリーデータを表しているかどうかはわかりません。

これらの課題を克服するために、camel-hbase はメッセージヘッダーを使用して、メッセージの HBase 列へのマッピングを指定します。また、HBase データをモデル化し、xml/json などとの間で簡単に変換できる、camel-hbase が提供するいくつかのクラスを使用する機能も提供します。
最後に、ユーザーが独自のマッピングストラテジーを実装して使用できるようにします。

マッピングストラテジーに関係なく、camel-hbase はメッセージを org.apache.camel.component.hbase.model.HBaseData オブジェクトに変換し、そのオブジェクトを内部操作に使用します。

134.3. コンポーネントの設定

HBase コンポーネントには、カスタム HBaseConfiguration オブジェクトをプロパティーとして提供するか、クラスパスにある HBase 関連リソースに基づいて HBase 設定オブジェクトを独自に作成できます。

```
<bean id="hbase" class="org.apache.camel.component.hbase.HBaseComponent">
   <property name="configuration" ref="config"/>
</bean>
```

コンポーネントに設定オブジェクトが提供されていない場合、コンポーネントは設定オブジェクトを作成します。作成された設定は、hbase-site.xml ファイルのクラスパスを検索し、そこから設定を引き出します。HBase クライアントの設定方法の詳細については、HBase クライアントの設定と依存関係 を参照してください。

134.4. HBASE プロデューサー

前述のように、camel は HBase のプロデューサーエンドポイントを提供します。これにより、camel ルートを使用して HBase からデータを保存、削除、取得、またはクエリーできます。

```
hbase://table[?options]
```

ここで、**table** はテーブル名です。

サポートされている演算は次のとおりです。

- Put
- Get
- Delete
- スキャン

134.4.1. サポートされている URI オプション

HBase コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有設定を使用するには		設定
poolMaxSize (Common)	HTable プール内のテーブルごとに保持する参照の最大数。デフォルト値は 10 です。	10	int
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HBase エンドポイントは、URI 構文を使用して設定されます。

```
hbase:tableName
```

パスおよびクエリーパラメーターを使用します。

134.4.2. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
tableName	**必須** テーブルの名前		String

134.4.3. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
cellMappingStrategyFactory (common)	セルのマッピングを担当するカスタム CellMappingStrategyFactory を使用する場合。		CellMappingStrategy ファクトリー
filters (common)	使用するフィルターのリスト。		List
mappingStrategyClassName (common)	カスタムマッピング戦略の実装のクラス名。		String
mappingStrategyName (common)	Camel メッセージを HBase 列にマッピングするために使用するストラテジー。サポートされている値: ヘッダーまたはボディー。		String
rowMapping (common)	キー/値を Map から HBaseRow にマップします。次のキーがサポートされています: rowId - 行の ID。行は通常、エクスチェンジごとに変更されるため、これは使用が制限されています。rowType - 行 ID を変換するタイプ。サポートされている操作: CamelHBaseScan。family - 列ファミリー。複数の列を参照するための数値接尾辞をサポートします。qualifier - 列修飾子。複数の列を参照するための数値接尾辞をサポートします。値 - 値。複数の列を参照するための数値接尾辞をサポートします。valueType - 値の型。複数の列を参照するための数値接尾辞をサポートします。サポートされている操作: CamelHBaseGet、および CamelHBaseScan。		Map
rowModel (common)	各行をモデル化する方法を記述する org.apache.camel.component.hbase.model.HBaseRow のインスタンス		HBaseRow
userGroupInformation (common)	kerberos の使用など、HBase と通信するための特権を定義します。		UserGroupInformation

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の数を使用して無制限として無効にします。		int
operation (consumer)	実行する HBase 操作		String
remove (consumer)	オプションが true の場合、Camel HBase コンシューマーは処理する行を削除します。	true	boolean
removeHandler (consumer)	行が削除されるときに実行されるカスタム HBaseRemoveHandler を使用するには。		HBaseRemoveHandler
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
maxResults (producer)	スキャンする行の最大数。	100	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

134.4.4. Put 操作

HBase は列ベースのストアであり、特定の行の特定の列にデータを格納できます。列はファミリーにグループ化されるため、列を指定するには、列ファミリーとその列の修飾子を指定する必要があります。特定の列にデータを格納するには、列と行の両方を指定する必要があります。

camel ルートから HBase にデータを格納する最も単純なシナリオは、メッセージボディーの一部を指定された HBase 列に格納することです。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <to uri="hbase:mytable?
operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier"/>
        </route>
```

上記のルートは、メッセージボディーに id および value プロパティーを持つオブジェクトが含まれていることを前提としており、id で指定された行の HBase 列 myfamily:myqualifier に value の内容を格納します。複数の列/値のペアを指定する必要がある場合は、追加の列マッピングを指定するだけで済みます。RowId2、RowId3、RowId4 など、2 番目のヘッダー以降の数字を使用する必要があることに注意してください。最初のヘッダーのみ番号 1 がありません。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row 1st column -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Row 2nd column -->
            <setHeader headerName="CamelHBaseRowId2">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value for 1st column -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <!-- Set the HBase Value for 2nd column -->
            <setHeader headerName="CamelHBaseValue2">
                <el>${in.body.othervalue}</el>
            </setHeader>
            <to uri="hbase:mytable?
operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier&amp;family2=myfamily&a
p;qualifier2=myqualifier2"/>
        </route>
```

uri オプション、メッセージヘッダー、または両方の組み合わせを使用できることを覚えておくことが重要です。uri の一部として定数を指定し、ヘッダーとして動的な値を指定することをお勧めします。ヘッダーと uri の両方で何かが定義されている場合は、ヘッダーが使用されます。

134.4.5. Get 操作

Get 操作は、指定された HBase 行から 1 つ以上の値を取得するために使用される操作です。取得する値を指定するには、それらを uri の一部またはメッセージヘッダーとして指定するだけです。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row of the Get -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <to uri="hbase:mytable?
operation=CamelHBaseGet&amp;family=myfamily&amp;qualifier=myqualifier&amp;valueType=java.lang
Long"/>
            <to uri="log:out"/>
        </route>
```

上記の例では、get 操作の結果は CamelHBaseValue という名前のヘッダーとして保存されます。

134.4.6. Delete 操作

また、camel-hbase を使用して HBase の削除操作を実行することもできます。削除操作は行全体を削除します。指定する必要があるのは、メッセージヘッダーの一部として 1 つ以上の行だけです。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row of the Get -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseDelete"/>
        </route>
```

134.4.7. スキャン操作。

スキャン操作は、HBase のクエリーに相当します。スキャン操作を使用して、複数の行を取得できます。結果の一部にする列を指定し、値をオブジェクトに変換する方法を指定するには、uri オプションまたはヘッダーを使用できます。

```
        <route>
            <from uri="direct:in"/>
            <to uri="hbase:mytable?
operation=CamelHBaseScan&amp;family=myfamily&amp;qualifier=myqualifier&amp;valueType=java.lan
g.Long&amp;rowType=java.lang.String"/>
            <to uri="log:out"/>
        </route>
```

この場合、結果を制限するためのフィルターのリストも指定する必要がある可能性があります。URI の一部としてフィルターのリストを指定すると、camel は **すべての** フィルターを満たす行のみを返します。
メッセージの一部である情報を認識するフィルターを持つために、camel は ModelAwareFilter を定義します。これにより、メッセージとマッピング戦略によって定義されたモデルをフィルターで考慮することができます。
ModelAwareFilter を使用すると、camel-hbase は選択されたマッピングストラテジーをメッセージに適用し、マッピングをモデル化するオブジェクトを作成し、そのオブジェクトをフィルターに渡します。

たとえば、メッセージヘッダーを条件として使用してスキャンを実行するには、以下に示すように ModelAwareColumnMatchingFilter を使用できます。

```
        <route>
            <from uri="direct:scan"/>
            <!-- Set the Criteria -->
            <setHeader headerName="CamelHBaseFamily">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier">
                <constant>first</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue">
                <el>in.body.firstName</el>
            </setHeader>
            <setHeader headerName="CamelHBaseFamily2">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier2">
                <constant>last</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue2">
                <el>in.body.lastName</el>
            </setHeader>
            <!-- Set additional fields that you want to be return by skipping value -->
            <setHeader headerName="CamelHBaseFamily3">
                <constant>address</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier3">
                <constant>country</constant>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseScan&amp;filters=#myFilterList"/>
        </route>

        <bean id="myFilters" class="java.util.ArrayList">
            <constructor-arg>
                <list>
                    <bean
class="org.apache.camel.component.hbase.filters.ModelAwareColumnMatchingFilter"/>
                </list>
            </constructor-arg>
        </bean>
```

上記のルートは、pojo にプロパティー firstName と lastName がメッセージボディーとして渡されることを想定しており、これらのプロパティーを取得して、メッセージヘッダーの一部として追加します。デフォルトのマッピングストラテジーは、ヘッダーを HBase 列にマップするモデルオブジェクトを作成し、そのモデルに ModelAwareColumnMatchingFilter を渡します。フィルターは、モデルに一致する列を含まない行を除外します。例によるクエリーのようなものです。

134.5. HBASE コンシューマー

Camel HBase コンシューマーは、指定された HBase テーブルで繰り返しスキャンを実行し、メッセージの一部としてスキャン結果を返します。ヘッダーマッピング (デフォルト) またはボディーマッピングのいずれかを指定できます。後者は、メッセージボディーの一部として org.apache.camel.component.hbase.model.HBaseData を追加するだけです。

```
hbase://table[?options]
```

返される列とそのタイプを uri オプションの一部として指定できます。

```
hbase:mutable?
family=name&qualifer=first&valueType=java.lang.String&family=address&qualifer=number&valueType2=
ava.lang.Integer&rowType=java.lang.Long
```

上記の例では、指定されたフィールドで設定されるモデルオブジェクトが作成され、スキャン結果によってモデルオブジェクトに値が入力されます。最後に、マッピングストラテジーを使用して、このモデルを camel メッセージにマッピングします。

134.6. HBASE べき等リポジトリー

camel-hbase コンポーネントは、各メッセージが 1 回だけ処理されるようにする場合に使用できるべき等リポジトリーも提供します。HBase べき等リポジトリーは、テーブル、列ファミリー、および列修飾子で設定され、メッセージごとにそのテーブルに行を作成します。

```
HBaseConfiguration configuration = HBaseConfiguration.create();
HBaseIdempotentRepository repository = new HBaseIdempotentRepository(configuration,
tableName, family, qualifier);

from("direct:in")
  .idempotentConsumer(header("messageId"), repository)
  .to("log:out);
```

134.7. HBASE マッピング

デフォルトのマッピングストラテジーは **ヘッダー** と **ボディの** マッピングであることは前述しました。以下に、各マッピングストラテジーがどのように機能するかの詳細な例をいくつか示します。

134.7.1. HBase ヘッダーマッピングの例

ヘッダーマッピングはデフォルトのマッピングです。値 myvalue を HBase の行 myrow と列 myfamily:mycolum に入れるには、メッセージに次のヘッダーを含める必要があります。

ヘッダー	値
Camel HBase RowId	myrow
Camel HBase Family	myfamily
Camel HBase Qualifier	myqualifier

ヘッダー	値
Camel HBase Value	myvalue

異なる列および/または異なる行にさらに値を入れるには、ヘッダーのインデックスを接尾辞として付けた追加のヘッダーを指定できます。次に例を示します。

ヘッダー	値
Camel HBase RowId	myrow
Camel HBase Family	myfamily
Camel HBase Qualifier	myqualifier
Camel HBase Value	myvalue
Camel HBase RowId2	myrow2
Camel HBase Family2	myfamily
Camel HBase Qualifier2	myqualifier
Camel HBase Value2	myvalue2

get や scan などの取得操作の場合、データを変換する型を列ごとに指定することもできます。例:

ヘッダー	値
Camel HBase Family	myfamily
Camel HBase Qualifier	myqualifier
Camel HBase ValueType	Long

以下に示すように、すべてのメッセージで一定と見なされるボイラープレートヘッダーを回避するために、それらをエンドポイント uri の一部として指定することもできます。

134.7.2. ボディーマッピングの例

ボディーマッピング戦略を使用するには、uri の一部としてオプション mappingStrategy を指定する必要があります。次に例を示します。

```
hbase:mytable?mappingStrategyName=body
```

ボディーマッピングストラテジーを使用するには、ボディーに org.apache.camel.component.hbase.model.HBaseData のインスタンスが含まれている必要があります。t を構築することができます

```
HBaseData data = new HBaseData();
HBaseRow row = new HBaseRow();
row.setId("myRowId");
HBaseCell cell = new HBaseCell();
cell.setFamily("myfamily");
cell.setQualifier("myqualifier");
cell.setValue("myValue");
row.getCells().add(cell);
data.addRows().add(row);
```

上記のオブジェクトは、たとえば put 操作で使用でき、id myRowId の行を作成または更新し、値 myvalue を列 myfamily:myqualifier に追加します。
ボディマッピングストラテジーは、最初はあまり魅力的ではないように見えるかもしれません。ヘッダーマッピングストラテジーに対する利点は、HBaseData オブジェクトを xml/json との間で簡単に変換できることです。

134.8. その他の参考資料

- Polling Consumer
- Apache HBase

第135章 HDFS コンポーネント (非推奨)

Camel バージョン 2.8 以降で利用可能

hdfs コンポーネントを使用すると、HDFS ファイルシステムとの間でメッセージを読み書きできます。HDFS は、Hadoop の中心にある分散ファイルシステムです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hdfs</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

135.1. URI 形式

```
hdfs://hostname[:port][/path][?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=value&...**
パスは次のように処理されます。

1. コンシューマーとして、それがファイルの場合はファイルを読み取るだけで、それ以外の場合はディレクトリーを表す場合は、設定されたパターンを満たすパスの下にあるすべてのファイルをスキャンします。そのディレクトリーの下にあるすべてのファイルは、同じタイプである必要があります。

2. プロデューサーとして、少なくとも 1 つの分割ストラテジーが定義されている場合、パスはディレクトリーと見なされ、そのディレクトリーの下で、プロデューサーは設定された UuidGenerator を使用して名前が付けられた分割ごとに異なるファイルを作成します。

注記

通常モードで hdfs から消費する場合、ファイルはチャンクに分割され、チャンクごとにメッセージが生成されます。chunkSize オプションを使用して、チャンクのサイズを設定できます。hdfs から読み取り、file コンポーネントを使用して通常のファイルに書き込みたい場合は、fileMode=Append を使用して各チャンクを一緒に追加できます。

135.2. オプション

HDFS コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
jAASConfiguration (Common)	指定された設定を JAAS でのセキュリティーに使用するため。		設定

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HDFS エンドポイントは、URI 構文を使用して設定されます。

```
hdfs:hostName:port/path
```

パスおよびクエリーパラメーターを使用します。

135.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
hostname	使用するために **必要な** HDFS ホスト		String
port	使用する HDFS ポート	8020	int
path	**必須** 使用するディレクトリーパス		String

135.2.2. クエリーパラメーター (38 パラメーター)

名前	説明	デフォルト	タイプ
connectOnStartup (Common)	プロデューサー/コンシューマーの起動時に HDFS ファイルシステムに接続するかどうか。false の場合、接続はオンデマンドで作成されます。HDFS は 45 x 20 秒の再配信をハードコーディングしているため、接続を確立するのに最大 15 分かかる場合があることに注意してください。このオプションを false に設定すると、アプリケーションが起動し、最大 15 分間ブロックされなくなります。	true	boolean
fileSystemType (Common)	HDFS を使用せず、代わりにローカルの java.io.File を使用するには、LOCAL に設定します。	HDFS	HdfsFileSystemType
fileType (Common)	使用するファイルの種類。詳細については、さまざまなファイルタイプに関する Hadoop HDFS のドキュメントを参照してください。	NORMAL_FILE	HdfsFileType

名前	説明	デフォルト	タイプ
keyType (Common)	シーケンスまたはマップファイルの場合のキーのタイプ。	NULL	WritableType
owner (Common)	コンシューマーがファイルを取得するには、ファイルの所有者がこの所有者と一致する必要があります。それ以外の場合、ファイルはスキップされます。		String
valueType (Common)	シーケンスまたはマップファイルの場合のキーのタイプ	BYTES	WritableType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	ディレクトリースキャンの間隔 (ミリ秒)。	1000	long
initialDelay (consumer)	コンシューマーの場合、ディレクトリーのスキャンを開始するまでに待機する時間 (ミリ秒)。		long
pattern (consumer)	ディレクトリーのスキャンに使用されるパターン	*	String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
append (producer)	既存のファイルに追加します。すべての HDFS ファイルシステムが追加オプションをサポートしているわけではないことに注意してください。	false	boolean
overwrite (producer)	同名の既存ファイルを上書きするかどうか	true	boolean
blockSize (advanced)	HDFS ブロックのサイズ	67108864	long
bufferSize (advanced)	HDFS が使用するバッファーサイズ	4096	int
checkIdleInterval (advanced)	アイドルチェッカーバックグラウンドタスクを実行する頻度 (ミリ秒単位)。このオプションは、スプリッタ戦略が IDLE の場合にのみ使用されます。	500	int
chunkSize (advanced)	通常のファイルを読み取る場合、これはチャンクに分割され、チャンクごとにメッセージが生成されます。	4096	int
compressionCodec (advanced)	使用する圧縮コーデック	DEFAULT	HdfsCompressionCodec
compressionType (advanced)	使用する圧縮タイプ (デフォルトでは使用されていません)	NONE	CompressionType
openedSuffix (advanced)	ファイルが読み取り/書き込み用に開かれると、ファイルはこの接尾辞で名前が変更され、書き込みフェーズ中に読み取られないようにします。	opened	String
readSuffix (advanced)	ファイルが読み取られると、この接尾辞を使用して名前が変更され、再度読み取られることがなくなります。	read	String
replication (advanced)	HDFS レプリケーション係数	3	short

名前	説明	デフォルト	タイプ
splitStrategy (advanced)	現在のバージョンの Hadoop では、ファイルを追加モードで開くことは信頼性が低いため無効になっています。そのため、現時点では、新しいファイルを作成することしかできません。Camel HDFS エンドポイントは、この問題を次のように解決しようとします。分割戦略オプションが定義されている場合、hdfs パスがディレクトリーとして使用され、設定された UuidGenerator を使用してファイルが作成されます。分割条件が満たされるたびに、新しいファイルが作成されます。splitStrategy オプションは、次の構文の文字列として定義されます。value MESSAGES 新しいファイルが作成され、書き込まれたメッセージの数が value ミリ秒を超えると古いファイルが閉じられます IDLE 新しいファイルが作成され、最後の value ミリ秒内に書き込みが発生しなかった場合に古いファイルが閉じられます		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

135.2.3. KeyType と ValueType

- NULL キーまたは値が存在しないことを意味します
- バイトを書き込むための BYTE。Java Byte クラスは BYTE にマップされます。
- 一連のバイトを書き込むための BYTES。Java ByteBuffer クラスをマップします
- Java 整数を書き込むための INT
- Java float を書き込むための FLOAT
- Java long を書き込むための LONG
- Java double を書き込むための DOUBLE
- Java 文字列を書き込むための TEXT

BYTES は他のすべてでも使用されます。たとえば、Camel では、ファイルは InputStream として送信されます。この場合、シーケンスファイルまたはマップファイルにバイトシーケンスとして書き込まれます。

135.3. 分割ストラテジー

現在のバージョンの Hadoop では、ファイルを追加モードで開くことは信頼性が低いため無効になっています。そのため、現時点では、新しいファイルを作成することしかできません。Camel HDFS エンドポイントは、この問題を次の方法で解決しようとします。

- 分割ストラテジーオプションが定義されている場合、hdfs パスがディレクトリーとして使用され、設定された UuidGenerator を使用してファイルが作成されます。

- 分割条件が満たされるたびに、新しいファイルが作成されます。
  splitStrategy オプションは、次の構文の文字列として定義されます。
  splitStrategy=<ST>:< 値 >,<ST>:< 値 >,*

ここで <ST> は次のとおりです。

- BYTES 新しいファイルが作成され、書き込まれたバイト数が <value> を超えると古いファイルが閉じられます
- MESSAGES 新しいファイルが作成され、書き込まれたメッセージの数が <value> を超えると古いファイルが閉じられます
- IDLE 新しいファイルが作成され、最後の <value> ミリ秒内に書き込みが発生しなかった場合、古いファイルは閉じられます

注記

この戦略では現在、IDLE 値を設定するか、HdfsConstants.HDFS_CLOSE ヘッダーを false に設定して BYTES/MESSAGES 設定を使用する必要があることに注意してください。それ以外の場合、ファイルはメッセージごとに閉じられます。

以下に例を示します。

```
hdfs://localhost/tmp/simple-file?splitStrategy=IDLE:1000,BYTES:5
```

つまり、1 秒以上アイドル状態だった場合、または 5 バイト以上が書き込まれた場合に、新しいファイルが作成されます。したがって、**hadoop fs -ls /tmp/simple-file** を実行すると、複数のファイルが作成されていることがわかります。

135.4. メッセージヘッダー

このコンポーネントでは、次のヘッダーがサポートされています。

135.4.1. プロデューサーのみ

ヘッダー	説明
CamelFileName	Camel 2.13: 書き込むファイルの名前を指定します (エンドポイントパスに相対的)。名前は **String** または Expression オブジェクトにすることができます。分割ストラテジーを使用しない場合にのみ関連します。

135.5. ファイルストリームを閉じるための制御

Camel 2.10.4 以降で利用可能

分割ストラテジー **なしで**HDFS プロデューサーを使用する場合、ファイル出力ストリームはデフォルトで書き込み後に閉じられます。ただし、ストリームを開いたままにし、後でストリームを明示的に閉じるだけにしたい場合があります。そのために、ヘッダー **HdfsConstants.HDFS_CLOSE** (value = **"CamelHdfsClose"**) を使用してこれを制御できます。この値をブール値に設定すると、ストリームを閉じるかどうかを明示的に制御できます。

ストリームがいつ閉じられるかを制御できるさまざまなストラテジーがあるため、分割ストラテジーを使用する場合、これは当てはまらないことに注意してください。

135.6. このコンポーネントを OSGI で使用する

このコンポーネントは OSGi 環境で完全に機能しますが、ユーザーによるいくつかのアクションが必要です。リソースをロードするために、Hadoop はスレッドコンテキストクラスローダーを使用します。通常、スレッドコンテキストクラスローダーは、ルートを含むバンドルのバンドルクラスローダーになります。したがって、デフォルトの設定ファイルは、バンドルクラスローダーから見えるようにする必要があります。これに対処する一般的な方法は、バンドルルートに core-default.xml のコピーを保持することです。そのファイルは hadoop-common.jar にあります。

第136章 HDFS2 コンポーネント

Camel バージョン 2.14 以降で利用可能

hdfs2 コンポーネントを使用すると、Hadoop 2.x を使用して HDFS ファイルシステムとの間でメッセージを読み書きできます。HDFS は、Hadoop の中心にある分散ファイルシステムです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hdfs2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

136.1. URI 形式

```
hdfs2://hostname[:port][/path][?options]
```

次の形式でクエリーオプションを URI に追加できます: **?option=value&option=value&...**
パスは次のように処理されます。

1. コンシューマーとして、それがファイルの場合はファイルを読み取るだけで、それ以外の場合はディレクトリーを表す場合は、設定されたパターンを満たすパスの下にあるすべてのファイルをスキャンします。そのディレクトリーの下にあるすべてのファイルは、同じタイプである必要があります。

2. プロデューサーとして、少なくとも 1 つの分割ストラテジーが定義されている場合、パスはディレクトリーと見なされ、そのディレクトリーの下で、プロデューサーは設定された UuidGenerator を使用して名前が付けられた分割ごとに異なるファイルを作成します。

hdfs2 から通常モードで消費すると、ファイルはチャンクに分割され、チャンクごとにメッセージが生成されます。chunkSize オプションを使用して、チャンクのサイズを設定できます。hdfs から読み取り、file コンポーネントを使用して通常のファイルに書き込みたい場合は、fileMode=Append を使用して各チャンクを一緒に追加できます。

136.2. オプション

HDFS2 コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
jAASConfiguration (Common)	指定された設定を JAAS でのセキュリティーに使用するため。		設定
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HDFS2 エンドポイントは、URI 構文を使用して設定されます。

```
hdfs2:hostName:port/path
```

パスおよびクエリーパラメーターを使用します。

136.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
hostname	使用するために **必要な** HDFS ホスト		String
port	使用する HDFS ポート	8020	int
path	**必須** 使用するディレクトリーパス		String

136.2.2. クエリーパラメーター (38 パラメーター)

名前	説明	デフォルト	タイプ
connectOnStartup (Common)	プロデューサー/コンシューマーの起動時に HDFS ファイルシステムに接続するかどうか。false の場合、接続はオンデマンドで作成されます。HDFS は 45 x 20 秒の再配信をハードコーディングしているため、接続を確立するのに最大 15 分かかる場合があることに注意してください。このオプションを false に設定すると、アプリケーションが起動し、最大 15 分間ブロックされなくなります。	true	boolean
fileSystemType (Common)	HDFS を使用せず、代わりにローカルの java.io.File を使用するには、LOCAL に設定します。	HDFS	HdfsFileSystemType
fileType (Common)	使用するファイルの種類。詳細については、さまざまなファイルタイプに関する Hadoop HDFS のドキュメントを参照してください。	NORMAL_FILE	HdfsFileType
keyType (Common)	シーケンスまたはマップファイルの場合のキーのタイプ。	NULL	WritableType
owner (Common)	コンシューマーがファイルを取得するには、ファイルの所有者がこの所有者と一致する必要があります。それ以外の場合、ファイルはスキップされます。		String
valueType (Common)	シーケンスまたはマップファイルの場合のキーのタイプ	BYTES	WritableType

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pattern (consumer)	ディレクトリーのスキャンに使用されるパターン	*	String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
append (producer)	既存のファイルに追加します。すべての HDFS ファイルシステムが追加オプションをサポートしているわけではないことに注意してください。	false	boolean
overwrite (producer)	同名の既存ファイルを上書きするかどうか	true	boolean
blockSize (advanced)	HDFS ブロックのサイズ	67108864	long
bufferSize (advanced)	HDFS が使用するバッファーサイズ	4096	int

名前	説明	デフォルト	タイプ
checkIdleInterval (advanced)	アイドルチェッカーバックグラウンドタスクを実行する頻度 (ミリ秒単位)。このオプションは、スプリッタ戦略が IDLE の場合にのみ使用されます。	500	int
chunkSize (advanced)	通常のファイルを読み取る場合、これはチャンクに分割され、チャンクごとにメッセージが生成されます。	4096	int
compressionCodec (advanced)	使用する圧縮コーデック	DEFAULT	HdfsCompressionCodec
compressionType (advanced)	使用する圧縮タイプ (デフォルトでは使用されていません)	NONE	CompressionType
openedSuffix (advanced)	ファイルが読み取り/書き込み用に開かれると、ファイルはこの接尾辞で名前が変更され、書き込みフェーズ中に読み取られないようにします。	opened	String
readSuffix (advanced)	ファイルが読み取られると、この接尾辞を使用して名前が変更され、再度読み取られることがなくなります。	read	String
replication (advanced)	HDFS レプリケーション係数	3	short
splitStrategy (advanced)	現在のバージョンの Hadoop では、ファイルを追加モードで開くことは信頼性が低いため無効になっています。そのため、現時点では、新しいファイルを作成することしかできません。Camel HDFS エンドポイントは、この問題を次のように解決しようとします。分割戦略オプションが定義されている場合、hdfs パスがディレクトリーとして使用され、設定された UuidGenerator を使用してファイルが作成されます。分割条件が満たされるたびに、新しいファイルが作成されます。splitStrategy オプションは、次の構文の文字列として定義されます。value MESSAGES 新しいファイルが作成され、書き込まれたメッセージの数が value ミリ秒を超えると古いファイルが閉じられます IDLE 新しいファイルが作成され、最後の value ミリ秒内に書き込みが発生しなかった場合に古いファイルが閉じられます		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean

名前	説明	デフォルト	タイプ
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

136.2.3. KeyType と ValueType

- NULL キーまたは値が存在しないことを意味します
- バイトを書き込むための BYTE。Java Byte クラスは BYTE にマップされます。
- 一連のバイトを書き込むための BYTES。Java ByteBuffer クラスをマップします
- Java 整数を書き込むための INT
- Java float を書き込むための FLOAT
- Java long を書き込むための LONG
- Java double を書き込むための DOUBLE
- Java 文字列を書き込むための TEXT

BYTES は他のすべてでも使用されます。たとえば、Camel では、ファイルは InputStream として送信されます。この場合、シーケンスファイルまたはマップファイルにバイトシーケンスとして書き込まれます。

136.3. 分割ストラテジー

現在のバージョンの Hadoop では、ファイルを追加モードで開くことは信頼性が低いため無効になっています。そのため、現時点では、新しいファイルを作成することしかできません。Camel HDFS エンドポイントは、この問題を次の方法で解決しようとします。

- 分割ストラテジーオプションが定義されている場合、hdfs パスがディレクトリーとして使用され、設定された UuidGenerator を使用してファイルが作成されます。
- 分割条件が満たされるたびに、新しいファイルが作成されます。
  splitStrategy オプションは、次の構文の文字列として定義されます: splitStrategy=<ST>:<value>, <ST>:<value>,*

ここで <ST> は次のとおりです。

- BYTES 新しいファイルが作成され、書き込まれたバイト数が <value> を超えると古いファイルが閉じられます
- MESSAGES 新しいファイルが作成され、書き込まれたメッセージの数が <value> を超えると古いファイルが閉じられます

- IDLE 新しいファイルが作成され、最後の <value> ミリ秒内に書き込みが発生しなかった場合、古いファイルは閉じられます

この戦略では現在、IDLE 値を設定するか、HdfsConstants.HDFS_CLOSE ヘッダーを false に設定して BYTES/MESSAGES 設定を使用する必要があることに注意してください。それ以外の場合、ファイルはメッセージごとに閉じられます。

以下に例を示します。

```
hdfs2://localhost/tmp/simple-file?splitStrategy=IDLE:1000,BYTES:5
```

つまり、1秒以上アイドル状態だった場合、または 5 バイト以上が書き込まれた場合に、新しいファイルが作成されます。したがって、**hadoop fs -ls/tmp/simple-file** を実行すると、複数のファイルが作成されていることがわかります。

136.4. メッセージヘッダー

このコンポーネントでは、次のヘッダーがサポートされています。

136.4.1. プロデューサーのみ

ヘッダー	説明
CamelFileName	Camel 2.13: 書き込むファイルの名前を指定します (エンドポイントパスに相対的)。名前は **String** または Expression オブジェクトにすることができます。分割ストラテジーを使用しない場合にのみ関連します。

136.5. ファイルストリームを閉じるための制御

分割ストラテジー **なしで** HDFS2 プロデューサーを使用する場合、ファイル出力ストリームはデフォルトで書き込み後に閉じられます。ただし、ストリームを開いたままにし、後でストリームを明示的に閉じるだけにしたい場合があります。そのために、ヘッダー **HdfsConstants.HDFS_CLOSE** (value = **"CamelHdfsClose"**) を使用してこれを制御できます。この値をブール値に設定すると、ストリームを閉じるかどうかを明示的に制御できます。

ストリームがいつ閉じられるかを制御できるさまざまなストラテジーがあるため、分割ストラテジーを使用する場合、これは当てはまらないことに注意してください。

136.6. このコンポーネントを OSGI で使用する

このコンポーネントを OSGi 環境で実行する場合、Hadoop 2.x がさまざまな **org.apache.hadoop.fs.FileSystem** 実装を検出するために使用するメカニズムに関連するいくつかの特異な点があります。Hadoop 2.x は、利用可能なファイルシステムのタイプと実装を定義する **/META-INF/services/org.apache.hadoop.fs.FileSystem** ファイルを探す **java.util.ServiceLoader** を使用します。OSGi 内で実行している場合、これらのリソースは使用できません。

camel-hdfs コンポーネントと同様に、デフォルトの設定ファイルはバンドルクラスローダーから見える必要があります。これに対処する一般的な方法は、**core-default.xml** (および **hdfs-default.xml** など) のコピーをバンドルルートに保持することです。

136.6.1. 手動で定義されたルートでこのコンポーネントを使用する

以下の 2 つのオプションがあります。

1. **/META-INF/services/org.apache.hadoop.fs.FileSystem** リソースを、ルートを定義するバンドルとともにパッケージ化します。このリソースには、必要な Hadoop 2.x ファイルシステムの実装がすべてリストされている必要があります。

2. **org.apache.hadoop.fs.FileSystem** クラス内に内部の静的キャッシュを設定するボイラープレート初期化コードを提供します。

```
org.apache.hadoop.conf.Configuration conf = new org.apache.hadoop.conf.Configuration();
conf.setClass("fs.file.impl", org.apache.hadoop.fs.LocalFileSystem.class, FileSystem.class);
conf.setClass("fs.hdfs.impl", org.apache.hadoop.hdfs.DistributedFileSystem.class, FileSystem.class);
...
FileSystem.get("file:///", conf);
FileSystem.get("hdfs://localhost:9000/", conf);
...
```

136.6.2. ブループリントコンテナーでこのコンポーネントを使用する

2 つのオプション:

1. **/META-INF/services/org.apache.hadoop.fs.FileSystem** リソースをブループリント定義を含むバンドルでパッケージ化します。

2. 以下をブループリント定義ファイルに追加します。

```
<bean id="hdfsOsgiHelper" class="org.apache.camel.component.hdfs2.HdfsOsgiHelper">
  <argument>
    <map>
      <entry key="file:///" value="org.apache.hadoop.fs.LocalFileSystem"  />
      <entry key="hdfs://localhost:9000/" value="org.apache.hadoop.hdfs.DistributedFileSystem" />
      ...
    </map>
  </argument>
</bean>

<bean id="hdfs2" class="org.apache.camel.component.hdfs2.HdfsComponent" depends-
on="hdfsOsgiHelper" />
```

このようにして、Hadoop 2.x は URI スキームをファイルシステム実装に正しくマッピングします。

第137章 HEADERSMAP

Camel 2.20 以降で利用可能

camel-headersmap は、大文字と小文字を区別しないマップのより高速な実装であり、Camel が実行時にプラグインして使用することで、Camel Message ヘッダーのパフォーマンスをわずかに高速化できます。

137.1. クラスパスからの自動検出

この実装を使用するには、**camel-headersmap** 依存関係をクラスパスに追加するだけです。Camel は起動時にこれを自動検出し、次のようにログに記録します。

```
Detected and using custom HeadersMapFactory:
org.apache.camel.component.headersmap.FastHeadersMapFactory@71e9ebae
```

spring-boot には、使用する必要がある **camel-headersmap-starter** 依存関係があります。

137.2. 手動有効化

OSGi を使用するか、実装がクラスパスに追加されていない場合は、次のように明示的に有効にする必要があります。

```
CamelContext camel = ...

camel.setHeadersMapFactory(new FastHeadersMapFactory());
```

または、XML DSL (Spring または Blueprint XML ファイル) では、ファクトリーを **<bean>** として宣言できます。

```
<bean id="fastMapFactory"
class="org.apache.camel.component.headersmap.FastHeadersMapFactory"/>
```

次に、Camel は Bean を検出し、ログに記録されたファクトリーを使用する必要があります。

第138章 HESSIAN DATAFORMAT (非推奨)

Camel バージョン 2.17 以降で利用可能

Hessian は、Caucho の Hessian 形式を使用してメッセージをマーシャリングおよびアンマーシャリングするためのデータ形式です。

Maven の Hessian Data Format を使用する場合は、次の依存関係を **pom.xml** に追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hessian</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

138.1. オプション

Hessian データ形式は、以下に示す 4 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
whitelistEnabled	**true**	**Boolean**	ホワイトリスト機能を有効にするかどうかを定義します
allowedUnmarshallObjects		**String**	非整列化を許可するオブジェクトを定義します
deniedUnmarshallObjects		**String**	拒否されたオブジェクトをアンマーシャリングするように定義する
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

138.2. JAVA DSL での HESSIAN データ形式の使用

```
from("direct:in")
    .marshal().hessian();
```

138.3. SPRING DSL での HESSIAN データ形式の使用

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:in"/>
```

```
        <marshal ref="hessian"/>
    </route>
</camelContext>
```

第139章 HIPCHAT コンポーネント

Camel バージョン 2.15 以降で利用可能

Hipchat コンポーネントは、Hipchat サービスとの間のメッセージの生成と消費をサポートします。

前提条件

有効な Hipchat ユーザーアカウントを持ち、メッセージの生成/消費に使用できる 個人用アクセストークン を取得する必要があります。

139.1. URI 形式

```
hipchat://[host][:port]?options
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

139.2. URI オプション

Hipchat コンポーネントにはオプションがありません。

Hipchat エンドポイントは、URI 構文を使用して設定されます。

```
hipchat:protocol:host:port
```

パスおよびクエリーパラメーターを使用します。

139.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
protocol	**必須** Hipchat サーバーのプロトコル (http など)。		String
host	**必須** api.hipchat.com などの hipchat サーバーのホスト		String
port	hipchat サーバーのポート。デフォルトは 80 です。	80	Integer

139.2.2. クエリーパラメーター(22 個のパラメーター):

名前	説明	デフォルト	タイプ
authToken (Common)	OAuth 2 認証トークン		String

名前	説明	デフォルト	タイプ
consumeUsers (Common)	hiptchat サーバーからのメッセージを消費するときのユーザー名。複数のユーザー名はコンマで区切ることができます。		String
httpClient (Common)	API HTTP 要求中に使用されるレジストリーからの CloseableHttpClient 参照。	HttpClient ライブラリーの CloseableHttpClient デフォルト	CloseableHttpClient
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean

名前	説明	デフォルト	タイプ
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

139.3. スケジュールされたポーリングコンシューマー

このコンポーネントは、ScheduledPollConsumer を実装します。指定された consumeUsers からの最後のメッセージのみが取得され、エクスチェンジボディーとして送信されます。次のポーリングで新しいメッセージがないときに同じメッセージを再度取得したくない場合は、以下に示すようにべき等コンシューマーを追加できます。ScheduledPollConsumer のすべてのオプションを使用して、コンシューマーをより詳細に制御することもできます。

```
@Override
public void configure() throws Exception {
 String hipchatEndpointUri = "hipchat://?authToken=XXXX&consumeUsers=@Joe,@John";
 from(hipchatEndpointUri)
  .idempotentConsumer(
    simple("${in.header.HipchatMessageDate} ${in.header.HipchatFromUser}"),
    MemoryIdempotentRepository.memoryIdempotentRepository(200)
  )
  .to("mock:result");
}
```

139.3.1. Hipchat コンシューマーによって設定されたメッセージヘッダー

ヘッダー	定数	タイプ	説明
HipchatFromUser	HipchatConstants.FROM_USER	**String**	本文には、このユーザーから authToken の所有者に送信されたメッセージが含まれています
HipchatMessageDate	HipchatConstants.MESSAGE_DATE	**String**	メッセージが送信された日付。形式は、Hipchat の レスポンス にある ISO-8601 です。

139.4. HIPCHAT プロデューサー

プロデューサーは、ルームとユーザーの両方に同時にメッセージを送信できます。エクスチェンジのボディーはメッセージとして送信されます。使用例を以下に示します。適切なヘッダーを設定する必要があります。

```
@Override
 public void configure() throws Exception {
  String hipchatEndpointUri = "hipchat://?authToken=XXXX";
  from("direct:start")
   .to(hipchatEndpointUri)
   .to("mock:result");
  }
```

139.4.1. Hipchat プロデューサーによって評価されるメッセージヘッダー

ヘッダー	定数	タイプ	説明
HipchatToUser	HipchatConstants.TO_USER	**String**	メッセージの送信先の Hipchat ユーザー。
HipchatToRoom	HipchatConstants.TO_ROOM	**String**	メッセージを送信する必要がある Hipchat ルーム。
HipchatMessageFormat	HipchatConstants.MESSAGE_FORMAT	String	有効な形式は text または html です。**Default: 'text'**
HipchatMessageBackgroundColor	HipchatConstants.MESSAGE_BACKGROUND_COLOR	**String**	有効な色の値は、'yellow'、'green'、'red'、'purple'、'gray'、'random' です。**デフォルト: 'yellow' (ルームのみ)**
HipchatTriggerNotification	HipchatConstants.TRIGGER_NOTIFY	**String**	有効な値は true または false です。このメッセージがユーザー通知をトリガーするかどうか (タブの色の変更、サウンドの再生、携帯電話への通知など)。**デフォルト: 'false' (ルームのみ)**

139.4.2. Hipchat プロデューサーによって設定されたメッセージヘッダー

ヘッダー	定数	タイプ	説明
HipchatToUserResponseStatus	HipchatConstants.TO_USER_RESPONSE_STATUS	Status Line メッセージがユーザーに送信されたときに受信した API 応答のステータス。	HipchatFromUserResponseStatus

139.4.3. HTTP クライアントの設定

HipChat コンポーネントでは、独自の **HttpClient** 設定が可能です。これは、レジストリー (Spring Context など) で **CloseableHttpClient** の参照を定義し、エンドポイントの定義中にパラメーターを設定することで実行できます (例: **hipchat:http://api.hipchat.com?httpClient=#myHttpClient**)。

```
CloseableHttpClient httpclient = HttpClients.custom()
    .setConnectionManager(connManager)
    .setDefaultCookieStore(cookieStore)
    .setDefaultCredentialsProvider(credentialsProvider)
    .setProxy(new HttpHost("myproxy", 8080))
    .setDefaultRequestConfig(defaultRequestConfig)
    .build();
```

HTTP クライアント設定の詳細については、公式ドキュメント を確認してください。

139.4.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-hipchat</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は、Camel の実際のバージョン (2.15.0 以降) に置き換える必要があります。

第140章 HL7 DATAFORMAT

Camel バージョン 2.0 以降で利用可能

HL7 コンポーネントは、HAPI ライブラリー を使用して HL7 MLLP プロトコルおよび HL7 v2 メッセージ を操作するために使用されます。

このコンポーネントは以下をサポートします。

- Mina 用 HL7 MLLP コーデック
- **Camel 2.15** 以降の Netty4 用の HL7 MLLP コーデック
- HAPI および文字列との間の型コンバーター
- HAPI ライブラリーを使用した HL7 DataFormat
- camel-mina2 コンポーネントとうまく統合されているため、さらに使いやすくなっています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hl7</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

140.1. HL7 MLLP プロトコル

HL7 は、多くの場合、テキストベースの TCP ソケットベースのプロトコルである HL7 MLLP プロトコルと共に使用されます。このコンポーネントには、MLLP プロトコルに準拠する Mina および Netty4 コーデックが付属しているため、TCP トランスポート層を介して HL7 要求を受け入れる HL7 リスナーを簡単に公開できます。HL7 リスナーサービスを公開するには、camel-mina2 または camel-netty4 コンポーネントを **HL7MLLPCodec** (mina2) または **HL7MLLPNettyDecoder**/**HL7MLLPNettyEncoder** (Netty4) と共に使用します。

HL7 MLLP コーデックは、次のように設定できます。

名前	デフォルト値	説明
startByte	**0x0b**	HL7 ペイロードにまたがる開始バイト。
endByte1	**0x1c**	HL7 ペイロードにまたがる最初のエンドバイト。
endByte2	**0x0d**	HL7 ペイロードにまたがる 2 番目のエンドバイト。

名前	デフォルト値	説明
charset	JVM デフォルト	コーデックに使用するエンコーディング (文字セット名)。指定しない場合、Camel は JVM のデフォルトの文字セット を使用します。
produceString	**true**	**(Camel 2.14.1 以降)** true の場合、コーデックは定義された文字セットを使用して文字列を作成します。false の場合、コーデックはプレーンバイト配列をルートに送信し、HL7 データ形式で HL7 メッセージコンテンツからの実際の文字セットを判別できるようにします。
convertLFtoCR	**false**	HL7 が **\r** をセグメント終端記号として規定しているため、**\n** を **\r** (**0x0d**、10 進数の 13) に変換します。HAPI ライブラリーでは **\r** を使用する必要があります。

140.1.1. Mina を使用した HL7 リスナーの公開

Spring XML ファイルでは、ポート **8888** で TCP を使用して HL7 リクエストをリッスンするように mina2 エンドポイントを設定します。

```
<endpoint id="hl7MinaListener" uri="mina2:tcp://localhost:8888?sync=true&amp;codec=#hl7codec"/>
```

sync=true は、このリスナーが同期的であるため、HL7 応答を呼び出し元に返すことを示します。HL7 コーデックは、**codec=#hl7codec** で設定されます。**hl7codec** は単なる Spring Bean ID であるため、**mygreatcodecforhl7** などの名前を指定できることに注意してください。コーデックは、Spring XML ファイルにも設定されています。

```
<bean id="hl7codec" class="org.apache.camel.component.hl7.HL7MLLPCodec">
    <property name="charset" value="iso-8859-1"/>
</bean>
```

次の Java DSL の例が示すように、エンドポイント **hl7MinaLlistener** をルートでコンシューマーとして使用できます。

```
from("hl7MinaListener")
  .bean("patientLookupService");
```

これは、HL7 をリッスンし、**patientLookupService** という名前のサービスにルーティングする非常に単純なルートです。これは Spring Bean ID でもあり、Spring XML で次のように設定されます。

```
<bean id="patientLookupService"
class="com.mycompany.healthcare.service.PatientLookupService"/>
```

次に示すように、Camel に依存しない POJO クラスでビジネスロジックを実装できます。

```
import ca.uhn.hl7v2.HL7Exception;
import ca.uhn.hl7v2.model.Message;
import ca.uhn.hl7v2.model.v24.segment.QRD;

public class PatientLookupService {
```

```
public Message lookupPatient(Message input) throws HL7Exception {
    QRD qrd = (QRD)input.get("QRD");
    String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();

    // find patient data based on the patient id and create a HL7 model object with the response
    Message response = ... create and set response data
    return response
}
```

140.1.2. Netty を使用した HL7 リスナーの公開 (Camel 2.15 以降で使用可能)

Spring XML ファイルでは、ポート **8888** で TCP を使用して HL7 リクエストをリッスンするように netty4 エンドポイントを設定します。

```
<endpoint id="hl7NettyListener" uri="netty4:tcp://localhost:8888?
sync=true&amp;encoder=#hl7encoder&amp;decoder=#hl7decoder"/>
```

sync=true は、このリスナーが同期的であるため、HL7 応答を呼び出し元に返すことを示します。HL7 コーデックは、**encoder=#hl7encoder*and*decoder=#hl7decoder** でセットアップされます。**hl7encoder** と **hl7decoder** は単なる Bean ID であるため、異なる名前が付けられる可能性があることに注意してください。Bean は Spring XML ファイルで設定できます。

```
<bean id="hl7decoder" class="org.apache.camel.component.hl7.HL7MLLPNettyDecoderFactory"/>
<bean id="hl7encoder" class="org.apache.camel.component.hl7.HL7MLLPNettyEncoderFactory"/>
```

次の Java DSL の例が示すように、エンドポイント **hl7NettyListener** をコンシューマーとしてルートで使用できます。

```
from("hl7NettyListener")
  .bean("patientLookupService");
```

140.2. JAVA.LANG.STRING または BYTE を使用する HL7 モデル

HL7 MLLP コーデックは、プレーンな String をデータ形式として使用します。Camel は Type Converter を使用して文字列を HAPI HL7 モデルオブジェクトに変換しますが、たとえば、データを自分で解析する場合など、必要に応じてプレーンな String オブジェクトを使用することもできます。

Camel 2.14.1 の時点では、**ProduceString** プロパティーを false に設定することで、Mina コーデックと Netty コーデックの両方でプレーンな **byte[]** をデータ形式として使用できるようにします。Type Converter は、**byte[]** を HAPI HL7 モデルオブジェクトとの間で変換することもできます。

140.3. HAPI を使用した HL7V2 モデル

HL7v2 モデルは、HAPI ライブラリーの Java オブジェクトを使用します。このライブラリーを使用すると、HL7v2 で主に使用される EDI 形式 (ER7) からエンコードおよびデコードできます。

以下のサンプルは、患者 ID が **0101701234** の患者を検索するリクエストです。

```
MSH|^~\\&|MYSENDER|MYRECEIVER|MYAPPLICATION||200612211200||QRY^A19|1234|P|2.4
QRD|200612211200|R|I|GetPatient|||1^RD|0101701234|DEM||
```

HL7 モデルを使用すると、**ca.uhn.hl7v2.model.Message** オブジェクトを操作して、患者 ID などを取得できます。

```
Message msg = exchange.getIn().getBody(Message.class);
QRD qrd = (QRD)msg.get("QRD");
String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();  // 0101701234
```

byte、**String**、またはその他の単純なオブジェクト形式を操作する必要がないため、これは HL7 リスナーと組み合わせると強力です。HAPI HL7v2 モデルオブジェクトをそのまま使用できます。メッセージのタイプが事前にわかっている場合は、型の安全性をさらに確保できます。

```
QRY_A19 msg = exchange.getIn().getBody(QRY_A19.class);
String patientId = msg.getQRD().getWhoSubjectFilter(0).getIDNumber().getValue();
```

140.4. HL7 DATAFORMAT

HL7 コンポーネントには、HL7 モデルオブジェクトのマーシャリングまたはアンマーシャリングに使用できる HL7 データ形式が付属しています。

HL7 データ形式は、以下にリストされている 2 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
validate	**true**	**Boolean**	HL7 メッセージを検証するかどうかデフォルトでは true です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

- **marshal** = メッセージからバイトストリームへ (HL7 MLLP コーデックを使用して応答するときに使用できます)
- **unmarshal** = バイトストリームからメッセージへ (HL7 MLLP からストリーミングデータを受信するときに使用できます)

データ形式を使用するには、単にインスタンスをインスタンス化し、ルートビルダーで整列化または非整列化操作を呼び出します。

```
DataFormat hl7 = new HL7DataFormat();

from("direct:hl7in")
  .marshal(hl7)
  .to("jms:queue:hl7out");
```

上記のサンプルでは、HL7 が HAPI メッセージオブジェクトからバイトストリームにマーシャリングされ、JMS キューに配置されます。
次の例は逆です。

```
DataFormat hl7 = new HL7DataFormat();

from("jms:queue:hl7out")
```

```
    .unmarshal(hl7)
    .to("patientLookupService");
```

ここでは、患者の検索サービスに渡される HAPI メッセージオブジェクトにバイトストリームを非整列化します。

140.4.1. シリアライズ可能なメッセージ

HAPI 2.0 (**Camel 2.11** で使用) の時点で、HL7v2 モデルクラスは完全にシリアライズ可能です。したがって、HL7v2 メッセージを直接 JMS キューに入れ (つまり、**marshal()** を呼び出さずに)、キューから直接 (つまり、**unmarshal()** を呼び出さずに) 再度読み取ることができます。

140.4.2. セグメント区切り

Camel 2.11 の時点で、**unmarshal** は **\n** を **\r** に変換することによってセグメントセパレーターを自動的に修正しなくなりました。
この変換が必要な場合には、**org.apache.camel.component.hl7.HL7#convertLFToCR** は、この目的向けに便利な **Expression** を提供します。

140.4.3. Charset

Camel 2.14.1 の時点で、**marshal and unmarshal** の両方が、フィールド **MSH-18** で提供される文字セットを評価します。このフィールドが空の場合には、デフォルトで、対応する Camel 文字セットプロパティー/ヘッダーに含まれる文字セットが想定されます。**HL7DataFormat** クラスから継承するときに、**guessCharsetName** メソッドをオーバーライドして、このデフォルトの動作を変更することもできます。

Camel には、一般的に使用される既知のデータ形式の簡略構文があります。**HL7DataFormat** オブジェクトのインスタンスを作成する必要はありません。

```
from("direct:hl7in")
  .marshal().hl7()
  .to("jms:queue:hl7out");

from("jms:queue:hl7out")
  .unmarshal().hl7()
  .to("patientLookupService");
```

140.5. メッセージヘッダー

非整列化操作は、MSH セグメントからこれらのフィールドを Camel メッセージのヘッダーとして追加します。

キー	MSH フィールド	例
CamelHL7SendingApplication	**MSH-3**	**MYSERVER**
CamelHL7SendingFacility	**MSH-4**	**MYSERVERAPP**
CamelHL7ReceivingApplication	**MSH-5**	**MYCLIENT**
CamelHL7ReceivingFacility	**MSH-6**	**MYCLIENTAPP**
CamelHL7Timestamp	**MSH-7**	**20071231235900**
CamelHL7Security	**MSH-8**	**null**
CamelHL7MessageType	**MSH-9-1**	**ADT**
CamelHL7TriggerEvent	**MSH-9-2**	**A01**

キー	MSH フィールド	例
CamelHL7MessageControl	**MSH-10**	**1234**
CamelHL7ProcessingId	**MSH-11**	**P**
CamelHL7VersionId	**MSH-12**	**2.4**
`CamelHL7Context	``	**(Camel 2.14)** には、メッセージの解析に使用された HapiContext が含まれています
CamelHL7Charset	**MSH-18**	(Camel 2.14.1) **UNICODE UTF-8**

CamelHL7Context 以外のヘッダーはすべて String 型です。ヘッダー値が欠落している場合、その値は **null** です。

140.6. オプション

HL7 データ形式は、次のオプションをサポートしています。

オプション	デフォルト	説明
validate	true	HAPI パーサーがデフォルトの検証規則を使用してメッセージを検証する必要があるかどうか。**パーサー** または **hapiContext** オプションを使用し、目的の HAPI **ValidationContext** で初期化することをお勧めします
parser	**ca.uhn.hl7v2.parser.GenericParser**	使用するカスタムパーサー。タイプ **ca.uhn.hl7v2.parser.Parser** である必要があります。**GenericParser** では、XML エンコードされた HL7v2 メッセージも解析できることに注意してください。

オプション	デフォルト	説明
hapiContext	**ca.uhn.hl7v2.DefaultHapiContext**	Camel 2.14: カスタムパーサー、カスタム ValidationContext などを定義できるカスタム HAPI コンテキスト。これにより、HL7 の解析およびレンダリングプロセスを完全に制御できます。

140.7. 依存関係

Camel ルートで HL7 を使用するには、このデータ形式を実装する上記の **camel-hl7** に依存関係を追加する必要があります。

HAPI ライブラリーは、基本ライブラリー といくつかの構造ライブラリー (HL7v2 メッセージバージョンごとに 1 つ) に分割されています。

- v2.1 構造ライブラリー
- v2.2 構造ライブラリー
- v2.3 構造ライブラリー
- v2.3.1 構造ライブラリー
- v2.4 構造ライブラリー
- v2.5 構造ライブラリー
- v2.5.1 構造ライブラリー
- v2.6 構造ライブラリー

デフォルトでは、**camel-hl7** は HAPI 基本ライブラリー のみを参照します。アプリケーションは、構造ライブラリー自体を追加します。たとえば、アプリケーションが HL7v2 メッセージバージョン 2.4 および 2.5 で動作する場合には、次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v24</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v25</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
```

あるいは、基本ライブラリー、すべての構造ライブラリー、および必要な依存関係 (バンドルクラスパス上) を含む OSGi バンドルを 中央の Maven リポジトリー からダウンロードできます。

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-osgi-base</artifactId>
  <version>2.2</version>
</dependency>
```

140.8. TERSER 言語

HAPI は、一般的に使用される簡潔なロケーション指定構文を使用してフィールドへのアクセスを提供する Terser クラスを提供します。Terser 言語では、この構文を使用してメッセージから値を抽出し、フィルタリングやコンテンツベースのルーティングなどの式や述語として使用できます。

例:

```
import static org.apache.camel.component.hl7.HL7.terser;

  // extract patient ID from field QRD-8 in the QRY_A19 message above and put into message header
  from("direct:test1")
    .setHeader("PATIENT_ID",terser("QRD-8(0)-1"))
    .to("mock:test1");

  // continue processing if extracted field equals a message header
  from("direct:test2")
    .filter(terser("QRD-8(0)-1").isEqualTo(header("PATIENT_ID"))
    .to("mock:test2");
```

140.9. HL7 検証述語

多くの場合、最初に HL7v2 メッセージを解析し、別の手順で HAPI ValidationContext に対して検証することをお勧めします。

例:

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import ca.uhn.hl7v2.validation.impl.DefaultValidation;

  // Use standard or define your own validation rules
  ValidationContext defaultContext = new DefaultValidation();

  // Throws PredicateValidationException if message does not validate
  from("direct:test1")
    .validate(messageConformsTo(defaultContext))
    .to("mock:test1");
```

140.10. HAPICONTEXT を使用した HL7 検証述語 (CAMEL 2.14)

HAPI コンテキストは常に ValidationContext (または ValidationRuleBuilder) で設定されるため、検証ルールに間接的にアクセスできます。さらに、HL7DataFormat をアンマーシャリングすると、設定された HAPI コンテキストが **CamelHL7Context** ヘッダーに転送され、このコンテキストの検証ルールを

簡単に再利用できます。

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import static org.apache.camel.component.hl7.HL7.messageConforms

  HapiContext hapiContext = new DefaultHapiContext();
  hapiContext.getParserConfiguration().setValidating(false); // don't validate during parsing

  // customize HapiContext some more ... e.g. enforce that PID-8 in ADT_A01 messages of version
2.4 is not empty
  ValidationRuleBuilder builder = new ValidationRuleBuilder() {
     @Override
     protected void configure() {
        forVersion(Version.V24)
            .message("ADT", "A01")
            .terser("PID-8", not(empty()));
        }
     };
  hapiContext.setValidationRuleBuilder(builder);

  HL7DataFormat hl7 = new HL7DataFormat();
  hl7.setHapiContext(hapiContext);

  from("direct:test1")
     .unmarshal(hl7)              // uses the GenericParser returned from the HapiContext
     .validate(messageConforms())   // uses the validation rules returned from the HapiContext
                                    // equivalent with .validate(messageConformsTo(hapiContext))
     // route continues from here
```

140.11. HL7 確認式

HL7v2 処理の一般的なタスクは、受信した HL7v2 メッセージへのレスポンスとして、たとえば検証結果に基づいて確認メッセージを生成することです。**ack** 式を使用すると、これを非常にエレガントに実現できます。

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import static org.apache.camel.component.hl7.HL7.ack;
import ca.uhn.hl7v2.validation.impl.DefaultValidation;

  // Use standard or define your own validation rules
   ValidationContext defaultContext = new DefaultValidation();

   from("direct:test1")
     .onException(Exception.class)
        .handled(true)
        .transform(ack()) // auto-generates negative ack because of exception in Exchange
        .end()
     .validate(messageConformsTo(defaultContext))
     // do something meaningful here

     // acknowledgement
     .transform(ack())
```

140.12. その他のサンプル

次の例では、プレーンな **String** HL7 リクエストが、レスポンスを返す HL7 リスナーに送信されます。

次のサンプルでは、HL7 リスナーからの HL7 リクエストがビジネスロジックにルーティングされます。ビジネスロジックは、レジストリーに **hl7service** として登録されたプレーンな POJO として実装されます。

次に、**RouteBuilder** を使用した Camel ルートは次のようになります。

HL7 DataFormat を使用すると、Camel メッセージヘッダーに MSH セグメントのフィールドが入力されることに注意してください。ヘッダーは、上記の例に示すように、フィルタリングやコンテンツベースのルーティングに特に役立ちます。

第141章 HTTP コンポーネント (非推奨)

Camel バージョン 1.0 以降で利用可能

http: コンポーネントは、外部 HTTP リソースを消費するための HTTP ベースのエンドポイントを提供します (HTTP を使用して外部サーバーを呼び出すクライアントとして)。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

141.1. URI 形式

```
http:hostname[:port][/resourceUri][?param1=value1][&param2=value2]
```

デフォルトでは、HTTP にはポート 80、HTTPS には 443 を使用します。

camel-http vs camel-jetty

HTTP コンポーネントによって生成されたエンドポイントに対してのみ生成できます。したがって、camel ルートへの入力として使用しないでください。HTTP サーバーを介して camel ルートへの入力として HTTP エンドポイントをバインド/公開するには、Jetty コンポーネント または Servlet コンポーネント を使用できます。

141.2. 例

POST を使用して本文で URL を呼び出し、応答を out メッセージとして返します。body が null の場合、GET を使用して URL を呼び出し、レスポンスを out メッセージとして返します。

Java DSL

Spring DSL

```
from("direct:start")
  .to("http://myhost/mypath");
```

```
<from uri="direct:start"/>
<to uri="http://oldhost"/>
```

ヘッダーを追加することで、HTTP エンドポイント URI をオーバーライドできます。Camel は http://newhost を呼び出します。これは、REST URL などで非常に便利です。

Java DSL

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, simple("http://myserver/orders/${header.orderId}"))
  .to("http://dummyhost");
```

URI パラメーターは、エンドポイント URI に直接設定するか、ヘッダーとして設定できます

Java DSL

```
from("direct:start")
  .to("http://oldhost?order=123&detail=short");
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http://oldhost");
```

HTTP リクエストメソッドを POST に設定します

Java DSL

Spring DSL

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD, constant("POST"))
  .to("http://www.google.com");
```

```
<from uri="direct:start"/>
<setHeader headerName="CamelHttpMethod">
  <constant>POST</constant>
</setHeader>
<to uri="http://www.google.com"/>
<to uri="mock:results"/>
```

141.3. HTTP オプション

HTTP コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
httpClientConfigurer (advanced)	カスタム HttpClientConfigurer を使用して、使用される HttpClient の設定を実行します。		HttpClientConfigurer
httpConnectionManager (advanced)	カスタム HttpConnectionManager を使用して接続を管理する場合		HttpConnectionManager
httpBinding (producer)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
httpConfiguration (producer)	共有 HttpConfiguration を基本設定として使用するには、以下を行います。		HttpConfiguration

名前	説明	デフォルト	タイプ
allowJavaSerialized Object (producer)	リクエストが context-type=application/x-java-serialized-object を使用するときに Java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HTTP エンドポイントは、URI 構文を使用して設定されます。

```
http:httpUri
```

パスおよびクエリーパラメーターを使用します。

141.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpUri	**必須** 呼び出す HTTP エンドポイントの URL。		URI

141.3.2. クエリーパラメーター (38 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
disableStreamCache (common)	サーブレットからの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultHttpBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。サーブレットを使用してエンドポイントをブリッジ/プロキシーする場合、メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にしてパフォーマンスを向上させることを検討してください。http/http4 プロデューサーは、デフォルトでレスポンスボディーストリームをキャッシュします。このオプションを true に設定すると、プロデューサは応答本文ストリームをキャッシュせず、応答ストリームをそのままメッセージ本文として使用します。	false	boolean
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
httpBinding (common)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI を要求に使用します。オプション throwExceptionOnFailure を false に設定して、HttpProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
chunked (producer)	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは、connectionClose は false です。	false	boolean

名前	説明	デフォルト	タイプ
copyHeaders (producer)	このオプションが true の場合、IN 交換ヘッダーは、コピー戦略に従って OUT 交換ヘッダーにコピーされます。これを false に設定すると、HTTP 応答からのヘッダーのみを含めることができます (IN ヘッダーは伝播されません)。	true	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。設定されている場合、HttpMethod ヘッダーはこのオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本文を読み取らず、入力ストリームをキャッシュしません。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在の Exchange Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受信した Host ヘッダーにアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで役立ちます。Host ヘッダーを使用するアプリケーションが、プロキシーされたサービスの正確な URL を生成できるようにします。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler

名前	説明	デフォルト	タイプ
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。	200-299	String
urlRewrite (producer)	**非推奨** カスタム org.apache.camel.component.http.UrlRewrite を参照して、エンドポイントをブリッジ/プロキシーするときに URL を書き換えることができます。詳細は、http://camel.apache.org/urlrewrite.html を参照してください。		UrlRewrite
httpClientConfigurer (advanced)	認証メカニズムなどを設定するために、プロデューサーまたはコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定戦略を登録します		HttpClientConfigurer
httpClientOptions (advanced)	マップのキー/値を使用して HttpClient を設定するには。		Map
httpConnectionManager (advanced)	カスタム HttpConnectionManager を使用して接続を管理する場合		HttpConnectionManager
httpConnectionManager Options (advanced)	マップのキー/値を使用して HttpConnectionManager を設定する場合		Map
mapHttpMessageBody (advanced)	このオプションが true の場合、交換の IN exchange ボディは HTTP ボディにマップされます。これを false に設定すると、HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、交換の IN exchange Form Encoded ボディは HTTP にマップされます。これを false に設定すると、HTTP Form Encoded ボディマッピングが回避されます。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、交換の IN exchange ヘッダーは HTTP ヘッダーにマップされます。これを false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
proxyAuthDomain (proxy)	NTML で使用するプロキシー認証ドメイン		String
proxyAuthHost (proxy)	プロキシー認証ホスト		String
proxyAuthMethod (proxy)	使用するプロキシー認証方法		String
proxyAuthPassword (proxy)	プロキシー認証パスワード		String
proxyAuthPort (proxy)	プロキシー認証ポート		int
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		String
proxyAuthUsername (proxy)	プロキシー認証のユーザー名		String
proxyHost (proxy)	使用するプロキシーホスト名		String
proxyPort (proxy)	使用するプロキシーポート		int
authDomain (security)	NTML で使用する認証ドメイン		String
authHost (security)	NTML で使用する認証ホスト		String
authMethod (security)	Basic、Digest、または NTLM の値のコンマ区切りリストとして使用できる認証方法。		String
authMethodPriority (security)	基本、ダイジェスト、または NTLM のいずれかとして、優先して使用する認証方法。		String
authPassword (security)	認証パスワード		String
authUsername (security)	認証ユーザー名		文字列

141.4. メッセージヘッダー

名前	タイプ	説明
Exchange.HTTP_URI	**String**	呼び出す URI。エンドポイントで直接設定された既存の URI をオーバーライドします。この uri は、呼び出す http サーバーの uri です。セキュリティーなどのエンドポイントオプションを設定できる Camel エンドポイント uri とは異なります。このヘッダーはそれをサポートしていません。http サーバーの唯一の uri です。
Exchange.HTTP_METHOD	**String**	HTTP メソッド/使用する動詞 (GET/POST/PUT/DELETE/HEAD/OPTIONS/TRACE)
Exchange.HTTP_PATH	**String**	リクエスト URI のパス。ヘッダーは、HTTP_URI でリクエスト URI を構築するために使用されます。**Camel 2.3.0:** パスが / で始まる場合には、http プロデューサーは Exchange.HTTP_BASE_URI ヘッダーまたは exchange.getFromEndpoint().getEndpointUri() に基づいて相対パスを見つけようとします。
Exchange.HTTP_QUERY	**String**	URI パラメーター。エンドポイントに直接設定された既存の URI パラメーターをオーバーライドします。
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_CHARACTER_ENCODING	**String**	文字エンコーディング。
Exchange.CONTENT_TYPE	**String**	HTTP コンテンツタイプ。**text/html** などのコンテンツタイプを提供するために、IN メッセージと OUT メッセージの両方に設定されます。

名前	タイプ	説明
Exchange.CONTENT_ENCODING	**String**	HTTP コンテンツエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために、IN メッセージと OUT メッセージの両方に設定されます。
Exchange.HTTP_SERVLET_REQUEST	**HttpServletRequest**	**HttpServletRequest** オブジェクト。
Exchange.HTTP_SERVLET_RESPONSE	**HttpServletResponse**	**HttpServletResponse** オブジェクト。
Exchange.HTTP_PROTOCOL_VERSION	**String**	**Camel 2.5:** このヘッダーで http プロトコルのバージョンを設定できます。"HTTP/1.0"。ヘッダーを指定しなかった場合、HttpProducer はデフォルト値の HTTP/1.1 を使用します。

上記のヘッダー名は定数です。Spring の DSL では、名前の代わりに定数の値を使用する必要があります。

141.5. メッセージボディー

Camel は、外部サーバーからの HTTP レスポンスを OUT ボディに保存します。IN メッセージのすべてのヘッダーが OUT メッセージにコピーされるため、ヘッダーはルーティング中に保持されます。さらに、Camel は HTTP 応答ヘッダーも OUT メッセージヘッダーに追加します。

141.6. レスポンスコード

Camel は HTTP 応答コードに従って処理します。

- 応答コードは 100..299 の範囲で、Camel はそれを成功応答と見なします。
- 応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答と見なし、情報とともに **HttpOperationFailedException** を出力します。

- 応答コードが 400+ の場合、Camel はこれを外部サーバーの障害と見なし、その情報とともに **HttpOperationFailedException** を出力します。

throwExceptionOnFailure

オプション **throwExceptionOnFailure** を **false** に設定して、失敗したレスポンスコードに対して **HttpOperationFailedException** が出力されないようにすることができます。これにより、リモートサーバーからの応答を取得できます。
これを示すサンプルを以下に示します。

141.7. HTTPOPERATIONFAILEDEXCEPTION

この例外には、次の情報が含まれています。

- HTTP ステータスコード
- HTTP ステータス行 (ステータスコードのテキスト)
- サーバーがリダイレクトを返した場合は、ロケーションをリダイレクトします
- サーバーがレスポンスとして本文を提供した場合、**java.lang.String** としてのレスポンス本文

141.8. どの HTTP メソッドを使用するか

次のアルゴリズムを使用して、使用する HTTP メソッドを決定します。
1.エンドポイント設定 (**httpMethod**) として提供されるメソッドを使用します。
2.ヘッダーで提供されるメソッドを使用します (**Exchange.HTTP_METHOD**)。
3.ヘッダーにクエリー文字列が指定されている場合は **GET**。
4.エンドポイントがクエリー文字列で設定されている場合は **GET**。
5.送信するデータがある場合は **POST** (本文が **null** ではない)。
6.それ以外の場合は **GET**。

141.9. HTTPSERVLETREQUEST と HTTPSERVLETRESPONSE にアクセスする方法

を使用して Camel 型コンバーターシステムを使用して、これら 2 つにアクセスできます。

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletRequest response = exchange.getIn().getBody(HttpServletResponse.class);
```

141.10. クライアントタイムアウトの使用 - SO_TIMEOUT

このリンク の単体テストを参照してください

141.11. その他の例

141.11.1. プロキシーの設定

Java DSL

```
from("direct:start")
  .to("http://oldhost?proxyHost=www.myproxy.com&proxyPort=80");
```

また、**proxyUsername** および **proxyPassword** オプションによるプロキシー認証もサポートされています。

141.11.2. URI の外部でプロキシー設定を使用する

Java DSL

Spring DSL

```
context.getProperties().put("http.proxyHost", "172.168.18.9");
context.getProperties().put("http.proxyPort" "8080");
```

```
<camelContext>
  <properties>
    <property key="http.proxyHost" value="172.168.18.9"/>
    <property key="http.proxyPort" value="8080"/>
  </properties>
</camelContext>
```

エンドポイントのオプションは、コンテキストのオプションをオーバーライドします。

141.12. 文字セットの設定

POST を使用してデータを送信している場合は、**charset** を設定できます

```
setProperty(Exchange.CHARSET_NAME, "iso-8859-1");
```

141.13. スケジュールされたポーリングのサンプル

このサンプルは、Google ホームページを 10 秒ごとにポーリングし、そのページをファイル **message.html** に書き込みます。

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
    .to("http://www.google.com")
    .setHeader(FileComponent.HEADER_FILE_NAME, "message.html").to("file:target/google");
```

141.14. 応答コードの取得

Exchange.HTTP_RESPONSE_CODE を使用して Out メッセージヘッダーから値を取得することにより、HTTP コンポーネントから HTTP 応答コードを取得できます。

```
Exchange exchange = template.send("http://www.google.com/search", new Processor() {
        public void process(Exchange exchange) throws Exception {
            exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
        }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

141.15. THROWEXCEPTIONONFAILURE=FALSE を使用して応答を取得する

以下のルートでは、リモート HTTP 呼び出しから返されたデータで強化するメッセージをルーティングします。リモートサーバーからの応答が必要なため、**throwExceptionOnFailure** オプションを **false** に設定して、**AggregationStrategy** でレスポンスを取得します。コードは HTTP ステータスコード 404 をシミュレートした単体テストに基づいているため、一部アサーションコードなどがあります。

141.16. クッキーの無効化

Cookie を無効にするには、次の URI オプションを追加して、HTTP クライアントが Cookie を無視するように設定します。
httpClient.cookiePolicy=ignoreCookies

141.17. 高度な使用方法

HTTP プロデューサーをさらに制御する必要がある場合は、さまざまなクラスを設定してカスタム動作を提供できる **HttpComponent** を使用する必要があります。

141.17.1. MaxConnectionsPerHost の設定

HTTP コンポーネントには、特定のコンポーネントのさまざまなグローバル設定を設定できる **org.apache.commons.httpclient.HttpConnectionManager** があります。
グローバルとは、コンポーネントが作成するすべてのエンドポイントが同じ共有 **HttpConnectionManager** を持つことを意味します。したがって、ホストごとの最大接続数に別の値を設定する場合は、通常使用するエンドポイント URI では **なく**、HTTP コンポーネントで定義する必要があります。ですから、次のようになります。

まず、Spring XML で **http** コンポーネントを定義します。はい、同じスキーム名 **http** を使用します。それ以外の場合、Camel は自動検出し、デフォルト設定でコンポーネントを作成します。必要なのは、これを却下して、オプションを設定できるようにすることです。以下のサンプルでは、最大接続数をデフォルトの 2 ではなく 5 に設定しています。

そして、ルートで通常どおりに使用できます。

141.17.2. プリエンプティブ認証の使用

あるエンドユーザーが、HTTPS での認証に問題があると報告しました。HTTPS サーバーが HTTP コード 401 Authorization Required を返さないことを発見したとき、問題は最終的に解決されました。解決策は、次の URI オプションを設定することでした: **httpClient.authenticationPreemptive=true**

141.17.3. リモートサーバーからの自己署名証明書の受け入れ

Apache Commons HTTP API でこれを行う方法を概説するいくつかのコードを含むメーリングリストディスカッションのこの リンク を参照してください。

141.17.4. HTTP クライアントの SSL の設定

JSSE 設定ユーティリティーの使用

Camel 2.8 の時点で、HTTP4 コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、HTTP4 コンポーネントでユーティリティーを使用する方法を示しています。

このコンポーネントで使用される Apache HTTP クライアントのバージョンは、グローバルなプロトコルレジストリーから SSL/TLS 情報を解決します。 このコンポーネントは、Camel JSSE 設定ユーティ

リティーの使用をサポートするために、HTTP クライアントのプロトコルソケットファクトリーの実装 **org.apache.camel.component.http.SSLContextParametersSecureProtocolSocketFactory** を提供します。 次の例は、プロトコルレジストリーを設定し、登録されたプロトコル情報をルートで使用する方法を示しています。

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

ProtocolSocketFactory factory =
   new SSLContextParametersSecureProtocolSocketFactory(scp);

Protocol.registerProtocol("https",
       new Protocol(
       "https",
       factory,
       443));

from("direct:start")
       .to("https://mail.google.com/mail/").to("mock:results");
```

Apache HTTP クライアントを直接設定する

基本的に camel-http コンポーネントは Apache HTTP クライアントの上に構築されており、カスタム **org.apache.camel.component.http.HttpClientConfigurer** を実装して、http クライアントを完全に制御する必要がある場合に設定を行うことができます。

ただし、キーストアとトラストストアを指定する **だけ** の場合は、Apache HTTP **HttpClientConfigurer** を使用してこれを行うことができます。次に例を示します。

```
Protocol authhttps = new Protocol("https", new AuthSSLProtocolSocketFactory(
  new URL("file:my.keystore"), "mypassword",
  new URL("file:my.truststore"), "mypassword"), 443);

Protocol.registerProtocol("https", authhttps);
```

次に、**HttpClientConfigurer** を実装するクラスを作成し、上記の例に従ってキーストアまたはトラストストアを提供する https プロトコルを登録する必要があります。次に、キャメルルートビルダークラスから次のように接続できます。

```
HttpComponent httpComponent = getContext().getComponent("http", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

Spring DSL を使用してこれを行う場合、URI を使用して **HttpClientConfigurer** を指定できます。以下に例を示します。

```
<bean id="myHttpClientConfigurer"
 class="my.https.HttpClientConfigurer">
```

```
</bean>

<to uri="https://myhostname.com:443/myURL?httpClientConfigurerRef=myHttpClientConfigurer"/>
```

上記のように HttpClientConfigurer を実装し、キーストアとトラストストアを設定する限り、問題なく動作します。

141.18. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Jetty

第142章 HTTP4 コンポーネント

Camel バージョン 2.3 以降で利用可能

http4: コンポーネントは、外部 HTTP リソースを呼び出すための HTTP ベースのエンドポイントを提供します (HTTP を使用して外部サーバーを呼び出すクライアントとして)。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-http4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-http4 vs camel-http

Camel-http4 は Apache HttpClient 4.x を使用し、camel-http は Apache HttpClient 3.x を使用します。

142.1. URI 形式

```
http4:hostname[:port][/resourceUri][?options]
```

デフォルトでは、HTTP にはポート 80、HTTPS には 443 を使用します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

camel-http4 と camel-jetty の比較

HTTP4 コンポーネントによって生成されたエンドポイントに対してのみ生成できます。したがって、Camel ルートへの入力として使用しないでください。Camel ルートへの入力として HTTP サーバー経由で HTTP エンドポイントをバインド/公開するには、代わりに Jetty コンポーネント を使用します。

142.2. HTTP4 コンポーネントオプション

HTTP4 コンポーネントは、以下に示す 18 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
httpClientConfigurer (advanced)	カスタム HttpClientConfigurer を使用して、使用される HttpClient の設定を実行します。		HttpClientConfigurer
clientConnectionManager (advanced)	カスタムおよび共有の HttpClientConnectionManager を使用して接続を管理するには。これが設定されている場合、これは、このコンポーネントによって作成されたすべてのエンドポイントに常に使用されます。		HttpClientConnection マネージャー

名前	説明	デフォルト	タイプ
httpContext (advanced)	リクエストの実行時にカスタム org.apache.http.protocol.HttpContext を使用するには。		HttpContext
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。重要: HttpComponent ごとにサポートされる org.apache.camel.util.jsse.SSLContextParameters のインスタンスは 1 つだけです。2 つ以上の異なるインスタンスを使用する必要がある場合は、必要なインスタンスごとに新しい HttpComponent を定義する必要があります。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
x509HostnameVerifier (security)	DefaultHostnameVerifier や org.apache.http.conn.ssl.NoopHostnameVerifier などのカスタム X509HostnameVerifier を使用するには。		HostnameVerifier
maxTotalConnections (advanced)	接続の最大数。	200	int
connectionsPerRoute (advanced)	ルートごとの接続の最大数。	20	int
connectionTimeToLive (advanced)	接続が有効になるまでの時間。単位はミリ秒で、デフォルト値は常にキープアライブです。		long
cookieStore (producer)	カスタム org.apache.http.client.CookieStore を使用するには。デフォルトでは、メモリー内のみの Cookie ストアである org.apache.http.impl.client.BasicCookieStore が使用されます。bridgeEndpoint=true の場合、Cookie ストアは強制的に noop cookie ストアになることに注意してください。これは、単にブリッジしているだけであるため (たとえば、プロキシーとして動作するため)、Cookie を保存するべきではないためです。		CookieStore
connectionRequest Timeout (タイムアウト)	接続マネージャーからの接続を要求するときに使用されるミリ秒単位のタイムアウト。タイムアウト値 0 は無限のタイムアウトとして解釈されます。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は未定義として解釈されます (該当する場合はシステムのデフォルト)。デフォルト: コード -1	-1	int

名前	説明	デフォルト	タイプ
connectTimeout (タイムアウト)	接続が確立されるまでのタイムアウトをミリ秒単位で決定します。タイムアウト値 0 は無限のタイムアウトとして解釈されます。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は未定義として解釈されます (該当する場合はシステムのデフォルト)。デフォルト: コード -1	-1	int
socketTimeout (タイムアウト)	ソケットのタイムアウト (SO_TIMEOUT) をミリ秒単位で定義します。これは、データを待機するためのタイムアウト、または別の言い方をすれば、2 つの連続するデータパケット間の最大非アクティブ期間です。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は未定義として解釈されます (該当する場合はシステムのデフォルト)。デフォルト: コード -1	-1	int
httpBinding (advanced)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration を基本設定として使用するには、以下を行います。		HttpConfiguration
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に Java シリアル化を許可するかどうか。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HTTP4 エンドポイントは、URI 構文を使用して設定されます。

```
http4:httpUri
```

パスおよびクエリーパラメーターを使用します。

142.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpUri	**必須** 呼び出す HTTP エンドポイントの URL。		URI

142.2.2. クエリーパラメーター (48 パラメーター)

名前	説明	デフォルト	タイプ
disableStreamCache (common)	サーブレットからの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultHttpBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。サーブレットを使用してエンドポイントをブリッジ/プロキシーする場合、メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にしてパフォーマンスを向上させることを検討してください。http/http4 プロデューサーは、デフォルトでレスポンスボディーストリームをキャッシュします。このオプションを true に設定すると、プロデューサは応答本文ストリームをキャッシュせず、応答ストリームをそのままメッセージ本文として使用します。	false	boolean
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
httpBinding (common)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
authenticationPreemptive (producer)	このオプションが true の場合、camel-http4 はプリエンプティブ基本認証をサーバーに送信します。	false	boolean

名前	説明	デフォルト	タイプ
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI を要求に使用します。オプション throwExceptionOnFailure を false に設定して、HttpProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
chunked (producer)	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
clearExpiredCookies (producer)	HTTP リクエストを送信する前に期限切れの Cookie をクリアするかどうか。これにより、有効期限が切れたときに削除される新しい Cookie を追加することで、Cookie ストアが成長し続けることがなくなります。	true	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは、connectionClose は false です。	false	boolean
cookieStore (producer)	カスタム CookieStore を使用するには。デフォルトでは、メモリー内のみの Cookie ストアである BasicCookieStore が使用されます。bridgeEndpoint=true の場合、Cookie ストアは強制的に noop cookie ストアになることに注意してください。これは、単にブリッジしているだけであるため (たとえば、プロキシーとして動作するため)、Cookie を保存するべきではないためです。cookieHandler が設定されている場合、cookie の処理は cookieHandler によって実行されるため、cookie ストアも強制的に noop cookie ストアになります。		CookieStore
copyHeaders (producer)	このオプションが true の場合、IN 交換ヘッダーは、コピー戦略に従って OUT 交換ヘッダーにコピーされます。これを false に設定すると、HTTP 応答からのヘッダーのみを含めることができます (IN ヘッダーは伝播されません)。	true	boolean
deleteWithBody (producer)	HTTP DELETE にメッセージ本文を含めるかどうか。デフォルトでは、HTTP DELETE には HTTP メッセージは含まれません。ただし、ごくまれに、ユーザーがメッセージ本文を含める必要がある場合があります。	false	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。設定されている場合、HttpMethod ヘッダーはこのオプションをオーバーライドできません。		HttpMethods

名前	説明	デフォルト	タイプ
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本文を読み取らず、入力ストリームをキャッシュしません。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在の Exchange Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受信した Host ヘッダーにアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで役立ちます。Host ヘッダーを使用するアプリケーションが、プロキシーされたサービスの正確な URL を生成できるようにします。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。	200-299	String
urlRewrite (producer)	**非推奨** カスタム org.apache.camel.component.http.UrlRewrite を参照して、エンドポイントをブリッジ/プロキシーするときに URL を書き換えることができます。詳細は、http://camel.apache.org/urlrewrite.html を参照してください。		UrlRewrite

名前	説明	デフォルト	タイプ
clientBuilder (advanced)	このエンドポイントのプロデューサーまたはコンシューマーによって使用される新しい RequestConfig インスタンスで使用される http クライアント要求パラメーターへのアクセスを提供します。		HttpClientBuilder
clientConnectionManager (advanced)	カスタム HttpClientConnectionManager を使用して接続を管理するには		HttpClientConnection マネージャー
connectionsPerRoute (advanced)	ルートごとの接続の最大数。	20	int
httpClient (advanced)	プロデューサーが使用するカスタム HttpClient を設定します		HttpClient
httpClientConfigurer (advanced)	認証メカニズムなどを設定するために、プロデューサーまたはコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定戦略を登録します		HttpClientConfigurer
httpClientOptions (advanced)	マップのキー/値を使用して HttpClient を設定するには。		Map
httpContext (advanced)	カスタム HttpContext インスタンスを使用します		HttpContext
mapHttpMessageBody (advanced)	このオプションが true の場合、交換の IN exchange ボディは HTTP ボディにマップされます。これを false に設定すると、HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、交換の IN exchange Form Encoded ボディは HTTP にマップされます。これを false に設定すると、HTTP Form Encoded ボディマッピングが回避されます。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、交換の IN exchange ヘッダーは HTTP ヘッダーにマップされます。これを false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
maxTotalConnections (advanced)	接続の最大数。	200	int

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
useSystemProperties (advanced)	システムプロパティーを設定のフォールバックとして使用します	false	boolean
proxyAuthDomain (proxy)	NTML で使用するプロキシー認証ドメイン		String
proxyAuthHost (proxy)	プロキシー認証ホスト		String
proxyAuthMethod (proxy)	使用するプロキシー認証方法		String
proxyAuthPassword (proxy)	プロキシー認証パスワード		String
proxyAuthPort (proxy)	プロキシー認証ポート		int
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		String
proxyAuthUsername (proxy)	プロキシー認証のユーザー名		String
proxyHost (proxy)	使用するプロキシーホスト名		String
proxyPort (proxy)	使用するプロキシーポート		int
authDomain (security)	NTML で使用する認証ドメイン		String
authHost (security)	NTML で使用する認証ホスト		String
authMethod (security)	Basic、Digest、または NTLM の値のコンマ区切りリストとして使用できる認証方法。		String
authMethodPriority (security)	基本、ダイジェスト、または NTLM のいずれかとして、優先して使用する認証方法。		String
authPassword (security)	認証パスワード		String

名前	説明	デフォルト	タイプ
authUsername (security)	認証ユーザー名		文字列
x509HostnameVerifier (security)	DefaultHostnameVerifier や org.apache.http.conn.ssl.NoopHostnameVerifier などのカスタム X509HostnameVerifier を使用するには。		HostnameVerifier

142.3. メッセージヘッダー

名前	タイプ	説明
Exchange.HTTP_URI	**String**	呼び出す URI。エンドポイントで直接設定された既存の URI をオーバーライドします。この uri は、呼び出す http サーバーの uri です。セキュリティーなどのエンドポイントオプションを設定できる Camel エンドポイント uri とは異なります。このヘッダーはそれをサポートしていません。http サーバーの唯一の uri です。
Exchange.HTTP_PATH	**String**	リクエスト URI のパス。ヘッダーは、HTTP_URI でリクエスト URI を構築するために使用されます。
Exchange.HTTP_QUERY	**String**	URI パラメーター。エンドポイントに直接設定された既存の URI パラメーターをオーバーライドします。
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_RESPONSE_TEXT	**String**	外部サーバーからの HTTP 応答テキスト。

名前	タイプ	説明
Exchange.HTTP_CHARACTER_ENCODING	**String**	文字エンコーディング。
Exchange.CONTENT_TYPE	**String**	HTTP コンテンツタイプ。**text/html** などのコンテンツタイプを提供するために、IN メッセージと OUT メッセージの両方に設定されます。
Exchange.CONTENT_ENCODING	**String**	HTTP コンテンツエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために、IN メッセージと OUT メッセージの両方に設定されます。

142.4. メッセージボディー

Camel は、外部サーバーからの HTTP レスポンスを OUT ボディに保存します。IN メッセージのすべてのヘッダーが OUT メッセージにコピーされるため、ヘッダーはルーティング中に保持されます。さらに、Camel は HTTP 応答ヘッダーも OUT メッセージヘッダーに追加します。

142.5. システムプロパティーの使用

useSystemProperties を true に設定すると、HTTP クライアントは次のシステムプロパティーを探して使用します。

- ssl.TrustManagerFactory.algorithm
- javax.net.ssl.trustStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStoreProvider
- javax.net.ssl.trustStorePassword
- java.home
- ssl.KeyManagerFactory.algorithm
- javax.net.ssl.keyStoreType

- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreProvider
- javax.net.ssl.keyStorePassword
- http.proxyHost
- http.proxyPort
- http.nonProxyHosts
- http.keepAlive
- http.maxConnections

142.6. レスポンスコード

Camel は HTTP 応答コードに従って処理します。

- 応答コードは 100..299 の範囲で、Camel はそれを成功応答と見なします。
- 応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答と見なし、情報とともに **HttpOperationFailedException** を出力します。
- 応答コードが 400+ の場合、Camel はこれを外部サーバーの障害と見なし、その情報とともに **HttpOperationFailedException** を出力します。

throwExceptionOnFailure オプション **throwExceptionOnFailure** を **false** に設定して、失敗した応答コードに対して **HttpOperationFailedException** が出力されないようにすることができます。これにより、リモートサーバーからの応答を取得できます。
これを示すサンプルを以下に示します。

142.7. HTTPOPERATIONFAILEDEXCEPTION

この例外には、次の情報が含まれています。

- HTTP ステータスコード
- HTTP ステータス行 (ステータスコードのテキスト)
- サーバーがリダイレクトを返した場合は、ロケーションをリダイレクトします
- サーバーがレスポンスとして本文を提供した場合、**java.lang.String** としてのレスポンス本文

142.8. どの HTTP メソッドを使用するか

次のアルゴリズムを使用して、使用する HTTP メソッドを決定します。
1.エンドポイント設定 (**httpMethod**) として提供されるメソッドを使用します。
2.ヘッダーで提供されるメソッドを使用します (**Exchange.HTTP_METHOD**)。
3.ヘッダーにクエリー文字列が指定されている場合は **GET**。
4.エンドポイントがクエリー文字列で設定されている場合は **GET**。
5.送信するデータがある場合は **POST** (本文が **null** ではない)。
6.それ以外の場合は **GET**。

142.9. HTTPSERVLETREQUEST と HTTPSERVLETRESPONSE にアクセスする方法

を使用して Camel 型コンバーターシステムを介し、これら 2 つにアクセスできます。**注記** camel-jetty または camel-cxf エンドポイントの後のプロセッサーからだけでなく、リクエストとレスポンスを取得できます。

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletRequest response = exchange.getIn().getBody(HttpServletResponse.class);
```

142.10. 呼び出す URI の設定

HTTP プロデューサーの URI は、エンドポイント URI から直接設定できます。以下のルートでは、Camel は HTTP を使用して外部サーバー **oldhost** を呼び出します。

```
from("direct:start")
        .to("http4://oldhost");
```

同等の Spring の例:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="http4://oldhost"/>
  </route>
</camelContext>
```

メッセージにキー **Exchange.HTTP_URI** を含むヘッダーを追加することで、HTTP エンドポイント URI をオーバーライドできます。

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
  .to("http4://oldhost");
```

上記のサンプルでは、エンドポイントが http4://oldhost で設定されているにもかかわらず、Camel は http://newhost を呼び出します。
http4 エンドポイントがブリッジモードで動作している場合、**Exchange.HTTP_URI** のメッセージヘッダーは無視されます。

142.11. URI パラメーターの設定

http プロデューサーは、HTTP サーバーに送信される URI パラメーターをサポートしています。URI パラメーターは、エンドポイント URI に直接設定するか、メッセージのキー **Exchange.HTTP_QUERY** を含むヘッダーとして設定できます。

```
from("direct:start")
  .to("http4://oldhost?order=123&detail=short");
```

または、ヘッダーで提供されるオプション:

```
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http4://oldhost");
```

142.12. HTTP プロデューサに HTTP メソッド (GET/PATCH/POST/PUT/DELETE/HEAD/OPTIONS/TRACE) を設定する方法

http PATCH メソッドの使用

http PATCH メソッドは、Camel 2.11.3/2.12.1 以降でサポートされています。

HTTP4 コンポーネントは、メッセージヘッダーを定めることにより、HTTP リクエストメソッドを設定する方法を提供します。以下に例を示します。

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD,
constant(org.apache.camel.component.http4.HttpMethods.POST))
  .to("http4://www.google.com")
  .to("mock:results");
```

このメソッドは、文字列定数を使用して少し短く書くことができます。

```
.setHeader("CamelHttpMethod", constant("POST"))
```

同等の Spring の例:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelHttpMethod">
        <constant>POST</constant>
    </setHeader>
    <to uri="http4://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

142.13. クライアントタイムアウトの使用 - SO_TIMEOUT

HttpSOTimeoutTest 単体テストを参照してください。

Camel 2.13.0 以降: 更新された HttpSOTimeoutTest 単体テストを参照してください。

142.14. プロキシーの設定

HTTP4 コンポーネントは、プロキシーを設定する方法を提供します。

```
from("direct:start")
  .to("http4://oldhost?proxyAuthHost=www.myproxy.com&proxyAuthPort=80");
```

また、**proxyAuthUsername** および **proxyAuthPassword** オプションによるプロキシー認証もサポートされています。

142.14.1. URI の外部でプロキシー設定を使用する

システムプロパティーの競合を避けるために、CamelContext または URI からのみプロキシー設定を設定できます。
Java DSL:

```
context.getProperties().put("http.proxyHost", "172.168.18.9");
context.getProperties().put("http.proxyPort" "8080");
```

Spring XML

```
<camelContext>
  <properties>
    <property key="http.proxyHost" value="172.168.18.9"/>
    <property key="http.proxyPort" value="8080"/>
  </properties>
</camelContext>
```

Camel は最初に Java システムまたは CamelContext プロパティーから設定を行い、次にエンドポイントプロキシーオプションが提供されている場合はそれを設定します。
そのため、システムプロパティーをエンドポイントオプションでオーバーライドできます。

Camel 2.8 には、使用するスキームを明示的に設定できる **http.proxyScheme** プロパティーもあります。

142.15. 文字セットの設定

POST を使用してデータを送信している場合は、**Exchange** プロパティーを使用して **charset** を設定できます。

```
exchange.setProperty(Exchange.CHARSET_NAME, "ISO-8859-1");
```

142.15.1. スケジュールされたポーリングのサンプル

このサンプルは、Google ホームページを 10 秒ごとにポーリングし、そのページをファイル **message.html** に書き込みます。

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
  .to("http4://www.google.com")
  .setHeader(FileComponent.HEADER_FILE_NAME, "message.html")
  .to("file:target/google");
```

142.15.2. エンドポイント URI からの URI パラメーター

このサンプルには、Web ブラウザーに入力したものとまったく同じ完全な URI エンドポイントがあります。もちろん、Web ブラウザーと同じように **&** 文字を区切り文字として使用して、複数の URI パラメーターを設定できます。Camel はここでトリックを行いません。

```
// we query for Camel at the Google page
template.sendBody("http4://www.google.com/search?q=Camel", null);
```

142.15.3. メッセージからの URI パラメーター

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("http4://www.google.com/search", null, headers);
```

上記のヘッダー値では、**?** を接頭辞として付けるべきでは **ない** ことに注意してください。**&** 文字を使用して、通常どおりパラメーターを区切ることができます。

142.15.4. 応答コードの取得

Exchange.HTTP_RESPONSE_CODE を使用して Out メッセージヘッダーから値を取得することにより、HTTP4 コンポーネントから HTTP レスポンスコードを取得できます。

```
Exchange exchange = template.send("http4://www.google.com/search", new Processor() {
  public void process(Exchange exchange) throws Exception {
    exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
  }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

142.16. クッキーの無効化

Cookie を無効にするには、次の URI オプションを追加して、HTTP クライアントが Cookie を無視するように設定します。
httpClient.cookiePolicy=ignoreCookies

142.17. 高度な使用方法

HTTP プロデューサーをさらに制御する必要がある場合は、さまざまなクラスを設定してカスタム動作を提供できる **HttpComponent** を使用する必要があります。

142.17.1. HTTP クライアントの SSL の設定

JSSE 設定ユーティリティーの使用

Camel 2.8 の時点で、HTTP4 コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、HTTP4 コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");
```

```
KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

HttpComponent httpComponent = getContext().getComponent("https4", HttpComponent.class);
httpComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="https4://127.0.0.1/mail/?sslContextParameters=#sslContextParameters"/>...
```

Apache HTTP クライアントを直接設定する

基本的に camel-http4 コンポーネントは Apache HttpClient の上に構築されます。詳細については、SSL/TLS customization を参照するか、**org.apache.camel.component.http4.HttpsServerTestSupport** 単体テスト基本クラスを調べてください。
カスタム **org.apache.camel.component.http4.HttpClientConfigurer** を実装して、http クライアントを完全に制御する必要がある場合は、その設定を行うこともできます。

ただし、キーストアとトラストストアを指定する **だけ** の場合は、Apache HTTP **HttpClientConfigurer** を使用してこれを行うことができます。次に例を示します。

```
KeyStore keystore = ...;
KeyStore truststore = ...;

SchemeRegistry registry = new SchemeRegistry();
registry.register(new Scheme("https", 443, new SSLSocketFactory(keystore, "mypassword",
truststore)));
```

次に、**HttpClientConfigurer** を実装するクラスを作成し、上記の例に従ってキーストアまたはトラストストアを提供する https プロトコルを登録する必要があります。次に、キャメルルートビルダークラスから次のように接続できます。

```
HttpComponent httpComponent = getContext().getComponent("http4", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

Spring DSL を使用してこれを行う場合、URI を使用して **HttpClientConfigurer** を指定できます。以下に例を示します。

```
<bean id="myHttpClientConfigurer"
```

```
 class="my.https.HttpClientConfigurer">
</bean>

<to uri="https4://myhostname.com:443/myURL?httpClientConfigurer=myHttpClientConfigurer"/>
```

上記のように HttpClientConfigurer を実装し、キーストアとトラストストアを設定する限り、問題なく動作します。

HTTPS を使用して落とし穴を認証する

あるエンドユーザーが、HTTPS での認証に問題があると報告しました。この問題は、カスタム設定の **org.apache.http.protocol.HttpContext** を提供することで最終的に解決されました。

- 1.HttpContexts の (Spring) ファクトリーを作成します。

```
public class HttpContextFactory {

  private String httpHost = "localhost";
  private String httpPort = 9001;

  private BasicHttpContext httpContext = new BasicHttpContext();
  private BasicAuthCache authCache = new BasicAuthCache();
  private BasicScheme basicAuth = new BasicScheme();

  public HttpContext getObject() {
    authCache.put(new HttpHost(httpHost, httpPort), basicAuth);

    httpContext.setAttribute(ClientContext.AUTH_CACHE, authCache);

    return httpContext;
  }

  // getter and setter
}
```

- 2.Spring アプリケーションコンテキストファイルで HttpContext を宣言します。

```
<bean id="myHttpContext" factory-bean="httpContextFactory" factory-method="getObject"/>
```

- 3.http4 URL でコンテキストを参照します。

```
<to uri="https4://myhostname.com:443/myURL?httpContext=myHttpContext"/>
```

異なる SSLContextParameters の使用

HTTP4 コンポーネントは、コンポーネントごとに **org.apache.camel.util.jsse.SSLContextParameters** の 1 つのインスタンスのみをサポートします。2 つ以上の異なるインスタンスを使用する必要がある場合は、以下に示すように複数の HTTP4 コンポーネントを設定する必要があります。2 つのコンポーネントがあり、それぞれが **sslContextParameters** プロパティーの独自のインスタンスを使用しています。

```
<bean id="http4-foo" class="org.apache.camel.component.http4.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams1"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>
```

```
<bean id="http4-bar" class="org.apache.camel.component.http4.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams2"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>
```

第143章 HYSTRIX コンポーネント

Camel バージョン 2.18 以降で利用可能

hystrix コンポーネントは、キャメルルートに Netflix Hystrix サーキットブレーカーを統合します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hystrix</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細は、Hystrix EIP を参照してください。

第144章 ICAL DATAFORMAT

Camel バージョン 2.12 以降で利用可能

ICal データ形式は、iCalendar メッセージの操作に使用されます。

典型的な iCalendar メッセージは次のようになります。

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Events Calendar//iCal4j 1.0//EN
CALSCALE:GREGORIAN
BEGIN:VEVENT
DTSTAMP:20130324T180000Z
DTSTART:20130401T170000
DTEND:20130401T210000
SUMMARY:Progress Meeting
TZID:America/New_York
UID:00000000
ATTENDEE;ROLE=REQ-PARTICIPANT;CN=Developer 1:mailto:dev1@mycompany.com
ATTENDEE;ROLE=OPT-PARTICIPANT;CN=Developer 2:mailto:dev2@mycompany.com
END:VEVENT
END:VCALENDAR
```

144.1. オプション

iCal データ形式は、以下にリストされている 2 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
検証中	**false**	**Boolean**	検証するかどうか。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

144.2. 基本的な使用方法

上記のメッセージをアンマーシャリングしてマーシャリングするには、ルートは次のようになります。

```
from("direct:ical-unmarshal")
  .unmarshal("ical")
  .to("mock:unmarshaled")
  .marshal("ical")
  .to("mock:marshaled");
```

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ical</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

144.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第145章 IEC 60870 CLIENT コンポーネント

Camel バージョン 2.20 以降で利用可能

IEC 60870-5-104 クライアント コンポーネントは、Eclipse NeoSCADA™ 実装を使用して IEC 60870 サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-iec60870</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

IEC 60870 クライアントコンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
defaultConnection Options (Common)	デフォルトの接続オプション		ClientOptions
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

145.1. URI 形式

エンドポイントの URI 構文は次のとおりです。

```
iec60870-client:host:port/00-01-02-03-04
```

情報オブジェクトのアドレスは、上記の構文でパスにエンコードされます。常に完全な 5 オクテットのアドレス形式が使用されていることに注意してください。未使用のオクテットはゼロで埋める必要があります。

145.2. URI オプション

IEC 60870 クライアントエンドポイントは、URI 構文を使用して設定されます。

```
iec60870-client:uriPath
```

パスおよびクエリーパラメーターを使用します。

145.2.1. パスパラメーター (1 個のパラメーター):

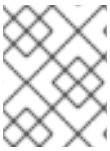
名前	説明	デフォルト	タイプ
uriPath	**必須** オブジェクト情報アドレス		ObjectAddress

145.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
dataModuleOptions (Common)	データモジュールオプション		DataModuleOptions
protocolOptions (Common)	プロトコルオプション		ProtocolOptions
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
acknowledgeWindow (接続)	パラメーター W - 確認ウィンドウ。	10	short
adsuAddressType (接続)	一般的な ASDU アドレスサイズ。SIZE_1 または SIZE_2 のいずれかです。		ASDUAddressType
causeOfTransmissionType (connection)	透過型の原因。SIZE_1 または SIZE_2 のいずれかです。		CauseOfTransmission 型

名前	説明	デフォルト	タイプ
informationObjectAddress Type (接続)	情報アドレスのサイズ。SIZE_1、SIZE_2、または SIZE_3 のいずれかです。		InformationObjectAddressType
maxUnacknowledged (接続)	パラメーター K - 未確認メッセージの最大数。	15	short
timeout1 (接続)	ミリ秒単位のタイムアウト T1。	15000	int
timeout2 (接続)	ミリ秒単位のタイムアウト T2。	10000	int
timeout3 (接続)	ミリ秒単位のタイムアウト T3。	20000	int
ignoreBackgroundScan (data)	バックグラウンドスキャン送信を無視するかどうか。	true	boolean
ignoreDaylightSavingTime (data)	DST を無視するか尊重するか	false	boolean
timeZone (data)	使用するタイムゾーン。任意の Java タイムゾーン文字列を指定できます	UTC	TimeZone
connectionId (id)	接続インスタンスをグループ化する識別子		文字列

URI のホストとポートの部分、および id グループのすべてのパラメーターによって識別される場合は、接続インスタンス。新しい接続 ID が検出されると、接続オプションが評価され、それらのオプションを使用して接続インスタンスが作成されます。



注記

2 つの URI が同じ接続 (ホスト、ポートなど) を指定しているが、接続オプションが異なる場合、それらの接続オプションのどちらが使用されるかは未定義です。

最終的な接続オプションは、次の順序で評価されます。

- 存在する場合、connectionOptions パラメーターが使用されます
- それ以外の場合は、次の手順で defaultConnectionOptions インスタンスがコピーされ、カスタマイズされます。
- 存在する場合は protocolOptions を適用します
- 存在する場合は dataModuleOptions を適用します
- すべての明示的な接続パラメーター (timeZone など) を適用します。

第146章 IEC 60870 SERVER コンポーネント

Camel バージョン 2.20 以降で利用可能

IEC 60870-5-104 サーバー コンポーネントは、Eclipse NeoSCADA™ 実装を使用して IEC 60870 サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-iec60870</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

IEC 60870 サーバーコンポーネントは、以下に示す 2個のオプションをサポートします。

名前	説明	デフォルト	タイプ
defaultConnection Options (Common)	デフォルトの接続オプション		ServerOptions
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

146.1. URI 形式

エンドポイントの URI 構文は次のとおりです。

```
iec60870-server:host:port/00-01-02-03-04
```

情報オブジェクトのアドレスは、上記の構文でパスにエンコードされます。常に完全な 5 オクテットのアドレス形式が使用されていることに注意してください。未使用のオクテットはゼロで埋める必要があります。

146.2. URI オプション

IEC 60870 サーバーエンドポイントは、URI 構文を使用して設定されます。

```
iec60870-server:uriPath
```

パスおよびクエリーパラメーターを使用します。

146.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
uriPath	**必須** オブジェクト情報アドレス		ObjectAddress

146.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
dataModuleOptions (Common)	データモジュールオプション		DataModuleOptions
filterNonExecute (Common)	実行ビットが設定されていないすべてのリクエストを除外します	true	boolean
protocolOptions (Common)	プロトコルオプション		ProtocolOptions
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
acknowledgeWindow (接続)	パラメーター W - 確認ウィンドウ。	10	short
adsuAddressType (接続)	一般的な ASDU アドレスサイズ。SIZE_1 または SIZE_2 のいずれかです。		ASDUAddressType

名前	説明	デフォルト	タイプ
causeOfTransmissionType (connection)	透過型の原因。SIZE_1 または SIZE_2 のいずれかです。		CauseOfTransmission 型
informationObjectAddress Type (接続)	情報アドレスのサイズ。SIZE_1、SIZE_2、または SIZE_3 のいずれかです。		InformationObjectAddressType
maxUnacknowledged (接続)	パラメーター K - 未確認メッセージの最大数。	15	short
timeout1 (接続)	ミリ秒単位のタイムアウト T1。	15000	int
timeout2 (接続)	ミリ秒単位のタイムアウト T2。	10000	int
timeout3 (接続)	ミリ秒単位のタイムアウト T3。	20000	int
ignoreBackgroundScan (data)	バックグラウンドスキャン送信を無視するかどうか。	true	boolean
ignoreDaylightSavingTime (data)	DST を無視するか尊重するか	false	boolean
timeZone (data)	使用するタイムゾーン。任意の Java タイムゾーン文字列を指定できます	UTC	TimeZone
connectionId (id)	接続インスタンスをグループ化する識別子		文字列

第147章 IGNITE CACHE コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Cache エンドポイントは、Ignite Cache との対話を可能にする camel-ignite エンドポイントの 1 つです。これにより、プロデューサー (Ignite キャッシュでキャッシュ操作を呼び出すため) とコンシューマー (継続的なクエリーからの変更を使用するため) の両方が提供されます。

キャッシュ値は常にメッセージのボディーですが、キャッシュキーは常に **IgniteConstants.IGNITE_CACHE_KEY** メッセージヘッダーに格納されます。

エンドポイント URI で固定操作を設定する場合でも、**IgniteConstants.IGNITE_CACHE_OPERATION** メッセージヘッダーを設定することでエクスチェンジごとに変更できます。

147.1. オプション

Ignite Cache コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (Common)	Ignite インスタンスを設定します。		Ignite
configurationResource (Common)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (Common)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Cache エンドポイントは、URI 構文を使用して設定されます。

```
ignite-cache:cacheName
```

パスおよびクエリーパラメーターを使用します。

147.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	***必須*** キャッシュ名。		String

147.1.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
propagateIncomingBodyIfNoReturnValue (Common)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (Common)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
autoUnsubscribe (consumer)	連続クエリーコンシューマーで自動サブスクライブ解除が有効になっているかどうか。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
fireExistingQueryResults (consumer)	クエリーに一致する既存の結果を処理するかどうか。連続クエリーコンシューマーの初期化で使用されます。	false	boolean
oneExchangePerUpdate (consumer)	複数の更新が 1 つのバッチで受信された場合でも、各更新を個別のエクスチェンジにパックするかどうか。連続クエリーコンシューマーによってのみ使用されます。	true	boolean
pageSize (consumer)	ページサイズ。連続クエリーコンシューマーによってのみ使用されます。	1	int
query (consumer)	実行するクエリー。それを必要とする操作と、連続クエリーコンシューマーにのみ必要です。		Object>>
remoteFilter (consumer)	連続クエリーコンシューマーによってのみ使用されるリモートフィルター。		Object>
timeInterval (consumer)	連続クエリーコンシューマーの時間間隔。	0	long

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cachePeekMode (producer)	CachePeekMode は、それを必要とする操作にのみ必要です (リンク IgniteCacheOperationSIZE)。	ALL	CachePeekMode
failIfInexistentCache (producer)	キャッシュが存在しない場合に初期化を失敗させるかどうか。	false	boolean
operation (producer)	呼び出すキャッシュ操作。設定可能な値: GET、PUT、REMOVE、SIZE、REBALANCE、QUERY、CLEAR。		IgniteCacheOperation
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

147.1.3. 使用されるヘッダー

このエンドポイントは、次のヘッダーを使用します。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteCacheKey	IgniteConstants.IGNITE_CACHE_KEY	String	メッセージ本文のエントリー値のキャッシュキー。
CamelIgniteCacheQuery	IgniteConstants.IGNITE_CACHE_QUERY	Query	QUERY 操作を呼び出すときに実行する (プロデューサー) クエリー。
CamelIgniteCacheOperation	IgniteConstants.IGNITE_CACHE_OPERATION	IgniteCacheOperation 列挙	実行するキャッシュ操作を動的に変更できます (プロデューサー)。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteCachePeekMode	IgniteConstants.IGNITE_CACHE_PEEK_MODE	CachePeekMode enum	SIZE 操作の実行時にキャッシュピークモードを動的に変更できます。
CamelIgniteCacheEventType	IgniteConstants.IGNITE_CACHE_EVENT_TYPE	int (EventType 定数)	このヘッダーは、連続クエリーコンシューマーを使用する場合に受信したイベントタイプを伝達します。
CamelIgniteCacheName	IgniteConstants.IGNITE_CACHE_NAME	String	このヘッダーには、連続クエリーイベントを受信したキャッシュ名 (consumer) が含まれます。プロデューサー操作が実行されるキャッシュを動的に変更することはできません。そのためには EIP を使用します (受信者リスト、動的ルーターなど)。
CamelIgniteCacheOldValue	IgniteConstants.IGNITE_CACHE_OLD_VALUE	Object	このヘッダーは、入力キャッシュイベント (consumer) で渡されるときに、古いキャッシュ値を保持します。

第148章 IGNITE COMPUTE コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Compute エンドポイントは camel-ignite エンドポイントの 1 つで、必要に応じてパラメーターと共に IgniteCallable、IgniteRunnable、IgniteClosure、またはそれらのコレクションを渡すことにより、クラスターで コンピュート操作 を実行できます。

このエンドポイントはプロデューサーのみをサポートします。

エンドポイント URI のホスト部分は、シンボリックエンドポイント ID であり、いかなる目的にも使用されません。

エンドポイントは、IN メッセージの本文で渡されたオブジェクトをコンピューティングジョブとして実行しようとします。実行タイプに応じて、さまざまなペイロードタイプが想定されます。

148.1. オプション

Ignite Compute コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (producer)	Ignite インスタンスを設定します。		Ignite
configurationResource (producer)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Compute エンドポイントは、URI 構文を使用して設定されます。

```
ignite-compute:endpointId
```

パスおよびクエリーパラメーターを使用します。

148.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
endpointId	**必須** エンドポイント ID (使用されません)。		String

148.1.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
clusterGroupExpression (producer)	IgniteCompute インスタンスのクラスターグループを返す式。		ClusterGroupExpression
computeName (producer)	リンク IgniteComputewithName (String) を介して設定されるコンピューティングジョブの名前。		文字列
executionType (producer)	**必須** 実行するコンピュート操作。可能な値: CALL、BROADCAST、APPLY、EXECUTE、RUN、AFFINITY_CALL、AFFINITY_RUN。コンポーネントは、操作に応じて異なるペイロードタイプを想定しています。		IgniteComputeExecution 型
propagateIncomingBodyIfNoReturnValue (producer)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
taskName (producer)	リンク IgniteComputeExecutionTypeEXECUTE 実行タイプを使用する場合にのみ適用されるタスク名。		文字列
timeoutMillis (producer)	トリガーされたジョブのタイムアウト間隔 (ミリ秒)。リンク IgniteComputewithTimeout(long) で設定されます。		Long
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

148.1.3. 予想されるペイロードの型

各操作には、指定された型が必要です。

操作	予想されるペイロード
CALL	IgniteCallable のコレクション、または単一の IgniteCallable。
BROADCAST	IgniteCallable, IgniteRunnable, IgniteClosure.
APPLY	IgniteClosure.

操作	予想されるペイロード
EXECUTE	taskName オプションが NULL でない場合は、ComputeTask、Class<? extends ComputeTask>、またはパラメータを表すオブジェクトを指定します。
実行	IgniteRunnables のコレクション、または単一の IgniteRunnable。
AFFINITY_CALL	IgniteCallable.
AFFINITY_RUN	IgniteRunnable.

148.1.4. 使用されるヘッダー

このエンドポイントは、次のヘッダーを使用します。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteComputeExecutionType	IgniteConstants.IGNITE_COMPUTE_EXECUTION_TYPE	IgniteComputeExecutionType enum	実行するコンピュート操作を動的に変更できます。
CamelIgniteComputeParameters	IgniteConstants.IGNITE_COMPUTE_PARAMS	任意のオブジェクトまたはオブジェクトのコレクション。	APPLY、BROADCAST、および EXECUTE 操作のパラメーター。
CamelIgniteComputeReducer	IgniteConstants.IGNITE_COMPUTE_REDUCER	IgniteReducer	APPLY および CALL 操作のリデューサー。
CamelIgniteComputeAffinityCacheName	IgniteConstants.IGNITE_COMPUTE_AFFINITY_CACHE_NAME	String	AFFINITY_CALL および AFFINITY_RUN 操作のアフィニティーキャッシュ名。
CamelIgniteComputeAffinityKey	IgniteConstants.IGNITE_COMPUTE_AFFINITY_KEY	Object	AFFINITY_CALL 操作と AFFINITY_RUN 操作のアフィニティーキー。

第149章 IGNITE EVENTS コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Events エンドポイントは camel-ignite エンドポイントの 1 つで、ローカルイベントリスナーを作成することで Ignite クラスターから イベントを受け取る ことができます。

このエンドポイントはコンシューマーのみをサポートします。このコンシューマーによって作成されたエクスチェンジは、受信した Event オブジェクトを IN メッセージのボディーに入れます。

149.1. オプション

Ignite Events コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (consumer)	Ignite インスタンスを設定します。		Ignite
configurationResource (consumer)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (consumer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Events エンドポイントは、URI 構文を使用して設定されます。

```
ignite-events:endpointId
```

パスおよびクエリーパラメーターを使用します。

149.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
endpointId	エンドポイント ID (未使用)。		String

149.1.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clusterGroupExpression (consumer)	クラスターグループ式。		ClusterGroupExpression
events (consumer)	ID が org.apache.ignite.events.EventType のさまざまな定数である Set として直接サブスクライブするイベント ID。	EventType.EVTS_ALL	Set<Integer>OrString
propagateIncomingBodyIfNoReturnValue (consumer)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (consumer)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第150章 IGNITE ID GENERATOR コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite ID Generator エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite アトミックシーケンスおよび ID ジェネレーター と対話できます。

このエンドポイントはプロデューサーのみをサポートします。

150.1. オプション

Ignite ID Generator コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (producer)	Ignite インスタンスを設定します。		Ignite
configurationResource (producer)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite ID ジェネレーターエンドポイントは、URI 構文を使用して設定されます。

```
ignite-idgen:name
```

パスおよびクエリーパラメーターを使用します。

150.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** シーケンス名。		String

150.1.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
batchSize (producer)	バッチサイズ。		Integer
initialValue (producer)	初期値。	0	Long
operation (producer)	Ignite ID ジェネレーターで呼び出す操作。IN メッセージの IgniteConstants.IGNITE_IDGEN_OPERATION ヘッダーに置き換えられました。設定可能な値: ADD_AND_GET、GET、GET_AND_ADD、GET_AND_INCREMENT、INCREMENT_AND_GET。		IgniteIdGenOperation
propagateIncomingBodyIfNoReturnValue (producer)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

第151章 IGNITE MESSAGING コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Messaging エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite トピック からメッセージを送信および使用できます。

このエンドポイントは、プロデューサー (メッセージを送信するため) とコンシューマー (メッセージを受信するため) をサポートします。

151.1. オプション

Ignite Messaging コンポーネントは、以下に示す 4 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (Common)	Ignite インスタンスを設定します。		Ignite
configurationResource (Common)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (Common)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Messaging エンドポイントは、URI 構文を使用して設定されます。

```
ignite-messaging:topic
```

パスおよびクエリーパラメーターを使用します。

151.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
topic	**必須** トピック名。		String

151.1.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
propagateIncomingBodyIfNoReturnValue (Common)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (Common)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
clusterGroupExpression (producer)	クラスターグループ式。		ClusterGroupExpression
sendMode (producer)	使用する送信モード。設定可能な値: UNORDERED、ORDERED。	UNORDERED	IgniteMessagingSend モード
timeout (producer)	順序付きメッセージを使用する場合の送信操作のタイムアウト。		Long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

151.1.3. 使用されるヘッダー

このエンドポイントは、次のヘッダーを使用します。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteMessagingTopic	IgniteConstants.IGNITE_MESSAGING_TOPIC	String	(プロデューサー) にメッセージを送信するトピックを動的に変更できます。また、メッセージが受信されたトピック (コンシューマー) も含まれます。
CamelIgniteMessagingUUID	IgniteConstants.IGNITE_MESSAGING_UUID	UUID	メッセージが到着すると、このヘッダーにはサブスクリプションの UUID が入力されます (コンシューマー)。

第152章 IGNITE QUEUES コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Queue エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Queue のデータ構造 を操作できるようにします。

このエンドポイントはプロデューサーのみをサポートします。

152.1. オプション

Ignite Queues コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (producer)	Ignite インスタンスを設定します。		Ignite
configurationResource (producer)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Queues エンドポイントは、URI 構文を使用して設定されます。

```
ignite-queue:name
```

パスおよびクエリーパラメーターを使用します。

152.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** キュー名。		String

152.1.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
capacity (producer)	キュー容量。デフォルト: 無制限。		int
configuration (producer)	コレクション設定。デフォルト: 空の設定。configuration.xyz=123 オプションを使用して内部プロパティーを簡単に設定することもできます。		CollectionConfiguration
operation (producer)	Ignite Queue で呼び出す操作。IN メッセージの IgniteConstants.IGNITE_QUEUE_OPERATION ヘッダーに置き換えられました。設定可能な値: CONTAINS、ADD、SIZE、REMOVE、ITERATOR、CLEAR、RETAIN_ALL、ARRAY、DRAIN、ELEMENT、PEEK、OFFER、POLL、TAKE、PUT。		IgniteQueueOperation
propagateIncomingBodyIfNoReturnValue (producer)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
timeoutMillis (producer)	キューのタイムアウト (ミリ秒)。デフォルト: タイムアウトなし。		Long
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

152.1.3. 使用されるヘッダー

このエンドポイントは、次のヘッダーを使用します。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteQueueOperation	IgniteConstants.IGNITE_QUEUE_OPERATION	IgniteQueueOperation enum	キュー操作を動的に変更できます。
CamelIgniteQueueMaxElements	IgniteConstants.IGNITE_QUEUE_MAX_ELEMENTS	Integer または int	DRAIN 操作を呼び出すときの、解放する項目の量。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteQueueTransferredCount	IgniteConstants.IGNITE_QUEUE_TRANSFERRED_COUNT	Integer または int	DRAIN 操作の結果として転送されたアイテムの量。
CamelIgniteQueueTimeoutMillis	IgniteConstants.IGNITE_QUEUE_TIMEOUT_MILLIS	Long または long	OFFER または POLL 操作を呼び出すときに使用するタイムアウトをミリ秒単位で動的に設定します。

第153章 IGNITE SETS コンポーネント

Camel バージョン 2.17 以降で利用可能

Ignite Sets エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Set データ構造 と対話できます。

このエンドポイントはプロデューサーのみをサポートします。

153.1. オプション

Ignite Sets コンポーネントは、以下に示す 4個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
ignite (producer)	Ignite インスタンスを設定します。		Ignite
configurationResource (producer)	設定をロードするリソースを設定します。URI、文字列 (URI)、または InputStream のいずれかです。		Object
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Sets エンドポイントは、URI 構文を使用して設定されます。

```
ignite-set:name
```

パスおよびクエリーパラメーターを使用します。

153.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** セット名。		String

153.1.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
configuration (producer)	コレクション設定。デフォルト: 空の設定。configuration.xyz=123 オプションを使用して内部プロパティーを簡単に設定することもできます。		CollectionConfiguration
operation (producer)	Ignite Set で呼び出す操作。IN メッセージの IgniteConstants.IGNITE_SETS_OPERATION ヘッダーに置き換えられました。可能な値: CONTAINS、ADD、SIZE、REMOVE、ITERATOR、CLEAR、RETAIN_ALL、ARRAY。実行するセット操作。		IgniteSetOperation
propagateIncomingBodyIfNoReturnValue (producer)	基になる Ignite 操作の戻り値の型が void の場合に、入力ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして扱うか、アイテムのコレクションとして扱うかを設定し、挿入/更新/計算などを行います。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

153.1.3. 使用されるヘッダー

このエンドポイントは、次のヘッダーを使用します。

ヘッダー名	定数	想定されるタイプ	説明
CamelIgniteSetsOperation	IgniteConstants.IGNITE_SETS_OPERATION	IgniteSetOperation 列挙	セット操作を動的に変更できます。

第154章 INFLUXDB コンポーネント

Camel バージョン 2.18 以降で利用可能

このコンポーネントを使用すると、InfluxDB https://influxdata.com/time-series-platform/influxdb/ 時系列データベースと対話できます。このコンポーネントのネイティブなボディタイプは Point (influxdb のネイティブクラス) ですが、メッセージボディーとして Map<String, Object> を受け取ることもでき、Point.class に変換されますが、Map には InfluxDbConstants.MEASUREMENT_NAME をキーとして持つ要素を含まなければならないことに注意してください。

もちろん、独自のコンバーターをポイントへのデータ型に登録するか、camel が提供する (非) マーシャリングツールを使用することもできます。

Camel 2.18 以降、Influxdb には Java 8 が必要です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-influxdb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

154.1. URI 形式

```
influxdb://beanName?[options]
```

154.2. URI オプション

プロデューサーは、ネイティブ Java ドライバーを使用して、レジストリーで設定された influxdb にメッセージを送信できます。

InfluxDB コンポーネントにはオプションがありません。

InfluxDB エンドポイントは、URI 構文を使用して設定されます。

```
influxdb:connectionBean
```

パスおよびクエリーパラメーターを使用します。

154.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionBean	クラス InfluxDB.class の influx データベースへの **必須** 接続		String

154.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
batch (producer)	この操作がバッチ操作かどうかを定義します	false	boolean
databaseName (producer)	時系列が保存されるデータベースの名前		String
operation (producer)	この操作が挿入かクエリーかを定義します	insert	String
query (producer)	操作クエリーの場合のクエリーを定義します		String
retentionPolicy (producer)	エンドポイントによって作成されたデータに対する保持ポリシーを定義する文字列	default	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

154.3. メッセージヘッダー

名前	デフォルト値	タイプ	コンテキスト	説明

154.4. 例

以下は、ポイントを db (URI から db 名を取得) 固有のキーに格納するルートの例です。

```
from("direct:start")
        .setHeader(InfluxDbConstants.DBNAME_HEADER, constant("myTimeSeriesDB"))
        .to("influxdb://connectionBean);
```

```
from("direct:start")
        .to("influxdb://connectionBean?databaseName=myTimeSeriesDB");
```

詳細については、これらのリソースを参照してください...

154.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- スタートガイド

第155章 IRC コンポーネント

Camel バージョン 1.1 以降で利用可能

irc コンポーネントは、IRC (Internet Relay Chat) トランスポートを実装します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-irc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

155.1. URI 形式

```
irc:nick@host[:port]/#room[?options]
irc:nick@host[:port]?channels=#channel1,#channel2,#channel3[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

155.2. オプション

IRC コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

IRC エンドポイントは、URI 構文を使用して設定されます。

```
irc:hostname:port
```

パスおよびクエリーパラメーターを使用します。

155.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
hostname	**必要** IRC チャットサーバーのホスト名		String
port	IRC チャットサーバーのポート番号。ポートが設定されていない場合は、6667、6668、または 6669 のいずれかのデフォルトポートが使用されます。		int

155.2.2. クエリーパラメーター (24 パラメーター)

名前	説明	デフォルト	タイプ
autoRejoin (common)	キックされたときに自動再参加するかどうか	true	boolean
namesOnJoin (common)	チャンネルに参加した後、NAMES コマンドをチャンネルに送信します。ヘッダー値 irc.num = '353' を持つ結果を処理するには、リンク onReply を true にする必要があります。	false	boolean
nickname (common)	チャットで使用されるニックネーム。		String
persistent (common)	**非推奨** 永続的なメッセージを使用します。	true	boolean
realname (common)	IRC ユーザーの実際の名前。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
colors (advanced)	サーバーがカラーコードをサポートするかどうか。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
onJoin (filter)	ユーザー参加イベントを処理します。	true	boolean
onKick (filter)	キックイベントを処理します。	true	boolean
onMode (filter)	モード変更イベントを処理します。	true	boolean
onNick (filter)	ニックネーム変更イベントを処理します。	true	boolean
onPart (filter)	ユーザーパーツイベントを処理します。	true	boolean
onPrivmsg (filter)	プライベートメッセージイベントを処理します。	true	boolean
onQuit (filter)	ユーザー終了イベントを処理します。	true	boolean
onReply (filter)	コマンドまたは情報メッセージに対する一般的な応答を処理するかどうか。	false	boolean
onTopic (filter)	トピック変更イベントを処理します。	true	boolean
nickPassword (security)	IRC サーバーのニックネームパスワード。		String
password (security)	IRC サーバーのパスワード。		String
sslContextParameters (security)	SSL を使用したセキュリティーの設定に使用されます。レジストリー内の org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、設定されている SSLContextParameters をコンポーネントレベルでオーバーライドします。この設定は trustManager オプションをオーバーライドすることに注意してください。		SSLContextParameters
trustManager (security)	SSL サーバーの証明書を検証するために使用されるトラストマネージャー。		SSLTrustManager

名前	説明	デフォルト	タイプ
username (security)	IRC サーバーのユーザー名。		文字列

155.3. SSL サポート

155.3.1. JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、IRC コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、IRC コンポーネントでユーティリティーを使用する方法を示しています。

エンドポイントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/truststore.jks");
ksp.setPassword("keystorePassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setTrustManagers(tmp);

Registry registry = ...
registry.bind("sslContextParameters", scp);

...

from(...)
    .to("ircs://camel-prd-user@server:6669/#camel-test?nickname=camel-
prd&password=password&sslContextParameters=#sslContextParameters");
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:trustManagers>
      <camel:keyStore
         resource="/users/home/server/truststore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="ircs://camel-prd-user@server:6669/#camel-test?nickname=camel-
prd&password=password&sslContextParameters=#sslContextParameters"/>...
```

155.3.2. 従来の基本設定オプションの使用

次のように、SSL 対応の IRC サーバーに接続することもできます。

```
ircs:host[:port]/#room?username=user&password=pass
```

デフォルトでは、IRC トランスポートは SSLDefaultTrustManager を使用します。独自のカスタムトラストマネージャーを提供する必要がある場合は、次のように **trustManager** パラメーターを使用します。

```
ircs:host[:port]/#room?
username=user&password=pass&trustManager=#referenceToMyTrustManagerBean
```

155.4. キーの使用

Camel 2.2 で利用可能

一部の irc ルームでは、そのチャネルに参加できるようにするためにキーを提供する必要があります。キーはただの秘密の言葉です。

たとえば、チャネル 1 と 3 だけがキーを使用する 3 つのチャネルに参加します。

```
irc:nick@irc.server.org?channels=#chan1,#chan2,#chan3&keys=chan1Key,,chan3key
```

155.5. チャンネルのユーザーのリストを取得する

コンポーネントがチャネルに参加した後、**namesOnJoin** オプションを使用して IRC- **NAMES** コマンドを呼び出すことができます。サーバーは **irc.num = 353** で応答します。したがって、結果を処理するには、プロパティー **onReply** を **true** にする必要があります。さらに、名前を取得するために、**onReply** エクスチェンジをフィルタリングする必要があります。

たとえば、チャネルのユーザー名を含むすべてのエクスチェンジを取得したい場合:

```
from("ircs:nick@myserver:1234/#mychannelname?namesOnJoin=true&onReply=true")
 .choice()
  .when(header("irc.messageType").isEqualToIgnoreCase("REPLY"))
   .filter(header("irc.num").isEqualTo("353"))
   .to("mock:result").stop();
```

155.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第156章 JACKSONXML DATAFORMAT

Camel バージョン 2.16 以降で利用可能

Jackson XML は、Jackson ライブラリー と XMLMapper 拡張機能 を使用して、XML ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを XML ペイロードにマーシャリングするデータ形式です。

情報: Jackson に精通している場合、この XML データ形式は対応する JSON と同じように動作するため、JSON シリアライゼーション/デシリアライゼーションのアノテーションが付けられたクラスで使用できます。

この拡張機能は、JAXB のコードファーストアプローチ も模倣しています。

このデータ形式は、高速で効率的な XML プロセッサーである Woodstox (特に Pretty Print などの機能) に依存しています。

```
from("activemq:My.Queue").
  unmarshal().jacksonxml().
  to("mqseries:Another.Queue");
```

156.1. JACKSONXML オプション

JacksonXML データ形式は、以下に示す 15 のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
xmlMapper		**String**	指定された ID を持つ既存の XmlMapper を検索して使用します。
prettyPrint	**false**	**Boolean**	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
unmarshalTypeName		**String**	アンアームシャリング時に使用する Java 型のクラス名
jsonView		**Class <?>**	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		**String**	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。
allowJmsType	**false**	**Boolean**	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。

名前	デフォルト	Java タイプ	説明
collectionTypeName		String	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。
useList	false	Boolean	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	false	Boolean	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		String	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		String	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		String	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		String	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
allowUnmarshallType	false	Boolean	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

156.1.1. Spring DSL での Jackson XML の使用

Spring DSL でデータ形式を使用する場合には、最初にデータ形式を宣言する必要があります。これは **DataFormats** XML タグで行われます。

```
    <dataFormats>
      <!-- here we define a Xml data format with the id jack and that it should use the TestPojo as
the class type when
           doing unmarshal. The unmarshalTypeName is optional, if not provided Camel will use a
Map as the type -->
      <jacksonxml id="jack"
unmarshalTypeName="org.apache.camel.component.jacksonxml.TestPojo"/>
    </dataFormats>
```

そして、ルートでこの ID を参照できます。

```
    <route>
      <from uri="direct:back"/>
      <unmarshal ref="jack"/>
      <to uri="mock:reverse"/>
    </route>
```

156.2. POJO フィールドをマーシャリングから除外する

POJO を XML にマーシャリングするときに、XML 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビュー を使用してこれを実現できます。まず、1 つ以上のマーカークラスを作成します。

@JsonView アノテーションでマーカークラスを使用して、特定のフィールドの追加や除外を行います。アノテーションは getter でも機能します。

最後に、Camel **JacksonXMLDataFormat** を使用して、上記の POJO を XML にマーシャリングします。

結果の XML に weight フィールドがないことに注意してください。

```
<pojo age="30" weight="70"/>
```

156.3. `JACKSONXML`DATAFORMAT で JSONVIEW 属性を使用する INCLUDE/EXCLUDE フィールド

この属性を使用する例として、代わりに次のことができます。

```
JacksonXMLDataFormat ageViewFormat = new JacksonXMLDataFormat(TestPojoView.class,
Views.Age.class);
from("direct:inPojoAgeView").
  marshal(ageViewFormat);
```

Java DSL 内で JSON ビュー を次のように直接指定します。

```
from("direct:inPojoAgeView").
  marshal().jacksonxml(TestPojoView.class, Views.Age.class);
```

XML DSL でも同じです。

```
<from uri="direct:inPojoAgeView"/>
  <marshal>
    <jacksonxml unmarshalTypeName="org.apache.camel.component.jacksonxml.TestPojoView"
jsonView="org.apache.camel.component.jacksonxml.Views$Age"/>
  </marshal>
```

156.4. シリアル化の INCLUDE オプションの設定

pojo を XML にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、pojo にアノテーションを設定する必要があります。

```
@JsonInclude(Include.NON_NULL)
public class MyPojo {
   ...
}
```

ただし、これには pojo ソースコードにそのアノテーションを含める必要があります。以下に示すように、Camel JacksonXMLDataFormat を設定して include オプションを設定することもできます。

```
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.setInclude("NON_NULL");
```

または、XML DSL から、これを次のように設定します。

```
    <dataFormats>
      <jacksonxml id="jacksonxml" include="NOT_NULL"/>
    </dataFormats>
```

156.5. 動的クラス名を使用した XML から POJO へのアンマーシャリング

jackson を使用して XML を POJO に非整列化する場合、非整列化先のクラス名を示すヘッダーをメッセージに指定できるようになりました。
そのヘッダーがメッセージに存在する場合、ヘッダーにはキー **CamelJacksonUnmarshalType** があり、Jackson はそれを POJO クラスの FQN として使用して、XML ペイロードを非整列化します。

JMS エンドユーザー向けには、JMS 仕様の JMSType ヘッダーもあり、これもそれを示しています。JMSType のサポートを有効にするには、次のように jackson データ形式で有効にする必要があります。

```
JacksonDataFormat format = new JacksonDataFormat();
format.setAllowJmsType(true);
```

または、XML DSL から、これを次のように設定します。

```
    <dataFormats>
      <jacksonxml id="jacksonxml" allowJmsType="true"/>
    </dataFormats>
```

156.6. XML から LIST<MAP> または LIST<POJO> へのアンマーシャリング

Jackson を使用して XML を map/pojo のリストに非整列化する場合には、**useList="true"** を設定するか、**org.apache.camel.component.jacksonxml.ListJacksonXMLDataFormat** を使用してこれを指定できるようになりました。たとえば、Java を使用すると、次のように実行できます。

```
JacksonXMLDataFormat format = new ListJacksonXMLDataFormat();
// or
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.useList();
// and you can specify the pojo class type also
format.setUnmarshalType(MyPojo.class);
```

また、XML DSL を使用する場合は、以下に示すように **useList** 属性でリストを使用するように設定します。

```
<dataFormats>
  <jacksonxml id="jack" useList="true"/>
</dataFormats>
```

また、pojo タイプも指定できます。

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalTypeName="com.foo.MyPojo"/>
</dataFormats>
```

156.7. カスタム JACKSON モジュールの使用

以下に示すように、moduleClassNames オプションを使用してそれらのクラス名を指定して、カスタム Jackson モジュールを使用できます。

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalTypeName="com.foo.MyPojo"
moduleClassNames="com.foo.MyModule,com.foo.MyOtherModule"/>
</dataFormats>
```

moduleClassNames を使用する場合には、デフォルトのコンストラクターを使って作成され、そのまま使用されるので、カスタム jackson モジュールは設定されません。カスタムモジュールにカスタム設定が必要な場合は、モジュールのインスタンスを作成して設定し、次に示すように modulesRefs を使用してモジュールを参照できます。

```
<bean id="myJacksonModule" class="com.foo.MyModule">
  ... // configure the module as you want
</bean>

<dataFormats>
  <jacksonxml id="jacksonxml" useList="true" unmarshalTypeName="com.foo.MyPojo"
moduleRefs="myJacksonModule"/>
</dataFormats>
```

moduleRefs="myJacksonModule,myOtherModule" のように、コンマで区切って複数のモジュールを指定できます。

156.8. JACKSON を使用した機能の有効化または無効化

Jackson には、有効または無効にできる機能が多くあり、その ObjectMapper が使用します。たとえば、マーシャリング時に不明なプロパティーでの失敗を無効にするには、disableFeatures を使用してこれを設定できます。

```
<dataFormats>
    <jacksonxml id="jacksonxml" unmarshalTypeName="com.foo.MyPojo"
disableFeatures="FAIL_ON_UNKNOWN_PROPERTIES"/>
</dataFormats>
```

コンマを使用して値を区切って、複数の機能を無効にすることができます。機能の値は、次の列挙型クラスの Jackson の列挙型の名前である必要があります

- com.fasterxml.jackson.databind.SerializationFeature
- com.fasterxml.jackson.databind.DeserializationFeature
- com.fasterxml.jackson.databind.MapperFeature

機能を有効にするには、代わりに enableFeatures オプションを使用します。

Java コードからは、camel-jackson モジュールのタイプセーフなメソッドを使用できます。

```
JacksonDataFormat df = new JacksonDataFormat(MyPojo.class);
df.disableFeature(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
df.disableFeature(DeserializationFeature.FAIL_ON_NULL_FOR_PRIMITIVES);
```

156.9. JACKSON を使用してマップを POJO に変換する

Jackson **ObjectMapper** を使用して、マップを POJO オブジェクトに変換できます。Jackson コンポーネントには、**java.util.Map** インスタンスを String、Primitive、および Number 以外のオブジェクトに変換のに使用できるデータコンバーターが付属しています。

```
Map<String, Object> invoiceData = new HashMap<String, Object>();
invoiceData.put("netValue", 500);
producerTemplate.sendBody("direct:mapToInvoice", invoiceData);
...
// Later in the processor
Invoice invoice = exchange.getIn().getBody(Invoice.class);
```

Camel レジストリーで利用可能な **ObjectMapper** インスタンスが 1 つある場合には、コンバーターはそれを使用して変換します。それ以外の場合は、デフォルトのマッパーが使用されます。

156.10. フォーマット済み XML マーシャリング (PRETTY-PRINTING)

prettyPrint オプションを使用すると、マーシャリング中に適切にフォーマットされた XML を出力できます。

```
<dataFormats>
    <jacksonxml id="jack" prettyPrint="true"/>
</dataFormats>
```

そして Java DSL の場合:

```
from("direct:inPretty").marshal().jacksonxml(true);
```

unmarshalType、**jsonView** などの他の設定と組み合わせて **prettyPrint** オプションをサポートする、オーバーロードされた **jacksonxml ()** DSL メソッドが 5 種あることに注意してください。

156.11. 依存関係

camel ルートで Jackson XML を使用するには、このデータ形式を実装する **camel-jacksonxml** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jacksonxml</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第157章 JASYPT コンポーネント

Camel 2.5 で利用可能

Jasypt は、暗号化と復号化を簡単にする単純化された暗号化ライブラリーです。Camel は Jasypt と統合して、プロパティー ファイル内の機密情報を暗号化できるようにします。クラスパスに **camel-jasypt を** ドロップすると、これらの暗号化された値は Camel によってオンザフライで自動的に復号化されます。これにより、ユーザー名やパスワードなどの機密情報を人間の目で簡単に見つけることができなくなります。

Maven を使用している場合は、このコンポーネントの **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jasypt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Apache Karaf コンテナーを使用している場合は、このコンポーネントの **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.karaf.jaas</groupId>
  <artifactId>org.apache.karaf.jaas.jasypt</artifactId>
  <version>x.x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

157.1. ツール

Jasypt コンポーネントは、値を暗号化または復号化するための小さなコマンドラインツールを提供します。

コンソールは、構文とそれが提供するオプションを出力します。

```
Apache Camel Jasypt takes the following options

  -h or -help = Displays the help screen
  -c or -command <command> = Command either encrypt or decrypt
  -p or -password <password> = Password to use
  -i or -input <input> = Text to encrypt or decrypt
  -a or -algorithm <algorithm> = Optional algorithm to use
```

たとえば、次のパラメーターを使用して実行する値 **tiger** を暗号化します。apache camel kit では、cd で lib フォルダーに移動し、次の Java コマンドを実行します。**<CAMEL_HOME>** は、Camel ディストリビューションをダウンロードして展開した場所です。

```
$ cd <CAMEL_HOME>/lib
$ java -jar camel-jasypt-2.5.0.jar -c encrypt -p secret -i tiger
```

次の結果を出力するもの

-

```
Encrypted text: qaEEacuW7BUti8LcMgyjKw==
```

これは、暗号化された表現 **qaEEacuW7BUti8LcMgyjKw==** を 復号化して、**secret** のマスターパスワードを知っていれば **tiger** に戻すことができることを意味します。
ツールを再度実行すると、暗号化された値は別の結果を返します。ただし、値を復号化すると、常に正しい元の値が返されます。

したがって、次のパラメーターを使用してツールを実行することでテストできます。

```
$ cd <CAMEL_HOME>/lib
$ java -jar camel-jasypt-2.5.0.jar -c decrypt -p secret -i qaEEacuW7BUti8LcMgyjKw==
```

次の結果が出力されます。

```
Decrypted text: tiger
```

次に、これらの暗号化された値を プロパティー ファイルで使用することを考えます。パスワード値がどのように暗号化され、値が **ENC(value here)** を囲むトークンを持っているかに注目してください。

ヒント

jasypt ツールの実行中に **java.lang.NoClassDefFoundError: org/jasypt/encryption/pbe/StandardPBEStringEncryptor** に遭遇した場合は、クラスパスに jasypt7.2.jar を含める必要があることを意味します。jar をクラスパスに追加する例は、**java -jar ...** として実行する場合に、jasypt7.2.jar を $JAVA_HOME\jre\lib\ext にコピーすることです。後者は **-cp** を使用してクラスパスに jasypt7.4.jarを追加している可能性があります。その場合、実行するメインクラスを (**java -cp jasypt-1.9.2.jar:camel-jasypt-2.18.2.jar org.apache.camel.component.jasypt.Main -c encrypt -p secret -i tiger**) のように指定する必要があります。

157.2. URI オプション

以下のオプションは、Jasypt コンポーネント専用です。

名前	デフォルト値	タイプ	説明
password	**null**	**String**	復号化に使用するマスターパスワードを指定します。このオプションは必須です。詳細は、こちらを参照してください。
algorithm	**null**	**String**	使用するオプションのアルゴリズムの名前。

157.3. マスターパスワードの保護

Jasypt が使用するマスターパスワードを提供して、値を解読できるようにする必要があります。ただし、このマスターパスワードを公開することは、理想的な解決策ではない可能性があります。したがって、たとえば、JVM システムプロパティーまたは OS 環境設定として提供できます。そうすることにした場合、**password** オプションはこれを指示する接頭辞をサポートします。**sysenv:** 指定されたキーで OS システム環境を検索することを意味します。**sys:** JVM システムプロパティーを検索することを意味します。

たとえば、アプリケーションを起動する前にパスワードを提供できます

```
$ export CAMEL_ENCRYPTION_PASSWORD=secret
```

次に、起動スクリプトを実行するなどして、アプリケーションを起動します。

アプリケーションが起動して実行されたら、環境の設定を解除できます。

```
$ unset CAMEL_ENCRYPTION_PASSWORD
```

password オプションは、**password=sysenv:CAMEL_ENCRYPTION_PASSWORD** のように定義するだけです。

157.4. JAVA DSL の例

Java DSL では、Jasypt を **JasyptPropertiesParser** インスタンスとして設定し、以下に示すように Properties コンポーネントで設定する必要があります。

プロパティーファイル **myproperties.properties** には、次に示すように、暗号化された値が含まれます。パスワード値がどのように暗号化され、値が **ENC(value here)** を囲むトークンを持っているかに注目してください。

157.5. SPRING XML の例

Spring XML では、以下に示す **JasyptPropertiesParser** を設定する必要があります。次に、Camel Properties コンポーネントはプロパティーパーサーとして **jasypt** を使用するように指示されます。これは、Jasypt がプロパティーで検索された値を復号化する機会があることを意味します。

```
<!-- define the jasypt properties parser with the given password to be used -->
<bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <property name="password" value="secret"/>
</bean>

<!-- define the camel properties component -->
<bean id="properties" class="org.apache.camel.component.properties.PropertiesComponent">
    <!-- the properties file is in the classpath -->
    <property name="location"
value="classpath:org/apache/camel/component/jasypt/myproperties.properties"/>
    <!-- and let it leverage the jasypt parser -->
    <property name="propertiesParser" ref="jasypt"/>
</bean>
```

Properties コンポーネントは、以下に示す **<camelContext>** タグ内にインライン化することもできます。**propertiesParserRef** 属性を使用して Jasypt を参照する方法に注目してください。

```
<!-- define the jasypt properties parser with the given password to be used -->
<bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <!-- password is mandatory, you can prefix it with sysenv: or sys: to indicate it should use
         an OS environment or JVM system property value, so you dont have the master password
defined here -->
    <property name="password" value="secret"/>
</bean>

<camelContext xmlns="http://camel.apache.org/schema/spring">
```

```
    <!-- define the camel properties placeholder, and let it leverage jasypt -->
    <propertyPlaceholder id="properties"
                         location="classpath:org/apache/camel/component/jasypt/myproperties.properties"
                         propertiesParserRef="jasypt"/>
    <route>
      <from uri="direct:start"/>
      <to uri="{{cool.result}}"/>
    </route>
</camelContext>
```

157.6. ブループリント XML を使用した例

ブループリント XML では、以下に示す **JasyptPropertiesParser** を設定する必要があります。次に、Camel Properties コンポーネントはプロパティーパーサーとして **jasypt** を使用するように指示されます。これは、Jasypt がプロパティーで検索された値を復号化する機会があることを意味します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xsi:schemaLocation="
           http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

  <cm:property-placeholder id="myblue" persistent-id="mypersistent">
    <!-- list some properties for this test -->
    <cm:default-properties>
      <cm:property name="cool.result" value="mock:{{cool.password}}"/>
      <cm:property name="cool.password" value="ENC(bsW9uV37gQ0QHFu7KO03Ww==)"/>
    </cm:default-properties>
  </cm:property-placeholder>

  <!-- define the jasypt properties parser with the given password to be used -->
  <bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <property name="password" value="secret"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <!-- define the camel properties placeholder, and let it leverage jasypt -->
    <propertyPlaceholder id="properties"
                         location="blueprint:myblue"
                         propertiesParserRef="jasypt"/>
      <route>
        <from uri="direct:start"/>
        <to uri="{{cool.result}}"/>
      </route>
  </camelContext>

</blueprint>
```

Properties コンポーネントは、以下に示す **<camelContext>** タグ内にインライン化することもできます。**propertiesParserRef** 属性を使用して Jasypt を参照する方法に注目してください。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
```

```
        xsi:schemaLocation="
        http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

  <!-- define the jasypt properties parser with the given password to be used -->
  <bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <property name="password" value="secret"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <!-- define the camel properties placeholder, and let it leverage jasypt -->
    <propertyPlaceholder id="properties"
                         location="classpath:org/apache/camel/component/jasypt/myproperties.properties"
                         propertiesParserRef="jasypt"/>
    <route>
      <from uri="direct:start"/>
      <to uri="{{cool.result}}"/>
    </route>
  </camelContext>

</blueprint>
```

157.7. 関連項目

- セキュリティー
- Properties
- ActiveMQ の暗号化されたパスワード - ActiveMQ には、この **camel-jasypt** コンポーネントと同様の機能があります。

第158章 JAXB DATAFORMAT

Camel バージョン 1.0 以降で利用可能

JAXB は、Java 6 に含まれている JAXB2 XML マーシャリング標準を使用して、XML ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを XML ペイロードにマーシャリングするデータ形式です。

158.1. オプション

JAXB データ形式は、以下にリストされているオプションを 18 個サポートしています。

名前	デフォルト	Java タイプ	説明
contextPath		String	JAXB クラスが配置されているパッケージ名。
schema		String	既存のスキーマに対して検証します。接頭辞 classpath:、file:、または http: を使用して、リソースの解決方法を指定できます。「,」文字を使用して、複数のスキーマファイルを区切ることができます。
schemaSeverityLevel	0	Integer	スキーマに対して検証するときに使用するスキーマの重大度レベルを設定します。このレベルは、重大度が最小のエラーを判断し、JAXB をトリガーして解析の続行を停止します。デフォルト値の 0 (警告) は、エラー (警告、エラー、または致命的なエラー) によって JAXB が停止することを意味します。0=警告、1=エラー、2=致命的なエラーという 3 つのレベルがあります。
prettyPrint	false	Boolean	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
objectFactory	false	Boolean	マーシャリング中に ObjectFactory クラスを使用して POJO クラスを作成できるようにするかどうか。これは、JAXB でアノテーションが付けられておらず、jaxb.index 記述子ファイルを提供していない POJO クラスにのみ適用されます。
ignoreJAXBElement	false	Boolean	JAXBElement 要素を無視するかどうか。非常に特殊なユースケースでのみ false に設定する必要があります。
mustBeJAXBElement	false	Boolean	マーシャリングが JAXB アノテーション付きの Java オブジェクトでなければならないかどうか。そうでない場合は失敗します。このオプションを false に設定すると、データがすでに XML 形式になっている場合など、設定が緩和されます。
filterNonXmlChars	false	Boolean	xml 以外の文字を無視し、空白に置き換えます。
encoding		String	特定のエンコーディングを無効にして使用します

名前	デフォルト	Java タイプ	説明
fragment	**false**	**Boolean**	XML フラグメントツリーのマーシャリングをオンにします。デフォルトでは、JAXB は指定されたクラスの XmlRootElement アノテーションを探して、XML ツリー全体を操作します。これは便利ですが、常に有用であるわけではありません。たとえば、生成されたコードに XmlRootElement アノテーションがない場合や、ツリーの一部だけを非整列化する必要がある場合などが挙げられます。その場合、部分的なアンマーシャリングを使用できます。この動作を有効にするには、プロパティー partClass を設定する必要があります。Camel は、このクラスを JAXB のアンマーシャラーに渡します。
partClass		**String**	フラグメント解析に使用されるクラスの名前。詳細については、フラグメントオプションを参照してください。
partNamespace		**String**	フラグメントの解析に使用する XML 名前空間。詳細については、フラグメントオプションを参照してください。
namespacePrefixRef		**String**	JAXB または SOAP を使用してマーシャリングする場合、JAXB 実装は、ns2、ns3、ns4 などの名前空間接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では目的のマッピングを含むマップを参照できます。
xmlStreamWriterWrapper		**String**	カスタム xml ストリームライターを使用します。
schemaLocation		**String**	スキーマのロケーションを定義します
noNamespaceSchemaLocation		**String**	名前空間のないスキーマのロケーションを定義します
jaxbProviderProperties		**String**	カスタム java.util.Map を参照して、JAXB マーシャラーで使用されるカスタム JAXB プロバイダープロパティーを含むレジストリーを検索します。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

158.2. JAVA DSL を使用

たとえば、次の例では、多数の Java パッケージ名で設定された **jaxb** の名前付き DataFormat を使用して JAXBContext を初期化します。

```
DataFormat jaxb = new JaxbDataFormat("com.acme.model");
```

```
from("activemq:My.Queue").
  unmarshal(jaxb).
  to("mqseries:Another.Queue");
```

必要に応じて、Spring XML ファイルなどを介してレジストリーで定義できるデータ形式への名前付き参照を使用できます。以下の例を示します。

```
from("activemq:My.Queue").
  unmarshal("myJaxbDataType").
  to("mqseries:Another.Queue");
```

158.3. SPRING XML の使用

次の例は、jaxb データ型を設定する Spring を使用して JAXB を使用して非整列化する方法を示しています。

この例は、データ型を 1 回だけ設定し、それを複数のルートで再利用する方法を示しています。

複数のコンテキストパス

このデータ形式は、複数のコンテキストパスで使用できます。**com.mycompany:com.mycompany2** のように、セパレータとして **:** を使用してコンテキストパスを指定できます。これは JAXB 実装によって処理され、RI とは異なるベンダーを使用すると変更される可能性があることに注意してください。

158.4. 部分マーシャリング/アンマーシャリング

この機能は Camel 2.2.0 の新機能です。
JAXB 2 は、XML ツリーフラグメントのマーシャリングとアンマーシャリングをサポートしています。デフォルトでは、JAXB は指定されたクラスの **@XmlRootElement** アノテーションを探して、XML ツリー全体を操作します。これは便利ですが、常に有用であるわけではありません。たとえば、生成されたコードに @XmlRootElement アノテーションがない場合や、ツリーの一部だけを非整列化する必要がある場合などが挙げられます。
その場合、部分的なアンマーシャリングを使用できます。この動作を有効にするには、プロパティー **partClass** を設定する必要があります。Camel は、このクラスを JAXB のアンマーシャラーに渡します。**JaxbConstants.JAXB_PART_CLASS** がヘッダーの 1 つとして設定されている場合 (DataFormat に partClass プロパティーが設定されていても)、DataFormat のプロパティーが優先され、ヘッダーに設定されているものが使用されます。

マーシャリング用に、宛先名前空間の QName を含む **partNamespace** 属性を追加する必要があります。Spring DSL の例は、上記で確認できます。**JaxbConstants.JAXB_PART_NAMESPACE** がヘッダーの 1 つとして設定されている場合 (DataFormat で partNamespace プロパティーが設定されていても)、DataFormat のプロパティーが優先されて、ヘッダーに設定されているものが使用されます。**JaxbConstants.JAXB_PART_NAMESPACE** を使用して **partNamespace** を設定する際、それに対する値 \{namespaceUri}localPart を指定する必要があることに注意してください。

```
   ...
   .setHeader(JaxbConstants.JAXB_PART_NAMESPACE, simple("
{http://www.camel.apache.org/jaxb/example/address/1}address"));
   ...
```

158.5. フラグメント

この機能は Camel 2.8.0 の新機能です。
JaxbDataFormat には、JAXB Marshaller で **Marshaller.JAXB_FRAGMENT** エンコーディングプロパティーを設定できる新しいプロパティーフラグメントがあります。JAXB Marshaller に XML 宣言を生成さない場合は、このオプションを true に設定できます。このプロパティーのデフォルト値は false です。

158.6. XML 以外の文字の無視

この機能は Camel 2.2.0 の新機能です。
JaxbDataFormat では、NonXML Character を無視できます。filterNonXmlChars プロパティーを true に設定するだけで、JaxbDataFormat はメッセージのマーシャリングまたはアンマーシャリング時に NonXML 文字を " " に置き換えます。Exchange プロパティー **Exchange.FILTER_NON_XML_CHARS** を設定して行うこともできます。

	JDK 1.5	JDK 1.6 以降
フィルタリングの使用	StAX API と実装	いいえ
フィルタリングの使用なし	StAX API のみ	いいえ

この機能は、Woodstox 3.2.9 および Sun JDK 1.6 StAX 実装でテストされています。

New for Camel 2.12.1
JaxbDataFormat では、ストリームを XML にマーシャリングするために使用される XMLStreamWriter をカスタマイズできるようになりました。この設定を使用すると、独自のストリームライターを追加して、xml 以外の文字を完全に削除、エスケープ、または置換できます。

```
JaxbDataFormat customWriterFormat = new JaxbDataFormat("org.apache.camel.foo.bar");
customWriterFormat.setXmlStreamWriterWrapper(new TestXmlStreamWriter());
```

次の例は、Spring DSL を使用し、Camel の NonXML フィルタリングも有効にすることを示しています。

```
<bean id="testXmlStreamWriterWrapper" class="org.apache.camel.jaxb.TestXmlStreamWriter"/>
<jaxb filterNonXmlChars="true"  contextPath="org.apache.camel.foo.bar"
xmlStreamWriterWrapper="#testXmlStreamWriterWrapper" />
```

158.7. OBJECTFACTORY の操作

XJC を使用してスキーマから Java クラスを作成すると、JAXB コンテキストの ObjectFactory が取得されます。ObjectFactory は JAXBElement を使用してスキーマと要素インスタンス値の参照を保持するため、jaxbDataformat はデフォルトで JAXBElement を無視し、アンマーシャリングされたメッセージボディーから JAXBElement オブジェクトの代わりに要素インスタンス値を取得します。
アンマーシャリングされたメッセージボディーから JAXBElement オブジェクトを取得する場合は、JaxbDataFormat オブジェクトの ignoreJAXBElement プロパティーを false に設定する必要があります。

158.8. エンコーディングの設定

マーシャリング時に使用する **エンコード** オプションを設定できます。JAXB Marshaller の **Marshaller.JAXB_ENCODING** エンコーディングプロパティーです。
JAXB データ形式を宣言するときに使用するエンコーディングを設定できます。Exchange プロパティー **Exchange.CHARSET_NAME** でエンコーディングを指定することもできます。このプロパティーは、JAXB データ形式に設定されたエンコーディングを無効にします。

この Spring DSL で、エンコードとして **iso-8859-1** を使用するように定義しました。

158.9. 名前空間接頭辞のマッピングの制御

Camel 2.11 から利用可能

JAXB または SOAP を使用してマーシャリングする場合に、JAXB 実装は、ns2、ns3、ns4 などの名前空間の接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では目的のマッピングを含むマップを参照できます。

マッピング機能は、サポートされているかどうかにかかわらず、JAXB の実装に依存するため、クラスパスに JAXB-RI 2.1 以降 (SUN から) が必要であることに注意してください。

たとえば、Spring XML では、マッピングを使用して Map を定義できます。以下のマッピングファイルでは、SOAP を接頭辞として使用するように SOAP をマッピングします。カスタムの名前空間 http://www.mycompany.com/foo/2 は接頭辞を使用していません。

```
<util:map id="myMap">
  <entry key="http://www.w3.org/2003/05/soap-envelope" value="soap"/>
  <!-- we dont want any prefix for our namespace -->
  <entry key="http://www.mycompany.com/foo/2" value=""/>
</util:map>
```

これを JAXB または SOAP で使用するには、以下に示すように **namespacePrefixRef** 属性を使用して、このマップを参照します。次に、Camel はレジストリーで、上記で定義した ID "myMap" を持つ **java.util.Map** を検索します。

```
<marshal>
  <soapjaxb version="1.2" contextPath="com.mycompany.foo" namespacePrefixRef="myMap"/>
</marshal>
```

158.10. スキーマ検証

Camel 2.11 から利用可能

JAXB データ形式は、XML との間でマーシャリングおよびアンマーシャリングによる検証をサポートしています。接頭辞 **classpath:**、**file:**、または **http:** を使用して、リソースの解決方法を指定できます。',' 文字を使用して、複数のスキーマファイルを区切ることができます。

既知の問題

Camel 2.11.0 および 2.11.1 には、複数のエクスチェンジを並行して検証することによる既知の問題があります。CAMEL-6630 を参照してください。これは Camel 2.11.2/2.12.0 で修正されています。

Java DSL を使用すると、次のように設定できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchema("classpath:person.xsd,classpath:address.xsd");
```

XML DSL を使用して同じことができます。

```
<marshal>
  <jaxb id="jaxb" schema="classpath:person.xsd,classpath:address.xsd"/>
</marshal>
```

JDK に同梱されている **SchemaFactory** はスレッドセーフではないため、Camel はその場で下層の **SchemaFactory** インスタンスを作成してプールします。
ただし、スレッドセーフな **SchemaFactory** 実装がある場合は、JAXB データ形式を設定して、この実装を使用できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setSchemaFactory(thradSafeSchemaFactory);
```

158.11. スキーマのロケーション

Camel 2.14 から利用可能

JAXB データ形式は、XML のマーシャリング時に SchemaLocation の指定をサポートしています。

Java DSL を使用すると、次のように設定できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchemaLocation("schema/person.xsd");
```

XML DSL を使用して同じことができます。

```
<marshal>
  <jaxb id="jaxb" schemaLocation="schema/person.xsd"/>
</marshal>
```

158.12. すでに XML になっているデータのマーシャリング

Camel 2.14.1 以降で利用可能

JAXB マーシャラーは、メッセージボディーが JAXB 互換である必要があります (例: JAXBElement、JAXB アノテーションを持つ、または JAXBElement を拡張する Java インスタンス)。メッセージボディーがすでに XML になっている場合があります (例: String 型)。このチェックを緩和するために、false に設定できる新しいオプション **mustBeJAXBElement** があるため、JAXB マーシャラーは JAXBElement のみをマーシャリングしようとします (javax.xml.bind.JAXBIntrospector#isElement は true を返します)。このような状況では、マーシャラーはフォールバックして、メッセージボディーをそのままマーシャリングします。

158.13. 依存関係

camel ルートで JAXB を使用するには、このデータ形式を実装する **camel-jaxb** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jaxb</artifactId>
  <version>x.x.x</version>
</dependency>
```

第159章 JBOSS DATA GRID コンポーネント

159.1. RED HAT JBOSS DATA GRID コンポーネントと APACHE CAMEL

Red Hat JBoss Data Grid および Red Hat JBoss Fuse で Camel を使用すると、接続を追加するさまざまなトランスポートと API が提供されるため、大規模なエンタープライズアプリケーションでの統合が簡素化されます。

JBoss Data Grid は、JBoss Fuse の Camel ルートでのキャッシングをサポートし、部分的に Ehcache を置き換えます。JBoss Data Grid は、埋め込みキャッシュ (ローカルまたはクラスター化) または Camel ルートのリモートキャッシュとしてサポートされます。

Red Hat JBoss Data Grid を Apache Camel で実行する方法は、Red Hat JBoss Data Grid の紹介 を参照してください。

第160章 JCACHE COMPONENT

Camel バージョン 2.17 以降で利用可能

jcache コンポーネントを使用すると、JSR107/JCache をキャッシュ実装として使用してキャッシュ操作を実行できます。

160.1. URI 形式

```
jcache:cacheName[?options]
```

160.2. URI オプション

JCache エンドポイントは、URI 構文を使用して設定されます。

```
jcache:cacheName
```

パスおよびクエリーパラメーターを使用します。

160.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheName	**必須**: キャッシュの名前		String

160.2.2. クエリーパラメーター(22 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheConfiguration (common)	キャッシュの設定		設定
cacheConfigurationProperties (common)	CacheManager を作成するための javax.cache.spi.CachingProvider のプロパティー		Properties
cachingProvider (common)	javax.cache.spi.CachingProvider の完全修飾クラス名		String
configurationUri (common)	CacheManager の実装固有の URI		String
managementEnabled (common)	管理収集が有効かどうか	false	boolean

名前	説明	デフォルト	タイプ
readThrough (common)	リードスルーキャッシュを使用する必要がある場合	false	boolean
statisticsEnabled (common)	統計情報収集が有効かどうか	false	boolean
storeByValue (common)	キャッシュが値によるストアまたは参照によるストアのセマンティクスを使用する必要がある場合	true	boolean
writeThrough (common)	ライトスルーキャッシュを使用する必要がある場合	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
filteredEvents (consumer)	コンシューマーがフィルタリングする必要があるイベント。filteredEvents オプションを使用する場合、eventFilters は無視されます		List
oldValueRequired (consumer)	イベントに古い値が必要な場合	false	boolean
synchronous (consumer)	イベントリスナーがイベントの原因となっているスレッドをブロックする必要がある場合	false	boolean
eventFilters (consumer)	CacheEntryEventFilter。eventFilters オプションを使用する場合、filteredEvents は無視されます。		List
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
action (producer)	デフォルトでキャッシュ操作を使用して設定する場合。メッセージヘッダー内の操作の場合、ヘッダーからの操作が優先されます。		String

名前	説明	デフォルト	タイプ
cacheLoaderFactory (advanced)	CacheLoader ファクトリー		CacheLoader>
cacheWriterFactory (advanced)	CacheWriter ファクトリー		CacheWriter>
createCacheIfNotExists (advanced)	キャッシュが存在する場合、または事前設定できない場合にキャッシュを作成する必要があるかどうかを設定します。	true	boolean
expiryPolicyFactory (advanced)	ExpiryPolicy ファクトリー		ExpiryPolicy>
lookupProviders (advanced)	camel-cache が OSGi のようなランタイムで jcache api の実装を見つけようとするかどうかを設定します。	false	boolean

JCache コンポーネントは、以下に示す 5 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
cachingProvider (common)	javax.cache.spi.CachingProvider の完全修飾クラス名		String
cacheConfiguration (common)	キャッシュの設定		設定
cacheConfiguration Properties (common)	CacheManager を作成するための javax.cache.spi.CachingProvider のプロパティー		Properties
configurationUri (common)	CacheManager の実装固有の URI		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

第161章 JCLOUDS COMPONENT

Camel バージョン 2.9 以降で利用可能

このコンポーネントにより、クラウドプロバイダーのキー値エンジン (BlobStores) およびコンピューティングサービスとのやり取りが可能になります。コンポーネントは jclouds を使用します。BlobStores とコンピューティングサービスの抽象化を提供するライブラリー。

ComputeService は、クラウド内のマシンを管理するタスクを簡素化します。たとえば、ComputeService を使用して 5 台のマシンを起動し、それらにソフトウェアをインストールできます。**BlobStore** は、Amazon S3 などのキー値プロバイダーの処理を簡素化します。たとえば、BlobStore を使用すると、コンテナーの単純なマップビューを表示できます。

camel jclouds コンポーネントでは、JcloudsBlobStoreEndpoint と JcloudsComputeEndpoint という 2 種類のエンドポイントを指定するため、両方の抽象化を使用できます。blobstore エンドポイントにはプロデューサーとコンシューマーの両方を配置できますが、コンピューティングエンドポイントにはプロデューサーしか配置できません。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jclouds</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

161.1. コンポーネントの設定

camel jclouds コンポーネントは、初期化中にコンポーネントに渡される限り、複数の jclouds blobstore と計算サービスを利用します。このコンポーネントは、blobstores とコンピュートサービスのリストを受け入れます。設定方法は次のとおりです。

```
  <bean id="jclouds" class="org.apache.camel.component.jclouds.JcloudsComponent">
    <property name="computeServices">
      <list>
        <ref bean="computeService"/>
      </list>
    </property>
    <property name="blobStores">
      <list>
        <ref bean="blobStore"/>
      </list>
    </property>
  </bean>

  <!-- Creating a blobstore from spring / blueprint xml -->
  <bean id="blobStoreContextFactory" class="org.jclouds.blobstore.BlobStoreContextFactory"/>

  <bean id="blobStoreContext" factory-bean="blobStoreContextFactory" factory-
method="createContext">
    <constructor-arg name="provider" value="PROVIDER_NAME"/>
    <constructor-arg name="identity"  value="IDENTITY"/>
    <constructor-arg name="credential" value="CREDENTIAL"/>
  </bean>
```

```
    <bean id="blobStore" factory-bean="blobStoreContext" factory-method="getBlobStore"/>

    <!-- Creating a compute service from spring / blueprint xml -->
    <bean id="computeServiceContextFactory"
class="org.jclouds.compute.ComputeServiceContextFactory"/>

    <bean id="computeServiceContext" factory-bean="computeServiceContextFactory" factory-
method="createContext">
        <constructor-arg name="provider" value="PROVIDER_NAME"/>
        <constructor-arg name="identity"  value="IDENTITY"/>
        <constructor-arg name="credential" value="CREDENTIAL"/>
    </bean>

    <bean id="computeService" factory-bean="computeServiceContext" factory-
method="getComputeService"/>
```

ご覧のとおり、コンポーネントは複数の blobstores とコンピュートサービスを処理できます。各エンドポイントで使用される実際の実装は、URI 内でプロバイダーを渡すことによって指定されます。

161.2. JCLOUD オプション

```
jclouds:blobstore:[provider id][?options]
jclouds:compute:[provider id][?options]
```

プロバイダー ID は、ターゲットサービスを提供するクラウドプロバイダーの名前です (**例: aws-s3 または aws_ec2**)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

161.3. ブロブストア URI オプション

JClouds コンポーネントは、以下にリストされている 3 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
blobStores (Common)	blobstore を使用するときに設定する必要がある特定の BlobStore を使用する場合。		List
computeServices (Common)	コンピュートを使用するときに設定する必要がある特定の ComputeService を使用する場合。		List
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JClouds エンドポイントは、URI 構文を使用して設定されます。

```
jclouds:command:providerId
```

パスおよびクエリーパラメーターを使用します。

161.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
command	**必須** blobstore やコンピュートなど、実行するコマンド。		JcloudsCommand
providerId	**必須** ターゲットサービスを提供するクラウドプロバイダーの名前 (例: aws-s3 または aws_ec2)。		String

161.3.2. クエリーパラメーター(15 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
blobName (blobstore)	ブロブの名前。		String
container (blobstore)	ブロブコンテナーの名前。		String

名前	説明	デフォルト	タイプ
directory (blobstore)	使用するオプションのディレクトリー名		String
group (compute)	新しく作成されたノードに割り当てられるグループ。値は、実際のクラウドプロバイダーによって異なります。		String
hardwareId (compute)	ノードの作成に使用されるハードウェア。値は、実際のクラウドプロバイダーによって異なります。		String
imageId (compute)	ノードの作成に使用される imageId。値は、実際のクラウドプロバイダーによって異なります。		String
locationId (compute)	ノードの作成に使用されるロケーション。値は、実際のクラウドプロバイダーによって異なります。		String
nodeId (compute)	スクリプトを実行するか破棄するノードの ID。		String
nodeState (compute)	ノードのステータスでフィルタリングして、実行中のノードのみを選択するなど。		String
operation (compute)	blobstore に対して実行される操作のタイプを指定します。		String
user (compute)	スクリプトを実行するターゲットノードのユーザー。		String

これらのオプションはいくつでも使用できます。

```
jclouds:blobstore:aws-s3?
operation=CamelJcloudsGet&container=mycontainer&blobName=someblob
```

プロデューサーエンドポイントの場合、適切なヘッダーをメッセージに渡すことで、上記の URI オプションをすべてオーバーライドできます。

161.3.3. blobstore のメッセージヘッダー

ヘッダー	説明
CamelJcloudsOperation	ブロブに対して実行される操作。有効なオプションは * PUT * GET です

ヘッダー	説明
CamelJcloudsContainer	ブロブコンテナーの名前。
CamelJcloudsBlobName	ブロブの名前。

161.4. BLOBSTORE の使用例

161.4.1. 例 1: ブロブへの書き込み

この例では、jclouds コンポーネントを使用して blob 内にメッセージを保存する方法を示します。

```
from("direct:start")
    .to("jclouds:blobstore:aws-s3" +
        "?operation=PUT" +
        "&container=mycontainer" +
        "&blobName=myblob");
```

上記の例では、URI パラメーターのいずれかをメッセージのヘッダーでオーバーライドできます。上記の例が xml を使用してルートを定義すると、次のようになります。

```
<route>
    <from uri="direct:start"/>
    <to uri="jclouds:blobstore:aws-s3?operation=PUT&container=mycontainer&blobName=myblob"/>
</route>
```

161.4.2. 例 2: ブロブからの取得/読み取り

この例では、jclouds コンポーネントを使用してブロブの内容を読み取る方法を示します。

```
from("direct:start")
    .to("jclouds:blobstore:aws-s3" +
        "?operation=GET" +
        "&container=mycontainer" +
        "&blobName=myblob");
```

上記の例では、URI パラメーターのいずれかをメッセージのヘッダーでオーバーライドできます。上記の例が xml を使用してルートを定義すると、次のようになります。

```
<route>
    <from uri="direct:start"/>
```

```
    <to uri="jclouds:blobstore:aws-s3?operation=PUT&container=mycontainer&blobName=myblob"/>
</route>
```

161.4.3. 例 3: ブロブの消費

この例では、指定されたコンテナーの下にあるすべてのブロブが消費されます。生成された交換には、ブロブのペイロードが本文として含まれます。

```
    from("jclouds:blobstore:aws-s3" +
        "?container=mycontainer")
        .to("direct:next");
```

以下に示すように、xml を使用して同じ目標を達成できます。

```
<route>
    <from uri="jclouds:blobstore:aws-s3?
operation=GET&container=mycontainer&blobName=myblob"/>
    <to uri="direct:next"/>
</route>
```

```
jclouds:compute:aws-ec2?
operation=CamelJcloudsCreateNode&imageId=AMI_XXXXX&locationId=eu-west-1&group=mygroup
```

161.5. コンピュート使用状況のサンプル

以下は、java dsl および spring/blueprint xml での jclouds コンピュートプロデューサーの使用を示すいくつかの例です。

161.5.1. 例 1: 利用可能なイメージを一覧表示します。

```
    from("jclouds:compute:aws-ec2" +
        "&operation=CamelJCloudsListImages")
        .to("direct:next");
```

これにより、本文内にイメージのリストを含むメッセージが作成されます。xml を使用して同じことを行うこともできます。

```
<route>
    <from uri="jclouds:compute:aws-ec2?operation=CamelJCloudsListImages"/>
    <to uri="direct:next"/>
</route>
```

161.5.2. 例 2: 新しいノードを作成します。

```
    from("direct:start").
    to("jclouds:compute:aws-ec2" +
        "?operation=CamelJcloudsCreateNode" +
        "&imageId=AMI_XXXXX" +
        "&locationId=XXXXX" +
        "&group=myGroup");
```

これにより、クラウドプロバイダーに新しいノードが作成されます。この場合の出力メッセージは、新しく作成されたノードに関する情報 (IP、ホスト名など) を含む一連のメタデータになります。これは、Spring xml を使用した場合と同じです。

```
<route>
    <from uri="direct:start"/>
    <to uri="jclouds:compute:aws-ec2?
operation=CamelJcloudsCreateNode&imageId=AMI_XXXXX&locationId=XXXXX&group=myGroup"/>
</route>
```

161.5.3. 例 3: 実行中のノードでシェルスクリプトを実行します。

```
    from("direct:start").
    to("jclouds:compute:aws-ec2" +
        "?operation=CamelJcloudsRunScript" +
        "?nodeId=10" +
        "&user=ubuntu");
```

上記のサンプルは、実行されるシェルスクリプトを含むと予想される in メッセージのボディーを取得します。スクリプトが取得されると、指定されたユーザー (**ubuntu の場合**) の下で実行するためにノードに送信されます。ターゲットノードは、その nodeId を使用して指定されます。nodeId は、ノードの作成時に、結果のメタデータの一部になるか、または LIST_NODES 操作を実行することによって取得できます。

注記 これには、コンポーネントに渡されるコンピュートサービスが、適切な jclouds ssh 対応モジュール (**jsch または sshj など**) で初期化される必要があります。

これは、Spring xml を使用した場合と同じです。

```
<route>
    <from uri="direct:start"/>
    <to uri="jclouds:compute:aws-ec2?operation=CamelJcloudsListNodes&?
nodeId=10&user=ubuntu"/>
</route>
```

161.5.4. その他の参考資料

jclouds について詳しく知りたい場合は、興味深いリソースのリストをご覧ください。

Jclouds Blobstore wiki

Jclouds コンピュート wiki Compute wiki

第162章 JCR コンポーネント

Camel バージョン 1.3 以降で利用可能

jcr コンポーネントを使用すると、プロデューサーを使用して JCR 準拠のコンテンツリポジトリー (Apache Jackrabbit など) にノードを追加したり、そこからノードを読み取ったり、コンシューマーに EventListener を登録したりできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jcr</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

162.1. URI 形式

```
jcr://user:password@repository/path/to/node
```

コンシューマーが追加された

Camel 2.10 以降では、consumer を JCR の EventListener として、またはプロデューサーとして使用して、識別子でノードを読み取ることができます。

162.2. 使用方法

URI の **repository** 要素は、Camel コンテキストレジストリーで JCR **Repository** オブジェクトを検索するために使用されます。

162.2.1. JCR オプション

JCR コンポーネントにはオプションがありません。

JCR エンドポイントは、URI 構文を使用して設定されます。

```
jcr:host/base
```

パスおよびクエリーパラメーターを使用します。

162.2.2. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** 使用する Camel レジストリーからルックアップする javax.jcr.Repository の名前。		String
base	リポジトリーにアクセスするときにベースノードを取得する		String

162.2.3. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
deep (Common)	isDeep が true の場合、関連する親ノードが absPath またはそのサブグラフ内にあるイベントが受信されます。	false	boolean
eventTypes (Common)	eventTypes (javax.jcr.observation.Event.NODE_ADDED, javax.jcr.observation.Event.NODE_REMOVED などのビットマスク値としてエンコードされた 1 つ以上のイベントタイプの組み合わせ)。		int
nodeTypeNames (Common)	コンマで区切られた nodeTypeName リスト文字列が設定されている場合、関連付けられた親ノードがこのリスト内のいずれかのノードタイプ (またはいずれかのノードタイプのサブタイプ) を持つイベントのみが受信されます。		String
noLocal (Common)	noLocal が true の場合、リスナーが登録されたセッションによって生成されたイベントは無視されます。それ以外の場合、それらは無視されません。	false	boolean
password (common)	ログイン用パスワード		String
sessionLiveCheckInterval (common)	各セッションのライブチェックの前に待機するミリ秒単位の間隔。デフォルト値は 60000 ミリ秒です。	60000	long
sessionLiveCheckIntervalOnStart (Common)	最初のセッションのライブチェックの前に待機するミリ秒単位の間隔。デフォルト値は 3000 ミリ秒です。	3000	long
username (common)	ログイン用のユーザー名		String
uuids (Common)	コンマで区切られた uuid リスト文字列が設定されている場合、関連する親ノードがコンマで区切られた uuid リストの識別子の 1 つを持つイベントのみが受信されます。		String
workspaceName (Common)	アクセスするワークスペース。指定されていない場合は、デフォルトのものが使用されます		String

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

2.12.3 より前のバージョンの Camel では、JCR Producer がメッセージヘッダーの代わりにメッセージプロパティーを使用していたことに注意してください。詳細については、https://issues.apache.org/jira/browse/CAMEL-7067 を参照してください。

162.3. 例

以下のスニペットは、コンテンツリポジトリーの **/home/test** ノードの下に node という名前の **node** を作成します。送信されるメッセージの本文を含む **my.contents.property** という 1 つの追加プロパティーもノードに追加されます。

```
from("direct:a").setHeader(JcrConstants.JCR_NODE_NAME, constant("node"))
    .setHeader("my.contents.property", body())
    .to("jcr://user:pass@repository/home/test");
```

次のコードは、Event.NODE_ADDED および Event.NODE_REMOVED イベント (イベントタイプ 1 および 2、両方とも 3 としてマスク) のパス import-application/inbox の下に EventListener を登録し、すべての子を深くリッスンします。

```
<route>
    <from uri="jcr://user:pass@repository/import-application/inbox?eventTypes=3&deep=true" />
    <to uri="direct:execute-import-application" />
</route>
```

162.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第163章 JDBC コンポーネント

Camel バージョン 1.2 以降で利用可能

jdbc コンポーネントを使用すると、SQL クエリー (SELECT) と操作 (INSERT、UPDATE など) がメッセージ本文で送信される JDBC を介してデータベースにアクセスできます。このコンポーネントは、spring-jdbc を使用する SQL コンポーネント コンポーネントとは異なり、標準の JDBC API を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jdbc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントはプロデューサーエンドポイントの定義にのみ使用できます。つまり、**from()** ステートメントで JDBC コンポーネントを使用することはできません。

163.1. URI 形式

```
jdbc:dataSourceName[?options]
```

このコンポーネントはプロデューサーエンドポイントのみをサポートします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

163.2. オプション

JDBC コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
dataSource (producer)	レジストリーから名前でデータソースを検索する代わりに、DataSource インスタンスを使用します。		DataSource
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JDBC エンドポイントは、URI 構文を使用して設定されます。

```
jdbc:dataSourceName
```

パスおよびクエリーパラメーターを使用します。

163.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
dataSourceName	**必須** レジストリーで検索するための DataSource の名前。		文字列

163.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
allowNamedParameters (producer)	クエリーで名前付きパラメーターの使用を許可するかどうか。	true	boolean
outputClass (producer)	outputType=SelectOne または SelectList の場合、変換として使用する完全なパッケージ名とクラス名を指定します。		String
outputType (producer)	プロデューサーが使用する出力を決定します。	SelectList	JdbcOutputType
parameters (producer)	java.sql.Statement へのオプションパラメーター。たとえば、maxRows、fetchSize などを設定します。		Map
readSize (producer)	ポーリングクエリーで読み取ることができるデフォルトの最大行数。デフォルト値は 0 です。		int
resetAutoCommit (producer)	Camel は JDBC 接続の autoCommit を false に設定し、ステートメントの実行後に変更をコミットし、resetAutoCommit が true の場合、最後に接続の autoCommit フラグをリセットします。JDBC 接続が autoCommit フラグのリセットをサポートしていない場合、resetAutoCommit フラグを false に設定すると、Camel は autoCommit フラグをリセットしようとしません。XA トランザクションで使用する場合は、トランザクションマネージャーがこの tx のコミットを担当するように、おそらく false に設定する必要があります。	true	boolean
transacted (producer)	トランザクションが使用中かどうか。	false	boolean
useGetBytesForBlob (producer)	BLOB 列を文字列データではなくバイトとして読み取る。これは、BLOB 列をバイトとして読み取る必要がある Oracle などの特定のデータベースで必要になる場合があります。	false	boolean

名前	説明	デフォルト	タイプ
useHeadersAsParameters (producer)	名前付きパラメーターで prepareStatementStrategy を使用するには、このオプションを true に設定します。これにより、名前付きプレースホルダーを使用してクエリーを定義し、クエリープレースホルダーの動的な値を持つヘッダーを使用できます。	false	boolean
useJDBC4ColumnNameAndLabelSemantics (producer)	列名を取得するときに、JDBC 4 または JDBC 3.0 以前のセマンティックを使用するかどうかを設定します。JDBC 4.0 は columnLabel を使用して列名を取得しますが、JDBC 3.0 は columnName または columnLabel の両方を使用します。残念ながら、JDBC ドライバーは動作が異なるため、このコンポーネントを使用して問題が発生した場合は、このオプションを使用して JDBC ドライバーに関する問題を解決できます。このオプションはデフォルトで true です。	true	boolean
beanRowMapper (advanced)	outputClass の使用時にカスタム org.apache.camel.component.jdbc.BeanRowMapper を使用するには。デフォルトの実装では、行名が小文字になり、アンダースコアとダッシュがスキップされます。たとえば、CUST_ID は custId としてマップされます。		BeanRowMapper
prepareStatementStrategy (advanced)	プラグインがカスタム org.apache.camel.component.jdbc.JdbcPrepareStatementStrategy を使用してクエリーと準備済みステートメントの準備を制御できるようにします。		JdbcPrepareStatement Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

163.3. 結果

デフォルトでは、結果は **ArrayList<HashMap<String, Object>>** として OUT ボディーに返されます。**List** オブジェクトには行のリストが含まれ、**Map** オブジェクトには列名として **String** キーを持つ各行が含まれます。オプション **outputType** を使用して、結果を制御できます。

注記: このコンポーネントは **ResultSetMetaData** をフェッチして、列名を **Map** のキーとして返すことができるようにします。

163.3.1. メッセージヘッダー

ヘッダー	説明
CamelJdbcRowCount	クエリーが **SELECT** の場合、この OUT ヘッダーに行数が返されます。
CamelJdbcUpdateCount	クエリーが **UPDATE** の場合、この OUT ヘッダーで更新カウントが返されます。
CamelGeneratedKeysRows	Camel 2.10: 生成された kets を含む行。
CamelGeneratedKeysRowCount	Camel 2.10: 生成されたキーを含むヘッダー内の行数。
CamelJdbcColumnNames	Camel 2.11.1: **java.util.Set** 型としての ResultSet からの列名。
CamelJdbcParametes	Camel 2.12: **useHeadersAsParameters** が有効になっている場合に使用されるヘッダーを持つ **java.util.Map**。

163.4. 生成されたキー

Camel 2.10 以降で利用可能

SQL INSERT を使用してデータを挿入する場合、RDBMS は自動生成されたキーをサポートしている可能性があります。生成されたキーをヘッダーで返すように JDBC プロデューサーに指示できます。これを行うには、ヘッダー **CamelRetrieveGeneratedKeys=true** を設定します。次に、生成されたキーは、上記の表にリストされているキーを含むヘッダーとして提供されます。

この 単体テスト で詳細を確認できます。

生成されたキーの使用は、名前付きパラメーターと一緒には機能しません。

163.5. 名前付きパラメーターの使用

Camel 2.12 以降で利用可能

以下の特定のルートでは、projects テーブルからすべてのプロジェクトを取得します。SQL クエリーには、:?lic と :?min という 2 つの名前付きパラメーターがあることに注意してください。
その後、Camel はメッセージヘッダーからこれらのパラメーターを検索します。上記の例では、2 つのヘッダーに定数値を設定していることに注意してください。
名前付きパラメーターの場合:

```
from("direct:projects")
   .setHeader("lic", constant("ASF"))
   .setHeader("min", constant(123))
   .setBody("select * from projects where license = :?lic and id > :?min order by id")
   .to("jdbc:myDataSource?useHeadersAsParameters=true")
```

ヘッダー値を **java.util.Map** に保存し、キー **CamelJdbcParameters** を使用してヘッダーにマップを保存することもできます。

163.6. サンプル

次の例では、customer テーブルから行をフェッチします。

まず、データソースを Camel レジストリーに **testdb** として登録します。

次に、JDBC コンポーネントにルーティングするルートを設定して、SQL が実行されるようにします。前のステップでバインドされた **testdb** データソースを参照する方法に注意してください。

または、次のように Spring で **DataSource** を作成できます。

エンドポイントを作成し、IN メッセージの本文に SQL クエリーを追加して、エクスチェンジを送信します。クエリーの結果は、OUT ボディーで返されます。

一度に ResultSet 全体ではなく、行を 1 つずつ処理する場合は、次のようなスプリッター EIP を使用する必要があります。

Camel 2.13.x 以前

Camel 2.14.x 以降

```
from("direct:hello")
// here we split the data from the testdb into new messages one by one
// so the mock endpoint will receive a message per row in the table
// the StreamList option allows to stream the result of the query without creating a List of rows
// and notice we also enable streaming mode on the splitter
.to("jdbc:testdb?outputType=StreamList")
  .split(body()).streaming()
  .to("mock:result");
```

163.7. サンプル - データベースを毎分ポーリングする

JDBC コンポーネントを使用してデータベースをポーリングする場合は、Timer や Quartz などのポーリングスケジューラーと組み合わせる必要があります。次の例では、60 秒ごとにデータベースからデータを取得します。

```
from("timer://foo?period=60000").setBody(constant("select * from
customer")).to("jdbc:testdb").to("activemq:queue:customers");
```

163.8. サンプル - データソース間でデータを移動する

一般的なユースケースは、データのクエリー、処理、および別のデータソースへの移動 (ETL 操作) です。次の例では、1時間ごとにソーステーブルから新しい顧客レコードを取得し、それらをフィルター処理/変換して、宛先テーブルに移動します。

```
from("timer://MoveNewCustomersEveryHour?period=3600000")
    .setBody(constant("select * from customer where create_time > (sysdate-1/24)"))
    .to("jdbc:testdb")
    .split(body())
        .process(new MyCustomerProcessor()) //filter/transform results as needed
        .setBody(simple("insert into processed_customer values('${body[ID]}','${body[NAME]}')"))
        .to("jdbc:testdb");
```

163.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- SQL

第164章 JETTY 9 コンポーネント

Camel バージョン 1.2 以降で利用可能

警告

プロデューサーは非推奨です - 使用しないでください。Jetty をコンシューマーとしてのみ使用することをお勧めします (例: jetty から)

jetty コンポーネントは、HTTP 要求を消費および生成するための HTTP ベースのエンドポイントを提供します。つまり、Jetty コンポーネントは単純な Web サーバーとして動作します。
Jetty は http クライアントとしても使用できます。つまり、プロデューサーとして Camel で使用することもできます。

ストリーム

コードは単体テストの一部であるため、この例では **assert** 呼び出しが表示されます。Jetty はストリームベースです。つまり、受信した入力はストリームとして Camel に送信されます。つまり、ストリームのコンテンツを **一度** だけ読み取ることができます。
メッセージボディーが空のように見える場合や、何度も Exchange.HTTP_RESPONSE_CODE データにアクセスする必要がある場合 (例: マルチキャストや再配送エラー処理)、ストリームキャッシュを使用するか、何度読み込んでも安全な **String** にメッセージボディーを変換する必要があります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jetty</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

164.1. URI 形式

```
jetty:http://hostname[:port][/resourceUri][?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

164.2. オプション

Jetty 9 コンポーネントは、以下に示す 33 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
sslKeyPassword (security)	キーストア内の証明書のキーエントリーにアクセスするために使用されるキーパスワード (これは、キーストアコマンドの -keypass オプションに指定されるパスワードと同じです)。		String
sslPassword (security)	キーストアファイルにアクセスするために必要な ssl パスワード (これは、キーストアコマンドの -storepass オプションに指定されるパスワードと同じです)。		String
keystore (security)	Java キーストアファイルのロケーションを指定します。このファイルには、Jetty サーバー独自の X.509 証明書がキーエントリーに含まれています。		String
errorHandler (advanced)	このオプションは、Jetty サーバーが使用する ErrorHandler を設定するために使用されます。		ErrorHandler
sslSocketConnectors (security)	ポート番号ごとに特定の SSL コネクターを含むマップ。		Map
socketConnectors (security)	ポート番号ごとに特定の HTTP コネクターを含むマップ。sslSocketConnectors と同じ原則を使用します。		Map
httpClientMinThreads (producer)	HttpClient スレッドプールのスレッドの最小数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		Integer
httpClientMaxThreads (producer)	HttpClient スレッドプールの最大スレッド数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		Integer
minThreads (consumer)	サーバースレッドプール内のスレッドの最小数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		Integer
maxThreads (consumer)	サーバースレッドプールのスレッドの最大数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		Integer
threadPool (consumer)	サーバーのカスタムスレッドプールを使用する場合。このオプションは、特別な状況でのみ使用してください。		ThreadPool
enableJmx (common)	このオプションが true の場合、Jetty JMX サポートがこのエンドポイントに対して有効になります。	false	boolean

名前	説明	デフォルト	タイプ
jettyHttpBinding (advanced)	カスタム org.apache.camel.component.jetty.JettyHttpBinding を使用するには、プロデューサーに対する応答の書き込み方法をカスタマイズするために使用します。		JettyHttpBinding
httpBinding (advanced)	使用しないでください - 代わりに JettyHttpBinding を使用してください。		HttpBinding
httpConfiguration (advanced)	Jetty コンポーネントは HttpConfiguration を使用しません。		HttpConfiguration
mbContainer (advanced)	Jetty が mbeans の登録に使用する JMX が有効な場合、既存の設定済み org.eclipse.jetty.jmx.MBeanContainer を使用します。		MBeanContainer
sslSocketConnector Properties (security)	一般的な SSL コネクタープロパティーを含むマップ。		Map
socketConnector Properties (security)	一般的な HTTP コネクタープロパティーを含むマップ。sslSocketConnectorProperties と同じ原則を使用します。		Map
continuationTimeout (consumer)	Jetty をコンシューマー (サーバー) として使用する場合、タイムアウトをミリ秒単位で設定できます。デフォルトでは、Jetty は 30000 を使用します。= 0 の値を使用して、期限切れにならないようにすることができます。タイムアウトが発生すると、リクエストは期限切れになり、Jetty は HTTP エラー 503 をクライアントに返します。このオプションは、Asynchronous Routing Engine で Jetty を使用する場合にのみ使用されます。	30000	Long
useContinuation (consumer)	Jetty サーバーに Jetty 継続を使用するかどうか。	true	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします	false	boolean
responseBufferSize (common)	Jetty コネクターの応答バッファーサイズのカスタム値を設定できます。		Integer

名前	説明	デフォルト	タイプ
requestBufferSize (common)	Jetty コネクターでリクエストバッファーサイズのカスタム値を設定できます。		Integer
requestHeaderSize (common)	Jetty コネクターで要求ヘッダーサイズのカスタム値を設定できます。		Integer
responseHeaderSize (common)	Jetty コネクターの応答ヘッダーサイズのカスタム値を設定できます。		Integer
proxyHost (proxy)	http プロキシーを使用してホスト名を設定する場合。		String
proxyPort (proxy)	http プロキシーを使用してポート番号を設定する場合。		Integer
useXForwardedFor Header (common)	HttpServletRequest.getRemoteAddr で X-Forwarded-For ヘッダーを使用する場合。	false	boolean
sendServerVersion (consumer)	オプションが true の場合、jetty サーバーは、リクエストを送信するクライアントに日付ヘッダーを送信します。注: 他の camel-jetty エンドポイントが同じポートを共有していないことを確認してください。そうしないと、このオプションが期待どおりに機能しない可能性があります。	true	boolean
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に Java シリアル化を許可するかどうか。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Jetty 9 エンドポイントは、URI 構文を使用して設定されます。

```
jetty:httpUri
```

パスおよびクエリーパラメーターを使用します。

164.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpUri	**必須** 呼び出す HTTP エンドポイントの URL。		URI

164.2.2. クエリーパラメーター(54 個のパラメーター):

名前	説明	デフォルト	タイプ
chunked (common)	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
disableStreamCache (common)	サーブレットからの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultHttpBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。サーブレットを使用してエンドポイントをブリッジ/プロキシーする場合、メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にしてパフォーマンスを向上させることを検討してください。http/http4 プロデューサーは、デフォルトでレスポンスボディーストリームをキャッシュします。このオプションを true に設定すると、プロデューサは応答本文ストリームをキャッシュせず、応答ストリームをそのままメッセージ本文として使用します。	false	boolean
enableMultipartFilter (Common)	Jetty org.eclipse.jetty.servlets.MultiPartFilter が有効かどうか。エンドポイントをブリッジするときは、この値を false に設定して、マルチパートリクエストも確実にプロキシー/ブリッジされるようにする必要があります。	false	boolean

名前	説明	デフォルト	タイプ
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
transferException (common)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
httpBinding (common)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
async (consumer)	非同期モードで動作するようにコンシューマーを設定します	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
continuationTimeout (consumer)	Jetty をコンシューマー (サーバー) として使用する場合、タイムアウトをミリ秒単位で設定できます。デフォルトでは、Jetty は 30000 を使用します。= 0 の値を使用して、期限切れにならないようにすることができます。タイムアウトが発生すると、リクエストは期限切れになり、Jetty は HTTP エラー 503 をクライアントに返します。このオプションは、Asynchronous Routing Engine で Jetty を使用する場合にのみ使用されます。	30000	Long
enableCORS (consumer)	オプションが true の場合、Jetty サーバーは、すぐに使用できる CORS をサポートする CrossOriginFilter をセットアップします。	false	boolean

名前	説明	デフォルト	タイプ
enableJmx (consumer)	このオプションが true の場合、Jetty JMX サポートがこのエンドポイントに対して有効になります。詳細については、Jetty JMX サポートを参照してください。	false	boolean
httpMethodRestrict (consumer)	GET/POST/PUT など、HttpMethod が一致する場合にのみ消費を許可するために使用されます。複数のメソッドをコンマで区切って指定できます。		String
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、コンシューマーが URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse.		Integer
sendDateHeader (consumer)	オプションが true の場合、jetty サーバーは、リクエストを送信するクライアントに日付ヘッダーを送信します。注: 他の camel-jetty エンドポイントが同じポートを共有していないことを確認してください。そうしないと、このオプションが期待どおりに機能しない可能性があります。	false	boolean
sendServerVersion (consumer)	オプションが true の場合、jetty はリクエストを送信するクライアントに、jetty のバージョン情報を含むサーバーヘッダーを送信します。注: 他の camel-jetty エンドポイントが同じポートを共有していないことを確認してください。そうしないと、このオプションが期待どおりに機能しない可能性があります。	true	boolean
sessionSupport (consumer)	Jetty のサーバー側でセッションマネージャーを有効にするかどうかを指定します。	false	boolean
useContinuation (consumer)	Jetty サーバーに Jetty 継続を使用するかどうか。		Boolean
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 または存在しない場合に、HTTP リクエストにコンテンツがあるかどうかを先行チェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
filterInitParameters (consumer)	フィルター初期パラメーターの設定。これらのパラメーターは、jetty サーバーを起動する前にフィルターリストに適用されます。		Map
filtersRef (consumer)	リストに入れられ、レジストリーで見つけることができるカスタムフィルターを使用できます。複数の値はコンマで区切ることができます。		String
handlers (consumer)	レジストリーで検索する Handler インスタンスのコンマ区切りのセットを指定します。これらのハンドラーは、Jetty サーブレットコンテキストに追加されます (たとえば、セキュリティーを追加するため)。重要: 同じポート番号を使用して、異なる Jetty エンドポイントで異なるハンドラーを使用することはできません。ハンドラーはポート番号に関連付けられています。別のハンドラーが必要な場合は、別のポート番号を使用してください。		String
httpBindingRef (consumer)	**非推奨** リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。		String
multipartFilter (consumer)	カスタムマルチパートフィルターの使用を許可します。注記: multipartFilterRef を設定すると、enableMultipartFilter の値が強制的に true になります。		Filter
multipartFilterRef (consumer)	**非推奨** カスタムマルチパートフィルターの使用を許可します。注記: multipartFilterRef を設定すると、enableMultipartFilter の値が強制的に true になります。		String
optionsEnabled (consumer)	このサーブレットコンシューマーに対して HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは、OPTIONS はオフになっています。	false	boolean

名前	説明	デフォルト	タイプ
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI を要求に使用します。オプション throwExceptionOnFailure を false に設定して、HttpProducer がすべての障害応答を送り返すようにすることもできます。	false	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは、connectionClose は false です。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
copyHeaders (producer)	このオプションが true の場合、IN 交換ヘッダーは、コピー戦略に従って OUT 交換ヘッダーにコピーされます。これを false に設定すると、HTTP 応答からのヘッダーのみを含めることができます (IN ヘッダーは伝播されません)。	true	boolean
httpClientMaxThreads (producer)	HttpClient スレッドプールの最大スレッド数の値を設定します。この設定は、コンポーネントレベルで設定されたすべての設定を上書きします。最小サイズと最大サイズの両方を設定する必要があることに注意してください。設定されていない場合、Jettys スレッドプールで使用される最大 254 スレッドにデフォルト設定されます。	254	Integer
httpClientMinThreads (producer)	HttpClient スレッドプールのスレッドの最小数の値を設定します。この設定は、コンポーネントレベルで設定されたすべての設定を上書きします。最小サイズと最大サイズの両方を設定する必要があることに注意してください。設定されていない場合、Jettys スレッドプールで使用される最小 8 スレッドにデフォルト設定されます。	8	Integer
httpMethod (producer)	使用する HTTP メソッドを設定します。設定されている場合、HttpMethod ヘッダーはこのオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本文を読み取らず、入力ストリームをキャッシュしません。	false	boolean

名前	説明	デフォルト	タイプ
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在の Exchange Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受信した Host ヘッダーにアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで役立ちます。Host ヘッダーを使用するアプリケーションが、プロキシーされたサービスの正確な URL を生成できるようにします。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
httpClient (producer)	このエンドポイントによって作成されたすべてのプロデューサーに使用する共有 HttpClient を設定します。デフォルトでは、各プロデューサーは新しい http クライアントを使用し、共有しません。重要: クライアントが使用されなくなった場合は、クライアントを停止するなど、共有クライアントのライフサイクルを必ず処理してください。Camel はクライアントで start メソッドを呼び出して、このエンドポイントがプロデューサーを作成するときに確実に開始されるようにします。このオプションは、特別な状況でのみ使用してください。		HttpClient
httpClientParameters (producer)	Jetty の HttpClient の設定。たとえば、httpClient.idleTimeout=30000 を設定すると、アイドルタイムアウトが 30 秒に設定されます。また、httpClient.timeout=30000 は、リクエスト/レスポンスコールを長時間実行している場合にタイムアウトを早くしたい場合に備えて、リクエストタイムアウトを 30 秒に設定します。		Map
jettyBinding (producer)	プロデューサーへの応答の書き込み方法をカスタマイズするために使用されるカスタム JettyHttpBinding を使用します。		JettyHttpBinding
jettyBindingRef (producer)	**非推奨** プロデューサー向けの応答の書き込み方法をカスタマイズするために使用されるカスタム JettyHttpBinding を使用すること。		String
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。	200-299	String

名前	説明	デフォルト	タイプ
urlRewrite (producer)	**非推奨** カスタム org.apache.camel.component.http.UrlRewrite を参照して、エンドポイントをブリッジ/プロキシーするときに URL を書き換えることができます。詳細は、http://camel.apache.org/urlrewrite.html を参照してください。		UrlRewrite
mapHttpMessageBody (advanced)	このオプションが true の場合、交換の IN exchange ボディは HTTP ボディにマップされます。これを false に設定すると、HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、交換の IN exchange Form Encoded ボディは HTTP にマップされます。これを false に設定すると、HTTP Form Encoded ボディマッピングが回避されます。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、交換の IN exchange ヘッダーは HTTP ヘッダーにマップされます。これを false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		String
proxyHost (proxy)	使用するプロキシーホスト名		String
proxyPort (proxy)	使用するプロキシーポート		int
authHost (security)	NTML で使用する認証ホスト		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

164.3. メッセージヘッダー

Camel は、HTTP コンポーネントと同じメッセージヘッダーを使用します。Camel 2.2 からは、(Exchange.HTTP_CHUNKED,CamelHttpChunked) ヘッダーも使用して、camel-jetty コンシューマーで chuched エンコーディングをオンまたはオフにします。

Camel は **すべての** request.parameter と request.headers にもデータを取り込みます。たとえば、クライアントリクエストの URL が **http://myserver/myserver?orderid=123** の場合、エクスチェンジには、値が 123 の **orderid** という名前のヘッダーが含まれます。

Camel 2.2.0 から、Get メソッドだけでなく、他の HTTP メソッドからもメッセージヘッダーから request.parameter を取得できるようになりました。

164.4. 使用方法

Jetty コンポーネントは、コンシューマーエンドポイントとプロデューサーエンドポイントの両方をサポートします。他の HTTP エンドポイントにプロデュースするもう 1 つのオプションは、HTTP コンポーネント を使用することです。

164.5. プロデューサーの例

警告

プロデューサーは非推奨です - 使用しないでください。Jetty をコンシューマーとしてのみ使用することをお勧めします (例: jetty から)

以下は、HTTP 要求を既存の HTTP エンドポイントに送信する方法の基本的な例です。

Java DSL で

```
from("direct:start").to("jetty://http://www.google.com");
```

または Spring XML で

```
<route>
  <from uri="direct:start"/>
  <to uri="jetty://http://www.google.com"/>
<route>
```

164.6. コンシューマーの例

このサンプルでは、**http://localhost:8080/myapp/myservice** で HTTP サービスを公開するルートを定義します。

ローカルホストの使用

URL で **localhost** を指定すると、Camel はローカルの TCP/IP ネットワークインターフェイスでのみエンドポイントを公開するため、動作するマシンの外部からはアクセスできません。

特定のネットワークインターフェイスで Jetty エンドポイントを公開する必要がある場合は、このインターフェイスの数値 IP アドレスをホストとして使用する必要があります。すべてのネットワークインターフェイスで Jetty エンドポイントを公開する必要がある場合は、**0.0.0.0** アドレスを使用する必要があります。

URI 接頭辞全体をリッスンするには、Jetty でワイルドカードをマッチさせるには を参照してください。

実際に HTTP でルートを公開する必要があり、すでにサーブレットがある場合は、代わりに サーブレットトランスポート を参照する必要があります。

ビジネスロジックは **MyBookService** クラスに実装されており、HTTP 要求の内容にアクセスして応答を返します。
注記: コードは単体テストの一部であるため、この例では **assert** 呼び出しが表示されます。

次のサンプルは、URI パラメーター **one** を含むすべてのリクエストをエンドポイント **mock:one** にルーティングし、その他すべてを **mock:other** にルーティングするコンテンツベースのルートを示しています。

したがって、クライアントが HTTP リクエスト **http://serverUri?one=hello** を送信すると、Jetty コンポーネントは HTTP リクエストパラメーターを **one** エクスチェンジの **in.header** にコピーします。次に、**simple** 言語を使用して、このヘッダーを含むエクスチェンジを特定のエンドポイントにルーティングし、他のすべてのエクスチェンジを別のエンドポイントにルーティングできます。Simple よりも強力な言語 (OGNL など) を使用した場合は、パラメーター値をテストし、ヘッダー値に基づいてルーティングを行うこともできます。

164.7. セッションサポート

セッションサポートオプション **sessionSupport** を使用して、**HttpSession** オブジェクトを有効にし、エクスチェンジの処理中にセッションオブジェクトにアクセスできます。たとえば、次のルートはセッションを有効にします。

```
<route>
  <from uri="jetty:http://0.0.0.0/myapp/myservice/?sessionSupport=true"/>
  <processRef ref="myCode"/>
<route>
```

myCode プロセッサーは、Spring **Bean** 要素によってインスタンス化できます。

```
<bean id="myCode"class="com.mycompany.MyCodeProcessor"/>
```

プロセッサーの実装は、次のように **HttpSession** にアクセスできます。

```
public void process(Exchange exchange) throws Exception {
  HttpSession session = exchange.getIn(HttpMessage.class).getRequest().getSession();
  ...
}
```

164.8. SSL サポート (HTTPS)

JSSE 設定ユーティリティーの使用

Camel 2.8 の時点で、Jetty コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、Jetty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

JettyComponent jettyComponent = getContext().getComponent("jetty", JettyComponent.class);
jettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="jetty:https://127.0.0.1/mail/?sslContextParameters=#sslContextParameters"/>
...
```

Jetty を直接設定する

Jetty は、すぐに使える SSL サポートを提供します。Jetty を SSL モードで実行できるようにするには、URI を **https://** 接頭辞でフォーマットします。たとえば、次のようになります。

```
<from uri="jetty:https://0.0.0.0/myapp/myservice/"/>
```

Jetty は、正しい SSL 証明書をロードするために、キーストアのロード元と使用するパスワードを知る必要もあります。次の JVM システムプロパティーを設定します。

Camel 2.2 まで

- **jetty.ssl.keystore** は、Jetty サーバー独自の X.509 証明書を **キーエントリー** に含む Java キーストアファイルのロケーションを指定します。キーエントリーには、X.509 証明書 (事実上、**公開鍵**) とそれに関連付けられた秘密鍵が格納されます。
- **jetty.ssl.password** キーストアファイルにアクセスするために必要なストアパスワード (これは、**keystore** コマンドの **-storepass** オプションに指定されるパスワードと同じです)。
- **jetty.ssl.keypassword** キーストア内の証明書のキーエントリーにアクセスするために使用されるキーパスワード (これは、**keystore** コマンドの **-keypass** オプションに指定されるパスワードと同じです)。

Camel 2.3 以降

- **org.eclipse.jetty.ssl.keystore** は、Jetty サーバー独自の X.509 証明書を **キーエントリー** に含む Java キーストアファイルのロケーションを指定します。キーエントリーには、X.509 証明書 (事実上、**公開鍵**) とそれに関連付けられた秘密鍵が格納されます。

- **org.eclipse.jetty.ssl.password** キーストアファイルにアクセスするために必要なストアパスワード (これは、**keystore** コマンドの **-storepass** オプションに指定されるパスワードと同じです)。

- **org.eclipse.jetty.ssl.keypassword** キーストア内の証明書のキーエントリーにアクセスするために使用されるキーパスワード (これは、**keystore** コマンドの **-keypass** オプションに提供されるパスワードと同じです)。

Jetty エンドポイントで SSL を設定する方法の詳細については、Jetty サイトの次のドキュメントを参照してください: http://docs.codehaus.org/display/JETTY/How+to+configure+SSL

一部の SSL プロパティは Camel によって直接公開されていませんが、Camel は基礎となる SslSocketConnector を公開しており、これにより、クライアント証明書を必要とする相互認証のための needClientAuth や、クライアントが証明書を必要としないが証明書を持つことができる相互認証のための wantClientAuth などのプロパティを設定することができます。さまざまな Camel バージョンにはわずかな違いがあります。

Camel 2.2 まで

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectors">
    <map>
      <entry key="8043">
        <bean class="org.mortbay.jetty.security.SslSocketConnector">
          <property name="password"value="..."/>
          <property name="keyPassword"value="..."/>
          <property name="keystore"value="..."/>
          <property name="needClientAuth"value="..."/>
          <property name="truststore"value="..."/>
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

Camel 2.3, 2.4

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectors">
    <map>
      <entry key="8043">
        <bean class="org.eclipse.jetty.server.ssl.SslSocketConnector">
          <property name="password"value="..."/>
          <property name="keyPassword"value="..."/>
          <property name="keystore"value="..."/>
          <property name="needClientAuth"value="..."/>
          <property name="truststore"value="..."/>
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

-

*Camel 2.5 から SslSelectChannelConnector を使用するように切り替えます *

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectors">
    <map>
      <entry key="8043">
        <bean class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
          <property name="password"value="..."/>
          <property name="keyPassword"value="..."/>
          <property name="keystore"value="..."/>
          <property name="needClientAuth"value="..."/>
          <property name="truststore"value="..."/>
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

上記のマップでキーとして使用する値は、Jetty がリッスンするように設定したポートです。

164.8.1. 一般的な SSL プロパティーの設定

Camel 2.5 で利用可能

ポート番号ごとに固有の SSL ソケットコネクター (上記のように) の代わりに、すべての SSL ソケットコネクターに適用される一般的なプロパティーを設定できるようになりました (これは、上記のようにポート番号をエントリーとして明示的に設定するものではありません)。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectorProperties">
    <map>
      <entry key="password"value="..."/>
      <entry key="keyPassword"value="..."/>
      <entry key="keystore"value="..."/>
      <entry key="needClientAuth"value="..."/>
      <entry key="truststore"value="..."/>
    </map>
  </property>
</bean>
```

164.8.2. X509Certificate への参照を取得する方法

Jetty は、次のようにコードからアクセスできる HttpServletRequest に証明書への参照を格納します。

```
HttpServletRequest req = exchange.getIn().getBody(HttpServletRequest.class);
X509Certificate cert = (X509Certificate) req.getAttribute("javax.servlet.request.X509Certificate")
```

164.8.3. 一般的な HTTP プロパティーの設定

Camel 2.5 で利用可能

ポート番号ごとの特定の HTTP ソケットコネクター (上記のように) の代わりに、すべての HTTP ソケットコネクターに適用される一般的なプロパティーを設定できるようになりました (これは、上記のようにポート番号をエントリーとして明示的に設定するものではありません)。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="socketConnectorProperties">
    <map>
      <entry key="acceptors" value="4"/>
      <entry key="maxIdleTime" value="300000"/>
    </map>
  </property>
</bean>
```

164.8.4. Obtaining X-Forwarded-For header with HttpServletRequest.getRemoteAddr()

HTTP 要求が Apache サーバーによって処理され、mod_proxy を使用して jetty に転送される場合、元のクライアント IP アドレスは X-Forwarded-For ヘッダーにあり、HttpServletRequest.getRemoteAddr () は Apache プロキシーのアドレスを返します。

Jetty には、X-Forwarded-For から値を取得して HttpServletRequest remoteAddr プロパティーに配置する forwarded プロパティーがあります。 このプロパティーは、エンドポイント設定から直接利用することはできませんが、socketConnectors プロパティーを使用して簡単に追加できます。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="socketConnectors">
    <map>
      <entry key="8080">
        <bean class="org.eclipse.jetty.server.nio.SelectChannelConnector">
          <property name="forwarded" value="true"/>
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

これは、既存の Apache サーバーがドメインの TLS 接続を処理し、それらを内部でアプリケーションサーバーにプロキシーする場合に特に役立ちます。

164.9. HTTP ステータスコードを返すデフォルトの動作

HTTP ステータスコードのデフォルトの動作は、**org.apache.camel.component.http.DefaultHttpBinding** クラスによって定義されます。このクラスは、レスポンスの書き込み方法を処理し、HTTP ステータスコードも設定します。

エクスチェンジが正常に処理された場合、200 HTTP ステータスコードが返されます。
エクスチェンジが例外で失敗した場合、500 HTTP ステータスコードが返され、スタックトレースがボディーで返されます。返す HTTP ステータスコードを指定する場合は、OUT メッセージの **Exchange.HTTP_RESPONSE_CODE** ヘッダーにコードを設定します。

164.10. CUSTOMIZING HTTPBINDING

デフォルトでは、Camel は **org.apache.camel.component.http.DefaultHttpBinding** を使用して、レスポンスの書き込み方法を処理します。必要に応じて、独自の **HttpBinding** クラスを実装する

か、**DefaultHttpBinding** を 拡張して適切なメソッドをオーバーライドすることにより、この動作をカスタマイズできます。

次の例は、例外が返される方法を変更するために **DefaultHttpBinding** をカスタマイズする方法を示しています。

次に、バインディングのインスタンスを作成し、次のように Spring レジストリーに登録できます。

```
<bean id="mybinding"class="com.mycompany.MyHttpBinding"/>
```

そして、ルートを定義するときにこのバインディングを参照できます。

```
<route><from uri="jetty:http://0.0.0.0:8080/myapp/myservice?httpBindingRef=mybinding"/><to
uri="bean:doSomething"/></route>
```

164.11. JETTY ハンドラーとセキュリティー設定

エンドポイントで Jetty ハンドラーのリストを設定できます。これは、高度な Jetty セキュリティー機能を有効にするのに役立ちます。これらのハンドラーは、Spring XML で次のように設定されます。

```
<-- Jetty Security handling -->
<bean id="userRealm" class="org.mortbay.jetty.plus.jaas.JAASUserRealm">
   <property name="name" value="tracker-users"/>
   <property name="loginModuleName" value="ldaploginmodule"/>
</bean>

<bean id="constraint" class="org.mortbay.jetty.security.Constraint">
   <property name="name" value="BASIC"/>
   <property name="roles" value="tracker-users"/>
   <property name="authenticate" value="true"/>
</bean>

<bean id="constraintMapping" class="org.mortbay.jetty.security.ConstraintMapping">
   <property name="constraint" ref="constraint"/>
   <property name="pathSpec" value="/*"/>
</bean>

<bean id="securityHandler" class="org.mortbay.jetty.security.SecurityHandler">
   <property name="userRealm" ref="userRealm"/>
   <property name="constraintMappings" ref="constraintMapping"/>
</bean>
```

Camel 2.3 以降では、 次のように Jetty ハンドラーのリストを設定できます。

```
<-- Jetty Security handling -->
<bean id="constraint" class="org.eclipse.jetty.http.security.Constraint">
   <property name="name" value="BASIC"/>
   <property name="roles" value="tracker-users"/>
   <property name="authenticate" value="true"/>
</bean>

<bean id="constraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
   <property name="constraint" ref="constraint"/>
   <property name="pathSpec" value="/*"/>
```

```
</bean>

<bean id="securityHandler" class="org.eclipse.jetty.security.ConstraintSecurityHandler">
   <property name="authenticator">
      <bean class="org.eclipse.jetty.security.authentication.BasicAuthenticator"/>
   </property>
   <property name="constraintMappings">
      <list>
         <ref bean="constraintMapping"/>
      </list>
   </property>
</bean>
```

次に、エンドポイントを次のように定義できます。

```
from("jetty:http://0.0.0.0:9080/myservice?handlers=securityHandler")
```

さらにハンドラーが必要な場合は、**handlers** オプションを Bean ID のコンマ区切りのリストに等しく設定します。

164.12. カスタム HTTP 500 応答メッセージを返す方法

Camel Jetty が返信するデフォルトの返信メッセージの代わりに、何か問題が発生したときにカスタムの返信メッセージを返したい場合があります。
カスタム **HttpBinding** を使用してメッセージマッピングを制御することもできますが、多くの場合、Camel の Exception Clause を使用してカスタムレスポンスメッセージを作成する方が簡単です。例えば、このように HTTP エラーコード 500 で **Dude something went wrong** を返しています。

164.13. マルチパートフォームのサポート

Camel 2.3.0 から、マルチパートフォームポストへの camel-jetty のサポートがすぐに使用できるようになりました。送信されたフォームデータは、メッセージヘッダーにマップされます。Camel-jetty は、アップロードされたファイルごとに添付を作成します。ファイル名は、添付の名前にマップされます。コンテンツタイプは、添付名のコンテンツタイプとして設定されます。ここで例を見つけることができます。

注記: getName() は、バージョン 2.5 以降では以下のように機能します。以前のバージョンでは、代わりに添付の一時ファイル名を受け取ります

164.14. JETTY JMX サポート

Camel 2.3.0 から、camel-jetty は、コンポーネントおよびエンドポイントレベルでの Jetty の JMX 機能の有効化をサポートし、エンドポイント設定が優先されます。コンポーネントは Camel コンテキストに登録された MBeanServer への参照を Jetty に提供するため、このコンポーネントで JMX サポートを有効にするには、Camel コンテキスト内で JMX を有効にする必要があることに注意してください。
camel-jetty コンポーネントは、特定のプロトコル/ホスト/ポートの組み合わせに対して Jetty リソースをキャッシュして再利用するため、この設定オプションは、プロトコル/ホスト/ポートの組み合わせを使用する最初のエンドポイントの作成中にのみ評価されます。たとえば、次の XML フラグメントから作成された 2 つのルートがある場合、https://0.0.0.0 でリッスンするすべてのエンドポイントに対して JMX サポートが有効なままになります。

```
<from uri="jetty:https://0.0.0.0/myapp/myservice1/?enableJmx=true"/>
```

```
<from uri="jetty:https://0.0.0.0/myapp/myservice2/?enableJmx=false"/>
```

camel-jetty コンポーネントは、Jetty MBeanContainer の直接設定も提供します。Jetty は MBean 名を動的に作成します。Camel コンテキストの外部で Jetty の別のインスタンスを実行し、インスタンス間で同じ MBeanServer を共有している場合、Jetty MBean の登録時に名前の衝突を回避するために、両方のインスタンスに同じ MBeanContainer への参照を提供できます。

164.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- HTTP

第165章 JGROUPS COMPONENT

Camel バージョン 2.13 以降で利用可能

JGroups は、信頼できるマルチキャスト通信のためのツールキットです。**jgroups:** コンポーネントは、Camel インフラストラクチャーと JGroups クラスター間のメッセージエクスチェンジを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache-extras.camel-extra</groupId>
  <artifactId>camel-jgroups</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

Camel **2.13.0** から、JGroups コンポーネントは Camel Extra から Apache Camel の傘下に移されました。Camel **2.13.0** 以降を使用している場合は、代わりに次の POM エントリーを使用してください。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jgroups</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

165.1. URI 形式

```
jgroups:clusterName[?options]
```

clusterName は、コンポーネントが接続する JGroups クラスターの名前を表します。

165.2. オプション

JGroups コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
channel (Common)	使用するチャネル		JChannel
channelProperties (Common)	エンドポイントが使用する JChannel の設定プロパティーを指定します。		String
enableViewMessages (consumer)	true に設定すると、コンシューマーエンドポイントは (org.jgroups.Message インスタンスだけでなく) org.jgroups.View メッセージも受信します。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。	false	boolean

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JGroups エンドポイントは、URI 構文を使用して設定されます。

```
jgroups:clusterName
```

パスおよびクエリーパラメーターを使用します。

165.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
clusterName	**必須** コンポーネントが接続する JGroups クラスターの名前。		String

165.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
channelProperties (Common)	エンドポイントが使用する JChannel の設定プロパティーを指定します。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
enableViewMessages (consumer)	true に設定すると、コンシューマーエンドポイントは (org.jgroups.Message インスタンスだけでなく) org.jgroups.View メッセージも受信します。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

165.3. ヘッダー

ヘッダー	定数	以降のバージョン	説明
JGROUPS_ORIGINAL_MESSAGE	**JGroupsEndpoint.HEADER_JGROUPS_ORIGINAL_MESSAGE**	2.13.0	消費されたメッセージの本文が抽出された元の **org.jgroups.Message** インスタンス。
JGROUPS_SRC	`JGroupsEndpoint.`HEADER_JGROUPS_SRC	2.10.0	**Consumer** : 消費されたメッセージの **org.jgroups.Message**.getSrc()メソッドによって展開された **org.jgroups.Address** インスタンス。**Producer** : 送信されるメッセージのカスタムソース **org.jgroups.Address**。

ヘッダー	定数	以降のバージョン	説明
JGROUPS_DEST	`JGroupsEndpoint.`HEADER_JGROUPS_DEST	2.10.0	**Consumer** : 消費されたメッセージの **org.jgroups.Message**.getDest() メソッドによって抽出された **org.jgroups.Address** インスタンス。**Producer** : 送信されるメッセージのカスタム宛先 **org.jgroups.Address**。
JGROUPS_CHANNEL_ADDRESS	`JGroupsEndpoint.`HEADER_JGROUPS_CHANNEL_ADDRESS	2.13.0	エンドポイントに関連付けられたチャネルのアドレス (**org.jgroups.Address**)。

用途

ルートのコンシューマー側で **jgroups** コンポーネントを使用すると、エンドポイントに関連付けられた **JChannel** によって受信されたメッセージがキャプチャーされ、Camel ルートに転送されます。JGroups コンシューマーは、入力メッセージを 非同期的に 処理します。

```
// Capture messages from cluster named
// 'clusterName' and send them to Camel route.
from("jgroups:clusterName").to("seda:queue");
```

ルートのプロデューサー側で **jgroups** コンポーネントを使用すると、Camel エクスチェンジのボディーがエンドポイントによって管理される **JChannel** インスタンスに転送されます。

```
// Send message to the cluster named 'clusterName'
from("direct:start").to("jgroups:clusterName");
```

165.4. 定義済みフィルター

Camel のバージョン 2.13.0 から、JGroups コンポーネントには、**JGroupsFilters.** という名前の定義済みフィルターファクトリークラスが付属しています。

クラスターのコーディネーターに送信されたビュー変更通知のみを使用する (スレーブノードに送信されたこれらの通知を無視する) 場合は、**JGroupsFilters.dropNonCoordinatorViews()** フィルターを使用します。このフィルターは、単一の Camel ノードをクラスター内のマスターにしたい場合に特に便利です。これは、このフィルターを通過するメッセージが、特定のノードがクラスターのコーディネーターになったことを通知するためです。以下のスニペットは、マスターノードによって受信されたメッセージのみを収集する方法を示しています。

```
import static org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  to("seda:masterNodeEventsQueue");
```

165.5. 定義済みの式

Camel のバージョン 2.13.0 から、JGroups コンポーネントには、**JGroupsExpressions.** という名前の定義済み式ファクトリークラスが付属しています。

Camel コンテキストがまだ開始されていない場合にのみルートに影響を与える遅延を作成する場合は、**JGroupsExpressions.delayIfContextNotStarted(long delay)** ファクトリーメソッドを使用します。このファクトリーメソッドによって作成された式は、Camel コンテキストが **started** とは異なる状態にある場合にのみ、指定された遅延値を返します。この式は、クラスター内でシングルトン (マスター) ルートを維持するために JGroups コンポーネントを使用する場合に特に役立ちます。Camel コンテキストがまだ開始されていない場合、コントロールバスの**開始** コマンドはシングルトンルートを初期化しません。そのため、Camel コンテキストの起動後にマスタールートが確実に初期化されるように、マスタールートの起動を遅らせる必要があります。このようなシナリオはクラスターの初期化中にのみ発生する可能性があるため、新しいマスターになるスレーブノードの起動を遅らせたくありません。そのため、条件付き遅延式が必要です。

以下のスニペットは、JGroups コンポーネントで条件付き遅延を使用して、クラスター内のマスターノードの初期起動を遅らせる方法を示しています。

```
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.apache.camel.component.jgroups.JGroupsExpressions.delayIfContextNotStarted;
import static org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  threads().delay(delayIfContextNotStarted(SECONDS.toMillis(5))). // run in separated and delayed thread. Delay only if the context hasn't been started already.
  to("controlbus:route?routeId=masterRoute&action=start&async=true");

from("timer://master?repeatCount=1").routeId("masterRoute").autoStartup(false).to(masterMockUri);
```

165.6. 例

165.6.1. JGroups クラスターへの (からの) メッセージの送信 (受信)

JGroups クラスターにメッセージを送信するには、以下のスニペットで示されているように、プロデューサーエンドポイントを使用します。

```
from("direct:start").to("jgroups:myCluster");
...
producerTemplate.sendBody("direct:start", "msg")
```

上記のスニペットからメッセージを受信するには (同じ物理マシンまたは別の物理マシンで)、以下のコードフラグメントで示されているように、特定のクラスターからのメッセージをリッスンします。

```
mockEndpoint.setExpectedMessageCount(1);
mockEndpoint.message(0).body().isEqualTo("msg");
```

```
...
from("jgroups:myCluster").to("mock:messagesFromTheCluster");
...
mockEndpoint.assertIsSatisfied();
```

165.6.2. クラスタービューの変更通知を受け取る

以下のスニペットは、クラスターメンバーシップの変更に関する通知をリッスンするコンシューマーエンドポイントを作成する方法を示しています。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。

```
mockEndpoint.setExpectedMessageCount(1);
mockEndpoint.message(0).body().isInstanceOf(org.jgroups.View.class);
...
from("jgroups:clusterName?enableViewMessages=true").to(mockEndpoint);
...
mockEndpoint.assertIsSatisfied();
```

165.6.3. クラスター内にシングルトンルートを保持する

以下のスニペットは、キャメルコンテキストのクラスターにシングルトンコンシューマールートを保持する方法を示しています。マスターノードが停止するとすぐに、スレーブの1つが新しいマスターとして選出され、開始されます。この特定の例では、アドレス `http://localhost:8080/orders` でリクエストをリッスンするシングルトン jetty インスタンスを維持したいと考えています。

```
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.apache.camel.component.jgroups.JGroupsExpressions.delayIfContextNotStarted;
import static org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  threads().delay(delayIfContextNotStarted(SECONDS.toMillis(5))). // run in separated and delayed
thread. Delay only if the context hasn't been started already.
  to("controlbus:route?routeId=masterRoute&action=start&async=true");

from("jetty:http://localhost:8080/orders").routeId("masterRoute").autoStartup(false).to("jms:orders");
```

第166章 JIBX DATAFORMAT

Camel バージョン 2.6 以降で利用可能

JiBX は、JiBX ライブラリー を使用して Java オブジェクトを XML との間でマーシャリングおよびアンマーシャリングするデータ形式です。

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().jibx().
  to("mqseries:Another.Queue");
```

マーシャリングプロセスは実行時にメッセージタイプを認識できることに注意してください。ただし、XML からメッセージをアンマーシャリングするときは、ターゲットクラスを明示的に指定する必要があります。

```
// lets turn XML into PurchaseOrder message
from("mqseries:Another.Queue").
  unmarshal().jibx(PurchaseOrder.class).
  to("activemq:My.Queue");
```

166.1. オプション

JiBX データ形式は、以下に示す 3 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
unmarshallClass		**String**	XML から Java に非整列化するときに使用するクラス名。
bindingName		**String**	カスタムバインディングファクトリーを使用する場合。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

166.2. JIBX SPRING DSL

JiBX データ形式は、Camel Spring DSL でもサポートされています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

  <!-- Define data formats -->
  <dataFormats>
    <jibx id="jibx" unmarshallClass="org.apache.camel.dataformat.jibx.PurchaseOrder"/>
  </dataFormats>

  <!-- Marshal message to XML -->
  <route>
```

```
    <from uri="direct:marshal"/>
    <marshal ref="jibx"/>
    <to uri="mock:result"/>
  </route>

  <!-- Unmarshal message from XML -->
  <route>
    <from uri="direct:unmarshal"/>
    <unmarshal ref="jibx"/>
    <to uri="mock:result"/>
  </route>

</camelContext>
```

166.3. 依存関係

camel ルートで JiBX を使用するには、このデータ形式を実装する **camel-jibx** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jibx</artifactId>
  <version>2.6.0</version>
</dependency>
```

第167章 JING コンポーネント

Camel バージョン 1.1 以降で利用可能

Jing コンポーネントは、Jing ライブラリー を使用して、メッセージボディーの XML 検証を実行します。

- RelaxNG XML 構文
- RelaxNG Compact 構文

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jing</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

MSV コンポーネントは、RelaxNG XML 構文もサポートできることに注意してください。

167.1. URI 形式 CAMEL 2.16

```
jing:someLocalOrRemoteResource
```

Camel 2.16 から、コンポーネントは jing を名前として使用し、オプション compactSyntax を使用して RNG または RNC モードをオンにすることができます。

167.2. オプション

Jing コンポーネントにはオプションがありません。

Jing エンドポイントは、URI 構文を使用して設定されます。

```
jing:resourceUri
```

パスおよびクエリーパラメーターを使用します。

167.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	クラスパス上のローカルリソースへの **必須** URL、または検証対象のスキーマを含むファイルシステム上のリモートリソースまたはリソースへの完全な URL。		String

167.2.2. クエリーパラメーター(2 個のパラメーター):

名前	説明	デフォルト	タイプ
compactSyntax (producer)	RelaxNG コンパクト構文を使用して検証するかどうか。デフォルトでは、これは RelaxNG XML 構文 (rng) を使用する場合は false であり、RelaxNG コンパクト構文 (rnc) を使用する場合は true です。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

167.3. 例

次の 例 は、エンドポイント **direct:start** からのルートを設定する方法を示しており、このルートは、XML が指定された RelaxNG Compact Syntax スキーマ (クラスパスで提供される) と一致するかどうかに基づいて、**mock:valid** または **mock:invalid** の 2 つのエンドポイントのいずれかに移動します。

167.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第168章 JIRA コンポーネント

Camel バージョン 2.15 以降で利用可能

JIRA コンポーネントは、Atlassian の REST Java Client for JIRA をカプセル化することにより、JIRA API と対話します。現在、新しい問題と新しいコメントのポーリングを提供しています。 また、新しい問題を作成することもできます。

このエンドポイントは Webhook ではなく、単純なポーリングに依存しています。理由は次のとおりです。

- 信頼性安定性への懸念
- 通常、ポーリングするペイロードの種類は大きくありません (さらに、ページングは API で利用できます)。
- Webhook が失敗するような一般公開されていない場所で実行されているアプリをサポートする必要性

JIRA API はかなり拡張性があることに注意してください。 したがって、このコンポーネントを簡単に拡張して、追加の相互作用を提供できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jira</artifactId>
  <version>${camel-version}</version>
</dependency>
```

168.1. URI 形式

```
jira://endpoint[?options]
```

168.2. JIRA オプション

JIRA コンポーネントにはオプションがありません。

JIRA エンドポイントは、URI 構文を使用して設定されます。

```
jira:type
```

パスおよびクエリーパラメーターを使用します。

168.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
type	課題の新規作成やコメントの新規作成など、**必須** 操作		JIRAType

168.2.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
password (common)	ログイン用パスワード		String
serverUrl (common)	JIRA サーバーへの **必須** URL		String
username (common)	ログイン用のユーザー名		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	コンシューマーを使用して JIRA をクエリーするときの遅延 (秒単位)。	6000	int
jql (consumer)	JQL は、必要なデータを取得できるようにする JIRA のクエリー言語です。例: jql=project=MyProject ここで、MyProject は Jira のプロダクトキーです。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

168.3. JQL:

JQL URI オプションは、両方のコンシューマーエンドポイントで使用されます。 理論的には、プロジェクトキーなどの項目は URI オプション自体である可能性があります。 ただし、JQL の使用を要求することで、コンシューマーはより柔軟で強力になります。

最低限、コンシューマーは以下を必要とします。

```
jira://[endpoint]?[required options]&jql=project=[project key]
```

注意すべき重要な点の 1 つは、newIssue コンシューマーが自動的に ORDER BY key desc を JQL に追加することです。 これは、プロジェクト内のすべての問題をインデックス化するのではなく、起動処理を最適化するためです。

もう 1 つの注意点は、同様に、newComment コンシューマーは、プロジェクト内のすべての問題 **と** コメントをインデックス化する必要があることです。 したがって、大規模なプロジェクトでは、JQL 式を可能な限り最適化することが **不可欠** です。 たとえば、JIRA ツールキットプラグインにはコメント数カスタムフィールドが含まれています。クエリーでコメント数 > 0 を使用します。また、状態 (status=Open) に基づいて最小化したり、ポーリングの遅延を増やしたりするなどしてください。 例:

```
jira://[endpoint]?[required options]&jql=RAW(project=[project key] AND status in (Open, \"Coding In Progress\") AND \"Number of comments\">0)"
```

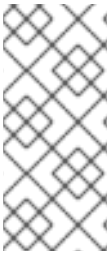
第169章 JMS コンポーネント

169.1. JMS コンポーネント

ヒント

ActiveMQ の使用

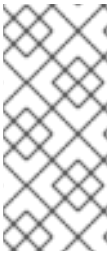
Apache ActiveMQ を使用している場合は、ActiveMQ 用に最適化されている ActiveMQ コンポーネントを優先する必要があります。このページのすべてのオプションとサンプルは、ActiveMQ コンポーネントにも有効です。



注記

トランザクションとキャッシング

JMS でトランザクションを使用している場合は、パフォーマンスに影響を与える可能性があるため、以下の **トランザクションとキャッシュレベル** セクションを参照してください。



注記

JMS を介したリクエスト/リプライ

Camel はパフォーマンスとクラスター化された環境を設定する多くのオプションを提供するため、リクエスト/リプライに関する重要な注意事項については、このページのさらに下にあるセクション **JMS を介したリクエスト - リプライ** を必ずお読みください。

このコンポーネントを使用すると、メッセージを JMS キューまたはトピックに送信 (またはそこから消費) できます。送信用の Spring の **JmsTemplate** や消費用の **MessageListenerContainer** など、Spring の JMS サポートを宣言型トランザクションに使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

169.2. URI 形式

```
jms:[queue:|topic:]destinationName[?options]
```

ここで、**destinationName** は JMS キューまたはトピック名です。デフォルトでは、**destinationName** はキュー名として解釈されます。たとえば、キューに接続するには、**FOO.BAR** を次のように使用します。

```
jms:FOO.BAR
```

必要に応じて、オプションの **queue:** 接頭辞を含めることができます。

```
jms:queue:FOO.BAR
```

トピックに接続するには、**topic:** 接頭辞を含める **必要** があります。たとえば、トピック **Stocks.Prices** に接続する
には、次を使用します。

```
jms:topic:Stocks.Prices
```

?option=value&option=value&... の形式を使用して、クエリーオプションを URI に追加します。

169.3. 注記

169.3.1. ActiveMQ の使用

JMS コンポーネントは、Spring 2 の **JmsTemplate** を再利用してメッセージを送信します。これは非 J2EE コンテナーでの使用には理想的ではなく、通常、パフォーマンスの低下 を避けるために JMS プロバイダーでのキャッシュが必要になります。

Message Broker として Apache ActiveMQ を使用する場合 (ActiveMQ が優れているため、これは適切な選択です)、次のいずれかを推奨します。

- ActiveMQ を効率的に使用するためにすでに最適化されている ActiveMQ コンポーネントを使用する
- ActiveMQ で **PoolingConnectionFactory** を使用します。

169.3.2. トランザクションとキャッシュレベル

メッセージを消費してトランザクションを使用している場合 (**transacted=true**)、キャッシュレベルのデフォルト設定がパフォーマンスに影響を与える可能性があります。

XA トランザクションを使用している場合は、XA トランザクションが正しく機能しなくなる可能性があるため、キャッシュできません。

XA を使用して **いない** 場合は、**cacheLevelName=CACHE_CONSUMER** を設定するなど、キャッシュを使用してパフォーマンスを高速化することを検討する必要があります。

Camel 2.7.x では、**cacheLevelName** のデフォルト設定は **CACHE_CONSUMER** で
す。**cacheLevelName=CACHE_NONE** を明示的に設定する必要があります。

Camel 2.8 以降では、**cacheLevelName** のデフォルト設定は **CACHE_AUTO** です。このデフォルトの自動モードはモードを検出し、それに応じてキャッシュレベルを設定します。

- **transacted=false** の場合は **CACHE_CONSUMER**
- **transacted=true** の場合は **CACHE_NONE**

したがって、デフォルト設定は保守的であると言えます。非 XA トランザクションを使用している場合は、**cacheLevelName=CACHE_CONSUMER** の使用を検討してください。

169.3.3. 永続サブスクリプション

永続的なトピックサブスクリプションを使用する場合は、**clientId** と **durableSubscriptionName** の両方を指定する必要があります。**clientId** の値は一意である必要があり、ネットワーク全体で単一の JMS 接続インスタンスによってのみ使用できます。この制限を回避するために、代わりに 仮想トピック を使用することをお勧めします。耐久性のあるメッセージングの詳細については、こちら をご覧ください。

169.3.4. メッセージヘッダーのマッピング

JMS 仕様では、メッセージヘッダーを使用する場合、ヘッダー名は有効な Java 識別子である必要があると規定されています。そのため、有効な Java 識別子になるようにヘッダーに名前を付けるようにしてください。これを行う利点の 1 つは、JMS セレクター内でヘッダーを使用できることです (その SQL92 構文では、ヘッダーの Java 識別子構文が義務付けられています)。

デフォルトでは、ヘッダー名をマッピングする単純な方法が使用されます。以下に示すように、ヘッダー名のドットとハイフンをすべて置き換え、ネットワーク経由で送信された JMS メッセージからヘッダー名が復元されたときに置き換えを元に戻す方法です。意味を確認するBean コンポーネントで呼び出すメソッド名が失われたり、ファイルコンポーネントのファイル名ヘッダーが失われたりすることはもうありません。

Camel でヘッダー名を受け入れるための現在のヘッダー名戦略は次のとおりです。

- ドットは **DOT** に置き換えられ、Camel がメッセージを消費すると置換が逆になります
- ハイフンは **HYPHEN** に置き換えられ、Camel がメッセージを消費すると置換が逆になります

169.4. オプション

JMS エンドポイントでさまざまなプロパティーを設定できます。これらのプロパティーは、JMSConfiguration POJO オブジェクトのプロパティーにマップされます。

警告

Spring JMS へのマッピング

これらのプロパティーの多くは、Camel がメッセージの送受信に使用する Spring JMS のプロパティーにマップされます。したがって、関連する Spring ドキュメントを参照することで、これらのプロパティーに関する詳細情報を取得できます。

169.4.1. コンポーネントのオプション

JMS コンポーネントは、以下に示す 80 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 JMS 設定を使用します。		JmsConfiguration

名前	説明	デフォルト	タイプ
acceptMessagesWhileStopping (consumer)	コンシューマーが停止中にメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止するが、キューにメッセージが入れられている場合は、このオプションを有効にすることを検討してください。このオプションが false の場合は、JMS ルートを停止すると、メッセージが拒否される可能性があり、JMS ブローカは再配信を試行する必要がありますが、これも拒否される可能性があり、最終的にメッセージはJMS ブローカー上のデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることをお勧めします。	false	boolean
allowReplyManagerQuickStop (consumer)	JmsConfigurationisAcceptMessagesWhileStopping が有効で、org.apache.camel.CamelContext が現在停止している場合に、要求/応答メッセージングのリプライマネージャーで使用される DefaultMessageListenerContainer が、DefaultMessageListenerContainer.runningAllowed フラグを迅速に停止できるようにするかどうか。このクイック停止機能は、通常の JMS コンシューマーではデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
acknowledgementMode (consumer)	整数として定義された JMS 確認応答モード。ベンダー固有の拡張機能を確認モードに設定できます。通常のモードでは、代わりに acknowledgementModeName を使用することをお勧めします。		int
eagerLoadingOfProperties (consumer)	メッセージが読み込まれるとすぐに JMS プロパティーの先行読み込みを有効にします。これは、JMS プロパティーが必要ない場合があるため一般的に非効率的ですが、基盤となる JMS プロバイダーと JMS プロパティーの使用に関する問題を早期に発見できる場合があります。	false	boolean
acknowledgementModeName (consumer)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	String
autoStartup (consumer)	コンシューマーコンテナーを自動起動するかどうかを指定します。	true	boolean
cacheLevel (consumer)	基礎となる JMS リソースの ID によってキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int

名前	説明	デフォルト	タイプ
cacheLevelName (consumer)	基礎となる JMS リソースのキャッシュレベルを名前で設定します。可能な値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、および CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring のドキュメントとトランザクションキャッシュレベルを参照してください。	CACHE_AUTO	String
replyToCacheLevelName (producer)	JMS を介して要求/応答を行うときに、応答コンシューマーのキャッシュレベルを名前で設定します。このオプションは、固定応答キュー (一時的ではない) を使用する場合にのみ適用されます。Camel はデフォルトで次を使用します: 排他的または replyToSelectorName と共有の CACHE_CONSUMER。そして、replyToSelectorName なしで共有するための CACHE_SESSION。IBM WebSphere などの一部の JMS ブローカーは、replyToCacheLevelName=CACHE_NONE を機能させるために設定する必要がある場合があります。注: 一時キューを使用する場合、CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などのより高い値を使用する必要があります。		String
clientId (common)	使用する JMS クライアント ID を設定します。この値を指定する場合は、一意である必要があり、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続的なトピックサブスクリプションの場合にのみ必要です。Apache ActiveMQ を使用している場合は、代わりに仮想トピックを使用することをお勧めします。		String
concurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーのデフォルト数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。	1	int
replyToConcurrent Consumers (producer)	JMS を介して要求/応答を行うときの同時コンシューマーのデフォルト数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。	1	int

名前	説明	デフォルト	タイプ
connectionFactory (common)	使用する接続ファクトリー。コンポーネントまたはエンドポイントで接続ファクトリーを設定する必要があります。		ConnectionFactory
username (security)	ConnectionFactory で使用するユーザー名。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
password (security)	ConnectionFactory で使用するパスワード。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
deliveryPersistent (producer)	デフォルトで永続配信を使用するかどうかを指定します。	true	boolean
deliveryMode (producer)	使用する配信モードを指定します。設定可能な値は、javax.jms.DeliveryMode で定義された値です。NON_PERSISTENT = 1 および PERSISTENT = 2。		Integer
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		String
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
errorHandler (advanced)	Message の処理中にキャッチされない例外が出力された場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ログレベルとスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングするよりも設定がはるかに簡単になります。		ErrorHandler
errorHandlerLoggingLevel (ロギング)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (ロギング)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean

名前	説明	デフォルト	タイプ
explicitQosEnabled (producer)	メッセージの送信時に、deliveryMode、priority、または timeToLive のサービス品質を使用する必要があるかどうかを設定します。このオプションは、Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で動作し、Camel In メッセージヘッダーから排他的に QoS プロパティーを読み取る preserveMessageQos オプションとは対照的です。	false	boolean
exposeListenerSession (consumer)	メッセージを消費するときにリスナーセッションを公開するかどうかを指定します。	false	boolean
idleTaskExecutionLimit (advanced)	実行中にメッセージを受信していない、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスクに受信を任せます (動的スケジューリングの場合。maxConcurrentConsumers 設定を参照してください)。Spring から入手できる追加のドキュメントがあります。	1	int
idleConsumerLimit (advanced)	常にアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
maxConcurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーの最大数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToMaxConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。		int
replyToMaxConcurrent Consumers (producer)	JMS を介した要求/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。		int
replyOnTimeoutToMax ConcurrentConsumers (producer)	JMS 経由の要求/応答を使用するときにタイムアウトが発生したときに、ルーティングを継続するための同時コンシューマーの最大数を指定します。	1	int

名前	説明	デフォルト	タイプ
maxMessagesPerTask (advanced)	タスクあたりのメッセージ数。-1 は無制限です。同時コンシューマーの範囲 (例: min max) を使用する場合、このオプションを使用して値を 100 などに設定し、必要な作業が少ない場合にコンシューマーが縮小する速度を制御できます。	-1	int
messageConverter (advanced)	カスタム Spring org.springframework.jms.support.converter.MessageConverter を使用して、javax.jms.Message との間でどのようにマッピングするかを制御できるようにします。		MessageConverter
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。詳細については、以下のマッピングの仕組みに関するセクションを参照してください。	true	boolean
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカーへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージのメッセージ ID を null に設定する必要があります。プロバイダーがヒントを無視する場合、メッセージ ID は通常の一意の値に設定する必要があります	true	boolean
messageTimestampEnabled (advanced)	メッセージの送信時にデフォルトでタイムスタンプを有効にするかどうかを指定します。	true	boolean
alwaysCopyMessage (producer)	true の場合、メッセージがプロデューサーに渡されて送信されると、Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、状況によってはメッセージのコピーが必要です (また、replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します)。	false	boolean
useMessageIDAsCorrelationID (advanced)	InOut メッセージの JMSCorrelationID として JMSMessageID を常に使用するかどうかを指定します。	false	boolean

名前	説明	デフォルト	タイプ
priority (producer)	1より大きい値は、送信時のメッセージの優先度を指定します (0 が最低の優先度で、9 が最高の優先度です)。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
pubSubNoLocal (advanced)	独自の接続によってパブリッシュされたメッセージの配信を禁止するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージ受信のタイムアウト (ミリ秒単位)。	1000	long
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	long
taskExecutor (consumer)	メッセージを消費するためのカスタムタスクエグゼキュータを指定できます。		TaskExecutor
timeToLive (producer)	メッセージの送信時に、メッセージの有効期限をミリ秒単位で指定します。	-1	long
取引済み (取引)	トランザクションモードを使用するかどうかを指定します	false	boolean
lazyCreateTransaction Manager (トランザクション)	true の場合、オプション transacted=true のときに transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (トランザクション)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (トランザクション)	使用するトランザクションの名前。		String
transactionTimeout (トランザクション)	トランザクションモードを使用している場合の、トランザクションのタイムアウト値 (秒単位)。	-1	int

名前	説明	デフォルト	タイプ
testConnectionOn Startup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel の起動時に、すべての JMS コンシューマーが JMS ブローカーへの有効な接続を持つことが保証されます。接続を許可できない場合、Camel は起動時に例外を出力します。これにより、接続に失敗した状態で Camel が開始されなくなります。JMS プロデューサーもテストされています。	false	boolean
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
forceSendOriginal Message (producer)	mapJmsMessage=false を使用すると、ルート中にヘッダーに触れると (get または set)、Camel は新しい JMS メッセージを作成して新しい JMS 宛先に送信します。Camel が受信した元の JMS メッセージを強制的に送信するには、このオプションを true に設定します。	false	boolean
requestTimeout (producer)	InOut Exchange パターン使用時の応答待ちタイムアウト (ミリ秒単位)。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めて、このエンドポイントで設定されたタイムアウト値をオーバーライドし、メッセージごとに個別のタイムアウト値を持つことができます。requestTimeoutCheckerInterval オプションも参照してください。	20000	long
requestTimeoutChecker Interval (advanced)	JMS を介してリクエスト/リプライを行うときに、Camel がタイムアウトになった Exchange をチェックする頻度を設定します。デフォルトでは、Camel は 1 秒に 1 回確認します。ただし、タイムアウトが発生したときに迅速に対応する必要がある場合は、この間隔を短くして、より頻繁にチェックすることができます。タイムアウトは、オプション requestTimeout によって決定されます。	1000	long

名前	説明	デフォルト	タイプ
transferExchange (advanced)	本文とヘッダーだけでなく、電信送金で交換を転送できます。次のフィールドが転送されます: In body、Out body、Fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、交換プロパティー、交換例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。プロデューサ側とコンシューマー側の両方でこのオプションを有効にする必要があるため、Camel はペイロードが Exchange であり、通常のペイロードではないことを認識します。	false	boolean
transferException (advanced)	有効で、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で失敗した場合、原因となった例外が javax.jms.ObjectMessage として応答で返されます。クライアントが Camel の場合、返された Exception は再出力されます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続的なキューを使用して堅牢なルーティングを有効にできます。transferExchange も有効にしている場合は、このオプションが優先されることに注意してください。キャッチされた例外はシリアライズ可能である必要があります。コンシューマー側の元の Exception は、プロデューサーに返されるときに org.apache.camel.RuntimeCamelException などの外部例外にラップできます。	false	boolean
transferFault (advanced)	これを有効にし、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で SOAP エラー (例外ではない) で失敗した場合には、リンク org.apache.camel.MessageisFault() のエラーフラグがレスポンスで、リンク JmsConstantsJMS_TRANSFER_FAULT のキーを含んだ JMS ヘッダーとして送り返されます。クライアントが Camel の場合には、返される障害フラグはリンク org.apache.camel.MessagesetFault(boolean) に設定されます。cxf や spring-ws などの SOAP ベースなどの障害をサポートする Camel コンポーネントを使用する場合、これを有効できます。	false	boolean
jmsOperations (advanced)	org.springframework.jms.core.JmsOperations インターフェイスの独自の実装を使用できるようにします。Camel はデフォルトで JmsTemplate を使用します。テスト目的で使用できますが、Spring API ドキュメントに記載されているほどは使用されません。		JmsOperations

名前	説明	デフォルト	タイプ
destinationResolver (advanced)	独自のリゾルバーを使用できるようにするプラグ可能な org.springframework.jms.support.destination.DestinationResolver (たとえば、JNDI レジストリーで実際の宛先を検索するため)。		DestinationResolver
replyToType (producer)	JMS を介して要求/応答を行うときに、replyTo キューに使用する戦略の種類を明示的に指定できます。可能な値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、replyTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに専用キューを使用できます。詳細については、Camel JMS のドキュメントを参照してください。特に、クラスター化された環境で実行する場合の影響に関する注意事項と、共有応答キューは代替の一時および排他的キューよりもパフォーマンスが低いという事実を参照してください。		ReplyToType
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定ではなく、メッセージで指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。次の 3 つのヘッダーは、JMSPriority、JMSDeliveryMode、および JMSExpiration と見なされます。それらのすべてまたは一部のみを指定できます。指定されていない場合、Camel は代わりにエンドポイントからの値を使用するようにフォールバックします。したがって、このオプションを使用すると、ヘッダーはエンドポイントからの値をオーバーライドします。対照的に、explicitQosEnabled オプションは、エンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値は使用しません。	false	boolean
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得できますが、前のメッセージは (非同期ルーティングエンジンによって) 非同期に処理されます。これは、メッセージが 100% 厳密に順序どおりに処理されない可能性があることを意味します。無効になっている場合 (デフォルト)、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。transactioned が有効になっている場合、トランザクションは同期的に実行する必要があるため、asyncConsumer=true は非同期的に実行されないことに注意してください (Camel 3.0 は非同期トランザクションをサポートする場合があります)。	false	boolean

名前	説明	デフォルト	タイプ
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます (例: ファイアアンドフォーゲット)。このオプションを有効にすると、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用された実際の JMSMessageID で Camel Exchange が強化されます。	false	boolean
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean
defaultTaskExecutor Type (consumer)	コンシューマーエンドポイントとプロデューサエンドポイントの ReplyTo コンシューマーの両方に対して、DefaultMessageListenerContainer で使用するデフォルトの TaskExecutor タイプを指定します。可能な値: SimpleAsync (Spring の SimpleAsyncTaskExecutor を使用) または ThreadPool (Spring の ThreadPoolTaskExecutor を最適な値で使用 - キャッシュされたスレッドプールのようなもの)。設定されていない場合は、デフォルトで以前の動作になり、コンシューマーエンドポイントにはキャッシュされたスレッドプールが使用され、応答コンシューマーには SimpleAsync が使用されます。ThreadPool の使用は、同時コンシューマーが動的に増減するエラスティック設定でスレッドのゴミを減らすために推奨されます。		DefaultTaskExecutor Type
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy

名前	説明	デフォルト	タイプ
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値を持つ可能性がある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型の値を含む接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名でこれを行います。コンマで区切られた複数のヘッダー名を指定し、ワイルドカードマッチングの接尾辞として使用できます。		String
queueBrowseStrategy (advanced)	キューを参照するときにカスタム QueueBrowseStrategy を使用します。		QueueBrowseStrategy
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
waitForProvisionCorrelationToBeUpdated Counter (advanced)	JMS を介して要求/応答を行う場合、およびオプション useMessageIDAsCorrelationID が有効な場合に、暫定相関 ID が実際の相関 ID に更新されるのを待機する回数。	50	int
waitForProvisionCorrelationToBeUpdated ThreadSleepingTime (advanced)	暫定相関 ID が更新されるのを待機するたびにスリープする間隔 (ミリ単位)。	100	long
correlationProperty (producer)	JMSCorrelationID プロパティーの代わりに、この JMS プロパティーを使用して、InOut 交換パターン (要求 - 応答) でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用してメッセージと相関性のないシステムとメッセージを交換できます。JMSCorrelationID を使用すると、Camel によって使用または設定されません。ここで指定されたプロパティーの値は、同じ名前でメッセージのヘッダーに指定されていない場合に生成されます。		String

名前	説明	デフォルト	タイプ
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常、subscriptionName 値と組み合わせて永続的なサブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常は subscriptionName 値と組み合わせて共有サブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。共有サブスクリプションも永続的である可能性があるため、このフラグを subscriptionDurable と組み合わせることもできます (多くの場合は組み合わせます)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 互換のメッセージブローカーが必要です。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有または永続的なサブスクリプションを持つトピック (pub-sub ドメイン) の場合に適用されます。サブスクリプション名は、このクライアントの JMS クライアント ID 内で一意である必要があります。デフォルトは、指定されたメッセージリスナーのクラス名です。注: 共有サブスクリプション (JMS 2.0 が必要) を除き、サブスクリプションごとに 1 つの同時コンシューマー (このメッセージリスナコンテナーのデフォルト) のみが許可されます。		String
streamMessageType Enabled (producer)	StreamMessage タイプを有効にするかどうかを設定します。ファイル、InputStream などのストリーミングの種類のメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、どの種類が使用されるかを制御します。デフォルトでは、BytesMessage が使用され、メッセージペイロード全体がメモリーに読み込まれます。このオプションを有効にすると、メッセージペイロードがチャンク単位でメモリーに読み込まれ、データがなくなるまで各チャンクが StreamMessage に書き込まれます。	false	boolean

名前	説明	デフォルト	タイプ
formatDateHeadersTo Iso8601 (producer)	日付ヘッダーを ISO 8601 標準に従ってフォーマットするかどうかを設定します。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

169.4.2. エンドポイントオプション

JMS エンドポイントは、URI 構文を使用して設定されます。

```
jms:destinationType:destinationName
```

パスおよびクエリーパラメーターを使用します。

169.4.3. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
destinationType	使用する宛先の種類	queue	String
destinationName	**必須** 宛先として使用するキューまたはトピックの名前		文字列

169.4.4. クエリーパラメーター (91 パラメーター)

名前	説明	デフォルト	タイプ
clientId (common)	使用する JMS クライアント ID を設定します。この値を指定する場合は、一意である必要があり、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続的なトピックサブスクリプションの場合にのみ必要です。Apache ActiveMQ を使用している場合は、代わりに仮想トピックを使用することをお勧めします。		String

名前	説明	デフォルト	タイプ
connectionFactory (common)	リンク setTemplateConnectionFactory (ConnectionFactory) またはリンク setListenerConnectionFactory (ConnectionFactory) のどちらにも接続ファクトリーが指定されていない場合に使用されるデフォルトの接続ファクトリーを設定します。		ConnectionFactory
disableReplyTo (common)	Camel がメッセージの JMSReplyTo ヘッダーを無視するかどうかを指定します。true の場合、Camel は JMSReplyTo ヘッダーで指定された宛先に返信を送り返しません。Camel にルートから消費させたいが、コード内の別のコンポーネントが応答メッセージを処理するため、Camel に自動的に応答メッセージを送り返したくない場合は、このオプションを使用できます。Camel を異なるメッセージブローカー間のプロキシーとして使用し、あるシステムから別のシステムにメッセージをルーティングする場合にも、このオプションを使用できます。	false	boolean
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		String
jmsMessageType (common)	JMS メッセージの送信に特定の javax.jms.Message 実装を強制的に使用できるようにします。可能な値は、Bytes、Map、Object、Stream、Text です。デフォルトでは、Camel は In body タイプから使用する JMS メッセージタイプを決定します。このオプションで指定できます。		JmsMessageType
testConnectionOnStartup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel の起動時に、すべての JMS コンシューマーが JMS ブローカーへの有効な接続を持つことが保証されます。接続を許可できない場合、Camel は起動時に例外を出力します。これにより、接続に失敗した状態で Camel が開始されなくなります。JMS プロデューサーもテストされています。	false	boolean
acknowledgementModeName (consumer)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	String

名前	説明	デフォルト	タイプ
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得できますが、前のメッセージは (非同期ルーティングエンジンによって) 非同期に処理されます。これは、メッセージが 100% 厳密に順序どおりに処理されない可能性があることを意味します。無効になっている場合 (デフォルト)、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。transactioned が有効になっている場合、トランザクションは同期的に実行する必要があるため、asyncConsumer=true は非同期的に実行されないことに注意してください (Camel 3.0 は非同期トランザクションをサポートする場合があります)。	false	boolean
autoStartup (consumer)	コンシューマーコンテナーを自動起動するかどうかを指定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cacheLevel (consumer)	基礎となる JMS リソースの ID によってキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	基礎となる JMS リソースのキャッシュレベルを名前で設定します。可能な値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、および CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring のドキュメントとトランザクションキャッシュレベルを参照してください。	CACHE_AUTO	String

名前	説明	デフォルト	タイプ
concurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーのデフォルト数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。	1	int
maxConcurrentConsumers (consumer)	JMS から消費する場合の同時コンシューマーの最大数を指定します (JMS を介した要求/応答ではありません)。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。JMS を介して要求/応答を行う場合は、オプション replyToMaxConcurrentConsumers を使用して、応答メッセージリスナーの同時コンシューマーの数を制御します。		int
replyTo (consumer)	Message.getJMSReplyTo() の着信値をオーバーライドする明示的な ReplyTo 宛先を提供します。		String
replyToDeliveryPersistent (consumer)	返信に対してデフォルトで永続的な配信を使用するかどうかを指定します。	true	boolean
selector (consumer)	使用する JMS セレクターを設定します。		String
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常、subscriptionName 値と組み合わせて永続的なサブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean

名前	説明	デフォルト	タイプ
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有または永続的なサブスクリプションを持つトピック (pub-sub ドメイン) の場合に適用されます。サブスクリプション名は、このクライアントの JMS クライアント ID 内で一意である必要があります。デフォルトは、指定されたメッセージリスナーのクラス名です。注: 共有サブスクリプション (JMS 2.0 が必要) を除き、サブスクリプションごとに 1 つの同時コンシューマー (このメッセージリスナコンテナーのデフォルト) のみが許可されます。		String
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーで指定できます。デフォルトは false です。通常は subscriptionName 値と組み合わせて共有サブスクリプションを登録するには、これを true に設定します (メッセージリスナークラス名がサブスクリプション名として十分でない場合)。共有サブスクリプションも永続的である可能性があるため、このフラグを subscriptionDurable と組み合わせることもできます (多くの場合は組み合わせます)。トピック (pub-sub ドメイン) をリッスンする場合にのみ意味があるため、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 互換のメッセージブローカーが必要です。	false	boolean
acceptMessagesWhileStopping (consumer)	コンシューマーが停止中にメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止するが、キューにメッセージが入れられている場合は、このオプションを有効にすることを検討してください。このオプションが false の場合は、JMS ルートを停止すると、メッセージが拒否される可能性があり、JMS ブローカは再配信を試行する必要がありますが、これも拒否される可能性があり、最終的にメッセージはJMS ブローカー上のデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることをお勧めします。	false	boolean

名前	説明	デフォルト	タイプ
allowReplyManagerQuickStop (consumer)	リンク JmsConfigurationisAcceptMessagesWhileStopping() が有効で、org.apache.camel.CamelContext が現在停止されている場合に、リクエスト/リプライメッセージングのリプライマネージャーで使用される DefaultMessageListenerContainer が、リンク DefaultMessageListenerContainerrunningAllowed() フラグを迅速に停止できるようにするかどうか。このクイック停止機能は、通常の JMS コンシューマーではデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
consumerType (consumer)	使用するコンシューマータイプ。Simple、Default、または Custom のいずれかです。コンシューマータイプによって、使用する Spring JMS リスナーが決まります。デフォルトは org.springframework.jms.listener.DefaultMessageListenerContainer を使用し、Simple は org.springframework.jms.listener.SimpleMessageListenerContainer を使用します。Custom を指定した場合は、messageListenerContainerFactory オプションで定義された MessageListenerContainerFactory によって、使用する org.springframework.jms.listener.AbstractMessageListenerContainer が決まります。	デフォルト	ConsumerType
defaultTaskExecutorType (consumer)	コンシューマーエンドポイントとプロデューサエンドポイントの ReplyTo コンシューマーの両方に対して、DefaultMessageListenerContainer で使用するデフォルトの TaskExecutor タイプを指定します。可能な値: SimpleAsync (Spring の SimpleAsyncTaskExecutor を使用) または ThreadPool (Spring の ThreadPoolTaskExecutor を最適な値で使用 - キャッシュされたスレッドプールのようなもの)。設定されていない場合は、デフォルトで以前の動作になり、コンシューマーエンドポイントにはキャッシュされたスレッドプールが使用され、応答コンシューマーには SimpleAsync が使用されます。ThreadPool の使用は、同時コンシューマーが動的に増減するエラスティック設定でスレッドのゴミを減らすために推奨されます。		DefaultTaskExecutor Type
eagerLoadingOfProperties (consumer)	メッセージが読み込まれるとすぐに JMS プロパティーの先行読み込みを有効にします。これは、JMS プロパティーが必要ない場合があるため一般的に非効率的ですが、基盤となる JMS プロバイダーと JMS プロパティーの使用に関する問題を早期に発見できる場合があります。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
exposeListenerSession (consumer)	メッセージを消費するときにリスナーセッションを公開するかどうかを指定します。	false	boolean
replyToSameDestination Allowed (consumer)	JMS コンシューマーが、コンシューマーが使用しているのと同じ宛先に応答メッセージを送信できるかどうか。これにより、同じメッセージを消費してそれ自体に送り返すことで、無限ループが回避されます。	false	boolean
taskExecutor (consumer)	メッセージを消費するためのカスタムタスクエグゼキュータを指定できます。		TaskExecutor
deliveryMode (producer)	使用する配信モードを指定します。可能な値は、javax.jms.DeliveryMode によって定義された値です。NON_PERSISTENT = 1 および PERSISTENT = 2。		Integer
deliveryPersistent (producer)	デフォルトで永続配信を使用するかどうかを指定します。	true	boolean
explicitQosEnabled (producer)	メッセージの送信時に、deliveryMode、priority、または timeToLive のサービス品質を使用する必要があるかどうかを設定します。このオプションは、Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で動作し、Camel In メッセージヘッダーから排他的に QoS プロパティーを読み取る preserveMessageQos オプションとは対照的です。	false	Boolean
formatDateHeadersToIso8601 (producer)	日付ヘッダーを ISO 8601 標準に従ってフォーマットするかどうかを設定します。	false	boolean

名前	説明	デフォルト	タイプ
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定ではなく、メッセージで指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。次の 3 つのヘッダーは、JMSPriority、JMSDeliveryMode、および JMSExpiration と見なされます。それらのすべてまたは一部のみを指定できます。指定されていない場合、Camel は代わりにエンドポイントからの値を使用するようにフォールバックします。したがって、このオプションを使用すると、ヘッダーはエンドポイントからの値をオーバーライドします。対照的に、explicitQosEnabled オプションは、エンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値は使用しません。	false	boolean
priority (producer)	1 より大きい値は、送信時のメッセージの優先度を指定します (0 が最低の優先度で、9 が最高の優先度です)。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
replyToConcurrentConsumers (producer)	JMS を介して要求/応答を行うときの同時コンシューマーのデフォルト数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。	1	int
replyToMaxConcurrent Consumers (producer)	JMS を介した要求/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的なスケールアップ/ダウンを制御するには、maxMessagesPerTask オプションも参照してください。		int
replyToOnTimeoutMax ConcurrentConsumers (producer)	JMS 経由の要求/応答を使用するときにタイムアウトが発生したときに、ルーティングを継続するための同時コンシューマーの最大数を指定します。	1	int
replyToOverride (producer)	JMS メッセージで明示的な ReplyTo 宛先を提供します。これは、replyTo の設定をオーバーライドします。メッセージをリモート Queue に転送し、ReplyTo 宛先から応答メッセージを受け取る場合に便利です。		String

名前	説明	デフォルト	タイプ
replyToType (producer)	JMS を介して要求/応答を行うときに、replyTo キューに使用する戦略の種類を明示的に指定できます。可能な値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、replyTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに専用キューを使用できます。詳細については、Camel JMS のドキュメントを参照してください。特に、クラスター化された環境で実行する場合の影響に関する注意事項と、共有応答キューは代替の一時および排他的キューよりもパフォーマンスが低いという事実を参照してください。		ReplyToType
requestTimeout (producer)	InOut Exchange パターン使用時の応答待ちタイムアウト (ミリ秒単位)。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めて、このエンドポイントで設定されたタイムアウト値をオーバーライドし、メッセージごとに個別のタイムアウト値を持つことができます。requestTimeoutCheckerInterval オプションも参照してください。	20000	long
timeToLive (producer)	メッセージの送信時に、メッセージの有効期限をミリ秒単位で指定します。	-1	long
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値を持つ可能性がある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型の値を含む接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名でこれを行います。コンマで区切られた複数のヘッダー名を指定し、ワイルドカードマッチングの接尾辞として使用できます。		String
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
alwaysCopyMessage (producer)	true の場合、メッセージがプロデューサーに渡されて送信されると、Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、状況によってはメッセージのコピーが必要です (また、replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します)。	false	boolean

名前	説明	デフォルト	タイプ
correlationProperty (producer)	JMSCorrelationID プロパティーの代わりに、この JMS プロパティーを使用して、InOut 交換パターン (要求 - 応答) でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用してメッセージと相関性のないシステムとメッセージを交換できます。JMSCorrelationID を使用すると、Camel によって使用または設定されません。ここで指定されたプロパティーの値は、同じ名前でメッセージのヘッダーに指定されていない場合に生成されます。		String
disableTimeToLive (producer)	このオプションを使用して、有効期限を強制的に無効にします。たとえば、JMS を介して要求/応答を行う場合、Camel はデフォルトで、送信されるメッセージの存続時間として requestTimeout 値を使用します。問題は、送信側システムと受信側システムのクロックを同期させる必要があるため、同期していることです。これをアーカイブするのは必ずしも簡単ではありません。したがって、disableTimeToLive=true を使用して、送信されたメッセージに有効期限の値を設定しないようにすることができます。その後、メッセージは受信側システムで期限切れになりません。詳細については、以下の生存時間についてのセクションを参照してください。	false	boolean
forceSendOriginalMessage (producer)	mapJmsMessage=false を使用すると、ルート中にヘッダーに触れると (get または set)、Camel は新しい JMS メッセージを作成して新しい JMS 宛先に送信します。Camel が受信した元の JMS メッセージを強制的に送信するには、このオプションを true に設定します。	false	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます (例: ファイアアンドフォーゲット)。このオプションを有効にすると、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用された実際の JMSMessageID で Camel Exchange が強化されます。	false	boolean

名前	説明	デフォルト	タイプ
replyToCacheLevelName (producer)	JMS を介して要求/応答を行うときに、応答コンシューマーのキャッシュレベルを名前で設定します。このオプションは、固定応答キュー (一時的ではない) を使用する場合にのみ適用されます。Camel はデフォルトで次を使用します: 排他的または replyToSelectorName と共有の CACHE_CONSUMER。そして、replyToSelectorName なしで共有するための CACHE_SESSION。IBM WebSphere などの一部の JMS ブローカーは、replyToCacheLevelName=CACHE_NONE を機能させるために設定する必要がある場合があります。注: 一時キューを使用する場合、CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などのより高い値を使用する必要があります。		String
replyToDestinationSelector Name (producer)	使用する固定名を使用して JMS セレクターを設定し、共有キューを使用している場合 (つまり、一時的な応答キューを使用していない場合) に、他の応答から自分の応答を除外できるようにします。		String
streamMessageTypeEnabled (producer)	StreamMessage タイプを有効にするかどうかを設定します。ファイル、InputStream などのストリーミングの種類のメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、どの種類が使用されるかを制御します。デフォルトでは、BytesMessage が使用され、メッセージペイロード全体がメモリーに読み込まれます。このオプションを有効にすると、メッセージペイロードがチャンク単位でメモリーに読み込まれ、データがなくなるまで各チャンクが StreamMessage に書き込まれます。	false	boolean
allowSerializedHeaders (advanced)	シリアル化されたヘッダーを含めるかどうかを制御します。リンク isTransferExchange() が true の場合にのみ適用されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean

名前	説明	デフォルト	タイプ
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
destinationResolver (advanced)	独自のリゾルバーを使用できるようにするプラグ可能な org.springframework.jms.support.destination.DestinationResolver (たとえば、JNDI レジストリーで実際の宛先を検索するため)。		DestinationResolver
errorHandler (advanced)	Message の処理中にキャッチされない例外が出力された場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ログレベルとスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングするよりも設定がはるかに簡単になります。		ErrorHandler
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
idleConsumerLimit (advanced)	常にアイドル状態にできるコンシューマーの数の制限を指定します。	1	int

名前	説明	デフォルト	タイプ
idleTaskExecutionLimit (advanced)	実行中にメッセージを受信していない、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスクに受信を任せます (動的スケジューリングの場合。maxConcurrentConsumers 設定を参照してください)。Spring から入手できる追加のドキュメントがあります。	1	int
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		文字列
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。	true	boolean
maxMessagesPerTask (advanced)	タスクあたりのメッセージ数。-1 は無制限です。同時コンシューマーの範囲 (例: min max) を使用する場合、このオプションを使用して値を 100 などに設定し、必要な作業が少ない場合にコンシューマーが縮小する速度を制御できます。	-1	int
messageConverter (advanced)	カスタム Spring org.springframework.jms.support.converter.MessageConverter を使用して、javax.jms.Message との間でどのようにマッピングするかを制御できるようにします。		MessageConverter
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy

名前	説明	デフォルト	タイプ
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカーへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージのメッセージ ID を null に設定する必要があります。プロバイダーがヒントを無視する場合、メッセージ ID は通常の一意の値に設定する必要があります	true	boolean
messageListenerContainerFactory (advanced)	メッセージを消費するために使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定するために使用される MessageListenerContainerFactory のレジストリー ID。これを設定すると、consumerType が自動的に Custom に設定されます。		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	メッセージの送信時にデフォルトでタイムスタンプを有効にするかどうかを指定します。これは、JMS ブローカーへの単なるヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージのタイムスタンプをゼロに設定する必要があります。プロバイダーがヒントを無視する場合は、タイムスタンプを通常の値に設定する必要があります	true	boolean
pubSubNoLocal (advanced)	独自の接続によってパブリッシュされたメッセージの配信を禁止するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージ受信のタイムアウト (ミリ秒単位)。	1000	long
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	long
requestTimeoutCheckerInterval (advanced)	JMS を介してリクエスト/リプライを行うときに、Camel がタイムアウトになった Exchange をチェックする頻度を設定します。デフォルトでは、Camel は 1 秒に 1 回確認します。ただし、タイムアウトが発生したときに迅速に対応する必要がある場合は、この間隔を短くして、より頻繁にチェックすることができます。タイムアウトは、オプション requestTimeout によって決定されます。	1000	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
transferException (advanced)	有効で、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で失敗した場合、原因となった例外が javax.jms.ObjectMessage として応答で返されます。クライアントが Camel の場合、返された Exception は再出力されます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続的なキューを使用して堅牢なルーティングを有効にできます。transferExchange も有効にしている場合は、このオプションが優先されることに注意してください。キャッチされた例外はシリアライズ可能である必要があります。コンシューマー側の元の Exception は、プロデューサーに返されるときに org.apache.camel.RuntimeCamelException などの外部例外にラップできます。	false	boolean
transferExchange (advanced)	本文とヘッダーだけでなく、電信送金で交換を転送できます。次のフィールドが転送されます: In body、Out body、Fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、交換プロパティー、交換例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。プロデューサ側とコンシューマー側の両方でこのオプションを有効にする必要があるため、Camel はペイロードが Exchange であり、通常のペイロードではないことを認識します。	false	boolean
transferFault (advanced)	これを有効にし、Request Reply メッセージング (InOut) を使用していて、Exchange がコンシューマー側で SOAP エラー (例外ではない) で失敗した場合には、リンク org.apache.camel.MessageisFault() のエラーフラグがレスポンスで、リンク JmsConstantsJMS_TRANSFER_FAULT のキーを含んだ JMS ヘッダーとして送り返されます。クライアントが Camel の場合には、返される障害フラグはリンク org.apache.camel.MessagesetFault(boolean) に設定されます。cxf や spring-ws などの SOAP ベースなどの障害をサポートする Camel コンポーネントを使用する場合、これを有効できます。	false	boolean
useMessageIDAsCorrelation ID (advanced)	InOut メッセージの JMSCorrelationID として JMSMessageID を常に使用するかどうかを指定します。	false	boolean
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	JMS を介して要求/応答を行う場合、およびオプション useMessageIDAsCorrelationID が有効な場合に、暫定相関 ID が実際の相関 ID に更新されるのを待機する回数。	50	int

名前	説明	デフォルト	タイプ
waitForProvisionCorrelationToBeUpdatedThreadSleeping Time (advanced)	暫定相関 ID が更新されるのを待機するたびにスリープする間隔 (ミリ単位)。	100	long
errorHandlerLoggingLevel (logging)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean
password (security)	ConnectionFactory で使用するパスワード。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
username (security)	ConnectionFactory で使用するユーザー名。また、ConnectionFactory でユーザー名およびパスワードを直接設定することもできます。		String
取引済み (取引)	トランザクションモードを使用するかどうかを指定します	false	boolean
lazyCreateTransaction Manager (トランザクション)	true の場合、オプション transacted=true のときに transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (トランザクション)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (トランザクション)	使用するトランザクションの名前。		String
transactionTimeout (トランザクション)	トランザクションモードを使用している場合の、トランザクションのタイムアウト値 (秒単位)。	-1	int

169.5. JMS と CAMEL 間のメッセージマッピング

Camel は **javax.jms.Message** と **org.apache.camel.Message** の間でメッセージを自動的にマップします。

JMS メッセージを送信するとき、Camel はメッセージ本文を次の JMS メッセージタイプに変換します。

ボディタイプ	JMS Message	Comment
String	**javax.jms.TextMessage**	
org.w3c.dom.Node	**javax.jms.TextMessage**	DOM は **String** に変換されます。
Map	**javax.jms.MapMessage**	
java.io.Serializable	**javax.jms.ObjectMessage**	
byte[]	**javax.jms.BytesMessage**	
java.io.File	**javax.jms.BytesMessage**	
java.io.Reader	**javax.jms.BytesMessage**	
java.io.InputStream	**javax.jms.BytesMessage**	
java.nio.ByteBuffer	**javax.jms.BytesMessage**	

JMS メッセージを受信すると、Camel は JMS メッセージを次のボディタイプに変換します。

JMS Message	ボディタイプ
javax.jms.TextMessage	**String**
javax.jms.BytesMessage	**byte[]**
javax.jms.MapMessage	**Map<String, Object>**
javax.jms.ObjectMessage	**Object**

169.5.1. JMS メッセージの自動マッピングの無効化

mapJmsMessage オプションを使用して、上記の自動マッピングを無効にすることができます。無効にすると、Camel は受信した JMS メッセージをマップしようとせず、ペイロードとして直接使用します。これにより、マッピングのオーバーヘッドを回避し、Camel に JMS メッセージを通過させることができます。たとえば、クラスパスに **ない** クラスを使用して **javax.jms.ObjectMessage** JMS メッセージをルーティングすることもできます。

169.5.2. カスタム MessageConverter の使用

messageConverter オプションを使用して、Spring **org.springframework.jms.support.converter.MessageConverter** クラスで自分でマッピングを行うことができます。

たとえば、以下のルートでは、メッセージを JMS オーダーキューに送信するときにカスタムメッセージコンバーターを使用します。

```
from("file://inbox/order").to("jms:queue:order?messageConverter=#myMessageConverter");
```

JMS 宛先から消費する場合は、カスタムメッセージコンバーターを使用することもできます。

169.5.3. 選択したマッピング戦略の制御

エンドポイント URL で **jmsMessageType** オプションを使用して、すべてのメッセージに対して特定のメッセージタイプを強制することができます。

以下のルートでは、フォルダーからファイルをポーリングし、それらを **javax.jms.TextMessage** として送信します。これは、JMS プロデューサーエンドポイントにテキストメッセージの使用を強制したためです。

```
from("file://inbox/order").to("jms:queue:order?jmsMessageType=Text");
```

キー **CamelJmsMessageType** でヘッダーを設定することにより、各メッセージに使用するメッセージタイプを指定することもできます。以下に例を示します。

```
from("file://inbox/order").setHeader("CamelJmsMessageType",
JmsMessageType.Text).to("jms:queue:order");
```

可能な値は **enum** 型クラス **org.apache.camel.jms.JmsMessageType** で定義されています。

169.6. 送信時のメッセージ形式

JMS ワイヤを介して送信される交換は、JMS メッセージ仕様 に準拠する必要があります。

exchange.in.header の場合、次のルールがヘッダー **キー** に適用されます。

- **JMS** または **JMSX** で始まるキーは予約されています。
- **exchange.in.headers** キーはリテラルで、すべて有効な Java 識別子である必要があります (キー名にドットを使用しないでください)。
- Camel は、JMS メッセージを消費するときにドットとハイフンを置き換え、その逆を行います。
  . は **DOT** に置き換えられ、Camel がメッセージを消費するときは逆の置き換えになります。
  - は **HYPHEN** に置き換えられ、Camel がメッセージを消費するときは逆の置き換えになります。
- オプション **jmsKeyFormatStrategy** も参照してください。これにより、キーのフォーマットに独自のカスタム戦略を使用できます。

exchange.in.header の場合、次のルールがヘッダー **値** に適用されます。

- 値は、プリミティブまたはそのカウンターオブジェクト (**Integer**、**Long**、**Character** など) である必要があります。タイプ **String**、**CharSequence**、**Date**、**BigDecimal**、および **BigInteger** はすべて、それらの **toString()** 表現に変換されます。他のすべてのタイプはドロップされます。

Camel は、特定のヘッダー値を削除すると、カテゴリー **org.apache.camel.component.jms.JmsBinding** で **DEBUG** レベルでログに記録します。以下に例を示します。

```
2008-07-09 06:43:04,046 [main           ] DEBUG JmsBinding
  - Ignoring non primitive header: order of class:
org.apache.camel.component.jms.issues.DummyOrder with value: DummyOrder{orderId=333,
itemId=4444, quantity=2}
```

169.7. 受信時のメッセージフォーマット

Camel は、メッセージを受信すると、次のプロパティーを **Exchange** に追加します。

プロパティー	タイプ	説明
org.apache.camel.jms.replyDestination	**javax.jms.Destination**	返信先。

Camel は、JMS メッセージを受信すると、In メッセージヘッダーに次の JMS プロパティーを追加します。

ヘッダー	タイプ	説明
JMSCorrelationID	**String**	JMS 相関 ID。
JMSDeliveryMode	**int**	JMS 配信モード。
JMSDestination	**javax.jms.Destination**	JMS 宛先。
JMSExpiration	**long**	JMS の有効期限。
JMSMessageID	**String**	JMS 固有のメッセージ ID。
JMSPriority	**int**	JMS 優先度 (0 が最低の優先度、9 が最高の優先度)。
JMSRedelivered	**boolean**	JMS メッセージは再配信されますか。
JMSReplyTo	**javax.jms.Destination**	JMS 返信先の宛先。
JMSTimestamp	**long**	JMS タイムスタンプ。
JMSType	**String**	JMS タイプ。
JMSXGroupID	**String**	JMS グループ ID。

上記の情報はすべて標準の JMS であるため、詳細については JMS のドキュメント を参照してください。

169.8. CAMEL を使用したメッセージの送受信と JMSREPLYTO について

JMS コンポーネントは複雑で、場合によってはその動作に細心の注意を払う必要があります。したがって、これは探すべき領域/落とし穴のいくつかの簡単な要約です.

Camel が **JMSProducer** を使用してメッセージを送信すると、次の条件がチェックされます。

- メッセージ交換パターン、
- **JMSReplyTo** がエンドポイントまたはメッセージヘッダーで設定されたかどうか、
- 次のいずれかのオプションが JMS エンドポイントに設定されているかどうか: **disableReplyTo**、**preserveMessageQos**、**explicitQosEnabled**。

これらすべてを理解し、ユースケースをサポートするように設定するには、少し複雑になる可能性があります。

169.8.1. JmsProducer

JmsProducer は、設定に応じて次のように動作します。

交換パターン	その他のオプション	説明
InOut	-	Camel は応答を期待し、一時的な **JMSReplyTo** を設定し、メッセージを送信した後、一時キューで応答メッセージのリッスンを開始します。
InOut	**JMSReplyTo** が設定されている	Camel は応答を期待し、メッセージの送信後、指定された **JMSReplyTo** キューで応答メッセージのリッスンを開始します。
InOnly	-	Camel はメッセージを送信しますが、返信は期待 **しません**。
InOnly	**JMSReplyTo** が設定されている	デフォルトでは、Camel は **JMSReplyTo** 宛先を破棄し、メッセージを送信する前に **JMSReplyTo** ヘッダーをクリアします。その後、Camel はメッセージを送信し、応答を期待 **しません**。Camel はこれを **WARN** レベルでログに記録します (**Camel 2.6** 以降では **DEBUG** レベルに変更されました。**preserveMessageQuo=true** を使用して、Camel に **JMSReplyTo** を保持するように指示できます。すべての状況で、**JmsProducer** は応答を期待 **しない** ため、メッセージの送信後に続行します。

169.8.2. JmsConsumer

JmsConsumer は、設定に応じて次のように動作します。

交換パターン	その他のオプション	説明
InOut	-	Camel は応答を **JMSReplyTo** キューに送り返します。
InOnly	-	パターンが **InOnly** であるため、Camel は返信を返しません。
-	**disableReplyTo=true**	このオプションは、返信を抑制します。

そのため、取引所に設定されているメッセージ交換パターンに注意してください。

ルートの途中で JMS 宛先にメッセージを送信する場合は、使用する交換パターンを指定できます。詳細については、Request Reply を参照してください。
これは、**InOnly** メッセージを JMS トピックに送信する場合に便利です。

```
from("activemq:queue:in")
  .to("bean:validateOrder")
  .to(ExchangePattern.InOnly, "activemq:topic:order")
  .to("bean:handleOrder");
```

169.9. エンドポイントを再利用し、実行時に計算されたさまざまな宛先に送信します

多くの異なる JMS 宛先にメッセージを送信する必要がある場合は、JMS エンドポイントを再利用して、メッセージヘッダーで実際の宛先を指定するのが理にかなっています。これにより、Camel は同じエンドポイントを再利用できますが、異なる宛先に送信できます。これにより、作成されるエンドポイントの数が大幅に削減され、メモリーとスレッドリソースが節約されます。

次のヘッダーで宛先を指定できます。

ヘッダー	タイプ	説明
CamelJmsDestination	**javax.jms.Destination**	宛先オブジェクト。
CamelJmsDestinationName	**String**	宛先名。

たとえば、次のルートは、実行時に宛先を計算し、それを使用して JMS URL に表示される宛先をオーバーライドする方法を示しています。

```
from("file://inbox")
  .to("bean:computeDestination")
  .to("activemq:queue:dummy");
```

キュー名の **dummy** は単なるプレースホルダーです。JMS エンドポイント URL の一部として指定する必要がありますが、この例では無視されます。

computeDestination bean で、**CamelJmsDestinationName** ヘッダーを次のように設定して、実際の宛先を指定します。

```
public void setJmsHeader(Exchange exchange) {
   String id = ....
   exchange.getIn().setHeader("CamelJmsDestinationName", "order:" + id");
}
```

次に、Camel はこのヘッダーを読み取り、エンドポイントで設定されたものの代わりに宛先として使用します。したがって、この例では、**id** 値が 2 であると仮定して、Camel はメッセージを **activemq:queue:order:2** に送信します。

CamelJmsDestination ヘッダーと **CamelJmsDestinationName** ヘッダーの両方が設定されている場合、**CamelJmsDestination** が優先されます。JMS プロデューサーは、**CamelJmsDestination** ヘッダーと **CamelJmsDestinationName** ヘッダーの両方を交換から削除し、作成された JMS メッセージに伝播しないことに注意してください。これは、ルートでの偶発的なループを回避するためです (メッセージが別の JMS エンドポイントに転送されるシナリオで)。

169.10. 異なる JMS プロバイダーの設定

次のように、Spring XML で JMS プロバイダーを設定できます。

基本的に、必要な数の JMS コンポーネントインスタンスを設定し、**id 属性 を使用して一意の名前** を付けることができます。前の例では、**activemq** コンポーネントを設定しています。同じことを行って、MQSeries、TibCo、BEA、Sonic などを設定できます。

名前付き JMS コンポーネントを作成したら、URI を使用してそのコンポーネント内のエンドポイントを参照できます。たとえば、コンポーネント名 **activemq** の場合、URI 形式 **activemq:queue:|topic:destinationName** を使用して宛先を参照できます。他のすべての JMS プロバイダーに対して同じアプローチを使用できます。

これは、SpringCamelContext がエンドポイント URI に使用するスキーム名のスプリングコンテキストからコンポーネントを遅延フェッチし、コンポーネントにエンドポイント URI を解決させることによって機能します。

169.10.1. JNDI を使用して ConnectionFactory を見つける

J2EE コンテナーを使用している場合は、Spring で通常の **<bean>** メカニズムを使用するのではなく、JNDI を検索して JMS **ConnectionFactory** を見つける必要がある場合があります。これは、Spring のファクトリー bean または新しい Spring XML 名前空間を使用して行うことができます。以下に例を示します。

```
<bean id="weblogic" class="org.apache.camel.component.jms.JmsComponent">
  <property name="connectionFactory" ref="myConnectionFactory"/>
</bean>

<jee:jndi-lookup id="myConnectionFactory" jndi-name="jms/connectionFactory"/>
```

JNDI ルックアップの詳細については、Spring リファレンスドキュメントの jee スキーマ を参照してください。

169.11. 同時消費

JMS の一般的な要件は、アプリケーションの応答性を高めるために、複数のスレッドで同時にメッセージを消費することです。次のように、**concurrentConsumers** オプションを設定して、JMS エンドポイントにサービスを提供するスレッドの数を指定できます。

```
from("jms:SomeQueue?concurrentConsumers=20").
  bean(MyClass.class);
```

このオプションは、次のいずれかの方法で設定できます。

- **JmsComponent** で、
- エンドポイント URI または、
- **JmsEndpoint** で直接 **setConcurrentConsumers ()** を呼び出す。

169.11.1. 非同期コンシューマーによる同時消費

各同時コンシューマーは、現在のメッセージが完全に処理されたときに、JMS ブローカーから次に利用可能なメッセージのみを取得することに注意してください。オプション **asyncConsumer=true** を設定すると、前のメッセージが (非同期ルーティングエンジンによって) 非同期に処理されている間に、コンシューマーが JMS キューから次のメッセージをピックアップできるようになります。**asyncConsumer** オプションの詳細については、ページ上部の表を参照してください。

```
from("jms:SomeQueue?concurrentConsumers=20&asyncConsumer=true").
  bean(MyClass.class);
```

169.12. JMS を介したリクエスト/リプライ

Camel は JMS 経由の Request Reply をサポートしています。本質的に、メッセージを JMS キューに送信する場合、Exchange の MEP は **InOut** である必要があります。

Camel は、パフォーマンスとクラスター化された環境に影響を与える JMS を介した要求/応答を設定するための多くのオプションを提供します。次の表は、オプションをまとめたものです。

オプション	パフォーマンス	Cluster	説明
Temporary	高速	はい	一時キューは応答キューとして使用され、Camel によって自動作成されます。これを使用するには、replyTo キュー名を指定 **しないでください**。オプションで、**replyToType=Temporary** を設定して、一時キューが使用されていることを目立たせることもできます。

オプション	パフォーマンス	Cluster	説明
共有	Slow	はい	共有永続キューが応答キューとして使用されます。キューは事前に作成する必要がありますが、Apache ActiveMQ などの一部のブローカーはオンザフライで作成できます。これを使用するには、replyTo キュー名を指定する必要があります。オプションで、**replyToType=Shared** を設定して、共有キューが使用されていることを目立たせることもできます。共有キューは、複数のノードがこの Camel アプリケーションを同時に実行しているクラスター環境で使用できます。すべてが同じ共有応答キューを使用しています。これが可能なのは、予期される応答メッセージを関連付けるために JMS メッセージセレクターが使用されているためです。ただし、これはパフォーマンスに影響します。JMS メッセージセレクターは低速であるため、**Temporary** キューや **Exclusive** キューほど高速ではありません。パフォーマンスを向上させるためにこれを微調整する方法については、以下を参照してください。
排他的	高速	No (*Yes)	応答キューとして専用の永続キューが使用されます。キューは事前に作成する必要がありますが、Apache ActiveMQ などの一部のブローカーはオンザフライで作成できます。これを使用するには、replyTo キュー名を指定する必要があります。また、**replyToType=Exclusive** を設定して Camel に排他的キューを使用するように指示する **必要** があります。**replyTo** キュー名が設定されている場合はデフォルトで **Shared** が使用されるためです。排他的応答キューを使用する場合、JMS メッセージセレクターは使用され **ない** ため、他のアプリケーションもこのキューを使用してはなりません。複数のノードがこの Camel アプリケーションを同時に実行しているクラスター環境では、専用キューを使用 **できません**。応答キューが要求メッセージを送信したのと同じノードに戻ってきた場合、制御できないためです。これが、共有キューが JMS メッセージセレクターを使用してこれを確認する理由です。**ただし**、各 Exclusive 応答キューをノードごとに一意の名前で設定すると、クラスター化された環境でこれを実行できます。その後、応答メッセージは、応答メッセージを待機する特定のノードのキューに送り返されます。
concurrentConsumers	高速	○	Camel 2.10.3: 使用中の同時メッセージリスナーを使用して、応答メッセージを同時に処理できます。範囲は、**concurrentConsumers** および **maxConcurrentConsumers** オプションを使用して指定できます。**注意: Shared** 応答キューを使用すると、同時リスナーではうまく機能しない可能性があるため、このオプションは注意して使用してください。
maxConcurrentConsumers	高速	○	Camel 2.10.3: 使用中の同時メッセージリスナーを使用して、応答メッセージを同時に処理できます。範囲は、**concurrentConsumers** および **maxConcurrentConsumers** オプションを使用して指定できます。**注意: Shared** 応答キューを使用すると、同時リスナーではうまく機能しない可能性があるため、このオプションは注意して使用してください。

JmsProducer は **InOut** を検出し、使用する返信先を含む **JMSReplyTo** ヘッダーを提供します。デフォルトでは、Camel は一時キューを使用しますが、エンドポイントで **replyTo** オプションを使用して、固定応答キューを指定できます (固定応答キューについては以下を参照してください)。

Camel は応答キューをリッスンするコンシューマーを自動的にセットアップするので、何もする必要はあり **ません。**

このコンシューマーは、返信をリッスンする Spring **DefaultMessageListenerContainer** です。ただし、同時コンシューマーは 1 つに固定されています。
つまり、返信を処理するスレッドは 1 つしかないため、返信は順番に処理されます。返信をより速く処理したい場合は、同時実行を使用する必要があります。ただし、**concurrentConsumer** オプションは使用 **しません**。以下のルートに示すように、代わりに Camel DSL の **threads** を使用する必要があります。

Camel 2.10.3 以降を使用している場合は、スレッドを使用する代わりに、concurrentConsumers オプションを使用します。以下を参照してください。

```
from(xxx)
.inOut().to("activemq:queue:foo")
.threads(5)
.to(yyy)
.to(zzz);
```

このルートでは、5 つのスレッドを持つスレッドプールを使用して応答を非同期にルーティングするように Camel に指示します。

Camel 2.10.3 以降では、**concurrentConsumers** および **maxConcurrentConsumers** オプションを使用して、同時スレッドを使用するようにリスナーを設定できるようになりました。これにより、以下に示すように Camel でこれを簡単に設定できます。

```
from(xxx)
.inOut().to("activemq:queue:foo?concurrentConsumers=5")
.to(yyy)
.to(zzz);
```

169.12.1. JMS を介したリクエスト/リプライと共有固定リプライキューの使用

以下の例に示すように、JMS を介して Request Reply を実行するときに固定のリプライキューを使用する場合は、注意してください。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar")
.to(yyy)
```

この例では、bar という名前の固定応答キューが使用されています。デフォルトでは、Camel は、固定応答キューを使用する場合にキューが共有されていると想定するため、**JMSSelector** を使用して、予想される応答メッセージのみをピックアップします (たとえば、**JMSCorrelationID** に基づいて)。排他的な固定応答キューについては、次のセクションを参照してください。つまり、一時キューほど高速ではありません。**receiveTimeout** オプションを使用して、Camel が応答メッセージをプルする頻度を高速化できます。デフォルトでは 1000 ミリ秒です。したがって、より高速にするために、次のように 250 ミリ秒に設定して、1 秒あたり 4 回プルすることができます。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&receiveTimeout=250")
.to(yyy)
```

これにより、Camel がプルリクエストをメッセージブローカーにより頻繁に送信するようになり、より多くのネットワークトラフィックが必要になることに注意してください。
一般に、可能であれば一時キューを使用することをお勧めします。

169.12.2. JMS を介したリクエストとリプライ、および専用の固定リプライキューを使用

Camel 2.9 以降で利用可能

前の例では、Camel は bar という名前の固定応答キューが共有されていると予想し、**JMSSelector** を使用して、期待する応答メッセージのみを消費します。ただし、JMS selectos は低速であるため、これを行うには欠点があります。また、応答キューのコンシューマーは、新しい JMS セレクター ID での更新が遅くなります。実際には、**receiveTimeout** オプションがタイムアウトしたときにのみ更新されます。デフォルトでは 1 秒です。したがって、理論的には、応答メッセージが検出されるまでに約 1 秒かかる可能性があります。一方、固定リプライキューが Camel リプライコンシューマー専用である場合は、JMS セレクターの使用を避けることができるため、パフォーマンスが向上します。実際、一時キューを使用するのと同じくらい高速です。そのため、**Camel 2.9** 以降では、**Exclusive** に設定できる **ReplyToType** オプションを導入しました。
以下の例に示すように、応答キューが排他的であることを Camel に伝えます。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)
```

キューはすべてのエンドポイントに対して排他的でなければならないことに注意してください。したがって、ルートが 2 つある場合は、次の例に示すように、それぞれに一意の応答キューが必要です。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)

from(aaa)
.inOut().to("activemq:queue:order?replyTo=order.reply&replyToType=Exclusive")
.to(bbb)
```

クラスター環境で実行する場合も同様です。次に、クラスター内の各ノードは一意の応答キュー名を使用する必要があります。そうしないと、クラスター内の各ノードが、別のノードでの応答として意図されたメッセージを取得する可能性があります。クラスター化された環境では、代わりに共有応答キューを使用することをお勧めします。

169.13. 送信側と受信側のクロックの同期

システム間でメッセージングを行う場合、システムのクロックが同期していることが望ましいです。たとえば、JMS メッセージを送信する場合には、メッセージに Time to Live 値を設定できます。次に、受信者はこの値を検査し、メッセージがすでに期限切れになっているかどうかを判断し、メッセージを消費して処理する代わりにドロップします。ただし、これには、送信側と受信側の両方でクロックが同期されている必要があります。ActiveMQ を使用している場合は、タイムスタンププラグイン を使用してクロックを同期できます。

169.14. 生きる時間について

上記の同期クロックについて最初に読んでください。

Camel を使用して JMS 経由で要求/応答 (InOut) を行う場合に、Camel は送信側でタイムアウトを使用します。これは、**requestTimeout** オプションのデフォルトの 20 秒です。これは、より高い/より低い値を設定することで制御できます。ただし、送信される JMS メッセージには存続時間の値が設定されたままです。そのため、システム間でクロックを同期する必要があります。そうでない場合は、設定されている存続時間の値を無効にすることができます。これは、**Camel 2.8** 以降の **disableTimeToLive**

オプションを使用して可能になりました。したがって、このオプションを **disableTimeToLive=true** に設定すると、Camel では JMS メッセージの送信時に time to live の値は設定 **されません。ただし**、リクエストのタイムアウトはまだ有効です。たとえば、JMS を介してリクエスト/リプライを行い、time to live を無効にしている場合に、Camel は引き続き 20 秒のタイムアウトを使用します (**requestTimeout** オプション)。もちろん、そのオプションも設定できます。そのため、**requestTimeout** と **disableTimeToLive** の 2 つのオプションを使用すると、リクエスト/リプライを行うときにきめ細かい制御が可能になります。

Camel 2.13/2.12.3 以降では、メッセージにヘッダーを指定してオーバーライドし、エンドポイントの設定値の代わりにリクエストのタイムアウト値として使用できます。以下に例を示します。

```
from("direct:someWhere")
  .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
  .to("bean:processReply");
```

上記のルートでは、**requestTimeout** が 30 秒に設定されたエンドポイントがあります。そのため、Camel は、その応答メッセージがバーキューに戻ってくるまで最大 30 秒待機します。応答メッセージが受信されない場合、Exchange で **org.apache.camel.ExchangeTimedOutException** が設定され、Camel はメッセージのルーティングを続行しますが、例外が原因で失敗し、Camel のエラーハンドラーが反応します。

メッセージごとのタイムアウト値を使用する場合は、長型としてタイムアウト値を持つ定数値 **"CamelJmsRequestTimeout"** を持つキー **org.apache.camel.component.jms.JmsConstants#JMS_REQUEST_TIMEOUT** でヘッダーを設定できます。

たとえば、bean を使用して、以下に示すようにサービス Bean で **"whatIsTheTimeout"** メソッドを呼び出すなど、個々のメッセージごとのタイムアウト値を計算できます。

```
from("direct:someWhere")
  .setHeader("CamelJmsRequestTimeout", method(ServiceBean.class, "whatIsTheTimeout"))
  .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
  .to("bean:processReply");
```

Camel を使用して JMS を介して fire および forget (InOut) を行う場合には、Camel はデフォルトで、メッセージに Time to Live 値を設定 **しません**。**timeToLive** オプションを使用して値を設定できます。たとえば、5 秒を示すには、**timeToLive=5000** を設定します。オプション **disableTimeToLive** を使用して、有効期限を強制的に無効にすることができます。また、InOnly メッセージングについても同様です。**requestTimeout** オプションは、InOnly メッセージングには使用されていません。

169.15. 取引消費の有効化

一般的な要件は、トランザクション内のキューから消費し、Camel ルートを使用してメッセージを処理することです。これを行うには、コンポーネント/エンドポイントで次のプロパティーを設定していることを確認してください。

- **transacted** = true
- **transactionManager** = **トランザクションマネージャー** - 通常は **JmsTransactionManager**

詳細については、Transactional Client EIP パターンを参照してください。

JMS を介したトランザクションと [Request Reply]

Request Reply over JMS を使用する場合、単一のトランザクションを使用することはできません。JMS

はコミットが実行されるまでメッセージを送信しないため、サーバー側はトランザクションがコミットされるまで何も受信しません。したがって、リクエストリプライ を使用するには、リクエストの送信後にトランザクションをコミットし、別のトランザクションを使用してレスポンスを受信する必要があります。

この問題に対処するために、JMS コンポーネントは異なるプロパティーを使用して、一方向メッセージングと要求応答メッセージングのトランザクションの使用を指定します。

transacted プロパティーは、InOnly メッセージ交換パターン (MEP) に **のみ** 適用されます。

transactedInOut プロパティーは、InOut (Request Reply) メッセージエクスチェンジパターン (MEP) に適用されます。

Request Reply (InOut MEP) にトランザクションを使用する場合は、**transactedInOut=true** を設定する **必要があります**。

Camel 2.10 以降で利用可能

コンポーネント/エンドポイントで次のプロパティーを使用して、DMLC トランザクションセッション API を利用できます。

- **transacted** = true
- **lazyCreateTransactionManager** = false

そうすることの利点は、設定された TransactionManager なしでローカルトランザクションを使用するときに、cacheLevel 設定が受け入れられることです。TransactionManager が設定されている場合、DMLC レベルでキャッシュは発生せず、プールされた接続ファクトリーに依存する必要があります。この種のセットアップの詳細については、こちら と こちら を参照してください。

169.16. 遅延応答に対する JMSREPLYTO の使用

Camel を JMS リスナーとして使用する場合、キー **ReplyTo** を持つ ReplyTo **javax.jms.Destination** オブジェクトの値で Exchange プロパティーを設定します。この **Destination** は次のように取得できます。

```
Destination replyDestination =
exchange.getIn().getHeader(JmsConstants.JMS_REPLY_DESTINATION, Destination.class);
```

そして後でそれを使用して、通常の JMS または Camel を使用して応答を送信します。

```
// we need to pass in the JMS component, and in this sample we use ActiveMQ
JmsEndpoint endpoint = JmsEndpoint.newInstance(replyDestination, activeMQComponent);
// now we have the endpoint we can use regular Camel API to send a message to it
template.sendBody(endpoint, "Here is the late reply.");
```

返信を送信する別の解決策は、送信時に同じ Exchange プロパティーに **replyDestination** オブジェクトを提供することです。Camel はこのプロパティーを取得し、実際の目的地に使用します。ただし、エンドポイント URI にはダミーの宛先を含める必要があります。以下に例を示します。

```
// we pretend to send it to some non existing dummy queue
template.send("activemq:queue:dummy, new Processor() {
    public void process(Exchange exchange) throws Exception {
        // and here we override the destination with the ReplyTo destination object so the message is
sent to there instead of dummy
```

```
            exchange.getIn().setHeader(JmsConstants.JMS_DESTINATION, replyDestination);
            exchange.getIn().setBody("Here is the late reply.");
        }
    }
```

169.17. リクエストタイムアウトの使用

以下のサンプルでは、Request Reply スタイルのメッセージ Exchange (**requestBody** メソッド = **InOut** を使用) を出力 キューに送信して、Camel でさらに処理し、返信を待ちます。

169.18. サンプル

JMS は、他のコンポーネントの多くの例でも使用されています。ただし、開始するためのいくつかのサンプルを以下に示します。

169.18.1. JMS からの受信

次のサンプルでは、JMS メッセージを受信し、メッセージを POJO にルーティングするルートを設定します。

```
from("jms:queue:foo").
  to("bean:myBusinessLogic");
```

もちろん、任意の EIP パターンを使用して、ルートをコンテキストベースにすることができます。たとえば、次のように、高額の支出者向けに注文トピックをフィルター処理します。

```
from("jms:topic:OrdersTopic").
  filter().method("myBean", "isGoldCustomer").
    to("jms:queue:BigSpendersQueue");
```

169.18.2. JMS への送信

以下のサンプルでは、ファイルフォルダーをポーリングし、ファイルコンテンツを JMS トピックに送信します。ファイルの内容を **BytesMessage** ではなく **TextMessage** にしたいので、本文を **String** に変換する必要があります。

```
from("file://orders").
  convertBodyTo(String.class).
  to("jms:topic:OrdersTopic");
```

169.18.3. アノテーションの使用

Camel にはアノテーションもあるため、POJO Consuming と POJO Producing を使用できます。

169.18.4. Spring の DSL サンプル

前の例では、Java DSL を使用しています。Camel は Spring XML DSL もサポートしています。以下は、Spring DSL を使用した高額支出者のサンプルです。

```
<route>
  <from uri="jms:topic:OrdersTopic"/>
```

```
  <filter>
    <method bean="myBean" method="isGoldCustomer"/>
    <to uri="jms:queue:BigSpendersQueue"/>
  </filter>
</route>
```

169.18.5. その他のサンプル

JMS は、この Camel ドキュメントだけでなく、他のコンポーネントや EIP パターンの例の多くにも登場します。そのため、ドキュメントを自由に参照してください。時間があれば、JMS を使用するこのチュートリアルをチェックしてください。ただし、Spring Remoting と Camel がどのように連携するかに焦点を当てています。Tutorial-JmsRemoting.

169.18.6. JMS を Exchange を格納するデッドレターキューとして使用する

通常、JMS をトランスポートとして使用する場合、ペイロードとしてボディーとヘッダーのみを転送します。Dead Letter Channel で JMS を使用する場合、JMS キューをデッドレターキューとして使用する場合、通常、発生した例外は JMS メッセージに格納されません。ただし、JMS デッドレターキューで **transferExchange** オプションを使用して、Exchange 全体を **org.apache.camel.impl.DefaultExchangeHolder** を保持する **javax.jms.ObjectMessage** としてキューに格納するよう Camel に指示できます。これにより、デッドレターキューから消費し、キー **Exchange.EXCEPTION_CAUGHT** を使用して Exchange プロパティーから原因の例外を取得できます。以下のデモは、これを示しています。

```
// setup error handler to use JMS as queue and store the entire Exchange
errorHandler(deadLetterChannel("jms:queue:dead?transferExchange=true"));
```

次に、JMS キューから消費して問題を分析できます。

```
from("jms:queue:dead").to("bean:myErrorAnalyzer");

// and in our bean
String body = exchange.getIn().getBody();
Exception cause = exchange.getProperty(Exchange.EXCEPTION_CAUGHT, Exception.class);
// the cause message is
String problem = cause.getMessage();
```

169.18.7. エラーのみを格納するデッドレターチャネルとして JMS を使用する

JMS を使用して、原因エラーメッセージを格納したり、自分で初期化できるカスタムボディーを格納したりできます。次の例では、Message Translator EIP を使用して、失敗した交換を JMS デッドレターキューに移動する前に変換を行います。

```
// we sent it to a seda dead queue first
errorHandler(deadLetterChannel("seda:dead"));

// and on the seda dead queue we can do the custom transformation before its sent to the JMS queue
from("seda:dead").transform(exceptionMessage()).to("jms:queue:dead");
```

ここでは、元の原因のエラーメッセージのみを変換に保存します。ただし、任意の式を使用して、好きなものを送信できます。たとえば、Bean でメソッドを呼び出したり、カスタムプロセッサーを使用したりできます。

169.19. INONLY メッセージを送信し、JMSREPLYTO ヘッダーを保持する

camel-jms を使用して JMS 宛先に送信する場合に、プロデューサーは MEP を使用して、その **InOnly** または **InOut** メッセージングを検出します。ただし、**InOnly** メッセージを送信したいが、**JMSReplyTo** ヘッダーを保持したい場合があります。そのためには、Camel にそれを保持するように指示する必要があります。そうしないと、**JMSReplyTo** ヘッダーがドロップされます。

たとえば、**InOnly** メッセージを foo キューに送信しますが、**JMSReplyTo** と bar キューを使用すると、次のように実行できます。

```
template.send("activemq:queue:foo?preserveMessageQos=true", new Processor() {
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setBody("World");
        exchange.getIn().setHeader("JMSReplyTo", "bar");
    }
});
```

preserveMessageQos=true を使用して Camel に **JMSReplyTo** ヘッダーを保持するように指示していることに注意してください。

169.20. 宛先での JMS プロバイダーオプションの設定

IBM の WebSphere MQ などの一部の JMS プロバイダーでは、JMS 宛先にオプションを設定する必要があります。たとえば、**targetClient** オプションを指定する必要がある場合があります。**targetClient** は Camel URI オプションではなく WebSphere MQ オプションであるため、次のように JMS 宛先名に設定する必要があります。

```
// ...
.setHeader("CamelJmsDestinationName", constant("queue:///MY_QUEUE?targetClient=1"))
.to("wmq:queue:MY_QUEUE?useMessageIDAsCorrelationID=true");
```

WMQ の一部のバージョンは、宛先名でこのオプションを受け入れず、次のような例外が発生します。

```
com.ibm.msg.client.jms.DetailedJMSException: JMSCC0005: The specified
value 'MY_QUEUE?targetClient=1' is not allowed for
'XMSC_DESTINATION_NAME'
```

回避策は、カスタムの DestinationResolver を使用することです。

```
JmsComponent wmq = new JmsComponent(connectionFactory);

wmq.setDestinationResolver(new DestinationResolver() {
    public Destination resolveDestinationName(Session session, String destinationName, boolean
pubSubDomain) throws JMSException {
        MQQueueSession wmqSession = (MQQueueSession) session;
        return wmqSession.createQueue("queue:///" + destinationName + "?targetClient=1");
    }
});
```

169.21. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント
- エンドポイント
- スタートガイド
- Transactional Client
- Bean インテグレーション
- Tutorial-JmsRemoting
- JMSTemplate の落とし穴

第170章 JMX コンポーネント

170.1. CAMEL JMX

Apache Camel は JMX を幅広くサポートしており、JMX クライアントを使用して Camel 管理対象オブジェクトを監視および制御できます。

Camel は、MBean 通知をサブスクライブできる JMX コンポーネントも提供します。このページでは、JMX を使用して Camel を管理および監視する方法について説明します。

170.2. オプション

JMX コンポーネントにはオプションがありません。

JMX エンドポイントは、URI 構文を使用して設定されます。

```
jmx:serverURL
```

パスおよびクエリーパラメーターを使用します。

170.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
serverURL	サーバー URL は、残りのエンドポイントから取得されます。		文字列

170.2.2. クエリーパラメーター (29 個のパラメーター):

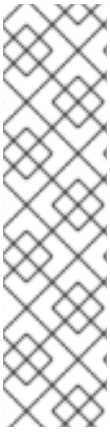
名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
format (consumer)	URI プロパティー: メッセージ本文の形式。xml または raw のいずれかです。xml の場合、通知は xml にシリアル化されます。raw の場合、raw Java オブジェクトがボディーとして設定されます。	xml	String

名前	説明	デフォルト	タイプ
granularityPeriod (consumer)	URI プロパティー: モニタータイプのみ。モニターをチェックするために Bean をポーリングする頻度。	10000	long
monitorType (consumer)	URI プロパティー: モニタータイプのみ 作成するモニターのタイプ。ストリング、ゲージ、カウンターのいずれか。		文字列
objectDomain (consumer)	**必須** URI プロパティー: 接続先の mbean のドメイン		文字列
objectName (consumer)	URI プロパティー: 接続先の mbean の名前キー。この値は、渡されるオブジェクトプロパティーと相互に排他的です。		String
observedAttribute (consumer)	URI プロパティー: モニタータイプのみ。モニター Bean について観察する属性。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handback (advanced)	URI プロパティー: 通知を受信したときにリスナーに返す値。この値は、キー jmx.handback とともにメッセージヘッダーに挿入されます。		Object
notificationFilter (advanced)	URI プロパティー: NotificationFilter を実装する Bean への参照。		NotificationFilter
objectProperties (advanced)	URI プロパティー: オブジェクト名のプロパティー。これらの値は、objectName パラメーターが設定されていない場合に使用されます		Map
reconnectDelay (advanced)	URI プロパティー: 初期接続の確立を再試行するか、失われた接続の再接続を試行する前に待機する秒数	10	int
reconnectOnConnection Failure (advanced)	URI プロパティー: true の場合、接続エラーが発生したときにコンシューマーは JMX サーバーへの再接続を試みます。コンシューマーは、接続が確立されるまで x 秒ごとに JMX 接続の再確立を試みます。ここで、x は設定された reconnectionDelay です。	false	boolean

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
testConnectionOnStartup (advanced)	URI プロパティー: true の場合、起動時に JMX 接続を確立できない場合、コンシューマーは例外を出力します。false の場合、コンシューマーは、接続が確立されるまで x 秒ごとに JMX 接続の確立を試みます。x は設定された reconnectionDelay です。	true	boolean
initThreshold (counter)	URI プロパティー: カウンターは、モニターの初期しきい値のみを監視します。通知が発生する前に、値がこれを超える必要があります。		int
modulus (counter)	URI プロパティー: カウンターモニターのみ。カウンターがゼロにリセットされる値		int
offset (counter)	URI プロパティー: カウンターモニターのみ。しきい値を超えた後に、しきい値を増やす量。		int
differenceMode (gauge)	URI プロパティー: カウンターゲージモニターのみ true の場合、通知で報告される値は、値そのものではなく、しきい値との差です。	false	boolean
notifyHigh (gauge)	URI プロパティー: ゲージモニターのみ true の場合、上限しきい値を超えたときにゲージが通知を送信します。	false	boolean
notifyLow (gauge)	URI プロパティー: ゲージモニターのみ true の場合、ゲージは下限しきい値を超えたときに通知を送信します。	false	boolean
thresholdHigh (gauge)	URI プロパティー: ゲージはゲージの高しきい値の値のみを監視します。		double
thresholdLow (gauge)	URI プロパティー: ゲージは、ゲージの低しきい値の値のみを監視します。		double
password (security)	URI プロパティー: リモート接続を確立するための認証情報		文字列
ユーザー (security)	URI プロパティー: リモート接続を確立するための認証情報		文字列

名前	説明	デフォルト	タイプ
notifyDiffer (string)	URI プロパティー: 文字列モニターのみ true にすると、文字列属性が比較対象の文字列と異なる場合に、文字列モニターは通知を起動します。	false	boolean
notifyMatch (string)	URI プロパティー: 文字列モニターのみ true にすると、文字列属性が比較対象の文字列と一致する場合に、文字列モニターは通知を起動します。	false	boolean
stringToCompare (string)	URI プロパティー: 文字列モニターのみ。比較する文字列モニターの文字列の値。		文字列

170.3. CAMEL で JMX を有効にする



注記

Camel 2.8 以前に必要な Spring JAR 依存関係

JMX インストルメンテーションを使用できるようにするには、Camel がクラスパスで **spring-context.jar**、**spring -aop.jar**、**spring -beans.jar**、および **spring-core.jar** を必要とします。これらの .jar がクラスパスにない場合、Camel は非 JMX モードにフォールバックします。この状況は、ロガー名 **org.apache.camel.impl.DefaultCamelContext** を使用して **WARN** レベルで記録されます。

Camel 2.9 以降、JMX モードで Camel を実行するために Spring JAR は **不要** になりました。

170.3.1. JMX を使用して Apache Camel を管理する

デフォルトでは、JMX 計測エージェントは Camel で有効になっています。これは、Camel ランタイムが MBean 管理オブジェクトを作成し、VM の **MBeanServer** インスタンスに登録することを意味します。これにより、Camel ユーザーは、Camel ルートが個々のプロセッサーレベルでどのように実行されるかについての洞察を即座に得ることができます。

サポートされている管理オブジェクトのタイプは、endpoint、route、service、および processor です。これらの管理オブジェクトの一部は、パフォーマンスカウンター属性に加えて、ライフサイクル操作も公開します。

DefaultManagementNamingStrategy は、MBean 登録に使用されるオブジェクト名を構築するデフォルトの命名ストラテジーです。デフォルトでは、**org.apache.camel はCamelNamingStrategy** によって作成されたすべてのオブジェクト名のドメイン名です。MBean オブジェクトのドメイン名は、Java VM システムプロパティーで設定できます。

```
-Dorg.apache.camel.jmx.mbeanObjectDomainName=your.domain.name
```

または、Spring 設定の camelContext 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" mbeanObjectDomainName="your.domain.name"/>
    ...
```

```
</camelContext>
```

Spring 設定は、両方が存在する場合、システムプロパティーよりも常に優先されます。これは、すべての JMX 関連の設定に当てはまります。

170.3.2. Camel で JMX インストルメンテーションエージェントを無効にする

Java VM システムプロパティーを次のように設定することで、JMX インストルメンテーションエージェントを無効にできます。

```
-Dorg.apache.camel.jmx.disabled=true
```

プロパティー値は **boolean** として扱われます。

または、Spring 設定の **camelContext** 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" disabled="true"/>
    ...
</camelContext>
```

または、**Camel 2.1** では、次のように無効にできるため、純粋な Java を使用している場合は少し簡単です (JVM システムプロパティーを使用する必要はありません)。

```
CamelContext camel = new DefaultCamelContext();
camel.disableJMX();
```

170.3.3. Java VM での MBeanServer の検索

各 CamelContext は、**InstrumentationLifecycleStrategy** 内にラップされた **InstrumentationAgent** のインスタンスを持つことができます。InstrumentationAgent は、Camel MBean を登録/登録解除するために **MBeanServer** とやり取りするオブジェクトです。複数の CamelContexts/InstrumentationAgent が **MBeanServer** を共有できる/共有する必要があります。デフォルトでは、Camel ランタイムは、MBeanServerFactory.findMBeanServer メソッド によって返された最初の **MBeanServer** を選択します。これは、**org.apache.camel** のデフォルトドメイン名と一致します。

アプリケーションですでに使用している **MBeanServer** インスタンスに一致するように、デフォルトのドメイン名を変更したい場合があります。特に、**MBeanServer** が JMX コネクターサーバーに接続されている場合、Camel でコネクターサーバーを作成する必要はありません。

システムプロパティーを使用して、一致するデフォルトドメイン名を設定できます。

```
-Dorg.apache.camel.jmx.mbeanServerDefaultDomain=<your.domain.name>
```

または、Spring 設定の camelContext 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" mbeanServerDefaultDomain="your.domain.name"/>
    ...
</camelContext>
```

一致する **MBeanServer** が見つからない場合は、新しい MBeanServer が作成され、新しい `MBeanServer のデフォルトドメイン名が、上記のデフォルトおよび設定に従って設定されます。

システムプロパティーを設定して JVM MBean を管理することが望ましい場合は、**PlatformMBeanServer** を使用することもできます。**MBeanServer** のデフォルトのドメイン名設定は、適用できないため無視されます。

注意

次のリリース (1.5) から、**usePlatformMBeanServer** のデフォルト値は **true** に変更されます。プロパティーを **false** に設定して、プラットフォーム **MBeanServer** の使用を無効にすることができます。

```
-Dorg.apache.camel.jmx.usePlatformMBeanServer=True
```

または、Spring 設定の **camelContext** 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" usePlatformMBeanServer="true"/>
    ...
</camelContext>
```

170.3.4. JMX RMI コネクターサーバーの作成

JMX コネクターサーバーを使用すると、JConsole などの JMX クライアントによって MBean をリモートで管理できます。Camel JMX RMI コネクターサーバーは、システムプロパティーを設定することでオプションで有効にすることができ、Camel が使用する **MBeanServer** はそのコネクターサーバーに接続されます。

```
-Dorg.apache.camel.jmx.createRmiConnector=True
```

または、Spring 設定の **camelContext** 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true"/>
    ...
</camelContext>
```

170.3.5. JMX Service URL

デフォルトの JMX サービス URL の形式は次のとおりです。

```
service:jmx:rmi:///jndi/rmi://localhost:<registryPort>/<serviceUrlPath>
```

registryPort は RMI レジストリーポートで、デフォルト値は **1099** です。

システムプロパティーで RMI レジストリーポートを設定できます。

```
-Dorg.apache.camel.jmx.rmiConnector.registryPort=<port number>
```

または、Spring 設定の **camelContext** 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" registryPort="port number"/>
    ...
```

```
</camelContext>
```

serviceUrlPath は URL のパス名で、デフォルト値は **/jmxrmi/camel** です。

システムプロパティーでサービス URL パスを設定できます。

```
-Dorg.apache.camel.jmx.serviceUrlPath=<path>
```

ヒント

Java での ManagementAgent 設定の設定

Camel 2.4 以降では、**ManagementAgent** でさまざまなオプションを設定することもできます。

```
context.getManagementStrategy().getManagementAgent().setServiceUrlPath("/foo/bar");
context.getManagementStrategy().getManagementAgent().setRegistryPort(2113);
context.getManagementStrategy().getManagementAgent().setCreateConnector(true);
```

または、Spring 設定の camelContext 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" serviceUrlPath="path"/>
    ...
</camelContext>
```

デフォルトでは、RMI サーバーオブジェクトは動的に生成されたポートをリッスンします。これは、ファイアウォール経由で確立された接続で問題になる可能性があります。このような状況では、RMI 接続ポートをシステムプロパティーで明示的に設定できます。

```
-Dorg.apache.camel.jmx.rmiConnector.connectorPort=<port number>
```

または、Spring 設定の **camelContext** 要素内に **jmxAgent** 要素を追加することによって:

```
<camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
  <jmxAgent id="agent" createConnector="true" connectorPort="port number"/>
    ...
</camelContext>
```

コネクターポートオプションが設定されている場合、JMX サービス URL は次のようになります。

```
service:jmx:rmi://localhost:<connectorPort>/jndi/rmi://localhost:<registryPort>/<serviceUrlPath>
```

170.3.6. Camel JMX サポートのシステムプロパティー

Property Name	value	説明
org.apache.camel.jmx	**true** または **false**	**true** の場合、Camel の jmx 機能が有効になります

以下のこのセクションで、システムプロパティーの詳細を参照してください: **jmxAgent プロパティーリファレンス**。

170.3.7. JMX で認証を使用する方法

JDK の JMX には、認証のための機能と、SSL を介した安全な接続を使用するための機能があります。これを使用する方法については、SUN のドキュメントを参照する必要があります。

- http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html
- http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html

170.3.8. アプリケーションサーバー内の JMX

170.3.8.1. Tomcat 6

Tomcat で JMX を有効にする方法の詳細については、このページ を参照してください。

つまり、catalina.sh (Windows では catalina.bat) ファイルを変更して、次のオプションを設定します。

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote \
  -Dcom.sun.management.jmxremote.port=1099 \
  -Dcom.sun.management.jmxremote.ssl=false \
  -Dcom.sun.management.jmxremote.authenticate=false
```

170.3.8.2. JBoss AS 4

デフォルトでは、JBoss は独自の **MBeanServer** を作成します。Camel が同じサーバーに公開できるようにするには、次の手順に従います。

1. プラットフォーム **MBeanServer** を使用するように Camel に指示します (Camel 1.5 ではデフォルトで true になります)。

```
<camel:camelContext id="camelContext">
  <camel:jmxAgent id="jmxAgent" mbeanObjectDomainName="org.yourname"
usePlatformMBeanServer="true"  />
</camel:camelContext>
```

1. Platform **MBeanServer** を使用するように JBoss インスタンスを変更します。
run.sh または **run.conf -Djboss.platform.mbeanserver** を編集して、次のプロパティーを **JAVA_OPTS** に追加します。See http://wiki.jboss.org/wiki/JBossMBeansInJConsole

170.3.8.3. WebSphere

mbeanServerDefaultDomain を **WebSphere** に変更します。

```
<camel:jmxAgent id="agent" createConnector="true" mbeanObjectDomainName="org.yourname"
usePlatformMBeanServer="false" mbeanServerDefaultDomain="WebSphere"/>
```

170.3.8.4. Oracle OC4j

Oracle OC4J J2EE アプリケーションサーバーは、Camel がプラットフォーム **MBeanServer** にアクセスすることを許可しません。Camel は **WARNING** をログに記録するため、ログでこれを識別できます。

```
xxx xx, xxxx xx:xx:xx xx org.apache.camel.management.InstrumentationLifecycleStrategy
onContextStart
WARNING: Could not register CamelContext MBean
java.lang.SecurityException: Unauthorized access from application: xx to MBean:
java.lang:type=ClassLoading
        at
oracle.oc4j.admin.jmx.shared.UserMBeanServer.checkRegisterAccess(UserMBeanServer.java:873)
```

これを解決するには、Camel で JMX エージェントを無効にする必要があります。**Camel で JMX インスツルメンテーションエージェントを無効にする** セクションを参照してください。

170.3.9. 高度な JMX 設定

Spring 設定ファイルを使用すると、管理のために Camel を JMX に公開する方法を設定できます。場合によっては、コネクターのポートやパス名などの詳細情報をここで指定できます。

170.3.10. 例:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" registryPort="2000"
mbeanServerDefaultDomain="org.apache.camel.test"/>
    <route>
      <from uri="seda:start"/>
      <to uri="mock:result"/>
    </route>
</camelContext>
```

Java 5 JMX 設定を変更する場合は、さまざまな JMX システムプロパティー を使用できます。

たとえば、次の環境変数を設定することにより (プラットフォームに応じて **set** または **export** を使用)、Sun JMX コネクターへのリモート JMX 接続を有効にすることができます。これらの設定は、Java 1.5+ 内の Sun JMX コネクターのみを設定し、Camel がデフォルトで作成する JMX コネクターは設定しません。

```
SUNJMX=-Dcom.sun.management.jmxremote=true -Dcom.sun.management.jmxremote.port=1616 \
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
```

(SUNJMX 環境変数は、JVM の追加の起動パラメーターとして、Camel の起動スクリプトによって単純に使用されます。Camel を直接起動する場合は、これらのパラメーターを自分で渡す必要があります。)

170.3.11. jmxAgent プロパティーリファレンス

Spring プロパティー	System プロパティー	デフォルト値	説明
id			JMX エージェント名。オプションではありません

Spring プロパティー	System プロパティー	デフォルト値	説明
usePlatformMBeanServer	**org.apache.camel.jmx.usePlatformMBeanServer**	**false**、**true** - リリース 1.5 以降	**true** の場合、JVM から **MBeanServer** を使用します
mbeanServerDefaultDomain	**org.apache.camel.jmx.mbeanServerDefaultDomain**	**org.apache.camel**	**MBeanServer** のデフォルトの JMX ドメイン
mbeanObjectDomainName	**org.apache.camel.jmx.mbeanObjectDomainName**	**org.apache.camel**	すべてのオブジェクト名が使用する JMX ドメイン
createConnector	**org.apache.camel.jmx.createRmiConnect**	**false**	**MBeanServer** の JMX コネクターを (リモート管理を可能にするために) 作成する必要がある場合
registryPort	**org.apache.camel.jmx.rmiConnector.registryPort**	**1099**	JMX RMI レジストリーが使用するポート
connectorPort	**org.apache.camel.jmx.rmiConnector.connectorPort**	-1 (動的)	JMX RMI サーバーが使用するポート
serviceUrlPath	**org.apache.camel.jmx.serviceUrlPath**	**/jmxrmi/camel**	JMX コネクターが登録されるパス
onlyRegisterProcessorWithCustomId	**org.apache.camel.jmx.onlyRegisterProcessorWithCustomId**	**false**	Camel 2.0: このオプションを有効にすると、カスタム ID セットを持つプロセッサーのみが登録されます。これにより、JMX コンソールで不要なプロセッサーを除外できます。

Spring プロパティー	System プロパティー	デフォルト値	説明
statisticsLevel		**All** / **Default**	Camel 2.1: MBean のパフォーマンス統計情報を有効にするかどうかのレベルを設定します。詳細については、**パフォーマンス統計情報の粒度レベルの設定** セクションを参照してください。Camel 2.16 以降では、All オプションの名前が Default に変更され、追加のランタイム JMX メトリックを収集できる新しい Extended オプションが導入されました。
includeHostName	**org.apache.camel.jmx.includeHostName**		Camel 2.13: MBean の命名にホスト名を含めるかどうか。Camel 2.13 以降では、これはデフォルトの **false** ですが、古いリリースではデフォルトの **true** です。本当に必要な場合は、このオプションを使用して古い動作を復元できます。
useHostIPAddress	**org.apache.camel.jmx.useHostIPAddress**	**false**	Camel 2.16: リモートコネクターの作成時にサービス URL でホスト名または IP アドレスを使用するかどうか。デフォルトでは、ホスト名が使用されます。
loadStatisticsEnabled	**org.apache.camel.jmx.loadStatisticsEnabled**	**false**	Camel 2.16: 負荷統計情報を有効にするかどうか (CamelContext ごとにバックグラウンドスレッドを使用して負荷統計情報を収集します)。
endpointRuntimeStatisticsEnabled	**org.apache.camel.jmx.endpointRuntimeStatisticsEnabled**	**true**	Camel 2.16: エンドポイントのランタイム統計情報を有効にするかどうか (各入力エンドポイントおよび出力エンドポイントのランタイム使用状況を収集します)。

170.3.12. MBean を常に登録するか、新しいルートに登録するか、デフォルトでのみ登録するかの設定

Camel 2.7 以降で利用可能

Camel は、mbeans を登録するかどうかを制御する 2 つの設定を提供するようになりました

オプション	デフォルト	説明
registerAlways	**false**	有効にすると、MBean は常に登録されます。
registerNewRoutes	**true**	有効にすると、CamelContext が開始された後に新しいルートを追加すると、その特定のルートから MBean も登録されます。

デフォルトでは、Camel は起動時に設定されたすべてのルートに MBean を登録します。**registerNewRoutes** オプションは、後で新しいルートを追加する場合に MBean も登録する必要があるかどうかを制御します。たとえば、管理が不要な一時的なルートを追加および削除する場合は、これを無効にすることができます。

一意のエンドポイントを持つ受信者リストなどの動的 EIP パターンを使用する場合は、**registerAlways** オプションを使用するように少し注意してください。その場合、それぞれの固有のエンドポイントとそれに関連付けられたサービス/プロデューサーも登録されます。これは、レジストリー内の mbean の数が増加するため、システムの機能低下につながる可能性があります。MBean は軽量オブジェクトではないため、メモリーを消費します。

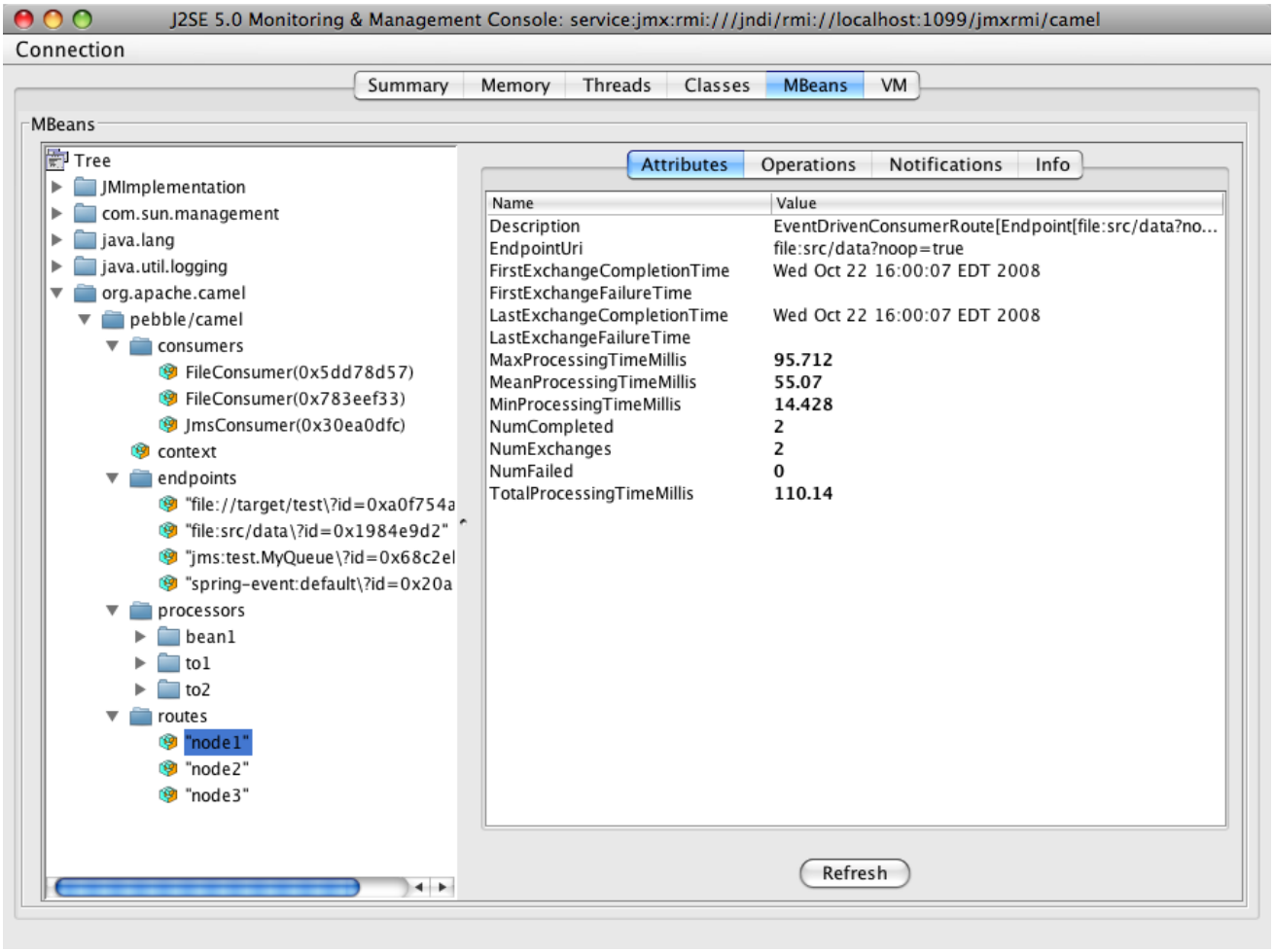
170.4. JMX を使用した CAMEL のモニタリング

170.4.1. JConsole を使用して Camel をモニターする

Camel と同じホストで JConsole を実行している場合は、ローカル接続のリストに **CamelContext** が表示されます。

リモートの Camel インスタンスに接続するには、またはローカルプロセスが表示されない場合は、リモートプロセスオプションを使用して URL を入力します。以下は localhost **URL:service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel** の例です。

JConsole で Apache Camel を使用する:



170.4.2. 登録されているエンドポイント

Camel 2.1 以降では、数千または数百万のエンドポイントが使用される場合、非シングルトンのオーバーヘッドがかなり大きくなるため、**singleton**エンドポイント **のみ** が登録されます。これは、受信者リスト EIP を使用する場合、または大量のメッセージを送信する **ProducerTemplate** から発生する可能性があります。

170.4.3. 登録されているプロセッサー

この FAQ を参照してください。

170.4.4. JMX NotificationListener を使用して camel イベントをリッスンする方法は?

Camel の通知イベントは、何が起こっているかを大まかに示します。コンテキストとエンドポイントからライフサイクルイベントを確認でき、エンドポイントで送受信されるエクスチェンジを確認できます。

Camel 2.4 から、カスタム JMX NotificationListener を使用して camel イベントをリッスンできます。

CamelContext を開始する前に、まず **JmxNotificationEventNotifier** を設定する必要があります。

```
// Set up the JmxNotificationEventNotifier
notifier = new JmxNotificationEventNotifier();
notifier.setSource("MyCamel");
notifier.setIgnoreCamelContextEvents(true);
notifier.setIgnoreRouteEvents(true);
notifier.setIgnoreServiceEvents(true);
```

```
CamelContext context = new DefaultCamelContext(createRegistry());
context.getManagementStrategy().addEventNotifier(notifier);
```

次に、イベントをリッスンするためのリスナーを登録できます。

```
// register the NotificationListener
ObjectName on = ObjectName.getInstance("org.apache.camel:context=camel-
1,type=eventnotifiers,name=JmxEventNotifier");
MyNotificationListener listener = new MyNotificationListener();
context.getManagementStrategy().getManagementAgent().getMBeanServer().addNotificationListener(o
n,
    listener,
    new NotificationFilter() {
        private static final long serialVersionUID = 1L;

        public boolean isNotificationEnabled(Notification notification) {
            return notification.getSource().equals("MyCamel");
        }
    }, null);
```

170.4.5. Tracer MBean を使用して粒度の細かいトレースを取得する

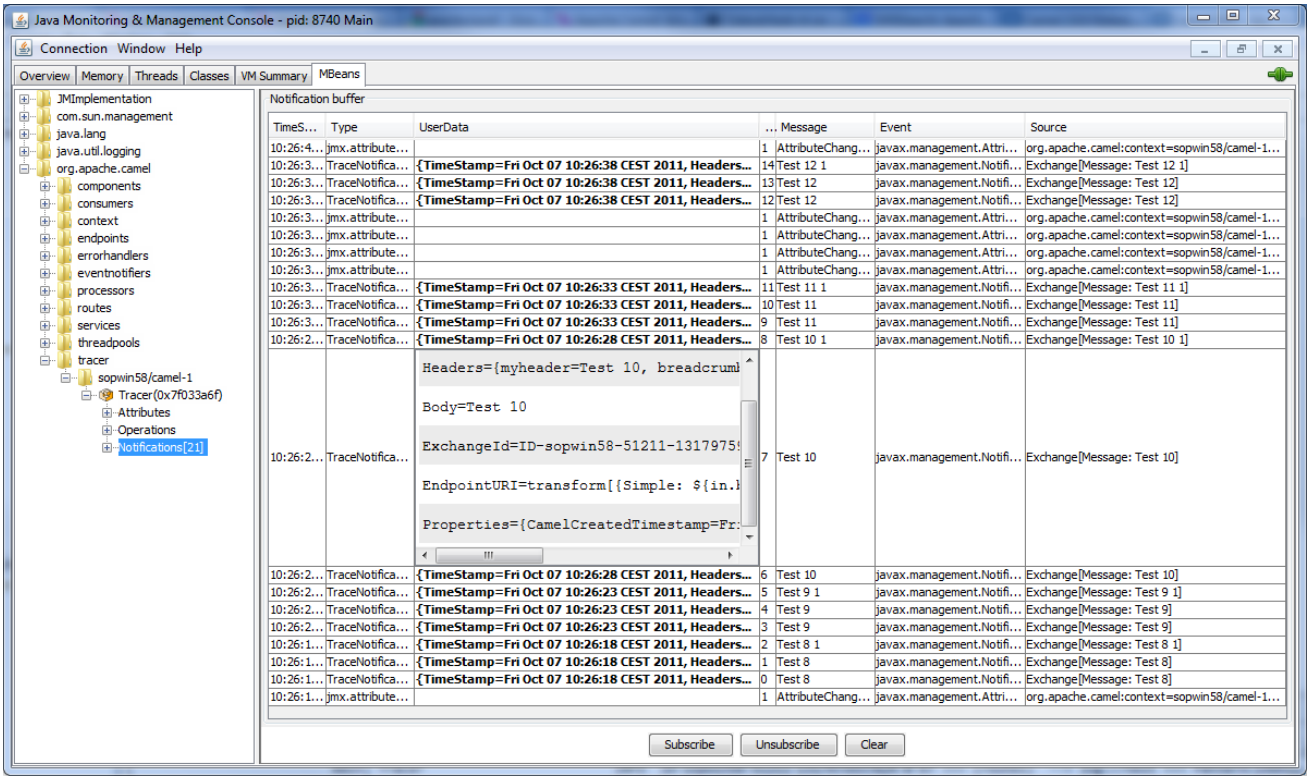
Camel 2.9.0 上のあらい粒度の通知に加えて、細かい粒度のトレースイベントの JMX 通知がサポートされます。

これらは Tracer MBean にあります。詳細なトレースを有効にするには、まずコンテキストまたはルートでトレースを有効にする必要があります。

これは、コンテキストの設定時、またはコンテキスト/ルート MBean で実行できます。

2 番目のステップとして、トレーサーで **jmxTraceNotifications** 属性を **true** に設定する必要があります。これは、コンテキストを設定するとき、または実行時にトレーサー MBean で実行できます。

これで、JConsole を使用してトレーサー MBean で TraceEvent 通知を登録できるようになりました。すべてのエクスチェンジとメッセージの詳細を含む、ルートのすべてのステップに対して 1 つの通知があります。



170.5. 独自の CAMEL コードに JMX を使用する

170.5.1. 独自のマネージドエンドポイントの登録

Camel 2.0 以降で利用可能

独自のエンドポイントを Spring マネージドアノテーション **@ManagedResource** で装飾して、それらを Camel **MBeanServer** に登録し、JMX を使用してカスタム MBean にアクセスできるようにすることができます。

注記

Camel 2.1 では、これをエンドポイント以外にも適用するように変更しましたが、インターフェイス **org.apache.camel.spi.ManagementAware** も実装する必要があります。これについては後で詳しく説明します。

たとえば、管理するオプションを定義する次のカスタムエンドポイントがあります。

```
@ManagedResource(description = "Our custom managed endpoint")
public class CustomEndpoint extends MockEndpoint implements
ManagementAware<CustomEndpoint> {

    public CustomEndpoint(final String endpointUri, final Component component) {
        super(endpointUri, component);
    }

    public Object getManagedObject(CustomEndpoint object) {
        return this;
    }

    public boolean isSingleton() {
        return true;
```

```
    }

    protected String createEndpointUri() {
        return "custom";
    }

    @ManagedAttribute
    public String getFoo() {
        return "bar";
    }

    @ManagedAttribute
    public String getEndpointUri() {
        return super.getEndpointUri();
    }
}
```

Camel 2.9 以降では、**org.apache.camel.api.management** パッケージの **@ManagedResource**、**@ManagedAttribute**、および **@ManagedOperation** を使用することが推奨されています。これにより、カスタムコードは Spring JAR に依存しなくなります。

170.5.2. 独自のマネージドサービスのプログラミング

Camel 2.1 以降で利用可能

Camel は、管理のためにサービスを登録するときに、独自の MBean を使用できるようになりました。つまり、たとえば、カスタム Camel コンポーネントを開発し、エンドポイント、コンシューマー、プロデューサーなどの MBean を公開することができます。インターフェイス **org.apache.camel.spi.ManagementAware** を実装し、Camel が使用する管理対象オブジェクトを返すだけです。

JMX API は本当に苦痛で酷いものだと考える前に、実際にそうです。Spring にとっても幸運なことに、既存の Bean の管理をエクスポートするために使用できるさまざまなアノテーションが作成されました。つまり、これを頻繁に使用し、**ManagementAware** インターフェイスから **getManagedObject** で **this** を返すだけです。例については、**CustomEndpoint** を使用した上記のコード例を参照してください。

現在、**Camel 2.1** では、Camel が管理のために登録するすべてのオブジェクトに対してこれを行うことができますが、これはすべてではありません。

この **ManagementAware** インターフェイスを実装しないサービスの場合、Camel は以下の表で定義されているデフォルトのラッパーを使用するようにフォールバックします。

タイプ	MBean ラッパー
CamelContext	**ManagedCamelContext**
コンポーネント	**ManagedComponent**
エンドポイント	**ManagedEndpoint**
コンシューマー	**ManagedConsumer**

タイプ	MBean ラッパー
Producer	**ManagedProducer**
ルート	**ManagedRoute**
プロセッサー	**ManagedProcessor**
トレーサー	**ManagedTracer**
サービス	**ManagedService**

それに加えて、次のような特殊なタイプの拡張ラッパーがいくつかあります。

タイプ	MBean ラッパー
ScheduledPollConsumer	**ManagedScheduledPollConsumer**
BrowsableEndpoint	**ManagedBrowseableEndpoint**
Throttler	**ManagedThrottler**
Delayer	**ManagedDelayer**
SendProcessor	**ManagedSendProcessor**

将来的には、より多くの EIP パターンのラッパーを追加する予定です。

170.5.3. ManagementNamingStrategy

Camel 2.1 以降で利用可能

Camel は、**org.apache.camel.spi.ManagementNamingStrategy** による命名戦略のためのプラグ可能な API を提供します。デフォルトの実装は、すべての MBean が登録されている MBean 名を計算するために使用されます。

170.5.4. 管理命名パターン

Camel 2.10 以降で利用可能

Camel 2.10 以降では、MBean の命名パターンを簡単に設定できるようになりました。パターンは、ドメイン名の後にキーとして **ObjectName** の一部として使用されます。

デフォルトでは、Camel は次のように **ManagedCamelContextMBean** の MBean 名を使用します。

```
org.apache.camel:context=localhost/camel-1,type=context,name=camel-1
```

また、**Camel 2.13** 以降では、ホスト名は MBean 名に含まれていないため、上記の例は次のようになります。

```
org.apache.camel:context=camel-1,type=context,name=camel-1
```

CamelContext で名前を設定すると、その名前も **ObjectName** の一部になります。たとえば、

```
<camelContext id="myCamel" ...>
```

MBean 名は次のようになります。

```
org.apache.camel:context=localhost/myCamel,type=context,name=myCamel
```

上記の指定された名前の MBean がすでに存在するなど、JVM で名前の競合が発生した場合、Camel はデフォルトで、カウンターを使用して **JMXMBeanServer** で新しいフリーネームを見つけてこれを自動修正しようとします。以下に示すように、カウンターが追加されたので、**ObjectName** の一部として **myCamel-1** があります。

```
org.apache.camel:context=localhost/myCamel-1,type=context,name=myCamel
```

これが可能なのは、Camel がデフォルトで次のトークンをサポートする命名パターンを使用しているためです。

- **camelId** = the CamelContext id (eg the name)
- **name** - **camelId** と同じ
- **counter** - 増分カウンター * **bundleId** - OSGi バンドル ID (OSGi 環境のみ)
- **symbolicName** - OSGi シンボリック名 (OSGi 環境のみ)
- **version** - OSGi バンドルのバージョン (OSGi 環境のみ)

デフォルトの命名パターンは、OSGi と非 OSGi で次のように区別されます。

- 非 OSGI: **name**
- OSGi: **bundleId-name**
- OSGi **Camel 2.13: symbolicName**

ただし、**JMXMBeanServer** に名前の競合がある場合、Camel は自動的にフォールバックし、パターン内の **counter** を使用してこれを修正します。したがって、次のパターンが使用されます。

- 非 OSGI: **name-counter**
- OSGi: **bundleId-name-counter**
- OSGi **Camel 2.13: symbolicName-counter**

明示的な命名パターンを設定すると、そのパターンが常に使用され、上記のデフォルトパターンは使用され **ません**。

これにより、レジストリーの **CamelContext** id と **JMXMBeanRegistry** の JMX MBean の両方の命名を非常に簡単に完全に制御できます。

Camel 2.15 以降では、JVM システムプロパティーを使用してデフォルトの管理名パターンを設定し、これを JVM に対してグローバルに設定できます。以下の例に示すように、このパターンを明示的に設定することでオーバーライドできることに注意してください。

JVM システムプロパティーを設定して、名前の前に cool を付けたデフォルトの管理名パターンを使用します。

```
System.setProperty(JmxSystemPropertyKeys.MANAGEMENT_NAME_PATTERN, "cool-#name#");
```

したがって、**CamelContext の** 両方に明示的な名前を付け、変更されない (たとえば、カウンターがない) 固定の MBean 名を使用する場合は、新しい **managementNamePattern** 属性を使用できます。

```
<camelContext id="myCamel" managementNamePattern="#name#">
```

その場合、MBean 名は常に次のようになります。

```
org.apache.camel:context=localhost/myCamel,type=context,name=myCamel
```

Java では、次のように **managementNamePattern** を設定できます。

```
context.getManagementNameStrategy().setNamePattern("#name#");
```

managementNamePattern で id とは異なる名前を使用することもできます。たとえば、次のようにできます。

```
<camelContext id="myCamel" managementNamePattern="coolCamel">
```

OSGi バンドル ID を MBean 名の一部として使用したくない場合は、OSGi 環境でこれを行うことができます。OSGi バンドル ID は、サーバーを再起動するか、同じアプリケーションをアンインストールしてインストールすると変更される可能性があるためです。OSGi バンドル ID を名前の一部として使用しないようにするには、次のようにします。

```
<camelContext id="myCamel" managementNamePattern="#name#">
```

これには、**myCamel が** JVM 全体で一意である必要があることに注意してください。同じ **CamelContext** id と **managementNamePattern** を持つ 2 番目の Camel アプリケーションをインストールすると、Camel は起動時に失敗し、MBean がすでに存在する例外を報告します。

170.5.5. ManagementStrategy

Camel 2.1 以降で利用可能

Camel は、完全にプラグ可能な管理ストラテジーを提供するようになり、管理を 100% 制御できるようになりました。これは、管理のための多くの方法を備えた豊富なインターフェイスです。管理対象オブジェクトを **MBeanServer** に追加および削除するためだけでなく、**org.apache.camel.spi.EventNotifier** API を使用してイベント通知も提供されます。たとえば、他の管理製品へのアダプターの提供を容易にします。さらに、Apache ですぐに提供される詳細と機能を提供することもできます。

170.5.6. パフォーマンス統計情報の粒度レベルの設定

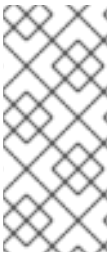
Camel 2.1 以降で利用可能

Camel の起動時にパフォーマンス統計情報が有効かどうかに関係なく、事前に設定されたレベルを設定できるようになりました。レベルは

- **Extended** - デフォルトですが、エンドポイントの使用状況のきめ細かいレベルなど、実行時に追加の統計が収集されます。このオプションには Camel 2.16 が必要です
- **All** / **Default** - Camel は、ルートとプロセッサーの両方の統計情報を有効にします (きめ細かい)。**Camel 2.16** 以降、All オプションの名前が Default に変更されました。
- **RoutesOnly** - Camel はルートの統計情報のみを有効にします (あらい粒度)
- **オフ** - Camel は統計情報を有効にしません。

Camel 2.9 以降、パフォーマンス統計情報には、CamelContext および Route MBean ごとの平均負荷統計情報も含まれます。統計は、1 分、5 分、および 15 分あたりのレートでの、進行中のエクスチェンジの数に基づく平均負荷です。これは、Unix システムの負荷統計情報に似ています。**Camel 2.11** 以降では、**<jmxAgent>** で **loadStatisticsEnabled=false** を設定することにより、負荷パフォーマンス統計情報を明示的に無効にすることができます。静的レベルもオフに設定されている場合はオフになることに注意してください。**Camel 2.13** 以降、ロードパフォーマンス統計情報はデフォルトで無効になっています。これを有効にするには、**<jmxAgent>** で **loadStatisticsEnabled=true** を設定します。

実行時に、管理コンソール (JConsole など) を使用して、特定のルートまたはプロセッサーの統計情報が有効かどうかをいつでも変更できます。



注記

統計情報が有効になっているとはどういう意味ですか?

Statistics enabled は、Camel がその特定の MBean の詳細なパフォーマンス統計情報を行うことを意味します。表示できる統計情報は、完了/失敗した交換の数、最後/合計/最小/最大/平均処理時間、最初/最後に失敗した時間など、多数あります。

Java DSL を使用して、次の方法でこのレベルを設定します。

```
// only enable routes when Camel starts
context.getManagementStrategy().setStatisticsLevel(ManagementStatisticsLevel.RoutesOnly);
```

そして、Spring DSL から次のことを行います。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
   <jmxAgent id="agent" statisticsLevel="RoutesOnly"/>
 ...
</camelContext>
```

170.6. 機密情報の隠蔽

Camel 2.12 以降で利用可能

デフォルトでは、Camel は URI を使用して設定されたエンドポイントなどの MBean を JMX に登録します。この設定では、パスワードなどの機密情報が含まれる場合があります。

この情報は、以下に示すように **mask** オプションを有効にすることで非表示にすることができます。

Java DSL を使用して、次のようにしてこれをオンにします。

```
// only enable routes when Camel starts
context.getManagementStrategy().getManagementAgent().setMask(true);
```

そして、Spring DSL から次のことを行います。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
   <jmxAgent id="agent" mask="true"/>
 ...
</camelContext>
```

これにより、パスワードやパスフレーズなどのオプションを持つ URI がマスクされ、代わりの値として **xxxxxx** が使用されます。

170.6.1. マスクする JMX 属性と操作の宣言

org.apache.camel.api.management.ManagedAttribute および **org.apache.camel.api.management.ManagedOperation** では、属性 **mask** を **true** に設定して、この JMX 属性/操作の結果をマスクする必要があることを示すことができます (JMX エージェントで有効化されています。上記を参照してください)。

たとえば、camel-core **org.apache.camel.api.management.mbean.ManagedEndpointMBean** のデフォルトのマネージドエンドポイントでは、**EndpointUri** JMX 属性がマスクされていると宣言しています。

```
@ManagedAttribute(description = "Endpoint URI", mask = true)
String getEndpointUri();
```

170.7. 関連項目

- 管理例
- プロセッサーが JConsole に表示されないのはなぜですか

第171章 JOLT コンポーネント

Camel バージョン 2.16 以降で利用可能

jolt: コンポーネントを使用すると、JOLT 仕様を使用して JSON メッセージを処理できます。これは、JSON から JSON への変換を行う場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jolt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

171.1. URI 形式

```
jolt:specName[?options]
```

ここで、**specName** は、呼び出す仕様のクラスパスローカル URI です。またはリモート仕様の完全な URL (例: file://folder/myfile.json)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

171.2. オプション

JOLT コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
transform (advanced)	使用する変換を明示的に設定します。設定されていない場合、transformDsl で指定された Transform が作成されます		Transform
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JOLT エンドポイントは、URI 構文を使用して設定されます。

```
jolt:resourceUri
```

パスおよびクエリーパラメーターを使用します。

171.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

171.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
inputType (producer)	入力がハイドレートされた JSON か JSON 文字列かを指定します。	Hydrated	JoltInputOutputType
outputType (producer)	出力をハイドレートされた JSON にするか、JSON 文字列にするかを指定します。	Hydrated	JoltInputOutputType
transformDsl (producer)	エンドポイントリソースの変換 DSL を指定します。何も指定されていない場合は、Chainr が使用されます。	Chainr	JoltTransformType
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

171.3. サンプル

たとえば、次のようなものを使用できます

```
from("activemq:My.Queue").
  to("jolt:com/acme/MyResponse.json");
```

そしてファイルベースのリソース:

```
from("activemq:My.Queue").
  to("jolt:file://myfolder/MyResponse.json?contentCache=true").
  to("activemq:Another.Queue");
```

コンポーネントがヘッダーを介して動的に使用する仕様を指定することもできます。たとえば、次のようになります。

```
from("direct:in").
  setHeader("CamelJoltResourceUri").constant("path/to/my/spec.json").
  to("jolt:dummy");
```

171.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第172章 JPA コンポーネント

Camel バージョン 1.0 以降で利用可能

jpa コンポーネントを使用すると、EJB 3 の Java Persistence Architecture (JPA) を使用して永続ストレージから Java オブジェクトを格納および取得できます。JPA は、OpenJPA、Hibernate、TopLink などのオブジェクト/リレーショナルマッピング (ORM) 製品をラップする標準インターフェイス層です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jpa</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

172.1. エンドポイントへの送信

Java エンティティー Bean を JPA プロデューサーエンドポイントに送信することにより、データベースに格納できます。**In** メッセージのボディーは、エンティティー Bean (つまり、@Entity アノテーションが付けられた POJO) またはエンティティー Bean のコレクションまたは配列であると想定されます。

ボディーがエンティティーのリストである場合は、プロデューサーエンドポイントに渡される設定として **entityType=java.util.ArrayList** を必ず使用してください。

本文に前にリストされたタイプのいずれも含まれていない場合は、エンドポイントの前に Message Translator を配置して、最初に必要な変換を実行します。

Camel 2.19 以降では、プロデューサーにも **query**、**namedQuery** または **nativeQuery** を使用できます。また、**parameters** の値では、メッセージボディー、ヘッダーなどからパラメーター値を取得できる単純な式を使用できます。これらのクエリーは、**SELECT** JPQL/SQL ステートメントを使用した一連のデータの取得、および **UPDATE**/**DELETE** JPQL/SQL ステートメントを使用した一括更新/削除の実行に使用できます。camel は **query** や **nativeQuery** とは異なり、名前付きクエリーを調べないため、**namedQuery** で **UPDATE**/**DELETE** を実行する場合は、**useExecuteUpdate** を **true** に指定する必要があることに注意してください。

172.2. エンドポイントからの消費

JPA コンシューマーエンドポイントからメッセージを消費すると、データベース内のエンティティー Bean が削除 (または更新) されます。これにより、データベーステーブルを論理キューとして使用できます。コンシューマーはキューからメッセージを取得し、それらを削除/更新してキューから論理的に削除します。

エンティティー Bean が処理されたとき (およびルーティングが完了したとき) にエンティティー Bean を削除したくない場合は、URI で **consumeDelete=false** を指定できます。これにより、ポーリングごとにエンティティーが処理されます。

エンティティーに何らかの更新を実行して、処理済みとしてマークする (今後のクエリーから除外するなど) 場合は、@Consumed で Bean にアノテーションを付けることができます。処理済み (およびルーティングが完了したとき)。

Camel 2.13 以降では、エンティティー Bean が処理される前 (ルーティング前) に呼び出される @PreConsumed を使用できます。

大量 (100K+) の行を消費していて OutOfMemory の問題が発生している場合は、maximumResults を適切な値に設定する必要があります。

172.3. URI 形式

```
jpa:entityClassName[?options]
```

エンドポイントに送信する場合、**entityClassName** はオプションです。指定すると、型コンバーターが本文が正しい型であることを確認するのに役立ちます。

消費する場合、**entityClassName** は必須です。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

172.4. オプション

JPA コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
entityManagerFactory (Common)	EntityManagerFactory を使用する場合。これを設定することを強くお勧めします。		EntityManagerFactory
transactionManager (Common)	PlatformTransactionManager を使用してトランザクションを管理する場合。		PlatformTransaction Manager
joinTransaction (Common)	camel-jpa コンポーネントはデフォルトでトランザクションに参加します。このオプションを使用して、これをオフにすることができます。たとえば、LOCAL_RESOURCE を使用していて、参加トランザクションが JPA プロバイダーで機能しない場合などです。このオプションは、すべてのエンドポイントで設定する代わりに、JpaComponent でグローバルに設定することもできます。	true	boolean
sharedEntityManager (common)	コンシューマー/プロデューサーに Spring の SharedEntityManager を使用するかどうか。これは EXTENDED EntityManager ではないため、ほとんどの場合、joinTransaction は false に設定する必要があります。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JPA エンドポイントは、URI 構文を使用して設定されます。

-

```
jpa:entityType
```

パスおよびクエリーパラメーターを使用します。

172.4.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
entityType	**必須** エンティティーとして使用する JPA アノテーション付きクラス。		Class<?>

172.4.2. クエリーパラメーター (42 個のパラメーター):

名前	説明	デフォルト	タイプ
joinTransaction (Common)	camel-jpa コンポーネントはデフォルトでトランザクションに参加します。このオプションを使用して、これをオフにすることができます。たとえば、LOCAL_RESOURCE を使用していて、参加トランザクションが JPA プロバイダーで機能しない場合などです。このオプションは、すべてのエンドポイントで設定する代わりに、JpaComponent でグローバルに設定することもできます。	true	boolean
maximumResults (common)	クエリーで取得する結果の最大数を設定します。	-1	int
namedQuery (common)	名前付きクエリーを使用する場合。		String
nativeQuery (common)	カスタムネイティブクエリーを使用する場合。ネイティブクエリーを使用する場合にも、オプション resultClass を使用することができます。		String
parameters (common)	このキーと値のマッピングは、クエリーパラメーターの作成に使用されます。これは、キーが特定の JPA クエリーの名前付きパラメーターであり、値が選択する対応する有効な値であるジェネリック型 java.util.Map であることが期待されます。プロデューサーに使用する場合は、単純式をパラメーター値として使用できます。メッセージボディー、ヘッダーなどからパラメーター値を取得できます。		Map
persistenceUnit (common)	**必須** デフォルトで使用される JPA 持続性ユニット。	camel	String
query (common)	カスタムクエリーを使用する場合。		String

名前	説明	デフォルト	タイプ
resultClass (common)	返されるペイロードのタイプを定義します (entityManager.createNativeQuery (nativeQuery) の代わりに entityManager.createNativeQuery (nativeQuery、resultClass) を呼び出します)。このオプションを指定しないと、オブジェクト配列が返されます。データを消費するときにネイティブクエリーと組み合わせて使用する場合にのみ影響します。		Class<?>
sharedEntityManager (common)	コンシューマー/プロデューサーに Spring の SharedEntityManager を使用するかどうか。これは EXTENDED EntityManager ではないため、ほとんどの場合、joinTransaction は false に設定する必要があります。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumeDelete (consumer)	true の場合、エンティティーは消費後に削除されます。false の場合、エンティティーは削除されません。	true	boolean
consumeLockEntity (consumer)	ポーリングの結果を処理する際に、各エンティティー Bean に排他ロックを設定するかどうかを指定します。	true	boolean
deleteHandler (consumer)	コンシューマーがエクスチェンジの処理を完了した後に、カスタム DeleteHandler を使用して行を削除する場合。		Object>
lockModeType (consumer)	コンシューマーでロックモードを設定する場合。	PESSIMISTIC_WRITE	LockModeType
maxMessagesPerPoll (consumer)	ポーリングごとに収集するメッセージの最大数を定義する整数値。デフォルトでは最大値は設定されていません。サーバーの起動時に何千ものメッセージをポーリングするのを避けるために使用できます。無効にするには、0 または負の値を設定します。		int

名前	説明	デフォルト	タイプ
preDeleteHandler (consumer)	カスタム Pre-DeleteHandler を使用して、コンシューマーがエンティティーを読み取った後に行を削除する場合。		Object>
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
skipLockedEntity (consumer)	ロック時に NOWAIT を使用し、サイレントにエンティティーをスキップするかどうかを設定する場合。	false	boolean
transacted (consumer)	バッチ全体が処理されたときに、すべてのメッセージがコミットまたはロールバックされるトランザクションモードでコンシューマーを実行するかどうか。デフォルトの動作 (false) は、以前に正常に処理されたすべてのメッセージをコミットし、最後に失敗したメッセージのみをロールバックします。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
flushOnSend (producer)	エンティティー Bean が永続化された後、EntityManager をフラッシュします。	true	boolean
remove (producer)	entityManager.remove (entity) を使用することを示します。	false	boolean

名前	説明	デフォルト	タイプ
useExecuteUpdate (producer)	プロデューサーがクエリーを実行するときに executeUpdate() を使用するかどうかを設定します。INSERT、UPDATE、または DELETE ステートメントを名前付きクエリーとして使用する場合、このオプションを true に指定する必要があります。		Boolean
usePassedInEntityManager (producer)	true に設定すると、Camel は、コンポーネント/エンドポイントで設定されたエンティティーマネージャーの代わりに、ヘッダー JpaConstants.ENTITYMANAGER の EntityManager を使用します。これにより、エンドユーザーはどのエンティティーマネージャーを使用するかを制御できます。	false	boolean
usePersist (producer)	entityManager.merge (entity) の代わりに entityManager.persist (entity) を使用することを示します。注記: entityManager.persist (entity) は、切り離されたエンティティー (EntityManager が INSERT クエリーの代わりに UPDATE を実行する必要がある場合) では機能しません!	false	boolean
entityManagerProperties (advanced)	エンティティーマネージャーが使用する追加のプロパティー。		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long

名前	説明	デフォルト	タイプ
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

172.5. メッセージヘッダー

Camel は、次のメッセージヘッダーをエクスチェンジに追加します。

ヘッダー	タイプ	説明
CamelJpaTemplate	**Jpa テンプレート**	Camel 2.12 **以降はサポートされていません:** エンティティー Bean へのアクセスに使用される **JpaTemplate** オブジェクト。このオブジェクトは、タイプコンバーターやカスタム処理を行う場合など、状況によっては必要になります。このヘッダーのサポートが中止された理由については、CAMEL-5932 を参照してください。
CamelEntityManager	**EntityManager**	Camel 2.12: JPA コンシューマー/Camel 2.12.2: JPA プロデューサー:**JpaConsumer** または **JpaProducer** によって使用される JPA **EntityManager** オブジェクト。

172.6. ENTITYMANAGERFACTORY の設定

特定の **EntityManagerFactory** インスタンスを使用するように JPA コンポーネントを設定することを強くお勧めします。そうしないと、各 **JpaEndpoint** が **EntityManagerFactory** の独自のインスタンスを自動作成しますが、これはほとんどの場合、必要なものではありません。

たとえば、次のように、**myEMFactory** エンティティーマネージャーファクトリーを参照する JPA コンポーネントをインスタンス化できます。

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
</bean>
```

Camel 2.3 では、**JpaComponent** はレジストリーから **EntityManagerFactory** を自動検索します。つまり、上記のように **JpaComponent** でこれを設定する必要はありません。あいまいさがある場合にのみそうする必要があります。その場合、Camel は WARN をログに記録します。

172.7. TRANSACTIONMANAGER の設定

Camel 2.3 以降、**JpaComponent** はレジストリーから **TransactionManager** を自動検索します。Camel が登録されている **TransactionManager** インスタンスを見つけられない場合、**TransactionTemplate** も検索し、そこから **TransactionManager** を抽出しようとします。

レジストリーで使用可能な **TransactionTemplate** がない場合、**JpaEndpoint** は **TransactionManager** の独自のインスタンスを自動的に作成しますが、これはほとんどの場合、必要なものではありません。

TransactionManager の複数のインスタンスが見つかった場合、Camel は WARN をログに記録します。このような場合、次のように、**myTransactionManager** トランザクションマネージャーを参照する JPA コンポーネントをインスタンス化し、明示的に設定することができます。

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
  <property name="transactionManager" ref="myTransactionManager"/>
</bean>
```

172.8. 名前付きクエリーでコンシューマーを使用する

選択したエンティティーのみを使用するには、**consumer.namedQuery** URI クエリーオプションを使用できます。まず、JPA Entity クラスで名前付きクエリーを定義する必要があります。

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
public class MultiSteps {
    ...
}
```

その後、次のようなコンシューマー uri を定義できます。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.namedQuery=step1")
.to("bean:myBusinessLogic");
```

172.9. クエリーでコンシューマーを使用する

選択したエンティティーのみを使用するには、**consumer.query** URI クエリーオプションを使用できます。クエリーオプションを定義するだけです。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1")
.to("bean:myBusinessLogic");
```

172.10. ネイティブクエリーでコンシューマーを使用する

選択したエンティティーのみを使用するには、**consumer.nativeQuery** URI クエリーオプションを使用できます。ネイティブクエリーオプションを定義するだけです。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.nativeQuery=select * from MultiSteps
where step = 1")
.to("bean:myBusinessLogic");
```

ネイティブクエリーオプションを使用すると、メッセージ本文でオブジェクト配列を受け取ります。

172.11. 名前付きクエリーでプロデューサーを使用する

選択したエンティティーを取得するか、一括更新/削除を実行するには、**namedQuery** URI クエリーオプションを使用できます。まず、JPA Entity クラスで名前付きクエリーを定義する必要があります。

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
public class MultiSteps {
    ...
}
```

その後、次のようなプロデューサー uri を定義できます。

```
from("direct:namedQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?namedQuery=step1");
```

UPDATE/DELETE ステートメントを名前付きクエリーとして実行するには、**useExecuteUpdate** オプションを **true** に指定する必要があることに注意してください。

172.12. クエリーでプロデューサーを使用する

選択したエンティティーを取得するか、一括更新/削除を実行するには、**クエリー** URI クエリーオプションを使用できます。クエリーオプションを定義するだけです。

```
from("direct:query")
.to("jpa://org.apache.camel.examples.MultiSteps?query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1");
```

172.13. ネイティブクエリーでプロデューサーを使用する

選択したエンティティーを取得するか、一括更新/削除を実行するには、**nativeQuery** URI クエリーオプションを使用できます。ネイティブクエリーオプションを定義するだけです。

```
from("direct:nativeQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?
resultClass=org.apache.camel.examples.MultiSteps&nativeQuery=select * from MultiSteps where
step = 1");
```

resultClass を指定せずにネイティブクエリーオプションを使用すると、メッセージ本文でオブジェクト配列を受け取ります。

172.14. 例

JPA を使用してトレースされたメッセージをデータベースに格納する例については、トレーサーの例を参照してください。

172.15. JPA ベースのべき等リポジトリーの使用

EIP パターン の Idempotent Consumer は、重複するメッセージを除外するために使用されます。JPA ベースのべき等リポジトリーが提供されます。

JPA ベースのべき等リポジトリーを使用するには。

手順

1. persistence.xml ファイルで **persistence-unit** を設定します。
2. **org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository** で使用される **org.springframework.orm.jpa.JpaTemplate** を設定します。
3. エラー形式のマクロを設定します: snippet: java.lang.IndexOutOfBoundsException: Index: 20、Size: 20
4. べき等リポジトリーを設定します: **org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository** :
5. Spring XML ファイルに JPA べき等リポジトリーを作成します。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
```

```
    <route id="JpaMessageIdRepositoryTest">
      <from uri="direct:start" />
      <idempotentConsumer messageIdRepositoryRef="jpaStore">
        <header>messageId</header>
        <to uri="mock:result" />
      </idempotentConsumer>
    </route>
</camelContext>
```

IDE 内でこの Camel コンポーネントテストを実行する場合

このコンポーネントのテスト を、Maven を介さずに IDE 内で直接実行すると、次のような例外が発生する可能性があります。

```
org.springframework.transaction.CannotCreateTransactionException: Could not open JPA
EntityManager for transaction; nested exception is
<openjpa-2.2.1-r422266:1396819 nonfatal user error>
org.apache.openjpa.persistence.ArgumentException: This configuration disallows runtime
optimization,
but the following listed types were not enhanced at build time or at class load time with a javaagent:
"org.apache.camel.examples.SendEmail".
    at
org.springframework.orm.jpa.JpaTransactionManager.doBegin(JpaTransactionManager.java:427)
    at
org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(Abstra
ctPlatformTransactionManager.java:371)
    at
org.springframework.transaction.support.TransactionTemplate.execute(TransactionTemplate.java:12
7)
    at org.apache.camel.processor.jpa.JpaRouteTest.cleanupRepository(JpaRouteTest.java:96)
    at org.apache.camel.processor.jpa.JpaRouteTest.createCamelContext(JpaRouteTest.java:67)
    at org.apache.camel.test.junit4.CamelTestSupport.doSetUp(CamelTestSupport.java:238)
    at org.apache.camel.test.junit4.CamelTestSupport.setUp(CamelTestSupport.java:208)
```

ここでの問題は、ソースが Maven ではなく IDE を介してコンパイルまたは再コンパイルされていることです。これにより、ビルド時にバイトコードが拡張 されます。これを克服するには、OpenJPA の動的バイトコード拡張 を有効にする必要があります。たとえば、Camel で使用されている現在の OpenJPA のバージョンが 2.2.1 であるとすると、IDE 内でテストを実行するには、次の引数を JVM に渡す必要があります。

```
-javaagent:<path_to_your_local_m2_cache>/org/apache/openjpa/openjpa/2.2.1/openjpa-2.2.1.jar
```

172.16. 関連項目

- Configuring Camel (Camel の設定)
- Component (コンポーネント)
- Endpoint (エンドポイント)
- スタートガイド
- トレーサーの例

第173章 JSON FASTJSON DATAFORMAT

Camel バージョン 2.20 以降で利用可能

Fastjson は、Fastjson ライブラリー を使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Fastjson).
  to("mqseries:Another.Queue");
```

173.1. FASTJSON オプション

JSON Fastjson データ形式は、以下に示す 19 のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
objectMapper		String	Jackson を使用する場合は、指定された ID で既存の ObjectMapper を検索して使用します。
useDefaultObjectMapper	true	Boolean	レジストリーからデフォルトの Jackson ObjectMapper を検索して使用するかどうか。
prettyPrint	false	Boolean	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		String	アンアームシャリング時に使用する Java 型のクラス名
jsonView		Class<?>	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		String	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。
allowJmsType	false	Boolean	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。
collectionTypeName		String	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。

名前	デフォルト	Java タイプ	説明
useList	false	Boolean	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	false	Boolean	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		String	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		String	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		String	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		String	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
permissions		String	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。

名前	デフォルト	Java タイプ	説明
allowUnmarshallType	**false**	**Boolean**	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
timezone		**String**	設定されている場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream など、他の Json DataFormat には影響を与えません。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

173.2. 依存関係

camel ルートで Fastjson を使用するには、このデータ形式を実装する **camel-fastjson** への依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-fastjson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第174章 JSON GSON DATAFORMAT

Camel バージョン 2.10 以降で利用可能

Gson は、Gson Library を使用するデータ形式です

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Gson).
  to("mqseries:Another.Queue");
```

174.1. GSON オプション

JSon GSon データ形式は、以下に示す 19 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
objectMapper		String	Jackson を使用する場合は、指定された ID で既存の ObjectMapper を検索して使用します。
useDefaultObjectMapper	true	Boolean	レジストリーからデフォルトの Jackson ObjectMapper を検索して使用するかどうか。
prettyPrint	false	Boolean	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		String	アンアームシャリング時に使用する Java 型のクラス名
jsonView		Class<?>	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		String	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。
allowJmsType	false	Boolean	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。
collectionTypeName		String	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。

名前	デフォルト	Java タイプ	説明
useList	**false**	**Boolean**	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	**false**	**Boolean**	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		**String**	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		**String**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
permissions		**String**	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。

名前	デフォルト	Java タイプ	説明
allowUnmarshallType	**false**	**Boolean**	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
timezone		**String**	設定されている場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream など、他の Json DataFormat には影響を与えません。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

174.2. 依存関係

camel ルートで Gson を使用するには、このデータ形式を実装する **camel-gson** への依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-gson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第175章 JSON JACKSON DATAFORMAT

Camel バージョン 2.0 以降で利用可能

Jackson は、Jackson Library を使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Jackson).
  to("mqseries:Another.Queue");
```

175.1. JACKSON オプション

JSon Jackson データ形式は、以下に示す 19 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
objectMapper		String	Jackson を使用する場合は、指定された ID で既存の ObjectMapper を検索して使用します。
useDefaultObjectMapper	true	Boolean	レジストリーからデフォルトの Jackson ObjectMapper を検索して使用するかどうか。
prettyPrint	false	Boolean	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		String	アンアームシャリング時に使用する Java 型のクラス名
jsonView		Class<?>	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		String	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。
allowJmsType	false	Boolean	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。
collectionTypeName		String	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。

名前	デフォルト	Java タイプ	説明
useList	**false**	**Boolean**	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	**false**	**Boolean**	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		**String**	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		**String**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
permissions		**String**	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。

名前	デフォルト	Java タイプ	説明
allowUnmarshallType	**false**	**Boolean**	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
timezone		**String**	設定されている場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream など、他の Json DataFormat には影響を与えません。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

175.2. カスタム OBJECTMAPPER の使用

マッピング設定をさらに制御する必要がある場合は、カスタムの **ObjectMapper** を使用するように **JacksonDataFormat** を設定できます。

レジストリーに単一の **ObjectMapper** を設定すると、Camel は自動検索してこの **ObjectMapper** を使用します。たとえば、Spring Boot を使用する場合、Spring MVC が有効になっている場合、Spring Boot はデフォルトの **ObjectMapper** を提供できます。これにより、Camel は、Spring Boot Bean レジストリーに **ObjectMapper** クラス型の Bean が 1 つあることを検出し、その Bean を使用します。このような場合には、Camel からの **INFO** ログを設定する必要があります。

175.3. 依存関係

camel ルートで Jackson を使用するには、このデータ形式を実装する **camel-jackson** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第176章 JSON JOHNZON DATAFORMAT

Camel バージョン 2.18 以降で利用可能

Johnzon は、Johnzon ライブラリー を使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Johnzon).
  to("mqseries:Another.Queue");
```

176.1. JOHNZON オプション

JSon Johnzon データ形式は、以下に示す 19 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
objectMapper		String	Jackson を使用する場合は、指定された ID で既存の ObjectMapper を検索して使用します。
useDefaultObjectMapper	true	Boolean	レジストリーからデフォルトの Jackson ObjectMapper を検索して使用するかどうか。
prettyPrint	false	Boolean	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		String	アンアームシャリング時に使用する Java 型のクラス名
jsonView		Class<?>	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		String	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。
allowJmsType	false	Boolean	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。
collectionTypeName		String	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。

名前	デフォルト	Java タイプ	説明
useList	**false**	**Boolean**	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	**false**	**Boolean**	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		**String**	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		**String**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
permissions		**String**	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。

名前	デフォルト	Java タイプ	説明
allowUnmarshallType	**false**	**Boolean**	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
timezone		**String**	設定されている場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream など、他の Json DataFormat には影響を与えません。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

176.2. 依存関係

camel ルートで Johnzon を使用するには、このデータ形式を実装する **camel-johnzon** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-johnzon</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第177章 JSON SCHEMA VALIDATOR コンポーネント

Camel バージョン 2.20 以降で利用可能

JSON Schema Validator コンポーネントは、NetworkNT JSON Schema ライブラリー (https://github.com/networknt/json-schema-validator) を使用して、JSON Schemas v4 ドラフトに対してメッセージ本文の Bean 検証を実行します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-json-validator</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

177.1. URI 形式

```
json-validator:resourceUri[?options]
```

ここで、**resourceUri** は、クラスパス上のローカルリソースへの URL、または検証対象の JSON スキーマを含むファイルシステム上のリモートリソースまたはリソースへの完全な URL です。

177.2. URI オプション

JSON Schema Validator コンポーネントにはオプションがありません。

JSON Schema Validator エンドポイントは、URI 構文を使用して設定されます。

```
json-validator:resourceUri
```

パスおよびクエリーパラメーターを使用します。

177.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

177.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
failOnNullBody (producer)	本文が存在しない場合に失敗するかどうか。	true	boolean
failOnNullHeader (producer)	ヘッダーに対して検証するときに、ヘッダーが存在しない場合に失敗するかどうか。	true	boolean
headerName (producer)	メッセージボディーではなくヘッダーに対して検証します。		String
errorHandler (advanced)	カスタム ValidatorErrorHandler を使用する場合。デフォルトのエラーハンドラーはエラーをキャプチャし、例外を出力します。		JsonValidatorError Handler
schemaLoader (advanced)	カスタムスキーマローダーを使用して、カスタム形式の検証を追加できるようにします。デフォルトの実装では、ドラフト v4 をサポートするスキーマローダーが作成されます。		JsonSchemaLoader
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

177.3. 例

次の JSON スキーマがあると仮定します

myschema.json

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {},
  "id": "my-schema",
  "properties": {
    "id": {
      "default": 1,
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/id",
      "title": "The id schema",
      "type": "integer"
    },
    "name": {
      "default": "A green door",
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/name",
      "title": "The name schema",
```

```
      "type": "string"
    },
    "price": {
      "default": 12.5,
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/price",
      "title": "The price schema",
      "type": "number"
    }
  },
  "required": [
    "name",
    "id",
    "price"
  ],
  "type": "object"
}
```

myschema.json が クラスパスから読み込まれる次の Camel ルートを使用して、入力 JSON を検証できます。

```
from("direct:start")
  .to("json-validator:myschema.json")
  .to("mock:end")
```

第178章 JSON XSTREAM DATAFORMAT

Camel バージョン 2.0 以降で利用可能

XStream は、XStream ライブラリー を使用して Java オブジェクトを XML との間でマーシャリングおよびアンマーシャリングするデータ形式です。

camel ルートで XStream を使用するには、このデータ形式を実装する **camel-xstream** に依存関係を追加する必要があります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xstream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

178.1. オプション

JSON XStream データ形式は、以下に示す 19 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
objectMapper		**String**	Jackson を使用する場合は、指定された ID で既存の ObjectMapper を検索して使用します。
useDefaultObjectMapper	**true**	**Boolean**	レジストリーからデフォルトの Jackson ObjectMapper を検索して使用するかどうか。
prettyPrint	**false**	**Boolean**	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
library	**XStream**	**JsonLibrary**	使用する json ライブラリー。
unmarshalTypeName		**String**	アンアームシャリング時に使用する Java 型のクラス名
jsonView		**Class<?>**	POJO を JSON にマーシャリングする際に、JSON 出力から特定のフィールドを除外する場合があります。Jackson では、JSON ビューを使用してこれを実現できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		**String**	pojo を JSON にマーシャリングする必要があり、pojo に null 値を持つフィールドがいくつかある場合。これらの null 値をスキップする場合は、このオプションを NOT_NULL に設定できます。

名前	デフォルト	Java タイプ	説明
allowJmsType	**false**	**Boolean**	JMS ユーザーが JMS 仕様の JMSType ヘッダーを使用して、アンマーシャリングに使用する FQN クラス名を指定できるようにするために使用されます。
collectionTypeName		**String**	使用するレジストリーを参照するカスタムコレクションタイプを参照します。このオプションはあまり使用しないでください。ただし、デフォルトとして java.util.Collection に基づくものとは異なるコレクションタイプを使用できます。
useList	**false**	**Boolean**	Map の List または Pojo の List にアンマーシャリングします。
enableJaxbAnnotationModule	**false**	**Boolean**	jackson の使用時に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson によって JAXB アノテーションを使用できます。
moduleClassNames		**String**	カスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用するには、FQN クラス名を持つ文字列として指定します。複数のクラスはコンマで区切ることができます。
moduleRefs		**String**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。
disableFeatures		**String**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、または com.fasterxml.jackson.databind.MapperFeature の列挙型と一致する名前である必要があります。複数の機能はコンマで区切ることができます。

名前	デフォルト	Java タイプ	説明
permissions		String	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。
allowUnmarshallType	false	Boolean	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する必要がある場合にのみ有効にする必要があります。
timezone		String	設定されている場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream など、他の Json DataFormat には影響を与えません。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

178.2. JAVA DSL を使用

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().xstream().
  to("mqseries:Another.Queue");
```

Camel がメッセージ変換に使用する **XStream** インスタンスを設定する場合は、DSL レベルでそのインスタンスへの参照を渡すだけです。

```
XStream xStream = new XStream();
xStream.aliasField("money", PurchaseOrder.class, "cash");
// new Added setModel option since Camel 2.14
xStream.setModel("NO_REFERENCES");
...
```

```
from("direct:marshal").
  marshal(new XStreamDataFormat(xStream)).
  to("mock:marshaled");
```

178.3. XMLINPUTFACTORY と XMLOUTPUTFACTORY

XStream ライブラリー は **javax.xml.stream.XMLInputFactory** と **javax.xml.stream.XMLOutputFactory** を使用します。このファクトリーのどの実装を使用するかを制御できます。

Factory は、次のアルゴリズムを使用して検出されます: 1.**javax.xml.stream.XMLInputFactory**、**javax.xml.stream.XMLOutputFactory** システムプロパティーを使用します。2.**JRE_HOME** ディレクトリーにある **lib/xml.stream.properties** ファイルを使用します。3.JRE で使用可能な jar 内の **META-INF/services/javax.xml.stream.XMLInputFactory**、**META-INF/services/javax.xml.stream.XMLOutputFactory** ファイルを調べて、可能であればサービス API を使用してクラス名を決定します。.4.プラットフォームのデフォルトの XMLInputFactory,XMLOutputFactory インスタンスを使用します。

178.4. XSTREAM DATAFORMAT で XML エンコーディングを設定するには?

Camel 2.2.0 から、キー **Exchange.CHARSET_NAME** を使用して Exchange のプロパティーを設定するか、DSL または Spring 設定から Xstream でエンコーディングプロパティーを設定することにより、Xstream DataFormat で XML のエンコーディングを設定できます。

```
from("activemq:My.Queue").
  marshal().xstream("UTF-8").
  to("mqseries:Another.Queue");
```

178.5. XSTREAM DATAFORMAT のタイプ権限の設定

Camel では、ルートで常に独自の処理ステップを使用して、XStream の unmarhall ステップにルーティングされる特定の XML ドキュメントをフィルタリングおよびブロックできます。Camel 2.16.1、2.15.5 から、XStream のタイプのパーミッション を設定して、特定のタイプのインスタンス化を自動的に許可または拒否できます。

Camel で使用されるデフォルトのタイプ権限設定は、java.lang および java.util パッケージのタイプを除くすべてのタイプを拒否します。この設定は、システムプロパティー org.apache.camel.xstream.permissions を設定することで変更できます。その値はコンマで区切られた許可条件の文字列であり、それぞれが許可または拒否されるタイプを表します。これは、用語の前に '' (注記 '' は省略される場合があります) または '-' がそれぞれ付けられているかどうかによって異なります。

各用語にはワイルドカード文字を含めることができます **''。たとえば、値 "-**,java.lang.,**java.util.**" は、java.lang.* および java.util.* クラスを除くすべてのタイプを拒否することを示します。この値を空の文字列 "" に設定すると、ブラックリストに登録された特定のクラスを拒否し、他のクラスを許可する、デフォルトの XStream の型パーミッション処理に戻ります。

タイプパーミッションの設定は、タイプパーミッションプロパティーを設定することにより、個々の XStream DataFormat インスタンスで拡張できます。

```
<dataFormats>
   <xstream id="xstream-default"
```

```
            permissions="org.apache.camel.samples.xstream.*"/>
    ...
```

第179章 JSONPATH 言語

Camel バージョン 2.13 以降で利用可能

Camel は JSonPath をサポートし、JSON メッセージで式または述語を使用できるようにします。

```
from("queue:books.new")
  .choice()
    .when().jsonpath("$.store.book[?(@.price < 10)]")
      .to("jms:queue:book.cheap")
    .when().jsonpath("$.store.book[?(@.price < 30)]")
      .to("jms:queue:book.average")
    .otherwise()
      .to("jms:queue:book.expensive")
```

179.1. JSONPATH オプション

JSonPath 言語は、以下にリストされている 7 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
resultType		**String**	結果の型 (出力からの型) のクラス名を設定します
suppressExceptions	**false**	**Boolean**	PathNotFoundException などの例外を抑制するかどうか。
allowSimple	**true**	**Boolean**	JSonPath 式でインライン化された単純な例外を許可するかどうか
allowEasyPredicate	**true**	**Boolean**	簡単な述語パーサーを使用して述語を事前解析できるようにするかどうか。
writeAsString	**false**	**Boolean**	各行/要素の出力をマップ/POJO 値ではなく JSON 文字列値として書き込むかどうか。
headerName		**String**	メッセージボディーの代わりに入力として使用するヘッダーの名前
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

179.2. XML 設定の使用

Spring XML ファイルでルートを設定する場合は、次のように JSonPath 式を使用できます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <choice>
```

```
      <when>
        <jsonpath>$.store.book[?(@.price &lt; 10)]</jsonpath>
        <to uri="mock:cheap"/>
      </when>
      <when>
        <jsonpath>$.store.book[?(@.price &lt; 30)]</jsonpath>
        <to uri="mock:average"/>
      </when>
      <otherwise>
        <to uri="mock:expensive"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
```

179.3. 構文

その他の例については、JSonPath プロジェクトページを参照してください。

179.4. 簡略化構文

Camel 2.19 以降で利用可能

jsonpath の構文を使って基本的な述語を定義する場合、構文を覚えるのが少し難しいかもしれません。たとえば、安価な書籍をすべて検索する場合は、以下を行う必要があります

```
$.store.book[?(@.price < 20)]
```

しかし、以下のように簡略化した記述もできます

```
store.book.price < 20
```

また、価格キーを持つノードを確認する場合は、パスを省略することもできます

```
price < 20
```

これをサポートするために、基本的なスタイルを使用して述語を定義した場合に起動する **EasyPredicateParser** があります。つまり、述語は **$** 記号で開始してはならず、演算子を 1 つだけ含める必要があります。

簡単な構文は次のとおりです。

```
left OP right
```

適切な演算子で Camel Simple 言語を使用できます。たとえば、次のようになります

```
store.book.price < ${header.limit}
```

179.5. サポートされるメッセージボディーのタイプ

Camel JSonPath は、以下のタイプのメッセージボディーをサポートしています。

タイプ	Comment
File	ファイルからの読み込み
String	プレーンテキスト
Map	**java.util.Map** 型としてのメッセージボディー
リスト	**java.util.List** 型としてのメッセージボディー
POJO	**任意** Jackson がクラスパス上にある場合、camel-jsonpath は Jackson を使用してメッセージボディーを POJO から **java.util.Map** (JSonPath でサポートされる) に変換の上、JSonPath で処理することができます。たとえば、依存関係として **camel-jackson** を追加して、Jackson を含めることができます。
InputStream	上記のタイプがどれも一致しない場合、Camel はメッセージボディーを **java.io.InputStream** 型で読み込みます。

メッセージボディーがサポートされないタイプの場合は、デフォルトでは例外が出力されますが、JSonPath の設定で例外を抑止できます (以下を参照)。

179.6. SUPPRESS EXCEPTIONS

Camel 2.16 以降で利用可能

デフォルトでは、設定された jsonpath 式に従って json ペイロードに有効なパスがない場合には、jsonpath は例外を出力します。ユースケースによっては、json ペイロードにオプションのデータが含まれている時にはこれを無視する場合があります。したがって、以下に示すように、オプション suppressExceptions を true に設定して、これを無視できます。

```
from("direct:start")
    .choice()
        // use true to suppress exceptions
        .when().jsonpath("person.middlename", true)
            .to("mock:middle")
        .otherwise()
            .to("mock:other");
```

そして XML DSL では:

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath suppressExceptions="true">person.middlename</jsonpath>
      <to uri="mock:middle"/>
    </when>
    <otherwise>
      <to uri="mock:other"/>
```

```
    </otherwise>
  </choice>
</route>
```

このオプションは、**@JsonPath** アノテーションでも使用できます。

179.7. インラインの単純な例外

Camel 2.18 から利用可能

単純な構文 ${xxx} を使用して、JSonPath 式で Simple 言語式をインライン化できるようになりました。以下に例を示します。

```
from("direct:start")
  .choice()
    .when().jsonpath("$.store.book[?(@.price < ${header.cheap})]")
      .to("mock:cheap")
    .when().jsonpath("$.store.book[?(@.price < ${header.average})]")
      .to("mock:average")
    .otherwise()
      .to("mock:expensive");
```

そして XML DSL では:

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath>$.store.book[?(@.price < ${header.cheap})]</jsonpath>
      <to uri="mock:cheap"/>
    </when>
    <when>
      <jsonpath>$.store.book[?(@.price < ${header.average})]</jsonpath>
      <to uri="mock:average"/>
    </when>
    <otherwise>
      <to uri="mock:expensive"/>
    </otherwise>
  </choice>
</route>
```

次のように、オプション allowSimple を false に設定して、インライン化された Simple 式のサポートを無効にすることができます。

```
.when().jsonpath("$.store.book[?(@.price < 10)]", false, false)
```

そして XML DSL では:

```
<jsonpath allowSimple="false">$.store.book[?(@.price < 10)]</jsonpath>
```

179.8. JSONPATH INJECTION

Bean インテグレーションを使用して Bean のメソッドを呼び出す場合、JSonPath (他の 言語も使用可) を使用してメッセージから値を抽出し、メソッドパラメーターにバインドすることができます。

以下に例を示します。

```
public class Foo {

    @Consume(uri = "activemq:queue:books.new")
    public void doSomething(@JsonPath("$.store.book[*].author") String author, @Body String json) {
      // process the inbound message here
    }
}
```

179.9. エンコーディング検出

Camel バージョン 2.16 以降、ドキュメントが RFC-4627 で指定されているように Unicode (UTF-8、UTF-16LE、UTF-16BE、UTF-32LE、UTF-32BE) でエンコードされている場合に、JSON ドキュメントのエンコードは自動的に検出されます。エンコーディングが非 Unicode エンコーディングの場合は、ドキュメントを文字列形式で JSONPath コンポーネントに入力するか、ヘッダー "**CamelJsonPathJsonEncoding**" (JsonpathConstants.HEADER_JSON_ENCODING) でエンコーディングを指定することができます。

179.10. JSON データを JSON としてサブ行に分割する

jsonpath を使用して、次のように JSon ドキュメントを分割できます。

```
from("direct:start")
    .split().jsonpath("$.store.book[*]")
    .to("log:book");
```

次に、各書籍がログに記録されますが、メッセージボディーは **Map** インスタンスです。**Camel 2.20** 以降では、次のように **writeAsString** オプションを使用して実行できます。

```
from("direct:start")
    .split().jsonpathWriteAsString("$.store.book[*]")
    .to("log:book");
```

次に、各書籍が String JSon 値としてログに記録されます。以前のバージョンの Camel では、camel-jackson データ形式を使用し、メッセージ本文をマーシャリングして、メッセージ本文を **Map** から **String** 型に変換する必要がありました。

179.11. ヘッダーの入力としての使用

Camel 2.20 以降で利用可能

デフォルトでは、jsonpath はメッセージボディーを入力ソースとして使用します。ただし、**headerName** オプションを指定して、ヘッダーを入力として使用することもできます。

たとえば、**books** という名前のヘッダーに保存された json ドキュメントから書籍数をカウントするには、次のようにします。

```
from("direct:start")
    .setHeader("numberOfBooks")
```

```
        .jsonpath("$..store.book.length()", false, int.class, "books")
    .to("mock:result");
```

上記の **jsonpath** 式では、ヘッダーの名前を **books** として指定し、結果を **int.class** で整数として変換することも指示しました。

XML DSL での同じ例は次のようになります。

```
<route>
  <from uri="direct:start"/>
  <setHeader headerName="numberOfBooks">
    <jsonpath headerName="books" resultType="int">$..store.book.length()</jsonpath>
  </transform>
  <to uri="mock:result"/>
</route>
```

179.12. 依存関係

camel ルートで JSonPath を使用するには、JSonPath 言語を実装する **camel-jsonpath** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jsonpath</artifactId>
  <version>x.x.x</version>
</dependency>
```

第180章 JT400 コンポーネント

Camel バージョン 1.5 以降で利用可能

jt400 コンポーネントを使用すると、データ待ち行列を使用して AS/400 システムとメッセージをエクスチェンジできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jt400</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

180.1. URI 形式

```
jt400://user:password@system/QSYS.LIB/LIBRARY.LIB/QUEUE.DTAQ[?options]
```

リモートプログラムを呼び出すには (**Camel 2.7**)

```
jt400://user:password@system/QSYS.LIB/LIBRARY.LIB/program.PGM[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

180.2. JT400 オプション

JT400 コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionPool (advanced)	このコンポーネントが使用するデフォルトの接続プールを返します。		AS400ConnectionPool
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JT400 エンドポイントは、URI 構文を使用して設定されます。

```
jt400:userID:password/systemName/objectPath.type
```

パスおよびクエリーパラメーターを使用します。

180.2.1. パスパラメーター (5 個のパラメーター)

名前	説明	デフォルト	タイプ
userID	**必須** AS/400 ユーザーの ID を返します。		String
password	**必須** AS/400 ユーザーのパスワードを返します。		String
システム名	**必須** AS/400 システムの名前を返します。		String
objectPath	**必須** このエンドポイントのターゲットオブジェクトの完全修飾統合ファイルシステムパス名を返します。		String
type	**必須** データ待ち行列またはリモートプログラム呼び出しを処理するかどうか		Jt400Type

180.2.2. クエリーパラメーター (30 パラメーター)

名前	説明	デフォルト	タイプ
ccsid (Common)	AS/400 システムとの接続に使用する CCSID を設定します。		int
format (Common)	メッセージを送信するためのデータ形式を設定します。	text	形式
guiAvailable (Common)	Camel を実行している環境で AS/400 プロンプトを有効にするかどうかを設定します。	false	boolean
keyed (Common)	キー付きデータ待ち行列またはキーなしデータ待ち行列のどちらを使用するか。	false	boolean
outputFieldsIdxArray (common)	どのフィールド (プログラムパラメーター) が出力パラメーターであるかを指定します。		Integer[]
outputFieldsLengthArray (common)	AS/400 プログラム定義と同様に、フィールド (プログラムパラメーター) の長さを指定します。		Integer[]
searchKey (Common)	キー付きデータ待ち行列の検索キー。		String
searchType (Common)	EQ for equal などの検索タイプ	EQ	SearchType

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
readTimeout (consumer)	コンシューマーがデータキューの新しいメッセージを読み取ろうとして待機するタイムアウト (ミリ秒単位)。	30000	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

名前	説明	デフォルト	タイプ
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit

名前	説明	デフォルト	タイプ
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
procedureName (procedureName)	サービスプログラムから呼び出す手続き名		String
secured (security)	AS/400 への接続が SSL で保護されているかどうか。	false	boolean

180.3. 使用方法

コンシューマーエンドポイントとして設定されている場合、エンドポイントはリモートシステムのデータキューをポーリングします。データキューのすべてのエントリーに対して、**In** メッセージのボディーにエントリーのデータが格納された新しい **Exchange** が送信され、形式に応じて **String** または **byte[]** としてフォーマットされます。プロバイダーエンドポイントの場合、**In** メッセージボディーの内容は、未加工のバイトまたはテキストとしてデータキューに配置されます。

180.4. 接続プール

Camel 2.10 以降で利用可能

接続プーリングは Camel 2.10 以降で使用されています。Jt400Component で接続プールを明示的に設定するか、エンドポイントで uri オプションとして設定できます。

180.4.1. リモートプログラムコール (Camel 2.7)

このエンドポイントは、入力が String 配列または byte[] 配列 (形式に応じて) であると想定し、ネイティブの jt400 ライブラリーメカニズムを介してすべての CCSID 処理を処理します。パラメーターは、その位置の値として null を渡すことで **省略** できます (リモートプログラムはそれをサポートする必要があります)。プログラムの実行後、エンドポイントは、プログラムによって返された値を含む String 配列または byte[] 配列を返します (入力のみのパラメーターには、呼び出しの開始時と同じデータが含まれます)。このエンドポイントはプロバイダーエンドポイントを実装していません!

180.5. 例

以下のスニペットでは、**direct:george** エンドポイントに送信されるエクスチェンジのデータは、**LIVERPOOL** という名前のシステム上のライブラリー **BEATLES** 内のデータキュー **PENNYLANE** に配置されます。
別のユーザーが同じデータキューに接続して、データキューから情報を受け取り、**mock:ringo** エンドポイントに転送します。

```
public class Jt400RouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {

from("direct:george").to("jt400://GEORGE:EGROEG@LIVERPOOL/QSYS.LIB/BEATLES.LIB/PENNYLANE.DTAQ");
```

```
from("jt400://RINGO:OGNIR@LIVERPOOL/QSYS.LIB/BEATLES.LIB/PENNYLANE.DTAQ").to("mock
:ringo");
    }
}
```

180.5.1. リモートプログラム呼び出しの例 (Camel 2.7)

以下のスニペットでは、Exchange が direct:work エンドポイントに送信するデータに 3 つの文字列が含まれます。これらの文字列は、ライブラリー assets 内のプログラム compute の引数として使用されます。このプログラムは、出力値を 2 番目と 3 番目のパラメーターに書き込みます。すべてのパラメーターが direct:play エンドポイントに送信されます。

```
public class Jt400RouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
       from("direct:work").to("jt400://GRUPO:ATWORK@server/QSYS.LIB/assets.LIB/compute.PGM?
fieldsLength=10,10,512&ouputFieldsIdx=2,3").to(“direct:play”);
    }
}
```

180.5.2. キー付きデータ待ち行列への書き込み

```
from("jms:queue:input")
.to("jt400://username:password@system/lib.lib/MSGINDQ.DTAQ?keyed=true");
```

180.5.3. キー付きデータ待ち行列からの読み取り

```
from("jt400://username:password@system/lib.lib/MSGOUTDQ.DTAQ?
keyed=true&searchKey=MYKEY&searchType=GE")
.to("jms:queue:output");
```

180.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第181章 KAFKA コンポーネント

Camel バージョン 2.13 以降で利用可能

kafka: コンポーネントは、Apache Kafka メッセージブローカーとの通信に使用されます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kafka</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

181.1. URI 形式

```
kafka:topic[?options]
```

181.2. オプション

Kafka コンポーネントは、以下にリストされている 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	エンドポイントが再利用する共通オプションを使用して、Kafka コンポーネントを事前設定できます。		KafkaConfiguration
brokers (Common)	使用する Kafka ブローカーの URL。形式は host1:port1,host2:port2 で、リストはブローカーのサブセットまたはブローカーのサブセットを指す VIP にすることができます。このオプションは、Kafka ドキュメントでは bootstrap.servers として知られています。		String
workerPool (advanced)	共有カスタムワーカープールを使用して、kafka サーバーが非同期のノンブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、Exchange のルーティングを続行します。このオプションを使用する場合は、スレッドプールのライフサイクルを処理して、不要になったときにプールをシャットダウンする必要があります。		ExecutorService
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean

名前	説明	デフォルト	タイプ
breakOnFirstError (consumer)	このオプションは、コンシューマーが交換を処理していて失敗した場合に何が起こるかを制御します。オプションが false の場合、コンシューマーは次のメッセージに進み、それを処理します。オプションが true の場合、コンシューマーは中断し、失敗の原因となったメッセージのオフセットに戻ってシークし、このメッセージの処理を再試行します。ただし、これは、たとえば有害なメッセージのように毎回失敗する場合、同じメッセージの無限の処理につながる可能性があります。したがって、Camel のエラーハンドラーを使用するなどして対処することをお勧めします。	false	boolean
allowManualCommit (consumer)	KafkaManualCommit による手動コミットを許可するかどうか。このオプションを有効にすると、KafkaManualCommit のインスタンスが Exchange メッセージヘッダーに格納されます。これにより、エンドユーザーはこの API にアクセスし、Kafka コンシューマーを介して手動でオフセットコミットを実行できます。	false	boolean
kafkaManualCommit Factory (consumer)	KafkaManualCommit インスタンスの作成に使用するファクトリー。これにより、カスタムファクトリーをプラグインしてカスタム KafkaManualCommit インスタンスを作成できます。これは、すぐに使用できるデフォルトの実装から逸脱する手動コミットを行うときに特別なロジックが必要な場合に備えています。		KafkaManualCommit ファクトリー
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Kafka エンドポイントは、URI 構文を使用して設定されます。

```
kafka:topic
```

パスおよびクエリーパラメーターを使用します。

181.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
topic	**必須** 使用するトピックの名前。コンシューマーでは、コンマを使用して複数のトピックを区切ることができます。プロデューサーは、1 つのトピックにのみメッセージを送信できます。		文字列

181.2.2. クエリーパラメーター (93 個のパラメーター):

名前	説明	デフォルト	タイプ
brokers (Common)	使用する Kafka ブローカーの URL。形式は host1:port1,host2:port2 で、リストはブローカーのサブセットまたはブローカーのサブセットを指す VIP にすることができます。このオプションは、Kafka ドキュメントでは bootstrap.servers として知られています。		String
clientId (common)	クライアント ID は、呼び出しの追跡に役立つように、各要求で送信されるユーザー指定の文字列です。要求を行っているアプリケーションを論理的に識別する必要があります。		String
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
reconnectBackoffMaxMs (common)	接続に繰り返し失敗したブローカーへの再接続時に待機する最大時間 (ミリ秒単位)。これが指定されている場合、ホストごとのバックオフは、連続して接続に失敗するたびに、この最大値まで指数関数的に増加します。バックオフの増加を計算した後、コネクションストームを回避するために 20% のランダムなジッターが追加されます。	1000	Integer
allowManualCommit (consumer)	KafkaManualCommit による手動コミットを許可するかどうか。このオプションを有効にすると、KafkaManualCommit のインスタンスが Exchange メッセージヘッダーに格納されます。これにより、エンドユーザーはこの API にアクセスし、Kafka コンシューマーを介して手動でオフセットコミットを実行できます。	false	boolean
autoCommitEnable (consumer)	true の場合、コンシューマーによってすでにフェッチされているメッセージのオフセットを ZooKeeper に定期的にコミットします。このコミットされたオフセットは、プロセスが失敗したときに、新しいコンシューマーが開始される位置として使用されます。	true	Boolean

名前	説明	デフォルト	タイプ
autoCommitIntervalMs (consumer)	コンシューマーオフセットが Zookeeper にコミットされるミリ秒単位の頻度。	5000	Integer
autoCommitOnStop (consumer)	コンシューマーが停止したときに明示的な自動コミットを実行して、ブローカーが最後に消費されたメッセージからコミットされていることを確認するかどうか。これには、autoCommitEnable オプションをオンにする必要があります。可能な値は、sync、async、または none です。sync がデフォルト値です。	sync	String
autoOffsetReset (consumer)	ZooKeeper に初期オフセットがない場合、またはオフセットが範囲外の場合の対処方法: 最小 : オフセットを最小オフセットに自動的にリセットする 最大 : オフセットを最大オフセットに自動的にリセットする 失敗: コンシューマーに例外を出力する	latest	String
breakOnFirstError (consumer)	このオプションは、コンシューマーが交換を処理していて失敗した場合に何が起こるかを制御します。オプションが false の場合、コンシューマーは次のメッセージに進み、それを処理します。オプションが true の場合、コンシューマーは中断し、失敗の原因となったメッセージのオフセットに戻ってシークし、このメッセージの処理を再試行します。ただし、これは、たとえば有害なメッセージのように毎回失敗する場合、同じメッセージの無限の処理につながる可能性があります。したがって、Camel のエラーハンドラーを使用するなどして対処することをお勧めします。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
checkCrcs (consumer)	消費されたレコードの CRC32 を自動的に確認します。これにより、メッセージのネットワーク上またはディスク上の破損が発生しなくなります。このチェックはオーバーヘッドを追加するため、極端なパフォーマンスを求める場合は無効になる可能性があります。	true	Boolean

名前	説明	デフォルト	タイプ
consumerRequestTimeoutMs (consumer)	この設定は、クライアントの要求の応答を待つ最大時間を制御します。タイムアウトが経過する前に応答が受信されない場合、クライアントは必要に応じてリクエストを再送信します。または、再試行が使い切られるとリクエストが失敗します。	40000	Integer
consumersCount (consumer)	kafka サーバーに接続するコンシューマーの数	1	int
consumerStreams (consumer)	コンシューマー上の同時コンシューマーの数	10	int
fetchMaxBytes (consumer)	サーバーがフェッチ要求に対して返す必要があるデータの最大量。これは絶対的な最大値ではありません。フェッチの最初の空でないパーティションの最初のメッセージがこの値よりも大きい場合でも、メッセージが返されて、コンシューマーは進歩することができます。ブローカーが受け入れる最大メッセージサイズは、message.max.bytes (ブローカー設定) または max.message.bytes (トピック設定) で定義されます。コンシューマーは複数のフェッチを並行して実行することに注意してください。	52428800	Integer
fetchMinBytes (consumer)	サーバーがフェッチ要求に対して返す必要のあるデータの最小量。利用可能なデータが不十分な場合、リクエストは、リクエストに応答する前に、十分なデータが蓄積されるのを待ちます。	1	Integer
fetchWaitMaxMs (consumer)	すぐに fetch.min.bytes を満たすのに十分なデータがない場合に、サーバーがフェッチ要求に応答する前にブロックする最大時間	500	Integer
groupId (consumer)	このコンシューマーが属するコンシューマープロセスのグループを一意に識別する文字列。同じグループ ID を設定することにより、複数のプロセスはそれらがすべて同じコンシューマーグループの一部であることを示します。このオプションは、コンシューマーに必要です。		String

名前	説明	デフォルト	タイプ
heartbeatIntervalMs (consumer)	Kafka のグループ管理機能を使用する場合の、ハートビートからコンシューマーコーディネーター間の想定される時間。ハートビートは、コンシューマーのセッションがアクティブな状態を維持し、新しいコンシューマーがグループに参加したり離脱したりする際のリバランスを促進するために使用されます。この値は session.timeout.ms よりも低く設定する必要がありますが、通常はその値の 1/3 以下に設定する必要があります。さらに低く調整することで、通常のリバランスの予想時間を制御することもできます。	3000	Integer
kafkaHeaderDeserializer (consumer)	デシリアライゼーション用の KafkaKafkaHeaderDeserializer 値をキャメルヘッダー値に設定します。		KafkaHeaderDeserializer
keyDeserializer (consumer)	Deserializer インターフェイスを実装する key の Deserializer クラス。	org.apache.kafka.common.serialization.StringDeserializer	String
maxPartitionFetchBytes (consumer)	サーバーが返すパーティションごとのデータの最大量。リクエストに使用される最大合計メモリーは partitions max.partition.fetch.bytes になります。このサイズは、少なくともサーバーが許可する最大メッセージサイズと同じである必要があります。そうしないと、プロデューサーがコンシューマーがフェッチできるよりも大きなメッセージを送信する可能性があります。その場合、コンシューマーは特定のパーティションで大きなメッセージを取得しようとしてスタックする可能性があります。	1048576	Integer
maxPollIntervalMs (consumer)	コンシューマーグループ管理を使用する場合の poll() の呼び出し間の最大遅延。これにより、コンシューマーがさらにレコードをフェッチする前にアイドル状態になることができる時間に上限が設定されます。このタイムアウトの期限が切れる前に poll() が呼び出されない場合、コンシューマーは失敗とみなされ、グループはパーティションを別のメンバーに再割り当てするためにリバランスします。		Long
maxPollRecords (consumer)	poll() への単一の呼び出しで返される最大レコード数	500	Integer

名前	説明	デフォルト	タイプ
offsetRepository (consumer)	トピックの各パーティションのオフセットをローカルに保存するために使用するオフセットリポジトリー。定義すると、自動コミットが無効になります。		String>
partitionAssignor (consumer)	グループ管理が使用されている場合に、クライアントがコンシューマーインスタンス間でパーティションの所有権を分散するために使用するパーティション割り当て戦略のクラス名	org.apache.kafka.clients.consumer.RangeAssignor	String
pollTimeoutMs (consumer)	KafkaConsumer をポーリングするときに使用されるタイムアウト。	5000	Long
seekTo (consumer)	KafkaConsumer が起動時に最初または最後から読み取るかどうかを設定します。begin : 最初から読み取る end : 最後から読み取るこれは、以前のプロパティー seekToBeginning を置き換えています		String
sessionTimeoutMs (consumer)	Kafka のグループ管理機能を使用するときに障害を検出するために使用されるタイムアウト。	10000	Integer
topicIsPattern (consumer)	トピックがパターン (正規表現) であるかどうか。これを使用して、パターンに一致する動的な数のトピックをサブスクライブできます。	false	boolean
valueDeserializer (consumer)	Deserializer インターフェイスを実装する値の Deserializer クラス。	org.apache.kafka.common.serialization.StringDeserializer	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
bridgeEndpoint (producer)	オプションが true の場合、KafkaProducer は受信メッセージの KafkaConstants.TOPIC ヘッダー設定を無視します。	false	boolean
bufferMemorySize (producer)	プロデューサーが、サーバーへの送信を待機しているレコードをバッファーリングするために使用できるメモリーの合計バイト数。レコードがサーバーに配信されるよりも速く送信された場合、プロデューサはブロックするか、block.on.buffer.full で指定された設定に基づいて例外を出力します。この設定は、プロデューサが使用する合計メモリーにほぼ対応する必要があります。ただし、プロデューサが使用するすべてのメモリーがバッファーリングに使用されるわけではないため、ハードバウンドではありません。一部の追加メモリーは、圧縮 (圧縮が有効な場合) やインフライトリクエストの維持に使用されます。	33554432	Integer
circularTopicDetection (producer)	オプションが true の場合、KafkaProducer は、メッセージが kafka コンシューマーからのオリジナルである場合、メッセージが送信元と同じトピックに送り返されようとしているかどうかを検出します。KafkaConstants.TOPIC ヘッダーが元の kafka コンシューマートピックと同じである場合、ヘッダー設定は無視され、プロデューサーエンドポイントのトピックが使用されます。つまり、これにより、同じメッセージが送信元に送り返されるのを回避できます。オプション bridgeEndpoint が true に設定されている場合、このオプションは使用されません。	true	boolean
compressionCodec (producer)	このパラメーターを使用すると、このプロデューサーによって生成されるすべてのデータの圧縮コーデックを指定できます。有効な値は none、gzip、snappy です。	none	String
connectionMaxIdleMs (producer)	この設定で指定された期間 (ミリ秒単位) の後にアイドル状態の接続を閉じます。	540000	Integer

名前	説明	デフォルト	タイプ
enableIdempotence (producer)	'true' に設定すると、プロデューサーは、プロデューサーは各メッセージのコピーが 1 つだけストリームに書き込まれるようにします。false の場合、プロデューサの再試行により、再試行されたメッセージの複製がストリームに書き込まれる可能性があります。true に設定した場合、このオプションでは max.in.flight.requests.per.connection を 1 に設定する必要があり、再試行をゼロにすることはできず、さらに ack を all に設定する必要があります。	false	boolean
kafkaHeaderSerializer (producer)	シリアル化 camel ヘッダー値のカスタム KafkaHeaderDeserializer を kafka ヘッダー値に設定します。		KafkaHeaderSerializer
key (producer)	レコードキー (キーが指定されていない場合は null)。このオプションが設定されている場合は、ヘッダーリンク KafkaConstantsKEY よりも優先されます。		文字列
keySerializerClass (producer)	キーのシリアライザクラス (何も指定されていない場合、デフォルトはメッセージと同じになります)。	org.apache.kafka.common.serialization.StringSerializer	String

名前	説明	デフォルト	タイプ
lingerMs (producer)	プロデューサーは、リクエストの送信の間に到着したレコードを 1 つのバッチリクエストにグループ化します。通常、これは、レコードが送信できるよりも早く到着した場合に、負荷がかかった状態でのみ発生します。ただし、状況によっては、中程度の負荷がかかっている場合でも、クライアントがリクエストの数を減らす場合があります。この設定は、少量の人為的な遅延を追加することでこれを実現します。つまり、レコードをすぐに送信するのではなく、プロデューサーは指定された遅延まで待機して他のレコードを送信できるようにし、送信をまとめてバッチ処理できるようにします。これは、TCP の Nagle アルゴリズムに類似するものと考えることができます。この設定は、バッチ処理の遅延の上限を提供します。あるパーティションで batch.size 相当のレコードを取得すると、この設定に関係なくすぐに送信されますが、このパーティションで蓄積されたバイト数がこれより少ない場合は、指定された時間の間、さらにレコードが取得されるのを待つことになります。デフォルトは 0 (つまり遅延なし) に設定されます。たとえば、linger.ms=5 を設定すると、送信されるリクエストの数が減りますが、負荷がない状態で送信されるレコードに最大 5 ミリ秒のレイテンシーが追加されます。	0	Integer
maxBlockMs (producer)	設定は、kafka への送信がブロックされる時間を制御します。これらのメソッドは、複数の理由でブロックされる可能性があります。例: バッファーがいっぱい、メタデータが利用できない。この設定では、メタデータのフェッチ、キーと値のシリアル化、send () を実行するときのバッファーメモリーの分割と割り当てに費やされる合計時間に最大制限が課されます。partitionsFor () の場合、この設定はメタデータの待機に最大時間のしきい値を課します	60000	Integer
maxInFlightRequest (producer)	ブロックする前にクライアントが 1 つの接続で送信する確認されていないリクエストの最大数。この設定が 1 より大きい値に設定されていて、送信に失敗した場合、再試行によりメッセージの順序が変更されるリスクがあることに注意してください (つまり、再試行が有効になっている場合)。	5	Integer
maxRequestSize (producer)	リクエストの最大サイズ。これは事実上、最大レコードサイズの上限でもあります。サーバーには、これとは異なる場合がある独自のレコードサイズの上限があることに注意してください。この設定により、プロデューサーが 1 回のリクエストで送信するレコードバッチの数が制限され、大量のリクエストが送信されないようになります。	1048576	Integer

名前	説明	デフォルト	タイプ
metadataMaxAgeMs (producer)	新しいブローカーまたはパーティションをプロアクティブに検出するためのパーティションリーダーシップの変更がない場合でも、メタデータの更新を強制するまでの期間 (ミリ秒単位)。	300000	Integer
metricReporters (producer)	メトリクスレポーターとして使用するクラスの一覧。MetricReporter インターフェイスを実装すると、新しいメトリックの作成が通知されるクラスをプラグインできます。JmxReporter は、JMX 統計を登録するために常に含まれます。		String
metricsSampleWindowMs (producer)	メトリクスを計算するために保持されるサンプルの数。	30000	Integer
noOfMetricsSample (producer)	メトリクスを計算するために保持されるサンプルの数。	2	Integer
partitioner (producer)	サブトピック間でメッセージを分割するパーティショナークラス。デフォルトのパーティショナーは、キーのハッシュに基づいています。	org.apache.kafka.clients.producer.internals.DefaultPartitioner	String
partitionKey (producer)	レコードの送信先のパーティション (パーティションが指定されていない場合は null)。このオプションが設定されている場合は、ヘッダーリンク KafkaConstantsPARTITION_KEY よりも優先されます		Integer

名前	説明	デフォルト	タイプ
producerBatchSize (producer)	複数のレコードが同じパーティションに送信されるときは常に、プロデューサーはレコードをまとめてより少ない要求にバッチ処理しようとします。これにより、クライアントとサーバーの両方でパフォーマンスが向上します。この設定では、デフォルトのバッチサイズをバイト単位で制御します。このサイズより大きいバッチレコードは試行されません。ブローカーに送信されるリクエストには、複数のバッチが含まれ、送信可能なデータを含む各パーティションに 1 つずつ含まれます。バッチサイズが小さいと、バッチ処理が一般的ではなくなり、スループット (バッチサイズゼロの場合、バッチ処理が完全に無効になります)。バッチサイズが非常に大きい場合は、追加のレコードを想定して、常に指定のバッチサイズのバッファーを割り当てるため、メモリーを多少無駄に使用する可能性があります。	16384	Integer
queueBufferingMaxMessages (producer)	プロデューサをブロックするか、データを削除する前に、非同期モードを使用するときにプロデューサのキューに入れることができる未送信メッセージの最大数。	10000	Integer
receiveBufferBytes (producer)	データの読み取り時に使用する TCP 受信バッファー (SO_RCVBUF) のサイズ。	65536	Integer
reconnectBackoffMs (producer)	特定のホストへの再接続を試行するまでの待機時間。これにより、タイトなループでホストに繰り返し接続することを回避します。このバックオフは、コンシューマーからブローカーに送信されるすべてのリクエストに適用されます。	50	Integer
recordMetadata (producer)	プロデューサが Kafka への送信から RecordMetadata の結果を保存する必要があるかどうか。結果は、RecordMetadata メタデータを含む List に保存されます。リストは、キーリンク KafkaConstantsKAFKA_RECORDMETA を持つヘッダーに格納されます。	true	boolean

名前	説明	デフォルト	タイプ
requestRequiredAcks (producer)	リクエストが完了したと見なす前に、プロデューサーがリーダーに受け取ったことを要求する確認の数。これは、送信されるレコードの耐久性を制御します。次の設定が一般的です: acks=0 ゼロに設定すると、プロデューサはサーバーからの確認をまったく待ちません。レコードは直ちにソケットバッファーに追加され、送信済みと見なされます。この場合、サーバーがレコードを受信したかどうかは保証されず、retries の設定は有効になりません (クライアントは通常、失敗を知ることができないからです)。各レコードに返されるオフセットは常に -1 に設定されます。acks=1 これは、リーダーがレコードをローカルログに書き込みますが、すべてのフォロワーからの完全な承認を待たずに応答することを意味します。この場合、レコードを承認した直後にリーダーが失敗したが、フォロワーがそれを複製する前に、レコードは失われます。acks=all これは、リーダーが同期レプリカの完全なセットがレコードを承認するまで待機することを意味します。これにより、少なくとも 1 つの In-Sync レプリカが動作している限り、レコードが失われないことが保証されます。これは利用可能な最強の保証になります。	1	String
requestTimeoutMs (producer)	クライアントにエラーを返す前に、ブローカーが request.required.acks 要件を満たすために待機する時間。	305000	Integer
retries (producer)	ゼロより大きい値を設定すると、クライアントは、一時的なエラーの可能性により送信に失敗したレコードを再送信します。この再試行は、クライアントがエラーを受信したときにレコードを再送した場合と同じであることに注意してください。再試行を許可すると、レコードの順序が変更される可能性があります。これは、2 つのレコードが 1 つのパーティションに送信され、最初のレコードが失敗して再試行され、2 番目のレコードが成功した場合、2 番目のレコードが最初に表示される可能性があるためです。	0	Integer
retryBackoffMs (producer)	各再試行の前に、プロデューサーは関連するトピックのメタデータを更新して、新しいリーダーが選出されたかどうかを確認します。リーダーの選択には少し時間がかかるため、このプロパティーはプロデューサーがメタデータを更新するまで待機する時間を指定します。	100	Integer
sendBufferBytes (producer)	ソケット書き込みバッファーサイズ	131072	Integer

名前	説明	デフォルト	タイプ
serializerClass (producer)	メッセージのシリアライザクラス。	org.apache.kafka.common.serialization.StringSerializer	String
workerPool (producer)	カスタムワーカープールを使用して、kafka サーバーが非同期のノンブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、Exchange のルーティングを続行します。		ExecutorService
workerPoolCoreSize (producer)	Kafka サーバーが非同期のノンブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、Exchange のルーティングを続行するためのワーカープールのコアスレッドの数。	10	Integer
workerPoolMaxSize (producer)	Kafka サーバーが、非同期のノンブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、Exchange のルーティングを続行するためのワーカープールのスレッドの最大数。	20	Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
interceptorClasses (monitoring)	プロデューサーまたはコンシューマーのインターセプターを設定します。Producer インターセプターは org.apache.kafka.clients.producer.ProducerInterceptor を実装するクラスでなければなりません Consumer インターセプターは org.apache.kafka.clients.consumer.ConsumerInterceptor を実装するクラスでなければなりません実行時のクラスキャスト例外		String
kerberosBeforeReloginMin Time (security)	更新試行間のログインスレッドのスリープ時間。	60000	Integer
kerberosInitCmd (security)	Kerberos kinit コマンドパス。デフォルトは/usr/bin/kinit です	/usr/bin/kinit	String

名前	説明	デフォルト	タイプ
kerberosPrincipalToLocal Rules (security)	プリンシパル名から短縮名 (通常はオペレーティングシステムのユーザー名) にマッピングするためのルールの一覧です。ルールは順番に評価され、プリンシパル名と一致する最初のルールは、これを短縮名にマップするために使用されます。一覧の後続のルールは無視されます。デフォルトでは、username/hostnameREALM 形式のプリンシパル名は username にマッピングされます。形式の詳細は、security authorization および acls を参照してください。複数の値はコンマで区切ることができます	DEFAULT	String
kerberosRenewJitter (security)	更新時間に追加されたランダムなジッターの割合。	0.05	double
kerberosRenewWindowFactor (security)	ログインスレッドは、最後の更新からチケットの有効期限までの指定された時間のウィンドウファクターに達するまでスリープし、その時点でチケットの更新を試みます。	0.8	double
saslJaasConfig (security)	kafka sasl.jaas.config パラメーターを公開します。例: org.apache.kafka.common.security.plain.PlainLoginModule required username=USERNAME password=PASSWORD;		String
saslKerberosServiceName (security)	Kafka が実行される Kerberos プリンシパル名。これは、Kafka の JAAS 設定または Kafka の設定で定義できます。		String
saslMechanism (security)	使用される Simple Authentication and Security Layer(SASL) メカニズム。有効な値については、http://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xhtml を参照してください。	GSSAPI	String
securityProtocol (security)	ブローカーとの通信に使用されるプロトコル。現在、PLAINTEXT と SSL のみがサポートされています。	PLAINTEXT	String
sslCipherSuites (security)	暗号化スイートの一覧。これは、TLS または SSL ネットワークプロトコルを使用してネットワーク接続のセキュリティー設定をネゴシエートするために使用される、認証、暗号化、MAC、およびキー交換アルゴリズムの名前付きの組み合わせです。デフォルトでは、利用可能なすべての暗号スイートがサポートされています。		String

名前	説明	デフォルト	タイプ
sslContextParameters (security)	Camel SSLContextParameters オブジェクトを使用した SSL 設定。設定されている場合、他の SSL エンドポイントパラメーターの前に適用されます。		SSLContextParameters
sslEnabledProtocols (security)	SSL 接続で有効なプロトコルの一覧。TLSv1.2、TLSv1.1、および TLSv1 はデフォルトで有効になっています。	TLSv1.2,TLSv1.1,TLSv1	String
sslEndpointAlgorithm (security)	サーバー証明書を使用してサーバーのホスト名を検証するエンドポイント識別アルゴリズム。		String
sslKeymanagerAlgorithm (security)	SSL 接続のキーマネージャーファクトリーによって使用されるアルゴリズム。デフォルト値は、Java 仮想マシンに設定されたキーマネージャーファクトリーアルゴリズムです。	SunX509	String
sslKeyPassword (security)	キーストアファイル内の秘密キーのパスワード。これはクライアントにとってオプションになります。		String
sslKeystoreLocation (security)	キーストアファイルのロケーション。これはクライアントではオプションで、クライアントの双方向認証に使用できます。		String
sslKeystorePassword (security)	キーストアファイルのストアパスワード。これはクライアントのオプションであり、ssl.keystore.location が設定されている場合にのみ必要です。		String
sslKeystoreType (security)	キーストアファイルのファイル形式。これはクライアントにとってオプションになります。デフォルト値は JKS です	JKS	String
sslProtocol (security)	SSLContext の生成に使用される SSL プロトコル。デフォルト設定は TLS で、ほとんどの場合はこれで問題ありません。最近の JVM で許可されている値は、TLS、TLSv1.1、および TLSv1.2 です。SSL、SSLv2、および SSLv3 は古い JVM でサポートされている可能性がありますが、セキュリティーの脆弱性が知られているため、それらの使用はお勧めできません。	TLS	String
sslProvider (security)	SSL 接続に使用されるセキュリティープロバイダーの名前。デフォルト値は JVM のデフォルトのセキュリティープロバイダーです。		String

名前	説明	デフォルト	タイプ
sslTrustmanagerAlgorithm (security)	SSL 接続のトラストマネージャーファクトリーによって使用されるアルゴリズム。デフォルト値は、Java 仮想マシンに設定されたトラストマネージャーファクトリーアルゴリズムです。	PKIX	String
sslTruststoreLocation (security)	トラストストアファイルのロケーション。		String
sslTruststorePassword (security)	トラストストアファイルのパスワード。		String
sslTruststoreType (security)	トラストストアファイルのファイル形式。デフォルト値は JKS です。	JKS	文字列

Producer/Consumer 設定の詳細については、以下を参照してください。

http://kafka.apache.org/documentation.html#newconsumerconfigs
http://kafka.apache.org/documentation.html#producerconfigs

181.3. メッセージヘッダー

181.3.1. コンシューマーヘッダー

次のヘッダーは、Kafka からのメッセージを使用するときに使用できます。

ヘッダー定数	ヘッダーの値	タイプ	説明
KafkaConstants.TOPIC	**"kafka.TOPIC"**	**String**	メッセージの発信元のトピック
KafkaConstants.PARTITION	**"kafka.PARTITION"**	**Integer**	メッセージが格納されたパーティション
KafkaConstants.OFFSET	**"kafka.OFFSET"**	**Long**	メッセージのオフセット
KafkaConstants.KEY	**"kafka.KEY"**	**Object**	メッセージのキー (設定されている場合)

ヘッダー定数	ヘッダーの値	タイプ	説明
KafkaConstants.HEADERS	**"kafka.HEADERS"**	**org.apache.kafka.common.header.Headers**	レコードのヘッダー
KafkaConstants.LAST_RECORD_BEFORE_COMMIT	**"kafka.LAST_RECORD_BEFORE_COMMIT"**	**Boolean**	コミット前の最後のレコードかどうか (**autoCommitEnable** エンドポイントパラメーターが **false** の場合にのみ使用可能)
KafkaConstants.MANUAL_COMMIT	**"CamelKafkaManualCommit"**	**KafkaManualCommit**	Kafka コンシューマーを使用する場合に手動オフセットコミットを強制するために使用できます。

181.3.2. プロデューサーヘッダー

メッセージを Kafka に送信する前に、次のヘッダーを設定できます。

ヘッダー定数	ヘッダーの値	タイプ	説明
KafkaConstants.KEY	**"kafka.KEY"**	**Object**	**必須** すべての関連メッセージが同じパーティションに入るようにするためのメッセージのキー
KafkaConstants.TOPIC	**"kafka.TOPIC"**	**String**	メッセージの送信先のトピック (**bridgeEndpoint** エンドポイントパラメーターが **true** の場合のみ読み取り)
KafkaConstants.PARTITION_KEY	**"kafka.PARTITION_KEY"**	**Integer**	明示的にパーティションを指定します (**KafkaConstants.KEY** ヘッダーが定義されている場合にのみ使用されます)

メッセージが Kafka に送信された後、次のヘッダーを使用できます

ヘッダー定数	ヘッダーの値	タイプ	説明
KafkaConstants.KAFKA_RECORDMETA	**"org.apache.kafka.clients.producer.RecordMetadata"**	**List<RecordMetadata>**	メタデータ (**recordMetadata** エンドポイントパラメーターが **true** の場合にのみ設定されます)

181.4. サンプル

181.4.1. Kafka からのメッセージの消費

Kafka からメッセージを読み取るために必要な最小限のルートを次に示します。

```
from("kafka:test?brokers=localhost:9092")
    .log("Message received from Kafka : ${body}")
    .log("    on the topic ${headers[kafka.TOPIC]}")
    .log("    on the partition ${headers[kafka.PARTITION]}")
    .log("    with the offset ${headers[kafka.OFFSET]}")
    .log("    with the key ${headers[kafka.KEY]}")
```

Kafka からのメッセージを消費する場合、独自のオフセット管理を使用でき、この管理を Kafka に委任する必要はありません。オフセットを保持するために、コンポーネントには **FileStateRepository** などの **StateRepository** 実装が必要です。この Bean は、レジストリーで使用できるはずです。ここでそれを使用する方法:

```
// Create the repository in which the Kafka offsets will be persisted
FileStateRepository repository = FileStateRepository.fileStateRepository(new
File("/path/to/repo.dat"));

// Bind this repository into the Camel registry
JndiRegistry registry = new JndiRegistry();
registry.bind("offsetRepo", repository);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
               "&groupId=A" +                          //
               "&autoOffsetReset=earliest" +           // Ask to start from the beginning if we have
unknown offset
               "&offsetRepository=#offsetRepo")        // Keep the offsets in the previously configured
repository
            .to("mock:result");
   }
});
```

181.4.2. Kafka へのメッセージの生成

Kafka にメッセージを書き込むために必要な最小限のルートを次に示します。

```
from("direct:start")
    .setBody(constant("Message from Camel"))         // Message to send
    .setHeader(KafkaConstants.KEY, constant("Camel")) // Key of the message
    .to("kafka:test?brokers=localhost:9092");
```

181.5. SSL 設定

Kafka` コンポーネントで SSL 通信を設定するには、2 つの異なる方法があります。

最初の方法は、多くの SSL エンドポイントパラメーターを使用する方法です。

```
from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
            "&groupId=A" +
            "&sslKeystoreLocation=/path/to/keystore.jks" +
            "&sslKeystorePassword=changeit" +
            "&sslKeyPassword=changeit")
        .to("mock:result");
```

2 つ目の方法は、**sslContextParameters** エンドポイントパラメーターを使用することです。

```
// Configure the SSLContextParameters object
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/path/to/keystore.jks");
ksp.setPassword("changeit");
KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("changeit");
SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

// Bind this SSLContextParameters into the Camel registry
JndiRegistry registry = new JndiRegistry();
registry.bind("ssl", scp);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
                "&groupId=A" +                          //
                "&sslContextParameters=#ssl")           // Reference the SSL configuration
            .to("mock:result");
    }
});
```

181.6. KAFKA べき等リポジトリーの使用

Camel 2.19 から利用可能

camel-kafka ライブラリーは、Kafka トピックベースのべき等リポジトリーを提供します。このリポジトリーは、べき等状態 (追加/削除) へのブロードキャストのすべての変更を Kafka トピックに保存し、イベントソーシングを通じて各リポジトリーのプロセスインスタンスのローカルインメモリーキャッシュにデータを取り込みます。

使用されるトピックは、べき等リポジトリーインスタンスごとに一意である必要があります。このメカニズムには、トピックパーティションの数に関する要件はありません。リポジトリーがすべてのパーティションから同時に消費するためです。また、トピックのレプリケーションファクターに関する要件もありません。

トピックを使用する各リポジトリーインスタンス (通常、並行して実行されている異なるマシン上) は独自のコンシューマーグループを制御するため、同じトピックを使用する 10 個の Camel プロセスのクラスターでは、それぞれが独自のオフセットを制御します。

起動時に、インスタンスはトピックにサブスクライブし、オフセットを先頭に巻き戻し、キャッシュを最新の状態に再構築します。**pollDurationMs** の長さの 1 つのポーリングで 0 レコードが返されるまで、キャッシュはウォームアップされたと見なされません。キャッシュがウォームアップするか、30 秒経過するまで、起動は完了しません。後者の場合、コンシューマーがトピックの最後に追いつくまで、冪等リポジトリーは一貫性のない状態になる可能性があります。

KafkaIdempotentRepository には次のプロパティーがあります。

プロパティー	説明
topic	変更をブロードキャストするために使用する Kafka トピックの名前。(必要)
bootstrapServers	内部 Kafka プロデューサーおよびコンシューマーの **bootstrap.servers** プロパティー。**consumerConfig** と **producerConfig** を設定しない場合は、これを省略形として使用します。このコンポーネントを使用すると、プロデューサーとコンシューマーに適切なデフォルト設定が適用されます。
producerConfig	変更をブロードキャストする Kafka プロデューサーによって使用されるプロパティーを設定します。**bootstrapServers** をオーバーライドするため、Kafka の **bootstrap.servers** プロパティー自体を定義する必要があります
consumerConfig	トピックからキャッシュを設定する Kafka コンシューマーによって使用されるプロパティーを設定します。**bootstrapServers** をオーバーライドするため、Kafka の **bootstrap.servers** プロパティー自体を定義する必要があります
maxCacheSize	最近使用したキーをメモリーに保存する数 (デフォルトは 1000)。
pollDurationMs	Kafka コンシューマーのポーリング期間。ローカルキャッシュはすぐに更新されます。この値は、トピックからキャッシュを更新する他のピアが、キャッシュアクションメッセージを送信した冪等のコンシューマーインスタンスと比べてどれだけ遅れているかに影響します。このデフォルト値は 100 ミリ秒です。 この値を明示的に設定する場合は、リモートキャッシュのライブ性と、このリポジトリーのコンシューマーと Kafka ブローカー間のネットワークトラフィックの量との間にトレードオフがあることに注意してください。キャッシュウォームアッププロセスは、何も取得しない 1 つのポーリングがあることにも依存します。これは、ストリームが現在の時点まで消費されたことを示します。トピックでメッセージが送信される速度に対してポーリング期間が長すぎる場合、キャッシュがウォームアップできず、キャッシュが追いつくまでピアに対して一貫性のない状態で動作する可能性があります。

topic と **bootstrapServers** を定義してリポジトリーをインスタンス化するか、または **producerConfig** および **consumerConfig** プロパティーセットを明示的に定義して、SSL/SASL などの機能を有効にすることができます。

使用するには、**CamelContext** に対応しているため、手動で、または Spring/Blueprint で Bean として登録することにより、このリポジトリーを Camel レジストリーに配置する必要があります。

使用例は次のとおりです。

```
KafkaIdempotentRepository kafkaIdempotentRepository = new
```

```
KafkaIdempotentRepository("idempotent-db-inserts", "localhost:9091");

SimpleRegistry registry = new SimpleRegistry();
registry.put("insertDbIdemRepo", kafkaIdempotentRepository); // must be registered in the registry, to enable access to the CamelContext
CamelContext context = new CamelContext(registry);

// later in RouteBuilder...
from("direct:performInsert")
    .idempotentConsumer(header("id")).messageIdRepositoryRef("insertDbIdemRepo")
        // once-only insert into database
    .end()
```

XML の場合:

```
<!-- simple -->
<bean id="insertDbIdemRepo"
class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="bootstrapServers" value="localhost:9091"/>
</bean>

<!-- complex -->
<bean id="insertDbIdemRepo"
class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="maxCacheSize" value="10000"/>
  <property name="consumerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
  <property name="producerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
</bean>
```

181.7. KAFKA コンシューマーでの手動コミットの使用

Camel 2.21 以降で利用可能

デフォルトでは、Kafka コンシューマーは自動コミットを使用します。オフセットは、指定された間隔を使用してバックグラウンドで自動的にコミットされます。

手動コミットを強制する場合は、メッセージヘッダーに保存されている Camel Exchange の **KafkaManualCommit** API を使用できます。これには、**KafkaComponent** またはエンドポイントでオプション **allowManualCommit** を **true** に設定して、手動コミットを有効にする必要があります。次に例を示します。

```
KafkaComponent kafka = new KafkaComponent();
kafka.setAllowManualCommit(true);
...
camelContext.addComponent("kafka", kafka);
```

その後、Camel **Processor** などの Java コードから **KafkaManualCommit** を使用できます。

```
public void process(Exchange exchange) {
    KafkaManualCommit manual = exchange.getIn().getHeader(KafkaConstants.MANUAL_COMMIT,
KafkaManualCommit.class);
    manual.commitSync();
}
```

これにより、同期コミットが強制され、Kafka でコミットが確認されるまでブロックされるか、失敗した場合は例外が出力されます。

KafkaManualCommit のカスタム実装を使用する場合は、カスタム実装のインスタンスを作成する **KafkaComponent** でカスタム **KafkaManualCommitFactory** を設定できます。

181.8. KAFKA ヘッダーの伝播

Camel 2.22 以降で利用可能

Kafka からメッセージを消費する場合、ヘッダーは camel exchange ヘッダーに自動的に伝播されます。同じ動作に裏打ちされたフローの生成 - 特定の交換のキャメルヘッダーは、kafka メッセージヘッダーに伝達されます。

kafka ヘッダーは **byte[]** 値のみを許可するため、camel exchange ヘッダーを伝播するには、その値を **bytes[]** にシリアル化する必要があります。そうしないと、ヘッダーはスキップされます。次のヘッダー値タイプがサポートされています: **String**、**Integer**、**Long**、**Double**、**Boolean**、**byte[]**。注: kafka **から** camel exchange **に** 伝播されるすべてのヘッダーには、デフォルトで **byte[]** 値が含まれます。デフォルトの機能をオーバーライドするために、URI パラメーターを設定できます。ルート **from** の **kafkaHeaderDeserializer** とルート **to** の **kafkaHeaderSerializer** です。例:

```
from("kafka:my_topic?kafkaHeaderDeserializer=#myDeserializer")
...
.to("kafka:my_topic?kafkaHeaderSerializer=#mySerializer")
```

デフォルトでは、すべてのヘッダーが **KafkaHeaderFilterStrategy** によってフィルタリングされています。Strategy は、**Camel** または **org.apache.camel** 接頭辞で始まるヘッダーを除外します。デフォルトの戦略は、**to** ルートと **from** ルートの両方で **headerFilterStrategy** uri パラメーターを使用してオーバーライドできます。

```
from("kafka:my_topic?headerFilterStrategy=#myStrategy")
...
.to("kafka:my_topic?headerFilterStrategy=#myStrategy")
```

myStrategy オブジェクトは **HeaderFilterStrategy** のサブクラスである必要があり、**CamelContext** に対応しているため、手動で、または Spring/Blueprint で Bean として登録することにより、Camel レジストリーに配置する必要があります。

第182章 KESTREL コンポーネント (非推奨)

Camel バージョン 2.6 以降で利用可能

Kestrel コンポーネントを使用すると、メッセージを Kestrel キューに送信したり、メッセージを Kestrel キューから消費したりできます。このコンポーネントは、Kestrel サーバーとの memcached プロトコル通信に spymemcached クライアントを使用します。

警告

kestrel プロジェクトは非アクティブであり、Camel チームはこのコンポーネントを非推奨と見なしています。

182.1. URI 形式

```
kestrel://[addresslist/]queuename[?options]
```

queuename は、Kestrel 上のキューの名前です。URI の **addresslist** 部分には、1 つ以上の **host:port** ペアが含まれる場合があります。たとえば、**kserver01:22133** のキュー **foo** に接続するには、次を使用します:

```
kestrel://kserver01:22133/foo
```

addresslist を省略すると、**localhost:22133** が想定されます。つまり、次のようになります。

```
kestrel://foo
```

同様に、ポートが addresslist の **host:port** ペアから省略されている場合、デフォルトのポート 22133 が想定されます。つまり、次のようになります。

```
kestrel://kserver01/foo
```

クラスター化されたキューを生成するために使用される Kestrel エンドポイント URI の例を次に示します。

```
kestrel://kserver01:22133,kserver02:22133,kserver03:22133/massive
```

キューから同時に消費するために使用される Kestrel エンドポイント URI の例を次に示します。

```
kestrel://kserver03:22133/massive?concurrentConsumers=25&waitTimeMs=500
```

182.2. オプション

Kestrel コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	新しいエンドポイントを作成するためのベースとして、設定済みの共有設定を使用するため。		KestrelConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Kestrel エンドポイントは、URI 構文を使用して設定されます。

```
kestrel:addresses/queue
```

パスおよびクエリーパラメーターを使用します。

182.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
addresses	kestrel が実行されているアドレス	localhost:22133	String[]
queue	**必須** ポーリングするキュー		String

182.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
concurrentConsumers (Common)	スレッドプールに対してスケジュールする同時リスナーの数	1	int
waitTimeMs (Common)	指定された待機が (サーバー側で) ブロックする時間 (ミリ秒単位)	100	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

182.3. SPRING XML を使用した KESTREL コンポーネントの設定

明示的な設定の最も単純な形式は次のとおりです。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <bean id="kestrel" class="org.apache.camel.component.kestrel.KestrelComponent"/>

  <camelContext xmlns="http://camel.apache.org/schema/spring">
  </camelContext>

</beans>
```

これにより、すべてのデフォルト設定で Kestrel コンポーネントが有効になります。つまり、デフォルトで **localhost:22133**、100 ミリ秒の待機時間、単一の非同時コンシューマーが使用されます。

基本設定 (**?properties** が指定されていないエンドポイントに設定を提供する) で特定のオプションを使用するには、次のように KestrelConfiguration POJO を設定できます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <bean id="kestrelConfiguration" class="org.apache.camel.component.kestrel.KestrelConfiguration">
    <property name="addresses" value="kestrel01:22133"/>
```

```
    <property name="waitTimeMs" value="100"/>
    <property name="concurrentConsumers" value="1"/>
  </bean>

  <bean id="kestrel" class="org.apache.camel.component.kestrel.KestrelComponent">
    <property name="configuration" ref="kestrelConfiguration"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/spring">
  </camelContext>

</beans>
```

182.4. 使用例

182.4.1. 例 1: Consuming

```
from("kestrel://kserver02:22133/massive?concurrentConsumers=10&waitTimeMs=500")
  .bean("myConsumer", "onMessage");
```

```
public class MyConsumer {
    public void onMessage(String message) {
        ...
    }
}
```

182.4.2. 例 2: Producing

```
public class MyProducer {
    @EndpointInject(uri = "kestrel://kserver01:22133,kserver02:22133/myqueue")
    ProducerTemplate producerTemplate;

    public void produceSomething() {
        producerTemplate.sendBody("Hello, world.");
    }
}
```

182.4.3. 例 3: Spring XML 設定

```
  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="kestrel://ks01:22133/sequential?concurrentConsumers=1&waitTimeMs=500"/>
      <bean ref="myBean" method="onMessage"/>
    </route>
    <route>
      <from uri="direct:start"/>
      <to uri="kestrel://ks02:22133/stuff"/>
    </route>
  </camelContext>
```

```
public class MyBean {
```

```
    public void onMessage(String message) {
        ...
    }
}
```

182.5. 依存関係

Kestrel コンポーネントには、次の依存関係があります。

- **spymemcached** 2.5 (またはそれ以上)

182.5.1. spymemcached

クラスパスに **spymemcached** jar が **必要です**。pom.xml で使用できるスニペットを次に示します。

```
<dependency>
  <groupId>spy</groupId>
  <artifactId>memcached</artifactId>
  <version>2.5</version>
</dependency>
```

または、jar を直接ダウンロードする こともできます。

警告: 制限事項

注記

JVM アサーションが有効になっている場合、spymemcached クライアントライブラリーは kestrel で正しく動作 **しません**。アサーションが有効で、要求されたキーに **/t=...** 拡張子が含まれている場合、spymemcached には既知の問題があります (つまり、エンドポイント URI で **waitTimeMs** オプションを使用している場合、これは強く推奨されます)。幸い、JVM のアサーションは、明示的に有効 にしない限り、**デフォルトで無効** なので、通常の環境では問題にならないはずです。注意すべき点は、Maven の Surefire テストプラグインがアサーションを **有効にすること** です。このコンポーネントを Maven テスト環境で使用している場合は、**enableAssertions** を **false** に設定する必要がある場合があります。詳細については、surefire:test リファレンス を参照してください。

182.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第183章 KIE-CAMEL

183.1. 概要

Kie-Camel は、KIE (Drools) と統合する Apache Camel コンポーネント (エンドポイント) です。ルートにプルして実行できる **KIE** モジュール (maven GAV を使用) を指定できます。また、ファクトとしてメッセージボディーの一部を指定することもできます。

kie-camel コンポーネントの詳細は、Apache Camel Integration を参照してください。

第184章 KRATI コンポーネント (非推奨)

Camel バージョン 2.9 以降で利用可能

このコンポーネントにより、Camel 内で krati データストアとデータセットを使用できます。Krati は、待ち時間が非常に短く、スループットが高い単純な永続データストアです。設定、パフォーマンス、および JVM ガベージコレクションの調整にほとんど労力をかけずに、読み書きが集中するアプリケーションと簡単に統合できるように設計されています。

Camel は krati datastore_(key/value engine)_ のプロデューサーとコンシューマーを提供します。また、重複メッセージを除外するためのべき等リポジトリーも提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-krati</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

184.1. URI 形式

```
krati:[the path of the datastore][?options]
```

データストアのパス は、krati がデータストアに使用するフォルダーの相対パスです。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

184.2. KRATI オプション

Krati コンポーネントにはオプションがありません。

Krati エンドポイントは、URI 構文を使用して設定されます。

```
krati:path
```

パスおよびクエリーパラメーターを使用します。

184.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
path	**必須** データストアのパスは、krati がデータストアに使用するフォルダーの相対パスです。		String

184.2.2. クエリーパラメーター (29 個のパラメーター):

名前	説明	デフォルト	タイプ
hashFunction (Common)	使用するハッシュ関数。		HashFunction<byte[]>
initialCapacity (Common)	ストアの初期容量。	100	int
keySerializer (Common)	キーのシリアル化に使用されるシリアライザー。		Object>
segmentFactory (Common)	対象ストアのセグメントファクトリーを設定します。		SegmentFactory
segmentFileSize (Common)	MB 単位のデータストアセグメントサイズ。	64	int
valueSerializer (Common)	値をシリアル化するために使用されるシリアライザー。		Object>
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessagesPerPoll (consumer)	1 回のポーリングで受信できるメッセージの最大数。これを使用して、大量のデータを読み込んでメモリーを消費しすぎないようにすることができます。		int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
key (producer)	キー。		String
operation (producer)	データストアに対して実行される操作のタイプを指定します。		String
value (producer)	値。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

```
krati:/tmp/krati?operation=CamelKratiGet&initialCapacity=10000&keySerializer=#myCustomSerializer
```

プロデューサーエンドポイントの場合、適切なヘッダーをメッセージに渡すことで、上記の URI オプションをすべてオーバーライドできます。

184.2.3. データストアのメッセージヘッダー

ヘッダー	説明
CamelKratiOperation	データストアで実行される操作。有効なオプションは、CamelKratiAdd、CamelKratiGet、CamelKratiDelete、CamelKratiDeleteAll です。

ヘッダー	説明
CamelKratiKey	キー。
CamelKratiValue	値。

184.3. 使用例

184.3.1. 例 1: データストアへの書き込み。

この例では、データストア内にメッセージを格納する方法を示します。

```
from("direct:put").to("krati:target/test/producertest");
```

上記の例では、URI パラメーターのいずれかをメッセージのヘッダーでオーバーライドできます。
上記の例が xml を使用してルートを定義すると、次のようになります。

```
<route>
    <from uri="direct:put"/>
    <to uri="krati:target/test/producerspringtest"/>
</route>
```

184.3.2. 例 2: データストアからの取得/読み取り

この例では、データストアのコンテンツを読み取る方法を示します。

```
from("direct:get")
    .setHeader(KratiConstants.KRATI_OPERATION,
constant(KratiConstants.KRATI_OPERATION_GET))
    .to("krati:target/test/producertest");
```

上記の例では、URI パラメーターのいずれかをメッセージのヘッダーでオーバーライドできます。
上記の例が xml を使用してルートを定義すると、次のようになります。

```
<route>
    <from uri="direct:get"/>
    <to uri="krati:target/test/producerspringtest?operation=CamelKratiGet"/>
</route>
```

184.3.3. 例 3: データストアからの消費

この例では、指定されたデータストアの下にあるすべてのアイテムを消費します。

```
    from("krati:target/test/consumertest")
        .to("direct:next");
```

-

以下に示すように、xml を使用して同じ目標を達成できます。

```
<route>
    <from uri="krati:target/test/consumerspringtest"/>
    <to uri="mock:results"/>
</route>
```

184.4. べき等リポジトリー

すでに述べたように、このコンポーネントは、重複メッセージをフィルタリングするために使用できる idemptonet リポジトリーも提供します。

```
from("direct://in").idempotentConsumer(header("messageId"), new
KratiIdempotentRepositroy("/tmp/idempotent").to("log://out");
```

184.4.1. その他の参考資料

Krati の Web サイト

第185章 KUBERNETES コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes コンポーネントは、アプリケーションを Kubernetes スタンドアロンまたは Openshift 上に統合します。

camel-kubernetes は 13 個のコンポーネントで設定されています。

- Kubernetes の ConfigMap
- Kubernetes Namespace
- Kubernetes Node
- Kubernetes 永続ボリューム
- Kubernetes 永続ボリューム要求
- Kubernetes Pod
- Kubernetes Replication Controller
- Kubernetes Resource Quota
- Kubernetes の Secret
- Kubernetes サービスアカウント
- Kubernetes Service

OpenShift では、次のこともできます。

- Kubernetes ビルド設定
- Kubernetes ビルド

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kubernetes</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

185.1. ヘッダー

名前	タイプ	説明
CamelKubernetesOperation	String	プロデューサーの操作

名前	タイプ	説明
CamelKubernetesNamespaceName	String	名前空間名
CamelKubernetesNamespaceLabels	Map	名前空間ラベル
CamelKubernetesServiceLabels	Map	サービスラベル
CamelKubernetesServiceName	String	サービス名
CamelKubernetesServiceSpec	io.fabric8.kubernetes.api.model.ServiceSpec	サービスの仕様
CamelKubernetesReplicationControllersLabels	Map	レプリケーションコントローラーのラベル
CamelKubernetesReplicationControllerName	String	レプリケーションコントローラー名

名前	タイプ	説明
CamelKubernetesReplicationControllerSpec	io.fabric8.kubernetes.api.model.ReplicationControllerSpec	レプリケーションコントローラーの仕様
CamelKubernetesReplicationControllerReplicas	Integer	スケール操作中のレプリケーションコントローラーのレプリカの数
CamelKubernetesPodsLabels	Map	Pod のラベル
CamelKubernetesPodName	String	Pod の名前
CamelKubernetesPodSpec	io.fabric8.kubernetes.api.model.PodSpec	Pod の仕様
CamelKubernetesPersistentVolumesLabels	Map	永続ボリュームラベル
CamelKubernetesPersistentVolumesName	String	永続ボリューム名

名前	タイプ	説明
CamelKubernetesPersistentVolumesClaimsLabels	Map	永続ボリューム要求のラベル
CamelKubernetesPersistentVolumesClaimsName	String	永続ボリューム要求 (PVC) の名前
CamelKubernetesPersistentVolumesClaimsSpec	io.fabric8.kubernetes.api.model.PersistentVolumeClaimSpec	永続ボリューム要求の仕様
CamelKubernetesSecretsLabels	Map	秘密のラベル
CamelKubernetesSecretsName	String	Secret 名
CamelKubernetesSecret	io.fabric8.kubernetes.api.model.Secret	秘密のオブジェクト

名前	タイプ	説明
CamelKubernetesResourcesQuotaLabels	Map	リソースクォータラベル
CamelKubernetesResourcesQuotaName	String	リソースクォータ名
CamelKubernetesResourceQuotaSpec	io.fabric8.kubernetes.api.model.ResourceQuotaSpec	リソースクォータの仕様
CamelKubernetesServiceAccountsLabels	Map	Service Account labels
CamelKubernetesServiceAccountName	String	サービスアカウント名
CamelKubernetesServiceAccount	io.fabric8.kubernetes.api.model.ServiceAccount	サービスアカウントオブジェクト
CamelKubernetesNodesLabels	Map	ノードラベル

名前	タイプ	説明
CamelKubernetesNodeName	String	ノード名
CamelKubernetesBuildsLabels	Map	Openshift ビルドラベル
CamelKubernetesBuildName	String	Openshift ビルド名
CamelKubernetesBuildConfigsLabels	Map	Openshift ビルド設定のラベル
CamelKubernetesBuildConfigName	String	Openshift ビルド設定名
CamelKubernetesEventAction	io.fabric8.kubernetes.client.Watcher.Action	コンシューマーが監視したアクション
CamelKubernetesEventTimestamp	String	コンシューマーが監視したアクションのタイムスタンプ
CamelKubernetesConfigMapName	String	ConfigMap 名

名前	タイプ	説明
CamelKubernetesConfigMapsLabels	Map	ConfigMap ラベル
CamelKubernetesConfigData	Map	ConfigMap データ

185.2. 使用方法

185.2.1. プロデューサーの例

ここでは、camel-kubernetes を使用したプロデューサーの例をいくつか示します。

185.2.2. Pod の作成

```
from("direct:createPod")
    .toF("kubernetes-pods://%s?oauthToken=%s&operation=createPod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_SPEC ヘッダーを使用することで、PodSpec を指定してこの操作に渡すことができます。

185.2.3. Pod の削除

```
from("direct:createPod")
    .toF("kubernetes-pods://%s?oauthToken=%s&operation=deletePod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_NAME ヘッダーを使用することで、Pod 名を指定してこの操作に渡すことができます。

第186章 KUBERNETES コンポーネント (非推奨)

Camel バージョン 2.17 以降で利用可能

重要

複合 kubernetes コンポーネントは非推奨になりました。次のように分割された個々のコンポーネントを使用します。

- Kubernetes コンポーネント
  - Kubernetes ビルド設定
  - Kubernetes ビルド
  - Kubernetes の ConfigMap
  - Kubernetes Namespace
  - Kubernetes Node
  - Kubernetes 永続ボリューム
  - Kubernetes 永続ボリューム要求
  - Kubernetes Pod
  - Kubernetes Replication Controller
  - Kubernetes Resource Quota
  - Kubernetes の Secret
  - Kubernetes サービスアカウント
  - Kubernetes Service

Kubernetes コンポーネントは、アプリケーションを Kubernetes スタンドアロンまたは Openshift 上に統合するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kubernetes</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

186.1. URI 形式

```
kubernetes:masterUrl[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

186.2. オプション

Kubernetes コンポーネントにはオプションがありません。

Kubernetes エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes:masterUrl
```

パスおよびクエリーパラメーターを使用します。

186.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		String

186.2.2. クエリーパラメーター(28 パラメーター):

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
category (Common)	**必須** Kubernetes プロデューサーおよびコンシューマーカテゴリー		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String

名前	説明	デフォルト	タイプ
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String

名前	説明	デフォルト	タイプ
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

186.3. ヘッダー

名前	タイプ	説明
CamelKubernetesOperation	String	プロデューサーの操作
CamelKubernetesNamespaceName	String	名前空間名
CamelKubernetesNamespaceLabels	Map	名前空間ラベル

名前	タイプ	説明
CamelKubernetesServiceLabels	Map	サービスラベル
CamelKubernetesServiceName	String	サービス名
CamelKubernetesServiceSpec	io.fabric8.kubernetes.api.model.ServiceSpec	サービスの仕様
CamelKubernetesReplicationControllersLabels	Map	レプリケーションコントローラーのラベル
CamelKubernetesReplicationControllerName	String	レプリケーションコントローラー名
CamelKubernetesReplicationControllerSpec	io.fabric8.kubernetes.api.model.ReplicationControllerSpec	レプリケーションコントローラーの仕様

名前	タイプ	説明
CamelKubernetesReplicationControllerReplicas	Integer	スケール操作中のレプリケーションコントローラーのレプリカの数
CamelKubernetesPodsLabels	Map	Pod のラベル
CamelKubernetesPodName	String	Pod の名前
CamelKubernetesPodSpec	io.fabric8.kubernetes.api.model.PodSpec	Pod の仕様
CamelKubernetesPersistentVolumesLabels	Map	永続ボリュームラベル
CamelKubernetesPersistentVolumesName	String	永続ボリューム名
CamelKubernetesPersistentVolumesClaimsLabels	Map	永続ボリューム要求のラベル

名前	タイプ	説明
CamelKubernetesPersistentVolumesClaimsName	String	永続ボリューム要求 (PVC) の名前
CamelKubernetesPersistentVolumesClaimsSpec	io.fabric8.kubernetes.api.model.PersistentVolumeClaimSpec	永続ボリューム要求の仕様
CamelKubernetesSecretsLabels	Map	秘密のラベル
CamelKubernetesSecretsName	String	Secret 名
CamelKubernetesSecret	io.fabric8.kubernetes.api.model.Secret	秘密のオブジェクト
CamelKubernetesResourcesQuotaLabels	Map	リソースクォータラベル
CamelKubernetesResourcesQuotaName	String	リソースクォータ名

名前	タイプ	説明
CamelKubernetesResourceQuotaSpec	io.fabric8.kubernetes.api.model.ResourceQuotaSpec	リソースクォータの仕様
CamelKubernetesServiceAccountsLabels	Map	Service Account labels
CamelKubernetesServiceAccountName	String	サービスアカウント名
CamelKubernetesServiceAccount	io.fabric8.kubernetes.api.model.ServiceAccount	サービスアカウントオブジェクト
CamelKubernetesNodesLabels	Map	ノードラベル
CamelKubernetesNodeName	String	ノード名
CamelKubernetesBuildsLabels	Map	Openshift ビルドラベル

名前	タイプ	説明
CamelKubernetesBuildName	String	Openshift ビルド名
CamelKubernetesBuildConfigsLabels	Map	Openshift ビルド設定のラベル
CamelKubernetesBuildConfigName	String	Openshift ビルド設定名
CamelKubernetesEventAction	io.fabric8.kubernetes.client.Watcher.Action	コンシューマーが監視したアクション
CamelKubernetesEventTimestamp	String	コンシューマーが監視したアクションのタイムスタンプ
CamelKubernetesConfigMapName	String	ConfigMap 名
CamelKubernetesConfigMapsLabels	Map	ConfigMap ラベル
CamelKubernetesConfigData	Map	ConfigMap データ

186.4. CATEGORIES

実際には camel-kubernetes コンポーネントは次の Kubernetes リソースをサポートしています

- Namespaces
- Pod
- レプリケーションコントローラー
- サービス
- Persistent Volumes
- Persistent Volume Claim (永続ボリューム要求、PVC)
- シークレット
- リソースクォータ
- サービスアカウント
- ノード
- Configmaps

オープンシフトでも

- ビルド
- BuildConfig

186.5. 使用方法

186.5.1. プロデューサーの例

ここでは、camel-kubernetes を使用したプロデューサーの例をいくつか示します。

186.5.2. Pod の作成

```
from("direct:createPod")
    .toF("kubernetes://%s?oauthToken=%s&category=pods&operation=createPod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_SPEC ヘッダーを使用することで、PodSpec を指定してこの操作に渡すことができます。

186.5.3. Pod の削除

```
from("direct:createPod")
    .toF("kubernetes://%s?oauthToken=%s&category=pods&operation=deletePod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_NAME ヘッダーを使用することで、Pod 名を指定してこの操作に渡すことができます。

第187章 KUBERNETES CONFIGMAP COMPONENT

Camel バージョン 2.17 以降で利用可能

Kubernetes ConfigMap コンポーネントは、kubernetes ConfigMap 操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

187.1. コンポーネントオプション

Kubernetes ConfigMap コンポーネントにはオプションがありません。

187.2. エンドポイントオプション

Kubernetes ConfigMap エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-config-maps:masterUrl
```

パスおよびクエリーパラメーターを使用します。

187.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

187.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第188章 KUBERNETES DEPLOYMENTS コンポーネント

Camel バージョン 2.20 以降で利用可能

Kubernetes Deployments コンポーネントは、kubernetes シークレットオペレーションを実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

188.1. コンポーネントオプション

Kubernetes Deployments コンポーネントにはオプションがありません。

188.2. エンドポイントオプション

Kubernetes Deployments エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-deployments:masterUrl
```

パスおよびクエリーパラメーターを使用します。

188.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

188.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第189章 KUBERNETES NAMESPACES コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Namespaces コンポーネントは、kubernetes 名前空間操作を実行するプロデューサーと kubernetes 名前空間イベントを消費するコンシューマーを提供する Kubernetes コンポーネント の 1 つです。

189.1. コンポーネントオプション

Kubernetes Namespaces コンポーネントにはオプションがありません。

189.2. エンドポイントオプション

Kubernetes Namespaces エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-namespaces:masterUrl
```

パスおよびクエリーパラメーターを使用します。

189.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

189.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String

名前	説明	デフォルト	タイプ
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第190章 KUBERNETES NODES コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Nodes コンポーネントは、kubernetes ノード操作を実行するプロデューサーと kubernetes ノードイベントを消費するコンシューマーを提供する Kubernetes コンポーネント の 1 つです。

190.1. コンポーネントオプション

Kubernetes Nodes コンポーネントにはオプションがありません。

190.2. エンドポイントオプション

Kubernetes Nodes エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-nodes:masterUrl
```

パスおよびクエリーパラメーターを使用します。

190.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

190.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String

名前	説明	デフォルト	タイプ
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第191章 KUBERNETES PERSISTENT VOLUME CLAIM コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Persistent Volume Claimコンポーネントは、Kubernetes コンポーネント の 1 つであり、kubernetes の永続ボリューム要求操作を実行するプロデューサーを提供します。

191.1. コンポーネントオプション

Kubernetes Persistent Volume Claim コンポーネントにはオプションがありません。

191.2. エンドポイントオプション

Kubernetes Persistent Volume Claim エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-persistent-volumes-claims:masterUrl
```

パスおよびクエリーパラメーターを使用します。

191.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

191.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第192章 KUBERNETES PERSISTENT VOLUME コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Persistent Volume コンポーネントは、kubernetes の永続ボリューム操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

192.1. コンポーネントオプション

Kubernetes Persistent Volume コンポーネントにはオプションがありません。

192.2. エンドポイントオプション

Kubernetes Persistent Volume エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-persistent-volumes:masterUrl
```

パスおよびクエリーパラメーターを使用します。

192.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

192.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第193章 KUBERNETES PODS コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Pods コンポーネントは、kubernetes Pod 操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

193.1. コンポーネントオプション

Kubernetes Pods コンポーネントにはオプションがありません。

193.2. エンドポイントオプション

Kubernetes Pods エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-pods:masterUrl
```

パスおよびクエリーパラメーターを使用します。

193.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

193.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第194章 KUBERNETES REPLICATION CONTROLLER コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Replication Controllerコンポーネントは、Kubernetes Replication Controller 操作を実行するプロデューサーと、Kubernetes Replication Controller イベントを消費するコンシューマーを提供する Kubernetes コンポーネント の 1 つです。

194.1. コンポーネントオプション

Kubernetes Replication Controller コンポーネントにはオプションがありません。

194.2. エンドポイントオプション

Kubernetes Replication Controller エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-replication-controllers:masterUrl
```

パスおよびクエリーパラメーターを使用します。

194.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

194.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String

名前	説明	デフォルト	タイプ
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第195章 KUBERNETES RESOURCES QUOTA コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Resources Quota コンポーネントは、Kubernetes Resources Quota 操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

195.1. コンポーネントオプション

Kubernetes Resources Quota コンポーネントにはオプションがありません。

195.2. エンドポイントオプション

Kubernetes Resources Quota エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-resources-quota:masterUrl
```

パスおよびクエリーパラメーターを使用します。

195.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

195.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第196章 KUBERNETES SECRETS コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Secrets コンポーネントは、Kubernetes Secrets 操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

196.1. コンポーネントオプション

Kubernetes Secrets コンポーネントにはオプションがありません。

196.2. エンドポイントオプション

Kubernetes Secrets エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-secrets:masterUrl
```

パスおよびクエリーパラメーターを使用します。

196.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

196.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第197章 KUBERNETES SERVICE ACCOUNT コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Service Accountコンポーネントは、Kubernetes Service Account 操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

197.1. コンポーネントオプション

Kubernetes Service Account コンポーネントにはオプションがありません。

197.2. エンドポイントオプション

Kubernetes Service Account エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-service-accounts:masterUrl
```

パスおよびクエリーパラメーターを使用します。

197.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

197.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

第198章 KUBERNETES SERVICES コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Services コンポーネントは、Kubernetes Services 操作を実行するプロデューサーと Kubernetes Services イベントを消費するコンシューマーを提供する Kubernetes コンポーネント の 1 つです。

198.1. コンポーネントオプション

Kubernetes Services コンポーネントにはオプションがありません。

198.2. エンドポイントオプション

Kubernetes Services エンドポイントは、URI 構文を使用して設定されます。

```
kubernetes-services:masterUrl
```

パスおよびクエリーパラメーターを使用します。

198.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

198.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	使用する Kubernetes API バージョン		String
dnsDomain (Common)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (Common)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (Common)	ServiceCall EIP に使用されるポート名		文字列

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	リソースの監視時の Consumer Label キー		String
labelValue (consumer)	リソースの監視時の Consumer Label の値		String
namespace (consumer)	namespace		String
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視する Consumer Resource Name		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	Kubernetes で実行するプロデューサー操作		String
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String

名前	説明	デフォルト	タイプ
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		文字列

198.3. ECLIPSE KURA コンポーネント

Camel 2.15 以降で利用可能

このドキュメントページでは、Camel と Eclipse Kura M2M ゲートウェイの統合オプションについて説明します。Camel ルートを Eclipse Kura にデプロイする一般的な理由は、エンタープライズ統合パターンと Camel コンポーネントをメッセージング M2M ゲートウェイに提供することです。たとえば、Kura を Raspberry PI にインストールし、Kura サービスを使用してその Raspberry PI に取り付けられたセンサーから温度を読み取り、最後に Camel EIP とコンポーネントを使用して現在の温度値をデータセンターサービスに転送することができます。

198.3.1. KuraRouter アクティベーター

Eclipse Kura にデプロイされたバンドルは、通常、バンドルアクティベーターとして開発されます。したがって、Apache Camel ルートを Kura にデプロイする最も簡単な方法は、**org.apache.camel.kura.KuraRouter** クラスを拡張するクラスを含む OSGi バンドルを作成することです。

```
public class MyKuraRouter extends KuraRouter {

  @Override
  public void configure() throws Exception {
    from("timer:trigger").
      to("netty-http:http://app.mydatacenter.com/api");
  }

}
```

KuraRouter は **org.osgi.framework.BundleActivator** インターフェイスを実装しているため、Kura バンドルコンポーネントクラスを作成する 際に、**start** および **stop** ライフサイクルメソッドを登録する必要があることに注意してください。

Kura ルーターは、独自の OSGi 対応の **CamelContext** を開始します。これは、**KuraRouter** を拡張するすべてのクラスに対して、専用の **CamelContext** インスタンスが存在することを意味します。理想的には、OSGi バンドルごとに 1 つの **KuraRouter** をデプロイすることをお勧めします。

198.3.2. KuraRouter のデプロイ

Kura ルータークラスを含むバンドルは、OSGi マニフェストに次のパッケージをインポートする必要があります。

```
Import-Package: org.osgi.framework;version="1.3.0",
  org.slf4j;version="1.6.4",

org.apache.camel,org.apache.camel.impl,org.apache.camel.core.osgi,org.apache.camel.builder,org.apache.camel.model,
  org.apache.camel.component.kura
```

Camel コンポーネントはランタイムレベルでサービスとして解決されるため、ルートで使用する予定のすべての Camel コンポーネントバンドルをインポートする必要はないことに注意してください。

ルーターバンドルをデプロイメントする前に、次の Camel コアバンドルをデプロイメント (および開始) していることを確認してください (Kura GoGo シェルを使用)...

```
install file:///home/user/.m2/repository/org/apache/camel/camel-core/2.15.0/camel-core-2.15.0.jar
start <camel-core-bundle-id>
install file:///home/user/.m2/repository/org/apache/camel/camel-core-osgi/2.15.0/camel-core-osgi-2.15.0.jar
start <camel-core-osgi-bundle-id>
install file:///home/user/.m2/repository/org/apache/camel/camel-kura/2.15.0/camel-kura-2.15.0.jar
start <camel-kura-bundle-id>
```

...そして、ルートで使用する予定のすべてのコンポーネント:

```
install file:///home/user/.m2/repository/org/apache/camel/camel-stream/2.15.0/camel-stream-2.15.0.jar
start <camel-stream-bundle-id>
```

最後に、ルーターバンドルをデプロイします。

```
install file:///home/user/.m2/repository/com/example/myrouter/1.0/myrouter-1.0.jar
start <your-bundle-id>
```

198.3.3. KuraRouter ユーティリティー

Kura ルーターの基本クラスは、多くの便利なユーティリティーを提供します。このセクションでは、それぞれについて説明します。

198.3.3.1. SLF4J ロガー

Kura は、ロギング目的で SLF4J ファサードを使用します。保護されたメンバー **log** は、指定された Kura ルーターに関連付けられた SLF4J ロガーインスタンスを返します。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        log.info("Configuring Camel routes!");
        ...
    }

}
```

198.3.3.2. BundleContext

保護されたメンバー **bundleContext** は、指定された Kura ルーターに関連付けられたバンドルコンテキストを返します。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        ServiceReference<MyService> serviceRef =
bundleContext.getServiceReference(LogService.class.getName());
        MyService myService = bundleContext.getService(serviceRef);
        ...
    }

}
```

198.3.3.3. CamelContext

保護されたメンバー **camelContext** は、指定された Kura ルーターに関連付けられた **CamelContext** です。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        camelContext.getStatus();
        ...
```

```
    }

}
```

198.3.3.4. ProducerTemplate

保護されたメンバーの **producerTemplate** は、指定された Camel コンテキストに関連付けられた **ProducerTemplate** インスタンスです。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        producerTemplate.sendBody("jms:temperature", 22.0);
        ...
    }

}
```

198.3.3.5. ConsumerTemplate

保護されたメンバーの **consumerTemplate** は、指定された Camel コンテキストに関連付けられた **ConsumerTemplate** インスタンスです。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        double currentTemperature = producerTemplate.receiveBody("jms:temperature", Double.class);
        ...
    }

}
```

198.3.3.6. OSGi サービスリゾルバー

OSGi サービスリゾルバー (**service(Class<T> serviceType)**) を使用すると、OSGi バンドルコンテキストからタイプ別にサービスを簡単に取得できます。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        MyService myService = service(MyService.class);
        ...
    }

}
```

サービスが見つからない場合は、**null** 値が返されます。サービスが利用できない場合にアプリケーションを失敗させたい場合は、代わりに **requiredService(Class)** メソッドを使用してください。サービスが見つからない場合、**requiredService** は **IllegalStateException** を出力します。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        MyService myService = requiredService(MyService.class);
        ...
    }

}
```

198.3.4. KuraRouter アクティベーターのコールバック

Kura ルーターには、Camel ルーターの動作方法をカスタマイズするために使用できるライフサイクルコールバックが付属しています。たとえば、ルーターが開始される直前にルーターに関連付けられた **CamelContext** インスタンスを設定するには、**KuraRouter** クラスの **beforeStart** メソッドをオーバーライドします。

```
public class MyKuraRouter extends KuraRouter {

  ...

  protected void beforeStart(CamelContext camelContext) {
    OsgiDefaultCamelContext osgiContext = (OsgiCamelContext) camelContext;
    osgiContext.setName("NameOfTheRouter");
  }

}
```

198.3.5. ConfigurationAdmin からの XML ルートのロード

サーバー設定からルートの XML 定義を読み取ることが必要な場合があります。これは、OTA 再デプロイのコストが大きくなる可能性がある IoT ゲートウェイの一般的なシナリオです。この要件に対処するために、各 **KuraRouter** は、OSGi ConfigurationAdmin を使用して **kura.camel** PID から **kura.camel.BUNDLE-SYMBOLIC-NAME.route** プロパティーを探します。このアプローチにより、デプロイされた **KuraRouter** ごとに Camel XML ルートファイルを定義できます。ルートを更新するには、適切な設定プロパティーを編集し、それに関連付けられたバンドルを再起動するだけです。**kura.camel.BUNDLE-SYMBOLIC-NAME.route** プロパティーのコンテンツは、次のような Camel XML ルートファイルである必要があります。

```
<routes xmlns="http://camel.apache.org/schema/spring">
    <route id="loaded">
        <from uri="direct:bar"/>
        <to uri="mock:bar"/>
    </route>
</routes>
```

198.3.6. Kura ルーターを宣言型 OSGi サービスとしてデプロイする

Kura ルーターを宣言型 OSGi サービスとしてデプロイしたい場合は、**KuraRouter** が提供する **activate** メソッドと **activate** メソッドを使用できます。

```
<scr:component name="org.eclipse.kura.example.camel.MyKuraRouter" activate="activate"
deactivate="deactivate" enabled="true" immediate="true">
  <implementation class="org.eclipse.kura.example.camel.MyKuraRouter"/>
</scr:component>
```

198.3.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第199章 LANGUAGE コンポーネント

Camel バージョン 2.5 以降で利用可能

言語コンポーネントを使用すると、Camel でサポートされている言語のいずれかでスクリプトを実行するエンドポイントに Exchange を送信できます。
言語スクリプトを実行するコンポーネントを持つことで、より動的なルーティング機能が可能になります。たとえば、Routing Slip または Dynamic Router EIP を使用して、スクリプトが動的に定義されている **language** エンドポイントにメッセージを送信できます。

このコンポーネントは **camel-core** でそのまま提供されるため、追加の JAR は必要ありません。Groovy や JavaScript 言語を使用するなど、選択した言語で必要な場合にのみ、追加の Camel コンポーネントを含める必要があります。

199.1. URI 形式

```
language://languageName[:script][?options]
```

また、Camel 2.11 以降では、Camel の他の 言語 でサポートされているのと同じ表記法を使用して、スクリプトの外部リソースを参照できます。

```
language://languageName:resource:scheme:location][?options]
```

199.2. URI オプション

Language コンポーネントにはオプションがありません。

Language エンドポイントは、URI 構文を使用して設定されます。

```
language:languageName:resourceUri
```

パスおよびクエリーパラメーターを使用します。

199.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
languageName	**必須** 使用する言語の名前を設定します		String
resourceUri	リソースへのパス、またはリソースとして使用するレジストリー内の Bean をルックアップするための参照		String

199.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
binary (producer)	スクリプトがバイナリーコンテンツかテキストコンテンツか。デフォルトでは、スクリプトはテキストコンテンツとして読み込まれます (例: java.lang.String)。	false	boolean
cacheScript (producer)	コンパイルされたスクリプトをキャッシュして再利用するかどうかスクリプトを再利用すると、1つの Camel org.apache.camel.Exchange の処理から次の org.apache.camel.Exchange への副作用が発生する可能性があることに注意してください。	false	boolean
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
script (producer)	実行するスクリプトを設定します		String
transform (producer)	スクリプトの結果をメッセージ本文として使用するかどうか。このオプションのデフォルトは true です。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

199.3. メッセージヘッダー

次のメッセージヘッダーを使用して、コンポーネントの動作に影響を与えることができます。

ヘッダー	説明
CamelLanguageScript	ヘッダーで提供される実行するスクリプト。エンドポイントで設定されたスクリプトよりも優先されます。

199.4. 例

たとえば、Simple 言語を使用してメッセージの変換を行います。

メッセージボディーのタイプを変換する場合は、次のようにすることもできます。

入力メッセージが 2 で乗算されるこの例のように、Groovy 言語を使用することもできます。

以下に示すように、スクリプトをヘッダーとして提供することもできます。ここでは、XPath 言語を使用して **<foo>** タグからテキストを抽出します。

```
Object out = producer.requestBodyAndHeader("language:xpath", "<foo>Hello World</foo>",
Exchange.LANGUAGE_SCRIPT, "/foo/text()");
assertEquals("Hello World", out);
```

199.5. リソースからのスクリプトのロード

Camel 2.9 以降で利用可能

エンドポイント uri または **Exchange.LANGUAGE_SCRIPT** ヘッダーのいずれかで、ロードするスクリプトのリソース uri を指定できます。
URI は、次のスキームのいずれかで始まる必要があります: file:、classpath:、または http:

たとえば、クラスパスからスクリプトをロードするには:

デフォルトでは、スクリプトは一度ロードされてキャッシュされます。ただし、**contentCache** オプションを無効にして、評価ごとにスクリプトをロードすることができます。
たとえば、ファイル myscript.txt がディスク上で変更された場合、更新されたスクリプトが使用されます。

Camel 2.11 以降では、以下に示すように **resource:** を前に付けることで、Camel の他の Language と同様のリソースを参照できます。

第200章 LDAP コンポーネント

Camel バージョン 1.5 以降で利用可能

ldap コンポーネントを使用すると、フィルターをメッセージペイロードとして使用して、LDAP サーバーで検索を実行できます。
このコンポーネントは、標準の JNDI (**javax.naming** パッケージ) を使用してサーバーにアクセスします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ldap</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

200.1. URI 形式

```
ldap:ldapServerBean[?options]
```

URI の **ldapServerBean** 部分は、レジストリー内の DirContext Bean を参照します。LDAP コンポーネントはプロデューサーエンドポイントのみをサポートします。つまり、**ldap** URI はルートの最初の **from** に表示されません。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

200.2. オプション

LDAP コンポーネントにはオプションがありません。

LDAP エンドポイントは、URI 構文を使用して設定されます。

```
ldap:dirContextName
```

パスおよびクエリーパラメーターを使用します。

200.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
dirContextName	**必須** javax.naming.directory.DirContext、または java.util.Hashtable、またはレジストリー内でルックアップする Map Bean のいずれかの名前。Bean が Hashtable または Map の場合、使用ごとに新しい javax.naming.directory.DirContext インスタンスが作成されます。Bean が javax.naming.directory.DirContext の場合、Bean は指定されたとおりに使用されます。後者は、javax.naming.directory.DirContext を共有してはならないすべての状況で可能であるとは限りません。そのような状況では、代わりに java.util.Hashtable または Map を使用することをお勧めします。		String

200.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
base (producer)	検索用のベース DN。	ou=system	String
pageSize (producer)	指定すると、ldap モジュールはページングを使用してすべての結果を取得します (ほとんどの LDAP サーバーは、1 つのクエリーで 1000 を超えるエントリーを取得しようとすると例外を出力します)。これを使用できるようにするには、LdapContext (DirContext のサブクラス) を ldapServerBean として渡す必要があります (そうしないと、例外が出力されます)		Integer
returnedAttributes (producer)	結果の各エントリーに設定する必要がある属性のコンマ区切りリスト		String
scope (producer)	ベース DN から開始して、エントリーのツリーをどの程度深く検索するかを指定します。	subtree	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

200.3. 結果

結果は、Out ボディーで **ArrayList<javax.naming.directory.SearchResult>** オブジェクトとして返されます。

200.4. DIRCONTEXT

URI **ldap:ldapserver** は、ID **ldapserver** を持つ Spring Bean を参照します。 **ldapserver** Bean は次のように定義できます。

```
<bean id="ldapserver" class="javax.naming.directory.InitialDirContext" scope="prototype">
  <constructor-arg>
    <props>
      <prop key="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</prop>
      <prop key="java.naming.provider.url">ldap://localhost:10389</prop>
      <prop key="java.naming.security.authentication">none</prop>
    </props>
  </constructor-arg>
</bean>
```

前の例では、ローカルでホストされている LDAP サーバーに匿名で接続する通常の Sun ベースの LDAP **DirContext** を宣言しています。

注記

DirContext オブジェクトは、コントラクトによる同時実行をサポートする必要は **ありません**。したがって、**Bean** 定義で **scope="prototype"** を設定してディレクトリーコンテキストを宣言するか、コンテキストが同時実行をサポートすることが重要です。Spring フレームワークでは、**prototype** スコープのオブジェクトは、ルックアップされるたびにインスタンス化されます。

200.5. サンプル

上記の Spring 設定に続いて、以下のコードサンプルは LDAP リクエストを送信して、メンバーのグループをフィルター検索します。次に、共通名がレスポンスから抽出されます。

```
ProducerTemplate<Exchange> template = exchange
  .getContext().createProducerTemplate();

Collection<?> results = (Collection<?>) (template
  .sendBody(
    "ldap:ldapserver?base=ou=mygroup,ou=groups,ou=system",
    "(member=uid=huntc,ou=users,ou=system)"));

if (results.size() > 0) {
  // Extract what we need from the device's profile

  Iterator<?> resultIter = results.iterator();
  SearchResult searchResult = (SearchResult) resultIter
      .next();
  Attributes attributes = searchResult
      .getAttributes();
  Attribute deviceCNAttr = attributes.get("cn");
  String deviceCN = (String) deviceCNAttr.get();

  ...
```

特定のフィルターが必要ない場合 (たとえば、単一のエントリーを検索するだけでよい場合) は、ワイルドカードフィルター式を指定します。たとえば、LDAP エントリーに共通名がある場合は、次のようなフィルター式を使用します。

```
(cn=*)
```

200.5.1. 認証情報を使用したバインディング

Camel のエンドユーザーから、認証情報を使用して ldap サーバーにバインドするために使用したサンプルコードを寄贈されました。

```
Properties props = new Properties();
props.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
props.setProperty(Context.PROVIDER_URL, "ldap://localhost:389");
props.setProperty(Context.URL_PKG_PREFIXES, "com.sun.jndi.url");
props.setProperty(Context.REFERRAL, "ignore");
props.setProperty(Context.SECURITY_AUTHENTICATION, "simple");
props.setProperty(Context.SECURITY_PRINCIPAL, "cn=Manager");
props.setProperty(Context.SECURITY_CREDENTIALS, "secret");

SimpleRegistry reg = new SimpleRegistry();
reg.put("myldap", new InitialLdapContext(props, null));

CamelContext context = new DefaultCamelContext(reg);
context.addRoutes(
   new RouteBuilder() {
      public void configure() throws Exception {
         from("direct:start").to("ldap:myldap?base=ou=test");
      }
   }
);
context.start();

ProducerTemplate template = context.createProducerTemplate();

Endpoint endpoint = context.getEndpoint("direct:start");
Exchange exchange = endpoint.createExchange();
exchange.getIn().setBody("(uid=test)");
Exchange out = template.send(endpoint, exchange);

Collection<SearchResult> data = out.getOut().getBody(Collection.class);
assert data != null;
assert !data.isEmpty();

System.out.println(out.getOut().getBody());

context.stop();
```

200.6. SSL の設定

必要なのは、カスタムソケットファクトリーを作成し、それを InitialDirContext Bean で参照することだけです。以下のサンプルを参照してください。

SSL 設定

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
            http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">


  <sslContextParameters xmlns="http://camel.apache.org/schema/blueprint"
                id="sslContextParameters">
    <keyManagers
        keyPassword="{{keystore.pwd}}">
      <keyStore
          resource="{{keystore.url}}"
          password="{{keystore.pwd}}"/>
    </keyManagers>
  </sslContextParameters>

  <bean id="customSocketFactory" class="zotix.co.util.CustomSocketFactory">
    <argument ref="sslContextParameters" />
  </bean>
  <bean id="ldapserver" class="javax.naming.directory.InitialDirContext" scope="prototype">
    <argument>
      <props>
        <prop key="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
        <prop key="java.naming.provider.url" value="ldaps://lab.zotix.co:636"/>
        <prop key="java.naming.security.protocol" value="ssl"/>
        <prop key="java.naming.security.authentication" value="simple" />
        <prop key="java.naming.security.principal" value="cn=Manager,dc=example,dc=com"/>
        <prop key="java.naming.security.credentials" value="passw0rd"/>
        <prop key="java.naming.ldap.factory.socket"
            value="zotix.co.util.CustomSocketFactory"/>
      </props>
    </argument>
  </bean>
</blueprint>
```

カスタムソケットファクトリー

```
import org.apache.camel.util.jsse.SSLContextParameters;

import javax.net.SocketFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManagerFactory;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.security.KeyStore;

/**
 * The CustomSocketFactory. Loads the KeyStore and creates an instance of SSLSocketFactory
 */
public class CustomSocketFactory extends SSLSocketFactory {

  private static SSLSocketFactory socketFactory;

  /**
```

```
     * Called by the getDefault() method.
     */
    public CustomSocketFactory() {

    }

    /**
     * Called by Blueprint DI to initialise an instance of SocketFactory
     *
     * @param sslContextParameters
     */
    public CustomSocketFactory(SSLContextParameters sslContextParameters) {
        try {
            KeyStore keyStore =
sslContextParameters.getKeyManagers().getKeyStore().createKeyStore();
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
            tmf.init(keyStore);
            SSLContext ctx = SSLContext.getInstance("TLS");
            ctx.init(null, tmf.getTrustManagers(), null);
            socketFactory = ctx.getSocketFactory();
        } catch (Exception ex) {
            ex.printStackTrace(System.err);  /* handle exception */
        }
    }

    /**
     * Getter for the SocketFactory
     *
     * @return
     */
    public static SocketFactory getDefault() {
        return new CustomSocketFactory();
    }

    @Override
    public String[] getDefaultCipherSuites() {
        return socketFactory.getDefaultCipherSuites();
    }

    @Override
    public String[] getSupportedCipherSuites() {
        return socketFactory.getSupportedCipherSuites();
    }

    @Override
    public Socket createSocket(Socket socket, String string, int i, boolean bln) throws IOException {
        return socketFactory.createSocket(socket, string, i, bln);
    }

    @Override
    public Socket createSocket(String string, int i) throws IOException {
        return socketFactory.createSocket(string, i);
    }

    @Override
    public Socket createSocket(String string, int i, InetAddress ia, int i1) throws IOException {
```

```
        return socketFactory.createSocket(string, i, ia, i1);
    }

    @Override
    public Socket createSocket(InetAddress ia, int i) throws IOException {
        return socketFactory.createSocket(ia, i);
    }

    @Override
    public Socket createSocket(InetAddress ia, int i, InetAddress ia1, int i1) throws IOException {
        return socketFactory.createSocket(ia, i, ia1, i1);
    }
}
```

200.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第201章 LDIF コンポーネント

Camel バージョン 2.20 以降で利用可能

ldif コンポーネントを使用すると、LDIF ボディーのコンテンツから LDAP サーバーで更新を行うことができます。

このコンポーネントは、基本的な URL 構文を使用してサーバーにアクセスします。Apache DS LDAP ライブラリーを使用して LDIF を処理します。LDIF を処理した後、レスポンスボディーは各エントリーの成功/失敗のステータスのリストになります。

注記

Apache LDAP API は、LDIF 構文エラーに非常に敏感です。不明な場合は、単体テストを参照して、各変更タイプの例を確認してください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ldif</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

201.1. URI 形式

```
ldap:ldapServerBean[?options]
```

URI の **ldapServerBean** 部分は LdapConnection を参照します。これは、接続タイムアウトを回避するために、使用時にファクトリーから構築する必要があります。LDIF コンポーネントはプロデューサーエンドポイントのみをサポートします。つまり、**ldif** URI はルートの先頭の **from** に表示できません。

SSL 設定については、カスタム SocketFactory インスタンスをセットアップする例がある **camel-ldap** コンポーネントを参照してください。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

201.2. オプション

LDIF コンポーネントにはオプションがありません。

LDIF エンドポイントは、URI 構文を使用して設定されます。

```
ldif:ldapConnectionName
```

パスおよびクエリーパラメーターを使用します。

201.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
ldapConnectionName	**必須** レジストリーから取得する LdapConnection Bean の名前。これは、スレッド間で共有されたり、タイムアウトした接続を使用したりしないように、スコーププロトタイプでなければならないことに注意してください。		String

201.2.2. クエリーパラメーター(1 個のパラメーター):

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

201.3. ボディータイプ

本文は、LDIF ファイルまたはインライン LDIF ファイルへの URL にすることができます。ボディータイプの違いを示すには、インライン LDIF を次のように開始する必要があります。

```
version: 1
```

そうでない場合、コンポーネントはボディーを URL として解析しようとします。

201.4. 結果

結果は、Out ボディーで **ArrayList<java.lang.String>** オブジェクトとして返されます。これには、LDIF エントリーごとに成功または例外メッセージが含まれます。

201.5. LDAPCONNECTION

URI **ldif:ldapConnectionName** は、ID **ldapConnectionName** を持つ Bean を参照します。ldapConnection は、**LdapConnectionConfig** Bean を使用して設定できます。接続が共有されたり、古い接続を取得したりしないように、スコープには **prototype** のスコープが必要であることに注意してください。

LdapConnection Bean は、Spring XML で次のように定義できます。

```
<bean id="ldapConnectionOptions"
class="org.apache.directory.ldap.client.api.LdapConnectionConfig">
  <property name="ldapHost" value="${ldap.host}"/>
  <property name="ldapPort" value="${ldap.port}"/>
  <property name="name" value="${ldap.username}"/>
  <property name="credentials" value="${ldap.password}"/>
  <property name="useSsl" value="false"/>
  <property name="useTls" value="false"/>
</bean>
```

```
<bean id="ldapConnectionFactory"
class="org.apache.directory.ldap.client.api.DefaultLdapConnectionFactory">
  <constructor-arg index="0" ref="ldapConnectionOptions"/>
</bean>

<bean id="ldapConnection" factory-bean="ldapConnectionFactory" factory-
method="newLdapConnection" scope="prototype"/>
```

または OSGi blueprint.xml で:

```
<bean id="ldapConnectionOptions"
class="org.apache.directory.ldap.client.api.LdapConnectionConfig">
  <property name="ldapHost" value="${ldap.host}"/>
  <property name="ldapPort" value="${ldap.port}"/>
  <property name="name" value="${ldap.username}"/>
  <property name="credentials" value="${ldap.password}"/>
  <property name="useSsl" value="false"/>
  <property name="useTls" value="false"/>
</bean>

<bean id="ldapConnectionFactory"
class="org.apache.directory.ldap.client.api.DefaultLdapConnectionFactory">
  <argument ref="ldapConnectionOptions"/>
</bean>

<bean id="ldapConnection" factory-ref="ldapConnectionFactory" factory-
method="newLdapConnection" scope="prototype"/>
```

201.6. サンプル

上記の Spring 設定に続いて、以下のコードサンプルは LDAP リクエストを送信して、メンバーのグループをフィルター検索します。次に、共通名がレスポンスから抽出されます。

```
ProducerTemplate<Exchange> template = exchange.getContext().createProducerTemplate();

List<?> results = (Collection<?>) template.sendBody("ldap:ldapConnection, "LDiff goes here");

if (results.size() > 0) {
  // Check for no errors

  for (String result : results) {
    if ("success".equalTo(result)) {
      // LDIF entry success
    } else {
      // LDIF entry failure
    }
  }
}
```

201.7. LEVELDB

Camel 2.10 以降で利用可能

Leveldb は、非常に軽量で組み込み可能なキー値データベースです。Camel と一緒に、Aggregator などのさまざまな Camel 機能の永続的なサポートを提供できます。

それが提供する現在の機能:

- LevelDBAggregationRepository

201.7.1. LevelDBAggregationRepository の使用

LevelDBAggregationRepository は **AggregationRepository** であり、その場で集約されたメッセージを永続化します。デフォルトのアグリゲーターはメモリー内のみの **AggregationRepository** を使用するため、これによりメッセージが失われないことが保証されます。

以下のオプションがあります。

オプション	タイプ	説明
repositoryName	String	必須のリポジトリー名。複数のリポジトリーで共有 **LevelDBFile** を使用できるようにします。
persistentFileName	String	永続ストレージのファイル名。起動時にファイルが存在しない場合は、新しいファイルが作成されます。
levelDBFile	LevelDBFile	既存の設定済み **org.apache.camel.component.leveldb.LevelDBFile** インスタンスを使用します。
sync	boolean	**Camel 2.12:** LevelDBFile が書き込み時に同期するかどうか。デフォルトは false です。書き込み時に同期することにより、すべての書き込みがディスクにスプールされるのを常に待機するため、更新が失われることはありません。非同期書き込みと同期書き込みの詳細については、LevelDB のドキュメント を参照してください。
returnOldExchange	boolean	古い既存の Exchange が存在する場合、get 操作でそれを返すかどうか。デフォルトでは、集約時に古いエクスチェンジは必要ないため、最適化するにはこのオプションは **false** です。
useRecovery	boolean	リカバリーが有効かどうか。このオプションはデフォルトで **true** です。Camel Aggregator の自動回復を有効にすると、集約されたエクスチェンジに失敗し、それらが再送信されます。
recoveryInterval	long	回復が有効になっている場合、x 回ごとにバックグラウンドタスクが実行され、失敗した交換をスキャンして回復し、再送信します。デフォルトでは、この間隔は 5000 ミリ秒です。
maximumRedeliveries	int	回復されたエクスチェンジの再配信試行の最大回数を制限できます。有効にすると、すべての再配信の試行が失敗した場合、エクスチェンジはデッドレターチャネルに移動されます。デフォルトでは、このオプションは無効となっています。このオプションを使用する場合は、**deadLetterUri** オプションも指定する必要があります。

オプション	タイプ	説明
deadLetterUri	String	枯渇した回復済みエクスチェンジが移動されるデッドレターチャネルのエンドポイント uri。このオプションを使用する場合、**maximumRedeliveries** オプションも指定する必要があります。

repositoryName オプションを指定する必要があります。次に、**persistentFileName** または **levelDBFile** を指定する必要があります。

201.7.2. 持続時に保持されるもの

LevelDBAggregationRepository は、**Serializable** な互換性のあるメッセージ本文のデータ型のみを保持します。メッセージヘッダーは、プリミティブ/文字列/数字/などである必要があります。データ型がそのような型でない場合は削除され、**WARN** がログに記録されます。また、**Message** ボディーと **Message** ヘッダーのみを保持します。**Exchange** プロパティーは保持され **ません**。

201.7.3. 復元

LevelDBAggregationRepository はデフォルトで、失敗した Exchange を回復します。これは、永続ストア内の失敗したエクスチェンジをスキャンするバックグラウンドタスクを持つことによって行われます。**checkInterval** オプションを使用して、このタスクの実行頻度を設定できます。回復はトランザクションとして機能するため、Camel は失敗したエクスチェンジを回復して再配信しようとします。回復されたことが判明したエクスチェンジは、永続ストアから復元され、再送信され、再度送信されます。

次のヘッダーは、エクスチェンジが回復または再配信されるときに設定されます。

ヘッダー	タイプ	説明
Exchange.REDELIVERED	Boolean	エクスチェンジが再配信されていることを示すために true に設定されます。
Exchange.REDELIVERY_COUNTER	Integer	1 から始まる再配信の試行。

エクスチェンジが正常に処理された場合にのみ、**AggregationRepository** で **confirm** メソッドが呼び出されたときに完了としてマークされます。これは、同じエクスチェンジが再び失敗した場合、成功するまで再試行されることを意味します。

オプション **maximumRedeliveries** を使用して、復元された特定の Exchange の再配信試行の最大回数を制限できます。**maximumRedeliveries** に達したときに Camel がエクスチェンジの送信先を認識できるように、**deadLetterUri** オプションも設定する必要があります。

このテスト など、camel-leveldb の単体テストでいくつかの例を確認できます。

201.7.3.1. Java DSL での LevelDBAggregationRepository の使用

この例では、集計されたメッセージを **target/data/leveldb.dat** ファイルに保存します。

201.7.3.2. Spring XML での LevelDBAggregationRepository の使用

同じ例ですが、代わりに Spring XML を使用しています。

201.7.4. 依存関係

camel ルートで LevelDB を使用するには、**camel-leveldb** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-leveldb</artifactId>
  <version>2.10.0</version>
</dependency>
```

201.7.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- アグリゲーター
- HawtDB
- コンポーネント

第202章 LINKEDIN コンポーネント

Camel バージョン 2.14 以降で利用可能

LinkedIn コンポーネントは、https://developer.linkedin.com/rest に記載されているすべての LinkedIn REST API へのアクセスを提供します。

LinkedIn は、すべてのクライアントアプリケーション認証に OAuth2.0 を使用します。アカウントで camel-linkedin を使用するには、https://www.linkedin.com/secure/developer で LinkedIn 用の新しいアプリケーションを作成する必要があります。LinkedIn アプリケーションのクライアント ID とシークレットは、現在のユーザーを必要とする LinkedIn REST API へのアクセスを許可します。ユーザーアクセストークンは、エンドユーザーのコンポーネントによって生成および管理されます。または、Camel アプリケーションは org.apache.camel.component.linkedin.api.OAuthSecureStorage の実装を登録して、org.apache.camel.component.linkedin.apiOAuthToken OAuth トークンを提供できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-linkedin</artifactId>
    <version>${camel-version}</version>
</dependency>
```

202.1. URI 形式

```
linkedin://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- コメント
- companies
- groups
- jobs
- ユーザー
- posts
- search

202.2. LINKEDINCOMPONENT

Linkedin コンポーネントは、以下に示す 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用する場合。		LinkedInConfiguration

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Linkedin エンドポイントは、URI 構文を使用して設定されます。

```
linkedin:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

202.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		LinkedInApiName
methodName	**必須**: 選択した操作に使用するサブ操作		文字列

202.2.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
accessToken (common)	ユーザー名とパスワードのログイン手順を回避するための LinkedIn アクセストークン。LinkedIn は、CAPTCHA を使用してログインフォームに応答します。これにより、スタンドアロンのヘッドレスプロセスが、ユーザー名とパスワードを指定して LinkedIn にログインできなくなります。これを回避するには、LinkedIn アクセストークンを取得し、そのトークンを accessToken パラメーターの設定として指定します。LinkedIn アクセストークンの取得は、複数の手順からなります。LinkedIn アプリケーションを設定し、LinkedIn 認証コードを取得して、そのコードを LinkedIn アクセストークンと交換する必要があります。詳細は、https://developer.linkedin.com/docs/oauth2 を参照してください。デフォルトの動作では、アクセストークンは 60 日後に期限切れになります。これを変更するには、expiryTime パラメーターの値を指定します。アクセストークンの有効期限が切れると、LinkedIn コンポーネントはユーザー名とパスワードを提供して LinkedIn へのログインを試みますが、その結果 CAPTCHA が発生するため、ログインは失敗します。LinkedIn コンポーネントはアクセストークンを更新できません。アクセストークンの有効期限が切れるたびに、新しいアクセストークンを手動で取得する必要があります。アクセストークンを更新するときは、新しいトークンを使用するようにアプリケーションを再起動する必要があります。		文字列
clientId (common)	LinkedIn アプリケーションクライアント ID		文字列
clientSecret (Common)	LinkedIn アプリケーションのクライアントシークレット		文字列
expiryTime (common)	UNIX エポックからのミリ秒数。デフォルトは 60 日です。LinkedIn アクセストークンは、トークンが使用されてからこの時間が経過すると期限切れになります。		Long
httpParams (common)	プロキシーホストやポートなどのカスタム HTTP パラメーター。AllClientPNames の定数を使用します。		Map
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		文字列
lazyAuth (common)	遅延 OAuth を有効/無効にするフラグ。デフォルトは true です。有効にすると、最初の REST 呼び出しまで OAuth トークンの取得または生成は行われません	true	boolean

名前	説明	デフォルト	タイプ
redirectUri (common)	アプリケーションのリダイレクト URI。ただし、リダイレクトサーバーが機能しなくても済むように、コンポーネントはこのページにリダイレクトしません。テストには、https://localhost を使用できます。		文字列
scopes (Common)	https://developer.linkedin.com/documents/authentication#granting で指定されている LinkedIn スコープのリスト		OAuthScope[]
secureStorage (common)	OAuth トークンを提供するため、またはコンポーネントによって生成されたトークンを保存するためのコールバックインターフェイス。コールバックは、最初の呼び出しで null を返し、作成されたトークンを saveToken() コールバックに保存する必要があります。コールバックが最初に null を返したら、userPassword を指定する必要があります。		OAuthSecureStorage
userName (common)	LinkedIn ユーザーアカウント名を指定する必要があります		文字列
userPassword (common)	LinkedIn アカウントのパスワード		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

202.3. SPRING BOOT AUTO-CONFIGURATION

コンポーネントは、以下に記載される 15 のオプションをサポートします。

名前	説明	デフォルト	タイプ
camel.component.linkedin.configuration.access-token	ユーザー名とパスワードのログイン手順を回避するための LinkedIn アクセストークン。LinkedIn は、CAPTCHA を使用してログインフォームに応答します。これにより、スタンドアロンのヘッドレスプロセスが、ユーザー名とパスワードを指定して LinkedIn にログインできなくなります。これを回避するには、LinkedIn アクセストークンを取得し、そのトークンを accessToken パラメーターの設定として指定します。LinkedIn アクセストークンの取得は、複数の手順からなります。LinkedIn アプリケーションを設定し、LinkedIn 認証コードを取得して、そのコードを LinkedIn アクセストークンと交換する必要があります。詳細は、https://developer.linkedin.com/docs/oauth2 を参照してください。デフォルトの動作では、アクセストークンは 60 日後に期限切れになります。これを変更するには、expiryTime パラメーターの値を指定します。アクセストークンの有効期限が切れると、LinkedIn コンポーネントはユーザー名とパスワードを提供して LinkedIn へのログインを試みますが、その結果 CAPTCHA が発生するため、ログインは失敗します。LinkedIn コンポーネントはアクセストークンを更新できません。アクセストークンの有効期限が切れるたびに、新しいアクセストークンを手動で取得する必要があります。アクセストークンを更新するときは、新しいトークンを使用するようにアプリケーションを再起動する必要があります。		文字列
camel.component.linkedin.configuration.api-name	実行する操作の種類		LinkedInApiName
camel.component.linkedin.configuration.client-id	LinkedIn アプリケーションクライアント ID		文字列
camel.component.linkedin.configuration.client-secret	LinkedIn アプリケーションのクライアントシークレット		文字列
camel.component.linkedin.configuration.expiry-time	UNIX エポックからのミリ秒数。デフォルトは 60 日です。LinkedIn アクセストークンは、トークンが使用されてからこの時間が経過すると期限切れになります。		Long

名前	説明	デフォルト	タイプ
camel.component.linkedin.configuration.http-params	プロキシーホストやポートなどのカスタム HTTP パラメーター。AllClientPNames の定数を使用します。		Map
camel.component.linkedin.configuration.lazy-auth	遅延 OAuth を有効/無効にするフラグ。デフォルトは true です。有効にすると、最初の REST 呼び出しまで OAuth トークンの取得または生成は行われません	true	Boolean
camel.component.linkedin.configuration.method-name	選択した操作に使用するサブ操作		文字列
camel.component.linkedin.configuration.redirect-uri	アプリケーションのリダイレクト URI。ただし、リダイレクトサーバーが機能しなくても済むように、コンポーネントはこのページにリダイレクトしません。テストには、https://localhost を使用できます。		文字列
camel.component.linkedin.configuration.scopes	https://developer.linkedin.com/documents/authentication#granting で指定されている LinkedIn スコープのリスト		OAuthScope[]
camel.component.linkedin.configuration.secure-storage	OAuth トークンを提供するため、またはコンポーネントによって生成されたトークンを保存するためのコールバックインターフェイス。コールバックは、最初の呼び出しで null を返し、作成されたトークンを saveToken() コールバックに保存する必要があります。コールバックが最初に null を返したら、userPassword を指定する必要があります。		OAuthSecureStorage
camel.component.linkedin.configuration.user-name	LinkedIn ユーザーアカウント名を指定する必要があります		文字列
camel.component.linkedin.configuration.user-password	LinkedIn アカウントのパスワード		文字列
camel.component.linkedin.enabled	linkedin コンポーネントを有効にする	true	Boolean
camel.component.linkedin.resolve-property-placeholders	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	Boolean

202.4. プロデューサーエンドポイント:

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。一部のエンドポイントには省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションはで示されます。エンドポイントに必須のオプションがない場合、オプションのセットの 1 つを提供する必要があります。プロデューサエンドポイントは、特別なオプション **inBody** を使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelLinkedIn.<option>** の形式にする必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelLinkedIn.option** ヘッダーをオーバーライドします。

エンドポイントとオプションの詳細は、https://developer.linkedin.com/rest にある LinkedIn REST API のドキュメントを参照してください。

202.4.1. エンドポイント接頭辞 comments

次のエンドポイントは、次のように接頭辞 **comments** を使用して呼び出すことができます。

```
linkedin://comments/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getComment	comment	comment_id、fields	org.apache.camel.component.linkedin.api.model.Comment
removeComment	comment	comment_id	

comments の URI オプション

名前	タイプ
comment_id	文字列
fields	文字列

202.4.2. エンドポイント接頭辞 companies

次のエンドポイントは、次のように接頭辞 **companies** を使用して呼び出すことができます。

```
linkedin://companies/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addCompanyUpdateComment	companyUpdateComment	company_id、update_key、updatecomment	
addCompanyUpdateCommentAsCompany	companyUpdateCommentAsCompany	company_id、update_key、updatecomment	
addShare	share	company_id、share	
getCompanies	companies	email_domain、fields、is_company_admin	org.apache.camel.component.linkedin.api.model.Companies
getCompanyById	companyById	company_id、fields	org.apache.camel.component.linkedin.api.model.Company
getCompanyByName	companyByName	fields、universal_name	org.apache.camel.component.linkedin.api.model.Company
getCompanyUpdateComments	companyUpdateComments	company_id、fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Comments
getCompanyUpdateLikes	companyUpdateLikes	company_id、fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Likes
getCompanyUpdates	companyUpdates	company_id、count、event_type、fields、start	org.apache.camel.component.linkedin.api.model.Updates
getHistoricalFollowStatistics	historicalFollowStatistics	company_id、end_timestamp、start_timestamp、time_granularity	org.apache.camel.component.linkedin.api.model.HistoricalFollowStatistics
getHistoricalStatusUpdateStatistics	historicalStatusUpdateStatistics	company_id、end_timestamp、start_timestamp、time_granularity、update_key	org.apache.camel.component.linkedin.api.model.HistoricalStatusUpdateStatistics

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getNumberOfFollowers	numberOfFollowers	companySizes、company_id、geos、industries、jobFunc、seniorities	org.apache.camel.component.linkedin.api.model.NumFollowers
getStatistics	statistics	company_id	org.apache.camel.component.linkedin.api.model.CompanyStatistics
isShareEnabled		company_id	org.apache.camel.component.linkedin.api.model.IsCompanyShareEnabled
isViewerShareEnabled		company_id	org.apache.camel.component.linkedin.api.model.IsCompanyShareEnabled
likeCompanyUpdate		company_id、isliked、update_key	

companies の URI オプション

エンドポイント URI またはメッセージヘッダーのいずれかの **[companySizes、count、email_domain、end_timestamp、event_type、geos、industries、is_company_admin、jobFunc、secure_urls、seniorities、start、start_timestamp、time_granularity]** オプションのいずれかに値が指定されていない場合は、**null** と見なされます。**null** 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

名前	タイプ
companySizes	java.util.List
company_id	Long
count	Long
email_domain	文字列
end_timestamp	Long
event_type	org.apache.camel.component.linkedin.api.Eventtype
fields	文字列
geos	java.util.List

名前	タイプ
industries	java.util.List
is_company_admin	Boolean
isliked	org.apache.camel.component.linkedin.api.model.IsLiked
jobFunc	java.util.List
secure_urls	Boolean
seniorities	java.util.List
share	org.apache.camel.component.linkedin.api.model.Share
start	Long
start_timestamp	Long
time_granularity	org.apache.camel.component.linkedin.api.Timegranularity
universal_name	文字列
update_key	文字列
updatecomment	org.apache.camel.component.linkedin.api.model.UpdateComment

202.4.3. エンドポイント接頭辞 groups

次のエンドポイントは、次のように接頭辞 **groups** を使用して呼び出すことができます。

```
linkedin://groups/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addPost	post	group_id、post	
getGroup	group	group_id	org.apache.camel.component.linkedin.api.model.Group

groups の URI オプション

名前	タイプ
group_id	Long
post	org.apache.camel.component.linkedin.api.model.Post

202.4.4. エンドポイント接頭辞 jobs

次のエンドポイントは、次のように接頭辞 **jobs** で呼び出すことができます。

```
linkedin://jobs/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addJob	job	job	
editJob		job、partner_job_id	
getJob	job	fields, job_id	org.apache.camel.component.linkedin.api.model.Job

jobs の URI オプション

名前	タイプ
fields	文字列
job	org.apache.camel.component.linkedin.api.model.Job
job_id	Long
partner_job_id	Long

202.4.5. エンドポイント接頭辞 people

次のエンドポイントは、次のように接頭辞 **people** で呼び出すことができます。

```
linkedin://people/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addActivity	activity	activity	
addGroupMembership	groupMembership	groupmembership	

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addInvite	invite	mailboxitem	
addJobBookmark	jobBookmark	jobbookmark	
addUpdateComment	updateComment	update_key、updatecomment	
followCompany		company	
getConnections	connections	fields、secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getConnectionsById	connectionsById	fields、person_id、secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getConnectionsByUrl	connectionsByUrl	fields, public_profile_url, secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getFollowedCompanies	followedCompanies	fields	org.apache.camel.component.linkedin.api.model.Companies
getGroupMembershipSettings	groupMembershipSettings	count、fields、group_id、start	org.apache.camel.component.linkedin.api.model.GroupMemberships
getGroupMemberships	groupMemberships	count、fields、membership_state、start	org.apache.camel.component.linkedin.api.model.GroupMemberships
getJobBookmarks	jobBookmarks		org.apache.camel.component.linkedin.api.model.JobBookmarks
getNetworkStats	networkStats		org.apache.camel.component.linkedin.api.model.NetworkStats
getNetworkUpdates	networkUpdates	after、before、count、fields、scope、secure_urls、show_hidden_members、start、type	org.apache.camel.component.linkedin.api.model.Updates

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
getNetworkUpdatesById	networkUpdatesById	after、before、count、fields、person_id、scope、secure_urls、show_hidden_members、start、type	org.apache.camel.component.linkedin.api.model.Updates
getPerson	person	fields、secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPersonById	personById	fields、person_id、secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPersonByUrl	personByUrl	fields, public_profile_url, secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPosts	posts	category、count、fields、group_id、modified_since、order、role、start	org.apache.camel.component.linkedin.api.model.Posts
getSuggestedCompanies	suggestedCompanies	fields	org.apache.camel.component.linkedin.api.model.Companies
getSuggestedGroupPosts	suggestedGroupPosts	category、count、fields、group_id、modified_since、order、role、start	org.apache.camel.component.linkedin.api.model.Posts
getSuggestedGroups	suggestedGroups	fields	org.apache.camel.component.linkedin.api.model.Groups
getSuggestedJobs	suggestedJobs	fields	org.apache.camel.component.linkedin.api.model.JobSuggestions
getUpdateComments	updateComments	fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Comments
getUpdateLikes	updateLikes	fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Likes
likeUpdate		isliked、update_key	

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
removeGroupMembership	groupMembership	group_id	
removeGroupSuggestion	groupSuggestion	group_id	
removeJobBookmark	jobBookmark	job_id	
share		share	org.apache.camel.component.linkedin.api.model.Update
stopFollowingCompany		company_id	
updateGroupMembership		group_id、groupmembership	

people の URI オプション

エンドポイント URI またはメッセージヘッダーのいずれかの **[after、before、category、count、membership_state、modified_since、order、public_profile_url、role、scope、secure_urls、show_hidden_members、start、type]** オプションのいずれかに値が指定されていない場合は、**null** と見なされます。**null** 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

名前	タイプ
activity	org.apache.camel.component.linkedin.api.model.Activity
after	Long
before	Long
category	org.apache.camel.component.linkedin.api.Category
company	org.apache.camel.component.linkedin.api.model.Company
company_id	Long
count	Long
fields	文字列

名前	タイプ
group_id	Long
groupmembership	org.apache.camel.component.linkedin.api.model.GroupMembership
isliked	org.apache.camel.component.linkedin.api.model.IsLiked
job_id	Long
jobbookmark	org.apache.camel.component.linkedin.api.model.JobBookmark
mailboxitem	org.apache.camel.component.linkedin.api.model.MailboxItem
membership_state	org.apache.camel.component.linkedin.api.model.MembershipState
modified_since	Long
order	org.apache.camel.component.linkedin.api.Order
person_id	文字列
public_profile_url	文字列
role	org.apache.camel.component.linkedin.api.Role
scope	文字列
secure_urls	Boolean
share	org.apache.camel.component.linkedin.api.model.Share
show_hidden_members	Boolean
start	Long
type	org.apache.camel.component.linkedin.api.Type
update_key	文字列
updatecomment	org.apache.camel.component.linkedin.api.model.UpdateComment

202.4.6. エンドポイント接頭辞 posts

次のエンドポイントは、次のように接頭辞 **posts** で呼び出すことができます。

```
linkedin://posts/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
addComment	comment	comment、post_id	
flagCategory		post_id, postcategorycode	
followPost		isfollowing, post_id	
getPost	post	count、fields、post_id、start	org.apache.camel.component.linkedin.api.model.Post
getPostComments	postComments	count、fields、post_id、start	org.apache.camel.component.linkedin.api.model.Comments
likePost		isliked, post_id	
removePost	post	post_id	

posts の URI オプション

エンドポイント URI またはメッセージヘッダーのいずれかの **[count、start]** オプションのいずれかに値が指定されていない場合は、**null** と見なされます。**null** 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

名前	タイプ
comment	org.apache.camel.component.linkedin.api.model.Comment
count	Long
fields	文字列
isfollowing	org.apache.camel.component.linkedin.api.model.IsFollowing
isliked	org.apache.camel.component.linkedin.api.model.IsLiked

名前	タイプ
post_id	文字列
postcategorycode	org.apache.camel.component.linkedin.api.model.PostCategoryCode
start	Long

202.4.7. エンドポイント接頭辞 search

次のエンドポイントは、次のように接頭辞 **search** で呼び出すことができます。

```
linkedin://search/endpoint?[options]
```

エンドポイント	短縮形エイリアス	オプション	結果ボディーのタイプ
searchCompanies	companies	count、facet、facets、fields、hq_only、keywords、sort、start	org.apache.camel.component.linkedin.api.model.CompanySearch
searchJobs	jobs	company_name、count、country_code、distance、facet、facets、fields、job_title、keywords、postal_code、sort、start	org.apache.camel.component.linkedin.api.model.JobSearch
searchPeople	ユーザー	company_name、count、country_code、current_company、current_school、current_title、distance、facet、facets、fields、first_name、keywords、last_name、postal_code、school_name、sort、start、title	org.apache.camel.component.linkedin.api.model.PeopleSearch

search の URI オプション

エンドポイント URI またはメッセージヘッダーのいずれかの **[company_name、count、country_code、current_company、current_school、current_title、distance、facet、facets、first_name、hq_only、job_title、keywords、last_name、postal_code、school_name、sort、start、title]** オプションのいずれかに値が指定されていない場合は、**null** と見なされます。**null** 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

名前	タイプ
company_name	文字列
count	Long
country_code	文字列
current_company	文字列
current_school	文字列
current_title	文字列
distance	org.apache.camel.component.linkedin.api.model.Distance
facet	文字列
ファセット	文字列
fields	文字列
first_name	文字列
hq_only	文字列
job_title	文字列
keywords	文字列
last_name	文字列
postal_code	文字列
school_name	文字列
sort	文字列
start	Long
title	文字列

202.5. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Optionsと**consumer.**の接頭辞を使用して、エンドポ

イントの呼び出しをスケジュールすることができます。デフォルトでは、配列またはコレクションを返す Consumer エンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。この動作を変更するには、プロパティー **consumer.splitResults=true** を使用して、リストまたは配列全体の単一のエクスチェンジを返します。

202.6. メッセージヘッダー

CamelLinkedIn. 接頭辞を使用するプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

202.7. メッセージボディー

すべての結果メッセージ本文は、Apache CXF JAX-RS を使用して構築された Camel LinkedIn API SDK によって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

202.8. ユースケース

次のルートは、ユーザーのプロファイルを取得します。

```
from("direct:foo")
    .to("linkedin://people/person");
```

次のルートは、ユーザーの接続を 30 秒ごとにポーリングします。

```
from("linkedin://people/connections?consumer.timeUnit=SECONDS&consumer.delay=30")
    .to("bean:foo");
```

次のルートでは、動的ヘッダーオプションを持つプロデューサーを使用します。 **personId** ヘッダーには LinkedIn の個人 ID が含まれているため、次のように **CamelLinkedIn.person_id** ヘッダーに割り当てられます。

```
from("direct:foo")
    .setHeader("CamelLinkedIn.person_id", header("personId"))
    .to("linkedin://people/connectionsById")
    .to("bean://bar");
```

第203章 LOG コンポーネント

Camel バージョン 1.1 以降で利用可能

log: コンポーネントは、メッセージエクスチェンジを基になるロギングメカニズムに記録します。

Camel は sfl4j を使用します。これにより、特に次の方法でロギングを設定できます。

- Log4j
- Logback
- Java Util Logging

203.1. URI 形式

```
log:loggingCategory[?options]
```

loggingCategory は、使用するログカテゴリーの名前です。URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

情報:*レジストリーからのロガーインスタンスの使用* **Camel 2.12.4/2.13.1** の時点で、レジストリーに **org.slf4j.Logger** の単一のインスタンスが見つかった場合、**loggingCategory** はロガーインスタンスの作成に使用されなくなりました。登録されたインスタンスが代わりに使用されます。また、**?logger=#myLogger** URI パラメーターを使用して、特定の **Logger** インスタンスを参照することもできます。最終的に、登録された URI **logger** パラメーターがない場合、ロガーインスタンスは **loggingCategory** を使用して作成されます。

たとえば、ログエンドポイントは通常、次のように **level** オプションを使用してログレベルを指定します。

```
log:org.apache.camel.example?level=DEBUG
```

デフォルトのロガーは、すべての交換をログに記録します (**通常のログ記録**)。ただし、Camel には **Throughput** ロガーも同梱されており、これは **groupSize** オプションが指定されている場合に常に使用されます。

ヒント:* DSL にもログ* DSL に直接 **log** もありますが、目的が異なります。軽量で人間のログ用です。詳細については、LogEIP を参照してください。

203.2. オプション

ログ コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
exchangeFormatter (advanced)	カスタム ExchangeFormatter を設定して、Exchange をログに適した文字列に変換します。指定しない場合は、デフォルトで DefaultExchangeFormatter になります。		ExchangeFormatter

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ログエンドポイントは、URI 構文を使用して設定されます。

```
log:loggerName
```

パスおよびクエリーパラメーターを使用します。

203.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
loggerName	***必須*** 使用するロガー名		String

203.2.2. クエリーパラメーター(26 個のパラメーター):

名前	説明	デフォルト	タイプ
groupActiveOnly (producer)	true の場合、新しいメッセージが一定期間受信されていない場合に統計を非表示にします。false の場合、メッセージトラフィックに関係なく統計を表示します。	true	Boolean
groupDelay (producer)	統計の初期遅延を設定します (ミリ秒単位)		Long
groupInterval (producer)	指定すると、この時間間隔 (ミリ秒単位) でメッセージ統計がグループ化されます		Long
groupSize (producer)	スループットログのグループサイズを指定する整数。		Integer
level (producer)	使用するロギングレベル。デフォルト値は INFO です。	INFO	String
logMask (producer)	true の場合、ログ内のパスワードやパスフレーズなどの機密情報をマスクします。		Boolean

名前	説明	デフォルト	タイプ
marker (producer)	使用するオプションのマーカー名。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
maxChars (formatting)	1 行に記録される文字数を制限します。	10000	int
multiline (formatting)	有効にすると、各情報が改行で出力されます。	false	boolean
showAll (formatting)	すべてのオプションをオンにするためのクイックオプション。(複数行、maxChars を使用する場合は手動で設定する必要があります)	false	boolean
showBody (formatting)	メッセージ本文を表示します。	true	boolean
showBodyType (formatting)	本文の Java タイプを表示します。	true	boolean
showCaughtException (formatting)	エクスチェンジでキャッチされた例外がある場合は、例外メッセージを表示します (スタックトレースなし)。キャッチされた例外は、エクスチェンジのプロパティーとして保存され (キーリンク org.apache.camel.ExchangeEXCEPTION_CAUGHT を使用)、たとえば doCatch は例外をキャッチできます。	false	boolean
showException (formatting)	交換に例外がある場合は、例外メッセージを表示します (スタックトレースなし)	false	boolean
showExchangeId (formatting)	一意の取引所 ID を表示します。	false	boolean
showExchangePattern (formatting)	メッセージ交換パターン (略して MEP) を表示します。	true	boolean
showFiles (formatting)	有効にすると、Camel はファイルを出力します	false	boolean
showFuture (formatting)	有効にすると、Camel は Future オブジェクトで、ログに記録されるペイロードを取得するために完了するのを待ちます。	false	boolean

名前	説明	デフォルト	タイプ
showHeaders (formatting)	メッセージヘッダーを表示します。	false	boolean
showOut (formatting)	エクスチェンジにアウトメッセージがある場合は、アウトメッセージを表示します。	false	boolean
showProperties (formatting)	エクスチェンジのプロパティーを表示します。	false	boolean
showStackTrace (formatting)	交換に例外がある場合、スタックトレースを表示します。showAll、showException、または showCaughtException のいずれかが有効になっている場合にのみ有効です。	false	boolean
showStreams (formatting)	Camel がストリーム本文を表示するかどうか (例: java.io.InputStream など)。このオプションを有効にすると、ストリームがこのロガーによってすでに読み取られているため、後でメッセージ本文にアクセスできなくなる可能性があることに注意してください。これを解決するには、ストリームキャッシングを使用する必要があります。	false	boolean
skipBodyLineSeparator (formatting)	メッセージ本文をログに記録するときに行区切りをスキップするかどうか。これにより、メッセージ本文を 1 行でログに記録できます。このオプションを false に設定すると、本文の行区切りが保持され、本文がそのままログに記録されます。	true	boolean
style (formatting)	使用する出力スタイルを設定します。	デフォルト	OutputStyle

203.3. 通常のロガーのサンプル

以下のルートでは、注文が処理される前に、受信した注文を **DEBUG** レベルでログに記録します。

```
from("activemq:orders").to("log:com.mycompany.order?level=DEBUG").to("bean:processOrder");
```

または、Spring XML を使用してルートを定義します。

```
<route>
  <from uri="activemq:orders"/>
  <to uri="log:com.mycompany.order?level=DEBUG"/>
  <to uri="bean:processOrder"/>
</route>
```

203.4. フォーマッターサンプル付きの通常のロガー

以下のルートでは、注文が処理される前に、受信した注文を **INFO** レベルでログに記録します。

```
from("activemq:orders").
    to("log:com.mycompany.order?showAll=true&multiline=true").to("bean:processOrder");
```

203.5. GROUPSIZE サンプルを使用したスループットロガー

以下のルートでは、10 個のメッセージでグループ化された **DEBUG** レベルでの受注のスループットをログに記録します。

```
from("activemq:orders").
    to("log:com.mycompany.order?level=DEBUG&groupSize=10").to("bean:processOrder");
```

203.6. GROUPINTERVAL サンプルを使用したスループットロガー

このルートにより、10 秒ごとにログに記録されるメッセージの統計が生成されます。最初は 60 秒の遅延があり、メッセージトラフィックがない場合でも統計が表示されます。

```
from("activemq:orders").
    to("log:com.mycompany.order?
level=DEBUG&groupInterval=10000&groupDelay=60000&groupActiveOnly=false").to("bean:process
Order");
```

以下がログに記録されます。

```
"Received: 1000 new messages, with total 2000 so far. Last group took: 10000 millis which is: 100
messages per second. average: 100"
```

203.7. パスワードなどの機密情報のマスク

Camel 2.19 以降で利用可能

logMask フラグを **true** に設定することで、ログのセキュリティーマスキングを有効にできます。このオプションはログ EIP にも影響することに注意してください。

CamelContext レベルで Java DSL でマスクを有効にするには、以下を行います。

```
camelContext.setLogMask(true);
```

XML では次のようになります。

```
<camelContext logMask="true">
```

エンドポイントレベルでオン/オフにすることもできます。エンドポイントレベルで Java DSL のマスクを有効にするには、ログエンドポイントの URI に logMask=true オプションを追加します。

```
from("direct:start").to("log:foo?logMask=true");
```

XML では次のようになります。

```
<route>
```

```
  <from uri="direct:foo"/>
  <to uri="log:foo?logMask=true"/>
</route>
```

デフォルトでは、**org.apache.camel.processor.DefaultMaskingFormatter** がマスキングに使用されます。カスタムマスキングフォーマッタを使用する場合は、**CamelCustomLogMask** という名前でレジストリーに配置します。マスキングフォーマッタは **org.apache.camel.spi.MaskingFormatter** を実装する必要があることに注意してください。

203.8. ログ出力の完全なカスタマイズ

Camel 2.11 から利用可能

#Formatting セクションで説明されているオプションを使用すると、ロガーの出力の大部分を制御できます。ただし、ログ行は常に次の構造に従います。

```
Exchange[Id:ID-machine-local-50656-1234567901234-1-2, ExchangePattern:InOut,
Properties:{CamelToEndpoint=log://org.apache.camel.component.log.TEST?showAll=true,
CamelCreatedTimestamp=Thu Mar 28 00:00:00 WET 2013},
Headers:{breadcrumbId=ID-machine-local-50656-1234567901234-1-1}, BodyType:String, Body:Hello
World, Out: null]
```

この形式は、場合によっては不適切です。おそらく、必要があるためです...

- ... 出力されるヘッダーとプロパティーをフィルタリングして、インサイトと冗長性のバランスをとります。
- ... ログメッセージを最も読みやすいと思われるものに調整します。
- ... 例えばSplunk などのログマイニングシステムによる消化用にログメッセージを調整します。
- ... 特定のボディタイプを異なる方法で印刷します。
- ...など

完全なカスタマイズが必要な場合はいつでも、**ExchangeFormatter** インターフェイスを実装するクラスを作成できます。**format(Exchange)** メソッド内では、完全な Exchange にアクセスできるため、必要な正確な情報を選択して抽出し、カスタムの方法でフォーマットして返すことができます。戻り値が最終的なログメッセージになります。

次の 2 つの方法のいずれかで、Log コンポーネントにカスタム **ExchangeFormatter** を取得させることができます。

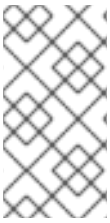
レジストリーで LogComponent を明示的にインスタンス化します。

```
<bean name="log" class="org.apache.camel.component.log.LogComponent">
  <property name="exchangeFormatter" ref="myCustomFormatter" />
</bean>
```

203.8.1. 設定より規約:*

logFormatter という名前の Bean を登録するだけです。ログコンポーネントは、それを自動的に取得するのに十分なほどインテリジェントです。

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" />
```



注記

ExchangeFormatter は 、**その Camel Context 内のすべての Log エンドポイント**に適用されます。エンドポイントごとに異なる ExchangeFormatters が必要な場合は、必要な回数だけ LogComponent をインスタンス化し、関連する Bean 名をエンドポイント 接頭辞として使用します。

Camel 2.11.2/2.12 以降、カスタムログフォーマッタを使用する場合、カスタムログフォーマッタで設定されるログ URI にパラメーターを指定できます。ただし、その場合、logFormatter をスコープ付きのプロトタイプとして定義する必要があるため、異なるパラメーターがある場合は共有されません。たとえば、

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" scope="prototype"/>
```

そして、さまざまなオプションでログ URI を使用して Camel ルートを作成できます。

```
<to uri="log:foo?param1=foo&amp;param2=100"/>

<to uri="log:bar?param1=bar&amp;param2=200"/>
```

203.9. OSGI で LOG コンポーネントを使用する

Camel 2.12.4/2.13.1 での改善

OSGi 内 (Karaf など) で Log コンポーネントを使用する場合、基礎となるロギングメカニズムは PAX ロギングによって提供されます。**org.slf4j.LoggerFactory.getLogger()** メソッドを呼び出すバンドルを検索し、バンドルをロガーインスタンスに関連付けます。カスタム **org.sfl4j.Logger** インスタンスを指定しないと、Log コンポーネントによって作成されたロガーが **camel-core** バンドルに関連付けられます。

一部のシナリオでは、ロガーに関連付けられたバンドルがルート定義を含むバンドルである必要があります。これを行うには、**org.slf4j.Logger** の単一インスタンスをレジストリーに登録するか、**logger** URI パラメーターを使用して参照します。

203.10. 関連項目

- 人間のログの DSL で **log** を直接使用するための LogEIP。

第204章 LUCENE コンポーネント

Camel バージョン 2.2 以降で利用可能

lucene コンポーネントは、Apache Lucene プロジェクトに基づいています。Apache Lucene は、完全に Java で記述された、強力で高性能なフル機能のテキスト検索エンジンライブラリーです。Lucene の詳細については、次のリンクを参照してください。

- http://lucene.apache.org/java/docs/
- http://lucene.apache.org/java/docs/features.html

camel の lucene コンポーネントは、エンタープライズ統合パターンとシナリオでの Lucene エンドポイントの統合と利用を容易にします。lucene コンポーネントは次のことを行います。

- ペイロードが Lucene エンドポイントに送信されると、ドキュメントの検索可能なインデックスを構築します
- Camel でのインデックス付き検索の実行を容易にします

このコンポーネントはプロデューサーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lucene</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

204.1. URI 形式

```
lucene:searcherName:insert[?options]
lucene:searcherName:query[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

204.2. 挿入オプション

Lucene コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
config (advanced)	共有 lucene 設定を使用するには		LuceneConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Lucene エンドポイントは、URI 構文を使用して設定されます。

```
lucene:host:operation
```

パスおよびクエリーパラメーターを使用します。

204.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** lucene サーバーへの URL		String
operation	挿入やクエリーなどの **必須** 操作。		LuceneOperation

204.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
analyzer (producer)	Analyzer は、テキストを分析する TokenStreams をビルドします。したがって、テキストから索引用語を展開するためのポリシーを表します。アナライザーの値は、抽象クラス org.apache.lucene.analysis.Analyzer を拡張する任意のクラスにすることができます。Lucene は、すぐに使用できる豊富なアナライザーのセットも提供します		アナライザー
indexDir (producer)	指定されたアナライザーによるドキュメントの分析時にインデックスファイルが作成されるファイルシステムディレクトリー		File
maxHits (producer)	検索操作の結果セットを制限する整数値		int
srcDir (producer)	プロデューサの起動時に分析され、インデックスに追加されるために使用されるファイルを含むオプションのディレクトリー。		File
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

204.3. キャッシュとのメッセージの送受信

204.3.1. メッセージヘッダー

ヘッダー	説明
QUERY	インデックスで実行する Lucene クエリー。クエリーには、ワイルドカードとフレーズを含めることができます
RETURN_LUCENE_DOCS	**Camel 2.15:** ヒット情報を返すときに実際の Lucene ドキュメントを含めるには、このヘッダーを true に設定します。

204.3.2. Lucene プロデューサー

このコンポーネントは、2 つのプロデューサーエンドポイントをサポートします。

insert - 挿入プロデューサーは、入力エクスチェンジのボディーを分析し、それをトークン (コンテンツ) に関連付けることによって、検索可能なインデックスをビルドします。**query** - クエリープロデューサーは、事前に作成されたインデックスで検索を実行します。クエリーは、検索可能なインデックスを使用して、スコアと関連性に基づく検索を実行します。クエリーは、QUERY と呼ばれるヘッダープロパティー名を含む入力エクスチェンジを介して送信されます。ヘッダープロパティー QUERY の値は Lucene Query です。Lucene クエリーの作成方法の詳細については、http://lucene.apache.org/java/3_0_0/queryparsersyntax.html を参照してください。

204.3.3. Lucene プロセッサー

プロデューサーを作成しなくても、lucene に対してクエリーを実行できる LuceneQueryProcessor というプロセッサーがあります。

204.4. LUCENE の使用例

204.4.1. 例 1: Lucene インデックスの作成

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("direct:start").
            to("lucene:whitespaceQuotesIndex:insert?
                analyzer=#whitespaceAnalyzer&indexDir=#whitespace&srcDir=#load_dir").
            to("mock:result");
    }
};
```

204.4.2. 例 2: Camel コンテキストでプロパティーを JNDI レジストリーにロードする

```
@Override
protected JndiRegistry createRegistry() throws Exception {
  JndiRegistry registry =
        new JndiRegistry(createJndiContext());
  registry.bind("whitespace", new File("./whitespaceIndexDir"));
  registry.bind("load_dir",
        new File("src/test/resources/sources"));
```

```
  registry.bind("whitespaceAnalyzer",
       new WhitespaceAnalyzer());
  return registry;
}
...
CamelContext context = new DefaultCamelContext(createRegistry());
```

204.4.3. 例 2: クエリープロデューサーを使用して検索を実行する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
      from("direct:start").
        setHeader("QUERY", constant("Seinfeld")).
        to("lucene:searchIndex:query?
          analyzer=#whitespaceAnalyzer&indexDir=#whitespace&maxHits=20").
        to("direct:next");

      from("direct:next").process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Hits hits = exchange.getIn().getBody(Hits.class);
          printResults(hits);
        }

        private void printResults(Hits hits) {
           LOG.debug("Number of hits: " + hits.getNumberOfHits());
           for (int i = 0; i < hits.getNumberOfHits(); i++) {
             LOG.debug("Hit " + i + " Index Location:" + hits.getHit().get(i).getHitLocation());
             LOG.debug("Hit " + i + " Score:" + hits.getHit().get(i).getScore());
             LOG.debug("Hit " + i + " Data:" + hits.getHit().get(i).getData());
           }
         }
      }).to("mock:searchResult");
    }
};
```

204.4.4. 例 3: クエリープロセッサーを使用して検索を実行する

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
       try {
          from("direct:start").
             setHeader("QUERY", constant("Rodney Dangerfield")).
             process(new LuceneQueryProcessor("target/stdindexDir", analyzer, null, 20)).
             to("direct:next");
       } catch (Exception e) {
          e.printStackTrace();
       }

       from("direct:next").process(new Processor() {
          public void process(Exchange exchange) throws Exception {
             Hits hits = exchange.getIn().getBody(Hits.class);
             printResults(hits);
          }
```

```
            private void printResults(Hits hits) {
                LOG.debug("Number of hits: " + hits.getNumberOfHits());
                for (int i = 0; i < hits.getNumberOfHits(); i++) {
                    LOG.debug("Hit " + i + " Index Location:" + hits.getHit().get(i).getHitLocation());
                    LOG.debug("Hit " + i + " Score:" + hits.getHit().get(i).getScore());
                    LOG.debug("Hit " + i + " Data:" + hits.getHit().get(i).getData());
                }
            }
        }).to("mock:searchResult");
    }
};
```

第205章 LUMBERJACK コンポーネント

Camel バージョン 2.18 以降で利用可能

Lumberjack コンポーネントは、Lumberjack プロトコルを使用してネットワーク経由で送信されたログを、たとえば Filebeat から取得します。ネットワーク通信は SSL で保護できます。

このコンポーネントは、コンシューマーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lumberjack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

205.1. URI 形式

```
lumberjack:host
lumberjack:host:port
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

205.2. オプション

Lumberjack コンポーネントは、以下に示す 3 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
sslContextParameters (security)	すべてのエンドポイントに使用するデフォルトの SSL 設定を設定します。エンドポイントレベルで直接設定することもできます。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Lumberjack エンドポイントは、URI 構文を使用して設定されます。

```
lumberjack:host:port
```

パスおよびクエリーパラメーターを使用します。

205.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** Lumberjack をリッスンするためのネットワークインターフェイス		String
port	Lumberjack をリッスンするネットワークポート	5044	int

205.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sslContextParameters (consumer)	SSL 設定		SSLContextParameters
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

205.3. 結果

結果のボディーは **Map<String, Object>** オブジェクトです。

205.4. LUMBERJACK の使用例

205.4.1. 例 1: ログメッセージのストリーミング

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("lumberjack:0.0.0.0").                 // Listen on all network interfaces using the default port
            setBody(simple("${body[message]}")).    // Select only the log message
            to("stream:out");                       // Write it into the output stream
    }
};
```

第206章 LZF DEFLATE COMPRESSION DATAFORMAT

Camel バージョン 2.17 以降で利用可能

LZF データ形式 は、メッセージの圧縮および解凍形式です。LZF deflate アルゴリズムを使用します。 LZF 圧縮を使用してマーシャリングされたメッセージは、エンドポイントで消費される直前に、LZF 解凍を使用してアンマーシャリングできます。圧縮機能は、大きな XML およびテキストベースのペイロードを処理する場合、または LZF アルゴリズムを使用して以前に圧縮されたメッセージを読み取る場合に非常に役立ちます。

206.1. オプション

LZF Deflate 圧縮データ形式は、以下に示す 2 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
usingParallelCompression	**false**	**Boolean**	複数の処理コアを使用してエンコード (圧縮) を有効にします。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

206.2. MARSHAL

この例では、通常のテキスト/XML ペイロードを LZF 圧縮形式を使用する圧縮ペイロードにマーシャリングし、MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().lzf().to("activemq:queue:MY_QUEUE");
```

206.3. UNMARSHAL

この例では、MY_QUEUE という ActiveMQ キューから LZF ファイルペイロードを元の形式にアンマーシャリングして **UnGZippedMessageProcessor** に転送し、処理します。

```
from("activemq:queue:MY_QUEUE").unmarshal().lzf().process(new
UnCompressedMessageProcessor());
```

206.4. 依存関係

camel ルートで LZF 圧縮を使用するには、このデータ形式を実装する **camel-lzf** に依存関係を追加する必要があります。

Maven を使用する場合は、**pom.xml** に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページ を参照してください)。

```
<dependency>
```

```
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lzf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第207章 MAIL コンポーネント

Camel バージョン 1.0 以降で利用可能

メールコンポーネントは、Spring の Mail サポートおよび基盤となる JavaMail システムを介して電子メールへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

警告

Geronimo mail .jar

添付付きのメールをポーリングする際に、geronimo mail **.jar** (v1.6) にバグがあることを発見しました。**Content-Type** を正しく識別できません。そのため、メールに **.jpeg** ファイルを添付してポーリングすると、**Content-Type** は **image/jpeg** ではなく **text/plain** として解決されます。そのため、**org.apache.camel.component.ContentTypeResolver** SPI インターフェイスを追加しました。これにより、独自の実装を提供し、ファイル名に基づいて正しい Mime タイプを返すことでこのバグを修正できます。したがって、ファイル名が **jpeg/jpg** で終わる場合、**image/jpeg** を返すことができます。

MailComponent インスタンスまたは **MailEndpoint** インスタンスでカスタムリゾルバーを設定できます。

ヒント

POP3 または IMAP POP3 にはいくつかの制限があるため、エンドユーザーは可能であれば IMAP を使用することをお勧めします。

情報: **テスト用の模擬メールの使用** 単体テストにモックフレームワークを使用できます。これにより、実際のメールサーバーを必要とせずにテストできます。ただし、実際のメールサーバーにメールを送信する必要がある本番環境やその他の環境に移行する場合は、モックメールを含めないことを忘れないでください。クラスパスに mock-javamail.jar が存在するということは、それが作動してメールの送信を回避することを意味します。

207.1. URI 形式

メールエンドポイントには、次の URI 形式のいずれかを使用できます (プロトコルはそれぞれ、SMTP、POP3、または IMAP)。

```
smtp://[username@]host[:port][?options]
pop3://[username@]host[:port][?options]
imap://[username@]host[:port][?options]
```

メールコンポーネントは、これらのプロトコルの安全なバリアント (SSL を介したレイヤー) もサポートします。スキームに **s** を追加することで、安全なプロトコルを有効にできます。

```
smtps://[username@]host[:port][?options]
pop3s://[username@]host[:port][?options]
imaps://[username@]host[:port][?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

207.2.

Mail コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	メール設定を設定します		MailConfiguration
contentTypeResolver (advanced)	添付の Content-Type を決定するリゾルバー。		ContentTypeResolver
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

207.3.

Mail エンドポイントは、URI 構文を使用して設定されます。

```
imap:host:port
```

パスおよびクエリーパラメーターを使用します。

207.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** メールサーバーのホスト名		String

名前	説明	デフォルト	タイプ
port	メールサーバーのポート番号		int

207.3.2. クエリーパラメーター(62 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
closeFolder (consumer)	ポーリング後にコンシューマーがフォルダーを閉じる必要があるかどうか。このオプションを false に設定し、disconnect=false も設定すると、コンシューマーはポーリング間でフォルダーを開いたままにします。	true	boolean
copyTo (consumer)	メールメッセージを処理した後、指定された名前のメールフォルダーにコピーできます。キー copyTo を含むヘッダーを使用して、この設定値をオーバーライドできます。これにより、実行時に設定されたフォルダー名にメッセージをコピーできます。		String
delete (consumer)	処理後にメッセージを削除します。これは、メールメッセージに DELETED フラグを設定することによって行われます。false の場合、代わりに SEEN フラグが設定されます。Camel 2.10 では、メールを削除するかどうかを決定するキー delete でヘッダーを設定することにより、この設定オプションをオーバーライドできます。	false	boolean
disconnect (consumer)	ポーリング後にコンシューマーを切断するかどうか。有効にすると、各ポーリングで Camel が強制的に接続されます。	false	boolean

名前	説明	デフォルト	タイプ
handleFailedMessage (consumer)	メールコンシューマーが特定のメールメッセージを取得できない場合、このオプションを使用すると、コンシューマーのエラーハンドラーによって発生した例外を処理できます。コンシューマーでブリッジエラーハンドラーを有効にすると、代わりに Camel ルーティングエラーハンドラーが例外を処理できます。デフォルトの動作では、コンシューマーが例外を出力し、バッチからのメールは Camel によってルーティングできません。	false	boolean
maxMessagesPerPoll (consumer)	ポーリングごとに収集するメッセージの最大数を指定します。デフォルトでは最大値は設定されていません。サーバーの起動時に数千のファイルをダウンロードしないように、たとえば 1000 の制限を設定するために使用できます。このオプションを無効にするには、0 または負の値を設定します。		int
mimeDecodeHeaders (consumer)	このオプションは、メールヘッダーの透過的な MIME デコードとデプロイメントを有効にします。	false	boolean
peek (consumer)	メールメッセージを処理する前に、javax.mail.Message をピークとしてマークします。これは、IMAPMessage メッセージタイプにのみ適用されます。peek を使用すると、メールはメールサーバー上で SEEN としてマークされません。これにより、Camel でエラー処理が発生した場合にメールメッセージをロールバックできます。	true	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
skipFailedMessage (consumer)	メールコンシューマーが特定のメールメッセージを取得できない場合、このオプションを使用すると、メッセージをスキップして次のメールメッセージの取得に進むことができます。デフォルトの動作では、コンシューマーが例外を出力し、バッチからのメールは Camel によってルーティングできません。	false	boolean
unseen (consumer)	未読メールのみで制限するかどうか。	true	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
fetchSize (consumer)	ポーリング中に消費するメッセージの最大数を設定します。これは、メールボックスフォルダーに大量のメッセージが含まれている場合に、メールサーバーの過負荷を回避するために使用できます。デフォルト値の -1 は、フェッチサイズがなく、すべてのメッセージが消費されることを意味します。値を 0 に設定するのは、Camel がメッセージをまったく消費しない特殊なケースです。	-1	int
folderName (consumer)	ポーリングするフォルダー。	INBOX	String
mailUidGenerator (consumer)	カスタムロジックを使用してメールメッセージの UUID を生成できるプラグ可能な MailUidGenerator。		MailUidGenerator
mapMailMessage (consumer)	Camel が受信したメールメッセージを Camel の本文/ヘッダーにマップするかどうかを指定します。true に設定すると、メールメッセージのボディーは Camel IN メッセージのボディーにマップされ、メールヘッダーは IN ヘッダーにマップされます。このオプションが false に設定されている場合、IN メッセージには生の javax.mail.Message が含まれます。exchange.getIn ().getBody (javax.mail.Message.class) を呼び出して、この生のメッセージを取得できます。	true	boolean
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
postProcessAction (consumer)	通常の処理が終了したら、メールボックスで後処理タスクを実行するための MailBoxPostProcessAction を参照します。		MailBoxPostProcess アクション
bcc (producer)	BCC メールアドレスを設定します。複数の電子メールアドレスはコンマで区切ります。		String

名前	説明	デフォルト	タイプ
cc (producer)	CC メールアドレスを設定します。複数の電子メールアドレスはコンマで区切ります。		String
from (producer)	差出人の電子メールアドレス	camel@localhost	String
replyTo (producer)	Reply-To 受信者 (応答メールの受信者)。複数のメールアドレスはコンマで区切ります。		String
subject (producer)	送信されるメッセージの件名。注: ヘッダーに件名を設定すると、このオプションよりも優先されます。		String
to (producer)	宛先メールアドレスを設定します。複数の電子メールアドレスはコンマで区切ります。		String
javaMailSender (producer)	メールの送信にカスタム org.apache.camel.component.mail.JavaMailSender を使用するには。		JavaMailSender
additionalJavaMail Properties (advanced)	他のすべてのオプションに基づいて設定されたデフォルトプロパティーを追加/オーバーライドする追加の Java メールプロパティーを設定します。これは、いくつかの特別なオプションを追加する必要があるが、他のオプションはそのままにしておきたい場合に便利です。		プロパティー
alternativeBodyHeader (advanced)	代替電子メール本文を含む IN メッセージヘッダーのキーを指定します。たとえば、メールを text/html 形式で送信し、HTML 以外のメールクライアントに代替メール本文を提供する場合は、このキーをヘッダーとして使用して代替メール本文を設定します。	CamelMailAlternativeBody	String
attachmentsContentTransferEncodingResolver (advanced)	カスタムの AttachmentsContentTransferEncodingResolver を使用して、添付に使用する content-type-encoding を解決するには。		AttachmentsContentTransferEncodingResolver
binding (advanced)	Camel メッセージと Mail メッセージの間の変換に使用されるバインディングを設定します		MailBinding
connectionTimeout (advanced)	ミリ秒単位の接続タイムアウト。	30000	int
contentType (advanced)	メールメッセージのコンテンツタイプ。HTML メールには text/html を使用します。	text/plain	String

名前	説明	デフォルト	タイプ
contentTypeResolver (advanced)	添付の Content-Type を決定するリゾルバー。		ContentTypeResolver
debugMode (advanced)	基礎となるメールフレームワークでデバッグモードを有効にします。SUN メールフレームワークは、デフォルトでデバッグメッセージを System.out に記録します。	false	boolean
headerFilterStrategy (advanced)	カスタム org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルタリングするには。		HeaderFilterStrategy
ignoreUnsupportedCharset (advanced)	メール送信時にローカル JVM でサポートされていない文字セットを Camel が無視できるようにするオプション。文字セットがサポートされていない場合は、charset=XXX (XXX はサポートされていない文字セットを表す) が content-type から削除され、代わりにプラットフォームのデフォルトに依存します。	false	boolean
ignoreUriScheme (advanced)	メール送信時にローカル JVM でサポートされていない文字セットを Camel が無視できるようにするオプション。文字セットがサポートされていない場合は、charset=XXX (XXX はサポートされていない文字セットを表す) が content-type から削除され、代わりにプラットフォームのデフォルトに依存します。	false	boolean
session (advanced)	camel がすべてのメールインタラクションに使用するメールセッションを指定します。メールセッションが JavaEE コンテナーなどの他のリソースによって作成および管理されるシナリオで役立ちます。これが指定されていない場合、Camel は自動的にメールセッションを作成します。		Session
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
useInlineAttachments (advanced)	ディスポジションインラインまたは添付を使用するかどうか。	false	boolean
idempotentRepository (filter)	プラグイン可能なリポジトリー org.apache.camel.spi.IdempotentRepository により、同じメールボックスからのクラスター消費が可能になり、コンシューマーが処理するメールメッセージが有効かどうかをリポジトリーで調整できます。デフォルトでは、リポジトリーは使用されていません。		String>

名前	説明	デフォルト	タイプ
idempotentRepositoryRemoveOnCommit (filter)	べき等リポジトリーを使用している場合、メールメッセージが正常に処理されてコミットされると、メッセージ ID がべき等リポジトリーから削除されるか (デフォルト)、リポジトリーに保持されます。デフォルトでは、メッセージ ID は一意であり、リポジトリーに保持する値がないと想定されます。これは、メールメッセージが閲覧済み、移動済み、または削除済みとしてマークされ、再度消費されるのを防ぐためです。したがって、メッセージ ID を冪等リポジトリーに格納してもほとんど価値がありません。ただし、このオプションを使用すると、何らかの理由でメッセージ ID を保存できます。	true	boolean
searchTerm (filter)	件名、本文、送信元、特定の日付以降に送信されたものなどの検索条件に基づいてメールをフィルタリングできる javax.mail.search.SearchTerm を参照します。		SearchTerm
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	60000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

名前	説明	デフォルト	タイプ
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
sortTerm (並べ替え)	メッセージのソート順。IMAP でのみネイティブにサポートされています。POP3 を使用する場合、または IMAP サーバーに SORT 機能がない場合に、ある程度エミュレートされます。		String
dummyTrustManager (security)	すべての証明書を信頼するためのダミーのセキュリティー設定を使用する場合。実稼働環境ではなく、開発モードでのみ使用してください。	false	boolean
password (security)	ログイン用のパスワード		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
username (security)	ログイン用のユーザー名		文字列

207.3.3. サンプルエンドポイント

通常、次のようにログイン認証情報を含む URI を指定します (例として SMTP を取り上げます)。

```
smtp://[username@]host[:port][?password=somepwd]
```

または、ユーザー名とパスワードの両方をクエリーオプションとして指定することもできます。

```
smtp://host[:port]?password=somepwd&username=someuser
```

以下に例を示します。

```
smtp://mycompany.mailserver:30?password=tiger&username=scott
```

207.4. コンポーネント

- IMAP
- IMAPs
- POP3s
- POP3s
- SMTP
- SMTPs

207.4.1. デフォルトのポート

デフォルトのポート番号がサポートされています。ポート番号が省略された場合、Camel はプロトコルに基づいて使用するポート番号を決定します。

Protocol	デフォルトのポート番号
SMTP	**25**
SMTPS	**465**
POP3	**110**
POP3S	**995**
IMAP	**143**
IMAPS	**993**

207.5. SSL サポート

基礎となるメールフレームワークは、SSL サポートの提供を担当します。 必要な Java Mail API 設定オプションを完全に指定して SSL/TLS サポートを設定するか、コンポーネントまたはエンドポイント設定を通じて設定済みの SSLContextParameters を提供することができます。

207.5.1. JSSE 設定ユーティリティーの使用

Camel 2.10 の時点で、メールコンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、メールコンポーネントでユーティリティーを使用する方法を示しています。

エンドポイントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/truststore.jks");
ksp.setPassword("keystorePassword");
TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);
SSLContextParameters scp = new SSLContextParameters();
scp.setTrustManagers(tmp);
Registry registry = ...
registry.bind("sslContextParameters", scp);
...
from(...)
&amp;nbsp; &amp;nbsp; .to("smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters");
```

エンドポイントの Spring DSL ベースの設定

```
...
<camel:sslContextParameters id="sslContextParameters">
  <camel:trustManagers>
    <camel:keyStore resource="/users/home/server/truststore.jks" password="keystorePassword"/>
  </camel:trustManagers>
</camel:sslContextParameters>...
...
<to uri="smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters"/
>...
```

207.5.2. JavaMail を直接設定する

Camel は SUN JavaMail を使用します。これは、よく知られている認証局 (デフォルトの JVM 信頼設定) によって発行された証明書のみを信頼します。独自の証明書を発行する場合は、CA 証明書を JVM の Java 信頼/キーストアファイルにインポートし、デフォルトの JVM 信頼/キーストアファイルを上書きする必要があります (詳細については、JavaMail の **SSLNOTES.txt** を参照してください)。

207.6. メールメッセージの内容

Camel は、メッセージ交換の IN 本文を MimeMessage テキストコンテンツとして使用します。ボディは **String.class** に変換されます。

Camel は、交換のすべての IN ヘッダーを MimeMessage ヘッダーにコピーします。

MimeMessage の件名は、IN メッセージのヘッダープロパティーを使用して設定できます。以下のコードはこれを示しています。

同じことが受信者などの他の MimeMessage ヘッダーにも当てはまるため、ヘッダープロパティーを **To** として使用できます。

Camel 2.11 以降、MailProducer を使用してメールをサーバーに送信する場合、Camel メッセージヘッダーからキー **CamelMailMessageId** を使用して MimeMessage のメッセージ ID を取得できるはずです。

207.7. ヘッダーは事前設定された受信者よりも優先されます

メッセージヘッダーで指定された受信者は、エンドポイント URI で事前設定された受信者より常に優先されます。アイデアは、メッセージヘッダーに受信者を指定すると、それが得られるというものです。エンドポイント URI で事前設定された受信者は、フォールバックとして扱われます。

以下のサンプルコードでは、電子メールメッセージは **davsclaus@apache.org** に送信されます。これは、事前設定された受信者 **info@mycompany.com** よりも優先されるためです。エンドポイント URI の **CC** および **BCC** 設定も無視され、それらの受信者はメールを受信しません。ヘッダーと事前設定された設定の選択は、すべてかゼロ **か** です。メールコンポーネントは、受信者をヘッダーから排他的に取得するか、事前設定済みの設定から排他的に取得します。ヘッダーと事前設定済みの設定を混在させることはできません。

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put("to", "davsclaus@apache.org");

        template.sendBodyAndHeaders("smtp://admin@localhost?to=info@mycompany.com", "Hello
World", headers);
```

207.8. 設定を容易にする複数の受信者

コンマ区切りまたはセミコロン区切りのリストを使用して、複数の受信者を設定できます。これは、ヘッダー設定とエンドポイント URI の設定の両方に適用されます。以下に例を示します。

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put("to", "davsclaus@apache.org ; jstrachan@apache.org ; ningjiang@apache.org");
```

前の例では、セミコロン **;** を使用しています。、区切り文字として。

207.9. 送信者名と電子メールの設定

name <email> の形式で受信者を指定して、受信者の名前と電子メールアドレスの両方を含めることができます。

たとえば、メッセージで次のヘッダーを定義します。

```
Map headers = new HashMap();
map.put("To", "Claus Ibsen <davsclaus@apache.org>");
map.put("From", "James Strachan <jstrachan@apache.org>");
map.put("Subject", "Camel is cool");
```

207.10. JAVAMAIL API (EX SUN JAVAMAIL)

JavaMail API は、メールの消費と生成のために内部で使用されます。
エンドユーザーが POP3 または IMAP プロトコルを使用する場合は、これらのリファレンスを参照する

ことをお勧めします。特に、POP3 の機能セットは IMAP よりもはるかに限定されていることに注意してください。

- JavaMail POP3 API
- JavaMail IMAP API
- そして、一般的に MAIL フラグ について

207.11. サンプル

JMS キューから受信したメッセージを電子メールとして送信する単純なルートから始めます。電子メールアカウントは **mymailserver.com** の **admin** アカウントです。

```
from("jms://queue:subscription").to("smtp://admin@mymailserver.com?password=secret");
```

次のサンプルでは、毎分 1 回メールボックスをポーリングして新しい電子メールを探します。ポーリング間隔、**consumer.delay** を 60000 ミリ秒 = 60 秒として設定するために、特別な **consumer** オプションを使用していることに注意してください。

```
from("imap://admin@mymailserver.com
    password=secret&unseen=true&consumer.delay=60000")
   .to("seda://mails");
```

このサンプルでは、複数の受信者にメールを送信します。

207.12. 添付付きメール送信サンプル

警告

アタッチメントはすべての Camel コンポーネントでサポートされているわけではありません Attachments API は Java Activation Framework に基づいており、通常はメール API でのみ使用されます。他の Camel コンポーネントの多くはアタッチメントをサポートしていないため、アタッチメントがルートに沿って伝播するにつれて、アタッチメントが失われる可能性があります。したがって、経験則として、メールエンドポイントにメッセージを送信する直前に添付を追加します。

メールコンポーネントは添付をサポートしています。以下のサンプルでは、ロゴファイルが添付されたプレーンテキストメッセージを含むメールメッセージを送信します。

207.13. SSL サンプル

このサンプルでは、Google メールの受信トレイでメールをポーリングします。メールをローカルメールクライアントにダウンロードするには、Google メールで SSL を有効にして設定する必要があります。これを行うには、Google メールアカウントにログインし、設定を変更して IMAP アクセスを許可します。Google には、これを行う方法に関する広範なドキュメントがあります。

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&consumer.delay=60000").to("log:newmail");
```

上記のルートは、毎分1回、Google メールの受信トレイに新しいメールがないかどうかをポーリングし、受信したメッセージを **newmail** ロガーカテゴリーに記録します。
DEBUG ログを有効にしてサンプルを実行すると、ログで進行状況を監視できます。

```
2008-05-08 06:32:09,640 DEBUG MailConsumer - Connecting to MailStore
imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX
2008-05-08 06:32:11,203 DEBUG MailConsumer - Polling mailfolder: imaps//imap.gmail.com:993
(SSL enabled), folder=INBOX
2008-05-08 06:32:11,640 DEBUG MailConsumer - Fetching 1 messages. Total 1 messages.
2008-05-08 06:32:12,171 DEBUG MailConsumer - Processing message: messageNumber=[332],
from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
2008-05-08 06:32:12,187 INFO  newmail - Exchange[MailMessage: messageNumber=[332], from=
[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
```

207.14. 添付ファイル付きメールの利用例

このサンプルでは、メールボックスをポーリングし、メールのすべての添付ファイルをファイルとして保存します。まず、メールボックスをポーリングするルートを定義します。このサンプルは Google メールに基づいているため、SSL サンプルに示されているのと同じルートを使用します。

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&consumer.delay=60000").process(new MyMailProcessor());
```

メールをログに記録する代わりに、Java コードからメールを処理できるプロセッサーを使用します。

```
public void process(Exchange exchange) throws Exception {
    // the API is a bit clunky so we need to loop
    Map<String, DataHandler> attachments = exchange.getIn().getAttachments();
    if (attachments.size() > 0) {
        for (String name : attachments.keySet()) {
            DataHandler dh = attachments.get(name);
            // get the file name
            String filename = dh.getName();

            // get the content and convert it to byte[]
            byte[] data = exchange.getContext().getTypeConverter()
                            .convertTo(byte[].class, dh.getInputStream());

            // write the data to a file
            FileOutputStream out = new FileOutputStream(filename);
            out.write(data);
            out.flush();
            out.close();
        }
    }
}
```

ご覧のとおり、添付を処理するための API は少し不格好ですが、**javax.activation.DataHandler** を取得できるので、標準 API を使用して添付を処理できます。

207.15. 添付ファイル付きのメールメッセージを分割する方法

この例では、多数の添付ファイルを含む可能性のあるメールメッセージを使用します。やりたいことは、個々の添付ファイルごとにスプリッタ EIP を使用して、添付ファイルを個別に処理することです。たとえば、メールメッセージに 5 つの添付ファイルがある場合、Splitter は、それぞれに 1 つの添付ファイルを持つ 5 つのメッセージを処理する必要があります。これを行うには、単一の添付ファイルを持つ 5 つのメッセージを含む List<Message> を提供するスプリッターにカスタム式を提供する必要があります。

このコードは、**camel-mail** コンポーネントの Camel 2.10 以降でそのまま提供されます。コードはクラス: **org.apache.camel.component.mail.SplitAttachmentsExpression** にあり、ソースコードは こちら にあります。

Camel ルートでは、次に示すようにルートでこの式を使用する必要があります。

XML DSL を使用する場合は、以下に示すように、Splitter でメソッド呼び出し式を宣言する必要があります。

```
<split>
  <method beanType="org.apache.camel.component.mail.SplitAttachmentsExpression"/>
  <to uri="mock:split"/>
</split>
```

Camel 2.16 以降では、添付ファイルをバイトとして分割して、メッセージボディーとして保存することもできます。これは、ブール値 true で式を作成することによって行われます

```
SplitAttachmentsExpression split = SplitAttachmentsExpression(true);
```

次に、式をスプリッター eip で使用します。

207.16. カスタム SEARCHTERM の使用

Camel 2.11 から利用可能

MailEndpoint で **searchTerm** を設定して、不要なメールを除外することができます。

たとえば、件名または本文に Camel が含まれるようにメールをフィルタリングするには、次のようにします。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subjectOrBody=Camel"/>
  <to uri="bean:myBean"/>
</route>
```

"searchTerm.subjectOrBody" をパラメーターキーとして使用して、"Camel" という単語を含むメールの件名または本文を検索することを示していることに注意してください。
クラス **org.apache.camel.component.mail.SimpleSearchTerm** には、設定可能な多くのオプションがあります。

または、目に見えない新しいメールを 24 時間さかのぼって取得することもできます。"now-24h" 構文に注意してください。詳細については、下の表を参照してください。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.fromSentDate=now-24h"/>
  <to uri="bean:myBean"/>
</route>
```

エンドポイント uri 設定に複数の searchTerm を含めることができます。次に、AND 演算子を使用してそれらを結合します。たとえば、両方の条件が一致する必要があります。たとえば、メールの件名に Camel が含まれる 24 時間前にさかのぼる最後の未読メールを取得するには、次のようにします。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subject=Camel&searchTerm.fromSentDate=now-
24h"/>
  <to uri="bean:myBean"/>
</route>
```

SimpleSearchTerm は POJO から簡単に設定できるように設計されているため、XML の <bean> スタイルを使用して設定することもできます。

```
<bean id="mySearchTerm" class="org.apache.camel.component.mail.SimpleSearchTerm">
  <property name="subject" value="Order"/>
  <property name="to" value="acme-order@acme.com"/>
  <property name="fromSentDate" value="now"/>
 </bean>
```

次に示すように、Camel ルートで #beanId を使用して、この Bean を参照できます。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm=#mySearchTerm"/>
  <to uri="bean:myBean"/>
</route>
```

Java には、**org.apache.camel.component.mail.SearchTermBuilder** クラスを使用して複合 **SearchTerms** を構築するビルダークラスがあります。これにより、次のような複雑な用語を作成できます。

```
// we just want the unseen mails which is not spam
SearchTermBuilder builder = new SearchTermBuilder();

builder.unseen().body(Op.not, "Spam").subject(Op.not, "Spam")
  // which was sent from either foo or bar
  .from("foo@somewhere.com").from(Op.or, "bar@somewhere.com");
  // .. and we could continue building the terms

SearchTerm term = builder.build();
```

207.17. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド

第208章 マスターコンポーネント

Camel バージョン 2.20 以降で利用可能

camel-master: エンドポイントは、クラスター内の単一のコンシューマーのみが特定のエンドポイントから消費するようにする方法を提供します。その JVM が停止した場合の自動フェイルオーバー。

これは、同時消費をサポートしていないレガシーバックエンドから消費する必要がある場合、または商業的または安定性の理由により、任意の時点で 1 つの接続しか持てない場合に非常に役立ちます。

208.1. マスターエンドポイントの使用

camel エンドポイントの前に **master:someName:** を付けるだけです。ここで、**someName** は論理名であり、マスターロックを取得するために使用されます。例えば

```
from("master:cheese:jms:foo").to("activemq:wine");
```

上記は、ActiveMQ の [Exclusive Consumers](http://activemq.apache.org/exclusive-consumer.html) タイプの機能をシミュレートしています。しかし、サードパーティーの JMS プロバイダーでは、Exclusive Consumers をサポートしていない場合があります。

208.2. URI 形式

```
master:namespace:endpoint[?options]
```

endpoint は、マスター/スレーブモードで実行する任意の Camel エンドポイントです。

208.3. オプション

マスターコンポーネントは、以下に示す 3 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
service (advanced)	使用するサービスを注入します。		CamelClusterService
serviceSelector (advanced)	使用する CamelClusterService を検索するために使用されるサービスセレクターを挿入します。		セレクター
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

マスターエンドポイントは、URI 構文を使用して設定されます。

```
master:namespace:delegateUri
```

パスおよびクエリーパラメーターを使用します。

208.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
namespace	**必須** 使用するクラスター名前空間の名前		String
delegateUri	**必須** マスター/スレーブモードで使用するエンドポイント uri		String

208.3.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

208.4. 例

クラスター化された Camel アプリケーションを保護して、1 つのアクティブノードからのファイルのみを消費することができます。

```
// the file endpoint we want to consume from
String url = "file:target/inbox?delete=true";

// use the camel master component in the clustered group named myGroup
// to run a master/slave mode in the following Camel url
from("master:myGroup:" + url)
```

```
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

マスターコンポーネントは、次を使用して設定できる CamelClusterService を活用します。

- **Java**

```
ZooKeeperClusterService service = new ZooKeeperClusterService();
service.setId("camel-node-1");
service.setNodes("myzk:2181");
service.setBasePath("/camel/cluster");

context.addService(service)
```

- **Xml (Spring/ブループリント)**

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring
    http://camel.apache.org/schema/spring/camel-spring.xsd">


 <bean id="cluster"
class="org.apache.camel.component.zookeeper.cluster.ZooKeeperClusterService">
   <property name="id" value="camel-node-1"/>
   <property name="basePath" value="/camel/cluster"/>
   <property name="nodes" value="myzk:2181"/>
 </bean>

 <camelContext xmlns="http://camel.apache.org/schema/spring" autoStartup="false">
   ...
 </camelContext>

</beans>
```

- **Spring boot**

```
camel.component.zookeeper.cluster.service.enabled   = true
camel.component.zookeeper.cluster.service.id        = camel-node-1
camel.component.zookeeper.cluster.service.base-path = /camel/cluster
camel.component.zookeeper.cluster.service.nodes     = myzk:2181
```

208.5. 実装

Camel は、次の ClusterService 実装を提供します。

- camel-atomix
- camel-consul

- camel-file
- camel-kubernetes
- camel-zookeeper

208.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第209章 METRICS コンポーネント

209.1. METRICS コンポーネント

metrics: コンポーネントを使用すると、さまざまなメトリックを Camel ルートから直接収集できます。サポートされているメトリックタイプは、counter、histogram、meter、timer、および Gauge です。 メトリクス は、アプリケーションの動作を測定する簡単な方法を提供します。設定可能なレポートバックエンドにより、統計情報の収集と視覚化のためのさまざまな統合オプションが有効になります。このコンポーネントは、Dropwizard Metrics を使用してルート統計情報を公開できるようにする **MetricsRoutePolicyFactory** も提供します。詳細については、ページの下部を参照してください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-metrics</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

209.2. URI 形式

```
metrics:[ meter | counter | histogram | timer | gauge ]:metricname[?options]
```

209.3. オプション

Metrics コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
metricRegistry (advanced)	カスタム設定された MetricRegistry を使用するには。		MetricRegistry
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Metrics エンドポイントは、URI 構文を使用して設定されます。

```
metrics:metricsType:metricsName
```

パスおよびクエリーパラメーターを使用します。

209.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
metricsType	**必須** 指標のタイプ		MetricsType
metricsName	**必須** メトリックの名前		String

209.3.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
action (producer)	タイマー型使用時のアクション		MetricsTimerAction
decrement (producer)	カウンター型使用時の値を減らします		Long
increment (producer)	カウンター型使用時のインクリメント値		Long
mark (producer)	メーター式使用時の目印		Long
subject (producer)	ゲージ式使用時の対象値		Object
value (producer)	ヒストグラムタイプを使用する場合の値の値		Long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

209.4. METRIC REGISTRY

Camel Metrics コンポーネントは、デフォルトで、60 秒のレポート間隔を持つ **Slf4jReporter** を持つ **MetricRegistry** インスタンスを使用します。このデフォルトのレジストリーは、**MetricRegistry** Bean を提供することでカスタムレジストリーに置き換えることができます。アプリケーションに複数の **MetricRegistry** Bean が存在する場合、名前が **metricRegistry** のものが使用されます。

たとえば、Spring Java 設定を使用すると、次のようになります。

```
@Configuration
public static class MyConfig extends SingleRouteCamelConfiguration {

    @Bean
    @Override
    public RouteBuilder route() {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
```

```
                // define Camel routes here
            }
        };
    }

    @Bean(name = MetricsComponent.METRIC_REGISTRY_NAME)
    public MetricRegistry getMetricRegistry() {
        MetricRegistry registry = ...;
        return registry;
    }
}
```

または CDI を使用:

```
class MyBean extends RouteBuilder {

    @Override
    public void configure() {
      from("...")
          // Register the 'my-meter' meter in the MetricRegistry below
          .to("metrics:meter:my-meter");
    }

    @Produces
    // If multiple MetricRegistry beans
    // @Named(MetricsComponent.METRIC_REGISTRY_NAME)
    MetricRegistry registry() {
        MetricRegistry registry = new MetricRegistry();
        // ...
        return registry;
    }
}
```

注意

MetricRegistry は、レポートに内部スレッドを使用します。バージョン DropWizard **3.0.1** には、ユーザーが終了時にクリーンアップするためのパブリック API はありません。したがって、Camel Metrics コンポーネントを使用すると、Java クラスローダーリークが発生し、場合によっては **OutOfMemoryErrors** が発生します。

209.5. 使用方法

各メトリックにはタイプと名前があります。サポートされているタイプは、counter、histogram、meter、timer、および gauge です。メトリック名は単純な文字列です。メトリックタイプが指定されていない場合、デフォルトでタイプ meter が使用されます。

209.5.1. ヘッダー

URI で定義されたメトリック名は、名前が **CamelMetricsName** の ヘッダーを使用してオーバーライドできます。

以下に例を示します。

```
from("direct:in")
```

```
    .setHeader(MetricsConstants.HEADER_METRIC_NAME, constant("new.name"))
    .to("metrics:counter:name.not.used")
    .to("direct:out");
```

name.not.used の代わりに **new.name** という名前でカウンターを更新します。

Metrics エンドポイントが交換の処理を完了すると、すべての Metrics 固有のヘッダーがメッセージから削除されます。exchange Metrics エンドポイントの処理中に、すべての例外をキャッチし、level **warn** を使用してログエントリーを書き込みます。

209.6. メトリクスタイプカウンター

```
metrics:counter:metricname[?options]
```

209.6.1. オプション

名前	デフォルト	説明
increment	-	カウンターに追加する long 値
decrement	-	カウンターから減算する long 値

increment も **decrement** も定義されていない場合、カウンター値は 1 ずつ増加します。**increment** と **decrement** の両方が定義されている場合、インクリメント操作のみが呼び出されます。

```
// update counter simple.counter by 7
from("direct:in")
    .to("metric:counter:simple.counter?increment=7")
    .to("direct:out");
```

```
// increment counter simple.counter by 1
from("direct:in")
    .to("metric:counter:simple.counter")
    .to("direct:out");
```

```
// decrement counter simple.counter by 3
from("direct:in")
    .to("metric:counter:simple.counter?decrement=3")
    .to("direct:out");
```

209.6.2. ヘッダー

メッセージヘッダーを使用して、メトリクスコンポーネント URI で指定された **increment** と **decrement** 値をオーバーライドできます。

名前	説明	想定されるタイプ
Camel Metrics CounterIncrement	URI のインクリメント値をオーバーライドする	Long
Camel Metrics CounterDecrement	URI のデクリメント値をオーバーライドする	Long

```
// update counter simple.counter by 417
from("direct:in")
    .setHeader(MetricsConstants.HEADER_COUNTER_INCREMENT, constant(417L))
    .to("metric:counter:simple.counter?increment=7")
    .to("direct:out");
```

```
// updates counter using simple language to evaluate body.length
from("direct:in")
    .setHeader(MetricsConstants.HEADER_COUNTER_INCREMENT, simple("${body.length}"))
    .to("metrics:counter:body.length")
    .to("mock:out");
```

209.7. メトリクスタイプのヒストグラム

```
metrics:histogram:metricname[?options]
```

209.7.1. オプション

名前	デフォルト	説明
value	-	ヒストグラムで使用する値

value が設定されていない場合、ヒストグラムには何も追加されず、警告がログに記録されます。

```
// adds value 9923 to simple.histogram
from("direct:in")
    .to("metric:histogram:simple.histogram?value=9923")
    .to("direct:out");
```

```
// nothing is added to simple.histogram; warning is logged
from("direct:in")
```

```
    .to("metric:histogram:simple.histogram")
    .to("direct:out");
```

209.7.2. ヘッダー

メッセージヘッダーを使用して、メトリクスコンポーネント URI で指定された値をオーバーライドできます。

名前	説明	想定されるタイプ
Camel Metrics HistogramValue	URI のヒストグラム値をオーバーライドする	Long

```
// adds value 992 to simple.histogram
from("direct:in")
    .setHeader(MetricsConstants.HEADER_HISTOGRAM_VALUE, constant(992L))
    .to("metric:histogram:simple.histogram?value=700")
    .to("direct:out")
```

209.8. メトリックタイプメーター

```
metrics:meter:metricname[?options]
```

209.8.1. オプション

名前	デフォルト	説明
マーク	-	マークとして使用する長い値

mark が設定されていない場合、 **meter.mark()** が引数なしで呼び出されます。

```
// marks simple.meter without value
from("direct:in")
    .to("metric:simple.meter")
    .to("direct:out");
```

```
// marks simple.meter with value 81
from("direct:in")
    .to("metric:meter:simple.meter?mark=81")
    .to("direct:out");
```

209.8.2. ヘッダー

メッセージヘッダーを使用して、Metrics コンポーネント URI で指定された **mark** 値をオーバーライドできます。

名前	説明	想定されるタイプ
Camel Metrics Meter Mark	URI のマーク値をオーバーライドする	Long

```
// updates meter simple.meter with value 345
from("direct:in")
    .setHeader(MetricsConstants.HEADER_METER_MARK, constant(345L))
    .to("metric:meter:simple.meter?mark=123")
    .to("direct:out");
```

209.9. メトリクスタイプタイマー

```
metrics:timer:metricname[?options]
```

209.9.1. オプション

名前	デフォルト	説明
action	-	開始または停止

action または無効な値が指定されていない場合、タイマーの更新なしで警告がログに記録されます。すでに実行中のタイマーでアクション **start** が呼び出された場合、または実行されていないタイマーで **stop** が呼び出された場合、何も更新されず、警告がログに記録されます。

```
// measure time taken by route "calculate"
from("direct:in")
    .to("metrics:timer:simple.timer?action=start")
    .to("direct:calculate")
    .to("metrics:timer:simple.timer?action=stop");
```

TimerContext オブジェクトは、異なる Metrics コンポーネントの呼び出し間で Exchange プロパティーとして格納されます。

209.9.2. ヘッダー

メッセージヘッダーを使用して、Metrics コンポーネント URI で指定されたアクション値をオーバーライドできます。

名前	説明	想定されるタイプ
Camel Metrics TimerAction	URI のタイマーアクションをオーバーライドする	**org.apache.camel.component.metrics.timer.TimerEndpoint.TimerAction**

```
// sets timer action using header
from("direct:in")
    .setHeader(MetricsConstants.HEADER_TIMER_ACTION, TimerAction.start)
    .to("metric:timer:simple.timer")
    .to("direct:out");
```

209.10. メトリックタイプゲージ

```
metrics:gauge:metricname[?options]
```

209.10.1. オプション

名前	デフォルト	説明
subject	-	ゲージによって観察されるすべてのオブジェクト

subject が定義されていない場合、単に無視されます。つまり、ゲージは登録されません。

```
// update gauge "simple.gauge" by a bean "mySubjectBean"
from("direct:in")
    .to("metric:gauge:simple.gauge?subject=#mySubjectBean")
    .to("direct:out");
```

209.10.2. ヘッダー

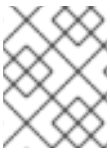
メッセージヘッダーを使用して、Metrics コンポーネント URI で指定された **subject** の値をオーバーライドできます。注記: **CamelMetricsName** ヘッダーが指定されている場合、URI で指定されたデフォルトのゲージに加えて、新しいゲージが登録されます。

名前	説明	想定されるタイプ
Camel Metrics Gauge Subject	URI のサブジェクト値をオーバーライドする	Object

```
// update gauge simple.gauge by a String literal "myUpdatedSubject"
from("direct:in")
    .setHeader(MetricsConstants.HEADER_GAUGE_SUBJECT, constant("myUpdatedSubject"))
    .to("metric:counter:simple.gauge?subject=#mySubjectBean")
    .to("direct:out");
```

209.11. METRICSROUTEPOLICYFACTORY

このファクトリーを使用すると、Dropwizard メトリックを使用してルート使用率の統計を公開するルートごとに RoutePolicy を追加できます。このファクトリーは、以下の例のように Java および XML で使用できます。



注記

MetricsRoutePolicyFactory を使用する代わりに、選択した少数のルートのみを計測する場合に備えて、計測するルートごとに MetricsRoutePolicy を定義できます。

Java DSL の場合は、以下のようにファクトリーを **CamelContext** に追加します。

```
context.addRoutePolicyFactory(new MetricsRoutePolicyFactory());
```

XML DSL の場合は、<bean> を以下のように定義します。

```
  <!-- use camel-metrics route policy to gather metrics for all routes -->
  <bean id="metricsRoutePolicyFactory"
class="org.apache.camel.component.metrics.routepolicy.MetricsRoutePolicyFactory"/>
```

MetricsRoutePolicyFactory および **MetricsRoutePolicy** は、以下のオプションをサポートします。

名前	デフォルト	説明
useJmx	false	**com.codahale.metrics.JmxReporter** を使って、詳細な統計情報を JMX に報告するかどうか。 CamelContext で JMX が有効になっている場合、JMX ツリーのサービスタイプの下に **MetricsRegistryService** mbean が登録されていることに注意してください。この mbean には、統計を JSON 出力する 1 つのオペレーションがあります。**useJmx** を true に設定する必要があるのは、統計タイプごとに細かい mbeans を生成する場合のみです。

名前	デフォルト	説明
jmxDomain	org.apache.camel.metrics	JMX ドメイン名
prettyPrint	false	統計情報を json 形式で出力する際に pretty print を使用するかどうか
metricsRegistry		共有 **com.codahale.metrics.MetricRegistry** の使用を許可します。指定しない場合は、Camel はこの CamelContext によって使用される共有インスタンスを作成します。
rateUnit	TimeUnit.SECONDS	メトリクスレポーターまたは統計を json 出力するときのレートに使用する単位。
durationUnit	TimeUnit.MILLISECONDS	メトリクスレポーターまたは統計を json 出力するときの期間に使用する単位。
namePattern	**name.routeId.type**	Camel 2.17: 使用する名前パターン。セパレータとしてドットを使用しますが、これは変更できます。値 **name**、**routeId**、および **type** は実際の値に置き換えられます。**name** は CamelContext の名前です。**routeId** はルートの名前です。**type** はレスポンスの値です。

以下に示すように、Java コードから
は、**org.apache.camel.component.metrics.routepolicy.MetricsRegistryService** から
com.codahale.metrics.MetricRegistry を取得できます。

```
MetricRegistryService registryService = context.hasService(MetricsRegistryService.class);
if (registryService != null) {
  MetricsRegistry registry = registryService.getMetricsRegistry();
  ...
}
```

209.12. METRICSMESSAGEHISTORYFACTORY

Camel 2.17 以降で利用可能

このファクトリーを使用すると、メトリクスを使用して、メッセージのルーティング中にメッセージ履歴のパフォーマンス統計を取得できます。これは、すべてのルートの各ノードにメトリクスタイマーを使用することで機能します。 このファクトリーは、以下の例のように Java および XML で使用できます。

Java DSL の場合は、以下のようにファクトリーを **CamelContext** に設定するだけです。

```
context.setMessageHistoryFactory(new MetricsMessageHistoryFactory());
```

XML DSL の場合は、<bean> を以下のように定義します。

```
<!-- use camel-metrics message history to gather metrics for all messages being routed -->
<bean id="metricsMessageHistoryFactory"
class="org.apache.camel.component.metrics.messagehistory.MetricsMessageHistoryFactory"/>
```

ファクトリーでは、次のオプションがサポートされています。

名前	デフォルト	説明
useJmx	false	**com.codahale.metrics.JmxReporter** を使って、詳細な統計情報を JMX に報告するかどうか。 CamelContext で JMX が有効になっている場合、JMX ツリーのサービスタイプの下に **MetricsRegistryService** mbean が登録されていることに注意してください。この mbean には、統計を JSON 出力する 1 つのオペレーションがあります。**useJmx** を true に設定する必要があるのは、統計タイプごとに細かい mbeans を生成する場合のみです。
jmxDomain	org.apache.camel.metrics	JMX ドメイン名
prettyPrint	false	統計情報を json 形式で出力する際に pretty print を使用するかどうか
metricsRegistry		共有 **com.codahale.metrics.MetricRegistry** の使用を許可します。指定しない場合は、Camel はこの CamelContext によって使用される共有インスタンスを作成します。
rateUnit	TimeUnit.SECONDS	メトリクスレポーターまたは統計を json 出力するときのレートに使用する単位。
durationUnit	TimeUnit.MILLISECONDS	メトリクスレポーターまたは統計を json 出力するときの期間に使用する単位。
namePattern	**name.routeId.id.type**	使用する名前パターン。セパレータとしてドットを使用しますが、これは変更できます。値 **name**、**routeId**、**type**、および **id** は実際の値に置き換えられます。**name** は CamelContext の名前です。**routeId** はルートの名前です。**id** パターンはノード ID を表します。そして **type** は履歴の値です。

実行時に、メトリクスは Java API または JMX からアクセスでき、JSON 出力としてデータを収集できます。

Java コードから、次のように CamelContext からサービスを取得できます。

```
MetricsMessageHistoryService service = context.hasService(MetricsMessageHistoryService.class);
String json = service.dumpStatisticsAsJson();
```

また、JMX API MBean は **type=services** ツリーに **name=MetricsMessageHistoryService** で登録されます。

209.13. INSTRUMENTEDTHREADPOOLFACTORY

Camel 2.18 から利用可能

このファクトリーを使用すると、Camel の内部から情報を収集する InstrumentedThreadPoolFactory を注入することで、Camel スレッドプールに関するパフォーマンス情報を収集できます。詳細については、Spring を使用した CamelContext の高度な設定を参照してください。

209.14. 関連項目

- Camel、Metrics、CDI の統合を示す **camel-example-cdi-metrics** の例。

第210章 OPC UA CLIENT コンポーネント

Camel バージョン 2.19 以降で利用可能

Milo Client コンポーネントは、Eclipse Milo™ 実装を使用して OPC UA サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-milo</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

OPC UA Client コンポーネントは、以下に示す 6 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
defaultConfiguration (common)	クライアントのすべてのデフォルトオプション		MiloClientConfiguration
applicationName (Common)	デフォルトのアプリケーション名		String
applicationUri (common)	デフォルトのアプリケーション URI		String
productUri (common)	デフォルトの製品 URI		String
reconnectTimeout (common)	デフォルトの再接続タイムアウト		Long
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

210.1. URI 形式

エンドポイントの URI 構文は次のとおりです。

```
milo-client:tcp://[user:password@]host:port/path/to/service?node=RAW(nsu=urn:foo:bar;s=item-1)
```

サーバーがパスを使用しない場合は、単純に省略できます。

```
milo-client:tcp://[user:password@]host:port?node=RAW(nsu=urn:foo:bar;s=item-1)
```

ユーザー認証情報が提供されない場合、クライアントは匿名モードに切り替わります。

210.2. URI オプション

グループクライアントのすべての設定オプションは、共有クライアントインスタンスに適用されます。エンドポイントは、エンドポイント URI ごとにクライアントインスタンスを共有します。したがって、そのエンドポイント URI に対する最初の要求が行われたときに、クライアントグループのオプションが適用されます。それ以降のインスタンスはすべて無視されます。

同じエンドポイント URI に別のオプションが必要な場合は、別の共有接続インスタンスを選択するためにエンドポイント URI に内部的に追加される clientId オプションを設定することができます。つまり、エンドポイント URI とクライアント ID の組み合わせによって特定される共有接続です。

OPC UA クライアントエンドポイントは、URI 構文を使用して設定されます。

```
milo-client:endpointUri
```

パスおよびクエリーパラメーターを使用します。

210.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
endpointUri	**必須** OPC UA サーバーのエンドポイント		文字列

210.2.2. クエリーパラメーター (24 パラメーター)

名前	説明	デフォルト	タイプ
clientId (common)	新しい接続インスタンスの作成を強制するための仮想クライアント ID		String
defaultAwaitWrites (Common)	書き込みのデフォルトの待機設定	false	boolean
node (Common)	ノード定義 (ノード ID を参照)		ExpandedNodeId
samplingInterval (Common)	ミリ秒単位のサンプリング間隔		double

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
allowedSecurityPolicies (client)	許可されたセキュリティーポリシー URI のセット。デフォルトでは、すべてを受け入れて最高のものを使用します。		String
applicationName (client)	アプリケーション名	Eclipse Milo 用の Apache Camel アダプター	String
applicationUri (client)	アプリケーション URI	http://camel.apache.org/EclipseMilo/Client	String
channelLifetime (クライアント)	チャネルの有効期間 (ミリ秒)		Long

名前	説明	デフォルト	タイプ
keyAlias (クライアント)	キーストアファイル内のキーの名前		String
keyPassword (クライアント)	キーのパスワード		String
keyStorePassword (client)	キーストアのパスワード		String
keyStoreType (client)	キーストアのタイプ		String
keyStoreUrl (client)	キーのロード元の URL		URL
maxPendingPublishRequests (クライアント)	保留中の公開リクエストの最大数		Long
maxResponseMessageSize (クライアント)	応答メッセージの最大バイト数		Long
overrideHost (クライアント)	サーバーが報告したエンドポイントホストを、エンドポイント URI のホストでオーバーライドします。	false	boolean
productUri (client)	製品の URI	http://camel.apache.org/EclipseMilo	String
requestTimeout (クライアント)	リクエストのタイムアウト (ミリ秒)		Long
sessionName (client)	セッション名		String
sessionTimeout (client)	ミリ秒単位のセッションタイムアウト		Long

210.2.3. Node ID

ターゲットノードを定義するには、名前空間とノード ID が必要です。以前のバージョンでは、これは

nodeId と **namespaceUri** または **namespaceIndex** のいずれかを指定することで可能でした。ただし、これは文字列ベースのノード ID の使用のみを許可していました。この設定はまだ可能ですが、新しい方が優先されます。

新しいアプローチは、完全な名前空間とノード ID を **ns=1;i=1** の形式で指定することです。これにより、他のノード ID 形式 (数値、GUID/UUID、不透明など) も使用できます。**node** パラメーターを使用する場合、古いものは使用しないでください。このノード形式の構文は、セミコロン (;) で区切られた **key=value** のペアのセットです。

正確に 1 つの名前空間と 1 つのノード ID キーを使用する必要があります。設定可能なキーについては、次の表を参照してください。

キー	タイプ	説明
ns	namespace	数値ネームスペースインデックス
nsu	namespace	名前空間 URI
s	node	文字列ノード ID
i	node	数値ノード ID
g	node	GUID/UUID ノード ID
b	node	不透明なノード ID の Base64 でエンコードされた文字列

構文によって生成された値は透過的に URI パラメーター値にエンコードできないため、エスケープする必要があります。ただし、Camel では実際の値を **RAW(...)** 内にラップできるため、エスケープが不要になります。以下はその例です。

```
milo-client://user:password@localhost:12345?node=RAW(nsu=http://foo.bar;s=foo/bar)
```

210.2.4. セキュリティーポリシー

許可するセキュリティーポリシーを設定する場合、既知の OPC UA URI (例: **http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15**) を使用するか、Milo 列挙型リテラル (例: **None**) を使用できます。不明なセキュリティーポリシーの URI または列挙を指定するとエラーになります。

既知のセキュリティーポリシー URI と列挙型リテラルは、SecurityPolicy.java で確認できます。

注記: いずれの場合も、セキュリティーポリシーでは大文字と小文字が区別されると見なされます。

210.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- スタートガイド

第211章 OPC UA サーバーコンポーネント

Camel バージョン 2.19 以降で利用可能

Milo サーバー コンポーネントは 、Eclipse Milo™ 実装を使用して OPC UA サーバーを提供します。

Java 8 : このコンポーネントには、実行時に Java 8 が必要です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-milo</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel からエンドポイントに送信されたメッセージは、OPC UA サーバーから OPC UA クライアントで利用できます。OPC UA クライアントからの値の書き込み要求は、Apache Camel に送信されるメッセージをトリガーします。

OPC UA サーバーコンポーネントは、以下に示す 19 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
namespaceUri (common)	名前空間の URI。デフォルトは urn:org:apache:camel です。		String
applicationName (Common)	アプリケーション名		String
applicationUri (common)	アプリケーション URI		String
productUri (common)	製品の URI		String
bindPort (common)	サーバーがバインドする TCP ポート		int
strictEndpointUrls Enabled (common)	厳密なエンドポイント URL を適用するかどうかを設定します	false	boolean
serverName (common)	サーバー名		String
hostname (common)	サーバーのホスト名		String

名前	説明	デフォルト	タイプ
securityPolicies (common)	セキュリティーポリシー		Set
securityPoliciesById (common)	URI または名前によるセキュリティーポリシー		String>
userAuthentication Credentials (common)	ユーザーパスワードの組み合わせを user1:pwd1,user2:pwd2 の形式で設定しますユーザー名とパスワードは URL デコードされます		String
enableAnonymous Authentication (common)	匿名認証を有効にします。デフォルトでは無効になっています	false	boolean
bindAddresses (common)	サーバーがバインドするローカルアドレスのアドレスを設定します。		String
buildInfo (common)	サーバーのビルド情報		BuildInfo
serverCertificate (common)	サーバー証明書		結果
certificateManager (common)	サーバー証明書マネージャー		CertificateManager
certificateValidator (common)	クライアント証明書のバリデーター		CertificateValidator>
defaultCertificateValidator (common)	デフォルトのファイルベースのアプローチを使用したクライアント証明書のバリデーター		File
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

211.1. URI 形式

```
milo-server:itemId[?options]
```

211.2. URI オプション

OPC UA サーバーエンドポイントは、URI 構文を使用して設定されます。

```
milo-server:itemId
```

パスおよびクエリーパラメーターを使用します。

211.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
itemId	**必須** アイテムの ID		String

211.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

211.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第212章 MIME マルチパートデータ形式

Camel バージョン 2.17 以降で利用可能

添付ファイル付きの Camel メッセージを、メッセージボディーとして MIME-Multipart メッセージを持つ (添付ファイルなしの) Camel メッセージに変換できるこのデータ形式。

これの使用例は、特別なプロトコルの実装 (例: HTTP エンドポイントを介して MIME マルチパートを送信する) または一種のトンネリングソリューション (例: camel のため) として、添付を直接サポートしていないエンドポイントを介してユーザーが添付を送信できるようにすることです。-jms は添付をサポートしませんが、添付のあるメッセージを MIME-Multipart にマーシャリングし、それを JMS キューに送信し、JMS キューからメッセージを受信して、再度アンマーシャリングします (添付のあるメッセージボディーに)。

mime-multipart データ形式のマーシャルオプションは、添付ファイル付きのメッセージを MIME-Multipart メッセージに変換します。パラメーター multipartWithoutAttachment が true に設定されている場合、添付ファイルのないメッセージも単一のパートを持つマルチパートメッセージにマーシャリングされます。パラメーターが false に設定されている場合、メッセージはそのままになります。

MIME-Version および Content-Type としてのマルチパートの MIME ヘッダーは、キャメルヘッダーとしてメッセージに設定されます。パラメーター headersInline が true に設定されている場合、MIME マルチパートメッセージも作成されます。
さらに、マルチパートの MIME ヘッダーは、camel ヘッダーとしてではなく、メッセージボディーの一部として書き込まれます。

mime-multipart データ形式の unmarshal オプションは、MIME-Multipart メッセージを添付ファイル付きの camel メッセージに変換し、他のメッセージはそのままにします。MIME-Multipart メッセージの MIME-Headers は、Camel ヘッダーとして設定する必要があります。アンマーシャリングは、Content-Type ヘッダーがマルチパートタイプに設定されている場合にのみ行われます。オプション headersInline が true に設定されている場合、ボディーは常に MIME メッセージとして解析されます。その結果、メッセージボディーがストリームであり、ストリームキャッシングが有効になっていない場合、実際にはメッセージボディーで MIME を使用した MIME メッセージではないメッセージボディーのヘッダーは空のメッセージに置き換えられます。Camel バージョン 2.17.1 までは、ボディタイプとストリームキャッシュ設定に関係なく、MIME マルチパートメッセージを含まないすべてのメッセージボディでこれが発生します。

212.1. オプション

MIME Multipart データ形式は、以下に示す 6 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
multipartSubType	mixed	String	MIME マルチパートのサブタイプを指定します。デフォルトは混合です。
multipartWithoutAttachment	false	Boolean	添付のないメッセージも MIME マルチパート (ボディパーツが 1 つだけ) にマーシャリングされるかどうかを定義します。デフォルトは false です。
headersInline	false	Boolean	MIME-Multipart ヘッダーがメッセージ本文の一部であるか (true)、Camel ヘッダーとして設定されているか (false) を定義します。デフォルトは false です。

名前	デフォルト	Java タイプ	説明
includeHeaders		**String**	MIME マルチパートに MIME ヘッダーとして含まれる Camel ヘッダーを定義する正規表現。これは、headersInline が true に設定されている場合にのみ機能します。デフォルトでは、ヘッダーは含まれません
binaryContent	**false**	**Boolean**	MIME マルチパートのバイナリーパートのコンテンツがバイナリー (true) か Base-64 エンコード (false) かを定義します。デフォルトは false です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

212.2. メッセージヘッダー (マーシャル)

名前	タイプ	説明
Message-Id	String	headersInline パラメーターが false に設定されている場合、マーシャル操作はこのパラメーターを生成された MIME メッセージ ID に設定します。
MIME-Version	String	headersInline パラメーターが false に設定されている場合、マーシャル操作はこのパラメーターを適用された MIME バージョン (1.0) に設定します。
Content-Type	String	このヘッダーのコンテンツは、メッセージボディー部分のコンテンツタイプとして使用されます。コンテンツタイプが設定されていない場合、application/octet-stream が想定されます。整列化操作の後、headersInline パラメーターが true に設定されている場合、コンテンツタイプは multipart/related または空に設定されます。
Content-Encoding	String	入力コンテンツタイプが text/*の場合、コンテンツエンコーディングは、ボディパーツの Content-Type MIME ヘッダーのエンコーディングパラメーターに設定されます。さらに、指定された文字セットは、テキストからバイナリーへの変換に適用されます。

212.3. メッセージヘッダー (非整列化)

名前	タイプ	説明

名前	タイプ	説明
Content-Type	String	このヘッダーが multipart/*に設定されていない場合、非整列化操作は何も実行しません。それ以外の場合、マルチパートは添付ファイル付きの camel メッセージに解析され、ヘッダーはボディー部分の Content-Type ヘッダーに設定されます。ただし、これが application/octet-stream の場合を除きます。後者の場合、ヘッダーは削除されます。
Content-Encoding	String	ボディー部分の content-type にエンコーディングパラメーターが含まれている場合、このヘッダーはこのエンコーディングパラメーターの値に設定されます (MIME エンコーディング記述子から Java エンコーディング記述子に変換されます)。
MIME-Version	String	非整列化操作はこのヘッダーを読み取り、MIME マルチパートの解析に使用します。ヘッダーは後で削除されます

212.4. 例

```
from(...).marshal().mimeMultipart()
```

Content-Type ヘッダーが設定されていないメッセージでは、次のメッセージ Camel ヘッダーを持つメッセージが作成されます。

Camel メッセージヘッダー

```
Content-Type=multipart/mixed; \n boundary="----=_Part_0_14180567.1447658227051"
Message-Id=<...>
MIME-Version=1.0
```

```
The message body will be:
```

Camel メッセージボディー

```
------=_Part_0_14180567.1447658227051
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Qm9keSB0ZXh0
------=_Part_0_14180567.1447658227051
Content-Type: application/binary
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Attachment File Name"
AAECAwQFBgc=
------=_Part_0_14180567.1447658227051--
```

ヘッダー Content-Type が text/plain に設定されたメッセージがルートに送信されます

```
from("...").marshal().mimeMultipart("related", true, true, "(included|x-.*)", true);
```

Camel ヘッダーとして設定された特定の MIME ヘッダーを持たないメッセージ (Content-Type ヘッダーは Camel メッセージから削除されます) と、x- で始まる元のメッセージのすべてのヘッダーと名前 include のヘッダーも含む次のメッセージボディーを作成します:

Camel メッセージボディー

```
Message-ID: <...>
MIME-Version: 1.0
Content-Type: multipart/related;
    boundary="----=_Part_0_1134128170.1447659361365"
x-bar: also there
included: must be included
x-foo: any value

------=_Part_0_1134128170.1447659361365
Content-Type: text/plain
Content-Transfer-Encoding: 8bit

Body text
------=_Part_0_1134128170.1447659361365
Content-Type: application/binary
Content-Transfer-Encoding: binary
Content-Disposition: attachment; filename="Attachment File Name"

[binary content]
------=_Part_0_1134128170.1447659361365
```

212.5. 依存関係

Camel ルートで MIME-Multipart を使用するには、このデータ形式を実装する **camel-mail** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけです。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail</artifactId>
  <version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

第213章 MINA2 コンポーネント

Camel バージョン 2.10 以降で利用可能

mina2: コンポーネントは、Apache MINA 2.x を操作するためのトランスポートです。

ヒント

Netty は現在の Apache Mina よりもはるかに活発に維持され、人気のあるプロジェクトであるため、Netty を使用することをお勧めします。

情報: コンシューマーエンドポイントで sync=false に注意してください。camel-mina2 以降、すべてのコンシューマーエクスチェンジは InOut です。これは camel-mina とは異なります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mina2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

213.1. URI 形式

```
mina2:tcp://hostname[:port][?options]
mina2:udp://hostname[:port][?options]
mina2:vm://hostname[:port][?options]
```

codec オプションを使用して、レジストリーでコーデックを指定できます。TCP を使用していて、コーデックが指定されていない場合は、**textline** フラグを使用して、代わりにテキスト行ベースのコーデックまたはオブジェクトのシリアライゼーションを使用するかどうかを決定します。デフォルトでは、オブジェクトのシリアル化が使用されます。

UDP の場合、コーデックが指定されていない場合、デフォルトでは基本的な **ByteBuffer** ベースのコーデックが使用されます。

仮想マシンプロトコルは、同じ JVM で直接転送メカニズムとして使用されます。

Mina プロデューサーには、リモートサーバーからの応答を待つ間のデフォルトのタイムアウト値が 30 秒あります。

通常の使用では、**camel-mina** はボディーコンテンツのマーシャリングのみをサポートします。メッセージヘッダーとエクスチェンジプロパティーは送信されません。
ただし、オプション **transferExchange** を使用すると、エクスチェンジ自体を転送することができます。以下のオプションを参照してください。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

213.2. オプション

Mina2 コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 mina 設定を使用する場合。		Mina2Configuration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Mina2 エンドポイントは、URI 構文を使用して設定されます。

```
mina2:protocol:host:port
```

パスおよびクエリーパラメーターを使用します。

213.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
protocol	**必須** 使用するプロトコル		String
host	**必須** 使用するホスト名。コンシューマーとしてのローカルサーバーには localhost または 0.0.0.0 を使用します。プロデューサーには、リモートサーバーのホスト名または IP アドレスを使用します。		String
port	**必須** ポート番号		int

213.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
disconnect (Common)	Mina セッションを使用直後に切断 (クローズ) するかどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
minaLogger (common)	Apache MINA ロギングフィルターを有効にできます。Apache MINA は、INFO レベルで slf4j ロギングを使用して、すべての入出力をログに記録します。	false	boolean

名前	説明	デフォルト	タイプ
sync (common)	エンドポイントを一方向または要求応答として設定する設定。	true	boolean
timeout (common)	リモートサーバーからのレスポンスを待機する時間を指定するタイムアウトを設定できます。タイムアウトの単位はミリ秒なので、60000 は 60 秒です。	30000	long
writeTimeout (common)	MINA セッションにデータを送信するのにかかる最大時間。デフォルトは 10000 ミリ秒です。	10000	long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clientMode (consumer)	clientMode が true の場合、mina コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean
disconnectOnNoReply (consumer)	同期が有効になっている場合、このオプションは、返信がない場合に MinaConsumer を切断するかどうかを指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
noReplyLogLevel (consumer)	同期が有効になっている場合、このオプションは MinaConsumer がログに記録するときに使用するログレベルを決定し、返信する応答がありません。	WARN	LoggingLevel
cachedAddress (producer)	InetAddress を一度作成して再利用するかどうか。これを false に設定すると、ネットワーク内の DNS の変更を取得できます。	true	boolean

名前	説明	デフォルト	タイプ
lazySessionCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにセッションを遅延作成できます。	true	boolean
maximumPoolSize (advanced)	TCP および UDP のワーカープール内のワーカースレッドの数	16	int
orderedThreadPoolExecutor (advanced)	順序付けられたスレッドプールを使用して、イベントが同じチャネルで順番に処理されるかどうか。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク経由でエクスチェンジを転送することができます。In body、Out body、fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、exchange プロパティー、exchange 例外フィールドが転送されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
allowDefaultCodec (codec)	mina コンポーネントは、codec が null で textline が false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、mina コンポーネントがデフォルトコーデックをフィルターチェーンの最初の要素としてインストールするのを防ぎます。これは、SSL フィルターのように、別のフィルターをフィルターチェーンの最初にする必要があるシナリオで役立ちます。	true	boolean
codec (codec)	カスタム Minda コーデックの実装を使用する場合。		ProtocolCodecFactory
decoderMaxLineLength (codec)	テキストラインプロトコルデコーダーの最大行長を設定する場合。デフォルトでは、Mina 自体のデフォルト値である 1024 が使用されます。	1024	int
encoderMaxLineLength (codec)	テキストラインプロトコルエンコーダーの最大行長を設定します。デフォルトでは、Mina 自体のデフォルト値である Integer.MAX_VALUE が使用されます。	-1	int

名前	説明	デフォルト	タイプ
encoding (codec)	TCP テキストラインコーデックと UDP プロトコルに使用するエンコーディング (文字セット名) を設定できます。指定しない場合、Camel は JVM のデフォルトの文字セット を使用します。		String
filters (codec)	使用する Mina IoFilter のリストを設定できます。		List
textline (codec)	TCP にのみ使用されます。コーデックが指定されていない場合、このフラグを使用して、テキスト行ベースのコーデックを示すことができます。指定されていない場合、または値が false の場合、オブジェクトのシリアル化は TCP 経由であると想定されます。	false	boolean
textlineDelimiter (codec)	TCP で textline=true の場合にのみ使用されます。使用するテキスト行区切り文字を設定します。何も指定されていない場合、Camel は DEFAULT を使用します。この区切り文字は、テキストの終わりを示すために使用されます。		Mina2TextLineDelimiter
autoStartTls (security)	SSL ハンドシェイクを自動開始するかどうか。	true	boolean
sslContextParameters (security)	SSL セキュリティーを設定します。		SSLContextParameters

213.3. カスタムコーデックの使用

Mina が独自のコーデックを作成する方法を参照してください。カスタムコーデックを **camel-mina** で使用するには、コーデックをレジストリーに登録する必要があります。たとえば、Spring XML ファイルで Bean を作成します。次に **codec** オプションを使用して、コーデックの Bean ID を指定します。カスタムコーデックを持つ HL7 を参照してください。

213.4. SYNC=FALSE のサンプル

このサンプルでは、Camel はポート 6200 で TCP 接続をリッスンするサービスを公開します。**テキストライン** コーデックを使用します。このルートでは、ポート 6200 でリッスンする Mina コンシューマーエンドポイントを作成します。

```
from("mina2:tcp://localhost:" + port1 + "?textline=true&sync=false").to("mock:result");
```

サンプルは単体テストの一部であるため、ポート 6200 でデータを送信してテストします。

```
MockEndpoint mock = getMockEndpoint("mock:result");
mock.expectedBodiesReceived("Hello World");
```

```
template.sendBody("mina2:tcp://localhost:" + port1 + "?textline=true&sync=false", "Hello World");

assertMockEndpointsSatisfied();
```

213.5. SYNC=TRUE のサンプル

次のサンプルでは、ポート 6201 で TCP サービスを公開し、テキストラインコーデックも使用する、より一般的なユースケースがあります。ただし、今回は応答を返す必要があるため、コンシューマーで **sync** オプションを **true** に設定します。

```
from("mina2:tcp://localhost:" + port2 + "?textline=true&sync=true").process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        String body = exchange.getIn().getBody(String.class);
        exchange.getOut().setBody("Bye " + body);
    }
});
```

次に、いくつかのデータを送信し、**template.requestBody()** メソッドを使用してレスポンスを取得することにより、サンプルをテストします。レスポンスが **String** であることがわかっているので、それを **String** にキャストし、レスポンスが実際にはプロセッサーコードロジックで動的に設定したものであると断言できます。

```
String response = (String)template.requestBody("mina2:tcp://localhost:" + port2 + "?
textline=true&sync=true", "World");
assertEquals("Bye World", response);
```

213.6. SPRING DSL を使用したサンプル

もちろん、Spring DSL は MINA にも使用できます。以下のサンプルでは、ポート 5555 で TCP サーバーを公開しています。

```
<route>
  <from uri="mina2:tcp://localhost:5555?textline=true"/>
  <to uri="bean:myTCPOrderHandler"/>
</route>
```

上記のルートでは、テキストラインコーデックを使用してポート 5555 で TCP サーバーを公開します。ID が **myTCPOrderHandler** の Spring Bean にリクエストを処理させ、応答を返します。たとえば、ハンドラー Bean は次のように実装できます。

```
public String handleOrder(String payload) {
    ...
    return "Order: OK"
}
```

213.7. 完了時にセッションを閉じる

サーバーとして機能している場合、たとえばクライアントの変換が終了したときにセッションを閉じたい場合があります。Camel にセッションを閉じるように指示するには、キー **CamelMinaCloseSessionWhenComplete** をブール値 **true** に設定したヘッダーを追加する必要があります。

たとえば、次の例では、**bye** メッセージをクライアントに書き戻した後にセッションを閉じます。

```
        from("mina2:tcp://localhost:8080?sync=true&textline=true").process(new Processor() {
            public void process(Exchange exchange) throws Exception {
                String body = exchange.getIn().getBody(String.class);
                exchange.getOut().setBody("Bye " + body);

exchange.getOut().setHeader(Mina2Constants.MINA_CLOSE_SESSION_WHEN_COMPLETE,
true);
            }
        });
```

213.8. メッセージの IOSESSION を取得する

このキー **Mina2Constants.MINA_IOSESSION** を使用してメッセージヘッダーから IoSession を取得できます。また、キー **Mina2Constants.MINA_LOCAL_ADDRESS** を使用してローカルホストアドレスを取得し、キー **Mina2Constants.MINA_REMOTE_ADDRESS** を使用してリモートホストアドレスを取得することもできます。

213.9. MINA フィルターの設定

フィルターを使用すると、**SslFilter** などの一部の Mina フィルターを使用できます。カスタマイズされたフィルターを実装することもできます。**codec** と **logger** も **IoFilter** タイプの Mina フィルターとして実装されることに注意してください。定義できるフィルターはすべて、フィルターチェーンの最後に追加されます。つまり、**codec** と **logger** の後です。

213.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Netty

第214章 MLLP コンポーネント

Camel バージョン 2.17 以降で利用可能

MLLP コンポーネントは、MLLP プロトコルのニュアンスを処理し、医療機関が MLLP プロトコルを使用して他のシステムと通信するために必要な機能を提供するように特別に設計されています。 MLLP コンポーネントは、単純な設定 URI、自動化された HL7 確認応答生成、および自動確認応答問い合わせを提供します。

MLLP プロトコルは、通常、多数の同時 TCP 接続を使用しません。通常、1 つのアクティブな TCP 接続が使用されます。 したがって、MLLP コンポーネントは、標準の Java ソケットに基づく単純な接続ごとのスレッドモデルを使用します。これにより、実装がシンプルになり、Camel 自体以外の依存関係がなくなります。

コンポーネントは以下をサポートします。

- TCP サーバーを使用する Camel コンシューマー
- TCP クライアントを使用する Camel プロデューサー

MLLP コンポーネントは byte[] ペイロードを使用し、Camel 型変換に依存して byte[] を他の型に変換します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mllp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

214.1. MLLP オプション

MLLP コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
logPhi (advanced)	PHI データをログに記録するようにコンポーネントを設定します。	true	Boolean
logPhiMaxBytes (advanced)	ログエントリーに記録される PHI の最大バイト数を設定します。	5120	Integer
defaultCharset (advanced)	バイトから文字列への変換に使用するデフォルトの文字セットを設定します。	ISO-8859-1	String
configuration (common)	MLLP エンドポイントの作成時に使用する既定の設定を設定します。		MllpConfiguration

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MLLP エンドポイントは、URI 構文を使用して設定されます。

```
mllp:hostname:port
```

パスおよびクエリーパラメーターを使用します。

214.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
hostname	TCP 接続の接続に **必要な** ホスト名または IP。デフォルト値は null で、これは任意のローカル IP アドレスを意味します		String
port	**必須** TCP 接続のポート番号		int

214.1.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
autoAck (Common)	MLLP 確認応答 MLLP コンシューマーのみの自動生成を有効または無効にします	true	boolean
bufferWrites (Common)	**非推奨** ソケットに書き込む前の HL7 ペイロードのバッファーリングを有効/無効にします。	false	boolean
hl7Headers (Common)	HL7 メッセージ MLLP コンシューマーのみからのメッセージヘッダーの自動生成を有効または無効にします	true	boolean

名前	説明	デフォルト	タイプ
requireEndOfData (common)	MLLP 標準への厳密な準拠を有効/無効にします。MLLP 標準は START_OF_BLOCKhl7 ペイロード END_OF_BLOCKEND_OF_DATA を指定していますが、一部のシステムは最後の END_OF_DATA バイトを送信しません。この設定は、最後の END_OF_DATA バイトが必須かオプションかを制御します。	true	boolean
stringPayload (common)	ペイロードの文字列への変換を有効または無効にします。有効にすると、外部システムから受信した HL7 ペイロードが検証され、文字列に変換されます。charsetName プロパティーが設定されている場合、その文字セットが変換に使用されます。charsetName プロパティーが設定されていない場合、適切な文字セットを決定するために MSH-18 の値が使用されます。MSH-18 が設定されていない場合、デフォルトの ISO-8859-1 文字セットが使用されます。	true	boolean
validatePayload (Common)	HL7 ペイロードの検証を有効/無効にする 有効にすると、外部システムから受信した HL7 ペイロードが検証されます (検証の詳細については、Hl7Util.generateInvalidPayloadExceptionMessage を参照してください)。無効なペイロードが検出された場合、MllpInvalidMessageException (コンシューマーの場合) または MllpInvalidAcknowledgementException が出力されます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。無効にすると、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外を WARN または ERROR レベルでログに記録し、無視することで例外を処理します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。	InOut	ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するかどうかを設定します (このコンポーネントは同期操作のみをサポートします)。	true	boolean
backlog (tcp)	着信接続指示 (接続要求) の最大キュー長は、backlog パラメーターに設定されます。キューがいっぱいのときに接続指示が到着すると、接続は拒否されます。	5	Integer
lenientBind (tcp)	TCP サーバーのみ - TCP ServerSocket がバインドされる前にエンドポイントを開始できるようにします。一部の環境では、TCP ServerSocket がバインドされる前にエンドポイントを開始できるようにすることが望ましい場合があります。	false	boolean
maxConcurrentConsumers (tcp)	許可される同時 MLLP コンシューマー接続の最大数。新しい接続が受信され、最大数がすでに確立されている場合、新しい接続はすぐにリセットされます。	5	int
reuseAddress (tcp)	SO_REUSEADDR ソケットオプションを有効/無効にします。	false	Boolean
acceptTimeout (タイムアウト)	TCP 接続の待機中のタイムアウト (ミリ秒単位) TCP サーバーのみ	60000	int
bindRetryInterval (timeout)	TCP サーバーのみ - バインド試行間で待機するミリ秒数	5000	int
bindTimeout (タイムアウト)	TCP サーバーのみ - サーバーポートへのバインドを再試行するミリ秒数	30000	int
connectTimeout (タイムアウト)	TCP 接続を確立するためのタイムアウト (ミリ秒単位)。TCP クライアントのみ	30000	int
idleTimeout (タイムアウト)	クライアント TCP 接続がリセットされるまでに許容されるおおよそのアイドル時間。null 値またはゼロ以下の値は、アイドルタイムアウトを無効にします。		Integer
maxReceiveTimeouts (タイムアウト)	非推奨 TCP 接続がリセットされるまでに **許容される** タイムアウト (receiveTimeout で指定) の最大数。		Integer

名前	説明	デフォルト	タイプ
keepAlive (tcp)	SO_KEEPALIVE ソケットオプションを有効/無効にします。	true	Boolean
receiveBufferSize (tcp)	SO_RCVBUF オプションを指定された値 (バイト単位) に設定します	8192	Integer
sendBufferSize (tcp)	SO_SNDBUF オプションを指定された値 (バイト単位) に設定します	8192	Integer
tcpNoDelay (tcp)	TCP_NODELAY ソケットオプションを有効/無効にします。	true	Boolean
readTimeout (タイムアウト)	MLLP フレームの開始後に使用される SO_TIMEOUT 値 (ミリ秒単位) が受信されました	5000	int
receiveTimeout (タイムアウト)	MLLP フレームの開始を待機するときに使用される SO_TIMEOUT 値 (ミリ秒単位)	15000	int
charsetName (codec)	エクスチェンジで CamelCharsetName プロパティーを設定します		文字列

214.2. MLLP コンシューマー

MLLP コンシューマーは、MLLP フレームメッセージの受信と HL7 確認応答の送信をサポートします。 MLLP コンシューマーは、HL7 確認応答 (HL7 アプリケーション確認応答のみ - AA、AE、および AR) を自動的に生成するか、CamelMllpAcknowledgement エクスチェンジプロパティーを使用して確認応答を指定できます。 さらに、生成される確認応答のタイプは、CamelMllpAcknowledgementType 交換プロパティーを設定することで制御できます。

214.3. メッセージヘッダー

MLLP コンシューマーは、Camel メッセージに次のヘッダーを追加します。

キー	説明	
CamelMllpLocalAddress	ソケットのローカル TCP アドレス	
CamelMllpRemoteAddress	ソケットのローカル TCP アドレス	
CamelMllpSendingApplication	MSH-3 値	
CamelMllpSendingFacility	MSH-4 値	

CamelMllpReceivingApplication	MSH-5 値	
CamelMllpReceivingFacility	MSH-6 値	
CamelMllpTimestamp	MSH-7 値	
CamelMllpSecurity	MSH-8 値	
CamelMllpMessageType	MSH-9 値	
CamelMllpEventType	MSH-9-1 値	
CamelMllpTriggerEvent	MSH-9-2 値	
CamelMllpMessageControlId	MSH-10 値	
CamelMllpProcessingId	MSH-11 値	
CamelMllpVersionId	MSH-12 値	
CamelMllpCharset	MSH-18 値	

すべてのヘッダーは文字列型です。ヘッダー値が欠落している場合、その値は null です。

214.4. エクスチェンジプロパティー

MLLP コンシューマーが生成する確認応答のタイプと TCP ソケットの状態は、Camel 交換の次のプロパティーによって制御できます。

キー	タイプ	説明
CamelMllpAcknowledgement	byte[]	存在する場合、このプロパティーは MLLP 確認応答としてクライアントに送信されます
CamelMllpAcknowledgementString	String	存在し、CamelMllpAcknowledgement が存在しない場合、このプロパティーは MLLP 確認としてクライアントに送信されます

CamelMllpAcknowledgementMsaText	文字列	CamelMllpAcknowledgement も CamelMllpAcknowledgementString も存在せず、autoAck が true の場合、このプロパティーを使用して、生成された HL7 確認応答で MSA-3 の内容を指定できます。
CamelMllpAcknowledgementType	String	CamelMllpAcknowledgement も CamelMllpAcknowledgementString も存在せず、autoAck が true の場合、このプロパティーを使用して HL7 確認応答タイプ (つまり、AA、AE、AR) を指定できます。
CamelMllpAutoAcknowledge	Boolean	autoAck クエリーパラメーターをオーバーライドします
CamelMllpCloseConnectionBeforeSend	Boolean	true の場合、データを送信する前にソケットが閉じられます
CamelMllpResetConnectionBeforeSend	Boolean	true の場合、データを送信する前にソケットがリセットされます
CamelMllpCloseConnectionAfterSend	Boolean	true の場合、Socket はデータ送信直後に閉じられます
CamelMllpResetConnectionAfterSend	Boolean	true の場合、データを送信した直後にソケットがリセットされます

214.5. MLLP プロデューサー

MLLP Producer は、MLLP フレームメッセージの送信と HL7 確認応答の受信をサポートしています。 MLLP プロデューサーは HL7 確認応答を調べ、否定応答を受信した場合は例外を発生させます。 受信した確認応答が調査され、否定応答の場合は例外が発生します。

214.6. メッセージヘッダー

MLLP プロデューサーは、Camel メッセージに次のヘッダーを追加します。

キー	説明	
CamelMllpLocalAddress	ソケットのローカル TCP アドレス	

CamelMllpRemoteAddress	ソケットのリモート TCP アドレス	
CamelMllpAcknowledgement	受信した HL7 確認バイト[]	
CamelMllpAcknowledgementString	受信した HL7 確認応答を文字列に変換	

214.7. エクスチェンジプロパティー

TCP ソケットの状態は、Camel 交換の次のプロパティーによって制御できます。

キー	タイプ	説明
CamelMllpCloseConnectionBeforeSend	Boolean	true の場合、データを送信する前にソケットが閉じられます
CamelMllpResetConnectionBeforeSend	Boolean	true の場合、データを送信する前にソケットがリセットされます
CamelMllpCloseConnectionAfterSend	Boolean	true の場合、Socket はデータ送信直後に閉じられます
CamelMllpResetConnectionAfterSend	Boolean	true の場合、データを送信した直後にソケットがリセットされます

第215章 MOCK コンポーネント

Mock コンポーネントのドキュメントは現在利用できません。

第216章 MONGODB COMPONENT

Camel バージョン 2.10 以降で利用可能

ウィキペディアによると、NoSQL は、リレーショナルデータベースと ACID 保証の長い歴史を破る、大まかに定義された非リレーショナルデータストアのクラスを促進する運動です。 ここ数年、NoSQL ソリューションの人気が高まっており、Facebook、LinkedIn、Twitter などの非常によく使用される主要なサイトやサービスは、スケーラビリティとアジリティを実現するために NoSQL ソリューションを広く使用することが知られています。

基本的に、NoSQL ソリューションは従来の RDBMS (リレーショナルデータベース管理システム) とは異なり、SQL をクエリー言語として使用せず、一般に ACID のようなトランザクション動作やリレーショナルデータを提供しません。代わりに、それらは柔軟なデータ構造とスキーマ (つまり、固定スキーマを持つデータベーステーブルの従来の概念が削除されたもの)、コモディティハードウェアでの極端なスケーラビリティ、および超高速処理の概念に基づいて設計されています。

MongoDB は非常に人気のある NoSQL ソリューションであり、camel-mongodb コンポーネントは Camel を MongoDB と統合し、MongoDB コレクションをプロデューサー (コレクションに対して操作を実行する) とコンシューマー (MongoDB コレクションからドキュメントを消費する) の両方として操作できるようにします。

MongoDB は、ドキュメント (オフィスドキュメントではなく、JSON/BSON で定義された階層データ) とコレクションの概念を中心にデプロイメントしています。このコンポーネントページは、それらに精通していることを前提としています。それ以外の場合は、http://www.mongodb.org/ にアクセスしてください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

216.1. URI 形式

```
mongodb:connectionBean?
database=databaseName&collection=collectionName&operation=operationName[&moreOptions...]
```

216.2. MONGODB オプション

MongoDB コンポーネントにはオプションがありません。

MongoDB エンドポイントは、URI 構文を使用して設定されます。

```
mongodb:connectionBean
```

パスおよびクエリーパラメーターを使用します。

216.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionBean	**必須** 使用する com.mongodb.Mongo の名前。		String

216.2.2. クエリーパラメーター(23 個のパラメーター):

名前	説明	デフォルト	タイプ
collection (Common)	このエンドポイントにバインドする MongoDB コレクションの名前を設定します		String
collectionIndex (Common)	コレクションのインデックスを設定します (JSON FORMAT : field1 : order1、 field2 : order2)		String
createCollection (Common)	コレクションが存在しない場合は、初期化中にコレクションを作成します。デフォルトは true です。	true	boolean
database (Common)	ターゲットに設定する MongoDB データベースの名前を設定します		String
operation (common)	このエンドポイントが MongoDB に対して実行する操作を設定します。可能な値については、MongoDbOperation を参照してください。		MongoDbOperation
outputType (common)	プロデューサーの出力を選択したタイプ (DBObjectList DBObject または DBCursor) に変換します。DBObjectList または DBCursor は、findAll および aggregate に適用されます。DBObject は、他のすべての操作に適用されます。		MongoDbOutputType
writeConcern (Common)	標準のものを使用して、MongoDB での書き込み操作の WriteConcern を設定します。リンクの WriteConcernvalueOf (String) メソッドを呼び出すことによって、WriteConcern クラスのフィールドから解決されます。	ACKNOWLEDGED	WriteConcern
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cursorRegenerationDelay (advanced)	MongoDB の Tailable カーソルは、新しいデータが到着するまでブロックされます。新しいデータが挿入されない場合、しばらくするとカーソルが自動的に解放され、MongoDB サーバーによって閉じられます。クライアントは、必要に応じてカーソルを再生成する必要があります。この値は、新しいカーソルのフェッチを試行するまでの待機時間と、試行が失敗した場合に次の試行が行われるまでの時間を指定します。デフォルト値は 1000ms です。	1000	long
dynamicity (advanced)	このエンドポイントが受信 Exchange プロパティーからターゲットデータベースとコレクションを動的に解決しようとするかどうかを設定します。それ以外の場合は静的なエンドポイント URI で指定されたデータベースとコレクションを実行時にオーバーライドするために使用できます。パフォーマンスを向上させるために、デフォルトでは無効になっています。有効にすると、パフォーマンスへの影響は最小限に抑えられます。	false	boolean
readPreference (advanced)	Mongo 接続で MongoDB ReadPreference を設定します。接続で直接設定された読み取り設定は、この設定によって上書きされます。リンク ReadPreferencevalueOf(String) ユーティリティーメソッドを使用して、渡された readPreference 値を解決します。可能な値の例として、nearest、primary、secondary などがあります。		ReadPreference
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
writeResultAsHeader (advanced)	書き込み操作では、OUT メッセージのボディとして WriteResult を返す代わりに、IN メッセージを OUT に転送し、WriteResult をヘッダーとして添付するかどうかを決定します。	false	boolean

名前	説明	デフォルト	タイプ
persistentId (tail)	1 つのテールトラッキングコレクションで、複数のテーラブルコンシューマー用に多数のトラッカーをホストできます。それらを分離しておくために、各トラッカーには独自の固有の persistentId が必要です。		String
persistentTailTracking (tail)	永続的な tail トラッキングを有効にします。これは、システムの再起動時に最後に消費されたメッセージを追跡するメカニズムです。次にシステムが起動すると、エンドポイントは最後にレコードを一気に読み込むのを停止した地点からカーソルを回復します。	false	boolean
persistRecords (tail)	テールトラッキングデータが MongoDB に永続化されるまでの、テールレコードの数を設定します。	-1	int
tailTrackCollection (tail)	テールトラッキング情報が保持されるコレクション。指定しないと、リンク MongoDbTailTrackingConfigDEFAULT_COLLECTION がデフォルトで使用されます。		文字列
tailTrackDb (tail)	テールトラッキングメカニズムが保持されるデータベースを示します。指定しない場合、現在のデータベースがデフォルトで選択されます。動的性は有効になっていても考慮されません。つまり、テールトラッキングデータベースは、エンドポイントの初期化を過ぎても変化しません。		String
tailTrackField (tail)	最後に追跡された値が配置されるフィールド。指定しない場合、リンク MongoDbTailTrackingConfigDEFAULT_FIELD がデフォルトで使用されます。		文字列
tailTrackIncreasingField (tail)	増加する性質の着信レコードの相関フィールドであり、生成されるたびに tail カーソルを配置するために使用されます。カーソルは、次のタイプのクエリーで (再) 作成されます: tailTrackIncreasingField lastValue (永続的なテールトラッキングから回復される可能性があります)。整数、日付、文字列などの型にすることができます。注: 現時点ではドット表記がサポートされていないため、フィールドはドキュメントの最上位にある必要があります。		String
tailTrackingStrategy (tail)	増加するフィールド値を展開し、テールカーソルを配置するクエリーを作成するために使用するストラテジーを設定します。	LITERAL	MongoDBTailTracking Enum

216.3. SPRING XML でのデータベースの設定

次の Spring XML は、MongoDB インスタンスへの接続を定義する Bean を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="mongoBean" class="com.mongodb.Mongo">
        <constructor-arg name="host" value="${mongodb.host}" />
        <constructor-arg name="port" value="${mongodb.port}" />
    </bean>
</beans>
```

216.4. サンプルルート

Spring XML で定義された次のルートは、コレクションに対して操作 **dbStats** を実行します。

指定されたコレクションの DB 統計を取得する

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb:mongoBean?
database=${mongodb.database}&amp;collection=${mongodb.collection}&amp;operation=getDbStats"
/>
  <to uri="direct:result" />
</route>
```

216.5. MONGODB 操作 - プロデューサーエンドポイント

216.5.1. クエリー操作

216.5.1.1. findById

この操作は、_id フィールドが IN メッセージ本文の内容と一致するコレクションから 1 つの要素のみを取得します。着信オブジェクトは、BSON 型と同等のものであれば何でもかまいません。http://bsonspec.org//specification[http://bsonspec.org//specification] および http://www.mongodb.org/display/DOCS/Java+Types を参照してください。

```
from("direct:findById")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findById")
    .to("mock:resultFindById");
```

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドフィルターの指定をサポートします。オプションのパラメーターの指定 を参照してください。

216.5.1.2. findOneByQuery

この操作を使用して、MongoDB クエリーに一致するコレクションから要素を 1 つだけ取得します。**クエリーオブジェクトは、IN メッセージボディーから抽出されます**。つまり、**DBObject** タイプであるか、**DBObject** に変換可能である必要があります。JSON 文字列または Hashmap にすることができます。詳細については、#型変換 を参照してください。

クエリーのない例 (コレクションの任意のオブジェクトを返します):

```
from("direct:findOneByQuery")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

クエリーのある例 (一致する結果を 1 つ返します):

```
from("direct:findOneByQuery")
    .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドフィルターおよび/または並べ替え句の指定をサポートします。オプションのパラメーターの指定 を参照してください。

216.5.1.3. findAll

findAll オペレーションは、クエリーに一致するすべてのドキュメントを返すか、またはまったく一致しないドキュメントを返します。この場合、コレクションに含まれるすべてのドキュメントが返されます。**クエリーオブジェクトは、IN メッセージボディーから抽出されます**。つまり、**DBObject** タイプであるか、**DBObject** に変換可能である必要があります。JSON 文字列または Hashmap にすることができます。詳細については、#型変換 を参照してください。

クエリーのない例 (コレクション内のすべてのオブジェクトを返します):

```
from("direct:findAll")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

クエリーのある例 (一致するすべての結果を返します):

```
from("direct:findAll")
    .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

ページングと効率的な検索は、次のヘッダーを介してサポートされています。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbNumToSkip	**MongoDbConstants.NUM_TO_SKIP**	カーソルの先頭にある指定された数の要素を破棄します。	int/Integer
CamelMongoDbLimit	**MongoDbConstants.LIMIT**	返される要素の数を制限します。	int/Integer
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	1 つのバッチで返される要素の数を制限します。通常、カーソルは結果オブジェクトのバッチをフェッチし、ローカルに保存します。batchSize が正の場合、取得されたオブジェクトの各	int/Integer

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
		バッチインスタンスを最適化し、データ転送を制限するように調整できます。batchSize が負の場合、最大バッチサイズ制限 (通常は 4MB) 内に収まる数のオブジェクトが返され、カーソルが閉じられます。たとえば、batchSize が -10 の場合、サーバーは最大 10 個のドキュメントを 4MB に収まる数だけ返し、カーソルを閉じます。この機能	

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
		は、ドキュメントが最大サイズ内に収まる必要があるという点で limit() とは異なり、サーバー側でカーソルを閉じる要求を送信する必要がないことに注意してください。バッチサイズは、カーソルが反復された後でも変更できます。その場合、設定は次のバッチ取得に適用されます。	

上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして outputType=DBCursor (Camel 2.16+) を含めることで、サーバーから返されたドキュメントをルートにストリーミングすることもできます。これにより、Mongo シェル内で findAll() を実行しているかのように、Mongo ドライバーから Exchange に DBCursor が渡され、ルートが結果を反復できるようになります。デフォルトでは、このオプションを指定しないと、このコンポーネントはドキュメントをドライバーのカーソルから List にロードし、これをルートに返します。これにより、多数のメモリー内オブジェクトが発生する可能性があります。DBCursor では、一致したドキュメントの数を問い合わせないでください。詳細については、MongoDB ドキュメントサイトを参照してください。

オプション outputType=DBCursor およびバッチサイズの例:

```
from("direct:findAll")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
    .to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll&outputType=DBCursor")
    .to("mock:resultFindAll");
```

ページングを使用している場合、**findAll** 操作は次の OUT ヘッダーも返し、結果ページを反復処理できるようにします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	データのタイプ
CamelMongoDbResultTotalSize	**MongoDbConstants.RESULT_TOTAL_SIZE**	クエリーに一致するオブジェクトの数。これは、制限/スキップを考慮していません。	int/Integer
CamelMongoDbResultPageSize	**MongoDbConstants.RESULT_PAGE_SIZE**	クエリーに一致するオブジェクトの数。これは、制限/スキップを考慮していません。	int/Integer

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドフィルターおよび/または並べ替え句の指定をサポートします。オプションのパラメーターの指定 を参照してください。

216.5.1.4. count

コレクション内のオブジェクトの総数を返し、OUT メッセージ本文として Long を返します。
次の例では、dynamicCollectionName コレクション内のレコード数をカウントします。動的性が有効になっていることに注意してください。その結果、操作は "notableScientists" コレクションではなく、"dynamicCollectionName" コレクションに対して実行されます。

```
// from("direct:count").to("mongodb:myDb?
database=tickets&collection=flights&operation=count&dynamicity=true");
Long result = template.requestBodyAndHeader("direct:count", "irrelevantBody",
MongoDbConstants.COLLECTION, "dynamicCollectionName");
assertTrue("Result is not of type Long", result instanceof Long);
```

Camel 2.14 以降では、メッセージボディーで **com.mongodb.DBObject** オブジェクトをクエリーとして提供できます。操作は、この条件に一致するドキュメントの量を返します。

```
DBObject query = ...
Long count = template.requestBodyAndHeader("direct:count", query,
MongoDbConstants.COLLECTION, "dynamicCollectionName");
```

216.5.1.5. フィールドフィルターの指定 (プロジェクション)

デフォルトでは、クエリー操作は、一致するオブジェクト全体を (すべてのフィールドとともに) 返します。ドキュメントが大きく、フィールドのサブセットのみを取得する必要がある場合は、関連する **DBObject** (または JSON 文字列、マップなどの **DBObject** に変換可能な型) を設定するだけで、すべてのクエリー操作でフィールドフィルターを指定できます。) **CamelMongoDbFieldsFilter** ヘッダーの定数ショートカット: **MongoDbConstants.FIELDS_FILTER**。

以下は、MongoDB の BasicDBObjectBuilder を使用して DBObject の作成を簡素化する例です。**_id** と **boringField** を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll")
DBObject fieldFilter = BasicDBObjectBuilder.start().add("_id", 0).add("boringField", 0).get();
Object result = template.requestBodyAndHeader("direct:findAll", (Object) null,
MongoDbConstants.FIELDS_FILTER, fieldFilter);
```

216.5.1.6. ソート句の指定

特定のフィールドによるソートに基づいて、コレクションから最小/最大レコードを取得する必要があることがよくあります。Mongo では、操作は次のような構文を使用して実行されます。

```
db.collection.find().sort({_id: -1}).limit(1)
// or
db.collection.findOne({$query:{},$orderby:{_id:-1}})
```

Camel ルートでは、SORT_BY ヘッダーを findOneByQuery 操作で使用して、同じ結果を得ることができます。FIELDS_FILTER ヘッダーも指定されている場合、操作は、別のコンポーネント (たとえば、パラメーター化された MyBatis SELECT クエリー) に直接渡すことができる単一のフィールド/値のペアを返します。この例では、コレクションから一時的に最新のドキュメントをフェッチし、結果を **documentTimestamp** フィールドに基づいて 1 つのフィールドに削減する方法を示します。

```
.from("direct:someTriggeringEvent")
.setHeader(MongoDbConstants.SORT_BY).constant("{\"documentTimestamp\": -1}")
.setHeader(MongoDbConstants.FIELDS_FILTER).constant("{\"documentTimestamp\": 1}")
.setBody().constant("{}")
.to("mongodb:myDb?database=local&collection=myDemoCollection&operation=findOneByQuery")
.to("direct:aMyBatisParameterizedSelect")
;
```

216.5.2. 操作の作成/更新

216.5.2.1. insert

IN メッセージ本文から取得した新しいオブジェクトを MongoDB コレクションに挿入します。**DBObject** または **List** に変換するために型変換が試行されます。
シングル挿入とマルチ挿入の 2 つのモードがサポートされています。複数の挿入の場合、エンドポイントは、DBObject である、または **DBObject** に変換できる限り、任意のタイプのオブジェクトのリスト、配列、またはコレクションを期待します。すべてのオブジェクトが一度に挿入されます。エンドポイントは、入力に応じて、呼び出すバックエンド操作 (単一または複数の挿入) をインテリジェントに決定します。

例:

```
from("direct:insert")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=insert");
```

オペレーションは WriteResult を返します。**WriteConcern** または **invokeGetLastError** オプションの値に応じて、**getLastError()** がすでに呼び出されているかどうかが決まります。書き込み操作の最終的な結果にアクセスする場合は、**WriteResult** で **getLastError()** または **getCachedLastError()** を呼び出して **CommandResult** を取得する必要があります。次に、**CommandResult.ok()**、**CommandResult.getErrorMessage()** および/または **CommandResult.getException()** を呼び出して結果を確認できます。

新しいオブジェクトの **_id** はコレクション内で一意である必要があることに注意してください。値を指定しない場合、MongoDB が自動的に値を生成します。ただし、指定しても一意でない場合、挿入操作は失敗します (Camel が気付くには、invokeGetLastError を有効にするか、書き込み結果を待機する WriteConcern を設定する必要があります)。

これはコンポーネントの制限ではありませんが、MongoDB でより高いスループットを実現する方法です。カスタム **_id** を使用している場合は、アプリケーションレベルで一意であることを確認する必要があります (これも良い方法です)。

Camel 2.15 以降: 挿入されたレコードの OID は、 **CamelMongoOid** キー (**MongoDbConstants.OID** 定数) の下のメッセージヘッダーに格納されます。保存される値は、単一の挿入の場合は **org.bson.types.ObjectId**、複数のレコードが挿入された場合は **java.util.List<org.bson.types.ObjectId>** です。

216.5.2.2. save

保存操作は **upsert** (UPdate、inSERT) 操作と同等で、レコードが更新され、レコードが存在しない場合は挿入されます。これらはすべて 1 つのアトミック操作で行われます。MongoDB は _id フィールドに基づいてマッチングを実行します。

更新の場合、オブジェクトは完全に置き換えられ、MongoDB の $modifiers の使用は許可されないことに注意してください。したがって、オブジェクトがすでに存在する場合にそのオブジェクトを操作する場合は、次の 2 つのオプションがあります。

1. 最初にオブジェクト全体とそのすべてのフィールドを取得するためのクエリーを実行し (効率的でない場合があります)、Camel 内で変更してから保存します。

2. $modifiers で更新操作を使用すると、代わりにサーバー側で更新が実行されます。upsert フラグを有効にできます。この場合、挿入が必要な場合、MongoDB は $modifiers をフィルタークエリーオブジェクトに適用し、結果を挿入します。

以下に例を示します。

```
from("direct:insert")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=save");
```

216.5.2.3. update

コレクションの 1 つまたは複数のレコードを更新します。正確に 2 つの要素を含む IN メッセージ本文として List<DBObject> が必要です。

- 要素 1 (インデックス 0) ⇒ フィルタークエリー ⇒ 通常のクエリーオブジェクトと同じように、影響を受けるオブジェクトを決定します
- 要素 2 (インデックス 1) ⇒ 更新ルール ⇒ 一致したオブジェクトがどのように更新されるか。MongoDB からのすべての 修飾子操作 がサポートされています。

注記

マルチアップデート。デフォルトでは、MongoDB は、複数のオブジェクトがフィルタークエリーに一致する場合でも、1 つのオブジェクトのみを更新します。一致する **すべての** レコードを更新するように MongoDB に指示するには、**CamelMongoDbMultiUpdate** IN メッセージヘッダーを **true** に設定します。

キー **CamelMongoDbRecordsAffected** を持つヘッダーが返されます (**MongoDbConstants.RECORDS_AFFECTED** 定数) 更新されたレコードの数 (**WriteResult.getN()** からコピーされます)。

次の IN メッセージヘッダーをサポートします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbMultiUpdate	**MongoDbConstants.MULTIUPDATE**	一致するすべてのオブジェクトに更新を適用する必要がある場合。http://www.mongodb.org/display/DOCS/Atomic+Operations を参照してください。	boolean/Boolean
CamelMongoDbUpsert	**MongoDbConstants.UPSERT**	存在しない場合にデータベースが要素を作成する必要があるかどうか	boolean/Boolean

たとえば、次の例では、"scientist" フィールドの値を "Darwin" に設定することで、filterField フィールドが true に等しい **すべての** レコードを更新します。

```
// route: from("direct:update").to("mongodb:myDb?database=science&collection=notableScientists&operation=update");
DBObject filterField = new BasicDBObject("filterField", true);
DBObject updateObj = new BasicDBObject("$set", new BasicDBObject("scientist", "Darwin"));
Object result = template.requestBodyAndHeader("direct:update", new Object[] {filterField, updateObj},
MongoDbConstants.MULTIUPDATE, true);
```

216.5.3. 操作の削除

216.5.3.1. remove

コレクションから一致するレコードを削除します。IN メッセージ本文は、削除フィルタークエリーとして機能し、**DBObject** タイプまたはそれに変換可能なタイプであることが期待されます。
次の例では、科学データベースの notableScientists コレクションで、フィールド 'conditionField' が true に等しいすべてのオブジェクトを削除します。

```
// route: from("direct:remove").to("mongodb:myDb?
database=science&collection=notableScientists&operation=remove");
DBObject conditionField = new BasicDBObject("conditionField", true);
Object result = template.requestBody("direct:remove", conditionField);
```

キー **CamelMongoDbRecordsAffected** を持つヘッダーが返されます (**MongoDbConstants.RECORDS_AFFECTED** 定数)。タイプは **int** で、削除された (**WriteResult.getN()** からコピーされた) レコードの数を含みます。

216.5.4. 一括書き込み操作

216.5.4.1. bulkWrite

Camel 2.21 以降で利用可能

実行順序を制御して書き込み操作を一括で実行します。挿入、更新、および削除操作のコマンドを含む IN メッセージ本文として **List<WriteModel<DBObject>>** が必要です。

次の例では、新しい科学者 "Pierre Curie" を挿入し、"scientist" フィールドの値を "Marie Curie" に設定して ID "5" のレコードを更新し、ID "3" のレコードを削除します。

```
// route: from("direct:bulkWrite").to("mongodb:myDb?
database=science&collection=notableScientists&operation=bulkWrite");
List<WriteModel<DBObject>> bulkOperations = Arrays.asList(
        new InsertOneModel<>(new BasicDBObject("scientist", "Pierre Curie")),
        new UpdateOneModel<>(new BasicDBObject("_id", "5"),
                new BasicDBObject("$set", new BasicDBObject("scientist", "Marie Curie"))),
        new DeleteOneModel<>(new BasicDBObject("_id", "3")));

BulkWriteResult result = template.requestBody("direct:bulkWrite", bulkOperations,
BulkWriteResult.class);
```

デフォルトでは、操作は順番に実行され、最初の書き込みエラーで中断され、リスト内の残りの書き込み操作は処理されません。リスト内の残りの書き込み操作の処理を続行するように MongoDB に指示するには、**CamelMongoDbBulkOrdered** IN メッセージヘッダーを **false** に設定します。順序付けされていない操作は並行して実行され、この動作は保証されません。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbBulkOrdered	**MongoDbConstants.BULK_ORDERED**	順序付きまたは順序なしの操作実行を実行します。デフォルトは true です。	boolean/Boolean

216.5.5. その他の操作

216.5.5.1. aggregate

Camel 2.14 から利用可能

本文に含まれる特定のパイプラインを使用して集計を実行します。**集約は、長く重い操作になる可能性があります。注意して使用してください。**

```
// route: from("direct:aggregate").to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate");
from("direct:aggregate")
    .setBody().constant("[{ $match : {$or : [{\"scientist\" : \"Darwin\"},{\"scientist\" : \"Einstein\"}]}},{
$group: { _id: \"$scientist\", count: { $sum: 1 }} } ]")
    .to("mongodb:myDb?database=science&collection=notableScientists&operation=aggregate")
    .to("mock:resultAggregate");
```

次の IN メッセージヘッダーをサポートします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	バッチごとに返すドキュメントの数を設定します。	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	集約パイプラインステージを有効にして、データを一時ファイルに書き込みます。	boolean/Boolean

outputType=DBCursor を使用すると、効率的な取得がサポートされます。

上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして outputType=DBCursor (Camel 2.21+) を含めることで、サーバーから返されたドキュメントをルートにストリーミングすることもできます。これにより、Mongo シェル内で aggregate() を実行しているかのように、Mongo ドライバーから Exchange に DBCursor が渡され、ルートが結果を反復できるようになります。デフォルトでは、このオプションを指定しないと、このコンポーネントはドキュメントをドライバーのカーソルから List にロードし、これをルートに返します。これにより、多数のメモリー内オブジェクトが発生する可能性があります。DBCursor では、一致したドキュメントの数を問い合わせないでください。詳細については、MongoDB ドキュメントサイトを参照してください。

オプション outputType=DBCursor およびバッチサイズの例:

```
// route: from("direct:aggregate").to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate");
from("direct:aggregate")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setBody().constant("[{ $match : {$or : [{\"scientist\" : \"Darwin\"},{\"scientist\" : \"Einstein\"}]}},{
$group: { _id: \"$scientist\", count: { $sum: 1 }} } ]")
    .to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=DBCursor")
    .to("mock:resultAggregate");
```

216.5.5.2. getDbStats

MongoDB シェルで **db.stats()** コマンドを実行するのと同じです。これは、データベースに関する有用な統計値を表示します。
以下に例を示します。

```
> db.stats();
{
    "db" : "test",
    "collections" : 7,
    "objects" : 719,
    "avgObjSize" : 59.73296244784423,
    "dataSize" : 42948,
    "storageSize" : 1000058880,
    "numExtents" : 9,
    "indexes" : 4,
    "indexSize" : 32704,
    "fileSize" : 1275068416,
    "nsSizeMB" : 16,
    "ok" : 1
}
```

使用例:

```
// from("direct:getDbStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getDbStats");
Object result = template.requestBody("direct:getDbStats", "irrelevantBody");
assertTrue("Result is not of type DBObject", result instanceof DBObject);
```

この操作は、シェルに表示されるものと同様のデータ構造を、OUT メッセージ本文の **DBObject** の形式で返します。

216.5.5.3. getColStats

コレクションに関する有用な統計値を表示する、MongoDB シェルで **db.collection.stats()** コマンドを実行するのと同じです。
以下に例を示します。

```
> db.camelTest.stats();
{
    "ns" : "test.camelTest",
    "count" : 100,
    "size" : 5792,
    "avgObjSize" : 57.92,
    "storageSize" : 20480,
    "numExtents" : 2,
    "nindexes" : 1,
    "lastExtentSize" : 16384,
    "paddingFactor" : 1,
    "flags" : 1,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
```

```
    },
    "ok" : 1
}
```

使用例:

```
// from("direct:getColStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getColStats");
Object result = template.requestBody("direct:getColStats", "irrelevantBody");
assertTrue("Result is not of type DBObject", result instanceof DBObject);
```

この操作は、シェルに表示されるものと同様のデータ構造を、OUT メッセージ本文の **DBObject** の形式で返します。

216.5.5.4. command

Camel 2.15 以降で利用可能

データベースで本文をコマンドとして実行します。ホスト情報、レプリケーション、またはシャーディングステータスを取得するなどの管理操作に役立ちます。

コレクションパラメーターは、この操作では使用されません。

```
// route: from("command").to("mongodb:myDb?database=science&operation=command");
DBObject commandBody = new BasicDBObject("hostInfo", "1");
Object result = template.requestBody("direct:command", commandBody);
```

216.5.6. 動的操作

エクスチェンジは、**MongoDbConstants.OPERATION_HEADER** 定数で定義された **CamelMongoDbOperation** ヘッダーを設定することにより、エンドポイントの固定操作をオーバーライドできます。
サポートされる値は、MongoDbOperation 列挙によって決定され、エンドポイント URI の **operation** パラメーターで受け入れられる値と一致します。

以下に例を示します。

```
// from("direct:insert").to("mongodb:myDb?database=flights&collection=tickets&operation=insert");
Object result = template.requestBodyAndHeader("direct:insert", "irrelevantBody",
MongoDbConstants.OPERATION_HEADER, "count");
assertTrue("Result is not of type Long", result instanceof Long);
```

216.6. TAILABLE カーソルコンシューマー

MongoDB は、*nix システムの **tail -f** コマンドのようにカーソルを開いたままにすることで、コレクションから進行中のデータを瞬時に消費するメカニズムを提供します。このメカニズムは、クライアントがスケジュールされた間隔で ping を返して新しいデータを取得するのではなく、新しいデータが利用可能になったときにサーバーがクライアントにプッシュするため、スケジュールされたポーリングよりもはるかに効率的です。また、冗長なネットワークトラフィックも削減されます。

tailable カーソルを使用するための必要条件は 1 つだけです。つまり、コレクションは " 上限付きコレクション " である必要があります。これは、N 個のオブジェクトのみを保持することを意味し、制限に達すると、MongoDB は最初に挿入されたのと同じ順序で古いオブジェクトをフラッシュします。詳細

は、http://www.mongodb.org/display/DOCS/Tailable+Cursors を参照してください。

Camel MongoDB コンポーネントは、tailable カーソルコンシューマーを実装しているため、この機能を Camel ルートで使用できるようになります。新しいオブジェクトが挿入されると、MongoDB はそれらを DBObjects として自然な順序で tailable カーソルコンシューマーにプッシュします。tailable カーソルコンシューマーはそれらをエクスチェンジに変換し、ルートロジックをトリガーします。

216.7. TAILABLE カーソルコンシューマーの仕組み

カーソルを tailable カーソルに変えるには、最初にカーソルを生成するときに、いくつかの特別なフラグを MongoDB に通知する必要があります。作成されると、カーソルは開いたままになり、新しいデータが到着するまで **DBCursor.next()** メソッドを呼び出すとブロックされます。ただし、MongoDB サーバーは、不確定な期間が経過しても新しいデータが表示されない場合、カーソルを強制終了する権利を留保します。新しいデータを引き続き使用する場合は、カーソルを再生成する必要があります。そのためには、中断した位置を覚えておく必要があります。そうしないと、もう一度最初から消費し始めます。

Camel MongoDB テーラブルカーソルコンシューマーは、これらすべてのタスクを処理します。タイムスタンプ、シーケンシャル ID など、再生成されるたびにカーソルを配置するマーカーとして機能する、増加する性質のデータ内のフィールドにキーを提供するだけで済みます。MongoDB でサポートされている任意のデータ型にすることができます。日付、文字列、および整数がうまく機能することがわかっています。このコンポーネントのコンテキストでは、このメカニズムをテールトラッキングと呼びます。

コンシューマーはこのフィールドの最後の値を記憶し、カーソルが再生成されるたびに、次のようなフィルターを使用してクエリーを実行します。**increasingField > lastValue** のようなフィルタでクエリを実行し、未読のデータのみが消費されるようにします。

増加フィールドの設定: エンドポイント URI の **tailTrackingIncreasingField** オプションで増加フィールドのキーを設定します。Camel 2.10 では、このフィールドのネストされたナビゲーションがまだサポートされていないため、データの最上位フィールドである必要があります。つまり、"timestamp" フィールドは問題ありませんが、"nested.timestamp" は機能しません。ネストされた増加するフィールドのサポートが必要な場合は、Camel JIRA でチケットを開いてください。

カーソル再生成の遅延: 注意すべきことの 1 つは、初期化時に新しいデータがまだ利用できない場合、MongoDB はカーソルを即座に強制終了することです。この場合、サーバーに負荷をかけたくないので、**cursorRegenerationDelay** オプションが導入されています (デフォルト値は 1000 ミリ秒です)。これは、ニーズに合わせて変更できます。

以下に例を示します。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime")
    .id("tailableCursorConsumer1")
    .autoStartup(false)
    .to("mock:test");
```

上記のルートは、departureTime を増加フィールドとして使用し、デフォルトの再生成カーソル遅延を 1000 ミリ秒にして、flights.cancellations キャップコレクションから消費します。

216.8. 永続的なテールトラッキング

標準のテールトラッキングは揮発性であり、最後の値はメモリーにのみ保持されます。ただし、実際には Camel コンテナーを時々再起動する必要がありますが、最後の値は失われ、tailable カーソルコンシューマーは再び先頭から消費を開始し、ルートに重複レコードを送信する可能性が非常に高くなりま

す。

この状況を克服するために、**永続的なテール追跡** 機能を有効にして、MongoDB データベース内の特別なコレクションで最後に消費された増加値を追跡することもできます。コンシューマーが再び初期化されると、最後に追跡された値が復元され、何も起こらなかったかのように続行されます。

最後に読み取られた値は、カーソルが再生成されるたびとコンシューマーがシャットダウンするときの 2 つの場合に保持されます。需要があれば、堅牢性を高めるために、将来的には定期的な間隔 (5 秒ごとにフラッシュ) で永続化することも検討する可能性があります。この機能をリクエストするには、Camel JIRA でチケットを開いてください。

216.9. 永続的なテールトラッキングを有効にする

この機能を有効にするには、エンドポイント URI で少なくとも次のオプションを設定します。

- **persistentTailTracking** オプションを **true** に設定
- このコンシューマーの一意の識別子に **persistentId** オプションを追加して、同じコレクションを多くのコンシューマーで再利用できるようにします

さらに、**tailTrackDb**、**tailTrackCollection**、および **tailTrackField** オプションを設定して、ランタイム情報が保存される場所をカスタマイズできます。各オプションの説明については、このページの上部にあるエンドポイントオプションの表を参照してください。

たとえば、次のルートは、departureTime を増加フィールドとして使用し、デフォルトの再生成カーソル遅延を 1000 ミリ秒に設定して、永続的なテールトラッキングをオンにし、cancellationsTracker の下で永続化して、flights.cancellations キャップコレクションから消費します。flights.camelTailTracking の id で、最後に処理された値を lastTrackingValue フィールドに格納します (**camelTailTracking** と **lastTrackingValue** はデフォルトです)。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker")
  .id("tailableCursorConsumer2")
  .autoStartup(false)
  .to("mock:test");
```

以下は、上記と同じ別の例ですが、永続的なテールトラッキングランタイム情報が trackers.camelTrackers コレクションの lastProcessedDepartureTime フィールドに格納されます。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker&tailTrackDb=trackers&tailTrackCollection=camelTrackers" +
    "&tailTrackField=lastProcessedDepartureTime")
  .id("tailableCursorConsumer3")
  .autoStartup(false)
  .to("mock:test");
```

216.10. OPLOG テールトラッキング

oplog コレクション追跡機能により、MongoDB にトリガーのような機能を実装できます。このコレクションをアクティブにするには、最初にレプリカセットをアクティブにする必要があります。このト

ピックの詳細については、https://docs.mongodb.com/manual/tutorial/deploy-replica-set/ を確認してください。

以下に、コンポーネントを使用して **oplog** コレクションを追跡する方法を示す Java DSL ベースのルートの例を示します。この特定のケースでは、データベース **optlog_test** 内のコレクション **customers** に影響を与えるイベントをフィルタリングしています。**tailTrackIncreasingField** はタイムスタンプフィールド ('ts') であることに注意してください。これは、**TIMESTAMP** 値で **tailTrackingStrategy** パラメーターを使用する必要があることを意味します。

```
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import org.apache.camel.Exchange;
import org.apache.camel.Message;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mongodb.MongoDBTailTrackingEnum;
import org.apache.camel.main.Main;

import java.io.InputStream;

/**
 * For this to work you need to turn on the replica set
 * <p>
 * Commands to create a replica set:
 * <p>
 * rs.initiate( {
 * _id : "rs0",
 * members: [ { _id : 0, host : "localhost:27017" } ]
 * })
 */
public class MongoDbTracker {

    private final String database;

    private final String collection;

    private final String increasingField;

    private MongoDBTailTrackingEnum trackingStrategy;

    private int persistRecords = -1;

    private boolean persistenTailTracking;

    public MongoDbTracker(String database, String collection, String increasingField) {
        this.database = database;
        this.collection = collection;
        this.increasingField = increasingField;
    }

    public static void main(String[] args) throws Exception {
        final MongoDbTracker mongoDbTracker = new MongoDbTracker("local", "oplog.rs", "ts");
        mongoDbTracker.setTrackingStrategy(MongoDBTailTrackingEnum.TIMESTAMP);
        mongoDbTracker.setPersistRecords(5);
        mongoDbTracker.setPersistenTailTracking(true);
        mongoDbTracker.startRouter();
```

```
        // run until you terminate the JVM
        System.out.println("Starting Camel. Use ctrl + c to terminate the JVM.\n");

    }

    public void setTrackingStrategy(MongoDBTailTrackingEnum trackingStrategy) {
        this.trackingStrategy = trackingStrategy;
    }

    public void setPersistRecords(int persistRecords) {
        this.persistRecords = persistRecords;
    }

    public void setPersistenTailTracking(boolean persistenTailTracking) {
        this.persistenTailTracking = persistenTailTracking;
    }

    void startRouter() throws Exception {
        // create a Main instance
        Main main = new Main();
        main.bind(MongoConstants.CONN_NAME, new MongoClient("localhost", 27017));
        main.addRouteBuilder(new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                getContext().getTypeConverterRegistry().addTypeConverter(InputStream.class,
BasicDBObject.class,
                        new MongoToInputStreamConverter());
                from("mongodb://" + MongoConstants.CONN_NAME + "?database=" + database
                        + "&collection=" + collection
                        + "&persistentTailTracking=" + persistenTailTracking
                        + "&persistentId=trackerName" + "&tailTrackDb=local"
                        + "&tailTrackCollection=talendTailTracking"
                        + "&tailTrackField=lastTrackingValue"
                        + "&tailTrackIncreasingField=" + increasingField
                        + "&tailTrackingStrategy=" + trackingStrategy.toString()
                        + "&persistRecords=" + persistRecords
                        + "&cursorRegenerationDelay=1000")
                        .filter().jsonpath("$[?(@.ns=='optlog_test.customers')]")
                        .id("logger")
                        .to("log:logger?level=WARN")
                        .process(new Processor() {
                            public void process(Exchange exchange) throws Exception {
                                Message message = exchange.getIn();
                                System.out.println(message.getBody().toString());
                                exchange.getOut().setBody(message.getBody().toString());
                            }
                        });
            }
        });
        main.run();
    }
}
```

216.11. 型変換

camel-mongodb コンポーネントに含まれる **MongoDbBasicConverters** 型コンバーターは、次の変換を提供します。

名前	タイプから	入力し	どのように変更を加えればよいですか？
fromMapToDBObject	**Map**	**DBObject**	**new BasicDBObject(Map m)** コンストラクターを介して新しい **BasicDBObject** を構築します
fromBasicDBObjectToMap	**BasicDBObject**	**Map**	**BasicDBObject** はすでに **Map** を実装しています
fromStringToDBObject	**String**	**DBObject**	**com.mongodb.util.JSON.parse(String s)** を使用する
fromAnyObjectToDBObject	**Object**	**DBObject**	Jackson ライブラリー を使用してオブジェクトを **Map** に変換し、これを使用して新しい **BasicDBObject** を初期化します

この型コンバーターは自動検出されるため、手動で設定する必要はありません。

216.12. その他の参考資料

- MongoDB web サイト
- NoSQL ウィキペディアの記事
- MongoDB Java ドライバー API ドキュメント - 最新バージョン * 使用例の 単体テスト

第217章 MONGODB GRIDFS コンポーネント

Camel バージョン 2.18 以降で利用可能

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb-gridfs</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

217.1. URI 形式

```
mongodb-gridfs:connectionBean?database=databaseName&bucket=bucketName[&moreOptions...]
```

217.2. MONGODB GRIDFS OPTIONS

MongoDB GridFS コンポーネントにはオプションがありません。

MongoDB GridFS エンドポイントは、URI 構文を使用して設定されます。

```
mongodb-gridfs:connectionBean
```

パスおよびクエリーパラメーターを使用します。

217.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionBean	**必須** 使用する com.mongodb.Mongo の名前。		String

217.2.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
bucket (common)	データベース内の GridFS バケットの名前を設定します。デフォルトは fs です。	fs	String
database (Common)	**必須** MongoDB データベースの名前をターゲットに設定します		String

名前	説明	デフォルト	タイプ
readPreference (Common)	Mongo 接続で MongoDB ReadPreference を設定します。接続で直接設定された読み取り設定は、この設定によって上書きされます。リンク com.mongodb.ReadPreference#valueOf(String) ユーティリティーメソッドを使用して、渡された readPreference 値を解決します。可能な値の例として、nearest、primary、secondary などがあります。		ReadPreference
writeConcern (Common)	標準のものを使用して、MongoDB での書き込み操作の WriteConcern を設定します。リンクの WriteConcernvalueOf (String) メソッドを呼び出すことによって、WriteConcern クラスのフィールドから解決されます。		WriteConcern
writeConcernRef (common)	MongoDB での書き込み操作の WriteConcern を設定し、Bean ref をレジストリーに存在するカスタム WriteConcern に渡します。キーを渡すことで、標準の WriteConcerns を使用することもできます。リンク setWriteConcern(String) setWriteConcern メソッドを参照してください。		WriteConcern
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	コンシューマー内のポーリング間の遅延を設定します。デフォルトは 500 ミリ秒です	500	long
fileAttributeName (consumer)	QueryType が FileAttribute を使用する場合、これは使用される属性の名前を設定します。デフォルトはキャメル処理です。	camel-processed	String
initialDelay (consumer)	コンシューマーがポーリングを開始する前に initialDelay を設定します。デフォルトは 1000 ミリ秒です	1000	long
persistentTSCollection (consumer)	QueryType が永続的なタイムスタンプを使用する場合、DB 内のコレクションの名前がタイムスタンプを格納するように設定されます。	camel-timestamps	String

名前	説明	デフォルト	タイプ
persistentTSObject (consumer)	QueryType が永続的なタイムスタンプを使用する場合、これはタイムスタンプを格納するコレクション内のオブジェクトの ID です。	camel-timestamp	String
query (consumer)	GridFsConsumer でファイルを検索するために使用されるクエリーを設定するために使用される追加のクエリーパラメーター (JSON 内)		String
queryStrategy (consumer)	新しいファイルのポーリングに使用される QueryStrategy を設定します。デフォルトはタイムスタンプです	TimeStamp	QueryStrategy
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
operation (producer)	このエンドポイントが GridRS に対して実行する操作を設定します。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

217.3. SPRING XML でのデータベースの設定

次の Spring XML は、MongoDB インスタンスへの接続を定義する Bean を作成します。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="mongoBean" class="com.mongodb.Mongo">
        <constructor-arg name="host" value="${mongodb.host}" />
        <constructor-arg name="port" value="${mongodb.port}" />
    </bean>
</beans>
```

217.4. サンプルルート

Spring XML で定義された次のルートは、コレクションに対して操作 findOne を実行します。

GridFS からファイルを取得する

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb-gridfs:mongoBean?database=${mongodb.database}&amp;operation=findOne" />
  <to uri="direct:result" />
</route>
```

217.5. GRIDFS 操作 - プロデューサーエンドポイント

217.5.1. count

コレクション内のファイルの総数を返し、OUT メッセージボディーとして整数を返します。

```
// from("direct:count").to("mongodb-gridfs?database=tickets&operation=count");
Integer result = template.requestBodyAndHeader("direct:count", "irrelevantBody");
assertTrue("Result is not of type Long", result instanceof Integer);
```

ファイル名ヘッダーを提供して、そのファイル名に一致するファイルの数を提供できます。

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
Integer count = template.requestBodyAndHeaders("direct:count", query, headers);
```

217.5.2. listAll

すべてのファイル名とその ID をタブ区切りのストリームで一覧表示する Reader を返します。

```
// from("direct:listAll").to("mongodb-gridfs?database=tickets&operation=listAll");
Reader result = template.requestBodyAndHeader("direct:listAll", "irrelevantBody");

filename1.txt   1252314321
filename2.txt   2897651254
```

217.5.3. findOne

GridFS システムでファイルを検索し、本文をコンテンツの InputStream に設定します。 また、ヘッダーを持つメタデータも提供します。 受信ヘッダーの Exchange.FILE_NAME を使用して、検索するファイルを決定します。

```
// from("direct:findOne").to("mongodb-gridfs?database=tickets&operation=findOne");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
InputStream result = template.requestBodyAndHeaders("direct:findOne", "irrelevantBody", headers);
```

217.5.4. create

GridFs データベースに新しいファイルを作成します。これは、着信ヘッダーの Exchange.FILE_NAME を名前として使用し、本文の内容を (InputStream として) コンテンツとして使用します。

```
// from("direct:create").to("mongodb-gridfs?database=tickets&operation=create");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
InputStream stream = ... the data for the file ...
template.requestBodyAndHeaders("direct:create", stream, headers);
```

217.5.5. remove

GridFS データベースからファイルを削除します。

```
// from("direct:remove").to("mongodb-gridfs?database=tickets&operation=remove");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
template.requestBodyAndHeaders("direct:remove", "", headers);
```

217.6. GRIDFS CONSUMER

その他の参考資料

- MongoDB web サイト
- NoSQL ウィキペディアの記事
- MongoDB Java ドライバー API ドキュメント - 最新バージョン * 使用例の 単体テスト

第218章 MONGODB COMPONENT

Camel バージョン 2.19 以降で利用可能

注意: Camel MongoDB3 コンポーネント Mongo Driver for Java 3.4 を使用します。プレビューバージョンを探している場合は、Camel MongoDB コンポーネントを探します

ウィキペディアによると、NoSQL は、リレーショナルデータベースと ACID 保証の長い歴史を破る、大まかに定義された非リレーショナルデータストアのクラスを促進する運動です。 ここ数年、NoSQL ソリューションの人気が高まっており、Facebook、LinkedIn、Twitter などの非常によく使用される主要なサイトやサービスは、スケーラビリティとアジリティを実現するために NoSQL ソリューションを広く使用することが知られています。

基本的に、NoSQL ソリューションは従来の RDBMS (リレーショナルデータベース管理システム) とは異なり、SQL をクエリー言語として使用せず、一般に ACID のようなトランザクション動作やリレーショナルデータを提供しません。代わりに、それらは柔軟なデータ構造とスキーマ (つまり、固定スキーマを持つデータベーステーブルの従来の概念が削除されたもの)、コモディティハードウェアでの極端なスケーラビリティ、および超高速処理の概念に基づいて設計されています。

MongoDB は非常に人気のある NoSQL ソリューションであり、camel-mongodb コンポーネントは Camel を MongoDB と統合し、MongoDB コレクションをプロデューサー (コレクションに対して操作を実行する) とコンシューマー (MongoDB コレクションからドキュメントを消費する) の両方として操作できるようにします。

MongoDB は、ドキュメント (オフィスドキュメントではなく、JSON/BSON で定義された階層データ) とコレクションの概念を中心にデプロイメントしています。このコンポーネントページは、それらに精通していることを前提としています。それ以外の場合は、http://www.mongodb.org/ にアクセスしてください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb3</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

218.1. URI 形式

```
mongodb3:connectionBean?
database=databaseName&collection=collectionName&operation=operationName[&moreOptions...]
```

218.2. MONGODB オプション

MongoDB コンポーネントにはオプションがありません。

MongoDB エンドポイントは、URI 構文を使用して設定されます。

```
mongodb3:connectionBean
```

パスおよびクエリーパラメーターを使用します。

218.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionBean	**必須** 使用する com.mongodb.Mongo の名前。		文字列

218.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
collection (Common)	このエンドポイントにバインドする MongoDB コレクションの名前を設定します		String
collectionIndex (Common)	コレクションのインデックスを設定します (JSON FORMAT : field1 : order1、field2 : order2)		String
createCollection (Common)	コレクションが存在しない場合は、初期化中にコレクションを作成します。デフォルトは true です。	true	boolean
database (Common)	ターゲットに設定する MongoDB データベースの名前を設定します		String
operation (common)	このエンドポイントが MongoDB に対して実行する操作を設定します。可能な値については、MongoDbOperation を参照してください。		MongoDbOperation
outputType (common)	プロデューサの出力を選択したタイプ (DocumentList Document または MongoIterable) に変換します。DocumentList または MongoIterable は、findAll および aggregate に適用されます。ドキュメントは、他のすべての操作に適用されます。		MongoDbOutputType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cursorRegenerationDelay (advanced)	MongoDB の Tailable カーソルは、新しいデータが到着するまでブロックされます。新しいデータが挿入されない場合、しばらくするとカーソルが自動的に解放され、MongoDB サーバーによって閉じられます。クライアントは、必要に応じてカーソルを再生成する必要があります。この値は、新しいカーソルのフェッチを試行するまでの待機時間と、試行が失敗した場合に次の試行が行われるまでの時間を指定します。デフォルト値は 1000ms です。	1000	long
dynamicity (advanced)	このエンドポイントが受信 Exchange プロパティーからターゲットデータベースとコレクションを動的に解決しようとするかどうかを設定します。それ以外の場合は静的なエンドポイント URI で指定されたデータベースとコレクションを実行時にオーバーライドするために使用できます。パフォーマンスを向上させるために、デフォルトでは無効になっています。有効にすると、パフォーマンスへの影響は最小限に抑えられます。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
writeResultAsHeader (advanced)	書き込み操作では、OUT メッセージのボディとして WriteResult を返す代わりに、IN メッセージを OUT に転送し、WriteResult をヘッダーとして添付するかどうかを決定します。	false	boolean
persistentId (tail)	1 つのテールトラッキングコレクションで、複数のテーラブルコンシューマー用に多数のトラッカーをホストできます。それらを分離しておくために、各トラッカーには独自の固有の persistentId が必要です。		String
persistentTailTracking (tail)	永続的な tail トラッキングを有効にします。これは、システムの再起動時に最後に消費されたメッセージを追跡するメカニズムです。次にシステムが起動すると、エンドポイントは最後にレコードを一気に読み込むのを停止した地点からカーソルを回復します。	false	boolean
tailTrackCollection (tail)	テールトラッキング情報が保持されるコレクション。指定しないと、リンク MongoDbTailTrackingConfigDEFAULT_COLLECTION がデフォルトで使用されます。		文字列

名前	説明	デフォルト	タイプ
tailTrackDb (tail)	テールトラッキングメカニズムが保持されるデータベースを示します。指定しない場合、現在のデータベースがデフォルトで選択されます。動的性は有効になっていても考慮されません。つまり、テールトラッキングデータベースは、エンドポイントの初期化を過ぎても変化しません。		String
tailTrackField (tail)	最後に追跡された値が配置されるフィールド。指定しない場合、リンク MongoDbTailTrackingConfigDEFAULT_FIELD がデフォルトで使用されます。		文字列
tailTrackIncreasingField (tail)	増加する性質の着信レコードの相関フィールドであり、生成されるたびに tail カーソルを配置するために使用されます。カーソルは、次のタイプのクエリーで (再) 作成されます: tailTrackIncreasingField lastValue (永続的なテールトラッキングから回復される可能性があります)。整数、日付、文字列などの型にすることができます。注: 現時点ではドット表記がサポートされていないため、フィールドはドキュメントの最上位にある必要があります。		文字列

MoongoDB コンポーネントのオプションに関する注意

writeConcern **camel 2.19 で削除。**Mongo クライアントのオプション 「Spring XML でのデータベースの設定」 を参照してください。標準のものを使用して、MongoDB での書き込み操作の WriteConcern を設定します。リンクの WriteConcernvalueOf (String) メソッドを呼び出すことによって、WriteConcern クラスのフィールドから解決されます。

readPreference **camel 2.19 で削除**Mongo クライアントのオプション 「Spring XML でのデータベースの設定」 を参照してください。Mongo 接続で MongoDB ReadPreference を設定します。接続で直接設定された読み取り設定は、この設定によって上書きされます。リンク com.mongodb.ReadPreference#valueOf(String) ユーティリティーメソッドを使用して、渡された readPreference 値を解決します。可能な値の例として、nearest、primary、secondary などがあります。

218.3. SPRING XML でのデータベースの設定

次の Spring XML は、MongoDB インスタンスへの接続を定義する Bean を作成します。

mongo Java driver 3 以降、WriteConcern および readPreference オプションは動的に変更できません。それらは mongoClient オブジェクトで定義されています

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mongo="http://www.springframework.org/schema/data/mongo"
xsi:schemaLocation="http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
```

```
      http://www.springframework.org/schema/data/mongo
      http://www.springframework.org/schema/data/mongo/spring-mongo.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

  <mongo:mongo-client id="mongoBean" host="${mongo.url}" port="${mongo.port}"
credentials="${mongo.user}:${mongo.pass}@${mongo.dbname}">
    <mongo:client-options write-concern="NORMAL" />
  </mongo:mongo-client>
</beans>
```

218.4. サンプルルート

Spring XML で定義された次のルートは、コレクションに対して操作 dbStats を実行します。

指定されたコレクションの DB 統計を取得する

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb3:mongoBean?
database=${mongodb.database}&amp;collection=${mongodb.collection}&amp;operation=getDbStats"
/>
  <to uri="direct:result" />
</route>
```

218.5. MONGODB 操作 - プロデューサーエンドポイント

218.5.1. クエリー操作

218.5.1.1. findById

この操作は、_id フィールドが IN メッセージ本文の内容と一致するコレクションから 1 つの要素のみを取得します。着信オブジェクトは、**Bson** 型と同等のものであれば何でもかまいません。http://bsonspec.org/specification[http://bsonspec.org/specification] および http://www.mongodb.org/display/DOCS/Java+Types を参照してください。

```
from("direct:findById")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findById")
    .to("mock:resultFindById");
```

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドフィルターの指定をサポートします。オプションのパラメーターの指定 を参照してください。

218.5.1.2. findOneByQuery

この操作を使用して、MongoDB クエリーに一致するコレクションから 1 つの要素 (最初の要素) だけを取得します。**クエリーオブジェクトは CamelMongoDbCriteria ヘッダーから抽出されます。** CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトは抽出されたメッセージ本文で

す。つまり、タイプ **Bson** であるか、または **Bson** に変換可能である必要があります。JSON 文字列または Hashmap にすることができます。詳しくは # 型変換 を参照してください。MongoDB Driver の Filters クラスを使用できます。

クエリーのない例 (コレクションの任意のオブジェクトを返します):

```
from("direct:findOneByQuery")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

クエリーのある例 (一致する結果を 1 つ返します):

```
from("direct:findOneByQuery")
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドプロジェクションおよび/または並べ替え句の指定をサポートします。オプションのパラメーターの指定 を参照してください。

218.5.1.3. findAll

findAll オペレーションは、クエリーに一致するすべてのドキュメントを返すか、またはまったく一致しないドキュメントを返します。この場合、コレクションに含まれるすべてのドキュメントが返されます。**クエリーオブジェクトは CamelMongoDbCriteria ヘッダーから抽出されます。**
CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトは抽出されたメッセージ本文です。つまり、タイプ **Bson** であるか、または **Bson** に変換可能である必要があります。JSON 文字列または Hashmap にすることができます。詳細については、#型変換 を参照してください。

クエリーのない例 (コレクション内のすべてのオブジェクトを返します):

```
from("direct:findAll")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

クエリーのある例 (一致するすべての結果を返します):

```
from("direct:findAll")
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

ページングと効率的な検索は、次のヘッダーを介してサポートされています。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbNumToSkip	**MongoDbConstants.NUM_TO_SKIP**	カーソルの先頭にある指定された数の要素を破棄します。	int/Integer
CamelMongoDbLimit	**MongoDbConstants.LIMIT**	返される要素の数を制限します。	int/Integer
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	1 つのバッチで返される要素の数を制限します。通常、カーソルは結果オブジェクトのバッチをフェッチし、ローカルに保存します。batchSize が正の場合、取得されたオブジェクトの各バッチのサイズを表します。パフォーマンスを最適化し、データ転送を制限するように調整できます。batchSize が負の場合、最大バッチサイズ制限 (通常は 4MB) 内に収まる数のオブジェクトが返され、カーソルが閉じられます。たとえば、batchSize が -10 の場合、サーバーは最大 10 個のドキュメントを 4MB に収まる数だけ返し、カーソルを閉じます。この機能は、ドキュメントが最大サイズ内に収まる必要があるという点で limit() とは異なり、サーバー側でカーソルを閉じる要求を送信する必要がないことに注意してください。バッチサイズは、カーソルが反復された後でも変更できます。その場合、設定は次のバッチ取得に適用されます。	int/Integer

オプション outputType=MongoIterable とバッチサイズの例:

```
from("direct:findAll")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll&outputType=MongoIterable")
    .to("mock:resultFindAll");
```

ページングを使用している場合、**findAll** 操作は次の OUT ヘッダーも返し、結果ページを反復処理できるようにします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	データのタイプ
CamelMongoDbResultTotalSize	**MongoDbConstants.RESULT_TOTAL_SIZE**	クエリーに一致するオブジェクトの数。これは、制限/スキップを考慮していません。	int/Integer
CamelMongoDbResultPageSize	**MongoDbConstants.RESULT_PAGE_SIZE**	クエリーに一致するオブジェクトの数。これは、制限/スキップを考慮していません。	int/Integer

ヒント

オプションのパラメーターをサポートします。この操作は、フィールドプロジェクションおよび/または並べ替え句の指定をサポートします。オプションのパラメーターの指定 を参照してください。

218.5.1.4. count

コレクション内のオブジェクトの総数を返し、OUT メッセージ本文として Long を返します。
次の例では、dynamicCollectionName コレクション内のレコード数をカウントします。動的性が有効になっていることに注意してください。その結果、操作は "notableScientists" コレクションではなく、"dynamicCollectionName" コレクションに対して実行されます。

```
// from("direct:count").to("mongodb3:myDb?
database=tickets&collection=flights&operation=count&dynamicity=true");
Long result = template.requestBodyAndHeader("direct:count", "irrelevantBody",
MongoDbConstants.COLLECTION, "dynamicCollectionName");
assertTrue("Result is not of type Long", result instanceof Long);
```

クエリオブジェクトは CamelMongoDbCriteria ヘッダーから展開されます。CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトは抽出されたメッセージ本文です。つまり、タイプ **Bson** または **Bson** に変換可能である必要があり、操作はこの条件に一致するドキュメントの量を返します。

```
Document query = ...
Long count = template.requestBodyAndHeader("direct:count", query,
MongoDbConstants.COLLECTION, "dynamicCollectionName");
```

218.5.1.5. フィールドフィルターの指定 (プロジェクション)

デフォルトでは、クエリー操作は、一致するオブジェクト全体を (すべてのフィールドとともに) 返しま

す。ドキュメントが大きく、フィールドのサブセットのみを取得する必要がある場合は、関連する **Bson** (または JSON 文字列、マップなどの **Bson** に変換可能な型) を設定するだけで、すべてのクエリー操作でフィールドフィルターを指定できます。) **CamelMongoDbFieldsProjection** ヘッダーの定数ショートカット: **MongoDbConstants.FIELDS_PROJECTION**。

これは、MongoDB の **Projections** を使用して Bson の作成を簡素化する例です。 **_id** と **boringField** を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

これは、MongoDB の **Projections** を使用して Bson の作成を簡素化する例です。 **_id** と **boringField** を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

218.5.1.6. ソート句の指定

Bson の作成を簡素化するために、MongoDB の **Sorts** を使用する特定のフィールドによるソートに基づいて、コレクションから最小/最大レコードを取得する必要があることがよくあります。**_id** と **boringField** を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson sorts = Sorts.descending("_id");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.SORT_BY, sorts);
```

Camel ルートでは、SORT_BY ヘッダーを findOneByQuery 操作で使用して、同じ結果を得ることができます。FIELDS_PROJECTION ヘッダーも指定されている場合、操作は、別のコンポーネント (たとえば、パラメーター化された MyBatis SELECT クエリー) に直接渡すことができる単一のフィールド/値のペアを返します。この例では、コレクションから一時的に最新のドキュメントをフェッチし、結果を **documentTimestamp** フィールドに基づいて 1 つのフィールドに削減する方法を示します。

```
.from("direct:someTriggeringEvent")
.setHeader(MongoDbConstants.SORT_BY).constant(Sorts.descending("documentTimestamp"))
.setHeader(MongoDbConstants.FIELDS_PROJECTION).constant(Projection.include("documentTime
stamp"))
.setBody().constant("{}")
.to("mongodb3:myDb?database=local&collection=myDemoCollection&operation=findOneByQuery")
.to("direct:aMyBatisParameterizedSelect")
;
```

218.5.2. 操作の作成/更新

218.5.2.1. insert

IN メッセージ本文から取得した新しいオブジェクトを MongoDB コレクションに挿入します。**Document** または **List** に変換するために型変換が試行されます。
シングル挿入とマルチ挿入の 2 つのモードがサポートされています。複数の挿入の場合、エンドポイントは、Document であるか、または **Document** に変換できる限り、任意のタイプのオブジェクトのリスト、配列、またはコレクションを期待します。以下に例を示します。

```
from("direct:insert")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=insert");
```

オペレーションは WriteResult を返します。**WriteConcern** または **invokeGetLastError** オプションの値に応じて、**getLastError()** がすでに呼び出されているかどうかが決まります。書き込み操作の最終的な結果にアクセスする場合は、**WriteResult** で **getLastError()** または **getCachedLastError()** を呼び出して **CommandResult** を取得する必要があります。次に、**CommandResult.ok()**、**CommandResult.getErrorMessage()** および/または **CommandResult.getException()** を呼び出して結果を確認できます。

新しいオブジェクトの **_id** はコレクション内で一意である必要があることに注意してください。値を指定しない場合、MongoDB が自動的に値を生成します。ただし、指定しても一意でない場合、挿入操作は失敗します (Camel が気付くには、invokeGetLastError を有効にするか、書き込み結果を待機する WriteConcern を設定する必要があります)。

これはコンポーネントの制限ではありませんが、MongoDB でより高いスループットを実現する方法です。カスタム **_id** を使用している場合は、アプリケーションレベルで一意であることを確認する必要があります (これも良い方法です)。

挿入されたレコードの OID は、**CamelMongoOid** キー (**MongoDbConstants.OID** 定数) の下のメッセージヘッダーに格納されます。保存される値は、単一の挿入の場合は **org.bson.types.ObjectId**、複数のレコードが挿入された場合は **java.util.List<org.bson.types.ObjectId>** です。

MongoDB Java Driver 3.x では、insertOne および insertMany オペレーションは void を返します。Camel 挿入操作は、挿入されたドキュメントまたはドキュメントのリストを返します。必要に応じて、各ドキュメントが新しい OID によって更新されることに注意してください。

218.5.2.2. save

保存操作は **upsert** (UPdate、inSERT) 操作と同等で、レコードが更新され、レコードが存在しない場合は挿入されます。これらはすべて 1 つのアトミック操作で行われます。MongoDB は **_id** フィールドに基づいてマッチングを実行します。

更新の場合、オブジェクトは完全に置き換えられ、MongoDB の $modifiers の使用は許可されないことに注意してください。したがって、オブジェクトがすでに存在する場合にそのオブジェクトを操作する場合は、次の 2 つのオプションがあります。

1. 最初にオブジェクト全体とそのすべてのフィールドを取得するためのクエリーを実行し (効率的でない場合があります)、Camel 内で変更してから保存します。
2. $modifiers で更新操作を使用すると、代わりにサーバー側で更新が実行されます。upsert フラグを有効にできます。この場合、挿入が必要な場合、MongoDB は $modifiers をフィルタークエリーオブジェクトに適用し、結果を挿入します。

保存するドキュメントに **_id** 属性が含まれていない場合、操作は挿入になり、作成された新しい **_id** が **CamelMongoOid** ヘッダーに配置されます。

以下に例を示します。

```
from("direct:insert")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=save");
```

218.5.2.3. update

コレクションの 1 つまたは複数のレコードを更新します。フィルタークエリーと更新ルールが必要です。

MongoDBConstants.CRITERIA ヘッダーを使用してフィルターを **Bson** として定義し、更新ルールを本文で **Bson** として定義できます。

2 番目の方法は、正確に 2 つの要素を含む IN メッセージボディーとして List<Bson> を要求します。

- 要素 1 (インデックス 0) ⇒ フィルタークエリー ⇒ 通常のクエリーオブジェクトと同じように、影響を受けるオブジェクトを決定します
- 要素 2 (インデックス 1) ⇒ 更新ルール ⇒ 一致したオブジェクトがどのように更新されるか。MongoDB からのすべての 修飾子操作 がサポートされています。

注記

マルチアップデート。デフォルトでは、MongoDB は、複数のオブジェクトがフィルタークエリーに一致する場合でも、1 つのオブジェクトのみを更新します。一致する **すべての** レコードを更新するように MongoDB に指示するには、**CamelMongoDbMultiUpdate** IN メッセージヘッダーを **true** に設定します。

キー **CamelMongoDbRecordsAffected** を持つヘッダーが返されます (**MongoDbConstants.RECORDS_AFFECTED** 定数) 更新されたレコードの数 (**WriteResult.getN()** からコピーされます)。

次の IN メッセージヘッダーをサポートします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbMultiUpdate	**MongoDbConstants.MULTIUPDATE**	一致するすべてのオブジェクトに更新を適用する必要がある場合。http://www.mongodb.org/display/DOCS/Atomic+Operations を参照してください。	boolean/Boolean
CamelMongoDbUpsert	**MongoDbConstants.UPSERT**	存在しない場合にデータベースが要素を作成する必要があるかどうか	boolean/Boolean

たとえば、次の例では、"scientist" フィールドの値を "Darwin" に設定することで、filterField フィールドが true に等しい **すべての** レコードを更新します。

```
// route: from("direct:update").to("mongodb3:myDb?database=science&collection=notableScientists&operation=update");
Bson filterField = Filters.eq("filterField", true);
String updateObj = Updates.set("scientist", "Darwin");
Object result = template.requestBodyAndHeader("direct:update", new Bson[] {filterField, Document.parse(updateObj)}, MongoDbConstants.MULTIUPDATE, true);
```

```
// route: from("direct:update").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=update");
Maps<String, Object> headers = new HashMap<>(2);
headers.add(MongoDbConstants.MULTIUPDATE, true);
headers.add(MongoDbConstants.FIELDS_FILTER, Filters.eq("filterField", true));
String updateObj = Updates.set("scientist", "Darwin");;
Object result = template.requestBodyAndHeaders("direct:update", updateObj, headers);
```

```
// route: from("direct:update").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=update");
String updateObj = "[{\"filterField\": true}, {\"$set\", {\"scientist\", \"Darwin\"}}]";
Object result = template.requestBodyAndHeader("direct:update", updateObj,
MongoDbConstants.MULTIUPDATE, true);
```

218.5.3. 操作の削除

218.5.3.1. remove

コレクションから一致するレコードを削除します。IN メッセージ本文は、削除フィルタークエリーとして機能し、**DBObject** タイプまたはそれに変換可能なタイプであることが期待されます。
次の例では、科学データベースの notableScientists コレクションで、フィールド 'conditionField' が true に等しいすべてのオブジェクトを削除します。

```
// route: from("direct:remove").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=remove");
Bson conditionField = Filters.eq("conditionField", true);
Object result = template.requestBody("direct:remove", conditionField);
```

キー **CamelMongoDbRecordsAffected** を持つヘッダーが返されます (**MongoDbConstants.RECORDS_AFFECTED** 定数)。タイプは **int** で、削除された (**WriteResult.getN()** からコピーされた) レコードの数を含みます。

218.5.4. 一括書き込み操作

218.5.4.1. bulkWrite

Camel 2.21 以降で利用可能

実行順序を制御して書き込み操作を一括で実行します。挿入、更新、および削除操作のコマンドを含む IN メッセージ本文として **List<WriteModel<Document>>** が必要です。

次の例では、新しい科学者 "Pierre Curie" を挿入し、"scientist" フィールドの値を "Marie Curie" に設定して ID "5" のレコードを更新し、ID "3" のレコードを削除します。

```
// route: from("direct:bulkWrite").to("mongodb:myDb?
database=science&collection=notableScientists&operation=bulkWrite");
List<WriteModel<Document>> bulkOperations = Arrays.asList(
        new InsertOneModel<>(new Document("scientist", "Pierre Curie")),
        new UpdateOneModel<>(new Document("_id", "5"),
                        new Document("$set", new Document("scientist", "Marie Curie"))),
        new DeleteOneModel<>(new Document("_id", "3")));
```

```
BulkWriteResult result = template.requestBody("direct:bulkWrite", bulkOperations,
BulkWriteResult.class);
```

デフォルトでは、操作は順番に実行され、最初の書き込みエラーで中断され、リスト内の残りの書き込み操作は処理されません。リスト内の残りの書き込み操作の処理を続行するように MongoDB に指示するには、**CamelMongoDbBulkOrdered** IN メッセージヘッダーを **false** に設定します。順序付けされていない操作は並行して実行され、この動作は保証されません。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbBulkOrdered	**MongoDbConstants.BULK_ORDERED**	順序付きまたは順序なしの操作実行を実行します。デフォルトは true です。	boolean/Boolean

218.5.5. その他の操作

218.5.5.1. aggregate

本文に含まれる特定のパイプラインを使用して集計を実行します。**集約は、長く重い操作になる可能性があります。注意して使用してください。**

```
// route: from("direct:aggregate").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate");
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
        group("$scientist", sum("count", 1)));
from("direct:aggregate")
    .setBody().constant(aggregate)
    .to("mongodb3:myDb?database=science&collection=notableScientists&operation=aggregate")
    .to("mock:resultAggregate");
```

次の IN メッセージヘッダーをサポートします。

ヘッダーのキー	クイック定数	説明 (MongoDB API ドキュメントから抜粋)	想定されるタイプ
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	バッチごとに返すドキュメントの数を設定します。	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	集約パイプラインステージを有効にして、データを一時ファイルに書き込みます。	boolean/Boolean

outputType=MongoIterable を使用すると、効率的な取得がサポートされます。

上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして outputType=DBCursor (Camel 2.21+) を含めることで、サーバーから返されたドキュメントをルートにストリーミングすることもできます。これにより、Mongo シェル内で aggregate() を実行しているかのように、Mongo ドライバーから Exchange に DBCursor が渡され、ルートが結果を反復できるようになります。デフォルトでは、このオプションを指定しないと、このコンポーネントはドキュメントをドライバーのカーソルから List にロードし、これをルートに返します。これにより、多数のメモリー内オブジェクトが発生する可能性があります。DBCursor では、一致したドキュメントの数を問い合わせないでください。詳細については、MongoDB ドキュメントサイトを参照してください。

オプション outputType=MongoIterable とバッチサイズの例:

```
// route: from("direct:aggregate").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=MongoIterable");
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
        group("$scientist", sum("count", 1)));
from("direct:aggregate")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setBody().constant(aggregate)
    .to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=MongoIterable")
    .to("mock:resultAggregate");
```

-

218.5.5.2. getDbStats

MongoDB シェルで **db.stats()** コマンドを実行するのと同じです。これは、データベースに関する有用な統計値を表示します。
以下に例を示します。

```
> db.stats();
{
    "db" : "test",
    "collections" : 7,
    "objects" : 719,
    "avgObjSize" : 59.73296244784423,
    "dataSize" : 42948,
    "storageSize" : 1000058880,
    "numExtents" : 9,
    "indexes" : 4,
    "indexSize" : 32704,
    "fileSize" : 1275068416,
    "nsSizeMB" : 16,
    "ok" : 1
}
```

使用例:

```
// from("direct:getDbStats").to("mongodb3:myDb?
database=flights&collection=tickets&operation=getDbStats");
Object result = template.requestBody("direct:getDbStats", "irrelevantBody");
assertTrue("Result is not of type Document", result instanceof Document);
```

この操作は、シェルに表示されるものと同様のデータ構造を、OUT メッセージ本文の **Document** の形式で返します。

218.5.5.3. getColStats

コレクションに関する有用な統計値を表示する、MongoDB シェルで **db.collection.stats()** コマンドを実行するのと同じです。
以下に例を示します。

```
> db.camelTest.stats();
{
    "ns" : "test.camelTest",
    "count" : 100,
    "size" : 5792,
    "avgObjSize" : 57.92,
    "storageSize" : 20480,
    "numExtents" : 2,
    "nindexes" : 1,
    "lastExtentSize" : 16384,
    "paddingFactor" : 1,
    "flags" : 1,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
```

```
    },
    "ok" : 1
}
```

使用例:

```
// from("direct:getColStats").to("mongodb3:myDb?
database=flights&collection=tickets&operation=getColStats");
Object result = template.requestBody("direct:getColStats", "irrelevantBody");
assertTrue("Result is not of type Document", result instanceof Document);
```

この操作は、シェルに表示されるものと同様のデータ構造を、OUT メッセージ本文の **Document** の形式で返します。

218.5.5.4. command

データベースで本文をコマンドとして実行します。ホスト情報、レプリケーション、またはシャーディングステータスを取得するなどの管理操作に役立ちます。

コレクションパラメーターは、この操作では使用されません。

```
// route: from("command").to("mongodb3:myDb?database=science&operation=command");
DBObject commandBody = new BasicDBObject("hostInfo", "1");
Object result = template.requestBody("direct:command", commandBody);
```

218.5.6. 動的操作

エクスチェンジは、**MongoDbConstants.OPERATION_HEADER** 定数で定義された **CamelMongoDbOperation** ヘッダーを設定することにより、エンドポイントの固定操作をオーバーライドできます。
サポートされる値は、MongoDbOperation 列挙によって決定され、エンドポイント URI の **operation** パラメーターで受け入れられる値と一致します。

以下に例を示します。

```
// from("direct:insert").to("mongodb3:myDb?database=flights&collection=tickets&operation=insert");
Object result = template.requestBodyAndHeader("direct:insert", "irrelevantBody",
MongoDbConstants.OPERATION_HEADER, "count");
assertTrue("Result is not of type Long", result instanceof Long);
```

218.6. TAILABLE カーソルコンシューマー

MongoDB は、*nix システムの **tail -f** コマンドのようにカーソルを開いたままにすることで、コレクションから進行中のデータを瞬時に消費するメカニズムを提供します。このメカニズムは、クライアントがスケジュールされた間隔で ping を返して新しいデータを取得するのではなく、新しいデータが利用可能になったときにサーバーがクライアントにプッシュするため、スケジュールされたポーリングよりもはるかに効率的です。また、冗長なネットワークトラフィックも削減されます。

tailable カーソルを使用するための必要条件は 1 つだけです。つまり、コレクションは " 上限付きコレクション " である必要があります。これは、N 個のオブジェクトのみを保持することを意味し、制限に達すると、MongoDB は最初に挿入されたのと同じ順序で古いオブジェクトをフラッシュします。詳細は、http://www.mongodb.org/display/DOCS/Tailable+Cursors を参照してください。

Camel MongoDB コンポーネントは、tailable カーソルコンシューマーを実装しているため、この機能を Camel ルートで使用できるようになります。新しいオブジェクトが挿入されると、MongoDB はそれらを **Document** として自然な順序で tailable カーソルコンシューマーにプッシュします。tailable カーソルコンシューマーはそれらをエクスチェンジに変換し、ルートロジックをトリガーします。

218.7. TAILABLE カーソルコンシューマーの仕組み

カーソルを tailable カーソルに変えるには、最初にカーソルを生成するときに、いくつかの特別なフラグを MongoDB に通知する必要があります。作成されると、カーソルは開いたままになり、新しいデータが到着するまで **MongoCursor.next()** メソッドを呼び出すとブロックされます。ただし、MongoDB サーバーは、不確定な期間が経過しても新しいデータが表示されない場合、カーソルを強制終了する権利を留保します。新しいデータを引き続き使用する場合は、カーソルを再生成する必要があります。そのためには、中断した位置を覚えておく必要があります。そうしないと、もう一度最初から消費し始めます。

Camel MongoDB テーラブルカーソルコンシューマーは、これらすべてのタスクを処理します。タイムスタンプ、シーケンシャル ID など、再生成されるたびにカーソルを配置するマーカーとして機能する、増加する性質のデータ内のフィールドにキーを提供するだけで済みます。MongoDB でサポートされている任意のデータ型にすることができます。日付、文字列、および整数がうまく機能することがわかっています。このコンポーネントのコンテキストでは、このメカニズムをテールトラッキングと呼びます。

コンシューマーはこのフィールドの最後の値を記憶し、カーソルが再生成されるたびに、次のようなフィルターを使用してクエリーを実行します。**increasingField > lastValue** のようなフィルタでクエリを実行し、未読のデータのみが消費されるようにします。

増加フィールドの設定: エンドポイント URI の **tailTrackingIncreasingField** オプションで増加フィールドのキーを設定します。Camel 2.10 では、このフィールドのネストされたナビゲーションがまだサポートされていないため、データの最上位フィールドである必要があります。つまり、"timestamp" フィールドは問題ありませんが、"nested.timestamp" は機能しません。ネストされた増加するフィールドのサポートが必要な場合は、Camel JIRA でチケットを開いてください。

カーソル再生成の遅延: 注意すべきことの 1 つは、初期化時に新しいデータがまだ利用できない場合、MongoDB はカーソルを即座に強制終了することです。この場合、サーバーに負荷をかけたくないので、**cursorRegenerationDelay** オプションが導入されています (デフォルト値は 1000 ミリ秒です)。これは、ニーズに合わせて変更できます。

以下に例を示します。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime")
    .id("tailableCursorConsumer1")
    .autoStartup(false)
    .to("mock:test");
```

上記のルートは、departureTime を増加フィールドとして使用し、デフォルトの再生成カーソル遅延を 1000 ミリ秒にして、flights.cancellations キャップコレクションから消費します。

218.8. 永続的なテールトラッキング

標準のテールトラッキングは揮発性であり、最後の値はメモリーにのみ保持されます。ただし、実際には Camel コンテナーを時々再起動する必要がありますが、最後の値は失われ、tailable カーソルコンシューマーは再び先頭から消費を開始し、ルートに重複レコードを送信する可能性が非常に高くなります。

この状況を克服するために、**永続的なテール追跡** 機能を有効にして、MongoDB データベース内の特別なコレクションで最後に消費された増加値を追跡することもできます。コンシューマーが再び初期化されると、最後に追跡された値が復元され、何も起こらなかったかのように続行されます。

最後に読み取られた値は、カーソルが再生成されるたびとコンシューマーがシャットダウンするときの 2 つの場合に保持されます。需要があれば、堅牢性を高めるために、将来的には定期的な間隔 (5 秒ごとにフラッシュ) で永続化することも検討する可能性があります。この機能をリクエストするには、Camel JIRA でチケットを開いてください。

218.9. 永続的なテールトラッキングを有効にする

この機能を有効にするには、エンドポイント URI で少なくとも次のオプションを設定します。

- **persistentTailTracking** オプションを **true** に設定
- このコンシューマーの一意の識別子に **persistentId** オプションを追加して、同じコレクションを多くのコンシューマーで再利用できるようにします

さらに、**tailTrackDb**、**tailTrackCollection**、および **tailTrackField** オプションを設定して、ランタイム情報が保存される場所をカスタマイズできます。各オプションの説明については、このページの上部にあるエンドポイントオプションの表を参照してください。

たとえば、次のルートは、departureTime を増加フィールドとして使用し、デフォルトの再生成カーソル遅延を 1000 ミリ秒に設定して、永続的なテールトラッキングをオンにし、cancellationsTracker の下で永続化して、flights.cancellations キャップコレクションから消費します。flights.camelTailTracking の id で、最後に処理された値を lastTrackingValue フィールドに格納します (**camelTailTracking** と **lastTrackingValue** はデフォルトです)。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker")
  .id("tailableCursorConsumer2")
  .autoStartup(false)
  .to("mock:test");
```

以下は、上記と同じ別の例ですが、永続的なテールトラッキングランタイム情報が trackers.camelTrackers コレクションの lastProcessedDepartureTime フィールドに格納されます。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker&tailTrackDb=trackers&tailTrackCollection=camelTrackers" +
    "&tailTrackField=lastProcessedDepartureTime")
  .id("tailableCursorConsumer3")
  .autoStartup(false)
  .to("mock:test");
```

218.10. 型変換

camel-mongodb コンポーネントに含まれる **MongoDbBasicConverters** 型コンバーターは、次の変換を提供します。

名前	タイプから	入力し	どのように変更を加えればよいですか？
fromMapToDocument	**Map**	**Document**	**new Document (Map m)** コンストラクターを介して新しい **Document** を構築します。
fromDocumentToMap	**Document**	**Map**	**ドキュメントは** すでに **Map** を実装しています。
fromStringToDocument	**String**	**Document**	**com.mongodb.Document.parse(String s)** を使用します。
fromAnyObjectToDocument	**Object**	**Document**	Jackson ライブラリー を使用してオブジェクトを **Map** に変換し、それを使用して新しい **Document** を初期化します。
fromStringToList	**String**	**List<Bson>**	**org.bson.codecs.configuration.CodecRegistries** を使用して BsonArray に変換し、次に List<Bson> に変換します。

この型コンバーターは自動検出されるため、手動で設定する必要はありません。

218.11. その他の参考資料

- MongoDB web サイト
- NoSQL ウィキペディアの記事
- MongoDB Java ドライバー API ドキュメント - 最新バージョン * 使用例の 単体テスト

第219章 MQTT コンポーネント

Camel バージョン 2.10 以降で利用可能

mqtt: コンポーネントは、Apache ActiveMQ や Mosquitto などの MQTT 準拠のメッセージブローカーとの通信に使用されます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mqtt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

219.1. URI 形式

```
mqtt://name[?options]
```

name は、コンポーネントに割り当てる名前です。

219.2. オプション

MQTT コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
host (Common)	接続する MQTT ブローカーの URI - このコンポーネントは SSL もサポートしています - 例: ssl://127.0.0.1:8883		String
userName (security)	MQTT ブローカーに対する認証に使用されるユーザー名		String
password (security)	MQTT ブローカーに対する認証に使用されるパスワード		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MQTT エンドポイントは、URI 構文を使用して設定されます。

```
mqtt:name
```

パスおよびクエリーパラメーターを使用します。

219.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** トピック名ではない、使用する論理名。		String

219.2.2. クエリーパラメーター(39 個のパラメーター):

名前	説明	デフォルト	タイプ
blockingExecutor (common)	SSL 接続は、setBlockingExecutor メソッドを呼び出して代わりに使用するエグゼキューターを設定しない限り、内部スレッドプールに対してブロック操作を実行します。		エグゼキューター (Executor)
byDefaultRetain (common)	MQTT ブローカーに送信されるメッセージで使用されるデフォルトの保持ポリシー	false	boolean
cleanSession (common)	MQTT サーバーがクライアントセッション間でトピックサブスクリプションと ACK 位置を保持するようにする場合は、false に設定します。デフォルトは true です。	false	boolean
clientId (common)	セッションのクライアント ID を設定するために使用します。これは、MQTT サーバーが setCleanSession(false); のセッションを識別するために使用するものです。使用されています。ID は 23 文字以下にする必要があります。デフォルトは、自動生成された ID (ソケットアドレス、ポート、およびタイムスタンプに基づく) です。		String
connectAttemptsMax (common)	クライアントがサーバーへの最初の接続試行時にエラーがクライアントに報告されるまでの再接続試行の最大回数。無制限の試行を使用するには、-1 に設定します。デフォルトは -1 です。	-1	long
connectWaitInSeconds (common)	MQTT ブローカーへの接続が確立されるまでコンポーネントが待機する遅延 (秒単位)	10	int
disconnectWaitInSeconds (common)	コンポーネントが MQTT ブローカーからの stop () での有効な切断を待機する秒数	5	int

名前	説明	デフォルト	タイプ
dispatchQueue (common)	HawtDispatch ディスパッチキューは、接続へのアクセスを同期するために使用されます。setDispatchQueue メソッドを使用して明示的なキューが設定されていない場合、接続用に新しいキューが作成されます。複数の接続で Synchronization のために同じキューを共有する場合は、明示的なキューを設定すると便利です。		DispatchQueue
host (Common)	接続する MQTT ブローカーの URI - このコンポーネントは SSL もサポートしています - 例: ssl://127.0.0.1:8883	tcp://127.0.0.1:1883	URI
keepAlive (common)	キープアライブタイマーを秒単位で設定します。クライアントから受信するメッセージ間の最大時間間隔を定義します。これにより、サーバーは長い TCP/IP タイムアウトを待たずに、クライアントへのネットワーク接続が切断されたことを検出できます。		short
localAddress (common)	使用するローカル InetAddress とポート		URI
maxReadRate (common)	このトランスポートがデータを受信する 1 秒あたりの最大バイト数を設定します。この設定は、レートを超えないように読み取りを調整します。デフォルトは 0 で、スロットリングを無効にします。		int
maxWriteRate (common)	このトランスポートがデータを送信する 1 秒あたりの最大バイト数を設定します。この設定は、レートを超えないように書き込みを抑制します。デフォルトは 0 で、スロットリングを無効にします。		int
mqttQosPropertyName (common)	公開された個々のメッセージを Exchange で検索するプロパティー名。これが設定されている場合 (AtMostOnce、AtLeastOnce または ExactlyOnce のいずれか) - MQTT メッセージブローカーに送信されるメッセージにその QoS が設定されます。	MQTT Qos	String
mqttRetainPropertyName (common)	公開された個々のメッセージを Exchange で検索するプロパティー名。これが設定されている場合 (ブール値が必要) - MQTT メッセージブローカーに送信されるメッセージに保持プロパティーが設定されます。	MQTT Retain	String

名前	説明	デフォルト	タイプ
mqttTopicPropertyName (common)	これらのプロパティは、Exchange - に公開するために探されるものです。	MQTTTopicPropertyName	String
publishTopicName (common)	メッセージを公開するデフォルトのトピック	camel/mqtt/test	String
qualityOfService (common)	トピックに使用するサービス品質レベル。	AtLeastOnce	String
receiveBufferSize (common)	内部ソケット受信バッファーのサイズを設定します。デフォルトは 65536 (64k)	65536	int
reconnectAttemptsMax (common)	サーバー接続が以前に確立された後、エラーがクライアントに報告されるまでの再接続の最大試行回数。無制限の試行を使用するには、-1 に設定します。デフォルトは -1 です。	-1	long
reconnectBackOffMultiplier (common)	再接続試行の間に指数バックオフが使用されます。指数バックオフを無効にするには、1 に設定します。デフォルトは 2 です。	2.0	double
reconnectDelay (common)	最初の再接続試行までの待機時間 (ミリ秒)。デフォルトは 10 です。	10	long
reconnectDelayMax (common)	次の再接続試行までの最大待機時間 (ミリ秒)。デフォルトは 30,000 です。	30000	long
sendBufferSize (common)	内部ソケット送信バッファーのサイズを設定します。デフォルトは 65536 (64k)	65536	int
sendWaitInSeconds (common)	例外を出力する前に、MQTT ブローカからの受信がパブリッシュされたメッセージを確認するのをコンポーネントが待機する最大時間	5	int
sslContext (common)	SSLContext 設定を使用してセキュリティーを設定するには		SSLContext
subscribeTopicName (common)	**非推奨** これらはエンドポイントに設定され、MQTT から継承されたプロパティと合わせて使用されます。		String

名前	説明	デフォルト	タイプ
subscribeTopicNames (common)	メッセージをサブスクライブするトピックのコンマ区切りリスト。このリストの各項目には、階層内の特定のパターンに一致するトピックをサブスクライブするために、MQTT ワイルドカード (および/または) を含めることができることに注意してください。たとえば、は階層内のあるレベルのすべてのトピックのワイルドカードであるため、ブローカーにトピック topic/one と topic/two がある場合、topics/ を使用して両方をサブスクライブできます。ここで考慮すべき注意点は、ブローカーがトピック/3 を追加すると、ルートもそのトピックからのメッセージの受信を開始することです。		String
trafficClass (common)	トランスポートから送信されるパケットの IP ヘッダーにトラフィッククラスまたはタイプオブサービスオクテットを設定します。デフォルトは 8 で、スループットのためにトラフィックを最適化する必要があることを意味します。	8	int
version (common)	MQTT バージョン 3.1.1 を使用するには、3.1.1 に設定します。それ以外の場合は、デフォルトで 3.1 プロトコルバージョンになります。	3.1	String
willMessage (common)	送信する Will メッセージ。デフォルトは長さゼロのメッセージです。		String
willQos (common)	Will メッセージに使用するサービスの品質を設定します。デフォルトは AT_MOST_ONCE です。	AtMostOnce	QoS
willRetain (common)	Will を保持オプション付きで公開する場合は、true に設定します。		QoS
willTopic (common)	設定されている場合、サーバーは、クライアントに予期しない切断が発生した場合に、クライアントの Will メッセージを指定されたトピックに公開します。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
lazySessionCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにセッションを遅延作成できます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

219.3. サンプル

メッセージの送信:

```
from("direct:foo").to("mqtt:cheese?publishTopicName=test.mqtt.topic");
```

メッセージの消費:

```
from("mqtt:bar?
subscribeTopicName=test.mqtt.topic").transform(body().convertToString()).to("mock:result")
```

219.4. エンドポイント

Camel は、エンドポイント インターフェイスを使用してメッセージエンドポイントパターンをサポートします。通常、エンドポイントはコンポーネントによって作成され、エンドポイントは通常、URI を介して DSL で参照されます。

エンドポイントから、次のメソッドを使用できます

- createProducer() は、メッセージエクスチェンジをエンドポイントに送信するための プロデューサー を作成します
- createConsumer() は、コンシューマー の作成時に、プロセッサー を介してエンドポイントからのメッセージ交換を消費するための Event Driven Consumer パターンを実装します。
- createPollingConsumer () は、 PollingConsumer を介してエンドポイントからのメッセージエクスチェンジを消費するためのポーリングコンシューマーパターンを実装します

219.5. 関連項目

- Configuring Camel (Camel の設定)
- Message Endpoint パターン
- URI
- コンポーネントの作成

第220章 MSV コンポーネント

Camel バージョン 1.1 以降で利用可能

MSV コンポーネントは、MSV ライブラリー と、XML スキーマ や RelaxNG XML Syntax などのサポートされている XML スキーマ言語のいずれかを使用して、メッセージボディーの XML バリデーションを実行します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-msv</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Jing コンポーネントは RelaxNG コンパクト構文 もサポートしていることに注意してください

220.1. URI 形式

```
msv:someLocalOrRemoteResource[?options]
```

someLocalOrRemoteResource は、クラスパス上のローカルリソースへの URL、またはリモートリソースまたはファイルシステム上のリソースへの完全な URL です。以下に例を示します。

```
msv:org/foo/bar.rng
msv:file:../foo/bar.rng
msv:http://acme.com/cheese.rng
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

220.2. オプション

MSV コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
schemaFactory (advanced)	javax.xml.validation.SchemaFactory を使用する場合。		SchemaFactory
resourceResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタム LSResourceResolver を使用する場合		ValidatorResourceResolverFactory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MSV エンドポイントは、URI 構文を使用して設定されます。

```
msv:resourceUri
```

パスおよびクエリーパラメーターを使用します。

220.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	クラスパス上のローカルリソースへの **必須** URL、またはレジストリー内の Bean を検索するための参照、または検証対象の XSD を含むファイルシステム上のリモートリソースまたはリソースへの完全な URL。		String

220.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
failOnNullBody (producer)	本文が存在しない場合に失敗するかどうか。	true	boolean
failOnNullHeader (producer)	ヘッダーに対して検証するときに、ヘッダーが存在しない場合に失敗するかどうか。	true	boolean
headerName (producer)	メッセージボディーではなくヘッダーに対して検証します。		String
errorHandler (advanced)	カスタム org.apache.camel.processor.validation.ValidatorErrorHandler を使用するには。デフォルトのエラーハンドラーはエラーをキャプチャし、例外を出力します。		ValidatorErrorHandler
resourceResolver (advanced)	カスタム LSResourceResolver を使用するには。resourceResolverFactory と一緒に使用しないでください		LSResourceResolver
resourceResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタム LSResourceResolver を使用するには。デフォルトのリソースリゾルバーファクトリーは、クラスパスとファイルシステムからファイルを読み取ることができるリソースリゾルバーを返します。resourceResolver と一緒に使用しないでください。		ValidatorResourceResolverFactory
schemaFactory (advanced)	カスタム javax.xml.validation.SchemaFactory を使用する場合		SchemaFactory

名前	説明	デフォルト	タイプ
schemaLanguage (advanced)	W3C XML スキーマの namespace URI を設定します。	http://www.w3.org/2001/XMLSchema	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
useDom (advanced)	バリデーターが DOMSource/DOMResult または SaxSource/SaxResult を使用するかどうか。	false	boolean
useSharedSchema (advanced)	Schema インスタンスを共有するかどうか。このオプションは、JDK 1.6.x のバグを回避するために導入されました。Xerces にはこの問題はありません。	true	boolean

220.3. 例

次の 例 は、エンドポイント **direct:start** からのルートを設定する方法を示しており、このルートは、XML が指定された RelaxNG XML Schema (クラスパスで提供される) と一致するかどうかに基づいて、**mock:valid** または **mock:invalid** の 2 つのエンドポイントのいずれかに移動します。.

220.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第221章 MUSTACHE コンポーネント

Camel バージョン 2.12 以降で利用可能

mustache: コンポーネントを使用すると、Mustache テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
<groupId>org.apache.camel</groupId>
<artifactId>camel-mustache</artifactId>
<version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

221.1. URI 形式

```
mustache:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI またはリモートテンプレートの完全な URL です (例: file://folder/myfile.mustache)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

221.2. オプション

Mustache コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
mustacheFactory (advanced)	カスタム MustacheFactory を使用する場合。		MustacheFactory
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Mustache エンドポイントは、URI 構文を使用して設定されます。

```
mustache:resourceUri
```

パスおよびクエリーパラメーターを使用します。

221.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

221.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding (producer)	リソースコンテンツの文字エンコード。		String
endDelimiter (producer)	テンプレートコードの終わりを示すために使用される文字。	}}	String
startDelimiter (producer)	テンプレートコードの開始を示すために使用される文字。	{{	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

221.3. MUSTACHE コンテキスト

Camel は、Mustache コンテキスト (単なる **Map**) でエクスチェンジ情報を提供します。**Exchange** は次のように転送されます。

key	value
exchange	**Exchange** 自体。
exchange.properties	**Exchange** プロパティー。

key	value
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト
request	IN メッセージ
body	In メッセージボディー
response	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。

221.4. 動的テンプレート

Camel は、テンプレートまたはテンプレートコンテンツ自体の異なるリソースのロケーションを定義できる 2 つのヘッダーを提供します。これらのヘッダーのいずれかが設定されている場合、Camel はエンドポイントで設定されたリソースに対してこれを使用します。これにより、実行時に動的なテンプレートを提供できます。

ヘッダー	タイプ	説明	サポートバージョン
MustacheConstants.MUSTACHE_RESOURCE_URI	String	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。	
MustacheConstants.MUSTACHE_TEMPLATE	String	設定されたエンドポイントの代わりに使用するテンプレート。	

221.5. サンプル

たとえば、次のようなものを使用できます。

```
from("activemq:My.Queue").
to("mustache:com/acme/MyResponse.mustache");
```

Mustache テンプレートを使用して、InOut メッセージエクスチェンジ (**JMSReplyTo** ヘッダーがある場合) のメッセージに対する応答を作成します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、次を使用できます。

```
from("activemq:My.Queue").
to("mustache:com/acme/MyResponse.mustache").
to("activemq:Another.Queue");
```

コンポーネントがヘッダーを介して動的に使用する必要があるテンプレートを指定することが可能です。たとえば、次のようになります。

```
from("direct:in").
setHeader(MustacheConstants.MUSTACHE_RESOURCE_URI).constant("path/to/my/template.mustache").
to("mustache:dummy");
```

221.6. 電子メールのサンプル

このサンプルでは、注文確認メールに Mustache テンプレートを使用します。電子メールテンプレートは、Mustache で次のようにレイアウトされます。

```
Dear {{headers.lastName}}}, {{headers.firstName}}

Thanks for the order of {{headers.item}}.

Regards Camel Riders Bookstore
{{body}}
```

221.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第222章 MVEL コンポーネント

Camel バージョン 2.12 以降で利用可能

mvel: コンポーネントを使用すると、MVEL テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mvel</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

222.1. URI 形式

```
mvel:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI です。またはリモートテンプレートの完全な URL (例: file://folder/myfile.mvel)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

222.2. オプション

MVEL コンポーネントにはオプションがありません。

MVEL エンドポイントは、URI 構文を使用して設定されます。

```
mvel:resourceUri
```

パスおよびクエリーパラメーターを使用します。

222.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		文字列

222.2.2. クエリーパラメーター(3 個のパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding (producer)	リソースコンテンツの文字エンコード。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

222.3. メッセージヘッダー

mvel コンポーネントは、メッセージにいくつかのヘッダーを設定します。

ヘッダー	説明
CamelMvelResourceUri	**文字列** オブジェクトとしての **templateName**。

222.4. MVEL コンテキスト

Camel は MVEL コンテキスト (単なる **Map**) でエクスチェンジ情報を提供します。**Exchange** は次のように転送されます。

key	value
exchange	**Exchange** 自体。
exchange.properties	**Exchange** プロパティー。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキストのインスタンス。

key	value
request	IN メッセージ
in	IN メッセージ
body	In メッセージボディー
out	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。
response	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。

222.5. ホットリロード

mvel テンプレートリソースは、デフォルトで、ファイルリソースとクラスパスリソース (拡張 jar) の両方でホットリロード可能です。**contentCache=true** を設定すると、Camel はリソースを 1 回だけロードするため、ホットリロードはできません。このシナリオは、リソースが変更されない場合、実稼働環境で使用できます。

222.6. 動的テンプレート

Camel は、テンプレートまたはテンプレートコンテンツ自体の異なるリソースのロケーションを定義できる 2 つのヘッダーを提供します。これらのヘッダーのいずれかが設定されている場合、Camel はエンドポイントで設定されたリソースに対してこれを使用します。これにより、実行時に動的なテンプレートを提供できます。

ヘッダー	タイプ	説明
CamelMvelResourceUri	String	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。
CamelMvelTemplate	String	設定されたエンドポイントの代わりに使用するテンプレート。

222.7. サンプル

たとえば、次のようなものを使用できます

```
from("activemq:My.Queue").
  to("mvel:com/acme/MyResponse.mvel");
```

MVEL テンプレートを使用して、InOut メッセージ交換 (**JMSReplyTo** ヘッダーがある場合) のメッセージへの応答を作成します。

コンポーネントがヘッダーを介して動的に使用するテンプレートを指定するには、次のようにします。

```
from("direct:in").
  setHeader("CamelMvelResourceUri").constant("path/to/my/template.mvel").
  to("mvel:dummy");
```

テンプレートをヘッダーとして直接指定するには、コンポーネントがヘッダーを介して動的に使用する必要があります。たとえば、次のようになります。

```
from("direct:in").
  setHeader("CamelMvelTemplate").constant("@{\"The result is \" + request.body * 3}\" }").
  to("velocity:dummy");
```

222.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第223章 MVEL 言語

Camel バージョン 2.0 以降で利用可能

Camel では、Mvel を DSL または Xml 設定の式または述語として使用できます。

Mvel を使用して、メッセージフィルターで述語を作成したり、受信者リストの式として使用したりできます。

Mvel ドット表記を使用して操作を呼び出すことができます。たとえば、**getFamiliyName** メソッドを持つ POJO を含むボディーがある場合、次のように構文を作成できます。

```
"request.body.familyName"
  // or
"getRequest().getBody().getFamilyName()"
```

223.1. MVEL オプション

MVEL 言語は、以下にリストされている 1 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

223.2. VARIABLES

変数	タイプ	説明
this	Exchange	Exchange はルートオブジェクトです
exchange	Exchange	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)
exchangeId	String	エクスチェンジ ID
fault	メッセージ	Fault メッセージ (ある場合)
request	メッセージ	exchange.in メッセージ

変数	タイプ	説明
response	メッセージ	exchange.out メッセージ (存在する場合)
properties	Map	エクスチェンジプロパティー
property(name)	Object	指定された名前によるプロパティー
property(name, type)	タイプ	指定されたタイプとして指定された名前によるプロパティー

223.3. サンプル

たとえば、XML の メッセージフィルター 内で Mvel を使用できます。

```
<route>
  <from uri="seda:foo"/>
  <filter>
    <mvel>request.headers.foo == 'bar'</mvel>
    <to uri="seda:bar"/>
  </filter>
</route>
```

Java DSL を使用したサンプル:

```
from("seda:foo").filter().mvel("request.headers.foo == 'bar'").to("seda:bar");
```

223.4. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").mvel("resource:classpath:script.mvel")
```

223.5. 依存関係

camel ルートで Mvel を使用するには、Mvel 言語を実装する **camel-mvel** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mvel</artifactId>
  <version>x.x.x</version>
</dependency>
```

第224章 MYBATIS コンポーネント

Camel バージョン 2.7 以降で利用可能

mybatis: コンポーネントを使用すると、MyBatis を使用してリレーショナルデータベースのデータをクエリー、ポーリング、挿入、更新、および削除できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mybatis</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

224.1. URI 形式

```
mybatis:statementName[?options]
```

statementName は、 評価するクエリー、挿入、更新、または削除操作にマップされる MyBatis XML マッピングファイル内のステートメント名です。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

このコンポーネントはデフォルトで、期待される名前 **SqlMapConfig.xml** でクラスパスのルートから MyBatis SqlMapConfig ファイルをロードします。
ファイルが別のロケーションにある場合は、**MyBatisComponent** コンポーネントで **configurationUri** オプションを設定する必要があります。

224.2. オプション

MyBatis コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
sqlSessionFactory (advanced)	SqlSessionFactory を使用する場合。		SqlSessionFactory
configurationUri (common)	MyBatis xml 設定ファイルのロケーション。デフォルト値は次のとおりです。クラスパスからロードされた SqlMapConfig.xml	SqlMapConfig.xml	String
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MyBatis エンドポイントは、URI 構文を使用して設定されます。

```
mybatis:statement
```

パスおよびクエリーパラメーターを使用します。

224.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
statement	**必須** 評価するクエリー、挿入、更新、または削除操作にマップする MyBatis XML マッピングファイル内のステートメント名。		String

224.2.2. クエリーパラメーター (29 個のパラメーター):

名前	説明	デフォルト	タイプ
outputHeader (common)	メッセージ本文ではなく、ヘッダーにクエリー結果を格納します。デフォルトでは、outputHeader == null で、クエリー結果はメッセージ本文に格納され、メッセージ本文の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を格納するヘッダーの名前として使用され、元のメッセージ本文は保持されます。outputHeader を設定すると、常に outputHeader と同じになるため、デフォルトの CamelMyBatisResult ヘッダーへの入力も省略されます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
inputHeader (コンシューマー)	メッセージボディーの代わりに、入力パラメーターのヘッダー値を使用します。デフォルトでは、inputHeader == null であり、入力パラメーターはメッセージボディーから取得されます。outputHeader が設定されている場合、値が使用され、クエリーパラメーターはボディーではなくヘッダーから取得されます。		文字列

名前	説明	デフォルト	タイプ
maxMessagesPerPoll (consumer)	このオプションは、データベースプールから返された結果をバッチに分割し、それらを複数のエクスチェンジで配信することを目的としています。この整数は、1 回のエクスチェンジで配信するメッセージの最大数を定義します。デフォルトでは最大値は設定されていません。たとえば制限を 1000 などに設定して、数千のファイルがあるサーバーの起動を回避できます。無効にするには、0 または負の値を設定します。	0	int
onConsume (consumer)	ルートでデータが処理された後に実行するステートメント		String
routeEmptyResultSet (consumer)	空の結果セットを次のホップにルーティングできるようにするかどうか	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
transacted (consumer)	トランザクションを有効または無効にします。有効にすると、交換の処理が失敗した場合に、コンシューマーはそれ以上の交換の処理を中断して、先行してロールバックを実行させます。	false	boolean
useIterator (consumer)	結果セットを個別に、またはリストとして処理する	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processingStrategy (consumer)	カスタム MyBatisProcessingStrategy を使用するには		MyBatisProcessing Strategy

名前	説明	デフォルト	タイプ
executorType (producer)	ステートメントの実行中に使用されるエグゼキュータのタイプ。simple - エグゼキュータは特別なことは何もしません。reuse - エグゼキュータは準備済みステートメントを再利用します。batch - エグゼキュータはステートメントを再利用し、更新をバッチ処理します。	SIMPLE	ExecutorType
statementType (producer)	プロデューサーが呼び出す操作の種類を制御するために指定する必要があります。		StatementType
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

224.3. メッセージヘッダー

Camel は、IN または OUT のいずれかの結果メッセージに、使用されたステートメントを含むヘッダーを入力します。

ヘッダー	タイプ	説明
CamelMyBatisStatementName	**String**	使用される **statementName** (例: insertAccount)。
CamelMyBatisResult	**Object**	いずれかの操作で MtBatis から返された **レスポンス**。たとえば、**INSERT** は自動生成されたキーや行数などを返すことができます。

224.4. メッセージボディー

MyBatis からのレスポンスは、**SELECT** ステートメントの場合のみボディーとして設定されます。つまり、たとえば **INSERT** ステートメントの場合、Camel はボディーを置き換えません。これにより、ルーティングを続行し、元のボディを維持できます。MyBatis からのレスポンスは、常に **CamelMyBatisResult** キーとともにヘッダーに格納されます。

224.5. サンプル

たとえば、JMS キューから Bean を消費してデータベースに挿入する場合は、次のようにします。

```
from("activemq:queue:newAccount").
  to("mybatis:insertAccount?statementType=Insert");
```

どの種類の操作を呼び出すかを Camel に指示する必要があるため、**statementType** を指定する必要があることに注意してください。

insertAccount は SQL マッピングファイルの MyBatis ID です。

```
<!-- Insert example, using the Account parameter class -->
<insert id="insertAccount" parameterType="Account">
  insert into ACCOUNT (
    ACC_ID,
    ACC_FIRST_NAME,
    ACC_LAST_NAME,
    ACC_EMAIL
  )
  values (
    #{id}, #{firstName}, #{lastName}, #{emailAddress}
  )
</insert>
```

224.6. STATEMENTTYPE を使用して MYBATIS をより適切に制御する

MyBatis エンドポイントにルーティングする場合、実行する SQL ステートメントが **SELECT**、**UPDATE**、**DELETE** または **INSERT** などであるかどうかを制御できるように、よりきめ細かい制御が必要になります。たとえば、IN ボディに **SELECT** ステートメントへのパラメーターが含まれている MyBatis エンドポイントにルーティングする場合は、次のようにします。

上記のコードでは、MyBatis ステートメントの **selectAccountById** を呼び出すことができます。IN 本文には、取得するアカウント ID (**Integer** 型など) が含まれている必要があります。

SelectList など、他のいくつかの操作についても同じことができます。

UPDATE についても同様で、**Account** オブジェクトを IN ボディとして MyBatis に送信できます。

224.6.1. InsertList StatementType の使用

Camel 2.10 以降で利用可能

MyBatis では、for-each バッチドライバーを使用して複数の行を挿入できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。たとえば、次のようになります。

次に、以下に示すように、**InsertList** ステートメントタイプを使用する **mybatis** エンドポイントに Camel メッセージを送信することにより、複数の行を挿入できます。

224.6.2. UpdateList StatementType の使用

Camel 2.11 から利用可能

MyBatis では、for-each バッチドライバーを使用して複数の行を更新できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。たとえば、次のようになります。

```
<update id="batchUpdateAccount" parameterType="java.util.Map">
    update ACCOUNT set
    ACC_EMAIL = #{emailAddress}
    where
    ACC_ID in
    <foreach item="Account" collection="list" open="(" close=")" separator=",">
        #{Account.id}
    </foreach>
</update>
```

次に、以下に示すように、UpdateList ステートメントタイプを使用する mybatis エンドポイントに Camel メッセージを送信することにより、複数の行を更新できます。

```
from("direct:start")
    .to("mybatis:batchUpdateAccount?statementType=UpdateList")
    .to("mock:result");
```

224.6.3. DeleteList StatementType の使用

Camel 2.11 から利用可能

MyBatis では、for-each バッチドライバーを使用して複数の行を削除できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。たとえば、次のようになります。

```
<delete id="batchDeleteAccountById" parameterType="java.util.List">
    delete from ACCOUNT
    where
    ACC_ID in
    <foreach item="AccountID" collection="list" open="(" close=")" separator=",">
        #{AccountID}
    </foreach>
</delete>
```

次に、以下に示すように、DeleteList ステートメントタイプを使用する mybatis エンドポイントに Camel メッセージを送信することにより、複数の行を削除できます。

```
from("direct:start")
    .to("mybatis:batchDeleteAccount?statementType=DeleteList")
    .to("mock:result");
```

224.6.4. InsertList、UpdateList、および DeleteList StatementTypes に関する通知

任意のタイプ (List、Map など) のパラメーターを mybatis に渡すことができ、エンドユーザーは mybatis 動的クエリー 機能の助けを借りて、必要に応じてそれを処理する責任があります
。

224.6.5. スケジュールされたポーリングの例

このコンポーネントはスケジュールされたポーリングをサポートしているため、ポーリングコンシューマーとして使用できます。たとえば、毎分データベースをポーリングするには:

```
from("mybatis:selectAllAccounts?delay=60000").to("activemq:queue:allAccounts");
```

その他のオプションについては、Polling Consumer の ScheduledPollConsumer オプションを参照してください。

あるいは、Timer や Quartz コンポーネントなど、スケジュールされたポーリングをトリガーする別のメカニズムを使用することもできます。 以下のサンプルでは、Timer コンポーネントを使用して 30 秒ごとにデータベースをポーリングし、データを JMS キューに送信します。

```
from("timer://pollTheDatabase?
delay=30000").to("mybatis:selectAllAccounts").to("activemq:queue:allAccounts");
```

そして、使用される MyBatis SQL マッピングファイル:

```
<!-- Select with no parameters using the result map for Account class. -->
<select id="selectAllAccounts" resultMap="AccountResult">
  select * from ACCOUNT
</select>
```

224.6.6. onConsume の使用

このコンポーネントは、データが Camel によって消費および処理された **後の** ステートメントの実行をサポートします。これにより、データベースでポスト更新を行うことができます。すべてのステートメントは **UPDATE** ステートメントでなければならないことに注意してください。Camel は、名前をコンマで区切る必要がある複数のステートメントの実行をサポートしています。

以下のルートは、**consumeAccount** ステートメントを実行してデータが処理されることを示しています。これにより、データベース内の行のステータスを処理済みに変更できるため、2 回以上消費することを回避できます。

そして、sqlmap ファイル内のステートメント:

224.6.7. トランザクションへの参加

camel-mybatis の下でトランザクションマネージャーをセットアップするのは、標準の MyBatis **SqlMapConfig.xml** ファイルの外部にデータベース設定を外部化する必要があるため、少し手間がかかる場合があります。

最初の部分では、**DataSource** のセットアップが必要です。これは通常、Spring プロキシーでラップする必要があるプール (DBCP または c3p0) です。このプロキシーにより、Spring 以外での **DataSource** の使用が Spring トランザクションに参加できるようになります (MyBatis **SqlSessionFactory** はまさにこれを行います)。

```
  <bean id="dataSource"
class="org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy">
```

```
        <constructor-arg>
          <bean class="com.mchange.v2.c3p0.ComboPooledDataSource">
            <property name="driverClass" value="org.postgresql.Driver"/>
            <property name="jdbcUrl" value="jdbc:postgresql://localhost:5432/myDatabase"/>
            <property name="user" value="myUser"/>
            <property name="password" value="myPassword"/>
          </bean>
        </constructor-arg>
    </bean>
```

これには、プロパティープレースホルダーを使用してデータベース設定を外部化できるという追加の利点があります。

次に、最も外側の **DataSource** を管理するようにトランザクションマネージャーを設定します。

```
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

mybatis-spring **SqlSessionFactoryBean** は、同じ **DataSource** をラップします。

```
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
      <property name="dataSource" ref="dataSource"/>
      <!-- standard mybatis config file -->
    <property name="configLocation" value="/META-INF/SqlMapConfig.xml"/>
      <!-- externalised mappers -->
    <property name="mapperLocations" value="classpath*:META-INF/mappers/**/*.xml"/>
    </bean>
```

camel-mybatis コンポーネントは、そのファクトリーで設定されます。

```
    <bean id="mybatis" class="org.apache.camel.component.mybatis.MyBatisComponent">
      <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
    </bean>
```

最後に、トランザクションマネージャーの上にトランザクションポリシーが定義され、通常どおり使用できます。

```
    <bean id="PROPAGATION_REQUIRED"
class="org.apache.camel.spring.spi.SpringTransactionPolicy">
        <property name="transactionManager" ref="txManager"/>
        <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
    </bean>

    <camelContext id="my-model-context" xmlns="http://camel.apache.org/schema/spring">
      <route id="insertModel">
        <from uri="direct:insert"/>
        <transacted ref="PROPAGATION_REQUIRED"/>
        <to uri="mybatis:myModel.insert?statementType=Insert"/>
      </route>
    </camelContext>
```

224.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第225章 NAGIOS コンポーネント

Camel バージョン 2.3 以降で利用可能

Nagios コンポーネントを使用すると、パッシブチェックを Nagios に送信できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-nagios</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

225.1. URI 形式

```
nagios://host[:port][?Options]
```

Camel は、Nagios コンポーネントで 2 つの機能を提供します。エンドポイントにメッセージを送信することで、パッシブチェックメッセージを送信できます。
Camel は、Nagios に通知を送信できるようにする EventNotifer も提供します。

225.2. オプション

Nagios コンポーネントは、以下にリストされている 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 NagiosConfiguration を使用するには		NagiosConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Nagios エンドポイントは、URI 構文を使用して設定されます。

```
nagios:host:port
```

パスおよびクエリーパラメーターを使用します。

225.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** これは、チェックが送信される Nagios ホストのアドレスです。		String
port	**必須** ホストのポート番号。		int

225.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
connectionTimeout (producer)	ミリ秒単位の接続タイムアウト。	5000	int
sendSync (producer)	パッシブチェックを送信するときに同期を使用するかどうか。false に設定すると、Camel はメッセージのルーティングを続行でき、パッシブチェックメッセージは非同期で送信されます。	true	boolean
timeout (producer)	送信タイムアウト (ミリ秒)。	5000	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
encryption (security)	暗号化方式を指定します。		暗号化
encryptionMethod (security)	**非推奨** 暗号化方式を指定する場合。		NagiosEncryptionMethod
password (security)	チェックを Nagios に送信するときに認証されるパスワード。		文字列

225.3. メッセージの送信例

メッセージペイロードにメッセージが含まれる Nagios にメッセージを送信できます。デフォルトでは **OK** レベルで、CamelContext 名をサービス名として使用します。上記のように、ヘッダーを使用してこれらの値を無効にすることができます。

たとえば、次のように **Hello Nagios** メッセージを Nagios に送信します。

```
template.sendBody("direct:start", "Hello Nagios");

from("direct:start").to("nagios:127.0.0.1:5667?password=secret").to("mock:result");
```

CRITICAL メッセージを送信するには、次のようなヘッダーを送信できます。

```
Map headers = new HashMap();
headers.put(NagiosConstants.LEVEL, "CRITICAL");
headers.put(NagiosConstants.HOST_NAME, "myHost");
headers.put(NagiosConstants.SERVICE_NAME, "myService");
template.sendBodyAndHeaders("direct:start", "Hello Nagios", headers);
```

225.4. NAGIOSEVENTNOTIFER の使用

Nagios コンポーネントは、イベントを Nagios に送信するために使用できる EventNotifer も提供します。たとえば、次のように Java からこれを有効にできます。

```
NagiosEventNotifier notifier = new NagiosEventNotifier();
notifier.getConfiguration().setHost("localhost");
notifier.getConfiguration().setPort(5667);
notifier.getConfiguration().setPassword("password");

CamelContext context = ...
context.getManagementStrategy().addEventNotifier(notifier);
return context;
```

Spring XML では、タイプ **EventNotifier** で Spring Bean を定義するだけで、Camel はここに記載されているようにそれを取得します: Spring を使用した CamelContext の高度な設定。

225.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第226章 NATS コンポーネント

Camel バージョン 2.17 以降で利用可能

NATS は、高速で信頼性の高いメッセージングプラットフォームです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-nats</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

226.1. URI 形式

```
nats:servers[?options]
```

ここで、**servers** は NATS サーバーのリストを表します。

226.2. オプション

Nats コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Nats エンドポイントは、URI 構文を使用して設定されます。

```
nats:servers
```

パスおよびクエリーパラメーターを使用します。

226.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
servers	**必須** 1 つ以上の NAT サーバーへの URL。複数のサーバーを指定する場合は、コンマを使用して URL を区切ります。		文字列

226.2.2. クエリーパラメーター(22 個のパラメーター):

名前	説明	デフォルト	タイプ
flushConnection (common)	接続をフラッシュするかどうかを定義します	false	boolean
flushTimeout (common)	フラッシュタイムアウトを設定する	1000	int
maxReconnectAttempts (common)	最大再接続試行回数	3	int
noRandomizeServers (common)	接続試行のためにサーバーの順序をランダム化するかどうか	false	boolean
pedantic (common)	ペダンティックモードで実行するかどうか (これはパフォーマンスに影響します)	false	boolean
pingInterval (common)	接続がまだ有効かどうかを確認するための ping 間隔 (ミリ秒単位)	4000	int
reconnect (common)	再接続機能の有無	true	boolean
reconnectTimeWait (common)	再接続を試みるまでの待機時間 (ミリ秒単位)	2000	int
topic (common)	**必須** 使用したいトピックの名前		String
verbose (common)	詳細モードで実行しているかどうか	false	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessages (consumer)	maxMessages 後にサブスクライブしているトピックからのメッセージの受信を停止します		String
poolSize (consumer)	コンシューマープールのサイズ	10	int
queueName (consumer)	キュー設定に nats を使用している場合のキュー名		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
replySubject (producer)	サブスクライバーがレスポンスを送信する対象		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
secure (security)	TLS が必要であることを示すセキュアオプションを設定する	false	boolean
ssl (security)	SSL を使用するかどうか	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
tlsDebug (security)	TLS デバッグ。追加のコンソール出力が追加されます	false	boolean

226.3. ヘッダー

名前	タイプ	説明
CamelNatsMessageTimestamp	long	消費されたメッセージのタイムスタンプ。
CamelNatsSubscriptionId	Integer	コンシューマーのサブスクリプション ID。

プロデューサーの例:

```
from("direct:send").to("nats://localhost:4222?topic=test");
```

コンシューマーの例:

```
from("nats://localhost:4222?topic=test&maxMessages=5&queueName=test").to("mock:result");
```

第227章 NETTY コンポーネント (非推奨)

Camel バージョン 2.3 以降で利用可能

警告

このコンポーネントは非推奨です。Netty4 を 使用する必要があります。

Camel の **netty** コンポーネントは、Netty プロジェクトに基づくソケット通信コンポーネントです。

Netty は、プロトコルサーバーやクライアントなどのネットワークアプリケーションの迅速かつ簡単な開発を可能にする NIO クライアントサーバーフレームワークです。
Netty は、TCP や UDP ソケットサーバーなどのネットワークプログラミングを大幅に簡素化および合理化します。

この camel コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方をサポートします。

Netty コンポーネントにはいくつかのオプションがあり、多数の TCP/UDP 通信パラメーター (バッファーサイズ、keepAlive、tcpNoDelay など) をきめ細かく制御し、Camel ルートでの In-Only 通信と In-Out 通信の両方を容易にします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

227.1. URI 形式

netty コンポーネントの URI スキームは次のとおりです。

```
netty:tcp://localhost:99999[?options]
netty:udp://remotehost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーエンドポイントとコンシューマーエンドポイントをサポートします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

227.2. オプション

Netty コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	エンドポイントの作成時に NettyConfiguration を設定として使用するには。		NettyConfiguration
maximumPoolSize (advanced)	順序付けられたスレッドプールのコアプールサイズ (使用中の場合)。デフォルト値は 16 です。	16	int
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty エンドポイントは、URI 構文を使用して設定されます。

```
netty:protocol:host:port
```

パスおよびクエリーパラメーターを使用します。

227.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
protocol	**必須** tcp または udp の使用するプロトコル。		String
host	**必須** ホスト名。コンシューマーの場合、ホスト名は localhost または 0.0.0.0 ですプロデューサーの場合、ホスト名は接続先のリモートホストです		String
port	**必須** ホストのポート番号		int

227.2.2. クエリーパラメーター(67 個のパラメーター):

名前	説明	デフォルト	タイプ
disconnect (Common)	使用直後に Netty Channel を切断 (クローズ) するかどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean

名前	説明	デフォルト	タイプ
keepAlive (common)	非アクティブのためにソケットが閉じられないようにするための設定	true	boolean
reuseAddress (Common)	ソケットの多重化を容易にするための設定	true	boolean
sync (common)	エンドポイントを一方向または要求応答として設定する設定	true	boolean
tcpNoDelay (Common)	TCP プロトコルのパフォーマンスを向上させるための設定	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
broadcast (consumer)	UDP 経由のマルチキャストを選択するための設定	false	boolean
clientMode (consumer)	clientMode が true の場合、netty コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean
backlog (consumer)	netty コンシューマー (サーバー) のバックログを設定できます。バックログは、OS によってはベストエフォートであることに注意してください。このオプションを 200、500、1000 などの値に設定すると、TCP スタックに受け入れキューの長さが通知されます。このオプションが設定されていない場合、バックログは OS の設定に依存します。		int
bossCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの BossCount パラメーターである 1 を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの BossCount をオーバーライドできます	1	int
bossPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.BossPool を Boss スレッドプールとして使用します。たとえば、スレッドプールを複数のコンシューマーと共有する場合などです。デフォルトでは、各コンシューマーには 1 つのコアスレッドを持つ独自の Boss プールがあります。		BossPool

名前	説明	デフォルト	タイプ
channelGroup (consumer)	明示的な ChannelGroup を使用するには。		ChannelGroup
disconnectOnNoReply (consumer)	同期が有効になっている場合、このオプションは、返信がない場合に NettyConsumer を切断するかどうかを指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
maxChannelMemorySize (consumer)	OrderedThreadPoolExecutor を使用する場合の、チャネルごとのキューに入れられたイベントの最大合計サイズ。無効にする場合は 0 を指定します。	1048576	long
maxTotalMemorySize (consumer)	このプールのキューに入れられたイベントの最大合計サイズ (orderedThreadPoolExecutor を使用する場合)。無効にする場合は 0 を指定します。	209715200	long
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
networkInterface (consumer)	UDP を使用する場合、このオプションを使用してネットワークインターフェイスをその名前で指定できます (マルチキャストグループに参加するための eth0 など)。		String
noReplyLogLevel (consumer)	同期が有効になっている場合、このオプションは NettyConsumer がログに記録するときに使用するログレベルを決定し、返信する応答がありません。	WARN	LoggingLevel
orderedThreadPoolExecutor (consumer)	順序付けられたスレッドプールを使用して、イベントが同じチャネルで順番に処理されるかどうか。詳細については、org.jboss.netty.handler.execution.OrderedMemoryAwareThreadPoolExecutor の netty javadoc を参照してください。	true	boolean

名前	説明	デフォルト	タイプ
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー (NettyConsumer) が java.nio.channels.ClosedChannelException をキャッチすると、このログレベルを使用してログに記録されます。これは、クローズドチャネル例外のログ記録を回避するために使用されます。これは、クライアントが突然切断され、Netty サーバーでクローズド例外のフラッドが発生する可能性があるためです。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー (NettyConsumer) が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverPipelineFactory (consumer)	カスタム ServerPipelineFactory を使用する場合。		ServerPipelineFactory
workerCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの workerCount パラメーター (cpu_core_threads2) を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの workerCount をオーバーライドできます。		int
workerPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.WorkerPool をワーカースレッドプールとして使用する場合。たとえば、スレッドプールを複数のコンシューマーと共有する場合などです。デフォルトでは、各コンシューマーには、2 x CPU カウントのコアスレッドを備えた独自のワーカープールがあります。		WorkerPool
connectTimeout (producer)	ソケット接続が使用可能になるまで待機する時間。値はミリ単位です。	10000	long
requestTimeout (producer)	リモートサーバーを呼び出すときに、Netty プロデューサーのタイムアウトを使用できるようにします。デフォルトでは、タイムアウトは使用されていません。値はミリ秒単位なので、たとえば 30000 は 30 秒です。requestTimeout は、Netty の ReadTimeoutHandler を使用してタイムアウトをトリガーしています。		long
clientPipelineFactory (producer)	カスタム ClientPipelineFactory を使用する場合。		ClientPipelineFactory
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにチャネルを遅延作成できます。	true	boolean

名前	説明	デフォルト	タイプ
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要: これをオフにしないでください。並行性と信頼できるリクエスト/レスポンスを処理するためにプーリングが必要です。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当てられる (クライアントにチェックアウトされるか、チェックアウトを待機するアイドル) オブジェクトの数に上限を設定します。無制限の場合は負の値を使用します。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンス数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドル状態のオブジェクト Evictor によるエビクションの対象となる前に、オブジェクトがプール内でアイドル状態になる最小時間 (ミリ単位の値) を設定します。	300000	long
producerPoolMinIdle (producer)	evictor スレッド (アクティブな場合) が新しいオブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
udpConnectionlessSending (producer)	このオプションは接続のない UDP 送信をサポートします。接続された udp send は、受信ポートでリッスンしている人がいない場合、PortUnreachableException を受け取ります。	false	boolean
useChannelBuffer (producer)	useChannelBuffer が true の場合、netty プロデューサーはメッセージボディーを送信前に ChannelBuffer に変換します。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyServerBootstrapConfiguration を使用するには。		NettyServerBootstrap 設定
options (advanced)	オプションを使用して、追加の netty オプションを設定できます。接頭辞として。たとえば、netty オプション child.keepAlive=false を設定するには、option.child.keepAlive=false とします。使用可能なオプションについては、Netty のドキュメントを参照してください。		Map
receiveBufferSize (advanced)	インバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	long
receiveBufferSizePredictor (advanced)	バッファーサイズプレディクターを設定します。詳細は、Jetty のドキュメントとこのメールスレッドを参照してください。		int

名前	説明	デフォルト	タイプ
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク経由でエクスチェンジを転送することができます。In body、Out body、fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、exchange プロパティー、exchange 例外フィールドが転送されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
allowDefaultCodec (codec)	netty コンポーネントは、encoder/deocder が null で textline が false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、netty コンポーネントがデフォルトコーデックをフィルターチェーンの最初の要素としてインストールするのを防ぎます。	true	boolean
autoAppendDelimiter (codec)	テキストラインコーデックを使用して送信するときに、不足している終了区切り文字を自動追加するかどうか。	true	boolean
decoder (codec)	**非推奨** 受信ペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。org.jboss.netty.channel.ChannelUpStreamHandler をオーバーライドする必要があります。		ChannelHandler
decoderMaxLineLength (codec)	テキストラインコーデックに使用する行の最大長。	1024	int
decoders (codec)	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
delimiter (codec)	テキストラインコーデックに使用する区切り文字。可能な値は LINE と NULL です。	LINE	TextLineDelimiter

名前	説明	デフォルト	タイプ
encoder (codec)	**非推奨** 送信ペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。Must override org.jboss.netty.channel.ChannelDownStreamHandler.		ChannelHandler
encoders (codec)	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
encoding (codec)	テキスト行コーデックに使用するエンコーディング (文字セット名)。指定しない場合、Camel は JVM のデフォルトの文字セットを使用します。		String
textline (codec)	TCP にのみ使用されます。コーデックが指定されていない場合、このフラグを使用して、テキスト行ベースのコーデックを示すことができます。指定されていない場合、または値が false の場合、オブジェクトのシリアル化は TCP 経由であると想定されます。	false	boolean
enabledProtocols (security)	SSL を使用するときに有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		File
keyStoreFormat (security)	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS	JKS	String
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
パスフレーズ (security)	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定		String
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。	SunX509	String

名前	説明	デフォルト	タイプ
ssl (security)	このエンドポイントに SSL 暗号化を適用するかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効で SSL モードの場合、Netty コンシューマーは、サブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を含むヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		File
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String

227.3. レジストリーベースのオプション

コーデックハンドラーと SSL キーストアは、Spring XML ファイルなどのレジストリーに登録できます。渡すことができる値は次のとおりです。

名前	説明
passphrase	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定
keyStoreFormat	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS
securityProvider	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。
keyStoreFile	**非推奨:** 暗号化に使用されるクライアント側の証明書キーストア

名前	説明
trustStoreFile	**非推奨:** 暗号化に使用されるサーバー側の証明書キーストア
keyStoreResource	Camel 2.11.1: 暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、**"classpath:"**、**"file:"**、または **"http:"** をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。
trustStoreResource	Camel 2.11.1: 暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、**"classpath:"**、**"file:"**、または **"http:"** をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。
sslHandler	SSL ハンドラーを返すために使用できるクラスへの参照
encoder	送信ペイロードの特別なマーシャリングを実行するために使用できるカスタム **ChannelHandler** クラス。**org.jboss.netty.channel.ChannelDownStreamHandler** をオーバーライドする必要があります。
encorders	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に # を付けることを忘れないでください。
decoder	受信ペイロードの特別なマーシャリングを実行するために使用できるカスタム **ChannelHandler** クラス。**org.jboss.netty.channel.ChannelUpStreamHandler** をオーバーライドする必要があります。
decoders	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に # を付けることを忘れないでください。

重要: 共有不可能なエンコーダー/デコーダーの使用については、以下をお読みください。

227.3.1. 共有不可能なエンコーダーまたはデコーダーの使用

エンコーダーまたはデコーダーが共有可能でない場合 (たとえば、@Shareable クラスアノテーションがある場合)、エンコーダー/デコーダーは **org.apache.camel.component.netty.ChannelHandlerFactory** インターフェイスを実装し、**newChannelHandler** メソッドで新しいインスタンスを返さなければなりません。これは、エンコーダー/デコーダーを安全に使用できるようにするためです。そうでない場合、Netty コンポーネントは、エンドポイントの作成時
に WARN をログに記録します。

Netty コンポーネントは、多くの一般的に使用されるメソッドを持つ **org.apache.camel.component.netty.ChannelHandlerFactories** ファクトリークラスを提供します。

227.4. NETTY エンドポイントとの間でメッセージを送信する

227.4.1. Netty プロデューサー

Producer モードでは、コンポーネントは、TCP または UDP プロトコル (オプションの SSL サポート付き) を使用してペイロードをソケットエンドポイント
に送信する機能を提供します。

プロデューサーモードは、一方向および要求/応答ベースの操作の両方をサポートします。

227.4.2. Netty コンシューマー

コンシューマーモードでは、コンポーネントは次の機能を提供します。

- TCP または UDP プロトコル (オプションの SSL サポート付き) を使用して、指定されたソケットでリッスンします。
- text/xml、バイナリーおよびシリアライズされたオブジェクトベースのペイロードを使用してソケットでリクエストを受信し、
- メッセージ交換としてルートに沿ってそれらを送信します。

コンシューマーモードは、一方向および要求/応答ベースの操作の両方をサポートします。

227.5. ヘッダー

次のヘッダーは、Netty コンシューマーによって作成されたエクスチェンジ用に入力されます。

ヘッダーのキー	Class	説明
NettyConstants.NETTY_CHANNEL_HANDLER_CONTEXT / CamelNettyChannelHandlerContext	**org.jboss.netty.channel.ChannelHandlerContext**	`ChannelHandlerContext` netty が受信した接続に関連付けられたインスタンス。

ヘッダーのキー	Class	説明
NettyConstants.NETTY_MESSAGE_EVENT / CamelNettyMessageEvent	**org.jboss.netty.channel.MessageEvent**	netty が受信した接続に関連付けられた `MessageEvent ` インスタンス。
NettyConstants.NETTY_REMOTE_ADDRESS / CamelNettyRemoteAddress	**java.net.SocketAddress**	入力ソケット接続のリモートアドレス。
NettyConstants.NETTY_LOCAL_ADDRESS / CamelNettyLocalAddress	**java.net.SocketAddress**	入力ソケット接続のローカルアドレス。

227.6. 使用例

227.6.1. Request-Reply とシリアライズされたオブジェクトペイロードを使用する UDP Netty エンドポイント

```
RouteBuilder builder = new RouteBuilder() {
```

```
  public void configure() {
    from("netty:udp://localhost:5155?sync=true")
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Poetry poetry = (Poetry) exchange.getIn().getBody();
          poetry.setPoet("Dr. Sarojini Naidu");
          exchange.getOut().setBody(poetry);
        }
      }
    }
};
```

227.6.2. 一方向通信を使用する TCP ベースの Netty コンシューマーエンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
      from("netty:tcp://localhost:5150")
        .to("mock:result");
  }
};
```

227.6.3. Request-Reply 通信を使用する SSL/TCP ベースの Netty コンシューマーエンドポイント

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Netty コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、Netty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

NettyComponent nettyComponent = getContext().getComponent("netty", NettyComponent.class);
nettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
```

```
      <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="netty:tcp://localhost:5150?
sync=true&ssl=true&sslContextParameters=#sslContextParameters"/>
...
```

Jetty コンポーネントでの基本的な SSL/TLS 設定の使用

```
JndiRegistry registry = new JndiRegistry(createJndiContext());
registry.bind("password", "changeit");
registry.bind("ksf", new File("src/test/resources/keystore.jks"));
registry.bind("tsf", new File("src/test/resources/keystore.jks"));

context.createRegistry(registry);
context.addRoutes(new RouteBuilder() {
  public void configure() {
      String netty_ssl_endpoint =
        "netty:tcp://localhost:5150?sync=true&ssl=true&passphrase=#password"
        + "&keyStoreFile=#ksf&trustStoreFile=#tsf";
      String return_string =
        "When You Go Home, Tell Them Of Us And Say,"
        + "For Your Tomorrow, We Gave Our Today.";

      from(netty_ssl_endpoint)
       .process(new Processor() {
          public void process(Exchange exchange) throws Exception {
            exchange.getOut().setBody(return_string);
          }
       }
  }
});
```

SSLSession とクライアント証明書へのアクセスを取得する

Camel 2.12 以降で利用可能

たとえば、クライアント証明書に関する詳細を取得する必要がある場合は、**javax.net.ssl.SSLSession** にアクセスできます。**ssl=true** の場合、以下に示すように、Netty コンポーネントは **SSLSession** を Camel メッセージのヘッダーとして保存します。

```
SSLSession session = exchange.getIn().getHeader(NettyConstants.NETTY_SSL_SESSION,
SSLSession.class);
// get the first certificate which is client certificate
javax.security.cert.X509Certificate cert = session.getPeerCertificateChain()[0];
Principal principal = cert.getSubjectDN();
```

クライアントを認証するには、必ず **needClientAuth=true** を設定してください。そうしないと、**SSLSession** はクライアント証明書に関する情報にアクセスできず、例外 **javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated** が発生する可能性があります。クライアント証明書の有効期限が切れているか、有効でない場合などにも、この例外が発生する可能性があります。

ヒント

オプション **sslClientCertHeaders をtrue** に設定すると、Camel メッセージがクライアント証明書に関する詳細を含むヘッダーで強化されます。たとえば、サブジェクト名はヘッダー **CamelNettySSLClientCertSubjectName** ですぐに利用できます。

227.6.4. 複数のコーデックの使用

場合によっては、エンコーダーとデコーダーのチェーンを netty パイプラインに追加する必要があります。複数のコーデックを camel netty エンドポイントに追加するには、'encoders' および 'decoders' uri パラメーターを使用する必要があります。encoder および decoder パラメーターと同様に、パイプラインに追加する必要がある (ChannelUpstreamHandlers および ChannelDownstreamHandlers のリストへの) 参照を提供するために使用されます。エンコーダーが指定されている場合、デコーダーとデコーダーのパラメーターと同様に、エンコーダーのパラメーターは無視されることに注意してください。

情報: 共有不可能なエンコーダー/デコーダーの使用については、上記をお読みください。

エンドポイントの作成時に解決できるように、コーデックのリストを Camel のレジストリーに追加する必要があります。

Spring のネイティブコレクションサポートを使用して、アプリケーションコンテキストでコーデックリストを指定できます。

Bean 名は、netty エンドポイント定義でコンマ区切りのリストとして使用するか、リストに含めることができます。

または Spring 経由。

227.7. 完了時にチャネルを閉じる

サーバーとして機能している場合、たとえばクライアントの変換が終了したときにチャネルを閉じたい場合があります。
これは、エンドポイントオプションの **disconnect=true** を設定するだけで実行できます。

ただし、次のようにメッセージごとに Camel に指示することもできます。
Camel にチャネルを閉じるように指示するには、キー **CamelNettyCloseChannelWhenComplete** をブール値 **true** に設定したヘッダーを追加する必要があります。
たとえば、次の例では、bye メッセージをクライアントに書き戻した後にチャネルを閉じます。

```
        from("netty:tcp://localhost:8080").process(new Processor() {
            public void process(Exchange exchange) throws Exception {
                String body = exchange.getIn().getBody(String.class);
                exchange.getOut().setBody("Bye " + body);
                // some condition which determines if we should close
                if (close) {

exchange.getOut().setHeader(NettyConstants.NETTY_CLOSE_CHANNEL_WHEN_COMPLETE,
true);
                }
            }
        });
```

227.8. カスタムチャネルパイプラインファクトリーを追加して、作成されたパイプラインを完全に制御します

Camel 2.5 で利用可能

カスタムチャネルパイプラインは、Netty エンドポイント URL で非常に簡単な方法で指定することなく、カスタムハンドラー、エンコーダー、デコーダーを挿入することで、ハンドラー/インターセプターチェーンを完全に制御します。

カスタムパイプラインを追加するには、カスタムチャネルパイプラインファクトリーを作成し、コンテキストレジストリー (JNDIRegistry、または camel-spring ApplicationContextRegistry など) を介してコンテキストに登録する必要があります。

カスタムパイプラインファクトリーは、次のように構築する必要があります。

- プロデューサーにリンクされたチャネルパイプラインファクトリーは、抽象クラス **ClientPipelineFactory** を拡張する必要があります。
- コンシューマーリンクチャネルパイプラインファクトリーは、抽象クラス **ServerPipelineFactory** を拡張する必要があります。
- カスタムハンドラー、エンコーダ、およびデコーダを挿入するには、クラスで getPipeline() メソッドをオーバーライドする必要があります。getPipeline() メソッドをオーバーライドしないと、パイプラインに接続されたハンドラー、エンコーダー、またはデコーダーのないパイプラインが作成されます。

以下の例は、ServerChannel パイプラインファクトリーを作成する方法を示しています。

カスタムパイプラインファクトリーの使用

```
public class SampleServerChannelPipelineFactory extends ServerPipelineFactory {
    private int maxLineSize = 1024;

    public ChannelPipeline getPipeline() throws Exception {
        ChannelPipeline channelPipeline = Channels.pipeline();

        channelPipeline.addLast("encoder-SD", new StringEncoder(CharsetUtil.UTF_8));
        channelPipeline.addLast("decoder-DELIM", new DelimiterBasedFrameDecoder(maxLineSize,
true, Delimiters.lineDelimiter()));
        channelPipeline.addLast("decoder-SD", new StringDecoder(CharsetUtil.UTF_8));
        // here we add the default Camel ServerChannelHandler for the consumer, to allow Camel to
route the message etc.
        channelPipeline.addLast("handler", new ServerChannelHandler(consumer));

        return channelPipeline;
    }
}
```

次に、カスタムチャネルパイプラインファクトリーをレジストリーに追加し、次の方法でキャメルルートでインスタンス化/利用できます。

```
Registry registry = camelContext.getRegistry();
serverPipelineFactory = new TestServerChannelPipelineFactory();
registry.bind("spf", serverPipelineFactory);
context.addRoutes(new RouteBuilder() {
  public void configure() {
      String netty_ssl_endpoint =
        "netty:tcp://localhost:5150?serverPipelineFactory=#spf"
      String return_string =
```

```
        "When You Go Home, Tell Them Of Us And Say,"
        + "For Your Tomorrow, We Gave Our Today.";

      from(netty_ssl_endpoint)
       .process(new Processor() {
         public void process(Exchange exchange) throws Exception {
           exchange.getOut().setBody(return_string);
         }
       }
  }
});
```

227.9. NETTY ボスおよびワーカースレッドプールの再利用

Camel 2.12 以降で利用可能

Netty には、ボスとワーカーの 2 種類のスレッドプールがあります。デフォルトでは、各 Netty コンシューマーとプロデューサーにはプライベートスレッドプールがあります。複数のコンシューマーまたはプロデューサー間でこれらのスレッドプールを再利用する場合は、スレッドプールを作成してレジストリーに登録する必要があります。

たとえば、Spring XML を使用すると、以下に示すように、2 つのワーカースレッドを持つ **NettyWorkerPoolBuilder** を使用して共有ワーカースレッドプールを作成できます。

```
<!-- use the worker pool builder to help create the shared thread pool -->
<bean id="poolBuilder" class="org.apache.camel.component.netty.NettyWorkerPoolBuilder">
  <property name="workerCount" value="2"/>
</bean>

<!-- the shared worker thread pool -->
<bean id="sharedPool" class="org.jboss.netty.channel.socket.nio.WorkerPool"
      factory-bean="poolBuilder" factory-method="build" destroy-method="shutdown">
</bean>
```

ヒント

Boss スレッドプールには、Netty コンシューマー用の **org.apache.camel.component.netty.NettyServerBossPoolBuilder** ビルダーと、Netty プロデューサー用の **org.apache.camel.component.netty.NettyClientBossPoolBuilder** があります。

次に、Camel ルートで、以下に示すように URI で **workerPool** オプションを設定することにより、このワーカープールを参照できます。

```
    <route>
      <from uri="netty:tcp://localhost:5021?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;orderedThreadPoolExecutor=false"
/>
      <to uri="log:result"/>
      ...
    </route>
```

別のルートがある場合は、共有ワーカープールを参照できます。

```
<route>
  <from uri="netty:tcp://localhost:5022?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;orderedThreadPoolExecutor=false"
/>
  <to uri="log:result"/>
  ...
</route>
```

i. などがあります。

227.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Netty HTTP
- MINA

第228章 NETTY HTTP コンポーネント (非推奨)

Camel バージョン 2.12 以降で利用可能

netty-http コンポーネントは、Netty による HTTP トランスポートを容易にする Netty コンポーネントの拡張機能です。

この camel コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方をサポートします。

警告

このコンポーネントは非推奨です。Netty4 HTTP を使用する必要があります。

情報: **ストリーム**。Netty はストリームベースです。つまり、受信した入力はストリームとして Camel に送信されます。つまり、ストリームのコンテンツを **一度** だけ読み取ることができます。もし、メッセージボディーが空のように見える場合や、何度もデータにアクセスする必要がある場合 (例: マルチキャストや再配送エラー処理)、ストリームキャッシュを使用するか、何度再読み込みしても安全な **String** にメッセージボディーを変換する必要があります。Netty4 HTTP は、**io.netty.handler.codec.http.HttpObjectAggregator** を使用してストリーム全体をメモリーに読み込み、完全な http メッセージ全体をビルドすることに注意してください。ただし、結果のメッセージは、一度読み取り可能なストリームベースのメッセージのままです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

228.1. URI 形式

netty コンポーネントの URI スキームは次のとおりです。

```
netty-http:http://localhost:8080[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

情報: **クエリーパラメーターとエンドポイントオプション**。Camel が URI クエリーパラメーターとエンドポイントオプションをどのように認識するのか疑問に思われるかもしれません。たとえば、次のようにエンドポイント URI を作成できます - **netty-http:http//example.com?myParam=myValue&compression=true**。この例では、**myParam** がHTTP パラメーターであり、**compression** が Camel エンドポイントオプションです。このような状況で Camel が使用するストラテジーは、利用可能なエンドポイントオプションを解決し、それらを URI から削除することです。これは、前述の例の場合、**圧縮** エンドポイントオプションが解決され、ターゲット URL から削除されるため、Netty HTTP プロデューサーによってエンドポイントに送信される HTTP リクエストが **http//example.com?myParam=myValue** のようになることを意味します。また、動的ヘッダー

(**CamelHttpQuery** など) を使用してエンドポイントオプションを指定できないことにも注意してください。エンドポイントオプションは、エンドポイント URI 定義レベル (DSL 要素 **to** または **from** など) でのみ指定できます。

228.2. HTTP オプション

情報: **さらに多くのオプションがあります。重要:** このコンポーネントは、Netty からすべてのオプションを継承します。したがって、Netty のドキュメントも参照してください。
UDP トランスポートに関連するオプションなど、この Netty HTTP コンポーネントを使用する場合、Netty の一部のオプションは適用されないことに注意してください。

Netty HTTP コンポーネントは、以下に示す 7 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
nettyHttpBinding (advanced)	カスタム org.apache.camel.component.netty.http.NettyHttpBinding を使用して、Netty および Camel Message API との間でバインドします。		NettyHttpBinding
configuration (common)	エンドポイントの作成時に NettyConfiguration を設定として使用するには。		NettyHttpConfiguration
headerFilterStrategy (advanced)	カスタム org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルタリングするには。		HeaderFilterStrategy
securityConfiguration (security)	安全な Web リソースを設定するための org.apache.camel.component.netty.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurity 設定
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
maximumPoolSize (advanced)	順序付けられたスレッドプールのコアプールサイズ (使用中の場合)。デフォルト値は 16 です。	16	int
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty HTTP エンドポイントは、URI 構文を使用して設定されます。

```
netty-http:protocol:host:port/path
```

パスおよびクエリーパラメーターを使用します。

228.2.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
protocol	**必須** 使用するプロトコル (http または https)		String
host	**必須** localhost などのローカルホスト名、またはコンシューマーの場合は 0.0.0.0。プロデューサーを使用する場合のリモート HTTP サーバーのホスト名。		String
port	ホストのポート番号。		int
path	リソースパス		String

228.2.2. クエリーパラメーター(78 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeEndpoint (common)	オプションが true の場合、プロデューサーは Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI をリクエストに使用します。また、throwExceptionOnFailure を false に設定して、プロデューサーがすべての障害応答を送り返すようにすることもできます。ブリッジモードで動作するコンシューマーは、gzip 圧縮と WWW URL フォームエンコーディングをスキップします (消費される Exchange に Exchange.SKIP_GZIP_ENCODING と Exchange.SKIP_WWW_FORM_URLENCODED ヘッダーを追加することで行います)。	false	boolean
disconnect (Common)	使用直後に Netty Channel を切断 (クローズ) するかどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためにソケットが閉じられないようにするための設定	true	boolean
reuseAddress (Common)	ソケットの多重化を容易にするための設定	true	boolean
sync (common)	エンドポイントを一方向または要求応答として設定する設定	true	boolean
tcpNoDelay (Common)	TCP プロトコルのパフォーマンスを向上させるための設定	true	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、Camel が URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
send503whenSuspended (consumer)	コンシューマーが中断されたときに HTTP ステータスコード 503 を返すかどうか。このオプションが false の場合、コンシューマーが中断されたときに Netty Acceptor がバインド解除されるため、クライアントはそれ以上接続できません。	true	boolean
backlog (consumer)	netty コンシューマー (サーバー) のバックログを設定できます。バックログは、OS によってはベストエフォートであることに注意してください。このオプションを 200、500、1000 などの値に設定すると、TCP スタックに受け入れキューの長さが通知されます。このオプションが設定されていない場合、バックログは OS の設定に依存します。		int
bossCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの BossCount パラメーターである 1 を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの BossCount をオーバーライドできます	1	int
bossPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.BossPool を Boss スレッドプールとして使用します。たとえば、スレッドプールを複数のコンシューマーと共有する場合などです。デフォルトでは、各コンシューマーには 1 つのコアスレッドを持つ独自の Boss プールがあります。		BossPool
channelGroup (consumer)	明示的な ChannelGroup を使用するには。		ChannelGroup
chunkedMaxContentLength (consumer)	Netty HTTP サーバーで受信したチャンクフレームごとのコンテンツの最大長 (バイト単位) の値。	1048576	int

名前	説明	デフォルト	タイプ
compression (consumer)	クライアントが HTTP ヘッダーからサポートしている場合、Netty HTTP サーバーでの圧縮に gzip/deflate の使用を許可します。	false	boolean
disableStreamCache (consumer)	Netty HttpRequestgetContent() からの未加工の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームを軽量メモリーベースのストリームキャッシュに読み込みます)。デフォルトでは、Camel は Netty 入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。このオプションを有効にすると、そのままでは Netty ストリームを複数回読み取ることができず、Netty raw ストリームのリーダーインデックスを手動でリセットする必要があることに注意してください。	false	boolean
disconnectOnNoReply (consumer)	同期が有効になっている場合、このオプションは、返信がない場合に NettyConsumer を切断するかどうかを指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpMethodRestrict (consumer)	Netty HTTP コンシューマーで HTTP メソッドを無効にします。コンマ区切りで複数指定できます。		String
mapHeaders (consumer)	このオプションを有効にすると、Netty から Camel Message へのバインド中にヘッダーもマッピングされます (たとえば、ヘッダーとして Camel Message にも追加されます)。このオプションをオフにして、これを無効にすることができます。ヘッダーは、Netty HTTP リクエスト org.jboss.netty.handler.codec.http.HttpRequest インスタンスを返すメソッド getHttpRequest() を使用して、org.apache.camel.component.netty.http.NettyHttpMessage メッセージから引き続きアクセスできます。	true	boolean

名前	説明	デフォルト	タイプ
maxChannelMemorySize (consumer)	OrderedThreadPoolExecutor を使用する場合の、チャネルごとのキューに入れられたイベントの最大合計サイズ。無効にする場合は 0 を指定します。	10485760	long
maxHeaderSize (consumer)	すべてのヘッダーの最大長。各ヘッダーの長さの合計がこの値を超えると、TooLongFrameException が発生します。	8192	int
maxTotalMemorySize (consumer)	このプールのキューに入れられたイベントの最大合計サイズ (orderedThreadPoolExecutor を使用する場合)。無効にする場合は 0 を指定します。	209715200	long
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
nettySharedHttpServer (consumer)	共有 Netty HTTP サーバーを使用するには。詳細については、Netty HTTP サーバーの例を参照してください。		NettySharedHttpServer
noReplyLogLevel (consumer)	同期が有効になっている場合、このオプションは NettyConsumer がログに記録するときに使用するログレベルを決定し、返信する応答がありません。	WARN	LoggingLevel
orderedThreadPoolExecutor (consumer)	順序付けられたスレッドプールを使用して、イベントが同じチャネルで順番に処理されるかどうか。詳細については、org.jboss.netty.handler.execution.OrderedMemoryAwareThreadPoolExecutor の netty javadoc を参照してください。	true	boolean
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー (NettyConsumer) が java.nio.channels.ClosedChannelException をキャッチすると、このログレベルを使用してログに記録されます。これは、クローズドチャネル例外のログ記録を回避するために使用されます。これは、クライアントが突然切断され、Netty サーバーでクローズド例外のフラッドが発生する可能性があるためです。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー (NettyConsumer) が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverPipelineFactory (consumer)	カスタム ServerPipelineFactory を使用する場合。		ServerPipelineFactory

名前	説明	デフォルト	タイプ
traceEnabled (consumer)	この Netty HTTP コンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
urlDecodeHeaders (consumer)	このオプションを有効にすると、Netty から Camel Message へのバインド中にヘッダー値が URL デコードされます (たとえば、%20 はスペース文字になります。このオプションはデフォルトの org.apache.camel.component.netty.http.NettyHttpBinding で使用されるため、カスタム org.apache.camel.component.netty.http.NettyHttpBinding を実装する場合は、このオプションに従ってヘッダーをデコードする必要があることに注意してください。	false	boolean
workerCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの workerCount パラメーター (cpu_core_threads2) を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの workerCount をオーバーライドできます。		int
workerPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.WorkerPool をワーカースレッドプールとして使用する場合。たとえば、スレッドプールを複数のコンシューマーと共有する場合などです。デフォルトでは、各コンシューマーには、2 x CPU カウントのコアスレッドを備えた独自のワーカープールがあります。		WorkerPool
connectTimeout (producer)	ソケット接続が使用可能になるまで待機する時間。値はミリ単位です。	10000	long
requestTimeout (producer)	リモートサーバーを呼び出すときに、Netty プロデューサーのタイムアウトを使用できるようにします。デフォルトでは、タイムアウトは使用されていません。値はミリ秒単位なので、たとえば 30000 は 30 秒です。requestTimeout は、Netty の ReadTimeoutHandler を使用してタイムアウトをトリガーしています。		long
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
clientPipelineFactory (producer)	カスタム ClientPipelineFactory を使用する場合。		ClientPipelineFactory

名前	説明	デフォルト	タイプ
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにチャネルを遅延作成できます。	true	boolean
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。デフォルトの範囲は 200-299 です。	200-299	String
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要: これをオフにしないでください。並行性と信頼できるリクエスト/レスポンスを処理するためにプーリングが必要です。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当てられる (クライアントにチェックアウトされるか、チェックアウトを待機するアイドル) オブジェクトの数に上限を設定します。無制限の場合は負の値を使用します。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンス数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドル状態のオブジェクト Evictor によるエビクションの対象となる前に、オブジェクトがプール内でアイドル状態になる最小時間 (ミリ単位の値) を設定します。	300000	long
producerPoolMinIdle (producer)	evictor スレッド (アクティブな場合) が新しいオブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
useChannelBuffer (producer)	useChannelBuffer が true の場合、netty プロデューサーはメッセージボディーを送信前に ChannelBuffer に変換します。	false	boolean
useRelativePath (producer)	HTTP リクエストで相対パスを使用するかどうかを設定します。IBM Datapower などの一部のサードパーティーバックエンドシステムは、HTTP POST で絶対 URI をサポートしていないため、このオプションを true に設定すると、この問題を回避できます。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyServerBootstrapConfiguration を使用するには。		NettyServerBootstrap 設定
configuration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyHttpConfiguration を使用するには。		NettyHttpConfiguration

名前	説明	デフォルト	タイプ
headerFilterStrategy (advanced)	カスタム org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルタリングするには。		HeaderFilterStrategy
nettyHttpBinding (advanced)	カスタム org.apache.camel.component.netty.http.NettyHttpBinding を使用して、Netty および Camel Message API との間でバインドします。		NettyHttpBinding
options (advanced)	オプションを使用して、追加の netty オプションを設定できます。接頭辞として。たとえば、netty オプション child.keepAlive=false を設定するには、option.child.keepAlive=false とします。使用可能なオプションについては、Netty のドキュメントを参照してください。		Map
receiveBufferSize (advanced)	インバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	long
receiveBufferSizePredictor (advanced)	バッファーサイズプレディクターを設定します。詳細は、Jetty のドキュメントとこのメールスレッドを参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transferException (advanced)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean

名前	説明	デフォルト	タイプ
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク経由でエクスチェンジを転送することができます。In body、Out body、fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、exchange プロパティー、exchange 例外フィールドが転送されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
decoder (codec)	**非推奨** 単一のデコーダーを使用すること。このオプションは非推奨であり、代わりにエンコーダーを使用してください。		ChannelHandler
decoders (codec)	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
encoder (codec)	**非推奨** 単一のエンコーダーを使用する場合。このオプションは非推奨であり、代わりにエンコーダーを使用してください。		ChannelHandler
encoders (codec)	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
enabledProtocols (security)	SSL を使用するときに有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		File
keyStoreFormat (security)	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS	JKS	String
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String

名前	説明	デフォルト	タイプ
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
パスフレーズ (security)	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定		String
securityConfiguration (security)	安全な Web リソースを設定するための org.apache.camel.component.netty.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurity 設定
securityOptions (security)	マップのキーと値のペアを使用して NettyHttpSecurityConfiguration を設定する場合。		Map
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。	SunX509	String
ssl (security)	このエンドポイントに SSL 暗号化を適用するかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効で SSL モードの場合、Netty コンシューマーは、サブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を含むヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		File
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

228.3. メッセージヘッダー

プロデューサーで次のヘッダーを使用して、HTTP リクエストを制御できます。

名前	タイプ	説明
CamelHttpMethod	**String**	GET、POST、TRACE など、使用する HTTP メソッドを制御できるようにします。タイプは **org.jboss.netty.handler.codec.http.HttpMethod** インスタンスにすることもできます。
CamelHttpQuery	**String**	エンドポイント設定をオーバーライドする **String** 値として URI クエリーパラメーターを提供できます。& 記号を使用して、複数のパラメーターを区切ります。例: **foo=bar&beer=yes** .
CamelHttpPath	**String**	Camel 2.13.1/2.12.4: URI コンテキストパスとクエリーパラメーターを、エンドポイント設定をオーバーライドする **String** 値として提供できるようにします。これにより、同じリモート http サーバーを呼び出すために同じプロデューサーを再利用できますが、動的なコンテキストパスとクエリーパラメーターを使用します。
Content-Type	**String**	HTTP ボディーの content-type を設定します。例： **text/plain; charset="UTF-8"**。
CamelHttpResponseCode	**int**	使用する HTTP ステータスコードを設定できます。デフォルトでは、成功には 200、失敗には 500 が使用されます。

次のヘッダーは、ルートが Netty HTTP エンドポイントから開始するときにメタデータとして提供されます。

表の説明は、次のルートでオフセットを取ります: **from("netty-http:http:0.0.0.0:8080/myapp")...**

名前	タイプ	説明
CamelHttpMethod	**String**	GET、POST、TRACE など、使用される HTTP メソッド。
CamelHttpUrl	**String**	プロトコル、ホスト、ポートなどを含む URL。
CamelHttpUri	**String**	プロトコル、ホスト、ポートなどを含まない URI。
CamelHttpQuery	**String**	**foo=bar&beer=yes** などの任意のクエリーパラメーター。

名前	タイプ	説明
CamelHttpRawQuery	**String**	Camel 2.13.0 : **foo=bar&beer=yes** などの任意のクエリーパラメーター。コンシューマーに到着したとき (つまり、URL デコード前) に未加工の形式で保存されます。
CamelHttpPath	**String**	追加のコンテキストパス。クライアントが context-path /**myapp** を呼び出した場合、この値は空です。クライアントが /**myapp**/**mystuff** を呼び出す場合、このヘッダー値は /**mystuff** です。つまり、ルートエンドポイントで設定された context-path の後の値です。
CamelHttpCharacterEncoding	**String**	content-type ヘッダーの文字セット。
CamelHttpAuthentication	**String**	ユーザーが HTTP Basic を使用して認証された場合、このヘッダーには値 **Basic** が追加されます。
Content-Type	**String**	コンテンツタイプ (提供されている場合)。例： **text/plain; charset="UTF-8"**。

228.4. NETTY 型へのアクセス

このコンポーネントは、Exchange でのメッセージ実装として **org.apache.camel.component.netty.http.NettyHttpMessage** を使用します。これにより、以下に示すように、エンドユーザーは必要に応じて元の Netty リクエスト/レスポンスインスタンスにアクセスできます。元のレスポンスに常にアクセスできるとは限らないことに注意してください。

```
org.jboss.netty.handler.codec.http.HttpRequest request =
exchange.getIn(NettyHttpMessage.class).getHttpRequest();
```

228.5. 例

以下のルートでは、ハードコーディングされた Bye World メッセージを返す HTTP サーバーとして Netty HTTP を使用します。

```
    from("netty-http:http://0.0.0.0:8080/foo")
      .transform().constant("Bye World");
```

また、以下に示すように、ProducerTemplate を使用して、Camel を使用してこの HTTP サーバーを呼び出すこともできます。

```
    String out = template.requestBody("netty-http:http://localhost:8080/foo", "Hello World",
String.class);
    System.out.println(out);
```

出力として Bye World が返されます。

228.6. NETTY にワイルドカードを一致させるにはどうすればよいですか

デフォルトでは、Netty HTTP は正確な uri にのみ一致します。ただし、Netty に接頭辞を一致させるように指示することはできます。以下に例を示します。

```
from("netty-http:http://0.0.0.0:8123/foo").to("mock:foo");
```

上記のルートで Netty HTTP は uri が完全に同じ場合のみ符合するため、次のように入力すると一致します。
http://0.0.0.0:8123/foo ですが、**http://0.0.0.0:8123/foo/bar** を実行すると一致しません。

したがって、ワイルドカードマッチングを有効にする場合は、次のようにします。

```
from("netty-http:http://0.0.0.0:8123/foo?matchOnUriPrefix=true").to("mock:foo");
```

したがって、Netty は **foo** で始まるすべてのエンドポイントに一致します。

任意の エンドポイントに一致させるには、次のようにします。

```
from("netty-http:http://0.0.0.0:8123?matchOnUriPrefix=true").to("mock:foo");
```

228.7. 同じポートで複数のルートを使用する

同じ CamelContext で、同じポート (**org.jboss.netty.bootstrap.ServerBootstrap** インスタンスなど) を共有する Netty HTTP からの複数のルートを持つことができます。ルートは同じ **org.jboss.netty.bootstrap.ServerBootstrap** インスタンスを共有するため、これを行うには、ルート内で多数のブートストラップオプションを同一にする必要があります。インスタンスは、最初に作成されたルートのオプションで設定されます。

ルートが同一に設定される必要があるオプションは、**org.apache.camel.component.netty.NettyServerBootstrapConfiguration** 設定クラスで定義されているすべてのオプションです。異なるオプションで別のルートを設定した場合、Camel は起動時に例外を出力し、オプションが同一でないことを示します。これを軽減するには、すべてのオプションが同一であることを確認してください。

同じポートを共有する 2 つのルートの例を次に示します。

同じポートを共有する 2 つのルート

```
from("netty-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

from("netty-http:http://0.0.0.0:{{port}}/bar")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

そして、これは、1番目のルートと同じ **org.apache.camel.component.netty.NettyServerBootstrapConfiguration** オプションを持たない、誤って設定された 2 番目のルートの例です。これにより、Camel は起動時に失敗します。

2 つのルートが同じポートを共有しているが、2 番目のルートが正しく設定されておらず、起動に失敗する

```
from("netty-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

// we cannot have a 2nd route on same port with SSL enabled, when the 1st route is NOT
from("netty-http:http://0.0.0.0:{{port}}/bar?ssl=true")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

228.7.1. 複数のルートで同じサーバーのブートストラップ設定を再利用する

org.apache.camel.component.netty.NettyServerBootstrapConfiguration タイプの単一インスタンスで共通サーバーブートストラップオプションを設定することにより、Netty HTTP コンシューマーで **bootstrapConfiguration** オプションを使用して、すべてのコンシューマーで同じオプションを参照および再利用できます。

```
<bean id="nettyHttpBootstrapOptions"
class="org.apache.camel.component.netty.NettyServerBootstrapConfiguration">
  <property name="backlog" value="200"/>
  <property name="connectTimeout" value="20000"/>
  <property name="workerCount" value="16"/>
</bean>
```

そしてルートでは、以下に示すようにこのオプションを参照します

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/bar?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/beer?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>
```

228.7.2. OSGi コンテナー内の複数のバンドルにまたがる複数のルートで同じサーバーのブートストラップ設定を再利用する

詳細とその方法の例については、Netty HTTP サーバーの例を参照してください。

228.8. HTTP BASIC 認証の使用

以下に示すように、Netty HTTP コンシューマーは、使用するセキュリティーレルム名を指定することにより、HTTP Basic 認証をサポートします。

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?securityConfiguration.realm=karaf"/>
  ...
</route>
```

Basic 認証を有効にするには、レルム名が必須です。デフォルトでは、JAAS ベースのオーセンティケーターが使用されます。これは、指定されたレルム名 (上記の例では karaf) を使用し、JAAS レルムとこのレルムの JAAS \{{LoginModule}} を認証に使用します。

Apache Karaf/ServiceMix のエンドユーザーはすぐに使用できる karaf レルムを持っているため、上記の例がこれらのコンテナーですぐに機能する理由です。

228.8.1. Web リソースに ACL を指定する

org.apache.camel.component.netty.http.SecurityConstraint を使用すると、Web リソースに対する制約を定義できます。また、**org.apache.camel.component.netty.http.SecurityConstraintMapping** はすぐに使用できるように提供されており、ロールを使用してインクルージョンとエクスクルージョンを簡単に定義できます。

たとえば、XML DSL で以下に示すように、制約 Bean を定義します。

```
<bean id="constraint" class="org.apache.camel.component.netty.http.SecurityConstraintMapping">
  <!-- inclusions defines url -> roles restrictions -->
  <!-- a * should be used for any role accepted (or even no roles) -->
  <property name="inclusions">
    <map>
      <entry key="/*" value="*"/>
      <entry key="/admin/*" value="admin"/>
      <entry key="/guest/*" value="admin,guest"/>
    </map>
  </property>
  <!-- exclusions is used to define public urls, which requires no authentication -->
  <property name="exclusions">
    <set>
      <value>/public/*</value>
    </set>
  </property>
</bean>
```

上記の制約は、次のように定義されます。

- /* へのアクセスは制限され、すべてのロールが受け入れられます (ユーザーがロールを持っていない場合も同様)
- /admin/* へのアクセスには管理者ロールが必要です
- /guest/* へのアクセスには、管理者またはゲストのロールが必要です
- /public/* へのアクセスは、認証が不要であることを意味するエクスクルージョンであるため、ログインしなくても全員に公開されます

この制約を使用するには、以下に示すように Bean ID を参照するだけです。

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?
matchOnUriPrefix=true&amp;securityConfiguration.realm=karaf&amp;securityConfiguration.securityCon
straint=#constraint"/>
  ...
</route>
```

228.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Netty
- Netty HTTP サーバーの例
- Jetty

第229章 NETTY4 コンポーネント

Camel バージョン 2.14 以降で利用可能

Camel の **netty4** コンポーネントは、Netty プロジェクトバージョン 4 に基づくソケット通信コンポーネントです。
Netty は、プロトコルサーバーやクライアントなどの netwServerInitializerFactoryork アプリケーションの迅速かつ簡単な開発を可能にする NIO クライアントサーバーフレームワークです。
Netty は、TCP や UDP ソケットサーバーなどのネットワークプログラミングを大幅に簡素化および合理化します。

この camel コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方をサポートします。

Netty コンポーネントにはいくつかのオプションがあり、多数の TCP/UDP 通信パラメーター (バッファーサイズ、keepAlive、tcpNoDelay など) をきめ細かく制御し、Camel ルートでの In-Only 通信と In-Out 通信の両方を容易にします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

229.1. URI 形式

netty コンポーネントの URI スキームは次のとおりです。

```
netty4:tcp://localhost:99999[?options]
netty4:udp://remotehost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーエンドポイントとコンシューマーエンドポイントをサポートします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

229.2. オプション

Netty4 コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
maximumPoolSize (advanced)	使用中の場合の EventExecutorGroup のスレッドプールサイズ。デフォルト値は 16 です。	16	int
configuration (advanced)	エンドポイントの作成時に NettyConfiguration を設定として使用するには。		NettyConfiguration

名前	説明	デフォルト	タイプ
executorService (advanced)	指定された EventExecutorGroup を使用します。		EventExecutorGroup
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty4 エンドポイントは、URI 構文を使用して設定されます。

```
netty4:protocol:host:port
```

パスおよびクエリーパラメーターを使用します。

229.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
protocol	**必須** tcp または udp の使用するプロトコル。		String
host	**必須** ホスト名。コンシューマーの場合、ホスト名は localhost または 0.0.0.0 ですプロデューサーの場合、ホスト名は接続先のリモートホストです		String
port	**必須** ホストのポート番号		int

229.2.2. クエリーパラメーター(72 個のパラメーター):

名前	説明	デフォルト	タイプ
disconnect (Common)	使用直後に Netty Channel を切断 (クローズ) するかどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためにソケットが閉じられないようにするための設定	true	boolean

名前	説明	デフォルト	タイプ
reuseAddress (Common)	ソケットの多重化を容易にするための設定	true	boolean
reuseChannel (common)	このオプションにより、プロデューサとコンシューマー (クライアントモード) は、エクスチェンジを処理するライフサイクルで同じ Netty チャネルを再利用できます。これは、Camel ルートでサーバーを複数回呼び出す必要があり、同じネットワーク接続を使用したい場合に便利です。これを使用すると、チャネルはエクスチェンジが完了するまで接続プールに返されません。または、切断オプションが true に設定されている場合は切断されます。再利用されたチャネルは、キーリンク NettyConstantsNETTY_CHANNEL を持つエクスチェンジプロパティーとしてエクスチェンジに保存されます。これにより、ルーティング中にチャネルを取得して使用することもできます。	false	boolean
sync (common)	エンドポイントを一方向または要求応答として設定する設定	true	boolean
tcpNoDelay (Common)	TCP プロトコルのパフォーマンスを向上させるための設定	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
broadcast (consumer)	UDP 経由のマルチキャストを選択するための設定	false	boolean
clientMode (consumer)	clientMode が true の場合、netty コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean
reconnect (consumer)	コンシューマーの clientMode でのみ使用されます。これが有効になっている場合、コンシューマーは切断時に再接続を試みます	true	boolean
reconnectInterval (consumer)	再接続し、clientMode が有効になっている場合に使用されます。再接続を試みる間隔 (ミリ秒)	10000	int

名前	説明	デフォルト	タイプ
backlog (consumer)	netty コンシューマー (サーバー) のバックログを設定できます。バックログは、OS によってはベストエフォートであることに注意してください。このオプションを 200、500、1000 などの値に設定すると、TCP スタックに受け入れキューの長さが通知されます。このオプションが設定されていない場合、バックログは OS の設定に依存します。		int
bossCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの BossCount パラメーターである 1 を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの BossCount をオーバーライドできます	1	int
bossGroup (consumer)	NettyEndpoint を介してサーバー側の新しい接続を処理するために使用できる BossGroup を設定します		EventLoopGroup
disconnectOnNoReply (consumer)	同期が有効になっている場合、このオプションは、返信がない場合に NettyConsumer を切断するかどうかを指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
networkInterface (consumer)	UDP を使用する場合、このオプションを使用してネットワークインターフェイスをその名前で指定できます (マルチキャストグループに参加するための eth0 など)。		String
noReplyLogLevel (consumer)	同期が有効になっている場合、このオプションは NettyConsumer がログに記録するときに使用するログレベルを決定し、返信する応答がありません。	WARN	LoggingLevel

名前	説明	デフォルト	タイプ
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー (NettyConsumer) が java.nio.channels.ClosedChannelException をキャッチすると、このログレベルを使用してログに記録されます。これは、クローズドチャネル例外のログ記録を回避するために使用されます。これは、クライアントが突然切断され、Netty サーバーでクローズド例外のフラッドが発生する可能性があるためです。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー (NettyConsumer) が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverInitializerFactory (consumer)	カスタム ServerInitializerFactory を使用する場合		ServerInitializer ファクトリー
usingExecutorService (consumer)	順序付けられたスレッドプールを使用して、イベントが同じチャネルで順番に処理されるかどうか。	true	boolean
connectTimeout (producer)	ソケット接続が使用可能になるまで待機する時間。値はミリ単位です。	10000	int
requestTimeout (producer)	リモートサーバーを呼び出すときに、Netty プロデューサーのタイムアウトを使用できるようにします。デフォルトでは、タイムアウトは使用されていません。値はミリ秒単位なので、たとえば 30000 は 30 秒です。requestTimeout は、Netty の ReadTimeoutHandler を使用してタイムアウトをトリガーしています。		long
clientInitializerFactory (producer)	カスタム ClientInitializerFactory を使用する場合		ClientInitializer ファクトリー

名前	説明	デフォルト	タイプ
correlationManager (producer)	カスタム相関マネージャーを使用して、netty プロデューサーで要求/応答を使用するときに、要求メッセージと応答メッセージがどのようにマップされるかを管理します。これは、要求メッセージと応答メッセージの両方に相関 ID がある場合など、要求を応答と一緒にマップする方法がある場合にのみ使用してください。これは、netty の同じチャネル (別名接続) で同時メッセージを多重化したい場合に使用できます。これを行う場合、リクエストメッセージと応答メッセージを相互に関連付ける方法が必要です。これにより、継続してルーティングされる前に、進行中の Camel エクスチェンジに正しい応答を格納できます。カスタム相関マネージャーを作成するときは、TimeoutCorrelationManagerSupport を拡張することをお勧めします。これにより、タイムアウトや他の方法で実装する必要があるその他の複雑さもサポートされます。詳細については、producerPoolEnabled オプションも参照してください。		NettyCamelStateCorrelationManager
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにチャネルを遅延作成できます。	true	boolean
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要: これをオフにすると、リクエスト/リプライを実行している場合にも、単一の共有接続がプロデューサーに使用されます。つまり、返信が順不同で戻ってきた場合、インタリーブされた応答に問題が生じる可能性があります。したがって、Camel でメッセージの処理を継続するロールを持つ Camel コールバックに応答を適切に関連付けることができるように、要求メッセージと応答メッセージの両方に相関 ID が必要です。これを行うには、NettyCamelStateCorrelationManager を相関マネージャーとして実装し、correlationManager オプションを介して設定する必要があります。詳細は、correlationManager オプションも参照してください。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当てられる (クライアントにチェックアウトされるか、チェックアウトを待機するアイドル) オブジェクトの数に上限を設定します。無制限の場合は負の値を使用します。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンス数の上限を設定します。	100	int

名前	説明	デフォルト	タイプ
producerPoolMinEvictable Idle (producer)	アイドル状態のオブジェクト Evictor によるエビクションの対象となる前に、オブジェクトがプール内でアイドル状態になる最小時間 (ミリ単位の値) を設定します。	300000	long
producerPoolMinIdle (producer)	evictor スレッド (アクティブな場合) が新しいオブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
udpConnectionlessSending (producer)	このオプションは接続のない UDP 送信をサポートします。接続された udp send は、受信ポートでリッスンしている人がいない場合、PortUnreachableException を受け取ります。	false	boolean
useByteBuf (producer)	useByteBuf が true の場合、netty プロデューサーはメッセージ本文を送信する前に ByteBuf に変換します。	false	boolean
allowSerializedHeaders (advanced)	transferExchange が true の場合にのみ TCP に使用されます。true に設定すると、ヘッダーとプロパティーのシリアル化可能なオブジェクトが交換に追加されます。そうしないと、Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyServerBootstrapConfiguration を使用するには。		NettyServerBootstrap 設定
channelGroup (advanced)	明示的な ChannelGroup を使用するには。		ChannelGroup
nativeTransport (advanced)	NIO の代わりにネイティブトランスポートを使用するかどうか。ネイティブトランスポートはホストオペレーティングシステムを利用し、一部のプラットフォームでのみサポートされます。使用しているホストオペレーティングシステムの netty JAR を追加する必要があります。詳細については、http://netty.io/wiki/native-transports.html を参照してください。	false	boolean
options (advanced)	オプションを使用して、追加の netty オプションを設定できます。接頭辞として。たとえば、netty オプション child.keepAlive=false を設定するには、option.child.keepAlive=false とします。使用可能なオプションについては、Netty のドキュメントを参照してください。		Map

名前	説明	デフォルト	タイプ
receiveBufferSize (advanced)	インバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	int
receiveBufferSizePredictor (advanced)	バッファーサイズプレディクターを設定します。詳細は、Jetty のドキュメントとこのメールスレッドを参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク経由でエクスチェンジを転送することができます。In body、Out body、fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、exchange プロパティー、exchange 例外フィールドが転送されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
udpByteArrayCodec (advanced)	UDP 専用です。Java シリアライゼーションプロトコルの代わりにバイト配列コーデックの使用を有効にした場合。	false	boolean
workerCount (advanced)	netty が nio モードで動作する場合、Netty のデフォルトの workerCount パラメーター (cpu_core_threads2) を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの workerCount をオーバーライドできます。		int
workerGroup (advanced)	ボススレッドプールとして明示的な EventLoopGroup を使用するには。たとえば、スレッドプールを複数のコンシューマーまたはプロデューサーと共有する場合などです。デフォルトでは、各コンシューマーまたはプロデューサーには、2 x CPU カウントのコアスレッドを備えた独自のワーカープールがあります。		EventLoopGroup
allowDefaultCodec (codec)	netty コンポーネントは、encoder/deocder が null で textline が false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、netty コンポーネントがデフォルトコーデックをフィルターチェーンの最初の要素としてインストールするのを防ぎます。	true	boolean

名前	説明	デフォルト	タイプ
autoAppendDelimiter (codec)	テキストラインコーデックを使用して送信するときに、不足している終了区切り文字を自動追加するかどうか。	true	boolean
decoder (codec)	**非推奨** 受信ペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。		ChannelHandler
decoderMaxLineLength (codec)	テキストラインコーデックに使用する行の最大長。	1024	int
decoders (codec)	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
delimiter (codec)	テキストラインコーデックに使用する区切り文字。可能な値は LINE と NULL です。	LINE	TextLineDelimiter
encoder (codec)	**非推奨** 送信ペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。		ChannelHandler
encoders (codec)	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
encoding (codec)	テキスト行コーデックに使用するエンコーディング (文字セット名)。指定しない場合、Camel は JVM のデフォルトの文字セットを使用します。		String
textline (codec)	TCP にのみ使用されます。コーデックが指定されていない場合、このフラグを使用して、テキスト行ベースのコーデックを示すことができます。指定されていない場合、または値が false の場合、オブジェクトのシリアル化は TCP 経由であると想定されます。	false	boolean
enabledProtocols (security)	SSL を使用するときに有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		File

名前	説明	デフォルト	タイプ
keyStoreFormat (security)	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS		String
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
パスフレーズ (security)	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定		String
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。		String
ssl (security)	このエンドポイントに SSL 暗号化を適用するかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効で SSL モードの場合、Netty コンシューマーは、サブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を含むヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		File
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String

229.3. レジストリーベースのオプション

コーデックハンドラーと SSL キーストアは、Spring XML ファイルなどのレジストリーに登録できます。渡すことができる値は次のとおりです。

名前	説明
passphrase	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定
keyStoreFormat	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS
securityProvider	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。
keyStoreFile	**非推奨:** 暗号化に使用されるクライアント側の証明書キーストア
trustStoreFile	**非推奨:** 暗号化に使用されるサーバー側の証明書キーストア
keyStoreResource	Camel 2.11.1: 暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、**"classpath:"**、**"file:"**、または **"http:"** をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。
trustStoreResource	Camel 2.11.1: 暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、**"classpath:"**、**"file:"**、または **"http:"** をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。
sslHandler	SSL ハンドラーを返すために使用できるクラスへの参照
encoder	送信ペイロードの特別なマーシャリングを実行するために使用できるカスタム **ChannelHandler** クラス。Must override io.netty.channel.ChannelInboundHandlerAdapter.
encoders	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に # を付けることを忘れないでください。
decoder	受信ペイロードの特別なマーシャリングを実行するために使用できるカスタム **ChannelHandler** クラス。Must override io.netty.channel.ChannelOutboundHandlerAdapter.
decoders	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に # を付けることを忘れないでください。

注記

共有不可能なエンコーダー/デコーダーの使用については、以下をお読みください。

229.3.1. 共有不可能なエンコーダーまたはデコーダーの使用

エンコーダーまたはデコーダーが共有可能でない場合 (たとえば、@Shareable クラスアノテーションがある場合)、エンコーダー/デコーダーは **org.apache.camel.component.netty.ChannelHandlerFactory** インターフェイスを実装し、**newChannelHandler** メソッドで新しいインスタンスを返さなければなりません。これは、エンコーダー/デコーダーを安全に使用できるようにするためです。そうでない場合、Netty コンポーネントは、エンドポイントの作成時
に WARN をログに記録します。

Netty コンポーネントは、多くの一般的に使用されるメソッドを持つ **org.apache.camel.component.netty.ChannelHandlerFactories** ファクトリークラスを提供します。

229.4. NETTY エンドポイントとの間でメッセージを送信する

229.4.1. Netty プロデューサー

Producer モードでは、コンポーネントは、TCP または UDP プロトコル (オプションの SSL サポート付き) を使用してペイロードをソケットエンドポイントに送信する機能を提供します。

プロデューサーモードは、一方向および要求/応答ベースの操作の両方をサポートします。

229.4.2. Netty コンシューマー

コンシューマーモードでは、コンポーネントは次の機能を提供します。

- TCP または UDP プロトコル (オプションの SSL サポート付き) を使用して、指定されたソケットでリッスンします。
- text/xml、バイナリーおよびシリアライズされたオブジェクトベースのペイロードを使用してソケットでリクエストを受信し、
- メッセージ交換としてルートに沿ってそれらを送信します。

コンシューマーモードは、一方向および要求/応答ベースの操作の両方をサポートします。

229.5. 例

229.5.1. Request-Reply とシリアライズされたオブジェクトペイロードを使用する UDP Netty エンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("netty4:udp://localhost:5155?sync=true")
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Poetry poetry = (Poetry) exchange.getIn().getBody();
          poetry.setPoet("Dr. Sarojini Naidu");
          exchange.getOut().setBody(poetry);
```

```
        }
      }
    }
};
```

229.5.2. 一方向通信を使用する TCP ベースの Netty コンシューマーエンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
      from("netty4:tcp://localhost:5150")
          .to("mock:result");
  }
};
```

229.5.3. Request-Reply 通信を使用する SSL/TCP ベースの Netty コンシューマーエンドポイント

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Netty コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、Netty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

NettyComponent nettyComponent = getContext().getComponent("netty4", NettyComponent.class);
nettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
```

```
  <to uri="netty4:tcp://localhost:5150?
sync=true&ssl=true&sslContextParameters=#sslContextParameters"/>
...
```

Netty4-Jetty コンポーネントでの基本的な SSL/TLS 設定の使用 Jetty コンポーネントでの基本的な SSL/TLS 設定の使用

```
JndiRegistry registry = new JndiRegistry(createJndiContext());
registry.bind("password", "changeit");
registry.bind("ksf", new File("src/test/resources/keystore.jks"));
registry.bind("tsf", new File("src/test/resources/keystore.jks"));

context.createRegistry(registry);
context.addRoutes(new RouteBuilder() {
  public void configure() {
      String netty_ssl_endpoint =
        "netty4:tcp://localhost:5150?sync=true&ssl=true&passphrase=#password"
        + "&keyStoreFile=#ksf&trustStoreFile=#tsf";
      String return_string =
        "When You Go Home, Tell Them Of Us And Say,"
        + "For Your Tomorrow, We Gave Our Today.";

      from(netty_ssl_endpoint)
       .process(new Processor() {
          public void process(Exchange exchange) throws Exception {
            exchange.getOut().setBody(return_string);
          }
       }
  }
});
```

SSLSession とクライアント証明書へのアクセスを取得する

たとえば、クライアント証明書に関する詳細を取得する必要がある場合は、**javax.net.ssl.SSLSession** にアクセスできます。**ssl=true** の場合、以下に示すように、Netty4 コンポーネントは **SSLSession** を Camel メッセージのヘッダーとして保存します。

```
SSLSession session = exchange.getIn().getHeader(NettyConstants.NETTY_SSL_SESSION,
SSLSession.class);
// get the first certificate which is client certificate
javax.security.cert.X509Certificate cert = session.getPeerCertificateChain()[0];
Principal principal = cert.getSubjectDN();
```

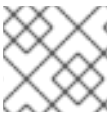
クライアントを認証するには、必ず **needClientAuth=true** を設定してください。そうしないと、**SSLSession** はクライアント証明書に関する情報にアクセスできず、例外 **javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated** が発生する可能性があります。クライアント証明書の有効期限が切れているか、有効でない場合などにも、この例外が発生する可能性があります。

ヒント

オプション **sslClientCertHeaders** を**true** に設定すると、Camel メッセージがクライアント証明書に関する詳細を含むヘッダーで強化されます。たとえば、サブジェクト名はヘッダー **CamelNettySSLClientCertSubjectName** ですぐに利用できます。

229.5.4. 複数のコーデックの使用

場合によっては、エンコーダーとデコーダーのチェーンを netty パイプラインに追加する必要があります。複数のコーデックを camel netty エンドポイントに追加するには、'encoders' および 'decoders' uri パラメーターを使用する必要があります。encoder および decoder パラメーターと同様に、パイプラインに追加する必要がある (ChannelUpstreamHandlers および ChannelDownstreamHandlers のリストへの) 参照を提供するために使用されます。エンコーダーが指定されている場合、デコーダーとデコーダーのパラメーターと同様に、エンコーダーのパラメーターは無視されることに注意してください。



注記

共有不可能なエンコーダー/デコーダーの使用については、上記をお読みください。

エンドポイントの作成時に解決できるように、コーデックのリストを Camel のレジストリーに追加する必要があります。

```
ChannelHandlerFactory lengthDecoder =
ChannelHandlerFactories.newLengthFieldBasedFrameDecoder(1048576, 0, 4, 0, 4);

StringDecoder stringDecoder = new StringDecoder();
registry.bind("length-decoder", lengthDecoder);
registry.bind("string-decoder", stringDecoder);

LengthFieldPrepender lengthEncoder = new LengthFieldPrepender(4);
StringEncoder stringEncoder = new StringEncoder();
registry.bind("length-encoder", lengthEncoder);
registry.bind("string-encoder", stringEncoder);

List<ChannelHandler> decoders = new ArrayList<ChannelHandler>();
decoders.add(lengthDecoder);
decoders.add(stringDecoder);

List<ChannelHandler> encoders = new ArrayList<ChannelHandler>();
encoders.add(lengthEncoder);
encoders.add(stringEncoder);

registry.bind("encoders", encoders);
registry.bind("decoders", decoders);
```

Spring のネイティブコレクションサポートを使用して、アプリケーションコンテキストでコーデックリストを指定できます。

```
<util:list id="decoders" list-class="java.util.LinkedList">
        <bean class="org.apache.camel.component.netty4.ChannelHandlerFactories" factory-
method="newLengthFieldBasedFrameDecoder">
            <constructor-arg value="1048576"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringDecoder"/>
    </util:list>

    <util:list id="encoders" list-class="java.util.LinkedList">
```

```
        <bean class="io.netty.handler.codec.LengthFieldPrepender">
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringEncoder"/>
    </util:list>

    <bean id="length-encoder" class="io.netty.handler.codec.LengthFieldPrepender">
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-encoder" class="io.netty.handler.codec.string.StringEncoder"/>

    <bean id="length-decoder" class="org.apache.camel.component.netty4.ChannelHandlerFactories"
factory-method="newLengthFieldBasedFrameDecoder">
        <constructor-arg value="1048576"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-decoder" class="io.netty.handler.codec.string.StringDecoder"/>
```

Bean 名は、netty エンドポイント定義でコンマ区切りのリストとして使用するか、リストに含めることができます。

```
 from("direct:multiple-codec").to("netty4:tcp://localhost:{{port}}?encoders=#encoders&sync=false");

 from("netty4:tcp://localhost:{{port}}?decoders=#length-decoder,#string-
decoder&sync=false").to("mock:multiple-codec");
```

または XML 経由。

```
<camelContext id="multiple-netty-codecs-context" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:multiple-codec"/>
        <to uri="netty4:tcp://localhost:5150?encoders=#encoders&amp;sync=false"/>
    </route>
    <route>
        <from uri="netty4:tcp://localhost:5150?decoders=#length-decoder,#string-
decoder&amp;sync=false"/>
        <to uri="mock:multiple-codec"/>
    </route>
</camelContext>
```

229.6. 完了時にチャネルを閉じる

サーバーとして機能している場合、たとえばクライアントの変換が終了したときにチャネルを閉じたい場合があります。
これは、エンドポイントオプションの **disconnect=true** を設定するだけで実行できます。

ただし、次のようにメッセージごとに Camel に指示することもできます。
Camel にチャネルを閉じるように指示するには、キー **CamelNettyCloseChannelWhenComplete** をブール値 **true** に設定したヘッダーを追加する必要があります。
たとえば、次の例では、bye メッセージをクライアントに書き戻した後にチャネルを閉じます。

```
from("netty4:tcp://localhost:8080").process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        String body = exchange.getIn().getBody(String.class);
        exchange.getOut().setBody("Bye " + body);
        // some condition which determines if we should close
        if (close) {

exchange.getOut().setHeader(NettyConstants.NETTY_CLOSE_CHANNEL_WHEN_COMPLETE,
true);
        }
    }
});
```

カスタムチャネルパイプラインファクトリーを追加して、

229.7. カスタムパイプラインを完全に制御します

カスタムチャネルパイプラインは、Netty エンドポイント URL で非常に簡単な方法で指定することなく、カスタムハンドラー、エンコーダー、デコーダーを挿入することで、ハンドラー/インターセプターチェーンを完全に制御します。

カスタムパイプラインを追加するには、カスタムチャネルパイプラインファクトリーを作成し、コンテキストレジストリー (Registry、または camel-spring ApplicationContextRegistry など) を介してコンテキストに登録する必要があります。

カスタムパイプラインファクトリーは、次のように構築する必要があります。

- プロデューサーにリンクされたチャネルパイプラインファクトリーは、抽象クラス **ClientPipelineFactory** を拡張する必要があります。
- コンシューマーリンクチャネルパイプラインファクトリーは、抽象クラス **ServerInitializerFactory** を拡張する必要があります。
- カスタムハンドラー、エンコーダ、およびデコーダを挿入するには、クラスで initChannel () メソッドをオーバーライドする必要があります。**initChannel()** メソッドをオーバーライドしないと、パイプラインに接続されたハンドラー、エンコーダー、またはデコーダーのないパイプラインが作成されます。

以下の例は、ServerInitializerFactory ファクトリーを作成する方法を示しています。

229.7.1. カスタムパイプラインファクトリーの使用

```
public class SampleServerInitializerFactory extends ServerInitializerFactory {
    private int maxLineSize = 1024;

    protected void initChannel(Channel ch) throws Exception {
        ChannelPipeline channelPipeline = ch.pipeline();

        channelPipeline.addLast("encoder-SD", new StringEncoder(CharsetUtil.UTF_8));
        channelPipeline.addLast("decoder-DELIM", new DelimiterBasedFrameDecoder(maxLineSize,
true, Delimiters.lineDelimiter()));
        channelPipeline.addLast("decoder-SD", new StringDecoder(CharsetUtil.UTF_8));
        // here we add the default Camel ServerChannelHandler for the consumer, to allow Camel to
route the message etc.
```

```
        channelPipeline.addLast("handler", new ServerChannelHandler(consumer));
    }
}
```

次に、カスタムチャネルパイプラインファクトリーをレジストリーに追加し、次の方法でキャメルルートでインスタンス化/利用できます。

```
Registry registry = camelContext.getRegistry();
ServerInitializerFactory factory = new TestServerInitializerFactory();
registry.bind("spf", factory);
context.addRoutes(new RouteBuilder() {
  public void configure() {
      String netty_ssl_endpoint =
        "netty4:tcp://localhost:5150?serverInitializerFactory=#spf"
      String return_string =
        "When You Go Home, Tell Them Of Us And Say,"
        + "For Your Tomorrow, We Gave Our Today.";

      from(netty_ssl_endpoint)
       .process(new Processor() {
          public void process(Exchange exchange) throws Exception {
            exchange.getOut().setBody(return_string);
          }
       }
  }
});
```

229.8. NETTY ボスおよびワーカースレッドプールの再利用

Netty には、ボスとワーカーの 2 種類のスレッドプールがあります。デフォルトでは、各 Netty コンシューマーとプロデューサーにはプライベートスレッドプールがあります。複数のコンシューマーまたはプロデューサー間でこれらのスレッドプールを再利用する場合は、スレッドプールを作成してレジストリーに登録する必要があります。

たとえば、Spring XML を使用すると、以下に示すように、2 つのワーカースレッドを持つ **NettyWorkerPoolBuilder** を使用して共有ワーカースレッドプールを作成できます。

```
<!-- use the worker pool builder to help create the shared thread pool -->
<bean id="poolBuilder" class="org.apache.camel.component.netty.NettyWorkerPoolBuilder">
  <property name="workerCount" value="2"/>
</bean>

<!-- the shared worker thread pool -->
<bean id="sharedPool" class="org.jboss.netty.channel.socket.nio.WorkerPool"
      factory-bean="poolBuilder" factory-method="build" destroy-method="shutdown">
</bean>
```

ヒント

Boss スレッドプールには、Netty コンシューマー用の **org.apache.camel.component.netty4.NettyServerBossPoolBuilder** ビルダーと、Netty プロデューサー用の **org.apache.camel.component.netty4.NettyClientBossPoolBuilder** があります。

次に、Camel ルートで、以下に示すように URI で **workerPool** オプションを設定することにより、このワーカープールを参照できます。

```
<route>
  <from uri="netty4:tcp://localhost:5021?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

別のルートがある場合は、共有ワーカープールを参照できます。

```
<route>
  <from uri="netty4:tcp://localhost:5022?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

などがあります。

229.9. リクエスト/リプライによる単一接続での同時メッセージの多重化

netty プロデューサーを介した要求/応答メッセージングに Netty を使用する場合、デフォルトでは、各メッセージは非共有接続 (プール) を介して送信されます。これにより、返信が自動的に正しいリクエストスレッドにマップされ、Camel でさらにルーティングできるようになります。言い換えると、リクエスト/リプライメッセージ間の相関関係は、リクエストの送信に使用されたのと同じ接続でリプライが返されるため、すぐに使用できます。この接続は他のユーザーと共有されません。応答が戻ってくると、接続は接続プールに戻され、他のユーザーが再利用できるようになります。

ただし、単一の共有接続で同時要求/応答を多重化したい場合は、**producerPoolEnabled=false** を設定して接続プールをオフにする必要があります。これは、返信が順不同で戻ってきた場合、インタリーブされた応答に潜在的な問題があることを意味します。したがって、Camel でメッセージの処理を継続するロールを持つ Camel コールバックに応答を適切に関連付けることができるように、要求メッセージと応答メッセージの両方に相関 ID が必要です。これを行うには、**NettyCamelStateCorrelationManager** を相関マネージャーとして実装し、**correlationManager=#myManager** オプションを介して設定する必要があります。

注記

カスタム相関マネージャーを作成するときは、**TimeoutCorrelationManagerSupport** を拡張することをお勧めします。これにより、タイムアウトや他の方法で実装する必要があるその他の複雑さもサポートされます。

229.10. 関連項目

- Netty HTTP
- MINA

第230章 NETTY4 HTTP コンポーネント

Camel バージョン 2.14 以降で利用可能

netty4-http コンポーネントは、Netty4 による HTTP トランスポートを容易にする Netty4 コンポーネントの拡張機能です。

この camel コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方をサポートします。

情報: **ストリーム**。Netty はストリームベースです。つまり、受信した入力はストリームとして Camel に送信されます。つまり、ストリームのコンテンツを **一度** だけ読み取ることができます。もし、メッセージボディーが空のように見える場合や、何度もデータにアクセスする必要がある場合 (例: マルチキャストや再配送エラー処理)、ストリームキャッシュを使用するか、何度再読み込みしても安全な **String** にメッセージボディーを変換する必要があります。Netty4 HTTP は、**io.netty.handler.codec.http.HttpObjectAggregator** を使用してストリーム全体をメモリーに読み込み、完全な http メッセージ全体をビルドすることに注意してください。ただし、結果のメッセージは、一度読み取り可能なストリームベースのメッセージのままです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty4-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

230.1. URI 形式

netty コンポーネントの URI スキームは次のとおりです。

```
netty4-http:http://localhost:8080[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

情報: **クエリーパラメーターとエンドポイントオプション**。Camel が URI クエリーパラメーターとエンドポイントオプションをどのように認識するのか疑問に思われるかもしれません。たとえば、次のようにエンドポイント URI を作成できます - **netty4-http:http//example.com?myParam=myValue&compression=true**。この例では、**myParam** がHTTP パラメーターであり、**compression** が Camel エンドポイントオプションです。このような状況で Camel が使用するストラテジーは、利用可能なエンドポイントオプションを解決し、それらを URI から削除することです。これは、前述の例の場合、**圧縮** エンドポイントオプションが解決され、ターゲット URL から削除されるため、Netty HTTP プロデューサーによってエンドポイントに送信される HTTP リクエストが **http//example.com?myParam=myValue** のようになることを意味します。また、動的ヘッダー (**CamelHttpQuery** など) を使用してエンドポイントオプションを指定できないことにも注意してください。エンドポイントオプションは、エンドポイント URI 定義レベル (DSL 要素 **to** または **from** など) でのみ指定できます。

230.2. HTTP オプション

情報: **さらに多くのオプションがあります。重要:** このコンポーネントは、Netty4 からすべてのオプションを継承します。したがって、Netty4 のドキュメントも参照してください。
この Netty4 HTTP コンポーネントを使用する場合、UDP トランスポートに関連するオプションな

ど、Netty4 の一部のオプションは適用されないことに注意してください。

Netty4 HTTP コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
nettyHttpBinding (advanced)	カスタム org.apache.camel.component.netty4.http.NettyHttpBinding を使用して、Netty および Camel Message API との間でバインドします。		NettyHttpBinding
configuration (common)	エンドポイントの作成時に NettyConfiguration を設定として使用するには。		NettyHttpConfiguration
headerFilterStrategy (advanced)	カスタム org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルタリングするには。		HeaderFilterStrategy
securityConfiguration (security)	安全な Web リソースを設定するための org.apache.camel.component.netty4.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurity 設定
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
maximumPoolSize (advanced)	使用中の場合の EventExecutorGroup のスレッドプールサイズ。デフォルト値は 16 です。	16	int
executorService (advanced)	指定された EventExecutorGroup を使用します。		EventExecutorGroup
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty4 HTTP エンドポイントは、URI 構文を使用して設定されます。

```
netty4-http:protocol:host:port/path
```

パスおよびクエリーパラメーターを使用します。

230.2.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
protocol	**必須** 使用するプロトコル (http または https)		String

名前	説明	デフォルト	タイプ
host	**必須** localhost などのローカルホスト名、またはコンシューマーの場合は 0.0.0.0。プロデューサーを使用する場合のリモート HTTP サーバーのホスト名。		String
port	ホストのポート番号。		int
path	リソースパス		String

230.2.2. クエリーパラメーター(79 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeEndpoint (common)	オプションが true の場合、プロデューサーは Exchange.HTTP_URI ヘッダーを無視し、エンドポイントの URI をリクエストに使用します。また、throwExceptionOnFailure を false に設定して、プロデューサーがすべての障害応答を送り返すようにすることもできます。ブリッジモードで動作するコンシューマーは、gzip 圧縮と WWW URL フォームエンコーディングをスキップします (消費される Exchange に Exchange.SKIP_GZIP_ENCODING と Exchange.SKIP_WWW_FORM_URLENCODED ヘッダーを追加することで行います)。	false	boolean
disconnect (Common)	使用直後に Netty Channel を切断 (クローズ) するかどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためにソケットが閉じられないようにするための設定	true	boolean
reuseAddress (Common)	ソケットの多重化を容易にするための設定	true	boolean

名前	説明	デフォルト	タイプ
reuseChannel (common)	このオプションにより、プロデューサとコンシューマー (クライアントモード) は、エクスチェンジを処理するライフサイクルで同じ Netty チャネルを再利用できます。これは、Camel ルートでサーバーを複数回呼び出す必要があり、同じネットワーク接続を使用したい場合に便利です。これを使用すると、チャネルはエクスチェンジが完了するまで接続プールに返されません。または、切断オプションが true に設定されている場合は切断されます。再利用されたチャネルは、キーリンク NettyConstantsNETTY_CHANNEL を持つエクスチェンジプロパティーとしてエクスチェンジに保存されます。これにより、ルーティング中にチャネルを取得して使用することもできます。	false	boolean
sync (common)	エンドポイントを一方向または要求応答として設定する設定	true	boolean
tcpNoDelay (Common)	TCP プロトコルのパフォーマンスを向上させるための設定	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、Camel が URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
send503whenSuspended (consumer)	コンシューマーが中断されたときに HTTP ステータスコード 503 を返すかどうか。このオプションが false の場合、コンシューマーが中断されたときに Netty Acceptor がバインド解除されるため、クライアントはそれ以上接続できません。	true	boolean

名前	説明	デフォルト	タイプ
backlog (consumer)	netty コンシューマー (サーバー) のバックログを設定できます。バックログは、OS によってはベストエフォートであることに注意してください。このオプションを 200、500、1000 などの値に設定すると、TCP スタックに受け入れキューの長さが通知されます。このオプションが設定されていない場合、バックログは OS の設定に依存します。		int
bossCount (consumer)	netty が nio モードで動作する場合、Netty のデフォルトの BossCount パラメーターである 1 を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの BossCount をオーバーライドできます	1	int
bossGroup (consumer)	NettyEndpoint を介してサーバー側の新しい接続を処理するために使用できる BossGroup を設定します		EventLoopGroup
chunkedMaxContentLength (consumer)	Netty HTTP サーバーで受信したチャンクフレームごとのコンテンツの最大長 (バイト単位) の値。	1048576	int
compression (consumer)	クライアントが HTTP ヘッダーからサポートしている場合、Netty HTTP サーバーでの圧縮に gzip/deflate の使用を許可します。	false	boolean
disconnectOnNoReply (consumer)	同期が有効になっている場合、このオプションは、返信がない場合に NettyConsumer を切断するかどうかを指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpMethodRestrict (consumer)	Netty HTTP コンシューマーで HTTP メソッドを無効にします。コンマ区切りで複数指定できます。		String

名前	説明	デフォルト	タイプ
mapHeaders (consumer)	このオプションを有効にすると、Netty から Camel Message へのバインド中にヘッダーもマッピングされます (たとえば、ヘッダーとして Camel Message にも追加されます)。このオプションをオフにして、これを無効にすることができます。ヘッダーには、Netty HTTP リクエスト io.netty.handler.codec.http.HttpRequest インスタンスを返すメソッド getHttpRequest() を使用して、org.apache.camel.component.netty.http.NettyHttpMessage メッセージから引き続きアクセスできます。	true	boolean
maxHeaderSize (consumer)	すべてのヘッダーの最大長。各ヘッダーの長さの合計がこの値を超えると、io.netty.handler.codec.TooLongFrameException が発生します。	8192	int
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
nettySharedHttpServer (consumer)	共有 Netty HTTP サーバーを使用するには。詳細については、Netty HTTP サーバーの例を参照してください。		NettySharedHttpServer
noReplyLogLevel (consumer)	同期が有効になっている場合、このオプションは NettyConsumer がログに記録するときに使用するログレベルを決定し、返信する応答がありません。	WARN	LoggingLevel
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー (NettyConsumer) が java.nio.channels.ClosedChannelException をキャッチすると、このログレベルを使用してログに記録されます。これは、クローズドチャネル例外のログ記録を回避するために使用されます。これは、クライアントが突然切断され、Netty サーバーでクローズド例外のフラッドが発生する可能性があるためです。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー (NettyConsumer) が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverInitializerFactory (consumer)	カスタム ServerInitializerFactory を使用する場合		ServerInitializer ファクトリー
traceEnabled (consumer)	この Netty HTTP コンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean

名前	説明	デフォルト	タイプ
urlDecodeHeaders (consumer)	このオプションを有効にすると、Netty から Camel Message へのバインド中にヘッダー値が URL デコードされます (たとえば、%20 はスペース文字になります。このオプションはデフォルトの org.apache.camel.component.netty.http.NettyHttpBinding で使用されるため、カスタム org.apache.camel.component.netty4.http.NettyHttpBinding を実装する場合は、このオプションに従ってヘッダーをデコードする必要があることに注意してください。	false	boolean
usingExecutorService (consumer)	順序付けられたスレッドプールを使用して、イベントが同じチャネルで順番に処理されるかどうか。	true	boolean
connectTimeout (producer)	ソケット接続が使用可能になるまで待機する時間。値はミリ単位です。	10000	int
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
requestTimeout (producer)	リモートサーバーを呼び出すときに、Netty プロデューサーのタイムアウトを使用できるようにします。デフォルトでは、タイムアウトは使用されていません。値はミリ秒単位なので、たとえば 30000 は 30 秒です。requestTimeout は、Netty の ReadTimeoutHandler を使用してタイムアウトをトリガーしています。		long
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
clientInitializerFactory (producer)	カスタム ClientInitializerFactory を使用する場合		ClientInitializer ファクトリー
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合は、例外を回避するためにチャネルを遅延作成できます。	true	boolean
okStatusCodeRange (producer)	正常な応答と見なされるステータスコード。値は含まれます。複数の範囲をコンマで区切って定義できます (例: 200-204,209,301-304)。各範囲は、ダッシュを含む 1 つの数字または from-to である必要があります。デフォルトの範囲は 200-299 です。	200-299	String

名前	説明	デフォルト	タイプ
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要: これをオフにすると、リクエスト/リプライを実行している場合にも、単一の共有接続がプロデューサーに使用されます。つまり、返信が順不同で戻ってきた場合、インタリーブされた応答に問題が生じる可能性があります。したがって、Camel でメッセージの処理を継続するロールを持つ Camel コールバックに応答を適切に関連付けることができるように、要求メッセージと応答メッセージの両方に相関 ID が必要です。これを行うには、NettyCamelStateCorrelationManager を相関マネージャーとして実装し、correlationManager オプションを介して設定する必要があります。詳細は、correlationManager オプションも参照してください。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当てられる (クライアントにチェックアウトされるか、チェックアウトを待機するアイドル) オブジェクトの数に上限を設定します。無制限の場合は負の値を使用します。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンス数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドル状態のオブジェクト Evictor によるエビクションの対象となる前に、オブジェクトがプール内でアイドル状態になる最小時間 (ミリ単位の値) を設定します。	300000	long
producerPoolMinIdle (producer)	evictor スレッド (アクティブな場合) が新しいオブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
useRelativePath (producer)	HTTP リクエストで相対パスを使用するかどうかを設定します。	false	boolean
allowSerializedHeaders (advanced)	transferExchange が true の場合にのみ TCP に使用されます。true に設定すると、ヘッダーとプロパティーのシリアル化可能なオブジェクトが交換に追加されます。そうしないと、Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyServerBootstrapConfiguration を使用するには。		NettyServerBootstrap 設定
channelGroup (advanced)	明示的な ChannelGroup を使用するには。		ChannelGroup

名前	説明	デフォルト	タイプ
configuration (advanced)	このエンドポイントを設定するためにカスタム設定された NettyHttpConfiguration を使用するには。		NettyHttpConfiguration
disableStreamCache (advanced)	Netty HttpRequestgetContent() または HttpResponsetgetContent() からの raw の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームを軽量メモリーベースのストリームキャッシュに読み込みます)。デフォルトでは、Camel は Netty 入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。このオプションを有効にすると、そのままでは Netty ストリームを複数回読み取ることができず、Netty raw ストリームのリーダーインデックスを手動でリセットする必要があることに注意してください。また、Netty は、Netty HTTP サーバー/HTTPクライアントの処理が終了すると、Nettyストリームを自動で閉じます。つまり、非同期ルーティングエンジンが使用されている場合、org.apache.camel.Exchange をルーティングし続ける非同期のスレッドは、Netty ストリームを閉じるために読み取ることができない場合があります。	false	boolean
headerFilterStrategy (advanced)	カスタム org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルタリングするには。		HeaderFilterStrategy
nativeTransport (advanced)	NIO の代わりにネイティブトランスポートを使用するかどうか。ネイティブトランスポートはホストオペレーティングシステムを利用し、一部のプラットフォームでのみサポートされます。使用しているホストオペレーティングシステムの netty JAR を追加する必要があります。詳細については、http://netty.io/wiki/native-transports.html を参照してください。	false	boolean
nettyHttpBinding (advanced)	カスタム org.apache.camel.component.netty4.http.NettyHttpBinding を使用して、Netty および Camel Message API との間でバインドします。		NettyHttpBinding
options (advanced)	オプションを使用して、追加の netty オプションを設定できます。接頭辞として。たとえば、netty オプション child.keepAlive=false を設定するには、option.child.keepAlive=false とします。使用可能なオプションについては、Netty のドキュメントを参照してください。		Map

名前	説明	デフォルト	タイプ
receiveBufferSize (advanced)	インバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	int
receiveBufferSizePredictor (advanced)	バッファーサイズプレディクターを設定します。詳細は、Jetty のドキュメントとこのメールスレッドを参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイトです。	65536	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transferException (advanced)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク経由でエクスチェンジを転送することができます。In body、Out body、fault body、In ヘッダー、Out ヘッダー、Fault ヘッダー、exchange プロパティー、exchange 例外フィールドが転送されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化できないオブジェクトを除外し、WARN レベルでログに記録します。	false	boolean
workerCount (advanced)	netty が nio モードで動作する場合、Netty のデフォルトの workerCount パラメーター (cpu_core_threads2) を使用します。ユーザーはこの操作を使用して、Netty のデフォルトの workerCount をオーバーライドできます。		int

名前	説明	デフォルト	タイプ
workerGroup (advanced)	ボススレッドプールとして明示的な EventLoopGroup を使用するには。たとえば、スレッドプールを複数のコンシューマーまたはプロデューサーと共有する場合などです。デフォルトでは、各コンシューマーまたはプロデューサーには、2 x CPU カウントのコアスレッドを備えた独自のワーカープールがあります。		EventLoopGroup
decoder (codec)	**非推奨** 単一のデコーダーを使用すること。このオプションは非推奨であり、代わりにエンコーダーを使用してください。		ChannelHandler
decoders (codec)	使用するデコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
encoder (codec)	**非推奨** 単一のエンコーダーを使用する場合。このオプションは非推奨であり、代わりにエンコーダーを使用してください。		ChannelHandler
encoders (codec)	使用するエンコーダーのリスト。コンマで区切られた値を持つ文字列を使用して、値をレジストリーで検索することができます。Camel がルックアップする必要があることを認識できるように、値の前に付けることを忘れないでください。		文字列
enabledProtocols (security)	SSL を使用するときに有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		File
keyStoreFormat (security)	ペイロードの暗号化に使用されるキーストア形式。設定されていない場合、デフォルトは JKS		String
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		String

名前	説明	デフォルト	タイプ
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
パスフレーズ (security)	SSH を使用して送信されたペイロードを暗号化/復号化するために使用するパスワード設定		String
securityConfiguration (security)	安全な Web リソースを設定するための org.apache.camel.component.netty4.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurity 設定
securityOptions (security)	マップのキーと値のペアを使用して NettyHttpSecurityConfiguration を設定する場合。		Map
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 です。		String
ssl (security)	このエンドポイントに SSL 暗号化を適用するかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効で SSL モードの場合、Netty コンシューマーは、サブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を含むヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		File
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

230.3. メッセージヘッダー

プロデューサーで次のヘッダーを使用して、HTTP リクエストを制御できます。

名前	タイプ	説明
CamelHttpMethod	**String**	GET、POST、TRACE など、使用する HTTP メソッドを制御できるようにします。タイプは、**io.netty.handler.codec.http.HttpMethod** インスタンスにすることもできます。
CamelHttpQuery	**String**	エンドポイント設定をオーバーライドする **String** 値として URI クエリーパラメーターを提供できます。& 記号を使用して、複数のパラメーターを区切ります。例: **foo=bar&beer=yes** .
CamelHttpPath	**String**	エンドポイント設定をオーバーライドする **String** 値として URI コンテキストパスとクエリーパラメーターを指定できます。これにより、同じリモート http サーバーを呼び出すために同じプロデューサーを再利用できますが、動的なコンテキストパスとクエリーパラメーターを使用します。
Content-Type	**String**	HTTP ボディーの content-type を設定します。例： **text/plain; charset="UTF-8"**。
CamelHttpResponseCode	**int**	使用する HTTP ステータスコードを設定できます。デフォルトでは、成功には 200、失敗には 500 が使用されます。

ルートが Netty4 HTTP エンドポイントから開始する場合、次のヘッダーがメタデータとして提供されます。

表の説明は、次のルートでオフセットを取ります: **from("netty4-http:http:0.0.0.0:8080/myapp")…**

名前	タイプ	説明
CamelHttpMethod	**String**	GET、POST、TRACE など、使用される HTTP メソッド。
CamelHttpUrl	**String**	プロトコル、ホスト、ポートなどを含む URL: **http://0.0.0.0:8080/myapp**
CamelHttpUri	**String**	プロトコル、ホスト、ポートなどを含まない URI: **/myapp**
CamelHttpQuery	**String**	**foo=bar&beer=yes** などの任意のクエリーパラメーター。

名前	タイプ	説明
CamelHttpRawQuery	**String**	**foo=bar&beer=yes** などの任意のクエリーパラメーター。コンシューマーに到着したとき (つまり、URL デコード前) に未加工の形式で保存されます。
CamelHttpPath	**String**	追加のコンテキストパス。クライアントが context-path /**myapp** を呼び出した場合、この値は空です。クライアントが /**myapp**/**mystuff** を呼び出す場合、このヘッダー値は /**mystuff** です。つまり、ルートエンドポイントで設定された context-path の後の値です。
CamelHttpCharacterEncoding	**String**	content-type ヘッダーの文字セット。
CamelHttpAuthentication	**String**	ユーザーが HTTP Basic を使用して認証された場合、このヘッダーには値 **Basic** が追加されます。
Content-Type	**String**	コンテンツタイプ (提供されている場合)。例： **text/plain; charset="UTF-8"**。

230.4. NETTY 型へのアクセス

このコンポーネントは、Exchange でのメッセージ実装として **org.apache.camel.component.netty4.http.NettyHttpMessage** を使用します。これにより、以下に示すように、エンドユーザーは必要に応じて元の Netty リクエスト/レスポンスインスタンスにアクセスできます。 元のレスポンスに常にアクセスできるとは限らないことに注意してください。

```
io.netty.handler.codec.http.HttpRequest request =
exchange.getIn(NettyHttpMessage.class).getHttpRequest();
```

230.5. 例

以下のルートでは、ハードコーディングされた Bye World メッセージを返す HTTP サーバーとして Netty4 HTTP を使用します。

```
    from("netty4-http:http://0.0.0.0:8080/foo")
      .transform().constant("Bye World");
```

また、以下に示すように、ProducerTemplate を使用して、Camel を使用してこの HTTP サーバーを呼び出すこともできます。

```
    String out = template.requestBody("netty4-http:http://localhost:8080/foo", "Hello World",
String.class);
    System.out.println(out);
```

出力として Bye World が返されます。

230.6. NETTY にワイルドカードを一致させるにはどうすればよいですか

デフォルトでは、Netty4 HTTP は正確な uri にのみ一致します。ただし、Netty に接頭辞を一致させるように指示することはできます。以下に例を示します。

```
from("netty4-http:http://0.0.0.0:8123/foo").to("mock:foo");
```

上記のルートで Netty4 HTTP は uri が完全一致する場合のみ一致するため、次のように入力すると一致します。
http://0.0.0.0:8123/foo。しかし、**http://0.0.0.0:8123/foo/bar** を実行すると一致しません。

したがって、ワイルドカードマッチングを有効にする場合は、次のようにします。

```
from("netty4-http:http://0.0.0.0:8123/foo?matchOnUriPrefix=true").to("mock:foo");
```

したがって、Netty は **foo** で始まるすべてのエンドポイントに一致します。

任意の エンドポイントに一致させるには、次のようにします。

```
from("netty4-http:http://0.0.0.0:8123?matchOnUriPrefix=true").to("mock:foo");
```

230.7. 同じポートで複数のルートを使用する

同じ CamelContext では、同じポート (例: **io.netty.bootstrap.ServerBootstrap** インスタンス) を共有する Netty4 HTTP からの複数のルートを持つことができます。ルートは同じ **io.netty.bootstrap.ServerBootstrap** インスタンスを共有するため、これを行うには、ルート内で多数のブートストラップオプションを同一にする必要があります。インスタンスは、最初に作成されたルートのオプションで設定されます。

ルートが同一に設定されている必要があるオプションは、**org.apache.camel.component.netty4.NettyServerBootstrapConfiguration** 設定クラスで定義されているすべてです。異なるオプションで別のルートを設定した場合、Camel は起動時に例外を出力し、オプションが同一でないことを示します。これを軽減するには、すべてのオプションが同一であることを確認してください。

同じポートを共有する 2 つのルートの例を次に示します。

同じポートを共有する 2 つのルート

```
from("netty4-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

from("netty4-http:http://0.0.0.0:{{port}}/bar")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

そして、これは、1番目のルートと同一の **org.apache.camel.component.netty4.NettyServerBootstrapConfiguration** オプションを持たない、誤って設定された 2 番目のルートの例です。これにより、Camel は起動時に失敗します。

2 つのルートが同じポートを共有しているが、2 番目のルートが正しく設定されておらず、起動に失敗する

```
from("netty4-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

// we cannot have a 2nd route on same port with SSL enabled, when the 1st route is NOT
from("netty4-http:http://0.0.0.0:{{port}}/bar?ssl=true")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

230.7.1. 複数のルートで同じサーバーのブートストラップ設定を再利用する

org.apache.camel.component.netty4.NettyServerBootstrapConfiguration タイプの単一インスタンスで共通サーバーブートストラップオプションを設定することにより、Netty4 HTTP コンシューマーで **bootstrapConfiguration** オプションを使用して、すべてのコンシューマーで同じオプションを参照および再利用できます。

```
<bean id="nettyHttpBootstrapOptions"
class="org.apache.camel.component.netty4.NettyServerBootstrapConfiguration">
  <property name="backlog" value="200"/>
  <property name="connectionTimeout" value="20000"/>
  <property name="workerCount" value="16"/>
</bean>
```

そしてルートでは、以下に示すようにこのオプションを参照します

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/bar?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/beer?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>
```

230.7.2. OSGi コンテナー内の複数のバンドルにまたがる複数のルートで同じサーバーのブートストラップ設定を再利用する

詳細とその方法の例については、Netty HTTP サーバーの例を参照してください。

230.8. HTTP BASIC 認証の使用

以下に示すように、Netty HTTP コンシューマーは、使用するセキュリティーレルム名を指定することにより、HTTP Basic 認証をサポートします。

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?securityConfiguration.realm=karaf"/>
  ...
</route>
```

Basic 認証を有効にするには、レルム名が必須です。デフォルトでは、JAAS ベースのオーセンティケーターが使用されます。これは、指定されたレルム名 (上記の例では karaf) を使用し、JAAS レルムとこのレルムの JAAS \{{LoginModule}} を認証に使用します。

Apache Karaf/ServiceMix のエンドユーザーはすぐに使用できる karaf レルムを持っているため、上記の例がこれらのコンテナーですぐに機能する理由です。

230.8.1. Web リソースに ACL を指定する

org.apache.camel.component.netty4.http.SecurityConstraint を使用すると、Web リソースに対する制約を定義できます。また、**org.apache.camel.component.netty.http.SecurityConstraintMapping** はすぐに使用できるように提供されており、ロールを使用してインクルージョンとエクスクルージョンを簡単に定義できます。

たとえば、XML DSL で以下に示すように、制約 Bean を定義します。

```
<bean id="constraint" class="org.apache.camel.component.netty4.http.SecurityConstraintMapping">
  <!-- inclusions defines url -> roles restrictions -->
  <!-- a * should be used for any role accepted (or even no roles) -->
  <property name="inclusions">
    <map>
      <entry key="/*" value="*"/>
      <entry key="/admin/*" value="admin"/>
      <entry key="/guest/*" value="admin,guest"/>
    </map>
  </property>
  <!-- exclusions is used to define public urls, which requires no authentication -->
  <property name="exclusions">
    <set>
      <value>/public/*</value>
    </set>
  </property>
</bean>
```

上記の制約は、次のように定義されます。

- /* へのアクセスは制限され、すべてのロールが受け入れられます (ユーザーがロールを持っていない場合も同様)
- /admin/* へのアクセスには管理者ロールが必要です
- /guest/* へのアクセスには、管理者またはゲストのロールが必要です
- /public/* へのアクセスは、認証が不要であることを意味するエクスクルージョンであるため、ログインしなくても全員に公開されます

この制約を使用するには、以下に示すように Bean ID を参照するだけです。

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?
matchOnUriPrefix=true&amp;securityConfiguration.realm=karaf&amp;securityConfiguration.securityCon
straint=#constraint"/>
  ...
</route>
```

230.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Netty
- Netty HTTP サーバーの例
- Jetty

第231章 OGNL 言語

Camel バージョン 1.1 以降で利用可能

Camel では、OGNL を DSL または Xml 設定の式または述語として使用できます。

OGNL を使用して、メッセージフィルターで述語を作成したり、受信者リストの式として使用したりできます。

OGNL ドット表記を使用して操作を呼び出すことができます。たとえば、**getFamilyName** メソッドを持つ POJO を含むボディーがある場合、次のように構文を作成できます。

```
"request.body.familyName"
  // or
"getRequest().getBody().getFamilyName()"
```

231.1. OGNL オプション

OGNL 言語は、以下にリストされている 1 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

231.2. VARIABLES

変数	タイプ	説明
this	Exchange	Exchange はルートオブジェクトです
exchange	Exchange	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)
exchangeId	String	エクスチェンジ ID
fault	メッセージ	Fault メッセージ (ある場合)
request	メッセージ	exchange.in メッセージ

変数	タイプ	説明
response	メッセージ	exchange.out メッセージ (存在する場合)
properties	Map	エクスチェンジプロパティー
property(name)	Object	指定された名前によるプロパティー
property(name, type)	タイプ	指定されたタイプとして指定された名前によるプロパティー

231.3. サンプル

たとえば、XML の メッセージフィルター 内で OGNL を使用できます。

```
<route>
  <from uri="seda:foo"/>
  <filter>
    <ognl>request.headers.foo == 'bar'</ognl>
    <to uri="seda:bar"/>
  </filter>
</route>
```

Java DSL を使用したサンプル:

```
from("seda:foo").filter().ognl("request.headers.foo == 'bar'").to("seda:bar");
```

231.4. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").ognl("resource:classpath:myognl.txt")
```

231.5. 依存関係

camel ルートで OGNL を使用するには、OGNL 言語を実装する **camel-ognl** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ognl</artifactId>
  <version>x.x.x</version>
</dependency>
```

それ以外の場合は、OGNL も必要になります

第232章 OLINGO2 コンポーネント

Camel バージョン 2.14 以降で利用可能

Olingo2 コンポーネントは、Apache Olingo バージョン 2.0 API を使用して、OData 2.0 準拠のサービスと対話します。多くの一般的な商用およびエンタープライズベンダーと製品が OData プロトコルをサポートしています。サポートされている製品のサンプルリストは、OData の Web サイト にあります。

Olingo2 コンポーネントは、カスタムおよび OData システムクエリーパラメーターを使用して、フィード、デルタフィード、エンティティー、単純および複雑なプロパティー、リンク、カウントの読み取りをサポートします。エンティティー、プロパティー、関連付けリンクの更新をサポートしています。また、クエリーと変更リクエストを単一の OData バッチ操作として送信することもサポートしています。

このコンポーネントは、OData サービス接続用の HTTP 接続パラメーターとヘッダーの設定をサポートしています。これにより、ターゲット OData サービスで必要な SSL、OAuth2.0 などの使用を設定できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-olingo2</artifactId>
  <version>${camel-version}</version>
</dependency>
```

232.1. URI 形式

```
olingo2://endpoint/<resource-path>?[options]
```

232.2. OLINGO2 オプション

Olingo2 コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		Olingo2Configuration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Olingo2 エンドポイントは、URI 構文を使用して設定されます。

```
olingo2:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

232.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		Olingo2ApiName
methodName	**必須**: 選択した操作に使用するサブ操作		String

232.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
connectTimeout (common)	ミリ秒単位の HTTP 接続作成タイムアウト。デフォルトは 30,000 (30 秒)	30000	int
contentType (common)	Content-Type ヘッダー値を使用して、JSON または XML メッセージ形式を指定できます。デフォルトは application/json;charset=utf-8 です。	application/json;charset=utf-8	String
httpAsyncClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP 非同期クライアントビルダーは、connectionTimeout、socketTimeout、proxy、および sslContext をオーバーライドします。ビルダーで socketTimeout を指定する必要があることに注意してください。そうしないと、OData 要求が無期限にブロックされる可能性があります。		HttpAsyncClientBuilder
httpClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP クライアントビルダーは、connectionTimeout、socketTimeout、proxy、および sslContext をオーバーライドします。ビルダーで socketTimeout を指定する必要があることに注意してください。そうしないと、OData 要求が無期限にブロックされる可能性があります。		HttpClientBuilder
httpHeaders (common)	すべてのリクエストに挿入するカスタム HTTP ヘッダー。これには OAuth トークンなどが含まれる場合があります。		Map
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String

名前	説明	デフォルト	タイプ
proxy (common)	HTTP プロキシーサーバーの設定		HttpHost
serviceUri (common)	ターゲット OData サービスベース URI (例: http://services.odata.org/OData/OData.svc)		String
socketTimeout (common)	ミリ秒単位の HTTP 要求タイムアウト。デフォルトは 30,000 (30 秒)	30000	int
sslContextParameters (common)	SSLContextParameters を使用してセキュリティーを設定する場合		SSLContextParameters
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

232.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、次に示すエンドポイント名とオプションを使用できます。 プロデューサエンドポイントは、特別なオプション **inBody** を 使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。**inBody** オプションのデフォルトは、そのオプションを使用するエンドポイントの **データ** です。

232.4. エンドポイントオプション

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelOlingo2.<option>** の形式である必要があります。**inBody** オプション

はメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelOlingo2.option** ヘッダーをオーバーライドします。さらに、クエリーパラメーターを指定できます。

名前	タイプ	説明
data	Object	OData リソースの作成または変更に使用される適切なタイプのデータ
keyPredicate	String	パラメーター化された OData リソースエンドポイントを作成するためのキー述語。キー述語の値がヘッダーで動的に提供される作成/更新操作に役立ちます
queryParams	java.util.Map<String,String>	OData システムオプションとカスタムクエリーオプション。詳細については、OData 2.0 URI 規則 を参照してください。
resourcePath	String	OData リソースパス。キー述語を含む場合と含まない場合があります
endpointHttpHeaders	java.util.Map<String, String>	エンドポイントに送信される動的 HTTP ヘッダー
responseHttpHeaders	java.util.Map<String, String>	エンドポイントからの動的 HTTP レスポンスヘッダー

resourcePath オプションは、URI パスの一部として、エンドポイントオプション ?resourcePath=<resource-path> として、またはヘッダー値 CamelOlingo2.resourcePath として、URI で指定できることに注意してください。OData エンティティーキーの述語は、**Manufacturers('1')** などのリソースパスの一部にすることもできます。ここで、**'_1'** はキーの述語です。または、リソースパスの **Manufacturers** と keyPredicate オプション **'1'** を使用して個別に指定することもできます。

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
batch	data, endpointHttpHeaders	マルチパート/混合バッチリクエストを使用した POST	java.util.List<org.apache.camel.component.olingo2.api.batch.Olingo2BatchResponse>

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
create	data, resourcePath, endpointHttpHeaders	POST	新しいエントリーの場合は org.apache.olingo.odata2.api.ep.entry.ODataEntry 他の OData リソースの場合は org.apache.olingo.odata2.api.commons.HttpStatusCodes
delete	resourcePath, endpointHttpHeaders	DELETE	org.apache.olingo.odata2.api.commons.HttpStatusCodes
merge	data, resourcePath, endpointHttpHeaders	MERGE	org.apache.olingo.odata2.api.commons.HttpStatusCodes
patch	data, resourcePath, endpointHttpHeaders	PATCH	org.apache.olingo.odata2.api.commons.HttpStatusCodes
read	queryParams, resourcePath, endpointHttpHeaders	GET	次に説明するように、クエリーされる OData リソースに依存します
update	data, resourcePath, endpointHttpHeaders	PUT	org.apache.olingo.odata2.api.commons.HttpStatusCodes

232.5. エンドポイント HTTP ヘッダー (2.20 以降)

コンポーネントレベルの設定プロパティー **httpHeaders** は、静的な HTTP ヘッダー情報を提供します。ただし、一部のシステムでは、エンドポイントとの間で動的なヘッダー情報をやり取りする必要があります。サンプルユースケースは、動的セキュリティートークンを必要とするシステムです。**endpointHttpHeaders** および **responseHttpHeaders** エンドポイントプロパティーは、この機能を提供します。**CamelOlingo2.endpointHttpHeaders** プロパティーでエンドポイントに渡す必要があるヘッダーを設定すると、レスポンスヘッダーが **CamelOlingo2.responseHttpHeaders** プロパティーで返されます。どちらのプロパティーも **java.util.Map<String, String>** 型です。

232.6. ODATA リソースタイプのマッピング

data オプションの **read** エンドポイントとデータ型の結果は、クエリー、作成、または変更される OData リソースによって異なります。

OData リソースタイプ	resourcePath と keyPredicate からのリソース URI	インまたはアウトのボディータイプ
エンティティーデータモデル	$metadata	org.apache.olingo.odata2.api.edm.Edm
サービスドキュメント	/	org.apache.olingo.odata2.api.servicedocument.ServiceDocument
OData フィード	<entity-set>	org.apache.olingo.odata2.api.ep.feed.ODataFeed
OData エントリー	<entity-set>(<key-predicate>)	出力ボディーの org.apache.olingo.odata2.api.ep.entry.ODataEntry (レスポンス) 入力ボディーの java.util.Map<String, Object> (リクエスト)

OData リソースタイプ	resourcePath と keyPredicate からのリソース URI	インまたはアウトのボディータイプ
単純なプロパティー	<entity-set>(<key-predicate>)/<simple-property>	Olingo EdmProperty で説明されている適切な Java データ型
単純なプロパティー値	<entity-set>(<key-predicate>)/<simple-property>/$value	Olingo EdmProperty で説明されている適切な Java データ型
複雑なプロパティー	<entity-set>(<key-predicate>)/<complex-property>	java.util.Map<String, Object>
ゼロまたは 1 つの関連リンク	<entity-set>(<key-predicate>/$link/<one-to-one-entity-set-property>	レスポンスの文字列 java.util.Map<String, Object> とリクエストのキープロパティー名と値

OData リソースタイプ	resourcePath と keyPredicate からのリソース URI	インまたはアウトのボディータイプ
ゼロまたは多数の関連リンク	<entity-set>(<key-predicate>/$link/<one-to-many-entity-set-property>	レスポンスの java.util.List<String> リクエストのキープロパティー名と値のリストを含む java.util.List<java.util.Map<String、Object>>
カウント	<resource-uri>/$count	java.lang.Long

232.7. コンシューマーエンドポイント

コンシューマーエンドポイントとして使用できるのは、**read** エンドポイントのみです。コンシューマーエンドポイントは、エンドポイントの呼び出しをスケジュールする接頭辞 **consumer.** で Scheduled Poll Consumer Options を使用できます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。この動作は、エンドポイントプロパティー **consumer.splitResult=false** を設定することで無効にできます。

232.8. メッセージヘッダー

CamelOlingo2 接頭辞を使用するプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

232.9. メッセージボディー

すべての結果メッセージ本文は、Olingo2Component によって使用される基礎となる Apache Olingo 2.0 API で提供されるオブジェクトを利用します。プロデューサエンドポイントは、**inBody** エンドポイント URI パラメーターで入力メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントの場合、**consumer.splitResult** が **false** に設定されていない限り、コンシューマーエンドポイントはすべての要素を個別のメッセージにマップします。

232.10. ユースケース

次のルートは、Manufacturer フィードから Name プロパティーの昇順で上位 5 つのエントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo2.$top", "5");
    .to("olingo2://read/Manufacturers?orderBy=Name%20asc");
```

次のルートは、入力 **id** ヘッダーのキープロパティー値を使用して Manufacturer エントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo2.keyPredicate", header("id"))
    .to("olingo2://read/Manufacturers");
```

次のルートは、ボディーメッセージで **java.util.Map<String, Object>** を使用して Manufacturer エントリーを作成します。

```
from("direct:...")
    .to("olingo2://create/Manufacturers");
```

次のルートは、Manufacturer の delta feed を 30 秒ごとにポーリングします。Bean **blah** は Bean **paramsBean** を更新して、更新された **!deltatoken** プロパティーを追加し、値は **ODataDeltaFeed** の結果で返されます。最初のデルタトークンが不明であるため、コンシューマーエンドポイントは最初に **ODataFeed** 値を生成し、その後のポーリングで **ODataDeltaFeed** を生成します。

```
from("olingo2://read/Manufacturers?
queryParams=#paramsBean&consumer.timeUnit=SECONDS&consumer.delay=30")
    .to("bean:blah");
```

第233章 OLINGO4 コンポーネント

Camel バージョン 2.19 以降で利用可能

Olingo4 コンポーネントは、Apache Olingo バージョン 4.0 API を使用して、OData 4.0 準拠のサービスと対話します。バージョン 4.0 以降、OData は OASIS 標準であり、多くの一般的なオープンソースおよび商用ベンダーと製品がこのプロトコルをサポートしています。サポートされている製品のサンプルリストは、OData の Web サイト にあります。

Olingo4 コンポーネントは、カスタムおよび OData システムクエリーパラメーターを使用して、エンティティーセット、エンティティー、単純および複雑なプロパティー、カウントの読み取りをサポートします。エンティティーとプロパティーの更新をサポートしています。また、クエリーと変更リクエストを単一の OData バッチ操作として送信することもサポートしています。

このコンポーネントは、OData サービス接続用の HTTP 接続パラメーターとヘッダーの設定をサポートしています。これにより、ターゲット OData サービスで必要な SSL、OAuth2.0 などの使用を設定できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-olingo4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

233.1. URI 形式

```
olingo4://endpoint/<resource-path>?[options]
```

233.2. OLINGO4 オプション

Olingo4 コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用するには		Olingo4Configuration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Olingo4 エンドポイントは、URI 構文を使用して設定されます。

```
olingo4:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

233.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		Olingo4ApiName
methodName	**必須**: 選択した操作に使用するサブ操作		String

233.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
connectTimeout (common)	ミリ秒単位の HTTP 接続作成タイムアウト。デフォルトは 30,000 (30 秒)	30000	int
contentType (common)	Content-Type ヘッダー値を使用して、JSON または XML メッセージ形式を指定できます。デフォルトは application/json;charset=utf-8 です。	application/json;charset=utf-8	String
httpAsyncClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP 非同期クライアントビルダーは、connectionTimeout、socketTimeout、proxy、および sslContext をオーバーライドします。ビルダーで socketTimeout を指定する必要があることに注意してください。そうしないと、OData 要求が無期限にブロックされる可能性があります。		HttpAsyncClientBuilder
httpClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP クライアントビルダーは、connectionTimeout、socketTimeout、proxy、および sslContext をオーバーライドします。ビルダーで socketTimeout を指定する必要があることに注意してください。そうしないと、OData 要求が無期限にブロックされる可能性があります。		HttpClientBuilder
httpHeaders (common)	すべてのリクエストに挿入するカスタム HTTP ヘッダー。これには OAuth トークンなどが含まれる場合があります。		Map
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String

名前	説明	デフォルト	タイプ
proxy (common)	HTTP プロキシーサーバーの設定		HttpHost
serviceUri (common)	ターゲット OData サービスベース URI (例: http://services.odata.org/OData/OData.svc)		String
socketTimeout (common)	ミリ秒単位の HTTP 要求タイムアウト。デフォルトは 30,000 (30 秒)	30000	int
sslContextParameters (common)	SSLContextParameters を使用してセキュリティーを設定する場合		SSLContextParameters
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

233.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、次に示すエンドポイント名とオプションを使用できます。 プロデューサエンドポイントは、特別なオプション **inBody** を 使用することもできます。このオプションには、値が Camel Exchange In メッセージに含まれるエンドポイントオプションの名前が含まれている必要があります。**inBody** オプションのデフォルトは、そのオプションを使用するエンドポイントの **データ** です。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelOlingo4.<option>** の形式である必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプション **inBody=option** は **CamelOlingo4.option** ヘッダーをオーバーライドします。さらに、クエリーパラメーターを指定できます

resourcePath オプションは、URI パスの一部として、エンドポイントオプション ?resourcePath=<resource-path> として、またはヘッダー値 CamelOlingo4.resourcePath として URI で指定できることに注意してください。OData エンティティーキーの述語は、**Manufacturers('1')** などのリソースパスの一部にすることもできます。ここで、**'1'** はキーの述語です。または、リソースパスの **Manufacturers** と keyPredicate オプション **'1'** を使用して個別に指定することもできます。

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
batch	data, endpointHttpHeaders	マルチパート/混合バッチリクエストを使用した POST	java.util.List<org.apache.camel.component.olingo4.api.batch.Olingo4BatchResponse>
create	data, resourcePath, endpointHttpHeaders	POST	新しいエントリーの org.apache.olingo.client.api.domain.ClientEntity 他の OData リソースの org.apache.olingo.commons.api.http.HttpStatusCode
delete	resourcePath, endpointHttpHeaders	DELETE	org.apache.olingo.commons.api.http.HttpStatusCode
merge	data, resourcePath, endpointHttpHeaders	MERGE	org.apache.olingo.commons.api.http.HttpStatusCode
patch	data, resourcePath, endpointHttpHeaders	PATCH	org.apache.olingo.commons.api.http.HttpStatusCode

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
read	queryParams, resourcePath, endpointHttpHeaders	GET	次に説明するように、クエリーされる OData リソースに依存します
update	data, resourcePath, endpointHttpHeaders	PUT	org.apache.olingo.commons.api.http.HttpStatusCode

233.4. エンドポイント HTTP ヘッダー (CAMEL 2.20 以降)

コンポーネントレベルの設定プロパティー **httpHeaders** は、静的な HTTP ヘッダー情報を提供します。ただし、一部のシステムでは、エンドポイントとの間で動的なヘッダー情報をやり取りする必要があります。サンプルユースケースは、動的セキュリティートークンを必要とするシステムです。**endpointHttpHeaders** および **responseHttpHeaders** エンドポイントプロパティーは、この機能を提供します。**CamelOlingo4.endpointHttpHeaders** プロパティーでエンドポイントに渡す必要があるヘッダーを設定すると、レスポンスヘッダーが **CamelOlingo4.responseHttpHeaders** プロパティーで返されます。どちらのプロパティーも **java.util.Map<String, String>** 型です。

233.5. ODATA リソースタイプのマッピング

data オプションの **read** エンドポイントとデータ型の結果は、クエリー、作成、または変更される OData リソースによって異なります。

OData リソースタイプ	resourcePath と keyPredicate からのリソース URI	インまたはアウトのボディータイプ
エンティティーデータモデル	$metadata	org.apache.olingo.commons.api.edm.Edm
サービスドキュメント	/	org.apache.olingo.client.api.domain.ClientServiceDocument
OData エンティティーセット	<entity-set>	org.apache.olingo.client.api.domain.ClientEntitySet
OData エンティティー	<entity-set>(<key-predicate>)	出力ボディーの org.apache.olingo.client.api.domain.ClientEntity (レスポンス) 入力ボディーの java.util.Map<String, Object> (リクエスト)
単純なプロパティー	<entity-set>(<key-predicate>)/<simple-property>	org.apache.olingo.client.api.domain.ClientPrimitiveValue
単純なプロパティー値	<entity-set>(<key-predicate>)/<simple-property>/$value	org.apache.olingo.client.api.domain.ClientPrimitiveValue

OData リソースタイプ	resourcePath と keyPredicate からのリソース URI	インまたはアウトのボディータイプ
複雑なプロパティー	<entity-set>(<key-predicate>)/<complex-property>	org.apache.olingo.client.api.domain.ClientComplexValue
カウント	<resource-uri>/$count	java.lang.Long

233.6. コンシューマーエンドポイント

コンシューマーエンドポイントとして使用できるのは、**read** エンドポイントのみです。コンシューマーエンドポイントは、エンドポイントの呼び出しをスケジュールする接頭辞 **consumer.** で Scheduled Poll Consumer Options を使用できます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。この動作は、エンドポイントプロパティー **consumer.splitResult=false** を設定することで無効にできます。

233.7. メッセージヘッダー

CamelOlingo4 接頭辞を使用するプロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプションを指定できます。

233.8. メッセージボディー

すべての結果メッセージボディーは、Olingo4Component によって使用される基礎となる Apache Olingo 4.0 API によって提供されるオブジェクトを利用します。プロデューサエンドポイントは、**inBody** エンドポイント URI パラメーターで入力メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントの場合、**consumer.splitResult** が **false** に設定されていない限り、コンシューマーエンドポイントはすべての要素を個別のメッセージにマップします。

233.9. ユースケース

次のルートは、FirstName プロパティーの昇順で並べ替えられた People エンティティーから上位 5 つのエントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo4.$top", "5");
    .to("olingo4://read/People?orderBy=FirstName%20asc");
```

次のルートは、入力 **id** ヘッダーの key プロパティー値を使用して Airports エンティティーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo4.keyPredicate", header("id"))
    .to("olingo4://read/Airports");
```

次のルートは、ボディーメッセージで **ClientEntity** を使用して People エンティティーを作成します。

```
from("direct:...")
    .to("olingo4://create/People");
```

第234章 OPENSHIFT コンポーネント (非推奨)

Camel バージョン 2.14 以降で利用可能

openshift コンポーネントは、OpenShift アプリケーションを管理するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openshift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

234.1. URI 形式

```
openshift:clientId[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

234.2. オプション

OpenShift コンポーネントは、以下に示す 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
username (security)	openshift サーバーにログインするためのユーザー名。		String
password (security)	openshift サーバーにログインするためのパスワード。		String
domain (Common)	Domain name.指定しない場合、デフォルトのドメインが使用されます。		String
server (common)	openshift サーバーへの URL。指定しない場合、ローカルの openshift 設定ファイル/.openshift/express.conf のデフォルト値が使用されます。それも失敗した場合は、openshift.redhat.com が使用されます。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OpenShift エンドポイントは、URI 構文を使用して設定されます。

```
openshift:clientId
```

パスおよびクエリーパラメーターを使用します。

234.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
clientId	**必須** クライアント ID		String

234.2.2. クエリーパラメーター(26 個のパラメーター):

名前	説明	デフォルト	タイプ
domain (Common)	Domain name.指定しない場合、デフォルトのドメインが使用されます。		String
password (common)	**必須** openshift サーバーにログインするためのパスワード。		String
server (common)	openshift サーバーへの URL。指定しない場合、ローカルの openshift 設定ファイル/.openshift/express.conf のデフォルト値が使用されます。それも失敗した場合は、openshift.redhat.com が使用されます。		String
username (common)	**必須** openshift サーバーにログインするためのユーザー名。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
application (producer)	開始、停止、再起動、または状態を取得するアプリケーション名。		String
mode (producer)	メッセージ本文を pojo または json として出力するかどうか。pojo の場合、メッセージは List 型です。		String
operation (producer)	実行する操作: list、start、stop、restart、および state。リスト操作は、すべてのアプリケーションに関する情報を json 形式で返します。状態操作は、開始、停止などの状態を返します。他の操作は値を返しません。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

名前	説明	デフォルト	タイプ
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

234.3. 例

234.3.1. すべてのアプリケーションの一覧表示

```
// sending route
from("direct:apps")
```

```
    .to("openshift:myClient?username=foo&password=secret&operation=list");
    .to("log:apps");
```

この場合、すべてのアプリケーションに関する情報が pojo として返されます。json レスポンスが必要な場合は、mode=json を設定します。

234.3.2. アプリケーションの停止

```
// stopping the foobar application
from("direct:control")
    .to("openshift:myClient?username=foo&password=secret&operation=stop&application=foobar");
```

上記の例では、foobar という名前のアプリケーションを停止します。

ギア状態の変更のポーリング

コンシューマーは、ギアの状態の変更をポーリングするために使用されます。新しいギアが追加/削除/またはそのライフサイクルが変更されたときなど (開始、停止など)。

```
// trigger when state changes on our gears
from("openshift:myClient?username=foo&password=secret&delay=30s")
    .log("Event ${header.CamelOpenShiftEventType} on application ${body.name} changed state to
${header.CamelOpenShiftEventNewState}");
```

コンシューマーが Exchange を発行すると、ボディーにはメッセージボディーとして **com.openshift.client.IApplication** が含まれます。そして、以下のヘッダーが含まれています。

ヘッダー	null の可能性があります	説明
CamelOpenShiftEventType	いいえ	追加、削除、または変更のいずれかのイベントのタイプ。
CamelOpenShiftEventOldState	○	イベントタイプが変更されたときの古い状態。
CamelOpenShiftEventNewState	いいえ	いずれかのイベントタイプの新しい状態

234.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第235章 OPENSHIFT BUILD CONFIG コンポーネント

Camel バージョン 2.17 以降で利用可能

OpenShift Build Config コンポーネントは、kubernetes ビルド設定操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

235.1. コンポーネントオプション

Openshift Build Config コンポーネントにはオプションがありません。

235.2. エンドポイントオプション

Openshift Build Config エンドポイントは、URI 構文を使用して設定されます。

```
openshift-build-configs:masterUrl
```

パスおよびクエリーパラメーターを使用します。

235.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

235.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		String

第236章 OPENSHIFT BUILDS コンポーネント

Camel バージョン 2.17 以降で利用可能

Kubernetes Builds コンポーネントは、kubernetes ビルド操作を実行するプロデューサーを提供する Kubernetes コンポーネント の 1 つです。

236.1. コンポーネントオプション

Openshift Builds コンポーネントにはオプションがありません。

236.2. エンドポイントオプション

Openshift Builds エンドポイントは、URI 構文を使用して設定されます。

```
openshift-builds:masterUrl
```

パスおよびクエリーパラメーターを使用します。

236.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
masterUrl	**必須** Kubernetes Master URL		文字列

236.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (producer)	使用する Kubernetes API バージョン		String
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		String
kubernetesClient (producer)	提供される場合に使用するデフォルトの KubernetesClient		KubernetesClient
operation (producer)	Kubernetes で実行するプロデューサー操作		String
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。		Integer

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
caCertData (security)	CA 証明書データ		String
caCertFile (security)	CA 証明書ファイル		String
clientCertData (security)	クライアント証明書データ		String
clientCertFile (security)	クライアント証明書ファイル		String
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		String
clientKeyData (security)	クライアントキーデータ		String
clientKeyFile (security)	クライアントキーファイル		String
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		String
oauthToken (security)	認証トークン		String
password (security)	Kubernetes に接続するパスワード		String
trustCerts (security)	使用した証明書が信頼できるかどうかを定義します。		Boolean
username (security)	Kubernetes に接続するユーザー名		String

236.3. OPENSTACK コンポーネント

Camel 2.19 以降で利用可能

openstack コンポーネントは、OpenStack アプリケーションを管理するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

OpenStack サービス	Camel コンポーネント	説明
OpenStack Cinder	openstack-cinder	OpenStack cinder を維持するためのコンポーネント。
OpenStack Glance	openstack-glance	OpenStack の外観を維持するためのコンポーネント。
OpenStack Keystone	openstack-keystone	OpenStack キーストーンを維持するためのコンポーネント。
OpenStack Neutron	openstack-neutron	OpenStack neutron を維持するためのコンポーネント。
OpenStack Nova	openstack-nova	OpenStack nova を維持するためのコンポーネント。
OpenStack Swift	openstack-swift	OpenStack swift を維持するためのコンポーネント。

第237章 OPENSTACK CINDER COMPONENT

Camel バージョン 2.19 以降で利用可能

openstack-cinder コンポーネントを使用すると、メッセージを OpenStack ブロックストレージサービスに送信できます。

237.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

237.2. URI 形式

```
openstack-cinder://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

237.3. URI オプション

OpenStack Cinder コンポーネントにはオプションがありません。

OpenStack Cinder エンドポイントは、URI 構文を使用して設定されます。

```
openstack-cinder:host
```

パスおよびクエリーパラメーターを使用します。

237.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須の** OpenStack ホスト URL		String

237.3.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
apiVersion (producer)	OpenStack API バージョン	V3	String
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
subsystem (producer)	**必要** OpenStack Cinder サブシステム		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

237.4. 使用方法

サブシステムごとに次の設定を使用できます。

237.5. VOLUMES

237.5.1. ボリュームプロデューサーで実行できる操作

操作	説明
create	新規ボリュームを作成します。
get	ボリュームを取得します。
getAll	すべてのボリュームを取得します。
getAllTypes	ボリュームタイプを取得します。

操作	説明
update	ボリュームを更新します。
delete	ボリュームを削除します。

237.5.2. ボリュームプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	ボリュームの ID。
name	**String**	ボリューム名。
description	**String**	ボリュームの説明。
size	**Integer**	ボリュームのサイズ。
volumeType	**String**	ボリュームタイプ
imageRef	**String**	イメージの ID。
snapshotId	**String**	スナップショットの ID。
isBootable	**Boolean**	起動可能です。

より正確なボリューム設定が必要な場合は、タイプ org.openstack4j.model.storage.block.Volume の新しいオブジェクトを作成し、メッセージボディーで送信できます。

237.6. SNAPSHOTS

237.6.1. スナップショットプロデューサーで実行できる操作

操作	説明
create	新しいスナップショットを作成します。
get	スナップショットを取得します。
getAll	すべてのスナップショットを取得します。
update	スナップショットを更新します。
delete	スナップショットを削除します。

237.6.2. スナップショットプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	サーバーの ID。
name	**String**	サーバー名。
description	**String**	スナップショットの説明。
VolumeId	**String**	ボリューム ID。
force	**Boolean**	フォース。

より正確なサーバー設定が必要な場合は、タイプ **org.openstack4j.model.storage.block.VolumeSnapshot** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

237.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- openstack コンポーネント

第238章 OPENSTACK GLANCE コンポーネント

Camel バージョン 2.19 以降で利用可能

openstack-glance コンポーネントを使用すると、メッセージを OpenStack イメージサービスに送信できます。

238.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

238.2. URI 形式

```
openstack-glance://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

238.3. URI オプション

OpenStack Glance コンポーネントにはオプションがありません。

OpenStack Glance エンドポイントは、URI 構文を使用して設定されます。

```
openstack-glance:host
```

パスおよびクエリーパラメーターを使用します。

238.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須の** OpenStack ホスト URL		String

238.3.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
apiVersion (producer)	OpenStack API バージョン	V3	String
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

238.4. 使用方法

操作	説明
reserve	Reserve イメージ。
create	新しいイメージを作成します。
update	イメージを更新します。
upload	イメージのアップロード。
get	イメージを取得します。
getAll	すべてのイメージを取得します。
delete	イメージを削除します。

238.4.1. Glance プロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	フレーバーの ID。
name	**String**	フレーバー名。
diskFormat	**org.openstack4j.model.image.DiskFormat**	フレーバー VCPU の数。
containerFormat	**org.openstack4j.model.image.ContainerFormat**	RAM のサイズ。
owner	**String**	イメージの所有者。
isPublic	**Boolean**	公開されています。
minRam	**Long**	最小 ram。
minDisk	**Long**	最小ディスク。
size	**Long**	サイズ。
checksum	**String**	Checksum。
properties	**Map**	Image プロパティー。

238.5. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント
- エンドポイント
- スタートガイド
- openstack コンポーネント

第239章 OPENSTACK KEYSTONE コンポーネント

Camel バージョン 2.19 以降で利用可能

openstack-keystone コンポーネントを使用すると、メッセージを OpenStack ID サービスに送信できます。

openstack-keystone コンポーネントは、Identity API v3 のみをサポートしています!

239.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-openstack</artifactId>
    <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

239.2. URI 形式

```
openstack-keystone://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

239.3. URI オプション

OpenStack Keystone コンポーネントにはオプションがありません。

OpenStack Keystone エンドポイントは、URI 構文を使用して設定されます。

```
openstack-keystone:host
```

パスおよびクエリーパラメーターを使用します。

239.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須の** OpenStack ホスト URL		String

239.3.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
subsystem (producer)	**必要** OpenStack Keystone サブシステム		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

239.4. 使用方法

サブシステムごとに次の設定を使用できます。

239.5. DOMAINS

239.5.1. ドメインプロデューサーで実行できる操作

操作	説明
create	新規ドメインを作成します。
get	ドメインを取得します。
getAll	すべてのドメインを取得します。
update	ドメインを更新します。
delete	ドメインを削除します。

239.5.2. ドメインプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	ドメインの ID。
name	**String**	ドメイン名。
description	**String**	ドメインの説明。

より正確なドメイン設定が必要な場合は、タイプ org.openstack4j.model.identity.v3.Domain の新しいオブジェクトを作成し、メッセージ本文で送信できます。

239.6. GROUPS

239.6.1. グループプロデューサーで実行できる操作

操作	説明
create	新規グループを作成します。
get	グループを取得します。
getAll	すべてのグループを取得します。
update	グループを更新します。
delete	グループを削除します。
addUserToGroup	ユーザーをグループに追加します。
checkUserGroup	ユーザーがグループ内にあるかどうかを確認します。
removeUserFromGroup	グループからユーザーを削除します。

239.6.2. グループプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
groupId	**String**	グループの ID。
name	**String**	グループ名。
userId	**String**	ユーザーの ID。
domainId	**String**	ドメインの ID。
description	**String**	グループの説明。

より正確なグループ設定が必要な場合は、タイプ org.openstack4j.model.identity.v3.Group の新しいオブジェクトを作成し、メッセージボディーで送信できます。

239.7. PROJECTS

239.7.1. プロジェクトプロデューサーで実行できる操作

操作	説明
create	新規プロジェクトを作成します。
get	プロジェクトを取得します。
getAll	すべてのプロジェクトを取得します。
update	プロジェクトを更新します。
delete	プロジェクトを削除します。

239.7.2. プロジェクトプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。

ヘッダー	タイプ	説明
ID	**String**	プロジェクトの ID。
name	**String**	プロジェクト名。
description	**String**	プロジェクトの説明。
domainId	**String**	ドメインの ID。
parentId	**String**	親プロジェクト ID。

より正確なプロジェクト設定が必要な場合は、タイプ **org.openstack4j.model.identity.v3.Project** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

239.8. REGIONS

239.8.1. Region プロデューサーで実行できる操作

操作	説明
create	新規リージョンを作成します。
get	リージョンを取得します。
getAll	すべてのリージョンを取得します。
update	リージョンを更新します。
delete	リージョンを削除します。

239.8.2. リージョンプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	リージョンの ID。

ヘッダー	タイプ	説明
description	**String**	リージョンの説明。

より正確なリージョン設定が必要な場合は、タイプ org.openstack4j.model.identity.v3.Region の新しいオブジェクトを作成し、メッセージボディーで送信できます。

239.9. USERS

239.9.1. ユーザープロデューサーで実行できる操作

操作	説明
create	新規ユーザーを作成します。
get	ユーザーを取得します。
getAll	すべてのユーザーを取得します。
update	ユーザーを更新します。
delete	ユーザーを削除します。

239.9.2. ユーザープロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	ユーザーの ID。
name	**String**	ユーザー名。
description	**String**	ユーザーの説明。
domainId	**String**	ドメインの ID。
password	**String**	ユーザーのパスワード。

ヘッダー	タイプ	説明
email	**String**	ユーザーの電子メール。

より正確なユーザー設定が必要な場合は、タイプ **org.openstack4j.model.identity.v3.User** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

239.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- openstack コンポーネント

第240章 OPENSTACK NEUTRON COMPONENT

Camel バージョン 2.19 以降で利用可能

openstack-neutron コンポーネントを使用すると、メッセージを OpenStack ネットワークサービスに送信できます。

240.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

240.2. URI 形式

```
openstack-neutron://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

240.3. URI オプション

OpenStack Neutron コンポーネントにはオプションがありません。

OpenStack Neutron エンドポイントは、URI 構文を使用して設定されます。

```
openstack-neutron:host
```

パスおよびクエリーパラメーターを使用します。

240.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須の** OpenStack ホスト URL		String

240.3.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
apiVersion (producer)	OpenStack API バージョン	V3	String
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
subsystem (producer)	**必要** OpenStack Neutron サブシステム		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

240.4. 使用方法

サブシステムごとに次の設定を使用できます。

240.5. NETWORKS

240.5.1. ネットワークプロデューサーで実行できる操作

操作	説明
create	新規ネットワークを作成します。
get	ネットワークを取得します。
getAll	すべてのネットワークを取得します。
delete	ネットワークを削除します。

操作	説明

240.5.2. ネットワークプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	ネットワークの ID。
name	**String**	ネットワーク名。
tenantId	**String**	Tenant ID.
adminStateUp	**Boolean**	AdminStateUp ヘッダー。
networkType	**org.openstack4j.model.network.NetworkType**	ネットワークタイプ。
physicalNetwork	**String**	物理ネットワーク。
segmentId	**String**	セグメント ID。
isShared	**Boolean**	共有される。
isRouterExternal	**Boolean**	ルーターは外部にあります。

より正確なネットワーク設定が必要な場合は、タイプ **org.openstack4j.model.network.Network** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

240.6. SUBNETS

240.6.1. サブネットプロデューサーで実行できる操作

操作	説明
create	新しいサブネットを作成します。
get	サブネットを取得します。
getAll	すべてのサブネットを取得します。
delete	サブネットを削除します。
action	サブネットでアクションを実行します。

240.6.2. サブネットプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	サブネットの ID。
name	**String**	サブネット名。
networkId	**String**	ネットワーク ID。
enableDHCP	**Boolean**	DHCP を有効化。
gateway	**String**	ゲートウェイ。

より正確なサブネット設定が必要な場合は、タイプ **org.openstack4j.model.network.Subnet** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

240.7. ポート

240.7.1. ポートプロデューサーで実行できる操作

操作	説明
create	新しいポートを作成します。
get	ポートを取得します。
getAll	すべてのポートを取得します。
update	ポートを更新します。
delete	ポートを削除します。

240.7.2. ポートプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
name	**String**	ポート名。
networkId	**String**	ネットワーク ID。
tenantId	**String**	Tenant ID。
deviceId	**String**	デバイス ID。
macAddress	**String**	MAC アドレス

240.8. ルーター

240.8.1. ルータープロデューサーで実行できる操作

操作	説明
create	新しいルーターを作成します。
get	ルーターを取得します。

操作	説明
getAll	すべてのルーターを取得します。
update	ルーターを更新します。
delete	ルーターを削除します。
attachInterface	インターフェイスを接続します。
detachInterface	インターフェイスを切り離します。

240.8.2. ポートプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
name	**String**	ルーター名。
routerId	**String**	ルーター ID。
subnetId	**String**	Subnet ID。
portId	**String**	Port ID。
interfaceType	**org.openstack4j.model.network.AttachInterfaceType**	インターフェイスの種類。
tenantId	**String**	Tenant ID。

240.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- スタートガイド
- openstack コンポーネント

第241章 OPENSTACK NOVA コンポーネント

Camel バージョン 2.19 以降で利用可能

openstack-nova コンポーネントを使用すると、メッセージを OpenStack コンピューティングサービスに送信できます。

241.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

241.2. URI 形式

```
openstack-nova://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

241.3. URI オプション

OpenStack Nova コンポーネントにはオプションがありません。

OpenStack Nova エンドポイントは、URI 構文を使用して設定されます。

```
openstack-nova:host
```

パスおよびクエリーパラメーターを使用します。

241.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須の** OpenStack ホスト URL		String

241.3.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
apiVersion (producer)	OpenStack API バージョン	V3	String
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
subsystem (producer)	**必要** OpenStack Nova サブシステム		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

241.4. 使用方法

サブシステムごとに次の設定を使用できます。

241.5. フレーバー

241.5.1. フレーバープロデューサーで実行できる操作

操作	説明
create	新規フレーバーを作成します。
get	フレーバーを取得します。
getAll	すべてのフレーバーを取得します。
delete	フレーバーを削除します。

操作	説明

241.5.2. フレーバープロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	フレーバーの ID。
name	**String**	フレーバー名。
VCPU	**Integer**	フレーバー VCPU の数。
ram	**Integer**	RAM のサイズ。
disk	**Integer**	ディスクのサイズ。
swap	**Integer**	スワップのサイズ。
rxtxFactor	**Integer**	Rxtx ファクター。

より正確なフレーバー設定が必要な場合は、タイプ **org.openstack4j.model.compute.Flavor** の新しいオブジェクトを作成し、メッセージボディーで送信できます。

241.6. SERVERS

241.6.1. サーバープロデューサーで実行できる操作

操作	説明
create	新しいサーバーを作成します。
createSnapshot	サーバーのスナップショットを作成します。
get	サーバーを取得します。

操作	説明
getAll	すべてのサーバーを取得します。
delete	サーバーを削除します。
action	サーバーでアクションを実行します。

241.6.2. サーバープロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
ID	**String**	サーバーの ID。
name	**String**	サーバー名。
ImageId	**String**	イメージ ID。
FlavorId	**String**	使用するフレーバーの ID。
KeypairName	**String**	キーペア名。
NetworkId	**String**	ネットワーク ID。
AdminPassword	**String**	新しいサーバーの管理者パスワード。
action	**org.openstack4j.model.compute.Action**	実行するアクション。

より正確なサーバー設定が必要な場合は、タイプ org.openstack4j.model.compute.ServerCreate の新しいオブジェクトを作成し、メッセージボディーで送信できます。

241.7. KEYPAIRS

241.7.1. キーペアプロデューサーで実行できる操作

操作	説明
create	新しいキーペアを作成します。
get	キーペアを取得します。
getAll	すべてのキーペアを取得します。
delete	キーペアを削除します。

241.7.2. キーペアプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
name	**String**	キーペア名。

241.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- openstack コンポーネント

第242章 OPENSTACK SWIFT COMPONENT

Camel バージョン 2.19 以降で利用可能

openstack-swift コンポーネントを使用すると、メッセージを OpenStack オブジェクトストレージサービスに送信できます。

242.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョンに置き換える必要があります。

242.2. URI 形式

```
openstack-swift://hosturl[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

242.3. URI オプション

OpenStack Swift コンポーネントにはオプションがありません。

OpenStack Swift エンドポイントは、URI 構文を使用して設定されます。

```
openstack-swift:host
```

パスおよびクエリーパラメーターを使用します。

242.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
host	***必須の*** OpenStack ホスト URL		String

242.3.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ
apiVersion (producer)	OpenStack API バージョン	V3	String
config (producer)	OpenStack 設定		Config
domain (producer)	認証ドメイン	default	String
operation (producer)	やるべき操作		String
password (producer)	**必須** OpenStack パスワード		String
project (producer)	**必須** プロジェクト ID		String
subsystem (producer)	**必要** OpenStack Swift サブシステム		String
username (producer)	**必須** OpenStack ユーザー名		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

242.4. 使用方法

サブシステムごとに次の設定を使用できます。

242.5. CONTAINERS

242.5.1. コンテナープロデューサーで実行できる操作

操作	説明
create	新規コンテナーを作成します。
get	コンテナーを取得します。
getAll	すべてのコンテナーを取得します。
update	コンテナーを更新します。

操作	説明
delete	コンテナーを削除します。
getMetadata	メタデータを取得します。
createUpdateMetadata	メタデータを作成/更新します。
deleteMetadata	メタデータを削除します。

242.5.2. ボリュームプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
name	**String**	コンテナー名。
X-Container-Meta-	**Map**	コンテナーメタデータの接頭辞。
X-Versions-Location	**String**	バージョンのロケーション。
X-Container-Read	**String**	ACL - コンテナー読み取り。
X-Container-Write	**String**	ACL - コンテナー書き込み。
limit	**Integer**	リストオプション - 制限。
marker	**String**	リストオプション - マーカー。
end_marker	**String**	リストオプション - エンドマーカー。
delimiter	**Character**	リストオプション - 区切り記号。
path	**String**	リストオプション - パス。

より正確なコンテナー設定が必要な場合は、コンテナーを一覧するために、タイプ org.openstack4j.model.storage.objects.createUpdateContainerOptions (作成または更新操作の場

合)、または **org.openstack4j.model.storage.option.ContainerListOptions** の新しいオブジェクトを作ってメッセージボディーに送信します。

242.6. OBJECTS

242.6.1. オブジェクトプロデューサーで実行できる操作

操作	説明
create	新しいオブジェクトを作成します。
get	オブジェクトを取得します。
getAll	すべてのオブジェクトを取得します。
update	オブジェクトを更新します。
delete	オブジェクトを削除します。
getMetadata	メタデータを取得します。
createUpdateMetadata	メタデータを作成/更新します。

242.6.2. オブジェクトプロデューサーによって評価されるメッセージヘッダー

ヘッダー	タイプ	説明
operation	**String**	実行する操作。
containerName	**String**	コンテナー名。
objectName	**String**	オブジェクト名。

242.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

- openstack コンポーネント

第243章 OPENTRACING コンポーネント

Camel 2.19 以降で利用可能

重要

Camel 2.21 以降では、OpenTracing Java API バージョン 0.31 以降と互換性のある OpenTracing complaint トレーサーを使用する必要があります。

camel-opentracing コンポーネントは、OpenTracing を使用して入力および出力 Camel メッセージをトレースおよびタイミングするために使用されます。

イベント (スパン) は、Camel との間で送受信される入力および出力メッセージに対してキャプチャーされます。

サポートされているトレーサーのリストについては、OpenTracing Web サイトを参照してください。

243.1. 設定

OpenTracing トレーサの設定プロパティーは次のとおりです。

オプション	デフォルト	説明
excludePatterns		パターンに一致する Camel メッセージのトレースを無効にする除外パターンを設定します。設定内容は、キーがパターンである Set<String> です。このパターンは Intercept のルールを使用します。

Camel アプリケーションの分散トレースを提供するように OpenTracing トレーサーを設定するには、次の 3 つの方法があります。

243.1.1. Explicit

選択した OpenTracing 準拠の Tracer に関連付けられた特定の依存関係と共に、**camel-opentracing** コンポーネントを POM に含めます。

OpenTracing サポートを明示的に設定するには、**OpenTracingTracer** をインスタンス化し、camel コンテキストを初期化します。オプションで **Tracer** を指定することも、代わりに **Registry** または **ServiceLoader** を使用して暗黙的に検出することもできます。

```
OpenTracingTracer ottracer = new OpenTracingTracer();
// By default it uses a Noop Tracer, but you can override it with a specific OpenTracing implementation.
ottracer.setTracer(...);
// And then initialize the context
ottracer.init(camelContext);
```

XML で OpenTracingTracer を使用するには、OpenTracing トレーサー Bean を定義するだけです。Camel はそれらを自動的に検出して使用します。

```
<bean id="tracer" class="..."/>
<bean id="ottracer" class="org.apache.camel.opentracing.OpenTracingTracer">
```

```
    <property name="tracer" ref="tracer"/>
  </bean>
```

243.1.2. Spring Boot

Spring Boot を使用している場合は、**camel-opentracing-starter** 依存関係を追加し、メインクラスに **@CamelOpenTracing** のアノテーションを付けて OpenTracing を有効にすることができます。

Tracer Bean がアプリケーションによって定義されていない限り、**Tracer** は camel コンテキストの **Registry** または **ServiceLoader** から暗黙的に取得されます。

243.1.3. Java Agent

3 番目の方法は、Java エージェントを使用して OpenTracing サポートを自動的に設定することです。

選択した OpenTracing 準拠の Tracer に関連付けられた特定の依存関係と共に、**camel-opentracing** コンポーネントを POM に含めます。

OpenTracing Java エージェントは、次の依存関係に関連付けられています。

```
<dependency>
  <groupId>io.opentracing.contrib</groupId>
  <artifactId>opentracing-agent</artifactId>
</dependency>
```

使用される **Tracer** は、キャメルコンテキスト **Registry** から、または **ServiceLoader** を使用して暗黙的にロードされます。

このエージェントの使用方法は、アプリケーションの実行方法によって異なります。camel-example-opentracing の **Service2** はエージェントをローカルフォルダーにダウンロードし、**exec-maven-plugin** を使用して **-javaagent** コマンドラインオプションでサービスを起動します。

243.2. 例

ここで、OpenTracing を設定する 3 つの方法を示す例を見つけることができます: camel-example-opentracing

第244章 OPTAPLANNER コンポーネント

Camel バージョン 2.13 以降で利用可能

optaplanner: コンポーネントは、メッセージに含まれるプランニングの問題を OptaPlanner で解決します。
例: 未解決の配車ルートの問題を入力すると、それが解決されます。

このコンポーネントは、Consumer を BestSolutionChangedEvent リスナーとしてサポートし、Solution および ProblemFactChange を処理するためのプロデューサーをサポートします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-optaplanner</artifactId>
  <version>x.x.x</version><!-- use the same version as your Camel core version -->
</dependency>
```

244.1. URI 形式

```
optaplanner:solverConfig[?options]
```

solverConfig は、SolverConfig のクラスパスローカル URI です (例: **/org/foo/barSolverConfig.xml**)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

244.2. OPTAPLANNER オプション

OptaPlanner コンポーネントにはオプションがありません。

OptaPlanner エンドポイントは、URI 構文を使用して設定されます。

```
optaplanner:configFile
```

パスおよびクエリーパラメーターを使用します。

244.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
configFile	**必須** ソルバーファイルのロケーションを指定します		String

244.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
solverId (common)	ソルバーインスタンスキーのユーザーへの SolverId を指定します。	DEFAULT_SOLVER	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
async (producer)	非同期モードで操作を実行するように指定します。	false	boolean
threadPoolSize (producer)	async が true の場合に使用するスレッドプールサイズを指定します。	10	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

244.3. メッセージヘッダー

名前	デフォルト値	タイプ	コンテキスト	説明
CamelOptaPlannerSolverId	null	String	共有	使用する SolverId を指定します。

名前	デフォルト値	タイプ	コンテキスト	説明
Camel OptaPlannerIsAsync	PUT	String	Producer	現在のスレッドをブロックするのではなく、Solution インスタンスの送信に別のスレッドを使用するかどうかを指定します。

244.4. メッセージボディー

Camel は IN ボディーのプランニング問題を受け取り、それを解決して OUT ボディーに返します。(v 2.16 以降) IN ボディーオブジェクトは、次のユースケースをサポートします。

- 本文が Solution のインスタンスである場合、solverId によって識別されるソルバーを使用して、同期的または非同期的に解決されます。
- 本文が ProblemFactChange のインスタンスである場合、addProblemFactChange がトリガーされます。処理が非同期の場合、結果を返す前に isEveryProblemFactChangeProcessed まで待機します。
- ボディーが上記のタイプのいずれでもない場合、プロデューサーは、solverId によって識別されるソルバーから最良の結果を返します。

244.5. 終了

解決には、**solverConfig** で指定された時間だけかかります。

```
<solver>
  ...
  <termination>
    <!-- Terminate after 10 seconds, unless it's not feasible by then yet -->
    <terminationCompositionStyle>AND</terminationCompositionStyle>
    <secondsSpentLimit>10</secondsSpentLimit>
    <bestScoreLimit>-1hard/0soft</bestScoreLimit>
  </termination>
  ...
<solver>
```

244.5.1. サンプル

OptaPlanner を使用して、ActiveMQ キューにあるプランニング問題を解決します。

```
from("activemq:My.Queue").
  .to("optaplanner:/org/foo/barSolverConfig.xml");
```

OptaPlanner を REST サービスとして公開します。

```
from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
  .to("optaplanner:/org/foo/barSolverConfig.xml");
```

244.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第245章 PAHO コンポーネント

Camel バージョン 2.16 以降で利用可能

Paho コンポーネントは、Eclipse Paho ライブラリー を使用して MQTT メッセージングプロトコルのコネクターを提供します。Paho は最も人気のある MQTT ライブラリーの 1 つであるため、これを Java プロジェクトに統合したい場合は、Camel Paho コネクターが最適です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paho</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Paho アーティファクトは Maven Central でホストされていないため、POM xml ファイルに Eclipse Paho リポジトリーを追加する必要があることに注意してください。

```
<repositories>
 <repository>
  <id>eclipse-paho</id>
  <url>https://repo.eclipse.org/content/repositories/paho-releases</url>
  <snapshots>
   <enabled>false</enabled>
  </snapshots>
 </repository>
</repositories>
```

245.1. URI 形式

```
paho:topic[?options]
```

topic はトピックの名前です。

245.2. オプション

Paho コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
brokerUrl (Common)	MQTT ブローカーの URL。		String
clientId (common)	MQTT クライアント識別子。		String
connectOptions (advanced)	クライアント接続オプション		MqttConnectOptions

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Paho エンドポイントは、URI 構文を使用して設定されます。

```
paho:topic
```

パスおよびクエリーパラメーターを使用します。

245.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
topic	**必須** トピックの名前		文字列

245.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
autoReconnect (Common)	接続が失われた場合、クライアントは自動的にサーバーへの再接続を試みます。	true	boolean
brokerUrl (Common)	MQTT ブローカーの URL。	tcp://localhost:1883	String
clientId (common)	MQTT クライアント識別子。		String
connectOptions (Common)	クライアント接続オプション		MqttConnectOptions
filePersistenceDirectory (Common)	ファイル永続化プロバイダーが使用するベースディレクトリー。		String
password (common)	MQTT ブローカーに対する認証に使用されるパスワード		String
persistence (Common)	使用するクライアントの永続性 - メモリーまたはファイル。	MEMORY	PahoPersistence

名前	説明	デフォルト	タイプ
qos (Common)	クライアントのサービス品質レベル (0 ~ 2)。	2	int
retained (Common)	オプションを保持します	false	boolean
userName (common)	MQTT ブローカーに対する認証に使用されるユーザー名		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

245.3. ヘッダー

次のヘッダーは、Paho コンポーネントによって認識されます。

ヘッダー	Java 定数	エンドポイントの種類	値のタイプ	説明
CamelMqttTopic	PahoConstants.MQTT_TOPIC	コンシューマー	String	トピックの名前。

ヘッダー	Java 定数	エンドポイントの種類	値のタイプ	説明
CamelPahoOverrideTopic	PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC	Producer	String	エンドポイントで指定されたトピックの代わりにオーバーライドして送信するトピックの名前

245.4. デフォルトのペイロードタイプ

デフォルトでは、Camel Paho コンポーネントは、MQTT メッセージから抽出された (または挿入された) バイナリーペイロードで動作します。

```
// Receive payload
byte[] payload = (byte[]) consumerTemplate.receiveBody("paho:topic");

// Send payload
byte[] payload = "message".getBytes();
producerTemplate.sendBody("paho:topic", payload);
```

もちろん、Camel 組み込み 型変換 API は、自動データ型変換を実行できます。以下の例では、Camel は自動的にバイナリーペイロードを **String** に (およびその逆に) 変換します。

```
// Receive payload
String payload = consumerTemplate.receiveBody("paho:topic", String.class);

// Send payload
String payload = "message";
producerTemplate.sendBody("paho:topic", payload);
```

245.5. サンプル

たとえば、次のスニペットは、Camel ルーターと同じホストにインストールされた MQTT ブローカーからメッセージを読み取ります。

```
from("paho:some/queue")
    .to("mock:test");
```

以下のスニペットは MQTT ブローカーにメッセージを送信します:

```
from("direct:test")
    .to("paho:some/target/queue");
```

たとえば、これはリモート MQTT ブローカーからメッセージを読み取る方法です。

```
from("paho:some/queue?brokerUrl=tcp://iot.eclipse.org:1883")
    .to("mock:test");
```

ここでは、デフォルトのトピックをオーバーライドして、動的トピックに設定します

```
from("direct:test")
    .setHeader(PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC, simple("${header.customerId}"))
    .to("paho:some/target/queue");
```

第246章 OSGI PAX LOGGING コンポーネント

Camel バージョン 2.6 以降で利用可能

paxlogging コンポーネントは、OSGi 環境で PaxLogging イベントを受信して処理するために使用できます。

246.1. 依存関係

Maven ユーザーは以下の依存関係を **pom.xml** に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paxlogging</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version} は Camel の実際のバージョン (2.6.0 以降) に置き換える必要があります。

246.2. URI 形式

```
paxlogging:appender[?options]
```

ここで、**appender** は、PaxLogging サービス設定で構成する必要がある pax アペンダーの名前です。

246.3. URI オプション

OSGi PAX Logging コンポーネントは、次に示す 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
bundleContext (consumer)	OSGi BundleContext は Camel によって自動的に注入されます		BundleContext
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OSGi PAX Logging エンドポイントは、URI 構文を使用して設定されます。

```
paxlogging:appender
```

パスおよびクエリーパラメーターを使用します。

246.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
appender	**必須** Appender は、PaxLogging サービス設定で設定する必要がある pax アペンダーの名前です。		String

246.3.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

246.4. メッセージボディー

in メッセージボディーは、受信した PaxLoggingEvent に設定されます。

246.5. 使用例

```
<route>
  <from uri="paxlogging:camel"/>
  <to uri="stream:out"/>
</route>
```

設定:

```
log4j.rootLogger=INFO, out, osgi:VmLogAppender, osgi:camel
```

第247章 PDF コンポーネント

Camel バージョン 2.16 以降で利用可能

PDF: コンポーネントは、PDF ドキュメントからコンテンツを作成、変更、または抽出する機能を提供します。このコンポーネントは、Apache PDFBox を基になるライブラリーとして使用して、PDF ドキュメントを操作します。

PDF コンポーネントを使用するには、Maven ユーザーは次の依存関係を **pom.xml** に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pdf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

247.1. URI 形式

PDF コンポーネントはプロデューサーエンドポイントのみをサポートします。

```
pdf:operation[?options]
```

247.2. オプション

PDF コンポーネントにはオプションがありません。

PDF エンドポイントは、URI 構文を使用して設定されます。

```
pdf:operation
```

パスおよびクエリーパラメーターを使用します。

247.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	***必須*** 操作タイプ		PdfOperation

247.2.2. クエリーパラメーター(9 パラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
font (producer)	フォント	Helvetica	PDFont
fontSize (producer)	フォントサイズ (ピクセル)	14	float
marginBottom (producer)	下部の余白 (ピクセル単位)	20	int
marginLeft (producer)	左側の余白 (ピクセル単位)	20	int
marginRight (producer)	右側の余白 (ピクセル単位)	40	int
marginTop (producer)	上部の余白 (ピクセル単位)	20	int
pageSize (producer)	ページサイズ	A4	PDRectangle
textProcessingFactory (producer)	autoFormatting を使用するテキスト処理: テキストが単語ごとにスライスされ、行に収まる最大量の単語が PDF ドキュメントに書き込まれます。このストラテジーでは、行に収まらないすべての単語が新しい行に移動されます。lineTermination: 行終了書き込みストラテジーのための一連のクラスをビルドします。テキストが行終端記号によってスライスされ、行に収まるかどうかに関係なく書き込まれます。	lineTermination	TextProcessingFactory
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

247.3. ヘッダー

ヘッダー	説明
pdf-document	**append** 操作の **必須** ヘッダーであり、他のすべての操作では無視されます。期待されるタイプは PDDocument です。追加操作に使用される PDF ドキュメントを格納します。
protection-policy	想定されるタイプは https://pdfbox.apache.org/docs/1.8.10/javadocs/org/apache/pdfbox/pdmodel/encryption/ProtectionPolicy.htmlProtectionPolicy です。指定すると、PDF ドキュメントが暗号化されます。

ヘッダー	説明
decryption-material	想定されるタイプは https://pdfbox.apache.org/docs/1.8.10/javadocs/org/apache/pdfbox/pdmodel/encryption/DecryptionMaterial.htmlDecryptionMaterial です。PDF ドキュメントが暗号化されている場合の **必須** ヘッダー。

247.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

- -

第248章 POSTGRESSQL EVENT COMPONENT

Camel バージョン 2.15 以降で利用可能

これは、PostgreSQL 8.3 以降に追加された LISTEN/NOTIFY コマンドに関連する、Producing/Consuming PostgreSQL イベントを可能にする Apache Camel のコンポーネントです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pgevent</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

URI 形式

pgevent コンポーネントは、次の 2 つのスタイルのエンドポイント URI 表記を使用します。

```
pgevent:datasource[?parameters]
pgevent://host:port/database/channel[?parameters]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

248.1. オプション

PostgresSQL Event コンポーネントにはオプションがありません。

PostgresSQL Event エンドポイントは、URI 構文を使用して設定されます。

```
pgevent:host:port/database/channel
```

パスおよびクエリーパラメーターを使用します。

248.1.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
host	ホスト名とポートを使用してデータベースに接続します。	localhost	String
port	ホスト名とポートを使用してデータベースに接続します。	5432	Integer
database	**必須** データベース名		String
channel	**必須** チャンネル名		String

248.1.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
datasource (Common)	ホスト名とポートを使用する代わりに、指定された javax.sql.DataSource を使用して接続します。		DataSource
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
pass (security)	ログイン用パスワード		String
ユーザー (security)	ログイン用のユーザー名	postgres	文字列

248.2. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第249章 PGP DATAFORMAT

Camel バージョン 2.9 以降で利用可能

PGP データフォーマットは、Java 暗号化拡張機能を Camel に統合し、Camel の使い慣れたマーシャルおよびアンマーシャルフォーマットメカニズムを使用して、メッセージのシンプルかつ柔軟な暗号化と復号化を可能にします。マーシャリングは暗号文への暗号化を意味し、アンマーシャリングは元の平文への復号化を意味すると想定しています。このデータ形式は、対称 (共有キー) 暗号化と復号化のみを実装します。

249.1. PGPDATAFORMAT オプション

PGP データ形式は、以下にリストされている 15 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
keyUserid		**String**	暗号化中に使用される PGP キーリング内のキーのユーザー ID。ユーザー ID の一部のみにすることもできます。たとえば、ユーザー ID が Test User の場合、Test User または部分を使用してユーザー ID を指定できます。
signatureKeyUserid		**String**	署名 (暗号化中) または署名検証 (復号化中) に使用される PGP キーリング内のキーのユーザー ID。署名検証プロセス中、指定されたユーザー ID は、検証に使用できる公開キーリングからの公開鍵を制限します。署名の検証にユーザー ID が指定されていない場合は、公開キーリング内の任意の公開鍵を検証に使用できます。ユーザー ID の一部のみにすることもできます。たとえば、ユーザー ID が Test User の場合、Test User の部分を使用したり、ユーザー ID をアドレス指定したりできます。
password		**String**	秘密鍵を開くときに使用されるパスワード (暗号化には使用されません)。
signaturePassword		**String**	署名に使用する秘密鍵を開くとき (暗号化時) に使用するパスワード。
keyFileName		**String**	キーリングのファイル名。クラスパスリソースとしてアクセスできる必要があります (ただし、file: 接頭辞を使用してファイルシステム内のロケーションを指定できます)。
signatureKeyFileName		**String**	署名 (暗号化中) または署名検証 (復号化中) に使用するキーリングのファイル名。クラスパスリソースとしてアクセスできる必要があります (ただし、file: 接頭辞を使用してファイルシステム内のロケーションを指定できます)。
signatureKeyRing		**String**	バイト配列としての署名/検証に使用されるキーリング。signatureKeyFileName と signatureKeyRing を同時に設定することはできません。

名前	デフォルト	Java タイプ	説明
armored	**false**	**Boolean**	このオプションにより、PGP は暗号化されたテキストを base64 でエンコードし、コピー/貼り付けなどに使用できるようにします。
integrity	**true**	**Boolean**	整合性チェック/署名を暗号化ファイルに追加します。デフォルト値は true です。
provider		**String**	Java Cryptography Extension (JCE) プロバイダー。デフォルトは Bouncy Castle (BC) です。あるいは、IAIK JCE プロバイダーなどを使用することもできます。この場合、プロバイダーは事前に登録する必要があり、Bouncy Castle プロバイダーは事前に登録しないでください。Sun JCE プロバイダーが機能しません。
algorithm		**Integer**	対称鍵暗号化アルゴリズム。可能な値は org.bouncycastle.bcpg.SymmetricKeyAlgorithmTags で定義されています。たとえば、2 (= TRIPLE DES)、3 (= CAST5)、4 (= BLOWFISH)、6 (= DES)、7 (= AES_128) です。暗号化にのみ関連します。
compressionAlgorithm		**Integer**	圧縮アルゴリズム;可能な値は org.bouncycastle.bcpg.CompressionAlgorithmTags で定義されています。たとえば、0 (= 非圧縮)、1 (= ZIP)、2 (= ZLIB)、3 (= BZIP2) です。暗号化にのみ関連します。
hashAlgorithm		**Integer**	署名ハッシュアルゴリズム。可能な値は org.bouncycastle.bcpg.HashAlgorithmTags で定義されています。たとえば、2 (= SHA1)、8 (= SHA256)、9 (= SHA384)、10 (= SHA512)、11 (=SHA224) です。署名にのみ関連します。
signatureVerificationOption		**String**	アンマーシャリング中に署名を検証するための動作を制御します。設定可能な値は 4 つあります。オプション: PGP メッセージには署名が含まれる場合と含まれない場合があります。署名が含まれている場合は、署名の検証が実行されます。必須: PGP メッセージには少なくとも 1 つの署名が含まれている必要があります。そうでない場合は、例外 (PGPException) が出力されます。署名検証が実行されます。ignore: PGP メッセージに含まれる署名は無視されます。署名の検証は実行されません。no_signature_allowed: PGP メッセージに署名を含めることはできません。それ以外の場合は、例外 (PGPException) が出力されます。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

249.2. PGPDATAFORMAT MESSAGE HEADERS

以下のヘッダーをメッセージに動的に適用することで、PGPDataFormat オプションをオーバーライドできます。

名前	タイプ	説明
CamelPGPDataFormatKeyFileName	**String**	**Camel 2.11.0 以降**; キーリングのファイル名; PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatEncryptionKeyRing	**byte[]**	**Camel 2.12.1 以降** ;暗号化キーリング; PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatKeyUserid	**String**	**Camel 2.11.0 以降**; PGP キーリング内のキーのユーザー ID。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatKeyUserids	**List<String>**	**camel 2.12.2 以降**: PGP キーリング内のキーのユーザー ID。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatKeyPassword	**String**	**Camel 2.11.0 以降**; 秘密鍵を開くときに使用するパスワード。PGPDataFormat の既存の設定を直接上書きします。

名前	タイプ	説明
CamelPGPDataFormatSignatureKeyFileName	**String**	**Camel 2.11.0 以降**; 署名キーリングのファイル名。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatSignatureKeyRing	**byte[]**	**Camel 2.12.1 以降**; 署名キーリング; PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatSignatureKeyUserid	**String**	**Camel 2.11.0 以降**; PGP キーリング内の署名キーのユーザー ID。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatSignatureKeyUserids	**List<String>**	**Camel 2.12.3 以降**; PGP キーリング内の署名キーのユーザー ID。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatSignatureKeyPassword	**String**	**Camel 2.11.0 以降**; 署名の秘密鍵を開くときに使用するパスワード。PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatEncryptionAlgorithm	**int**	**Camel 2.12.2 以降**; 対称鍵暗号化アルゴリズム; PGPDataFormat の既存の設定を直接上書きします。

名前	タイプ	説明
CamelPGPDataFormatSignatureHashAlgorithm	**int**	**Camel 2.12.2 以降**; 署名ハッシュアルゴリズム; PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatCompressionAlgorithm	**int**	**Camel 2.12.2 以降**; 圧縮アルゴリズム; PGPDataFormat の既存の設定を直接上書きします。
CamelPGPDataFormatNumberOfEncryptionKeys	**Integer**	*Camel 2.12.3 **以降**; *対称鍵の暗号化に使用される公開鍵の数。暗号化プロセス中に PGPDataFormat によって設定されます。
CamelPGPDataFormatNumberOfSigningKeys	**Integer**	*Camel 2.12.3 **以降**; *署名プロセス中に PGPDataFormat によって設定された、署名の作成に使用される秘密鍵の数

249.3. PGPDATAFORMAT による暗号化

次のサンプルでは、Bouncy Castle Java ライブラリー を使用してファイルを暗号化/復号化するために一般的な PGP 形式を使用しています。

次のサンプルでは、署名 + 暗号化を実行してから、署名検証 + 復号化を実行します。署名と暗号化の両方に同じキーリングを使用しますが、明らかに異なるキーを使用できます。

または Spring を使用して:

249.3.1. 前の例を使用するには、次のものが必要です

- データの暗号化に使用される公開鍵を含む公開キーリングファイル

- データの復号化に使用される鍵を含む秘密キーリングファイル
- キーリングのパスワード

249.3.2. キーリングの管理

キーリングを管理するには、コマンドラインツールを使用します。これが、キーを管理する最も簡単な方法だと思います。そうしたい場合は、http://www.bouncycastle.org/java.html から入手できる Java ライブラリーもあります。

Linux にコマンドラインユーティリティーをインストールする

```
apt-get install gnupg
```

安全なパスワードを入力して、キーリングを作成します

```
gpg --gen-key
```

ファイルを暗号化できるように、他の人の公開鍵をインポートする必要がある場合。

```
gpg --import <filename.key
```

次のファイルが存在するはずであり、例を実行するために使用できます

```
ls -l ~/.gnupg/pubring.gpg ~/.gnupg/secring.gpg
```

[Crypto-PGPDecrypting/VerifyingofMessagesEncrypted/SignedbyDifferentPrivate/PublicKeys]] です。異なる#で暗号化/署名されたメッセージの PGP 復号化/検証について秘密/公開鍵

Since **Camel 2.12.2**.

PGP Data Formater は、異なる公開鍵で暗号化されたメッセージや、異なる秘密鍵で署名されたメッセージを復号化/検証できます。対応する秘密鍵を秘密キーリングに、対応する公開鍵を公開キーリングに、パスフレーズをパスフレーズアクセサーに指定するだけです。

```
Map<String, String> userId2Passphrase = new HashMap<String, String>(2);
// add passphrases of several private keys whose corresponding public keys have been used to
encrypt the messages
userId2Passphrase.put("UserIdOfKey1","passphrase1"); // you must specify the exact User ID!
userId2Passphrase.put("UserIdOfKey2","passphrase2");
PGPPassphraseAccessor passphraseAccessor = new
PGPPassphraseAccessorDefault(userId2Passphrase);

PGPDataFormat pgpVerifyAndDecrypt = new PGPDataFormat();
pgpVerifyAndDecrypt.setPassphraseAccessor(passphraseAccessor);
// the method getSecKeyRing() provides the secret keyring as byte array containing the private keys
pgpVerifyAndDecrypt.setEncryptionKeyRing(getSecKeyRing()); // alternatively you can use
setKeyFileName(keyfileName)
// the method getPublicKeyRing() provides the public keyring as byte array containing the public keys
pgpVerifyAndDecrypt.setSignatureKeyRing((getPublicKeyRing()); // alternatively you can use
setSignatureKeyFileName(signatgureKeyfileName)
// it is not necessary to specify the encryption or signer  User Id

from("direct:start")
```

```
        ...
        .unmarshal(pgpVerifyAndDecrypt) // can decrypt/verify messages encrypted/signed by different
private/public keys
        ...
```

- この機能は、鍵交換をサポートするのに特に役立ちます。復号化のために秘密鍵を交換したい場合は、古いまたは新しい対応する公開鍵で暗号化されたメッセージを一定期間受け入れることができます。または、送信者が署名者の秘密鍵を交換したい場合は、古いまたは新しい署名者の鍵を一定期間受け入れることができます。

- 技術的背景: PGP で暗号化されたデータには、データの暗号化に使用された公開鍵のキー ID が含まれています。このキー ID を使用して、秘密キーリング内の秘密鍵を見つけ、データを復号化できます。同じメカニズムを使用して、署名を検証するための公開鍵を見つけます。したがって、アンマーシャリングのためにユーザー ID を指定する必要はなくなりました。

249.4. PGP 署名検証中の署名者 ID の制限

Since **Camel 2.12.3.**

署名を検証する場合、署名の正確性を検証するだけでなく、署名が特定の ID または特定の ID のセットに由来することも確認する必要があります。したがって、署名の検証に使用できる公開キーリングからの公開鍵の数を制限することができます。

署名ユーザー ID

```
// specify the User IDs of the expected signer identities
 List<String> expectedSigUserIds = new ArrayList<String>();
 expectedSigUserIds.add("Trusted company1");
 expectedSigUserIds.add("Trusted company2");

 PGPDataFormat pgpVerifyWithSpecificKeysAndDecrypt = new PGPDataFormat();
 pgpVerifyWithSpecificKeysAndDecrypt.setPassword("my password"); // for decrypting with private
key
 pgpVerifyWithSpecificKeysAndDecrypt.setKeyFileName(keyfileName);
 pgpVerifyWithSpecificKeysAndDecrypt.setSignatureKeyFileName(signatgureKeyfileName);
 pgpVerifyWithSpecificKeysAndDecrypt.setSignatureKeyUserids(expectedSigUserIds); // if you have
only one signer identity then you can also use setSignatureKeyUserid("expected Signer")

from("direct:start")
        ...
        .unmarshal(pgpVerifyWithSpecificKeysAndDecrypt)
        ...
```

- PGP コンテンツに複数の署名がある場合、1 つの署名が検証されるとすぐに検証が成功します。

- 検証のために署名者 ID を制限したくない場合は、署名鍵のユーザー ID を指定しないでください。この場合、公開キーリング内のすべての公開鍵が考慮されます。

249.5. 1 つの PGP データ形式での複数の署名

Since **Camel 2.12.3.**

PGP 仕様では、1 つの PGP データ形式に異なる鍵からの複数の署名を含めることができます。Camel 2.13.3 以降、秘密キーリング内の複数の秘密鍵に関連する署名ユーザー ID を指定することで、そのような種類の PGP コンテンツを作成できます。

複数の署名

```
PGPDataFormat pgpSignAndEncryptSeveralSignerKeys = new PGPDataFormat();
pgpSignAndEncryptSeveralSignerKeys.setKeyUserid(keyUserid); // for encrypting, you can also use setKeyUserids if you want to encrypt with several keys
pgpSignAndEncryptSeveralSignerKeys.setKeyFileName(keyfileName);
pgpSignAndEncryptSeveralSignerKeys.setSignatureKeyFileName(signatgureKeyfileName);
pgpSignAndEncryptSeveralSignerKeys.setSignaturePassword("sdude"); // here we assume that all private keys have the same password, if this is not the case then you can use setPassphraseAccessor

List<String> signerUserIds = new ArrayList<String>();
signerUserIds.add("company old key");
signerUserIds.add("company new key");
pgpSignAndEncryptSeveralSignerKeys.setSignatureKeyUserids(signerUserIds);

from("direct:start")
        ...
       .marshal(pgpSignAndEncryptSeveralSignerKeys)
       ...
```

249.6. PGP データ形式マーシャラーでのサブキーとキーフラグのサポート

*Camel 2.12.3 以降。
* OpenPGP V4 キーは、 プライマリーキーとサブキーを持つことができます。キーの使用法は、いわゆる キーフラグ によって示されます。たとえば、プライマリキーと 2 つのサブキーを持つことができます。プライマリキーは他のキーの認証にのみ使用し (キーフラグ 0x01)、最初のサブキーは署名にのみ使用し (キーフラグ0x02)、2番目のサブキーは暗号化 (キーフラグ 0x04 または 0x08) にのみ使用できるものとします。PGP データフォーマットマーシャラーは、署名と暗号化のための正しいキーを決定するために、プライマリーキーとサブキーのこれらのキーフラグを考慮に入れます。これは、プライマリーキーとそのサブキーのユーザー ID が同じであるため必要です。

249.7. カスタムキーアクセサーのサポート

*Camel 2.13.0 以降。
*暗号化/署名用のカスタムキーアクセサーを実装できます。上記の PGPDataFormat クラスは、署名/暗号化または検証/復号化に使用する必要がある特定の定義済みの方法でキーを選択します。鍵の選択方法に特別な要件がある場合は、代わりに PGPKeyAccessDataFormat クラスを使用し、インターフェイス PGPPublicKeyAccessor および PGPSecretKeyAccessor を Bean として実装する必要があります。キーをキャッシュする DefaultPGPPublicKeyAccessor と DefaultPGPSecretKeyAccessor のデフォルトの実装があるため、プロセッサーが呼び出されるたびにキーリングが解析されるわけではありません。

PGPKeyAccessDataFormat には、password、keyFileName、encryptionKeyRing、signaturePassword、signatureKeyFileName、および signatureKeyRing を除いて、PGPDataFormat と同じオプションがあります。

249.8. 依存関係

camel ルートで PGP データ形式を使用するには、次の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

249.9. 関連項目

- データ形式
- 暗号 (デジタル署名)
- http://www.bouncycastle.org/java.html

第250章 PROPERTIES コンポーネント

Camel バージョン 2.3 以降で利用可能

250.1. URI 形式

```
properties:key[?options]
```

key は、検索するプロパティーのキーです

250.2. オプション

Properties コンポーネントは、以下に示す 17 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
locations (common)	プロパティーを読み込むロケーションのリスト。このオプションは、デフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		List
location (common)	プロパティーを読み込むロケーションのリスト。コンマを使用して複数のロケーションを区切ることができます。このオプションは、デフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		String
encoding (common)	ファイルシステムまたはクラスパスからプロパティーファイルをロードするときに使用するエンコーディング。エンコーディングが設定されていない場合、リンク java.util.Propertiesload (java.io.InputStream) で説明されているように、プロパティーファイルは ISO-8859-1 エンコーディング (latin-1) を使用してロードされます。		文字列
propertiesResolver (common)	カスタム PropertiesResolver を使用する場合。		PropertiesResolver
propertiesParser (common)	カスタム PropertiesParser を使用する場合。		PropertiesParser
cache (common)	読み込まれたプロパティーをキャッシュするかどうか。デフォルト値は true です。	true	boolean
propertyPrefix (advanced)	解決前にプロパティー名の前に付けられるオプションの接頭辞。		String
propertySuffix (advanced)	解決前にプロパティー名に追加されるオプションの接尾辞。		String

名前	説明	デフォルト	タイプ
fallbackToUnaugmented Property (advanced)	true の場合、最初に propertyPrefix と propertySuffix で拡張されたプロパティー名の解決を試みてから、指定されたプレーンなプロパティー名にフォールバックします。false の場合、拡張されたプロパティー名のみが検索されます。	true	boolean
defaultFallbackEnabled (common)	false の場合、コンポーネントはコロン区切り記号を調べてキーのデフォルトを見つけようとしません。	true	boolean
ignoreMissingLocation (common)	プロパティーファイルが見つからないなど、ロケーションが見つからない場合に黙って無視するかどうか。	false	boolean
prefixToken (advanced)	置換するプロパティーを識別するために使用される接頭辞 トークンの値を設定します。null の値を設定すると、デフォルトのトークンが復元されます (リンク DEFAULT_PREFIX_TOKEN)。	{{	String
suffixToken (advanced)	置換するプロパティーを識別するために使用される接尾辞 トークンの値を設定します。null の値を設定すると、デフォルトのトークンが復元されます (リンク DEFAULT_SUFFIX_TOKEN)。	}}	String
initialProperties (advanced)	ロケーションが解決される前に使用される初期プロパティーを設定します。		Properties
overrideProperties (advanced)	プロパティーが存在する場合、優先されて最初に使用されるオーバーライドプロパティーの特別なリストを設定します。		Properties
systemPropertiesMode (common)	システムプロパティーモードを設定します。	2	int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Properties エンドポイントは、URI 構文を使用して設定されます。

```
properties:key
```

パスおよびクエリーパラメーターを使用します。

250.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
key	**必須** プレースホルダーとして使用するプロパティーキー		String

250.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
ignoreMissingLocation (common)	プロパティーファイルが見つからないなど、ロケーションが見つからない場合に黙って無視するかどうか。	false	boolean
locations (common)	プロパティーを読み込むロケーションのリスト。コンマを使用して複数のロケーションを区切ることができます。このオプションは、デフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

ヒント

Java コードからのプロパティーの解決
CamelContext で **resolvePropertyPlaceholders** メソッドを使用して、任意の Java コードからプロパティーを解決できます。

250.3. PROPERTYPLACEHOLDER の使用

Camel 2.3 の時点で利用可能

Camel は **camel-core** で新しい **PropertiesComponent** を提供するようになりました。これにより、Camel エンドポイント URI を定義するときにプロパティープレースホルダーを使用できます。

これは、Spring の **<property-placeholder>** タグを使用する場合と同じように機能します。ただし、Spring には、サードパーティーのフレームワークが Spring プロパティーのプレースホルダーを最大限に活用できないという制限があります。How do I use Spring Property Placeholder with Camel XML で詳細を参照してください。

ヒント

Spring と Camel のプロパティープレースホルダーのブリッジ
Camel 2.10 以降では、Spring プロパティープレースホルダーを Camel とブリッジできます。詳細については、以下を参照してください。

プロパティープレースホルダーは通常、次の場合に使用されます。

- ルックアップまたはエンドポイントの作成
- レジストリー内の Bean のルックアップ
- Spring XML で追加サポート (以下の例を参照)
- Camel Properties コンポーネントで Blueprint PropertyPlaceholder を使用する
- **@PropertyInject** を使用して POJO にプロパティーを注入する
- **Camel 2.14.1** プロパティーが存在しない場合のデフォルト値の使用
- **Camel 2.14.1** OS 環境変数、JVM システムプロパティー、またはサービスイディオムからプロパティー値を検索するために、すぐに使用できる関数が含まれている
- **Camel 2.14.1** プロパティーコンポーネントにプラグインできるカスタム関数の使用

250.4. 構文

Camel のプロパティープレースホルダーを使用する構文は、たとえば **{{file.uri}}** のように **{{key}}** を使用することです。ここで、**file.uri** はプロパティーキーです。

エンドポイント URI の一部でプロパティープレースホルダーを使用できます。たとえば、URI のパラメーターにプレースホルダーを使用できます。

Camel 2.14.1 以降では、キーを持つプロパティーが存在しない場合に使用するデフォルト値を指定できます。たとえば、デフォルト値がコロンの後のテキストである **file.url:/some/path** (例:/some/path)。

注記

プロパティーキーにコロンを使用しないでください。コロンは、**Camel 2.14.1** 以降でサポートされているデフォルト値を提供するときに区切りトークンとして使用されます。

250.5. PROPERTYRESOLVER

Camel は、サードパーティーが独自のリゾルバーをルックアッププロパティーに提供できるようにするプラグ可能なメカニズムを提供します。Camel は、ファイルシステム、クラスパス、またはレジストリーからプロパティーをロードできるデフォルトの実装 **org.apache.camel.component.properties.DefaultPropertiesResolver** を提供します。ロケーションの前に次のいずれかの接頭辞を付けることができます。

- **ref: Camel 2.4:** レジストリーを検索する
- **file:** from ファイルシステムをロードする
- **classpath:** クラスパスからロードする (接頭辞が指定されていない場合、これはデフォルトでもある)
- **blueprint: Camel 2.7:** 特定の OSGi blueprint プレースホルダーサービスを使用する

250.6. ロケーションの定義

PropertiesResolver は、プロパティーを解決するロケーションを把握する必要があります。1 から複数のロケーションを定義できます。単一の String プロパティーでロケーションを定義する場合、次のように複数のロケーションをコンマで区切ることができます。

```
pc.setLocation("com/mycompany/myprop.properties,com/mycompany/other.properties");
```

Camel 2.19.0 以降で利用可能

オプションの 属性を設定することで、欠落している場合に破棄できるロケーションを設定できます。デフォルトでは false です。

```
pc.setLocations(
    "com/mycompany/override.properties;optional=true"
    "com/mycompany/defaults.properties");
```

250.7. ロケーションでのシステム変数と環境変数の使用

Camel 2.7 以降で利用可能

このロケーションでは、JVM システムプロパティーと OS 環境変数のプレースホルダーの使用がサポートされるようになりました。

以下に例を示します。

```
location=file:${karaf.home}/etc/foo.properties
```

上記のロケーションでは、キー **karaf.home** を持つ JVM システムプロパティーを使用するファイルスキームを使用してロケーションを定義しました。

代わりに OS 環境変数を使用するには、env を接頭辞として付ける必要があります。

```
location=file:${env:APP_HOME}/etc/foo.properties
```

APP_HOME は OS 環境です。

次のように、複数のプレースホルダーを同じロケーションに配置できます。

```
location=file:${env:APP_HOME}/etc/${prop.name}.properties
```

#=== システム変数と環境変数を使用してプロパティーの接頭辞と接尾辞を設定する

Camel 2.12.5、2.13.3、2.14.0 以降で利用可能

propertyPrefix、**propertySuffix** 設定プロパティーは、JVM システムプロパティーおよび OS 環境変数のプレースホルダーの使用をサポートします。

たとえば、**PropertiesComponent** が次のプロパティーファイルで設定されている場合:

```
dev.endpoint = result1
test.endpoint = result2
```

次に、次のルート定義を使用します。

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.setPropertyPrefix("${stage}.");
// ...
context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").to("properties:mock:{{endpoint}}");
    }
});
```

システムプロパティー **stage** を **dev** (メッセージは **mock:result1** にルーティングされます) または **test** (メッセージは **mock:result2** にルーティングされます) に変更することで、ターゲットエンドポイントを変更できます。

250.8. JAVA DSL での設定

次のような名前の **properties** で **PropertiesComponent** を作成して登録する必要があります。

```
PropertiesComponent pc = new PropertiesComponent();
pc.setLocation("classpath:com/mycompany/myprop.properties");
context.addComponent("properties", pc);
```

250.9. SPRING XML での設定

Spring XML には、設定する 2 つのバリエーションがあります。Spring Bean を、Java DSL で行われる方法に似た **PropertiesComponent** として定義できます。または、**<propertyPlaceholder>** タグを使用できます。

```
<bean id="properties" class="org.apache.camel.component.properties.PropertiesComponent">
    <property name="location" value="classpath:com/mycompany/myprop.properties"/>
</bean>
```

<propertyPlaceholder> タグを使用すると、設定が次のように少し新しくなります。

```
<camelContext ...>
  <propertyPlaceholder id="properties" location="com/mycompany/myprop.properties"/>
</camelContext>
```

location タグを使用してプロパティーのロケーションを設定することは問題なく機能しますが、考慮すべきリソースが多数ある場合があり、**Camel 2.19.0** 以降では、専用の propertiesLocation を使用してプロパティーの場所を設定できます。

```
<camelContext ...>
  <propertyPlaceholder id="myPropertyPlaceholder">
    <propertiesLocation
      resolver = "classpath"
      path     = "com/my/company/something/my-properties-1.properties"
      optional = "false"/>
    <propertiesLocation
      resolver = "classpath"
      path     = "com/my/company/something/my-properties-2.properties"
      optional = "false"/>
    <propertiesLocation
      resolver = "file"
      path     = "${karaf.home}/etc/my-override.properties"
      optional = "true"/>
  </propertyPlaceholder>
</camelContext>
```

ヒント

XML 内のキャッシュオプションの指定
Camel 2.10 以降では、Spring 内と Blueprint XML 内の両方で cache オプションの値を指定できます。

250.10. レジストリーからのプロパティーの使用

Camel 2.4 以降で利用可能
たとえば、OSGi では、プロパティーを **java.util.Properties** オブジェクトとして返すサービスを公開したい場合があります。

次に、Properties コンポーネントを次のようにセットアップできます。

```
<propertyPlaceholder id="properties" location="ref:myProperties"/>
```

myProperties は、OSGi レジストリーでのルックアップに使用する ID です。 **ref:** 接頭辞を使用して、Camel にレジストリーのプロパティーを検索するように指示していることに注意してください。

250.11. プロパティーコンポーネントを使用した例

エンドポイント URI でプロパティープレースホルダーを使用する場合、**properties:** コンポーネントを使用するか、URI で直接プレースホルダーを定義できます。前者から始めて、両方のケースの例を示します。

```
// properties
cool.end=mock:result

// route
from("direct:start").to("properties:{{cool.end}}");
```

エンドポイント URI の一部としてプレースホルダーを使用することもできます。

```
// properties
cool.foo=result

// route
from("direct:start").to("properties:mock:{{cool.foo}}");
```

上記の例では、to エンドポイントは **mock:result** に解決されます。

次のように、互いに参照するプロパティーを持つこともできます。

```
// properties
cool.foo=result
cool.concat=mock:{{cool.foo}}

// route
from("direct:start").to("properties:mock:{{cool.concat}}");
```

cool.concat が別のプロパティーを参照する方法に注目してください。

properties: コンポーネントでは、**locations** オプションを使用して、指定された uri 内の場所をオーバーライドして提供することもできます。

```
from("direct:start").to("properties:bar.end?locations=com/mycompany/bar.properties");
```

250.12. 例

properties: を使用せずに、エンドポイント uris でプロパティープレースホルダーを直接使用することもできます。

```
// properties
cool.foo=result

// route
from("direct:start").to("mock:{{cool.foo}}");
```

そして、それらを任意の場所で複数使用することができます:

```
// properties
cool.start=direct:start
cool.showid=true
cool.result=result

// route
from("{{cool.start}}")
    .to("log:{{cool.start}}?showBodyType=false&showExchangeId={{cool.showid}}")
    .to("mock:{{cool.result}}");
```

たとえば、ProducerTemplate を使用する場合は、プロパティープレースホルダーを使用することもできます。

```
template.sendBody("{{cool.start}}", "Hello World");
```

250.13. SIMPLE 言語の例

Simple 言語は、プロパティープレースホルダーの使用もサポートするようになりました。たとえば、以下のルートで:

```
// properties
cheese.quote=Camel rocks

// route
from("direct:start")
    .transform().simple("Hi ${body} do you think ${properties:cheese.quote}?");
```

たとえば、Simple 言語でロケーションを指定することもできます。

```
// bar.properties
bar.quote=Beer tastes good

// route
from("direct:start")
    .transform().simple("Hi ${body}. ${properties:com/mycompany/bar.properties:bar.quote}.");
```

250.14. SPRING XML でサポートされる追加のプロパティープレースホルダー

プロパティープレースホルダーは、**<package>, <packageScan>, <contextScan>, <jmxAgent>, <endpoint>, <routeBuilder>, <proxy>** などの Camel Spring XML タグの多くでもサポートされています。

以下の例では、**<jmxAgent>** タグにプロパティープレースホルダーがあります。

次に示すように、**trace** などの **<camelContext>** タグのさまざまな属性でプロパティープレースホルダーを定義することもできます。

250.15. JVM システムプロパティーを使用したプロパティー設定のオーバーライド

Camel 2.5 以降で利用可能

変更を反映するためにアプリケーションを再起動しなくても、JVM システムプロパティーを使用して実行時にプロパティー値をオーバーライドできます。これは、新しい値に置き換えるプロパティーと同じ名前の JVM システムプロパティーを作成することにより、コマンドラインから実行することもできます。この例を以下に示します

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.setCache(false);

System.setProperty("cool.end", "mock:override");
System.setProperty("cool.result", "override");

context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").to("properties:cool.end");
        from("direct:foo").to("properties:mock:{{cool.result}}");
```

```
    }
});
context.start();

getMockEndpoint("mock:override").expectedMessageCount(2);

template.sendBody("direct:start", "Hello World");
template.sendBody("direct:foo", "Hello Foo");

System.clearProperty("cool.end");
System.clearProperty("cool.result");

assertMockEndpointsSatisfied();
```

250.16. XML DSL のあらゆる種類の属性にプロパティープレースホルダーを使用する

Camel 2.7 以降で利用可能

注記

OSGi Blueprint を使用する場合、これは 2.11.1 または 2.10.5 以降でのみ機能します。

以前は、プレースホルダーをサポートする XML DSL の **xs:string** タイプ属性のみでした。たとえば、多くの場合、タイムアウト属性は **xs:int** タイプであるため、文字列値をプレースホルダーキーとして設定することはできません。これは、特別なプレースホルダー名前空間を使用して Camel 2.7 以降で可能になりました。

以下の例では、名前空間 **http://camel.apache.org/schema/placeholder** に **prop** 接頭辞を使用しています。これにより、XML DSL の属性で **prop** 接頭辞を使用できます。マルチキャストでこれを使用して、オプション **stopOnException** がキー stop を持つプレースホルダーの値であることを示す方法に注意してください。

プロパティーファイルには、次のように定義された値があります。

```
stop=true
```

250.17. CAMEL ルートで BLUEPRINT プロパティープレースホルダーを使用する

Camel 2.7 以降で利用可能

Camel は、プロパティープレースホルダーサービスも提供する Blueprint をサポートしています。Camel は設定より規約をサポートしているため、以下に示すように XML ファイルで OSGi Blueprint プロパティープレースホルダーを定義するだけです。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xsi:schemaLocation="
           http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
```

```
    <!-- OSGI blueprint property placeholder -->
    <cm:property-placeholder id="myblueprint.placeholder" persistent-id="camel.blueprint">
        <!-- list some properties as needed -->
        <cm:default-properties>
            <cm:property name="result" value="mock:result"/>
        </cm:default-properties>
    </cm:property-placeholder>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <!-- in the route we can use {{ }} placeholders which will lookup in blueprint
             as Camel will auto detect the OSGi blueprint property placeholder and use it -->
        <route>
            <from uri="direct:start"/>
            <to uri="mock:foo"/>
            <to uri="{{result}}"/>
        </route>
    </camelContext>
</blueprint>
```

250.17.1. Camel ルートでの OSGi Blueprint プロパティープレースホルダーの使用

デフォルトでは、Camel は OSGi Blueprint プロパティープレースホルダーサービスを検出して使用します。**<camelContext>** 定義で属性 **useBlueprintPropertyResolver** を false に設定することで、これを無効にすることができます。

250.17.2. プレースホルダー構文について

Camel ルートのプレースホルダー **{{** および **}}** に Camel 構文を使用する方法に注目してください。これは、OSGi ブループリントから値を検索します。

プレースホルダーの blueprint 構文は **${ }** です。**<camelContext>** の外では、**${ }** 構文を使用する必要があります。**<camelContext>** の内部では、**{{** および **}}** 構文を使用する必要があります。

OSGi blueprint を使用すると構文を設定できるため、必要に応じて実際にそれらを調整できます。

特定の OSGi blueprint プロパティープレースホルダーをその ID で明示的に参照することもできます。そのためには、次の例に示すように、Camel の **<propertyPlaceholder>** を使用する必要があります。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xsi:schemaLocation="
           http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <!-- OSGI blueprint property placeholder -->
    <cm:property-placeholder id="myblueprint.placeholder" persistent-id="camel.blueprint">
        <!-- list some properties as needed -->
        <cm:default-properties>
            <cm:property name="prefix.result" value="mock:result"/>
        </cm:default-properties>
    </cm:property-placeholder>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
```

```
        <!-- using Camel properties component and refer to the blueprint property placeholder by its id --
>
        <propertyPlaceholder id="properties" location="blueprint:myblueprint.placeholder"
                             prefixToken="[[" suffixToken="]]"
                             propertyPrefix="prefix."/>

        <!-- in the route we can use {{ }} placeholders which will lookup in blueprint -->
        <route>
          <from uri="direct:start"/>
          <to uri="mock:foo"/>
          <to uri="[[result]]"/>
        </route>
      </camelContext>
    </blueprint>
```

250.18. CAMEL の OSGI BLUEPRINT プレースホルダーを明示的に参照する

blueprint スキームを使用して OSGi blueprint プレースホルダーを ID で参照する方法に注目してください。これにより、組み合わせて一致させることができます。たとえば、その場所に追加のスキームを含めることもできます。たとえば、クラスパスからファイルをロードするには、次のようにします。

```
location="blueprint:myblueprint.placeholder,classpath:myproperties.properties"
```

各場所はコンマで区切ります。

250.19. CAMELCONTEXT の外部で BLUEPRINT プロパティープレースホルダーをオーバーライドする

Camel 2.10.4 以降で利用可能

Blueprint XML ファイルで Blueprint プロパティープレースホルダーを使用する場合、以下に示すように、プロパティーを XML ファイルで直接宣言できます。

プロパティーの 1 つを参照する **<bean>** があることに注意してください。Camel ルートでは、**{{** と **}}** 表記を使用して他のルートを参照します。

これらの Blueprint プロパティーを単体テストからオーバーライドする場合は、以下のように実行できます。

これを行うには、**useOverridePropertiesWithConfigAdmin** メソッドをオーバーライドして実装します。次に、オーバーライドするプロパティーを指定された props パラメーターに配置できます。また、戻り値は、blueprint XML ファイルで定義した **<cm:property-placeholder>** タグの **persistence-id** でなければ **ばなりません**。

250.20. BLUEPRINT プロパティープレースホルダーに .CFG または .PROPERTIES ファイルを使用する

Camel 2.10.4 以降で利用可能

Blueprint XML ファイルでブループリントプロパティープレースホルダーを使用する場合、**.properties** または **.cfg** ファイルでプロパティーを宣言できます。Apache ServieMix/Karaf を使用する場合、このコンテナーには、**etc/pid.cfg** という名前の etc ディレクトリー内のファイルからプロパティーをロード

するという規則があります。ここで、**pid** は **persistence-id** です。

たとえば、blueprint XML ファイルには **persistence-id="stuff"** があります。これは、設定ファイルを **etc/stuff.cfg** としてロードすることを意味します。

この blueprint XML ファイルを単体テストする場合は、次のように **loadConfigAdminConfigurationFile** をオーバーライドして、ロードするファイルを Camel に指示できます。

このメソッドは、2 つの値を持つ **String** を返す必要があることに注意してください。最初の値は、ロードする設定ファイルのパスです。2 番目の値は、**<cm:property-placeholder>** タグの **persistence-id** です。

stuff.cfg ファイルは、次のようなプロパティープレースホルダーを含む単純なプロパティーファイルです。

```
== this is a comment
greeting=Bye
```

250.21. .CFG ファイルを使用し、BLUEPRINT プロパティープレースホルダーのプロパティーをオーバーライドする

両方を行うこともできます。これが完全な例です。まず、Blueprint XML ファイルがあります。

単体テストクラスでは、次のようにします。

etc/stuff.cfg 設定ファイルには以下を含みます

```
greeting=Bye
echo=Yay
destination=mock:result
```

250.22. SPRING と CAMEL のプロパティープレースホルダーのブリッジ

Camel 2.10 以降で利用可能

Spring Framework では、Apache Camel などのサードパーティーフレームワークが Spring プロパティープレースホルダーメカニズムにシームレスにフックすることはできません。ただし、Spring **org.springframework.beans.factory.config.PropertyPlaceholderConfigurer** タイプであるタイプ **org.apache.camel.spring.spi.BridgePropertyPlaceholderConfigurer** で Spring Bean を宣言することにより、Spring と Camel を簡単にブリッジできます。

Spring と Camel をブリッジするには、以下に示すように単一の Bean を定義する必要があります。

Spring と Camel のプロパティープレースホルダーのブリッジ

spring <context:property-placeholder> 名前空間を同時に使用し **ないでください**。これは不可能です。

この Bean を宣言した後、以下に示すように、<camelContext> タグ内で Spring スタイルと Camel スタイルの両方を使用してプロパティープレースホルダーを定義できます。

ブリッジプロパティープレースホルダーの使用

${ } 表記を使用して、hello Bean が純粋な Spring プロパティープレースホルダーをどのように使用しているかに注意してください。Camel ルートでは、**{{** と **}}** を使用して Camel プレースホルダー表記を使用します。

250.23. CAMELS SIMPLE 言語による SPRING プロパティーのプレースホルダーの競合

Spring ブリッジプレースホルダーを使用すると、Spring の **${ }** 構文が Camel の Simple と衝突することに注意してください。以下に例を示します。

```
<setHeader headerName="Exchange.FILE_NAME">
  <simple>{{file.rootdir}}/${in.header.CamelFileName}</simple>
</setHeader>
```

Spring プロパティーのプレースホルダーと衝突するため、**$simple{ }** を使用して、Camel で Simple 言語を使用していることを示す必要があります。

```
<setHeader headerName="Exchange.FILE_NAME">
  <simple>{{file.rootdir}}/$simple{in.header.CamelFileName}</simple>
</setHeader>
```

別の方法は、**ignoreUnresolvablePlaceholders** オプションを **true** にして **PropertyPlaceholderConfigurer** を設定することです。

250.24. CAMEL テストキットのプロパティーをオーバーライドする

Camel 2.10 以降で利用可能

Camel でテストし、Properties コンポーネントを使用する場合、ユニットテストソースコード内から直接使用するプロパティーを提供できるようにしたい場合があります。
これは Camel 2.10 以降で可能になりました。Camel テストキット (例: **CamelTestSupport** クラス) は次のメソッドを提供します。

- **useOverridePropertiesWithPropertiesComponent**
- **ignoreMissingLocationWithPropertiesComponent**

たとえば、単体テストクラスでは、**useOverridePropertiesWithPropertiesComponent** メソッドをオーバーライドして、優先的に使用する必要があるプロパティーを含む **java.util.Properties** を返すことができます。

250.24.1. 単体テストソース内からのプロパティーの提供

これは、camel-test、camel-test-spring、camel-test-blueprint などの任意の Camel テストキットから実行できます。

ignoreMissingLocationWithPropertiesComponent を使用して、プロパティーの場所にアクセスできない環境で単体テストを実行する場合など、発見できなかった場所を無視するように Camel に指示できます。

250.25. USING @PROPERTYINJECT

Camel 2.12 以降で利用可能

Camel では、フィールドとセッターメソッドに設定できる **@PropertyInject** アノテーションを使用して、POJO にプロパティープレースホルダーを挿入できます。

たとえば、以下に示すように **RouteBuilder** クラスで使用できます。

```
public class MyRouteBuilder extends RouteBuilder {

    @PropertyInject("hello")
    private String greeting;

    @Override
    public void configure() throws Exception {
        from("direct:start")
            .transform().constant(greeting)
            .to("{{result}}");
    }

}
```

グリーティングフィールドに **@PropertyInject** でアノテーションを付け、キー **"hello"** を使用するように定義していることに注意してください。Camel はこのキーを使用してプロパティーを検索し、その値を String 型に変換して注入します。

キーで複数のプレースホルダーとテキストを使用することもできます。たとえば、次のことができます。

```
@PropertyInject("Hello {{name}} how are you?")
private String greeting;
```

これにより、キー **"name"** でプレースホルダーが検索されます。

キーが存在しない場合は、次のようにデフォルト値を追加することもできます。

```
@PropertyInject(value = "myTimeout", defaultValue = "5000")
private int timeout;
```

250.26. すぐに使用できる機能の使用

Camel 2.14.1 以降で利用可能

Properties コンポーネントには、すぐに使用できる次の機能が含まれています

- **env** - OS 環境変数からプロパティーを検索する関数
- **sys** - Java JVM システムプロパティーからプロパティーを検索する関数
- **service** - サービス命名イディオムを使用して OS 環境変数からプロパティーを検索する関数
- **service.name** - **Camel 2.16.1:** ホスト名部分のみを返すサービス命名イディオムを使用して、OS 環境変数からプロパティーを検索する関数
- **service.port** - **Camel 2.16.1:** ポート部分のみを返すサービス命名イディオムを使用して、OS 環境変数からプロパティーを検索する関数

ご覧のとおり、これらの関数は、環境から値を簡単に検索できるようにすることを目的としています。これらはすぐに提供されるため、次のように簡単に使用できます。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="direct:start"/>
    <to uri="{`{env:SOMENAME}`}"/>
    <to uri="{`{sys:MyJvmPropertyName}`}"/>
  </route>
</camelContext>
```

デフォルト値も使用できるため、プロパティーが存在しない場合は、以下に示すようにデフォルト値を定義できます。デフォルト値は **log:foo** および **log:bar** 値です。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="direct:start"/>
    <to uri="{`{env:SOMENAME:log:foo}`}"/>
    <to uri="{`{sys:MyJvmPropertyName:log:bar}`}"/>
  </route>
</camelContext>
```

service 関数は、OS 環境変数を使用して定義されたサービスをサービス命名イディオムを使用して検索し、**hostname : port** を使用してサービスの場所を参照するためのものです。

- NAME_SERVICE_HOST
- NAME_SERVICE_PORT

つまり、サービスは **_SERVICE_HOST** と **_SERVICE_PORT** を接頭辞として使用します。したがって、サービスの名前が FOO の場合、OS 環境変数は次のように設定する必要があります。

```
export $FOO_SERVICE_HOST=myserver
export $FOO_SERVICE_PORT=8888
```

たとえば、FOO サービスがリモート HTTP サービスの場合、Camel エンドポイント uri でサービスを参照し、HTTP コンポーネントを使用して HTTP 呼び出しを行うことができます。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="direct:start"/>
    <to uri="http://{`{service:FOO}`}/myapp"/>
  </route>
</camelContext>
```

また、サービスが定義されていない場合は、デフォルト値を使用できます。たとえば、単体テストなどのために localhost でサービスを呼び出す場合などです。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="direct:start"/>
```

```
    <to uri="http://{`{service:FOO:localhost:8080}`}/myapp"/>
  </route>
</camelContext>
```

250.27. カスタム関数の使用

Camel 2.14.1 以降で利用可能

プロパティー コンポーネントを使用すると、プロパティープレースホルダーの解析中に使用できるサードパーティー関数をプラグインできます。これらの関数は、データベースの検索、カスタム計算の実行など、プレースホルダーを解決するためのカスタムロジックを実行できます。関数の名前は、プレースホルダーで使用される接頭辞になります。これは、以下のコード例で最もよく示されています

```
<bean id="beerFunction" class="MyBeerFunction"/>

<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <propertyPlaceholder id="properties">
    <propertiesFunction ref="beerFunction"/>
  </propertyPlaceholder>

  <route>
    <from uri="direct:start"/>
    <to uri="{`{beer:FOO}`}"/>
    <to uri="{`{beer:BAR}`}"/>
  </route>
</camelContext>
```

注記

camel 2.19.0 から、(propertyPlaceholder タグの) location 属性は必須ではなくなりました

ここでは、カスタム関数を使用するために **<propertyPlaceholder>** を定義した Camel XML ルートがあります。このカスタム関数は、bean id と呼ばれます (例: **beerFunction**)。beer 関数は名前として **"beer"** を使用するため、プレースホルダー構文は、**beer:value** で開始することにより、beer 関数をトリガーできます。

関数の実装は、以下に示すように 2 つの方法のみです。

```
public static final class MyBeerFunction implements PropertiesFunction {

    @Override
    public String getName() {
        return "beer";
    }

    @Override
    public String apply(String remainder) {
        return "mock:" + remainder.toLowerCase();
    }
}
```

関数は **org.apache.camel.component.properties.PropertiesFunction** インターフェイスを実装する必要があります。**getName** メソッドは、beer などの関数の名前です。 **apply** メソッドは、実行するカス

タムロジックを実装する場所です。サンプルコードは単体テストからのものであるため、モックエンドポイントを参照する値を返すだけです。

Java コードからカスタム関数を登録するには、次のようにします。

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.addFunction(new MyBeerFunction());
```

250.28. 関連項目

- Properties コンポーネント
- プロパティーで暗号化された値 (パスワードなど) を使用するための Jasypt

第251章 PROTOBUF DATAFORMAT

Camel バージョン 2.2.0 以降で利用可能

第252章 PROTOBUF - プロトコルバッファー

プロトコルバッファー - Google のデータエクスチェンジフォーマット

Camel は、Java と Protocol Buffer プロトコルの間でシリアル化するためのデータ形式を提供します。プロジェクトのサイトには、xml ではなくこの形式を選択する 理由が詳しく説明されています。Protocol Buffer は言語とプラットフォームに中立であるため、Camel ルートによって生成されたメッセージは、他の言語の実装によって消費される可能性があります。

API サイト
Protobuf の実装

Protobuf Java チュートリアル

252.1. PROTOBUF オプション

Protobuf データ形式は、以下に示す 3 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
instanceClass		String	アンアームシャリング時に使用するクラスの名前
contentTypeFormat	native	String	protobuf メッセージが Java から (to) シリアライズ/デシリアライズされるコンテンツタイプ形式を定義します。形式は、ネイティブの protobuf または json フィールド表現のネイティブまたは json のいずれかです。デフォルト値はネイティブです。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

252.2. コンテンツタイプ形式 (CAMEL 2.19 以降)

JSON メッセージを解析して protobuf 形式に変換し、ネイティブの util コンバーターを使用して解析を解除することができます。このオプションを使用するには、contentTypeFormat 値を 'json' に設定するか、2 番目のパラメーターで protobuf を呼び出します。デフォルトのインスタンスが指定されていない場合は、常にネイティブの protobuf 形式を使用します。サンプルコードを以下に示します。

```
from("direct:marshal")
    .unmarshal()
    .protobuf("org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Person", "json")
    .to("mock:reverse");
```

252.3. PROTOBUF の概要

Protobuf の使用方法の概要です。詳細については、完全なチュートリアル を参照してください

252.4. PROTO フォーマットの定義

最初のステップは、エクスチェンジのボディーの形式を定義することです。これは .proto ファイルで次のように定義されています。

addressbook.proto

```
syntax = "proto2";

package org.apache.camel.component.protobuf;

option java_package = "org.apache.camel.component.protobuf";
option java_outer_classname = "AddressBookProtos";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

252.5. JAVA クラスの生成

Protobuf SDK は、.proto ファイルで定義した形式の Java クラスを生成するコンパイラーを提供します。お使いのオペレーティングシステムが Protobuf Java コードジェネレーター maven プラグイン をサポートしている場合、pom.xml に次の設定を追加することで、protobuf Java コードの生成を自動化できます。

プロジェクト pom.xml の **<build>** タグ内にオペレーティングシステムと CPU アーキテクチャー検出拡張機能を挿入するか、手動で ${os.detected.classifier} パラメーターを設定します。

```
<extensions>
  <extension>
    <groupId>kr.motd.maven</groupId>
    <artifactId>os-maven-plugin</artifactId>
    <version>1.4.1.Final</version>
  </extension>
</extensions>
```

プロジェクト pom.xml の gRPC および protobuf Java コードジェネレータープラグイン **<plugins>** タグを挿入します。

```
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <version>0.5.0</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <goals>
        <goal>test-compile</goal>
        <goal>compile</goal>
      </goals>
      <configuration>
        <protocArtifact>com.google.protobuf:protoc:${protobuf-
version}:exe:${os.detected.classifier}</protocArtifact>
      </configuration>
    </execution>
  </executions>
</plugin>
```

手動で必要な追加のサポート対象言語に対してコンパイラーを実行することもできます。

protoc --java_out=. ./proto/addressbook.proto

これにより、Person と AddressBook の内部クラスを含む AddressBookProtos という名前の単一の Java クラスが生成されます。ビルダーも実装されています。生成されたクラスは、シリアル化メカニズムに必要な com.google.protobuf.Message を実装します。このため、これらのクラスのみがエクスチェンジのボディーで使用されることが重要です。com.google.protobuf.Message を実装しないクラスを使用するようにデータ形式に指示しようとすると、Camel はルート作成時に例外を出力します。生成されたビルダーを使用して、既存のドメインクラスのデータを変換します。

252.6. JAVA DSL

ProtobufDataFormat インスタンスを作成して、このように Camel DataFormat マーシャルおよびアンマーシャル API に渡すことができます。

```
ProtobufDataFormat format = new ProtobufDataFormat(Person.getDefaultInstance());

from("direct:in").marshal(format);
from("direct:back").unmarshal(format).to("mock:reverse");
```

または、非整列化デフォルトインスタンスまたはデフォルトインスタンスクラス名を次のように渡す DSL protobuf() を使用します。

```
// You don't need to specify the default instance for protobuf marshaling
from("direct:marshal").marshal().protobuf();
from("direct:unmarshalA").unmarshal()
    .protobuf("org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Person")
    .to("mock:reverse");

from("direct:unmarshalB").unmarshal().protobuf(Person.getDefaultInstance()).to("mock:reverse");
```

252.7. SPRING DSL

次の例は、Protobuf データ型を設定する Spring を使用して非整列化するために Protobuf を使用する方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
      <protobuf
instanceClass="org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Person" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

252.8. 依存関係

camel ルートで Protobuf を使用するには、このデータ形式を実装する **camel-protobuf** に依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-protobuf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

252.9. 関連項目

- Camel gRPC コンポーネント

第253章 PUBNUB コンポーネント

Camel バージョン 2.19 以降で利用可能

Camel PubNub コンポーネントを使用して、接続されたデバイスの PubNub データストリームネットワークと通信できます。このコンポーネントは pubnub Java ライブラリー を使用します。

ユースケースには以下が含まれます。

- チャットルーム: メッセージの送受信
- ロケーションとコネクティッドカー: タクシー配車
- スマートセンサー: データを視覚化するためにセンサーからデータを受信する
- ヘルス: 患者のウェアラブルデバイスからの心拍数のモニタリング
- マルチプレイヤーゲーム
- インタラクティブメディア: 視聴者参加型投票システム

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pubnub</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

253.1. URI 形式

```
pubnub:channel[?options]
```

channel は、パブリッシュまたはサブスクライブする PubNub チャネルです。

253.2. オプション

PubNub コンポーネントにはオプションがありません。

PubNub エンドポイントは、URI 構文を使用して設定されます。

```
pubnub:channel
```

パスおよびクエリーパラメーターを使用します。

253.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
channel	**必須** イベントのサブスクライブ/パブリッシュに使用されるチャネル		String

253.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
uuid (Common)	デバイス識別子として使用される UUID。渡されない場合、デフォルトの UUID が生成されます。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
withPresence (consumer)	関連するプレゼンス情報もサブスクライブする	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
operation (producer)	実行する操作。パブリッシュ: デフォルト。チャンネルのすべてのサブスクライバーにメッセージを送信します。FIRE: クライアントが BLOCKS イベントハンドラーにメッセージを送信できるようにします。これらのメッセージは、チャネルに登録されている任意のイベントハンドラーに直接送られます。HERENOW: チャネルに現在サブスクライブしている一意のユーザー ID のリストと合計占有数を含む、チャネルの現在の状態に関する情報を取得します。WHERENOW: uuid がサブスクライブされているチャネルの現在のリストに関する情報を取得します。GETSTATE: サブスクライバ uuid に固有のキーと値のペアを取得するために使用されます。状態情報は、キーと値のペアの JSON オブジェクトとして提供されます。SETSTATE: サブスクライバー uuid に固有のキーと値のペアを設定するために使用されます。GETHISTORY: チャネルの履歴メッセージを取得します。		String
pubnub (advanced)	レジストリー内の Pubnub クライアントへの参照。		PubNub
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
authKey (security)	Access Manager が使用されている場合、クライアントは制限されたすべてのリクエストでこの authKey を使用します。		String
cipherKey (security)	cipher が渡されると、PubNub との間のすべての通信が暗号化されます。		String
publishKey (security)	PubNub アカウントから取得した公開キー。メッセージを発行するときに必要です。		String
secretKey (security)	メッセージの署名に使用される秘密鍵。		String
secure (security)	安全な送信のために SSL を使用します。	true	boolean
subscribeKey (security)	PubNub アカウントから取得したサブスクライブキー。チャネルをサブスクライブするとき、またはプレゼンスイベントをリッスンするときに必要です。		文字列

253.3. サブスクライブ時のメッセージヘッダー

名前	説明
CamelPubNubTimeToken	イベントのタイムスタンプ。
CamelPubNubChannel	メッセージが属するチャネル。

253.4. メッセージボディー

メッセージボディーには、オブジェクト、配列、Int、および文字列を含む JSON シリアル化可能なデータを含めることができます。メッセージデータには、シリアル化されない特別な Java V4 クラスまたは関数を含めないでください。文字列コンテンツには、シングルバイトまたはマルチバイトの UTF-8 を含めることができます

送信時のオブジェクトのシリアル化は自動的に行われます。完全なオブジェクトをメッセージペイロードとして渡すだけです。PubNub がオブジェクトのシリアル化を処理します。

メッセージボディーを受信するときは、PubNub API によって提供されるオブジェクトを利用します。

253.5. 例

253.5.1. イベントの公開

生成時のデフォルトの動作。次のスニペットは、PojoBean によって生成されたイベントをチャネル iot に発行します。

```
from("timer:mytimer")
    // generate some data as POJO.
    .bean(PojoBean.class)
    .to("pubnub:iot?publishKey=mypublishKey");
```

253.5.2. BLOCKS イベントハンドラーとも呼ばれる Fire イベント

呼び出すことができるすべての種類のサーバーレス関数については、https://www.pubnub.com/blocks-catalog/ を参照してください。位置情報検索の例

```
from("timer:geotimer")
    .process(exchange -> exchange.getIn().setBody(new Foo("bar", "TEXT")))
    .to("pubnub:eon-maps-geolocation-input?
operation=fire&publishKey=mypubkey&subscribeKey=mysubkey");

from("pubnub:eon-map-geolocation-output?subscribeKey=mysubkey)
    // geolocation output will be logged here
    .log("${body}");
```

253.5.3. イベントのサブスクライブ

次のスニペットは、iot チャネルでイベントをリッスンします。オプション withPresens を追加できる場合は、チャネルの Join、Leave asf イベントも受信します。

-

```
from("pubnub:iot?subscribeKey=mySubscribeKey")
    .log("${body}")
    .to("mock:result");
```

253.5.4. 操作の実行

herenow : チャネルに現在サブスクライブしている一意のユーザー ID のリストや、チャネルの合計占有数など、チャネルの現在の状態に関する情報を取得します。

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=herenow")
    .to("mock:result");
```

wherenow : uuid がサブスクライブされているチャネルの現在のリストに関する情報を取得します

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=wherenow&uuid=spyonme")
    .to("mock:result");
```

setstate : サブスクライバー uuid に固有のキーと値のペアを設定するために使用されます。

```
from("direct:control")
    .bean(StateGenerator.class)
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=setstate&uuid=myuuid");
```

gethistory : チャネルの履歴メッセージを取得します。

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=gethistory");
```

テストディレクトリーには、PubNub 機能のいくつかを示す例がいくつかあります。パブリッシュキーとサブスクライブキーを取得できる PubNub アカウントが必要です。

PubNubSensorExample の例には、PubNub によって提供されたサブスクライブキーがすでに含まれているため、これはアカウントなしで実行する準備ができています。この例は、センサーデータの無限ストリームにサブスクライブする PubNub コンポーネントを示しています。

253.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- RSS

第254章 QUARTZ コンポーネント (非推奨)

Camel バージョン 1.0 以降で利用可能

quartz: コンポーネントは、Quartz Scheduler 1.x を使用してスケジュールされたメッセージ配信を提供します。
各エンドポイントは異なるタイマーを表します (Quartz 用語では、Trigger と JobDetail)。

ヒント

Quartz 2.x を使用している場合、Camel 2.12 以降では使用すべき Quartz2 コンポーネントがあります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-quartz</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

254.1. URI 形式

```
quartz://timerName?options
quartz://groupName/timerName?options
quartz://groupName/timerName?cron=expression
quartz://timerName?cron=expression
```

コンポーネントは **CronTrigger** または **SimpleTrigger** を使用します。cron 式が指定されていない場合、コンポーネントは単純なトリガーを使用します。**groupName** が指定されていない場合、quartz コンポーネントは **Camel** グループ名を使用します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

254.2. オプション

Quartz コンポーネントは、以下にリストされている 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
factory (advanced)	スケジューラーの作成に使用されるカスタム SchedulerFactory を使用するには。		SchedulerFactory
scheduler (advanced)	新しいスケジューラーを作成する代わりに、カスタム設定された Quartz スケジューラーを使用するには。		スケジューラー
properties (consumer)	Quartz スケジューラーを設定するためのプロパティー。		Properties

名前	説明	デフォルト	タイプ
propertiesFile (consumer)	クラスパスからロードするプロパティーのファイル名。		String
startDelayedSeconds (スケジューラー)	Quartz スケジューラーを開始する前に待機する秒数。		int
autoStartScheduler (consumer)	スケジューラーを自動起動するかどうか。このオプションのデフォルトは true です	true	boolean
enableJmx (consumer)	JMX から Quartz スケジューラーを管理できるようにする Quartz JMX を有効にするかどうか。このオプションのデフォルトは true です	true	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Quartz エンドポイントは、URI 構文を使用して設定されます。

```
quartz:groupName/timerName
```

パスおよびクエリーパラメーターを使用します。

254.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
groupName	使用する Quartz グループ名。グループ名とタイマー名の組み合わせは一意である必要があります。	Camel	String
timerName	**必須** 使用する quartz タイマー名。グループ名とタイマー名の組み合わせは一意である必要があります。		String

254.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cron (consumer)	トリガーするタイミングを定義する cron 式を指定します。		String
deleteJob (consumer)	true に設定すると、ルートが停止したときにトリガーが自動的に削除されます。それ以外の場合は、false に設定すると、スケジューラーに残ります。false に設定すると、ユーザーがキャメル Uri で事前設定されたトリガーを再利用できることも意味します。名前が一致していることを確認してください。deleteJob と pauseJob の両方を true に設定することはできないことに注意してください。	true	boolean
fireNow (consumer)	単純なトリガーを使用して開始されたときにスケジューラーをできるだけ早く起動するかどうか (このオプションは cron をサポートしていません)	false	boolean
pauseJob (consumer)	true に設定すると、ルートが停止したときにトリガーが自動的に一時停止します。それ以外の場合は、false に設定すると、スケジューラーに残ります。false に設定すると、ユーザーがキャメル Uri で事前設定されたトリガーを再利用できることも意味します。名前が一致していることを確認してください。deleteJob と pauseJob の両方を true に設定することはできないことに注意してください。	false	boolean
startDelayedSeconds (consumer)	Quartz スケジューラーを開始する前に待機する秒数。		int
stateful (consumer)	デフォルトのジョブの代わりに Quartz StatefulJob を使用します。	false	boolean
usingFixedCamelContextName (consumer)	true の場合、JobDataMap は CamelContext 名を直接使用して CamelContext を参照します。false の場合、JobDataMap はデプロイ時に変更される可能性のある CamelContext 管理名を使用します。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
jobParameters (advanced)	ジョブの追加オプションを設定するには。		Map
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
triggerParameters (advanced)	トリガーの追加オプションを設定します。		Map

StatefulJob を使用すると、JobDataMap はジョブの実行ごとに再永続化されるため、次の実行のために状態が保持されます。

情報: **OSGiで実行し、quartz ルートを持つ複数のバンドルを持つ。**Apache ServiceMix や Apache Karaf などの OSGi で実行し、Quartz エンドポイントから開始する Camel ルートを持つ複数のバンドルがある場合、<camelContext> に **id** を割り当てると
、OSGi コンテナの **QuartzScheduler** で必要とされるのでこの ID が一意であるか確認してください。<camelContext> に **id** を設定しない場合
一意の ID が自動的に割り当てられ、問題はありません。

254.3. QUARTZ.PROPERTIES ファイルの設定

デフォルトでは、Quartz はクラスパスの **org/quartz** ディレクトリーにある **Quartz.properties** ファイルを探します。WAR デプロイメントを使用している場合、これは、Quartz.properties を **WEB-INF/classes/org/quartz** にドロップするだけであることを意味します。

ただし、Camel Quartz コンポーネントでは、プロパティーを設定することもできます。

パラメーター	デフォルト	タイプ	説明
properties	**null**	**Properties**	Camel 2.4 : **java.util.Properties** インスタンスを設定できます。

パラメーター	デフォルト	タイプ	説明
propertiesFile	**null**	**String**	Camel 2.4 : クラスパスからロードするプロパティーのファイル名

これを行うには、Spring XML で次のように設定できます。

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
    <property name="propertiesFile" value="com/mycompany/myquartz.properties"/>
</bean>
```

254.4. JMX で QUARTZ スケジューラーを有効にする

JMX を有効にするには、Quartz スケジューラープロパティーを設定する必要があります。
これは通常、設定ファイルでオプション **org.quartz.scheduler.jmx.export** を **true** 値に設定することです。

Camel 2.13 以降では、明示的に無効にしない限り、Camel は自動的にこのオプションを true に設定します。

254.5. QUARTZ スケジューラーの開始

以下に例を示します。

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
    <property name="startDelayedSeconds" value="5"/>
</bean>
```

254.6. クラスタリング

Camel 2.4 以降で利用可能

クラスター化モードで Quartz を使用する場合 (例: **JobStore** がクラスター化される場合など)、その後、Camel 2.4 以降、Quartz コンポーネントは、ノードが停止/シャットダウンされているときにトリガーを一時停止/削除 **しません**。これにより、クラスター内の他のノードでトリガーを実行し続けることができます。

注記:クラスター化されたノードで実行している場合、エンドポイントのジョブ名/グループが一意であることを確認するためのチェックは行われません。

254.7. メッセージヘッダー

Camel は、Quartz 実行コンテキストからの getter をヘッダー値として追加します。次のヘッダーが追加されます。
calendar、**fireTime**、**jobDetail**、**jobInstance**、**jobRuntTime**、**mergedJobDataMap**、**nextFireTime**、

previousFireTime、**refireCount**、**result**、**ScheduledFireTime**、**scheduler**、**trigger**、**triggerName**、**triggerGroup**。

fireTime ヘッダーには、エクスチェンジが開始されたときの **java.util.Date** が含まれています。

254.8. CRON トリガーの使用

Quartz は、便利な形式でタイマーを指定するための Cron のような式 をサポートしています。これらの式は **cron** URI パラメーターで使用できます。ただし、有効な URI エンコーディングを維持するために、スペースの代わりに + を使用できます。Quartz は、cron 式の使用方法に関する 簡単なチュートリアル を提供します。

たとえば、次の例では、平日の午後 12 時 (正午) から午後 6 時まで、5 分ごとにメッセージが送信されます。

```
from("quartz://myGroup/myTimerName?cron=0+0/5+12-18+?+*+MON-
FRI").to("activemq:Totally.Rocks");
```

これは、cron 式を使用するのと同じです

```
0 0/5 12-18 ? * MON-FRI
```

次の表は、有効な URI 構文を維持するために使用する URI 文字エンコーディングを示しています。

URI 文字	Cron 文字
+	スペース

254.9. タイムゾーンの指定

Camel 2.8.1 以降で利用可能
Quartz Scheduler を使用すると、トリガーごとにタイムゾーンを設定できます。たとえば、自国のタイムゾーンを使用するには、次のようにします。

```
quartz://groupName/timerName?cron=0+0/5+12-18+?+*+MON-
FRI&trigger.timeZone=Europe/Stockholm
```

timeZone 値は、**java.util.TimeZone** によって受け入れられる値です。

Camel 2.8.0 以前のバージョンでは、カスタム **String** を **java.util.TimeZone** タイプコンバーター に提供して、エンドポイント uri からこれを設定できるようにする必要があります。
Camel 2.8.1 以降では、そのようなタイプコンバーターが camel-core に含まれています。

254.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

- Quartz2
- Timer

第255章 QUARTZ2 COMPONENT

Camel バージョン 2.12 以降で利用可能

quartz2: コンポーネントは、Quartz Scheduler 2.x を使用してスケジュールされたメッセージ配信を提供します。
各エンドポイントは異なるタイマーを表します (Quartz 用語では、Trigger と JobDetail)。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-quartz2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記: Quartz 2.x API は Quartz 1.x と互換性がありません。古い Quartz 1.x を使い続ける必要がある場合は、
代わりに古い Quartz コンポーネントを使用してください。

255.1. URI 形式

```
quartz2://timerName?options
quartz2://groupName/timerName?options
quartz2://groupName/timerName?cron=expression
quartz2://timerName?cron=expression
```

コンポーネントは **CronTrigger** または **SimpleTrigger** を使用します。cron 式が指定されていない場合、コンポーネントは単純なトリガーを使用します。**groupName** が指定されていない場合、quartz コンポーネントは **Camel** グループ名を使用します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

255.2. オプション

Quartz2 コンポーネントは、以下にリストされている 11 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
autoStartScheduler (スケジューラー)	スケジューラーを自動起動するかどうか。このオプションのデフォルトは true です	true	boolean
startDelayedSeconds (スケジューラー)	Quartz スケジューラーを開始する前に待機する秒数。		int
prefixJobNameWith EndpointId (consumer)	Quartz ジョブにエンドポイント ID の接頭辞を付けるかどうか。このオプションのデフォルトは false です。	false	boolean

名前	説明	デフォルト	タイプ
enableJmx (consumer)	JMX から Quartz スケジューラーを管理できるようにする Quartz JMX を有効にするかどうか。このオプションのデフォルトは true です	true	boolean
properties (consumer)	Quartz スケジューラーを設定するためのプロパティー。		Properties
propertiesFile (consumer)	クラスパスからロードするプロパティーのファイル名。		String
prefixInstanceName (consumer)	Quartz Scheduler インスタンス名の前に CamelContext 名を付けるかどうか。これはデフォルトで有効になっており、各 CamelContext がデフォルトで独自の Quartz スケジューラーインスタンスを使用できるようになっています。このオプションを false に設定すると、複数の CamelContext 間で Quartz スケジューラーインスタンスを再利用できます。	true	boolean
interruptJobsOnShutdown (スケジューラー)	シャットダウン時にジョブを中断するかどうか。これにより、スケジューラーがより迅速にシャットダウンし、実行中のジョブを中断しようとします。これを有効にすると、実行中のジョブが中断されて失敗する可能性があります。	false	boolean
schedulerFactory (advanced)	スケジューラーの作成に使用されるカスタム SchedulerFactory を使用するには。		SchedulerFactory
scheduler (advanced)	新しいスケジューラーを作成する代わりに、カスタム設定された Quartz スケジューラーを使用するには。		スケジューラー
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Quartz2 エンドポイントは、URI 構文を使用して設定されます。

```
quartz2:groupName/triggerName
```

パスおよびクエリーパラメーターを使用します。

255.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
groupName	使用する Quartz グループ名。グループ名とタイマー名の組み合わせは一意である必要があります。	Camel	String
triggerName	**必須** 使用する quartz タイマー名。グループ名とタイマー名の組み合わせは一意である必要があります。		String

255.2.2. クエリーパラメーター (19 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cron (consumer)	トリガーするタイミングを定義する cron 式を指定します。		String
deleteJob (consumer)	true に設定すると、ルートが停止したときにトリガーが自動的に削除されます。それ以外の場合は、false に設定すると、スケジューラーに残ります。false に設定すると、ユーザーがキャメル Uri で事前設定されたトリガーを再利用できることも意味します。名前が一致していることを確認してください。deleteJob と pauseJob の両方を true に設定することはできないことに注意してください。	true	boolean
durableJob (consumer)	ジョブが孤立した後もジョブを保存したままにするかどうか (それを指すトリガーはありません)。	false	boolean
pauseJob (consumer)	true に設定すると、ルートが停止したときにトリガーが自動的に一時停止します。それ以外の場合は、false に設定すると、スケジューラーに残ります。false に設定すると、ユーザーがキャメル Uri で事前設定されたトリガーを再利用できることも意味します。名前が一致していることを確認してください。deleteJob と pauseJob の両方を true に設定することはできないことに注意してください。	false	boolean

名前	説明	デフォルト	タイプ
recoverableJob (consumer)	リカバリーまたはフェイルオーバー状況が発生した場合に、ジョブを再実行するかどうかをスケジューラーに指示します。	false	boolean
stateful (consumer)	デフォルトのジョブの代わりに、Quartz PersistJobDataAfterExecution および DisallowConcurrentExecution を使用します。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
customCalendar (advanced)	特定の日付範囲を避けるカスタムカレンダーを指定します		Calendar
jobParameters (advanced)	ジョブの追加オプションを設定するには。		Map
prefixJobNameWithEndpoint Id (advanced)	ジョブ名の前にエンドポイント ID を付けるかどうか	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
triggerParameters (advanced)	トリガーの追加オプションを設定します。		Map
usingFixedCamelContextName (advanced)	true の場合、JobDataMap は CamelContext 名を直接使用して CamelContext を参照します。false の場合、JobDataMap はデプロイ時に変更される可能性のある CamelContext 管理名を使用します。	false	boolean
autoStartScheduler (スケジューラー)	スケジューラーを自動起動するかどうか。	true	boolean
fireNow (スケジューラー)	true の場合、SimpleTrigger の使用時にルートが開始されたときにトリガーが発生します。	false	boolean

名前	説明	デフォルト	タイプ
startDelayedSeconds (スケジューラー)	Quartz スケジューラーを開始する前に待機する秒数。		int
triggerStartDelay (スケジューラー)	スケジューラーがすでに開始されている場合は、現在の時間の少し後にトリガーを開始して、ジョブが開始される前にエンドポイントが完全に開始されるようにします。	500	long

たとえば、次のルーティングルールは、**mock:results** エンドポイントに対して 2 つのタイマーイベントを発生させます。

```
from("quartz2://myGroup/myTimerName?
trigger.repeatInterval=2&trigger.repeatCount=1").routeId("myRoute")
    .to("mock:result");
```

stateful=true を使用すると、JobDataMap はジョブの実行ごとに再永続化されるため、次の実行のために状態が維持されます。

情報: **OSGi で実行し、quartz ルートを持つ複数のバンドルを持つ。**Apache ServiceMix や Apache Karaf などの OSGi で実行し、Quartz2エンドポイントから始まる Camel ルートを持つ複数のバンドルがある場合、<camelContext> に **id** を割り当てると、OSGi コンテナの **QuartzScheduler** でこの id が必要となるため一意でなければなりません。<camelContext> に **id** を設定しない場合、一意の id が自動的に割り当てられ、問題はありません。

255.3. QUARTZ.PROPERTIES ファイルの設定

デフォルトでは、Quartz はクラスパスの **org/quartz** ディレクトリーにある **Quartz.properties** ファイルを探します。WAR デプロイメントを使用している場合、これは、Quartz.properties を **WEB-INF/classes/org/quartz** にドロップするだけであることを意味します。

ただし、Camel Quartz2 コンポーネントでは、プロパティーを設定することもできます。

パラメーター	デフォルト	タイプ	説明
properties	**null**	**プロパティー**	**java.util.Properties** インスタンスを設定できます。
propertiesFile	**null**	**String**	クラスパスからロードするプロパティーのファイル名。

これを行うには、Spring XML で次のように設定できます。

```
<bean id="quartz2" class="org.apache.camel.component.quartz2.QuartzComponent">
    <property name="propertiesFile" value="com/mycompany/myquartz.properties"/>
</bean>
```

255.4. JMX で QUARTZ スケジューラーを有効にする

JMX を有効にするには、Quartz スケジューラープロパティーを設定する必要があります。
これは通常、設定ファイルでオプション **org.quartz.scheduler.jmx.export** を **true** 値に設定することです。

Camel 2.13 以降では、明示的に無効にしない限り、Camel は自動的にこのオプションを true に設定します。

255.5. QUARTZ スケジューラーの開始

Quartz2 コンポーネントでは、Quartz スケジューラーを遅延して開始するか、または自動開始しないかを選択できます。

以下に例を示します。

```
<bean id="quartz2" class="org.apache.camel.component.quartz2.QuartzComponent">
    <property name="startDelayedSeconds" value="5"/>
</bean>
```

255.6. クラスタリング

クラスター化モードで Quartz を使用する場合 (例: **JobStore** がクラスター化される場合など)、Quartz2 コンポーネントでは、ノードの停止/シャットダウン時にはトリガーの一時停止や削除は行われ **ません**。これにより、クラスター内の他のノードでトリガーを実行し続けることができます。

注記:クラスター化されたノードで実行している場合、エンドポイントのジョブ名/グループが一意であることを確認するためのチェックは行われません。

255.7. メッセージヘッダー

Camel は、Quartz 実行コンテキストからの getter をヘッダー値として追加します。次のヘッダーが追加されます。
calendar、**fireTime**、**jobDetail**、**jobInstance**、**jobRuntTime**、**mergedJobDataMap**、**nextFireTime**、**previousFireTime**、**refireCount**、**result**、**ScheduledFireTime**、**scheduler**、**trigger**、**triggerName**、**triggerGroup**。

fireTime ヘッダーには、エクスチェンジが開始されたときの **java.util.Date** が含まれています。

255.8. CRON トリガーの使用

Quartz は、便利な形式でタイマーを指定するための Cron のような式 をサポートしています。これらの式は **cron** URI パラメーターで使用できます。ただし、有効な URI エンコーディングを維持するために、スペースの代わりに + を使用できます。

たとえば、次の例では、平日の午後 12 時 (正午) から午後 6 時まで、5 分ごとにメッセージが送信されます。

```
from("quartz2://myGroup/myTimerName?cron=0+0/5+12-18+?+*+MON-FRI")
    .to("activemq:Totally.Rocks");
```

これは、cron 式を使用するのと同じです

```
0 0/5 12-18 ? * MON-FRI
```

次の表は、有効な URI 構文を維持するために使用する URI 文字エンコーディングを示しています。

URI 文字	Cron 文字
+	スペース

255.9. タイムゾーンの指定

Quartz Scheduler を使用すると、トリガーごとにタイムゾーンを設定できます。たとえば、自国のタイムゾーンを使用するには、次のようにします。

```
quartz2://groupName/timerName?cron=0+0/5+12-18+?+*+MON-
FRI&trigger.timeZone=Europe/Stockholm
```

timeZone 値は、**java.util.TimeZone** によって受け入れられる値です。

255.10. QUARTZSCHEDULEDPOLLCONSUMERSCHEDULER の使用

Quartz2 コンポーネントは、File および FTP コンシューマーなどの Polling Consumer に cron ベースのスケジューリングを使用できるようにする Polling Consumer スケジューラーを提供します。

たとえば、cron ベースの式を使用して 2 秒ごとにファイルをポーリングするには、Camel ルートを次のように単純に定義できます。

```
from("file:inbox?scheduler=quartz2&scheduler.cron=0/2+*+*+*+*+?")
  .to("bean:process");
```

scheduler=quartz2 を定義して Camel に Quartz2 ベースのスケジューラーを使用するように指示していることに注意してください。次に、**scheduler.xxx** オプションを使用してスケジューラーを設定します。Quartz2 スケジューラーでは、cron オプションを設定する必要があります。

次のオプションがサポートされています。

パラメーター	デフォルト	タイプ	説明
quartzScheduler	**null**	**org.quartz.Scheduler**	カスタム Quartz スケジューラーを使用するには。何も設定しない場合、Quartz2 コンポーネントの共有スケジューラーが使用されます。
cron	**null**	**String**	**必須**: ポーリングをトリガーするための cron 式を定義します。

パラメーター	デフォルト	タイプ	説明
triggerId	**null**	**String**	トリガー ID を指定します。何も指定されていない場合は、UUID が生成されて使用されます。
triggerGroup	**QuartzScheduledPollConsumerScheduler**	**String**	トリガーグループを指定します。
timeZone	**デフォルト**	**TimeZone**	CRON トリガーに使用するタイムゾーン。

重要:エンドポイントからこれらのオプションを設定するには、URIの先頭に **scheduler.** を付ける必要があることを忘れないでください。たとえば、トリガー ID とグループを設定するには、次のようにします。

```
    from("file:inbox?scheduler=quartz2&scheduler.cron=0/2+*+*+*+*+?
&scheduler.triggerId=myId&scheduler.triggerGroup=myGroup")
        .to("bean:process");
```

Spring には CRON スケジューラーもあるので、以下も使用できます。

```
    from("file:inbox?scheduler=spring&scheduler.cron=0/2+*+*+*+*+?")
        .to("bean:process");
```

第256章 RABBITMQ コンポーネント

Camel バージョン 2.12 以降で利用可能

rabbitmq: コンポーネントを使用すると、RabbitMQ インスタンスからメッセージを生成および消費できます。このコンポーネントは、RabbitMQ AMQP クライアントを使用して、汎用 AMQP コンポーネントよりも純粋な RabbitMQ アプローチを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rabbitmq</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

256.1. URI 形式

古い構文は **非推奨** です:

```
rabbitmq://hostname[:port]/exchangeName?[options]
```

代わりに、ホスト名とポートをコンポーネントレベルで設定するか、代わりに uri クエリーパラメーターとして指定できます。

新しい構文は次のとおりです。

```
rabbitmq:exchangeName?[options]
```

hostname は 実行中の rabbitmq インスタンスまたはクラスターのホスト名です。ポートはオプションです。指定しない場合は、デフォルトで RabbitMQ クライアントのデフォルト (5672) になります。エクスチェンジ名によって、どのエクスチェンジで作成されたメッセージが送信されるかが決まります。コンシューマーの場合、エクスチェンジ名はキューがどのエクスチェンジにバインドするかを決定します。

256.2. オプション

RabbitMQ コンポーネントは、以下に示す 49 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
hostname (common)	実行中の RabbitMQ インスタンスまたはクラスターのホスト名。		String
portNumber (Common)	実行中の rabbitmq インスタンスまたはクラスターを持つホストのポート番号。	5672	int
username (security)	認証アクセスの場合のユーザー名	guest	String

名前	説明	デフォルト	タイプ
password (security)	認証アクセス用のパスワード	guest	String
vhost (Common)	チャネルの仮想ホスト	/	String
addresses (Common)	このオプションが設定されている場合、camel-rabbitmq はオプションアドレスの設定に基づいて接続を作成しようとします。アドレスの値は、server1:12345、server2:12345 のような文字列です。		String
connectionFactory (common)	カスタム RabbitMQ 接続ファクトリーを使用するには。このオプションが設定されている場合、URI に設定されているすべての接続オプション (connectionTimeout、requestedChannelMax など) は使用されません。		ConnectionFactory
threadPoolSize (consumer)	コンシューマーは、固定数のスレッドで Thread Pool Executor を使用します。この設定により、そのスレッド数を設定できます。	10	int
autoDetectConnection Factory (advanced)	レジストリーからの RabbitMQ 接続ファクトリーの検索を自動検出するかどうか。有効にすると、接続ファクトリーのインスタンスが 1 つだけ検出され、それが使用されます。明示的な接続ファクトリーは、優先されるコンポーネントまたはエンドポイントレベルで設定できます。	true	boolean
connectionTimeout (advanced)	Connection timeout	60000	int
requestedChannelMax (advanced)	接続要求されたチャネルの最大数 (提供されるチャネルの最大数)	0	int
requestedFrameMax (advanced)	接続要求フレーム最大 (提供されるフレームの最大サイズ)	0	int
requestedHeartbeat (advanced)	接続要求されたハートビート (提供される秒単位のハートビート)	60	int
automaticRecovery Enabled (advanced)	接続の自動回復を有効にします (接続のシャットダウンがアプリケーションによって開始されない場合に自動回復を実行する接続の実装を使用します)		Boolean
networkRecoveryInterval (advanced)	ミリ秒単位のネットワーク回復間隔 (ネットワーク障害からの回復時に使用される間隔)	5000	Integer

名前	説明	デフォルト	タイプ
topologyRecoveryEnabled (advanced)	接続トポロジーの回復を有効にします (トポロジーの回復を実行する必要があります)。		Boolean
prefetchEnabled (consumer)	RabbitMQConsumer 側でサービス品質を有効にします。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean
prefetchSize (consumer)	サーバーが配信するコンテンツの最大量 (オクテットで測定)。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchCount (consumer)	サーバーが配信するメッセージの最大数。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchGlobal (consumer)	各コンシューマーではなく、チャネル全体に設定を適用する場合 prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります	false	boolean
channelPoolMaxSize (producer)	プールで開かれているチャネルの最大数を取得する	10	int
channelPoolMaxWait (producer)	プールからのチャネルを待機する最大ミリ秒数を設定します	1000	long
requestTimeout (advanced)	InOut Exchange パターン使用時の応答待ちタイムアウトを設定する (ミリ秒単位) 。	20000	long
requestTimeoutCheckerInterval (advanced)	inOut エクスチェンジの requestTimeoutCheckerInterval を設定する	1000	long
transferException (advanced)	true の場合、コンシューマー側で inOut Exchange が失敗し、原因となった例外がレスポンスで返されます。	false	boolean
publisherAcknowledgements (producer)	true の場合、メッセージはパブリッシャーの確認をオンにしてパブリッシュされます。	false	boolean

名前	説明	デフォルト	タイプ
publisherAcknowledgementsTimeout (producer)	RabbitMQ サーバーからの basic.ack 応答を待機する時間 (ミリ秒)		long
guaranteedDeliveries (producer)	true の場合、メッセージを配信できず (basic.return)、メッセージが必須としてマークされている場合に例外が出力されます。この場合、PublisherAcknowledgement もアクティブ化されます。パブリッシャーの確認も参照してください - メッセージはいつ確認されますか。	false	boolean
mandatory (producer)	このフラグは、メッセージをキューにルーティングできない場合の対応方法をサーバーに指示します。このフラグが設定されている場合、サーバーは Return メソッドでルーティング不可能なメッセージを返します。このフラグがゼロの場合、サーバーはメッセージを通知せずにドロップします。ヘッダーが rabbitmq.MANDATORY である場合、このオプションはオーバーライドされます。	false	boolean
immediate (producer)	このフラグは、メッセージをすぐにキューコンシューマーにルーティングできない場合の対応方法をサーバーに指示します。このフラグが設定されている場合、サーバーは Return メソッドで配信不能メッセージを返します。このフラグがゼロの場合、サーバーはメッセージをキューに入れますが、メッセージが消費されるという保証はありません。ヘッダーが rabbitmq.IMMEDIATE である場合、このオプションはオーバーライドされます。	false	boolean
args (advanced)	さまざまな RabbitMQ の概念を設定するための引数を指定します。それぞれに異なる接頭辞が必要です: Exchange: arg.exchange。Queue: arg.queue.Binding: arg.binding.たとえば、メッセージ ttl 引数でキューを宣言するには: http://localhost:5672/exchange/queueargs=arg.queue.x-message-ttl=60000		Map
clientProperties (advanced)	接続クライアントプロパティー (サーバーとのネゴシエーションで使用されるクライアント情報)		Map
sslProtocol (security)	接続時に SSL を有効にします。受け入れられる値は true、TLS および SSLv3 です。		String

名前	説明	デフォルト	タイプ
trustManager (security)	SSL トラストマネージャーを設定します。このオプションを有効にするには、SSL を有効にする必要があります。		TrustManager
autoAck (consumer)	メッセージを自動確認する必要がある場合	true	boolean
autoDelete (Common)	true の場合、エクスチェンジは使用されなくなった時点で削除されます	true	boolean
durable (Common)	永続的なエクスチェンジを宣言している場合 (エクスチェンジはサーバーの再起動後も存続します)	true	boolean
exclusive (Common)	排他キューは、現在の接続によってのみアクセスでき、その接続が閉じると削除されます。	false	boolean
passive (Common)	パッシブキューは、RabbitMQ ですでに使用可能なキューに依存します。	false	boolean
skipQueueDeclare (Common)	true の場合、プロデューサーはキューを宣言およびバインドしません。これは、既存のルーティングキーを介してメッセージを送信するために使用できます。	false	boolean
skipQueueBind (Common)	true の場合、キューは宣言後にエクスチェンジにバインドされません	false	boolean
skipExchangeDeclare (Common)	これは、交換ではなくキューを宣言する必要がある場合に使用できます。	false	boolean
declare (Common)	オプションが true の場合、camel はエクスチェンジとキューの名前を宣言し、それらをバインドします。オプションが false の場合、camel はサーバー上でエクスチェンジとキューの名前を宣言しません。	true	boolean
deadLetterExchange (common)	デッドレターエクスチェンジの名前		String
deadLetterQueue (common)	配信不能キューの名前		String
deadLetterRoutingKey (common)	デッドレターエクスチェンジのルーティングキー		String

名前	説明	デフォルト	タイプ
deadLetterExchangeType (common)	デッドレターエクスチェンジの種類	直接的な	String
allowNullHeaders (producer)	null 値をヘッダーに渡すことを許可する	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

RabbitMQ エンドポイントは、URI 構文を使用して設定されます。

```
rabbitmq:exchangeName
```

パスおよびクエリーパラメーターを使用します。

256.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
exchangeName	**必須** エクスチェンジ名によって、どのエクスチェンジで作成されたメッセージが送信されるかが決まります。コンシューマーの場合、エクスチェンジ名はキューがどのエクスチェンジにバインドするかを決定します。		文字列

256.2.2. クエリーパラメーター (61 パラメーター)

名前	説明	デフォルト	タイプ
addresses (Common)	このオプションが設定されている場合、camel-rabbitmq はオプションアドレスの設定に基づいて接続を作成しようとします。アドレスの値は、server1:12345、server2:12345 のような文字列です。		Address[]
autoDelete (Common)	true の場合、エクスチェンジは使用されなくなった時点で削除されます	true	boolean

名前	説明	デフォルト	タイプ
connectionFactory (common)	カスタム RabbitMQ 接続ファクトリーを使用するには。このオプションが設定されている場合、URI に設定されているすべての接続オプション (connectionTimeout、requestedChannelMax など) は使用されません。		ConnectionFactory
deadLetterExchange (common)	デッドレターエクスチェンジの名前		String
deadLetterExchangeType (common)	デッドレターエクスチェンジの種類	直接的な	String
deadLetterQueue (common)	配信不能キューの名前		String
deadLetterRoutingKey (common)	デッドレターエクスチェンジのルーティングキー		String
declare (Common)	オプションが true の場合、camel はエクスチェンジとキューの名前を宣言し、それらをバインドします。オプションが false の場合、camel はサーバー上でエクスチェンジとキューの名前を宣言しません。	true	boolean
durable (Common)	永続的なエクスチェンジを宣言している場合 (エクスチェンジはサーバーの再起動後も存続します)	true	boolean
exchangeType (Common)	ダイレクトやトピックなどのエクスチェンジタイプ。	直接的な	String
exclusive (Common)	排他キューは、現在の接続によってのみアクセスでき、その接続が閉じると削除されます。	false	boolean
hostname (common)	実行中の rabbitmq インスタンスまたはクラスターのホスト名。		String
passive (Common)	パッシブキューは、RabbitMQ ですでに使用可能なキューに依存します。	false	boolean
portNumber (Common)	実行中の rabbitmq インスタンスまたはクラスターを持つホストのポート番号。デフォルト値は 5672 です。		int
queue (Common)	メッセージを受信するキュー		String

名前	説明	デフォルト	タイプ
routingKey (Common)	コンシューマーキューをエクスチェンジにバインドするときに使用するルーティングキー。プロデューサールーティングキーの場合は、ヘッダー rabbitmq.ROUTING_KEY を設定します。		String
skipExchangeDeclare (Common)	これは、交換ではなくキューを宣言する必要がある場合に使用できます。	false	boolean
skipQueueBind (Common)	true の場合、キューは宣言後にエクスチェンジにバインドされません	false	boolean
skipQueueDeclare (Common)	true の場合、プロデューサーはキューを宣言およびバインドしません。これは、既存のルーティングキーを介してメッセージを送信するために使用できます。	false	boolean
vhost (Common)	チャネルの仮想ホスト	/	String
autoAck (consumer)	メッセージを自動確認する必要がある場合	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	ブローカーから消費するときの同時コンシューマーの数。(たとえば、JMS コンポーネントの同じオプションと同様)。	1	int
prefetchCount (consumer)	サーバーが配信するメッセージの最大数。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchEnabled (consumer)	RabbitMQConsumer 側でサービス品質を有効にします。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean
prefetchGlobal (consumer)	各コンシューマーではなく、チャネル全体に設定を適用する場合 prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります	false	boolean

名前	説明	デフォルト	タイプ
prefetchSize (consumer)	サーバーが配信するコンテンツの最大量 (オクテットで測定)。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
threadPoolSize (consumer)	コンシューマーは、固定数のスレッドで Thread Pool Executor を使用します。この設定により、そのスレッド数を設定できます。	10	int
allowNullHeaders (producer)	null 値をヘッダーに渡すことを許可する	false	boolean
bridgeEndpoint (producer)	bridgeEndpoint が true の場合、プロデューサーは rabbitmq.EXCHANGE_NAME と rabbitmq.ROUTING_KEY のメッセージヘッダーを無視します。	false	boolean
channelPoolMaxSize (producer)	プールで開かれているチャネルの最大数を取得する	10	int
channelPoolMaxWait (producer)	プールからのチャネルを待機する最大ミリ秒数を設定します	1000	long
guaranteedDeliveries (producer)	true の場合、メッセージを配信できず (basic.return)、メッセージが必須としてマークされている場合に例外が出力されます。この場合、PublisherAcknowledgement もアクティブ化されます。パブリッシャーの確認も参照してください - メッセージはいつ確認されますか。	false	boolean

名前	説明	デフォルト	タイプ
immediate (producer)	このフラグは、メッセージをすぐにキューコンシューマーにルーティングできない場合の対応方法をサーバーに指示します。このフラグが設定されている場合、サーバーは Return メソッドで配信不能メッセージを返します。このフラグがゼロの場合、サーバーはメッセージをキューに入れますが、メッセージが消費されるという保証はありません。ヘッダーが rabbitmq.IMMEDIATE である場合、このオプションはオーバーライドされます。	false	boolean
mandatory (producer)	このフラグは、メッセージをキューにルーティングできない場合の対応方法をサーバーに指示します。このフラグが設定されている場合、サーバーは Return メソッドでルーティング不可能なメッセージを返します。このフラグがゼロの場合、サーバーはメッセージを通知せずにドロップします。ヘッダーが rabbitmq.MANDATORY である場合、このオプションはオーバーライドされます。	false	boolean
publisherAcknowledgements (producer)	true の場合、メッセージはパブリッシャーの確認をオンにしてパブリッシュされます。	false	boolean
publisherAcknowledgementsTimeout (producer)	RabbitMQ サーバーからの basic.ack 応答を待機する時間 (ミリ秒)		long
args (advanced)	さまざまな RabbitMQ の概念を設定するための引数を指定します。それぞれに異なる接頭辞が必要です: Exchange: arg.exchange。Queue: arg.queue.Binding: arg.binding.たとえば、メッセージ ttl 引数でキューを宣言するには: http://localhost:5672/exchange/queueeargs=arg.queue.x-message-ttl=60000		Map
automaticRecoveryEnabled (advanced)	接続の自動回復を有効にします (接続のシャットダウンがアプリケーションによって開始されない場合に自動回復を実行する接続の実装を使用します)		Boolean
bindingArgs (advanced)	**非推奨** declare=true の場合にキューバインディングパラメーターを設定するためのキー/値引数		Map
clientProperties (advanced)	接続クライアントプロパティー (サーバーとのネゴシエーションで使用されるクライアント情報)		Map
connectionTimeout (advanced)	Connection timeout	60000	int

名前	説明	デフォルト	タイプ
exchangeArgs (advanced)	**非推奨** declare=true の場合にエクスチェンジパラメーターを設定するためのキー/値引数		Map
exchangeArgsConfigurer (advanced)	**非推奨** Channel.exchangeDeclare でエクスチェンジ引数を設定するためのコンフィギュアラーを設定します		ArgsConfigurer
networkRecoveryInterval (advanced)	ミリ秒単位のネットワーク回復間隔 (ネットワーク障害からの回復時に使用される間隔)	5000	Integer
queueArgs (advanced)	**非推奨** declare=true の場合にキューパラメーターを設定するためのキー/値引数		Map
queueArgsConfigurer (advanced)	**非推奨** Channel.queueDeclare でキュー引数を設定するためのコンフィギュアラーを設定します		ArgsConfigurer
requestedChannelMax (advanced)	接続要求されたチャネルの最大数 (提供されるチャネルの最大数)	0	int
requestedFrameMax (advanced)	接続要求フレーム最大 (提供されるフレームの最大サイズ)	0	int
requestedHeartbeat (advanced)	接続要求されたハートビート (提供される秒単位のハートビート)	60	int
requestTimeout (advanced)	InOut Exchange パターン使用時の応答待ちタイムアウトを設定する (ミリ秒単位) 。	20000	long
requestTimeoutChecker Interval (advanced)	inOut エクスチェンジの requestTimeoutCheckerInterval を設定する	1000	long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
topologyRecoveryEnabled (advanced)	接続トポロジーの回復を有効にします (トポロジーの回復を実行する必要があります)。		Boolean
transferException (advanced)	true の場合、コンシューマー側で inOut Exchange が失敗し、原因となった例外がレスポンスで返されます。	false	boolean

名前	説明	デフォルト	タイプ
password (security)	認証アクセス用のパスワード	guest	String
sslProtocol (security)	接続時に SSL を有効にします。受け入れられる値は true、TLS および SSLv3 です。		String
trustManager (security)	SSL トラストマネージャーを設定します。このオプションを有効にするには、SSL を有効にする必要があります。		TrustManager
username (security)	認証アクセスの場合のユーザー名	guest	文字列

接続オプションの詳細については、http://www.rabbitmq.com/releases/rabbitmq-java-client/current-javadoc/com/rabbitmq/client/ConnectionFactory.html および AMQP 仕様を参照してください。

256.3. 接続ファクトリーの使用

RabbitMQ に接続するには、次のようなログインの詳細を使用して **ConnectionFactory** (JMS と同じ) をセットアップできます。

```
<bean id="rabbitConnectionFactory" class="com.rabbitmq.client.ConnectionFactory">
  <property name="host" value="localhost"/>
  <property name="port" value="5672"/>
  <property name="username" value="camel"/>
  <property name="password" value="bugsbunny"/>
</bean>

And then refer to the connection factory in the endpoint uri as shown below:

<camelContext>
  <route>
    <from uri="direct:rabbitMQEx2"/>
    <to uri="rabbitmq:ex2?connectionFactory=#rabbitConnectionFactory"/>
  </route>
</camelContext>
```

Camel 2.21 以降、**ConnectionFactory** はデフォルトで自動検出されるため、次のことができます。

```
<camelContext>
  <route>
    <from uri="direct:rabbitMQEx2"/>
    <to uri="rabbitmq:ex2"/>
  </route>
</camelContext>
```

256.4. メッセージヘッダー

次のヘッダーは、メッセージを消費するときにエクスチェンジに設定されます。

プロパティー	値
rabbitmq.ROUTING_KEY	メッセージの受信に使用されたルーティングキー、またはメッセージの生成時に使用されるルーティングキー
rabbitmq.EXCHANGE_NAME	メッセージを受信したエクスチェンジ
rabbitmq.DELIVERY_TAG	受信メッセージの rabbitmq 配信タグ
rabbitmq.REDELIVERY_TAG	メッセージが再配信されたかどうか
rabbitmq.REQUEUE	Camel 2.14.2: これは、メッセージの拒否を制御するためにコンシューマーによって使用されます。コンシューマーがエクスチェンジの処理を完了し、エクスチェンジが失敗した場合、コンシューマーは RabbitMQ ブローカーからのメッセージを拒否します。このヘッダーの値は、この動作を制御します。値が false の場合 (デフォルト)、メッセージは破棄/配信不能になります。値が true の場合、メッセージは再キューイングされます。

次のヘッダーはプロデューサーによって使用されます。これらが camel エクスチェンジで設定されている場合、RabbitMQ メッセージで設定されます。

プロパティー	値
rabbitmq.ROUTING_KEY	メッセージの送信時に使用されるルーティングキー

プロパティー	値
rabbitmq.EXCHANGE_NAME	メッセージを受信したエクスチェンジ
rabbitmq.EXCHANGE_OVERRIDE_NAME	Camel 2.21: プロデューサーでエンドポイントが設定された名前の代わりに、このエクスチェンジにメッセージを強制的に送信するために使用されます
rabbitmq.CONTENT_TYPE	RabbitMQ メッセージに設定する contentType
rabbitmq.PRIORITY	RabbitMQ メッセージに設定する優先ヘッダー
rabbitmq.CORRELATIONID	RabbitMQ メッセージに設定する correlationId
rabbitmq.MESSAGE_ID	RabbitMQ メッセージに設定するメッセージ ID
rabbitmq.DELIVERY_MODE	メッセージを永続化するかどうか
rabbitmq.USERID	RabbitMQ メッセージに設定する userId

プロパティー	値
rabbitmq.CLUSTERID	RabbitMQ メッセージに設定する clusterId
rabbitmq.REPLY_TO	RabbitMQ メッセージに設定する replyTo
rabbitmq.CONTENT_ENCODING	RabbitMQ メッセージに設定する contentEncoding
rabbitmq.TYPE	RabbitMQ メッセージに設定するタイプ
rabbitmq.EXPIRATION	RabbitMQ メッセージに設定する有効期限
rabbitmq.TIMESTAMP	RabbitMQ メッセージに設定するタイムスタンプ
rabbitmq.APP_ID	RabbitMQ メッセージに設定する appId

メッセージが受信されると、ヘッダーはコンシューマーによって設定されます。プロデューサーは、エクスチェンジが行われると、ダウンストリームプロセッサーのヘッダーも設定します。プロデューサーが設定する本番前に設定されたヘッダーはオーバーライドされます。

256.5. メッセージボディー

このコンポーネントは、ボディーで camel エクスチェンジを rabbit mq メッセージボディーとして使用します。オブジェクトの camel exchange は、バイト配列に変換可能でなければなりません。そうしないと、プロデューサーはサポートされていないボディータイプの例外を出力します。

256.6. サンプル

ルーティングキー B を持つエクスチェンジ A にバインドされたキューからメッセージを受信するには以下のようにします。

```
from("rabbitmq:A?routingKey=B")
```

自動応答が無効になっている単一のスレッドでキューからメッセージを受信するには以下のようにします。

```
from("rabbitmq:A?routingKey=B&threadPoolSize=1&autoAck=false")
```

C というエクスチェンジにメッセージを送信するには以下のようにします。

```
to("rabbitmq:C")
```

ヘッダーエクスチェンジとキューの宣言

```
from("rabbitmq:ex?exchangeType=headers&queue=q&bindingArgs=#bindArgs")
```

bindArgs という ID の **Map<String, Object>** をレジストリーに配置します。

たとえば、Spring にメソッドを宣言します

```
@Bean(name="bindArgs")
public Map<String, Object> bindArgsBuilder() {
    return Collections.singletonMap("foo", "bar");
}
```

256.6.1. エクスチェンジ間のルーティングの問題 (Camel 2.20.x 以前)

たとえば、次の例に示すように、ある Rabbit エクスチェンジから別の Rabbit エクスチェンジにメッセージをルーティングしたい場合は、foo → bar とします。

```
from("rabbitmq:foo")
  .to("rabbitmq:bar")
```

次に、Camel がメッセージを自分自身にルーティングすることに注意してください (例: foo → foo)。では、それはなぜでしょうか。これは、メッセージを受信するコンシューマー (送信元など) が、メッセージヘッダーの **rabbitmq.EXCHANGE_NAME** にエクスチェンジの名前 (**foo** など) を提供するためです。Camel プロデューサーがメッセージを **bar** に送信する場合、ヘッダー **rabbitmq.EXCHANGE_NAME** はこれをオーバーライドし、代わりにメッセージを **foo** に送信します。

これを回避するには、次のいずれかを行う必要があります。

- ヘッダーを削除します。

```
from("rabbitmq:foo")
  .removeHeader("rabbitmq.EXCHANGE_NAME")
  .to("rabbitmq:bar")
```

- または、プロデューサーで **bridgeEndpoint** モードをオンにします。

```
from("rabbitmq:foo")
  .to("rabbitmq:bar?bridgeEndpoint=true")
```

Camel 2.21 以降ではこれが改善され、エクスチェンジ間で簡単にルーティングできるようになりました。ヘッダー **rabbitmq.EXCHANGE_NAME** は、宛先エクスチェンジをオーバーライドするためにプロデューサーによって使用されなくなりました。代わりに、新しいヘッダー **rabbitmq.EXCHANGE_OVERRIDE_NAME** を使用して、別のエクスチェンジに送信できます。たとえば、cheese エクスチェンジに送信するには、次のことができます

```
from("rabbitmq:foo")
  .setHeader("rabbitmq.EXCHANGE_OVERRIDE_NAME", constant("cheese"))
  .to("rabbitmq:bar")
```

第257章 REACTIVE STREAMS コンポーネント

Camel バージョン 2.19 以降で利用可能

react-streams: コンポーネントを使用すると、リアクティブストリーム 標準と互換性のあるリアクティブストリーム処理ライブラリーとメッセージをエクスチェンジできます。

このコンポーネントはバックプレッシャーをサポートし、Reactive Stream Technology Compatibility Kit (TCK) を 使用してテストされています。

Camel モジュールは、ユーザーが Camel ルート内で入力および出力ストリームを定義できるようにする **reactive-streams** コンポーネントと、Camel エンドポイントを任意の外部リアクティブフレームワークに直接使用できるようにするダイレクトクライアント API を提供します。

Camel は、リアクティブストリーム **Publisher** および **Subscriber** の内部実装を使用するため、特定のフレームワークに関連付けられていません。統合テストでは、次のリアクティブフレームワークが使用されています: Reactor Core 3、RxJava 2。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactive-streams</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

257.1. URI 形式

```
reactive-streams://stream?[options]
```

stream は、外部ストリーム処理システムへの BIND Camel ルートに使用される論理ストリーム名です。

257.2. オプション

Reactive Streams コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
internalEngineConfiguration (advanced)	Reactive Streams の内部エンジンを設定します。		ReactiveStreamsEngine 設定
backpressureStrategy (producer)	イベントを遅いサブスクライバーにプッシュするときに使用するバックプレッシャーストラテジー。	BUFFER	ReactiveStreamsBackpressureStrategy

名前	説明	デフォルト	タイプ
serviceType (advanced)	使用する基礎となるリアクティブストリームの実装のタイプを設定します。実装はレジストリーから、または ServiceLoader を使用して検索されます。デフォルトの実装は DefaultCamelReactiveStreamsService です。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Reactive Streams エンドポイントは、URI 構文を使用して設定されます。

```
reactive-streams:stream
```

パスおよびクエリーパラメーターを使用します。

257.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
stream	エンドポイントがメッセージをエクスチェンジするために使用するストリームチャネルの名前。		String

257.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	Camel ルートでエクスチェンジを処理するために使用されるスレッドの数。	1	int

名前	説明	デフォルト	タイプ
exchangesRefillLowWatermark (consumer)	maxInflightExchanges のパーセンテージとして、リクエストされたエクスチェンジの下限をアクティブなサブスクリプションに設定します。アップストリームソースからの保留中のアイテムの数がウォーターマークより少ない場合、新しいアイテムをサブスクリプションにリクエストできます。0 に設定すると、サブスクライバーは、前のバッチのすべてのアイテムが処理された後でのみ、maxInflightExchanges のバッチでアイテムをリクエストします。1 に設定すると、サブスクライバーは、エクスチェンジが処理されるたびに新しいアイテムをリクエストできます (chatty)。任意の中間値を使用できます。	0.25	double
forwardOnComplete (consumer)	onComplete イベントを Camel ルートにプッシュするかどうかを決定します。	false	boolean
forwardOnError (consumer)	onError イベントを Camel ルートにプッシュするかどうかを決定します。例外はメッセージボディーとして設定されます。	false	boolean
maxInflightExchanges (consumer)	Camel によって同時に処理されるエクスチェンジの最大数。このパラメーターは、ストリームのバックプレッシャーを制御します。正でない値を設定すると、バックプレッシャーが無効になります。	128	Integer
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
backpressureStrategy (producer)	イベントを遅いスサブスクライバーにプッシュするときに使用するバックプレッシャーストラテジー。		ReactiveStreamsBackpressureStrategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

257.3. 使用方法

このライブラリーは、アプリケーションが Camel データと対話するために必要なすべての通信モードをサポートすることを目的としています。

- **Get** Camel ルートからのデータを取得 (Camel からの入力のみ)
- **Send** Camel ルートにデータを送信 (Camel への入力のみ)
- **Request** Camel ルートへの変換をリクエスト (Camel への入力のみ)
- **Process** リアクティブ処理ステップを使用して Camel ルートから流れるデータを処理 (Camel からの入出力)

257.4. CAMEL からのデータの取得

Camel ルートから流れるデータをサブスクライブするには、次のスニペットのように、エクスチェンジを名前付きストリームにリダイレクトする必要があります。

```
from("timer:clock")
.setBody().header(Exchange.TIMER_COUNTER)
.to("reactive-streams:numbers");
```

ルートは、XML DSL を使用して記述することもできます。

この例では、数字の無制限のストリームが名前 **numbers** に関連付けられています。ストリームには、**CamelReactiveStreams** ユーティリティークラスを使用してアクセスできます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Getting a stream of exchanges
Publisher<Exchange> exchanges = camel.fromStream("numbers");

// Getting a stream of Integers (using Camel standard conversion system)
Publisher<Integer> numbers = camel.fromStream("numbers", Integer.class);
```

ストリームは、リアクティブストリームと互換性のあるライブラリーで簡単に使用できます。これを RxJava 2 で使用する方法の例を次に示します (ただし、イベントの処理には任意のリアクティブフレームワークを使用できます)。

```
Flowable.fromPublisher(integers)
    .doOnNext(System.out::println)
    .subscribe();
```

この例では、Camel によって生成されたすべての数値を **System.out** に出力します。

257.4.1. ダイレクト API を使用した Camel からのデータの取得

短い Camel ルートの場合、および (Camel DSL をまったく使用せずに) リアクティブフレームワークの関数構造を使用して処理フロー全体を定義することを好むユーザーの場合、Camel URI を使用してストリームを定義することもできます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Get a stream from all the files in a directory
Publisher<String> files = camel.from("file:folder", String.class);
```

```
// Use the stream in RxJava2
Flowable.fromPublisher(files)
    .doOnNext(System.out::println)
    .subscribe();
```

257.5. CAMEL へのデータの送信

外部ライブラリーがイベントを Camel ルートにプッシュする必要がある場合、リアクティブストリームエンドポイントをコンシューマーとして設定する必要があります。

```
from("reactive-streams:elements")
.to("log:INFO");
```

elements ストリームへのハンドルは、**CamelReactiveStreams** ユーティリティークラスから取得できます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

Subscriber<String> elements = camel.streamSubscriber("elements", String.class);
```

サブスクライバーは、**elements** ストリームから消費する Camel ルートにイベントをプッシュするために使用できます。

これを RxJava 2 で使用する方法の例を次に示します (ただし、イベントを発行するために任意のリアクティブフレームワークを使用できます)。

```
Flowable.interval(1, TimeUnit.SECONDS)
    .map(i -> "Item " + i)
    .subscribe(elements);
```

この例では、文字列アイテムが RxJava によって毎秒生成され、上記で定義された Camel ルートにプッシュされます。

257.5.1. ダイレクト API を使用した Camel へのデータの送信

この場合も、ダイレクト API を使用して、エンドポイント URI から Camel サブスクライバーを取得できます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Send two strings to the "seda:queue" endpoint
Flowable.just("hello", "world")
    .subscribe(camel.subscriber("seda:queue", String.class));
```

257.6. CAMEL への変換のリクエスト

一部の Camel DSL で定義されたルートは、特定の変換を実行するためにリアクティブストリームフレームワーク内で使用できます (同じメカニズムを使用して、たとえば、**http** エンドポイントにデータを送信して続行することもできます)。

次のスニペットは、RxJava 機能コードがファイルをロードして Camel にマーシャリングするタスクを要求する方法を示しています。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Process files starting from their names
Flowable.just(new File("file1.txt"), new File("file2.txt"))
    .flatMap(file -> camel.toStream("readAndMarshal", String.class))
    // Camel output will be converted to String
    // other steps
    .subscribe();
```

これを機能させるには、次のようなルートを Camel コンテキストで定義する必要があります。

```
from("reactive-streams:readAndMarshal")
.marshal() // ... other details
```

257.6.1. ダイレクト API を使用した Camel への変換のリクエスト

別のアプローチは、URI エンドポイントをリアクティブフローで直接使用することです。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Process files starting from their names
Flowable.just(new File("file1.txt"), new File("file2.txt"))
    .flatMap(file -> camel.to("direct:process", String.class))
    // Camel output will be converted to String
    // other steps
    .subscribe();
```

toStream の代わりに **to()** メソッドを使用する場合、reactive-streams: エンドポイントを使用してルートを定義する必要はありません (内部では使用されますが)。

この場合、Camel 変換は次のようになります。

```
from("direct:process")
.marshal() // ... other details
```

257.7. CAMEL データをリアクティブフレームワークに処理する

リアクティブストリーム **パブリッシャー** は一方向のデータエクスチェンジを可能にしますが、Camel ルートはインアウトエクスチェンジパターンを使用することがよくあります (たとえば、REST エンドポイントを定義し、一般に、各リクエストに対してレスポンスが必要な場合)。

このような状況では、ユーザーはリアクティブ処理ステップをフローに追加して、Camel ルートを強化したり、リアクティブフレームワークを使用して変換全体を定義したりできます。

たとえば、次のルートがあるとします。

```
from("timer:clock")
.setBody().header(Exchange.TIMER_COUNTER)
.to("direct:reactive")
.log("Continue with Camel route... n=${body}");
```

リアクティブ処理ステップは、direct:reactive エンドポイントに関連付けることができます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

camel.process("direct:reactive", Integer.class, items ->
    Flowable.fromPublisher(items) // RxJava2
        .map(n -> -n)); // make every number negative
```

Camel ルートを流れるデータは、外部のリアクティブフレームワークによって処理され、Camel 内で処理フローが続行されます。

このメカニズムは、完全にリアクティブな方法で In-Out エクスチェンジを定義するためにも使用できます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// requires a rest-capable Camel component
camel.process("rest:get:orders", exchange ->
                Flowable.fromPublisher(exchange)
                    .flatMap(ex -> allOrders())); // retrieve orders asynchronously
```

詳細については、Camel の例 (**camel-example-reactive-streams**) を参照してください。

257.8. 高度なトピック

257.8.1. バックプレッシャーの制御 (プロデューサー側)

Camel エクスチェンジを外部サブスクライバーにルーティングする場合、バックプレッシャーは、エクスチェンジを配信する前にキャッシュする内部バッファーによって処理されます。サブスクライバーがエクスチェンジレートよりも遅い場合、バッファーが大きくなりすぎる可能性があります。多くの場合、これは避ける必要があります。

次のルートを検討します。

```
from("jms:queue")
.to("reactive-streams:flow");
```

JMS キューに多数のメッセージが含まれており、**flow** ストリームに関連付けられているサブスクライバーが遅すぎる場合、メッセージは JMS からデキューされてバッファーに追加されるため、out of memory エラーが発生する可能性があります。このような問題を回避するために、**ThrottlingInflightRoutePolicy** をルートに設定できます。

```
ThrottlingInflightRoutePolicy policy = new ThrottlingInflightRoutePolicy();
policy.setMaxInflightExchanges(10);

from("jms:queue")
.routePolicy(policy)
.to("reactive-streams:flow");
```

このポリシーは、アクティブなエクスチェンジの最大数 (およびバッファーの最大サイズ) を制限し、しきい値 (例では **10**) よりも低く保ちます。**10** を超えるメッセージが処理中の場合、ルートは中断され、サブスクライバーがそれらを処理するのを待ちます。

このメカニズムにより、サブスクライバーは、バックプレッシャを通じてルートの一時停止/再開を自動的に制御します。複数のサブスクライバーが同じストリームからアイテムを消費している場合、最も遅いサブスクライバーがルートステータスを自動的に制御します。

他の状況では、例えば。**http** コンシューマーを使用する場合、ルートの一時停止により http サービスが使用できなくなるため、デフォルトの設定 (ポリシーなし、無制限のバッファー) を使用することをお勧めします。ユーザーは、http サービスへの要求の数を制限する (例: スケールアウト) ことによって、メモリーの問題を回避するようにしてください。

一定量のデータ損失が許容されるコンテキストでは、**BUFFER** 以外のバックプレッシャストラテジーを設定することが、高速なソースを処理するためのソリューションになる可能性があります。

```
from("direct:thermostat")
.to("reactive-streams:flow?backpressureStrategy=LATEST");
```

LATEST バックプレッシャーストラテジーが使用される場合、ルートから受信した最後のエクスチェンジのみがパブリッシャーによって保持され、古いデータは破棄されます (他のオプションが利用可能です)。

257.8.2. バックプレッシャーの制御 (コンシューマー側)

Camel がリアクティブストリームパブリッシャーからアイテムを消費する場合、インフライトエクスチェンジの最大数をエンドポイントオプションとして設定できます。

コンシューマーに関連付けられたサブスクライバーは、パブリッシャーと対話して、ルート内のメッセージ数をしきい値より低く保ちます。

バックプレッシャ対応ルートの例:

```
from("reactive-streams:numbers?maxInflightExchanges=10")
.to("direct:endpoint");
```

Camel がソースパブリッシャーに (リアクティブストリームバックプレッシャーメカニズムを介して) リクエストするアイテムの数は、常に **10** 未満です。メッセージは、Camel 側の単一のスレッドによって処理されます。

同時コンシューマー (スレッド) の数は、エンドポイントオプション (**concurrentConsumers**) として設定することもできます。1 つのコンシューマー (デフォルト) を使用する場合、ソースストリーム内の項目の順序が維持されます。この値を大きくすると、アイテムは複数のスレッドによって同時に処理されます (したがって、順序は保持されません)。

257.9. CAMEL REACTIVE STREAMS スターター

spring-boot ユーザーは、スターターモジュールを利用できます。スターターを使用する場合、**CamelReactiveStreamsService** を直接 Spring コンポーネントに注入できます。

スターターを使用するには、Spring Boot pom.xml ファイルに以下を追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactive-streams-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>
```

257.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第258章 リアクターコンポーネント

Camel バージョン 2.20 以降で利用可能

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactor</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第259章 REF コンポーネント

Camel バージョン 1.2 以降で利用可能

ref: コンポーネントは、レジストリーにバインドされている既存のエンドポイントの検索に使用されます。

259.1. URI 形式

```
ref:someName[?options]
```

someName は、 レジストリー (常にではありませんが、通常は Spring レジストリー) 内のエンドポイントの名前です。Spring レジストリーを使用している場合、**someName は** Spring レジストリー内のエンドポイントの Bean ID になります。

259.2. REF オプション

Ref コンポーネントにはオプションがありません。

Ref エンドポイントは、URI 構文を使用して設定されます。

```
ref:name
```

パスおよびクエリーパラメーターを使用します。

259.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** レジストリーで検索するエンドポイントの名前。		String

259.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

259.3. ランタイムルックアップ

このコンポーネントは、実行時に URI を計算できるレジストリーでエンドポイントを動的に検出する必要がある場合に使用できます。次に、次のコードを使用してエンドポイントを検索できます。

```
// lookup the endpoint
String myEndpointRef = "bigspenderOrder";
Endpoint endpoint = context.getEndpoint("ref:" + myEndpointRef);

Producer producer = endpoint.createProducer();
Exchange exchange = producer.createExchange();
exchange.getIn().setBody(payloadToSend);
// send the exchange
producer.process(exchange);
```

また、次のようなレジストリーで定義されたエンドポイントのリストを作成できます。

```
<camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <endpoint id="normalOrder" uri="activemq:order.slow"/>
    <endpoint id="bigspenderOrder" uri="activemq:order.high"/>
</camelContext>
```

259.4. 例

以下のサンプルでは、URI で **ref:** を使用して、SpringID **endpoint2** を持つエンドポイントを参照します。

もちろん、代わりに **ref** 属性を使用することもできます。

```
<to ref="endpoint2"/>
```

より一般的な書き方です。

第260章 REST コンポーネント

Camel バージョン 2.14 以降で利用可能

rest コンポーネントを使用すると、Rest DSL を使用して REST エンドポイント (consumer) を定義して、REST トランスポートとして他の Camel コンポーネントにプラグインできます。

Camel 2.18 以降では、残りのコンポーネントをクライアント (プロデューサー) として使用して REST サービスを呼び出すこともできます。

260.1. URI 形式

```
rest://method:path[:uriTemplate]?[options]
```

260.2. URI オプション

REST コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
componentName (Common)	restlet、spark-rest などの REST トランスポートに使用する Camel Rest コンポーネント。コンポーネントが明示的に設定されていない場合、Rest DSL と統合する Camel コンポーネントがあるか、または org.apache.camel.spi.RestConsumerFactory (コンシューマー) または org.apache.camel.spi.RestProducerFactory (プロデューサー) がレジストリーに登録されていルカを Camel は検索します。いずれかが見つかった場合は、それが使用されています。		String
apiDoc (プロデューサー)	使用する swagger api doc リソース。リソースはデフォルトでクラスパスからロードされ、JSon 形式である必要があります。		String
host (プロデューサー)	使用する HTTP サービスのホストとポート (swagger スキーマでホストをオーバーライドします)		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

REST エンドポイントは、URI 構文を使用して設定されます。

```
rest:method:path:uriTemplate
```

パスおよびクエリーパラメーターを使用します。

260.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
メソッド	使用する **必須** の HTTP メソッド。		String
path	**必須** ベースパス		String
uriTemplate	uri テンプレート		String

260.2.2. クエリーパラメーター(15 個のパラメーター):

名前	説明	デフォルト	タイプ
componentName (Common)	restlet、spark-rest などの REST トランスポートに使用する Camel Rest コンポーネント。コンポーネントが明示的に設定されていない場合、Rest DSL と統合する Camel コンポーネントがある場合、または org.apache.camel.spi.RestConsumerFactory がレジストリーに登録されている場合、Camel はルックアップします。いずれかが見つかった場合は、それが使用されています。		String
consumes (Common)	この REST サービスが受け入れる text/xml または application/json などのメディアタイプ。デフォルトでは、すべての種類のタイプを受け入れます。		String
inType (Common)	着信 POJO バインディングタイプを FQN クラス名として宣言します		String
outType (Common)	発信 POJO バインディングタイプを FQN クラス名として宣言します		String
produces (Common)	この REST サービスが返す text/xml または application/json などのメディアタイプ。		String
routeId (Common)	この REST サービスが作成するルートの名前		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
description (consumer)	この REST サービスを文書化する人間による説明		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
apiDoc (プロデューサー)	使用する swagger api doc リソース。リソースはデフォルトでクラスパスからロードされ、JSon 形式である必要があります。		String
bindingMode (producer)	プロデューサーのバインディングモードを設定します。off 以外に設定されている場合、プロデューサは着信メッセージの本文を inType から json または xml に変換し、json または xml からの応答を outType に変換しようとします。		RestBindingMode
host (プロデューサー)	使用する HTTP サービスのホストとポート (swagger スキーマでホストをオーバーライドします)		文字列
queryParameters (producer)	呼び出す HTTP サービスのクエリーパラメーター		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

260.3. サポートされている残りのコンポーネント

次のコンポーネントは、残りのコンシューマー (Rest DSL) をサポートしています。

- camel-coap
- camel-netty-http
- camel-netty4-http
- camel-jetty
- camel-restlet
- camel-servlet
- camel-spark-rest
- camel-undertow

次のコンポーネントは、rest プロデューサーをサポートしています。

- camel-http
- camel-http4
- camel-netty4-http
- camel-jetty
- camel-restlet
- camel-undertow

260.4. パスと URITEMPLATE の構文

path および uriTemplate オプションは、パラメーターのサポートを使用して REST コンテキストパスを定義する REST 構文を使用して定義されます。

ヒント: uriTemplate が設定されていない場合、パスオプションは同じように機能します。パスのみを設定するか、両方のオプションを設定するかは問題ではありません。パスと uriTemplate の両方を設定することは、REST ではより一般的な方法ですが。

以下は、パスのみを使用した Camel ルートです。

```
from("rest:get:hello")
  .transform().constant("Bye World");
```

次のルートでは、キー me を持つ Camel ヘッダーにマップされたパラメーターを使用します。

```
from("rest:get:hello/{me}")
  .transform().simple("Bye ${header.me}");
```

次の例では、基本パスを hello として設定し、uriTemplates を使用して 2 つの REST サービスを設定しています。

```
from("rest:get:hello:/{me}")
  .transform().simple("Hi ${header.me}");
```

```
from("rest:get:hello:/french/{me}")
  .transform().simple("Bonjour ${header.me}");
```

260.5. REST プロデューサーの例

rest コンポーネントを使用して、他の Camel コンポーネントと同様に REST サービスを呼び出すことができます。

たとえば、**hello/{me}** を使用して REST サービスを呼び出すには、次のようにします。

```
from("direct:start")
  .to("rest:get:hello/{me}");
```

そして、動的な値 **{me}** が同じ名前の Camel メッセージにマップされます。したがって、この REST サービスを呼び出すには、次のように空のメッセージ本文とヘッダーを送信できます。

```
template.sendBodyAndHeader("direct:start", null, "me", "Donald Duck");
```

Rest プロデューサーは、次のようにホストオプションを使用して設定できる REST サービスのホスト名とポートを認識する必要があります。

```
from("direct:start")
  .to("rest:get:hello/{me}?host=myserver:8080/foo");
```

host オプションを使用する代わりに、次のように **restConfiguration** でホストを設定できます。

```
restConfiguration().host("myserver:8080/foo");

from("direct:start")
  .to("rest:get:hello/{me}");
```

ProducerComponent を使用して、HTTP クライアントとして使用する Camel コンポーネントを選択できます。たとえば、http4 を使用するには、次のようにします。

```
restConfiguration().host("myserver:8080/foo").producerComponent("http4");

from("direct:start")
  .to("rest:get:hello/{me}");
```

260.6. REST プロデューサーバーインディング

REST プロデューサーは、rest-dsl と同様に JSON または XML を使用したバインディングをサポートします。

たとえば、json バインディングモードをオンにして jetty を使用するには、残りの構成でこれを設定できます。

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBindingMode.json
);

from("direct:start")
  .to("rest:post:user");
```

次に、rest プロデューサーを使用して REST サービスを呼び出すと、REST サービスを呼び出す前に、すべての POJO が json に自動的にバインドされます。

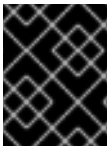
```
UserPojo user = new UserPojo();
user.setId(123);
user.setName("Donald Duck");

template.sendBody("direct:start", user);
```

上記の例では、POJO インスタンス **UserPojo** をメッセージ本文として送信します。また、REST 設定で JSON バインディングをオンにしたため、REST サービスを呼び出す前に POJO が POJO から JSON にマーシャリングされます。

ただし、応答メッセージ (たとえば、REST サービスが応答として送り返すもの) のバインドも実行する場合は、**outType** オプションを設定して、JSon から POJO に非整列化する POJO のクラス名を指定する必要があります。

たとえば、REST サービスが **com.foo.MyResponsePojo** にバインドする JSON ペイロードを返す場合、次のように設定できます。

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBindingMode.json
);

  from("direct:start")
    .to("rest:post:user?outType=com.foo.MyResponsePojo");
```



重要

REST サービスの呼び出しから受信した応答メッセージに対して POJO バインディングを行う場合は、**outType** オプションを設定する必要があります。

260.7. その他の例

より多くの例を提供する Rest DSL と、Rest DSL を使用してより良い RESTful な方法でそれらを定義する方法を参照してください。

Apache Camel ディストリビューションには **camel-example-servlet-rest-tomcat** の例があり、Apache Tomcat または同様の Web コンテナーにデプロイできるトランスポートとして SERVLET で Rest DSL を使用する方法を示しています。

260.8. 関連項目

- REST DSL
- SERVLET

第261章 REST SWAGGER コンポーネント

Camel バージョン 2.19 以降で利用可能

rest-swagger は、Swagger (Open API) 仕様ドキュメントから残りのプロデューサーを設定し、**RestProducerFactory** インターフェイスを実装するコンポーネントに委譲します。現在既知の動作コンポーネントは次のとおりです。

- http
- http4
- netty4-http
- restlet
- jetty
- undertow

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rest-swagger</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

261.1. URI 形式

```
rest-swagger:[specificationPath#]operationId
```

ここで、**operationId** は Swagger 仕様の操作の ID であり、 **specificationPath** は仕様へのパスです。**specificationPath** が指定されていない場合、デフォルトで **swagger.json** になります。検索メカニズムは、Camels **ResourceHelper** を使用してリソースをロードします。つまり、CLASSPATH リソース (**classpath:my-specification.json**)、ファイル (**file:/some/path.json**)、Web (**http://api.example.com/swagger.json**) を使用できます。または Bean を参照する (**ref:nameOfBean**) か、Bean のメソッド (**bean:nameOfBean.methodName**) を使用して仕様リソースを取得し、その Swagger 独自のリソース読み込みサポートに失敗します。

このコンポーネントは HTTP クライアントとしては機能せず、上記の別のコンポーネントに委任します。検索メカニズムは、**RestProducerFactory** インターフェイスを実装する単一のコンポーネントを検索し、それを使用します。CLASSPATH に複数のパスが含まれている場合は、プロパティー **componentName** を設定して、委譲先のコンポーネントを示す必要があります。

設定の大部分は Swagger 仕様から取得されますが、コンポーネントまたはエンドポイントで指定することによってそれらをオーバーライドするオプションが存在します。通常、**host** または **basePath** が仕様と異なる場合は、オーバーライドする必要があります。

注記

host パラメーターには、スキーム、ホスト名、およびポート番号を含む絶対 URI を含める必要があります。たとえば、**https://api.example.com** です。

componentName を使用して、リクエストを実行するために使用するコンポーネントを指定します。この名前付きコンポーネントは Camel コンテキストに存在し、必要な **RestProducerFactory** インターフェイスを実装する必要があります – 上部にリストされているコンポーネントも同様です。

コンポーネントレベルでもエンドポイントレベルでも **componentName** を指定しない場合、CLASSPATH で適切なデリゲートが検索されます。これが機能するためには、**RestProducerFactory** インターフェイスを実装する CLASSPATH に 1 つのコンポーネントのみが存在する必要があります。

このコンポーネントのエンドポイント URI は寛大です。つまり、メッセージヘッダーに加えて、REST 操作のパラメーターをエンドポイントパラメーターとして指定できます。これらは、後続のすべての呼び出しで一定であるため、すべての呼び出しで実際に一定であるパラメーターに対してのみこの機能を使用するのが理にかなっています。たとえば、**/api/7.2/users/{id}** などのパスの API バージョンです。

261.2. オプション

REST Swagger コンポーネントは、以下に示す 7 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
basePath (producer)	/v2 などの API ベースパス。設定が Swagger 仕様に存在する値をオーバーライドする場合、デフォルトは未設定です。		String
componentName (producer)	リクエストを実行する Camel コンポーネントの名前。コンポーネントは Camel レジストリーに存在する必要があり、RestProducerFactory サービスプロバイダーインターフェイスを実装する必要があります。設定されていない場合、RestProducerFactory SPI を実装する単一のコンポーネントの CLASSPATH が検索されます。エンドポイント設定で上書きできます。		String
consumes (producer)	このコンポーネントが消費できるペイロードの種類。application/json のように 1 つのタイプにすることも、application/json、application/xml のように複数のタイプにすることもできます。RFC7231 によると、q=0.5 です。これは Accept HTTP ヘッダーの値に相当します。設定すると、Swagger 仕様で見つかった値がオーバーライドされます。エンドポイント設定で上書きできます		String
host (producer)	https://hostname:port の形式で HTTP リクエストを送信するスキームのホスト名とポート。エンドポイント、コンポーネント、または Camel コンテキストの対応する REST 設定で設定できます。このコンポーネントに名前 (petstore など) を付けると、最初に REST 設定が参照され、次に rest-swagger が参照され、最後にグローバル設定が参照されます。設定すると、Swagger 仕様で見つかったあらゆる値を上書きする、RestConfiguration。エンドポイント設定で上書きできます。		String

名前	説明	デフォルト	タイプ
produces (producer)	このコンポーネントが生成するペイロードタイプ。たとえば、RFC7231 に従った application/json です。これは、Content-Type HTTP ヘッダーの値に相当します。設定すると、Swagger 仕様に存在するすべての値がオーバーライドされます。エンドポイント設定で上書きできます。		String
specificationUri (producer)	Swagger 仕様ファイルへのパス。スキーム、ホスト、ベースパスはこの仕様から取得されますが、これらはコンポーネントまたはエンドポイントレベルのプロパティーでオーバーライドできます。指定されていない場合、コンポーネントは swagger.json リソースを読み込もうとします。このコンポーネントのコンポーネントとエンドポイントで定義されたホストには、スキーマ、ホスト名、およびオプションで URI 構文のポートが含まれている必要があることに注意してください (つまり、https://api.example.com:8080)。エンドポイント設定で上書きできます。	swagger.json	URI
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

REST Swagger エンドポイントは、URI 構文を使用して設定されます。

```
rest-swagger:specificationUri#operationId
```

パスおよびクエリーパラメーターを使用します。

261.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
specificationUri	Swagger 仕様ファイルへのパス。スキーム、ホスト、ベースパスはこの仕様から取得されますが、これらはコンポーネントまたはエンドポイントレベルのプロパティーでオーバーライドできます。指定されていない場合、コンポーネントは swagger.json リソースを読み込もうとします。このコンポーネントのコンポーネントとエンドポイントで定義されたホストには、スキーマ、ホスト名、およびオプションで URI 構文のポートが含まれている必要があることに注意してください (つまり、https://api.example.com:8080)。コンポーネント設定をオーバーライドします。	swagger.json	URI
operationId	**必須** Swagger 仕様からのオペレーションの ID。		String

261.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
basePath (producer)	/v2 などの API ベースパス。設定が Swagger 仕様およびコンポーネント設定に存在する値をオーバーライドする場合、デフォルトは未設定です。		String
componentName (producer)	リクエストを実行する Camel コンポーネントの名前。コンポーネントは Camel レジストリーに存在する必要があり、RestProducerFactory サービスプロバイダーインターフェイスを実装する必要があります。設定されていない場合、RestProducerFactory SPI を実装する単一のコンポーネントの CLASSPATH が検索されます。コンポーネント設定をオーバーライドします。		String
consumes (producer)	このコンポーネントが消費できるペイロードの種類。application/json のように 1 つのタイプにすることも、application/json、application/xml のように複数のタイプにすることもできます。RFC7231 によると、q=0.5 です。これは Accept HTTP ヘッダーの値に相当します。設定すると、Swagger 仕様およびコンポーネント構成で見つかったすべての値をオーバーライドします。		String

名前	説明	デフォルト	タイプ
host (producer)	https://hostname:port の形式で HTTP リクエストを送信するスキームのホスト名とポート。エンドポイント、コンポーネント、または Camel コンテキストの対応する REST 設定で設定できます。このコンポーネントに名前 (petstore など) を付けると、最初に REST 設定が参照され、次に rest-swagger が参照され、最後にグローバル設定が参照されます。設定すると、Swagger 仕様で見つかったあらゆる値を上書きする、RestConfiguration。他のすべての設定をオーバーライドします。		String
produces (producer)	このコンポーネントが生成するペイロードタイプ。たとえば、RFC7231 に従った application/json です。これは、Content-Type HTTP ヘッダーの値に相当します。設定すると、Swagger 仕様に存在するすべての値がオーバーライドされます。他のすべての設定をオーバーライドします。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

261.3. 例: ペットストア

examples ディレクトリーにある **camel-example-rest-swagger** プロジェクトの例を確認してください。

たとえば、**PetStore** が提供する REST API を使用する場合は、Swagger 仕様から仕様 URI と目的の操作 ID を参照するか、仕様をダウンロードして CLASSPATH の (ルートに) **swagger.json** として保存し、自動的に使用されるようにします。.Undertow コンポーネントを使用してすべてのリクエストを実行し、Camels は Spring Boot の優れたサポートを提供します。

Maven POM ファイルで定義されている依存関係は次のとおりです。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-undertow-starter</artifactId>
</dependency>

<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rest-swagger-starter</artifactId>
</dependency>
```

Undertow コンポーネントと **RestSwaggerComponent** を定義することから始めます。

```
@Bean
public Component petstore(CamelContext camelContext, UndertowComponent undertow) {
    RestSwaggerComponent petstore = new RestSwaggerComponent(camelContext);
```

```
    petstore.setSpecificationUri("http://petstore.swagger.io/v2/swagger.json");
    petstore.setDelegate(undertow);

    return petstore;
}
```

注記

Camel での Spring Boot のサポートにより、**UndertowComponent** Spring Bean が自動作成され、接頭辞 **camel.component.undertow** を使用して **application.properties** (または **application.yml**) を使用して設定できます。.ここで **petstore** コンポーネントを定義しているのは、Camel コンテキストで PetStore REST API との対話に使用できる名前のコンポーネントを持つためで、これが唯一の **rest-swagger** コンポーネントであれば、同じ方法で設定することができます (**application.properties** を使用)。

これで、アプリケーションで **ProducerTemplate** を使用して PetStore REST メソッドを呼び出すことができます。

```
@Autowired
ProducerTemplate template;

String getPetJsonById(int petId) {
    return template.requestBodyAndHeaders("petstore:getPetById", null, "petId", petId);
}
```

第262章 RESTLET コンポーネント

Camel バージョン 2.0 以降で利用可能

Restlet コンポーネントは、RESTful リソースを消費および生成するための Restlet ベースのエンドポイントを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-restlet</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

262.1. URI 形式

```
restlet:restletUrl[?options]
```

restletUrl のフォーマット:

```
protocol://hostname[:port][/resourcePattern]
```

Restlet は、プロトコルとアプリケーションの問題の分離を促進します。Restlet エンジン のリファレンス実装は、多くのプロトコルをサポートしています。ただし、HTTP プロトコルのみをテストしました。デフォルトのポートはポート 80 です。プロトコルに基づいてデフォルトポートを自動的に切り替えることはまだありません。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

情報: Restlet はヘッダーを理解する際に大文字と小文字を区別しているようです。たとえば、コンテンツタイプを使用するには Content-Type を使用し、場所には Location などを使用します。

警告

Camel 2.14.0 および 2.14.1 で camel-restlet のパフォーマンスが低下したという報告を受けました。これについては、issue 996 で Restlet チームに報告しました。この問題を解決するには、Camel 2.14.2 以降、エンドポイント uris のオプションとして synchronous=true を設定するか、グローバルオプションとして RestletComponent に設定して、すべてのエンドポイントがこのオプションを継承するようにします。

262.2. オプション

Restlet コンポーネントは、以下に示す 22 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
controllerDaemon (consumer)	コントローラースレッドがデーモンである必要があるかどうかを示します (JVM の終了をブロックしません)。		Boolean
controllerSleepTimeMs (consumer)	コントローラースレッドが各コントロール間でスリープする時間。		Integer
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
inboundBufferSize (consumer)	メッセージを読み取るときのバッファーのサイズ。		Integer
maxConnectionsPerHost (common)	ホスト (IP アドレス) あたりの同時接続の最大数。		Integer
maxThreads (consumer)	リクエストを処理する最大スレッド。		Integer
lowThreads (consumer)	コネクターが過負荷と見なされるタイミングを決定するワーカースレッドの数。		Integer
maxTotalConnections (common)	同時接続の合計最大数。		Integer
minThreads (consumer)	リクエストを処理するために待機しているスレッドの最小数。		Integer
outboundBufferSize (consumer)	メッセージを書き込むときのバッファーのサイズ。		Integer
persistingConnections (consumer)	通話後に接続を維持する必要があるかどうかを示します。		Boolean
pipeliningConnections (consumer)	パイプライン接続がサポートされているかどうかを示します。		Boolean
threadMaxIdleTimeMs (consumer)	アイドル状態のスレッドが収集される前に操作を待機する時間。		Integer

名前	説明	デフォルト	タイプ
useForwardedForHeader (consumer)	一般的なプロキシーとキャッシュでサポートされている X-Forwarded-For ヘッダーを検索し、それを使用して Request.getClientAddresses() メソッドの結果を入力します。この情報は、ローカルネットワーク内の中間コンポーネントに対してのみ安全です。他のアドレスは、偽のヘッダーを設定することで簡単に変更できるため、重大なセキュリティーチェックで信頼するべきではありません。		Boolean
reuseAddress (consumer)	SO_REUSEADDR ソケットオプションを有効/無効にします。詳細については、java.io.ServerSocketreuseAddress プロパティーを参照してください。		Boolean
maxQueued (consumer)	サービスに使用できるワーカースレッドがない場合にキューに入れることができる呼び出しの最大数。値が 0 の場合、キューは使用されず、ワーカースレッドがすぐに利用できない場合、呼び出しは拒否されます。値が -1 の場合、無制限のキューが使用され、呼び出しが拒否されることはありません。		Integer
disableStreamCache (consumer)	Restlet からの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel は Restlet 入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultRestletBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。	false	boolean
port (consumer)	restlet コンシューマールートのポート番号を設定するには。これにより、これを一度設定して、これらのコンシューマーに同じポートを再利用できます。		int
synchronous (producer)	プロデューサに同期 Restlet クライアントを使用するかどうか。このオプションを true に設定すると、Restlet 同期クライアントのほうがうまく機能するように見えるため、パフォーマンスが向上する可能性があります。		Boolean

名前	説明	デフォルト	タイプ
enabledConverters (advanced)	完全なクラス名または単純なクラス名として有効にするコンバーターのリスト。空または null の場合、自動的に登録されたすべてのコンバーターが有効になります		List
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Restlet エンドポイントは、URI 構文を使用して設定されます。

```
restlet:protocol:host:port/uriPattern
```

パスおよびクエリーパラメーターを使用します。

262.2.1. パスパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
protocol	**必須** 使用するプロトコル (http または https)		String
host	**必須** restlet サービスのホスト名		String
port	**必須** restlet サービスのポート番号	80	int
uriPattern	/customer/id などのリソースパターン		String

262.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
restletMethod (common)	プロデューサーエンドポイントで、使用するリクエストメソッドを指定します。コンシューマーエンドポイントで、エンドポイントが restletMethod リクエストのみを消費することを指定します。	GET	メソッド

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
restletMethods (consumer)	コンマで区切られた 1 つ以上のメソッドを指定します (restletMethods=post,put など)。restlet コンシューマーエンドポイントによって処理されます。restletMethod と restletMethods の両方のオプションが指定されている場合、restletMethod の設定は無視されます。可能なメソッドは次のとおりです: ALL、CONNECT、DELETE、GET、HEAD、OPTIONS、PATCH、POST、PUT、TRACE		String
disableStreamCache (consumer)	Restlet からの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel は Restlet 入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultRestletBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
restletUriPatterns (consumer)	**非推奨** Camel レジストリーのリストを参照する表記を使用して、restlet コンシューマーエンドポイントによってサービスされる 1 つ以上の URI テンプレートを指定します。URI パターンがエンドポイント URI で定義されている場合、エンドポイントで定義された URI パターンと restletUriPatterns オプションの両方が受け入れられます。		List
connectTimeout (producer)	接続がタイムアウトの場合、クライアントは接続を断念します。無制限の待機の場合は 0 です。	30000	int
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
socketTimeout (producer)	クライアントソケットの受信タイムアウト。無制限の待機の場合は 0。	30000	int
throwExceptionOnFailure (producer)	プロデューサーの失敗時に例外を出力するかどうか。このオプションが false の場合、http ステータスコードは、エラー値があるかどうかを確認できるメッセージヘッダーとして設定されます。	true	boolean
autoCloseStream (producer)	restlet プロデューサーを使用した REST サービスの呼び出しからの応答として、ストリーム表現を自動的に閉じるかどうか。応答がストリーミングで、オプション streamRepresentation が有効になっている場合は、ストリーミング応答から InputStream を自動的に閉じて、Camel Exchange のルーティングが完了したときに入力ストリームが閉じられるようにすることができます。ただし、Camel ルートの外でストリームを読み取る必要がある場合は、ストリームを自動的に閉じる必要がない場合があります。	false	boolean
streamRepresentation (producer)	restlet プロデューサーを使用した REST サービスの呼び出しからの応答としてストリーム表現をサポートするかどうか。応答がストリーミングの場合、このオプションを有効にして、Camel Message 本文のメッセージ本文として java.io.InputStream を使用できます。このオプションを使用する場合、autoCloseStream オプションも有効にして、Camel Exchange のルーティングが完了したときに入力ストリームが確実に閉じられるようにすることができます。ただし、Camel ルートの外でストリームを読み取る必要がある場合は、ストリームを自動的に閉じる必要がない場合があります。	false	boolean
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
restletBinding (advanced)	カスタム RestletBinding を使用して、Restlet と Camel メッセージをバインドします。		RestletBinding
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
restletRealm (security)	restlet のセキュリティーレルムをマップとして設定します。		Map
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

262.3. メッセージヘッダー

名前	タイプ	説明
Content-Type	**String**	アプリケーション/プロセッサーによって OUT メッセージに設定できるコンテンツタイプを指定します。値は、レスポンスメッセージの **content-type** です。このヘッダーが設定されていない場合、コンテンツタイプは OUT メッセージボディーのオブジェクトタイプに基づきます。Camel 2.3 以降では、Content-Type ヘッダーが Camel IN メッセージで指定されている場合、ヘッダーの値によって Restlet リクエストメッセージのコンテンツタイプが決定されます。 それ以外の場合は、デフォルトで application/x-www-form-urlencoded になります。リリース 2.3 より前では、リクエストのコンテンツタイプのデフォルトを変更することはできません。
CamelAcceptContentType	**String**	**Camel 2.9.3、2.10.0 以降:** HTTP Accept リクエストヘッダー。
CamelHttpMethod	**String**	HTTP リクエストメソッド。これは、IN メッセージヘッダーで設定されます。
CamelHttpQuery	**String**	リクエスト URI のクエリー文字列。restlet コンポーネントがリクエストを受信すると、**DefaultRestletBinding** によって IN メッセージに設定されます。
CamelHttpResponseCode	**String** または **Integer**	応答コードは、アプリケーション/プロセッサーによって OUT メッセージに設定できます。値は、レスポンスメッセージのレスポンスコードです。このヘッダーが設定されていない場合、応答コードは restlet ランタイムエンジンによって設定されます。

名前	タイプ	説明
CamelHttpUri	**String**	HTTP リクエスト URI。これは、IN メッセージヘッダーで設定されます。
CamelRestletLogin	**String**	Basic 認証のログイン名。アプリケーションによって IN メッセージに設定され、Camel によって restlet リクエストヘッダーの前にフィルターされます。
CamelRestletPassword	**String**	Basic 認証のパスワード名。アプリケーションによって IN メッセージに設定され、Camel によって restlet リクエストヘッダーの前にフィルターされます。
CamelRestletRequest	**Request**	Camel 2.8: すべてのリクエストの詳細を保持する **org.restlet.Request** オブジェクト。
CamelRestletResponse	**応答**	Camel 2.8: **org.restlet.Response** オブジェクト。これを使用して、Restlet の API を使用して応答を作成できます。以下の例を参照してください。
org.restlet.*		Camel IN ヘッダーに伝播される Restlet メッセージの属性。
cache-control	**String** または **List<CacheDirective>**	Camel 2.11: ユーザーは camel メッセージヘッダーから文字列値または Restlet の CacheDirective のリストを使用してキャッシュコントロールを設定できます。

262.4. メッセージボディー

Camel は、外部サーバーからの restlet レスポンスを OUT ボディーに格納します。ヘッダーはルーティング中に保持されるように、IN メッセージのすべてのヘッダーが OUT メッセージにコピーされます。

262.5. サンプル

262.5.1. 認証付き Restlet エンドポイント

次のルートは、http://localhost:8080 で **POST** リクエストをリッスンする **restlet** コンシューマーエンドポイントを開始します。プロセッサーは、リクエストの本文と **id** ヘッダーの値をエコーするレスポンスを作成します。

URI クエリーの **restletRealm** 設定は、レジストリーでレルムマップを検索するために使用されます。

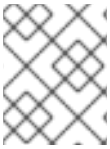
このオプションが指定されている場合、restlet コンシューマーはこの情報を使用してユーザーのログインを認証します。**認証された** リクエストのみがリソースにアクセスできます。このサンプルでは、レジストリーとして機能する Spring アプリケーションコンテキストを作成します。Realm Map の Bean ID は **restletRealmRef** と一致する必要があります。

次のサンプルでは、http://localhost:8080 上のサーバーにリクエストを送信する **direct** エンドポイント (つまり、restlet コンシューマーエンドポイント) を開始します。

必要なのはそれだけです。リクエストを送信して、restlet コンポーネントを試す準備ができました。

サンプルクライアントは、次のヘッダーを含むリクエストを **direct:start-auth** エンドポイントに送信します。

- **CamelRestletLogin** (Camel によって内部的に使用されます)
- **CamelRestletPassword** (Camel によって内部的に使用されます)
- **id** (アプリケーションヘッダー)



注記

org.apache.camel.restlet.auth.login および **org.apache.camel.restlet.auth.password** は、Restlet ヘッダーとして伝播されません。

サンプルクライアントは、次のようなレスポンスを受け取ります。

```
received [<order foo='1'/>] as an order id = 89531
```

262.5.2. 複数のメソッドと URI テンプレートを提供する単一の restlet エンドポイント (非推奨)

この機能は **推奨されていない** ため、使用しないでください。

restletMethods オプションを使用して、複数の HTTP メソッドにサービスを提供する単一のルートを作成することができます。このスニペットは、ヘッダーからリクエストメソッドを取得する方法も示しています。

複数のメソッドを提供することに加えて、次のスニペットは、**restletUriPatterns** オプションを使用して複数の URI テンプレートをサポートするエンドポイントを作成する方法を示しています。リクエスト URI は、IN メッセージのヘッダーでも使用できます。URI パターンがエンドポイント URI で定義されている場合 (このサンプルではそうではありません)、エンドポイントで定義された URI パターンと **restletUriPatterns** オプションの両方が受け入れられます。

restletUriPatterns=#uriTemplates オプションは、Spring XML 設定で定義された **List<String>** Bean を参照します。

```
<util:list id="uriTemplates">
   <value>/users/{username}</value>
   <value>/atom/collection/{id}/component/{cid}</value>
</util:list>
```

262.5.3. Restlet API を使用して応答を設定する

Camel 2.8 から利用可能

org.restlet.Response API を使用して応答を入力することをお勧めします。これにより、Restlet API に完全にアクセスし、応答をきめ細かく制御できます。インライン Camel プロセッサーからの応答を生成する以下のルートスニペットを参照してください。

Restlet Response API を使用したレスポンスの生成

262.5.4. コンポーネントの最大スレッドの設定

最大スレッドオプションを設定するには、次のようにコンポーネントでこれを行う必要があります。

```
<bean id="restlet" class="org.apache.camel.component.restlet.RestletComponent">
  <property name="maxThreads" value="100"/>
</bean>
```

262.5.5. webapp 内で Restlet サーブレットを使用する

Camel 2.8 以降で利用可能
サーブレットコンテナー内で Restlet アプリケーションを設定するには 3 つの方法 があり、サブクラス化された SpringServerServlet を使用すると、Restlet コンポーネントを注入することで Camel 内で設定できます。

サーブレットコンテナ内で Restlet サーブレットを使用すると、URI の相対パスでルートを構成し (ハードコードされた絶対 URI の制約を取り除く)、ホストするサーブレットコンテナが (新しいポートで別のサーバープロセスを生成する必要がなく) 入力リクエストを処理することができます。

設定するには、camel-context.xml に以下を追加します。

```
<camelContext>
  <route id="RS_RestletDemo">
    <from uri="restlet:/demo/{id}" />
    <transform>
      <simple>Request type : ${header.CamelHttpMethod} and ID : ${header.id}</simple>
    </transform>
  </route>
</camelContext>

<bean id="RestletComponent" class="org.restlet.Component" />

<bean id="RestletComponentService"
class="org.apache.camel.component.restlet.RestletComponent">
  <constructor-arg index="0">
    <ref bean="RestletComponent" />
  </constructor-arg>
</bean>
```

これを web.xml に追加します。

```
<!-- Restlet Servlet -->
<servlet>
  <servlet-name>RestletServlet</servlet-name>
  <servlet-class>org.restlet.ext.spring.SpringServerServlet</servlet-class>
  <init-param>
    <param-name>org.restlet.component</param-name>
    <param-value>RestletComponent</param-value>
```

```
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>RestletServlet</servlet-name>
  <url-pattern>/rs/*</url-pattern>
</servlet-mapping>
```

その後、**http://localhost:8080/mywebapp/rs/demo/1234** でデプロイされたルートにアクセスできるようになります。

localhost:8080 は、サーブレットコンテナーのサーバーとポートです。
mywebapp は、デプロイされた webapp の名前です
ブラウザーには次のコンテンツが表示されます。

```
"Request type : GET and ID : 1234"
```

Maven pom.xml ファイルで行うことができる restlet への Spring 拡張機能への依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.restlet.jee</groupId>
  <artifactId>org.restlet.ext.spring</artifactId>
  <version>${restlet-version}</version>
</dependency>
```

また、restlet maven リポジトリーにも依存関係を追加する必要があります。

```
<repository>
   <id>maven-restlet</id>
   <name>Public online Restlet repository</name>
   <url>http://maven.restlet.org</url>
</repository>
```

第263章 RIBBON コンポーネント

Camel バージョン 2.18 以降で利用可能

Ribbon コンポーネントは、クライアント側の負荷分散に Netflix リボンを使用できるようにします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ribbon</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントは、ServiceCall EIP を使用するときに、クライアント側で負荷分散機能を適用するのに役立ちます。

263.1. 設定

- **プログラマティック**

```
RibbonConfiguration configuration = new RibbonConfiguration();
configuration.addProperties("ServerListRefreshInterval", "250");

RibbonLoadBalancer loadBalancer = new RibbonLoadBalancer(configuration);

from("direct:start")
    .serviceCall()
        .name("myService")
        .loadBalancer(loadBalancer)
        .consulServiceDiscovery()
        .end()
    .to("mock:result");
```

- **Spring Boot**

  application.properties

```
camel.cloud.ribbon.properties[ServerListRefreshInterval] = 250
```

  routes

```
from("direct:start")
    .serviceCall()
        .name("myService")
        .ribbonLoadBalancer()
        .consulServiceDiscovery()
        .end()
    .to("mock:result");
```

- **XML**

```
<route>
    <from uri="direct:start"/>
    <serviceCall name="myService">
        <!-- enable ribbon load balancer -->
        <ribbonLoadBalancer>
            <properties key="ServerListRefreshInterval" value="250"/>
        </ribbonLoadBalancer>
    </serviceCall>
</route>
```

263.2. 関連項目

- ServiceCall EIP

第264章 RMI コンポーネント

Camel バージョン 1.0 以降で利用可能

rmi: コンポーネントは、Exchange を RMI プロトコル (JRMP) にバインドします。

このバインディングは RMI を使用しているだけなので、呼び出すことができるメソッドに関しては通常の RMI ルールが引き続き適用されます。このコンポーネントは、Remote インターフェイスを拡張するインターフェイスからメソッド呼び出しを実行する Exchange のみをサポートします。メソッド内のすべてのパラメーターは、Serializable または **Remote** オブジェクトのいずれかである必要があります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rmi</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

264.1. URI 形式

```
rmi://rmi-regisitry-host:rmi-registry-port/registry-path[?options]
```

以下に例を示します。

```
rmi://localhost:1099/path/to/service
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

264.2. オプション

RMI コンポーネントにはオプションがありません。

RMI エンドポイントは、URI 構文を使用して設定されます。

```
rmi:hostname:port/name
```

パスおよびクエリーパラメーターを使用します。

264.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
hostname	RMI サーバーのホスト名	localhost	String
name	**必須** RMI サーバーにバインドするときに使用する名前		String

名前	説明	デフォルト	タイプ
port	RMI サーバーのポート番号	1099	int

264.2.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
method (Common)	呼び出すメソッドの名前を設定できます。		String
remoteInterfaces (common)	リモートインターフェイスを指定します。		List
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

264.3. 使用

RMI レジストリーに登録されている既存の RMI サービスを呼び出すには、次のようなルートを作成します。

```
from("pojo:foo").to("rmi://localhost:1099/foo");
```

RMI レジストリー内の既存の camel プロセッサーまたはサービスを BIND するには、次のように RMI エンドポイントを定義します。

-

```
RmiEndpoint endpoint= (RmiEndpoint) endpoint("rmi://localhost:1099/bar");
endpoint.setRemoteInterfaces(ISay.class);
from(endpoint).to("pojo:bar");
```

RMI コンシューマーエンドポイントをバインドするときは、公開する **Remote** インターフェイスを指定する必要があることに注意してください。

XML DSL では、**Camel 2.7** 以降では次のようにできます。

```
<camel:route>
    <from uri="rmi://localhost:37541/helloServiceBean?
remoteInterfaces=org.apache.camel.example.osgi.HelloService"/>
    <to uri="bean:helloServiceBean"/>
</camel:route>
```

264.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第265章 ROUTEBOX コンポーネント (非推奨)

Camel バージョン 2.6 以降で利用可能

変更の対象となるルートボックス

routebox コンポーネントを使用すると、自動作成またはユーザーが挿入した camel コンテキストでホストされる camel ルートのコレクションに、カプセル化とストラテジーベースの間接サービスを提供する特殊なエンドポイントを作成できます。

Routebox エンドポイントは、camel ルートで直接呼び出すことができる camel エンドポイントです。routebox エンドポイントは、次の主要な機能を実行します

- カプセル化 - ブラックボックスとして機能し、内部 camel コンテキストに格納された camel ルートのコレクションをホストします。内部コンテキストは完全に routebox コンポーネントの制御下にあり、**JVM にバインドされています**。
- ストラテジーベースの間接化 - ユーザー定義の内部ルーティングストラテジーまたはディスパッチマップに基づいて、camel ルートに沿ってルートボックスエンドポイントに送信されたペイロードを特定の内部ルートに直接送信します。
- エクスチェンジ伝播 - ルートボックスエンドポイントによって変更されたエクスチェンジを camel ルートの次のセグメントに転送します。

routebox コンポーネントは、コンシューマーエンドポイントとプロデューサーエンドポイントの両方をサポートします。

プロデューサーエンドポイントには 2 つの種類があります

- 外部のルートボックスコンシューマーエンドポイントに入力リクエストを送信またはディスパッチするプロデューサー
- 内部に埋め込まれた camel コンテキストでルートを直接呼び出すプロデューサーは、外部のコンシューマーにリクエストを送信しません。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-routebox</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

265.1. CAMEL ROUTEBOX エンドポイントの必要性

routebox コンポーネントは、複雑な環境での統合を容易にするように設計されています。

- ルートの大規模なコレクションと
- さまざまな方法での統合を必要とする幅広いエンドポイントテクノロジーを含む

このような環境では、camel ルートを効果的に整理し、重ね合わせることで統合ソリューションを構築することが必要になることがあります。

- 粗粒度または高レベルのルート - 統合の重点領域を表す Routebox エンドポイントとして公開される内部または低レベルのルートの集約コレクション。以下に例を示します。

注目される点	きめ細かいルートの例
部門フォーカス	人材ルート、販売ルートなど
サプライチェーンと B2B フォーカス	配送ルート、フルフィルメントルート、サードパーティーサービスなど
テクノロジーフォーカス	データベースルート、JMS ルート、スケジュールされたバッチルートなど

- きめ細かいルート - 単一の特定のビジネスおよび/または統合パターンを実行するルート。

きめ細かいルートの Routebox エンドポイントに送信されたリクエストは、リクエストを内部のきめ細かいルートに委任して、特定の統合目標を達成し、最終的な内部結果を収集して、きめ細かいルートに沿って次のステップに進むことができます。

265.2. URI 形式

```
routebox:routeboxname[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

265.3. オプション

RouteBox コンポーネントにはオプションがありません。

RouteBox エンドポイントは、URI 構文を使用して設定されます。

```
routebox:routeboxName
```

パスおよびクエリーパラメーターを使用します。

265.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
routeboxName	***必須*** routebox の論理名 (キュー名など)		String

265.3.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
dispatchMap (common)	タイプ HashMap のオブジェクト値に一致する Camel レジストリーのキーを表す文字列。HashMap キーには、交換ヘッダー ROUTE_DISPATCH_KEY に設定された値と照合できる文字列が含まれている必要があります。HashMap 値には、リクエストの送信先となる内部ルートコンシューマー URI が含まれている必要があります。		Map
dispatchStrategy (common)	カスタム RouteboxDispatchStrategy を使用し、デフォルトの代わりにカスタムディスパッチを使用する場合。		RouteboxDispatchストラテジー
forkContext (common)	同じ CamelContext を再利用する代わりに、新しい内部 CamelContext を fork して作成するかどうか。	true	boolean
innerProtocol (common)	Routebox コンポーネントによって内部的に使用されるプロトコル。Direct または SEDA を指定できます。Routebox コンポーネントは現在、JVM にバインドされたプロトコルを提供しています。	直接的な	String
queueSize (Common)	リクエストを受信する固定サイズのキューを作成します。		int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollInterval (consumer)	seda からのポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	long
threads (consumer)	リクエストを受信するためにルートボックスが使用するスレッドの数。	20	int

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
connectionTimeout (producer)	メッセージの送信時にプロデューサによって使用されるミリ秒単位のタイムアウト。	20000	long
sendToConsumer (producer)	プロデューサーエンドポイントが外部ルートボックスコンシューマーにリクエストを送信するかどうかを決定します。設定が false の場合、Producer は埋め込みの内部コンテキストを作成し、リクエストを内部で処理します。	true	boolean
innerContext (advanced)	タイプ org.apache.camel.CamelContext のオブジェクト値に一致する Camel レジストリーのキーを表す文字列。CamelContext がユーザーによって提供されない場合、内部ルートのデプロイのために CamelContext が自動的に作成されます。		CamelContext
innerProducerTemplate (advanced)	内部に埋め込まれた CamelContext によって使用される ProducerTemplate。		ProducerTemplate
innerRegistry (advanced)	内部に埋め込まれた CamelContext にカスタムレジストリーを使用する場合。		Registry
routeBuilders (advanced)	リスト型のオブジェクト値に一致する Camel レジストリーのキーを表す文字列。ユーザーが内部ルートで事前準備された innerContext を提供しない場合、内部ルートを含む RouteBuilders の空でないリストとして routeBuilders オプションを提供する必要があります。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

265.4. ルートボックスとのメッセージの送受信

リクエストを送信する前に、以下に示すように必要な URI パラメーターをレジストリーにロードして、ルートボックスを適切に設定する必要があります。Spring の場合、必要な Bean が正しく宣言されていれば、レジストリーは Camel によって自動的に入力されます。

265.4.1. ステップ 1: 内部ルートの詳細をレジストリーにロードする

```
@Override
protected JndiRegistry createRegistry() throws Exception {
    JndiRegistry registry = new JndiRegistry(createJndiContext());

    // Wire the routeDefinitions & dispatchStrategy to the outer camelContext where the routebox is
declared
    List<RouteBuilder> routes = new ArrayList<RouteBuilder>();
    routes.add(new SimpleRouteBuilder());
    registry.bind("registry", createInnerRegistry());
    registry.bind("routes", routes);

    // Wire a dispatch map to registry
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("addToCatalog", "seda:addToCatalog");
    map.put("findBook", "seda:findBook");
    registry.bind("map", map);

    // Alternatively wiring a dispatch strategy to the registry
    registry.bind("strategy", new SimpleRouteDispatchStrategy());

    return registry;
}

private JndiRegistry createInnerRegistry() throws Exception {
    JndiRegistry innerRegistry = new JndiRegistry(createJndiContext());
    BookCatalog catalogBean = new BookCatalog();
    innerRegistry.bind("library", catalogBean);

    return innerRegistry;
}
...
CamelContext context = new DefaultCamelContext(createRegistry());
```

265.4.2. ステップ 2: オプションで Dispatch Map の代わりに Dispatch Strategy を使用する

ディスパッチストラテジーを使用するには、以下の例に示すように、インターフェイス **org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy** を実装する必要があります。

```
public class SimpleRouteDispatchStrategy implements RouteboxDispatchStrategy {

    /* (non-Javadoc)
     * @see
org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy#selectDestinationUri(java.u
il.List, org.apache.camel.Exchange)
     */
    public URI selectDestinationUri(List<URI> activeDestinations,
            Exchange exchange) {
        URI dispatchDestination = null;

        String operation = exchange.getIn().getHeader("ROUTE_DISPATCH_KEY", String.class);
        for (URI destination : activeDestinations) {
```

```
            if (destination.toASCIIString().equalsIgnoreCase("seda:" + operation)) {
                dispatchDestination = destination;
                break;
            }
        }

        return dispatchDestination;
    }
}
```

265.4.3. ステップ 2: routebox コンシューマーの起動

ルートコンシューマーを作成するときは、routeboxUri の # エントリーが、作成された内部レジストリー、routebuilder リスト、および CamelContext レジストリーの dispatchStrategy/dispatchMap と一致することに注意してください。すべてのルートビルダーと関連するルートは、ルートボックスで作成された内部コンテキストで起動されることに注意してください

```
private String routeboxUri = "routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map";

public void testRouteboxRequests() throws Exception {
    CamelContext context = createCamelContext();
    template = new DefaultProducerTemplate(context);
    template.start();

    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from(routeboxUri)
                .to("log:Routes operation performed?showAll=true");
        }
    });
    context.start();

    // Now use the ProducerTemplate to send the request to the routebox
    template.requestBodyAndHeader(routeboxUri, book, "ROUTE_DISPATCH_KEY",
"addToCatalog");
}
```

265.4.4. ステップ 3: routebox プロデューサーを使用する

リクエストをルートボックスに送信する場合、プロデューサーは内部ルートエンドポイント URI を知る必要はなく、以下に示すように、ディスパッチストラテジーまたは dispatchMap を使用してルートボックス URI エンドポイントを呼び出すだけで済みます。

リクエストが正しい内部ルートに送信されるように、**ROUTE_DISPATCH_KEY** (Dispatch Strategy のオプション) と呼ばれる特別なエクチェンジヘッダーをディスパッチマップのキーと一致するキーに設定する必要があります。

```
from("direct:sendToStrategyBasedRoutebox")
    .to("routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchStrategy=#strategy")
    .to("log:Routes operation performed?showAll=true");

from ("direct:sendToMapBasedRoutebox")
```

```
    .setHeader("ROUTE_DISPATCH_KEY", constant("addToCatalog"))
    .to("routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map")
    .to("log:Routes operation performed?showAll=true");
```

第266章 RSS コンポーネント

Camel バージョン 2.0 以降で利用可能

rss: コンポーネントは、RSS フィードのポーリングに使用されます。Camel はデフォルトで 60 秒ごとにフィードをポーリングします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rss</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記: コンポーネントは現在、ポーリング (consuming) フィードのみをサポートしています。

注記

Camel-rss は、ServiceMix でホストされている ROME のパッチを 適用したバージョンを 内部的に使用して、OSGi クラスのロードの問題 を解決します。

266.1. URI 形式

```
rss:rssUri
```

rssUri は、ポーリングする RSS フィードへの URI です。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

266.2. オプション

RSS コンポーネントにはオプションがありません。

RSS エンドポイントは、URI 構文を使用して設定されます。

```
rss:feedUri
```

パスおよびクエリーパラメーターを使用します。

266.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
feedUri	**必須** ポーリングするフィードへの URI。		String

266.2.2. クエリーパラメーター (27 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
feedHeader (consumer)	フィードオブジェクトをヘッダーとして追加するかどうかを設定します	true	boolean
filter (consumer)	エントリーのフィルタリングを使用するかどうかを設定します。	true	boolean
lastUpdate (consumer)	Atom フィードからのエントリーのフィルタリングに使用するタイムスタンプを設定します。このオプションは、splitEntries と組み合わせてのみ使用できます。		Date
password (consumer)	HTTP フィードからのポーリング時に Basic 認証に使用するパスワードを設定します		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sortEntries (consumer)	エントリーを公開日順にソートするかどうかを設定します。splitEntries = true の場合のみ機能します。	false	boolean
splitEntries (consumer)	エントリーを個別に送信するか、フィード全体を 1 つのメッセージとして送信するかを設定します	true	boolean
throttleEntries (consumer)	1 回のフィードポーリングで識別されたすべてのエントリーをすぐに配信するかどうかを設定します。true の場合、consumer.delay ごとに 1 つのエントリーのみが処理されます。splitEntries = true の場合にのみ適用されます。	true	boolean
username (consumer)	HTTP フィードからのポーリング時に Basic 認証に使用されるユーザー名を設定します		String

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

266.3. エクスチェンジデータタイプ

Camel は Exchange の In ボディーを ROME **SyndFeed** で初期化します。**splitEntries** フラグの値に応じて、Camel は 1 つの **SyndEntry** を含む **SyndFeed** か **SyndEntrys** の **java.util.List** を返します。

オプション	値	動作
splitEntries	**true**	現在のフィードからの単一のエントリーがエクスチェンジに設定されます。
splitEntries	**false**	現在のフィードからのエントリーのリスト全体がエクスチェンジに設定されます。

266.4. メッセージヘッダー

ヘッダー	説明
CamelRssFeed	**SyncFeed** オブジェクト全体。

266.5. RSS DATAFORMAT

RSS コンポーネントには、String (XML として) と ROME RSS モデルオブジェクト間のエクスチェンジに使用できる RSS データ形式が付属しています。

- marshal = ROME **SyndFeed** から XML **String** へ
- unmarshal = XML **String** から ROME **SyndFeed** へ

RSS データ形式を使用するルートは次のようになります: **from("rss:file:src/test/data/rss20.xml?splitEntries=false&consumer.delay=1000").marshal().rss().to("mock:marshal");**

この機能の目的は、Camel の組み込み式を使用して RSS メッセージを操作できるようにすることです。以下に示すように、XPath 式を使用して RSS メッセージをフィルタリングできます。次の例では、タイトルに Camel を含むエントリーのみがフィルターを通過します。

```
`from("rss:file:src/test/data/rss20.xml?
splitEntries=true&consumer.delay=100").marshal().rss().filter().xpath("//item/title[contains(.,'Camel')]").to
("mock:result");`
```

ヒント

クエリーパラメーター RSS フィードの URL がクエリーパラメーターを使用している場合、このコンポーネントはそれらを解決します。たとえば、フィードが **alt=rss** を使用している場合、次の例は解決されます: **from("rss:http://someserver.com/feeds/posts/default?alt=rss&splitEntries=false&consumer.delay=1000").to("bean:rss");**

266.6. エントリーのフィルタリング

上記のデータ形式のセクションで示したように、XPath を使用してエントリーを除外できます。Camel の Bean Integration を利用して、独自の条件を実装することもできます。たとえば、上記の XPath の例と同等のフィルターは次のようになります。

from("rss:file:src/test/data/rss20.xml?splitEntries=true&consumer.delay=100").filter().method("myFilterBean", "titleContainsCamel").to("mock:result");

このためのカスタム Bean は次のようになります。

```
public static class FilterBean {
      public boolean titleContainsCamel(@Body SyndFeed feed) {
         SyndEntry firstEntry = (SyndEntry) feed.getEntries().get(0);
return firstEntry.getTitle().contains("Camel");
```

266.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Atom

第267章 RSS DATAFORMAT

Camel バージョン 2.1 以降で利用可能

RSS コンポーネントには、String (XML として) と ROME RSS モデルオブジェクト間のエクスチェンジに使用できる RSS データ形式が付属しています。

- marshal = ROME **SyndFeed** から XML **String** へ
- unmarshal = XML **String** から ROME **SyndFeed** へ

これを使用したルートは次のようになります。

この機能の目的は、Camel の素敵な組み込み式を使用して RSS メッセージを操作できるようにすることです。以下に示すように、XPath 式を使用して RSS メッセージをフィルタリングできます。

ヒント

Query parameters RSS フィードの URL がクエリパラメーターを使用している場合、このコンポーネントはクエリパラメーターも理解します。フィードが **alt=rss** を使用している場合、たとえば **from("rss:http://someserver.com/feeds/posts/default?alt=rss&splitEntries=false&consumer.delay=1000").to("bean:rss");** を行うことができます。

267.1. オプション

RSS データ形式は、以下にリストされている 1 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

第268章 SALESFORCE コンポーネント

Camel バージョン 2.12 以降で利用可能

このコンポーネントは、Java DTO を使用して Salesforce と通信するプロデューサーエンドポイントとコンシューマーエンドポイントをサポートします。
これらの DTO を生成する付属の maven プラグイン Camel Salesforce Plugin があります (以下を参照)。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-salesforce</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記

コンポーネントへの貢献を希望する開発者は、README.md ファイルを参照して、統合テストを実行するための環境の開始方法とセットアップ方法を確認するように指示されています。

268.1. SALESFORCE への認証

このコンポーネントは、次の 3 つの OAuth 認証フローをサポートしています。

- OAuth 2.0 ユーザー名とパスワードのフロー
- OAuth 2.0 リフレッシュトークンフロー
- OAuth 2.0 JWT Bearer Token Flow

フローごとに、異なるプロパティーセットを設定する必要があります。

表268.1 認証フローごとに設定するプロパティー

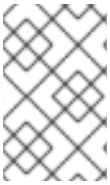
プロパティー	Salesforce のどこで見つけることができますか	フロー
clientId	接続されたアプリケーション、コンシューマーキー	すべてのフロー
clientSecret	コネクテッドアプリ、コンシューマーシークレット	ユーザー名 - パスワード、リフレッシュトークン
userName	Salesforce ユーザーのユーザー名	ユーザー名 - パスワード、JWT ベアラートークン
password	Salesforce ユーザーのパスワード	Username-Password

プロパティー	Salesforce のどこで見つけることができますか	フロー
refreshToken	OAuth フローコールバックから	トークンの更新
keystore	接続アプリケーション、デジタル証明書	JWT ベアラートークン

コンポーネントは、設定しようとしているフローを自動的に判断し、あいまいさを取り除くために **authenticationType** プロパティーを設定します。

注記

本番環境でユーザー名とパスワードのフローを使用することはお勧めしません。



注記

JWT ベアラートークンフローで使用される証明書は、自己署名証明書にすることができます。証明書と秘密鍵を保持する KeyStore には、証明書と秘密鍵のエントリーを 1 つだけ含める必要があります。

268.2. URI 形式

ストリーミングイベントを受信するコンシューマーとして使用する場合、URI スキームは次のようになります。

```
salesforce:topic?options
```

プロデューサとして使用し、Salesforce REST API を呼び出す場合、URI スキームは次のようになります。

```
salesforce:operationName?options
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

268.3. SALESFORCE ヘッダーを渡し、SALESFORCEレスポンスヘッダーを取得する

Camel 2.21 では、受信メッセージヘッダーを介して Salesforce ヘッダー を渡すことがサポートされています。Camel メッセージの **Sforce** または **x-sfdc** で始まるヘッダー名は要求で渡され、**Sforce** で始まるレスポンスヘッダーは送信メッセージヘッダーに存在します。

たとえば、API 制限を取得するには、次のように指定できます。

```
// in your Camel route set the header before Salesforce endpoint
//...
  .setHeader("Sforce-Limit-Info", constant("api-usage"))
  .to("salesforce:getGlobalObjects")
  .to(myProcessor);

// myProcessor will receive `Sforce-Limit-Info` header on the outbound
```

```
// message
class MyProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message in = exchange.getIn();
        String apiLimits = in.getHeader("Sforce-Limit-Info", String.class);
    }
}
```

268.4. サポートされている SALESFORCE API

このコンポーネントは、次の Salesforce API をサポートしています

プロデューサーエンドポイントは、次の API を使用できます。ほとんどの API は一度に 1 つのレコードを処理しますが、Query API は複数のレコードを取得できます。

268.4.1. Rest API

operationName には以下を使用できます。

- getVersions - サポートされている Salesforce REST API バージョンを取得します
- getResources - 利用可能な Salesforce REST Resource エンドポイントを取得します
- getGlobalObjects - 使用可能なすべての SObject タイプのメタデータを取得します
- getBasicInfo - 特定の SObject タイプの基本的なメタデータを取得します
- getDescription - 特定の SObject タイプの包括的なメタデータを取得します
- getSObject - Salesforce ID を使用して SObject を取得します
- createSObject - SObject を作成します
- updateSObject - Id を使用して SObject を更新します
- deleteSObject - Id を使用して SObject を削除します
- getSObjectWithId - 外部 (ユーザー定義) id フィールドを使用して SObject を取得します
- upsertSObject - 外部 ID を使用して SObject を更新または挿入します
- deleteSObjectWithId - 外部 ID を使用して SObject を削除します
- query - Salesforce SOQL クエリーを実行します
- queryMore - クエリー API から返された結果リンクを使用して、より多くの結果を取得します (結果が多数の場合)。
- queryAll - SOQL クエリーを実行します。マージまたは削除のために削除された結果を返します。また、アーカイブされたタスクおよびイベントレコードに関する情報も返します。
- search - Salesforce SOSL クエリーを実行します
- 制限 - 組織 API の使用制限をフェッチする
- recent - 最近のアイテムの取得

- approval - 承認プロセスのために 1 つまたは複数のレコード (バッチ) を送信します
- approvals - すべての承認プロセスのリストを取得します
- composite - 関連する可能性のある最大 25 個の REST リクエストを送信し、個々のレスポンスを受け取ります
- composite-tree - 親子関係 (最大 5 レベル) を持つ最大 200 レコードを一度に作成します
- composite-batch - リクエストの設定をバッチで送信します
- getBlobField - 個々のレコードから指定された BLOB フィールドを取得します。
- apexCall - ユーザー定義の APEX REST API 呼び出しを実行します。

たとえば、次のプロデューサーエンドポイントは upsertSObject API を使用し、sObjectIdName パラメーターで Name を外部 ID フィールドとして指定します。リクエストメッセージの本文は、maven プラグインを使用して生成された SObject DTO である必要があります。レスポンスメッセージは、既存のレコードが更新された場合は **null** になるか、新しいレコードの ID を持つ **CreateSObjectResult**、または新しいオブジェクトの作成中のエラーのリストのいずれかになります。

```
...to("salesforce:upsertSObject?sObjectIdName=Name")...
```

268.4.2. Rest Bulk API

プロデューサーエンドポイントは、次の API を使用できます。すべてのジョブデータ形式、つまり xml、csv、zip/xml、および zip/csv がサポートされています。
リクエストとレスポンスは、ルートによってマーシャリング/マーシャリング解除する必要があります。通常、リクエストは CSV ファイルなどのストリームソースになります。
また、応答をファイルに保存して、要求と関連付けることもできます。

operationName には以下を使用できます。

- createJob - Salesforce 一括ジョブを作成します
- getJob - Salesforce ID を使用してジョブを取得します
- closeJob - ジョブを閉じます
- abortJob - ジョブを中止します
- createBatch - 一括ジョブ内でバッチを送信します
- getBatch - Id を使用してバッチを取得します
- getAllBatches - 一括ジョブ ID のすべてのバッチを取得します
- getRequest - バッチのリクエストデータ (XML/CSV) を取得します
- getResults - 完了時にバッチの結果を取得します
- createBatchQuery - SOQL クエリーからバッチを作成します
- getQueryResultIds - バッチクエリーの結果 ID のリストを取得します
- getQueryResult - 結果 ID の結果を取得します

- getRecentReports - Report List リソースに GET リクエストを送信して、最近表示したレポートを最大 200 件取得します
- getReportDescription - レポート、レポートタイプ、およびレポートの関連メタデータを表形式、要約形式、またはマトリックス形式で取得します。
- executeSyncReport - フィルターを変更して、または変更せずにレポートを同期的に実行し、最新の概要データを返します
- executeAsyncReport - フィルターの有無にかかわらずレポートのインスタンスを非同期的に実行し、詳細の有無にかかわらず概要データを返します
- getReportInstances - 非同期実行を要求したレポートのインスタンスのリストを返します。リスト内の各項目は、レポートの個別のインスタンスとして扱われます。
- getReportResults : レポートの実行結果が含まれます。

たとえば、次のプロデューサーエンドポイントは、createBatch API を使用してジョブバッチを作成します。in メッセージには、**InputStream** に変換できる本文 (通常は、ファイルからの UTF-8 CSV または XML コンテンツなど) と、ジョブのヘッダーフィールド jobId およびジョブコンテンツタイプの contentType が含まれている必要があります。XML、CSV、ZIP_XML、または ZIP_CSV を指定できます。put メッセージの本文には、成功した場合は **BatchInfo** が含まれ、エラーが発生した場合は **SalesforceException** が出力されます。

```
...to("salesforce:createBatchJob")..
```

268.4.3. Rest Streaming API

コンシューマーエンドポイントは、ストリーミングエンドポイントに次の構文を使用して、作成/更新時に Salesforce 通知を受信できます。

トピックを作成してサブスクライブするには

```
from("salesforce:CamelTestTopic?
notifyForFields=ALL&notifyForOperations=ALL&sObjectName=Merchandise__c&updateTopic=true&sO
bjectQuery=SELECT Id, Name FROM Merchandise__c")...
```

既存のトピックをサブスクライブするには

```
from("salesforce:CamelTestTopic&sObjectName=Merchandise__c")...
```

268.4.4. プラットフォームイベント

プラットフォームイベントを発行するには、**createSObject** 操作を使用します。また、メッセージ本文を JSON 文字列またはキー値データを含む InputStream に設定します。その場合、**sObjectName** をイベントの API 名、またはイベントの適切なクラス名を持つ AbstractDTOBase から拡張されたクラスに設定する必要があります。

たとえば、DTO を使用すると、次のようになります。

```
class Order_Event__e extends AbstractDTOBase {
  @JsonProperty("OrderNumber")
  private String orderNumber;
  // ... other properties and getters/setters
```

```
}

from("timer:tick")
    .process(exchange -> {
        final Message in = exchange.getIn();
        String orderNumber = "ORD" + String.valueOf(in.getHeader(Exchange.TIMER_COUNTER));
        Order_Event__e event = new Order_Event__e();
        event.setOrderNumber(orderNumber);
        in.setBody(event);
    })
    .to("salesforce:createSObject");
```

または、JSON イベントデータを使用します。

```
from("timer:tick")
    .process(exchange -> {
        final Message in = exchange.getIn();
        String orderNumber = "ORD" + String.valueOf(in.getHeader(Exchange.TIMER_COUNTER));
        in.setBody("{\"OrderNumber\":\"" + orderNumber + "\"}");
    })
    .to("salesforce:createSObject?sObjectName=Order_Event__e");
```

プラットフォームイベントを受信するには、プラットフォームイベントの API 名の前に **event/** (または **/event/**) を付けたコンシューマーエンドポイントを使用します (例: **salesforce:events/Order_Event__e**)。そのエンドポイントから消費するプロセッサーは、**rawPayload** がそれぞれ **false** または **true** であるかに応じて、本文で **org.apache.camel.component.salesforce.api.dto.PlatformEvent** オブジェクトまたは **org.cometd.bayeux.Message** のいずれかを受け取ります。

たとえば、1 つのイベントを消費する最も単純な形式では、次のようになります。

```
PlatformEvent event = consumer.receiveBody("salesforce:event/Order_Event__e",
PlatformEvent.class);
```

268.5. 例

268.5.1. ContentWorkspace へのドキュメントのアップロード

Processor インスタンスを使用して、Java で ContentVersion を作成します。

```
public class ContentProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message message = exchange.getIn();

        ContentVersion cv = new ContentVersion();
        ContentWorkspace cw = getWorkspace(exchange);
        cv.setFirstPublishLocationId(cw.getId());
        cv.setTitle("test document");
        cv.setPathOnClient("test_doc.html");
        byte[] document = message.getBody(byte[].class);
        ObjectMapper mapper = new ObjectMapper();
        String enc = mapper.convertValue(document, String.class);
        cv.setVersionDataUrl(enc);
        message.setBody(cv);
```

```
    }

    protected ContentWorkspace getWorkSpace(Exchange exchange) {
        // Look up the content workspace somehow, maybe use enrich() to add it to a
        // header that can be extracted here
        ....
    }
}
```

プロセッサーからの出力を Salesforce コンポーネントに渡します。

```
from("file:///home/camel/library")
    .to(new ContentProcessor())     // convert bytes from the file into a ContentVersion SObject
                                    // for the salesforce component
    .to("salesforce:createSObject");
```

268.6. SALESFORCE 制限 API の使用

salesforce:limits 操作を使用すると、Salesforce から API 制限を取得して、受信したデータに基づいて行動できます。**salesforce:limits** 操作の結果は **org.apache.camel.component.salesforce.api.dto.Limits** クラスにマップされ、カスタムプロセッサーまたは式で使用できます。

たとえば、Salesforce の API 使用を制限して、毎日の API リクエストの 10% を他のルートに割り当てる必要があるとします。出力メッセージの本文には **org.apache.camel.component.salesforce.api.dto.Limits** オブジェクトのインスタンスが含まれています。これは、Content Based Router および Content Based Router と Spring Expression Language (SpEL) と組み合わせて使用して、いつ選択するかを選択できます。クエリーを実行します。

body.dailyApiRequests.remaining に保持されている整数値で **1.0** を乗算すると、式が浮動小数点演算と同じように評価されることに注意してください。それがなければ、整数除算が行われ、**0** (いくつかの API 制限が消費されます) または **1** (API 制限は消費されません)。

```
from("direct:querySalesforce")
    .to("salesforce:limits")
    .choice()
    .when(spel("#{1.0 * body.dailyApiRequests.remaining / body.dailyApiRequests.max < 0.1}"))
        .to("salesforce:query?...")
    .otherwise()
        .setBody(constant("Used up Salesforce API limits, leaving 10% for critical routes"))
    .endChoice()
```

268.7. 承認の操作

すべてのプロパティーの名前は、Salesforce REST API とまったく同じで、**approval.**の接頭辞が付いています。テンプレートとして使用されるエンドポイントの **approval.PropertyName** を設定することで、承認プロパティーを設定できます。つまり、本文にもヘッダーにも存在しないプロパティーは、エンドポイント設定から取得されます。または、**approval** プロパティーをレジストリー内の Bean への参照に割り当てることで、エンドポイントに承認テンプレートを設定できます。

受信メッセージヘッダーで同じ **approval.PropertyName** を使用してヘッダー値を指定することもできます。

最後に、本文には、バッチとして処理する 1 つの **AprovalRequest** または **ApprovalRequest** オブジェクトの **Iterable** を含めることができます。

覚えておくべき重要なことは、これら 3 つのメカニズムで指定された値の優先度です。

1. 本文の値は他の値よりも優先されます
2. メッセージヘッダーの値はテンプレート値よりも優先されます
3. ヘッダーまたは本文に他の値が指定されていない場合、テンプレートの値が設定されます

たとえば、ヘッダーの値を使用して承認のために 1 つのレコードを送信するには、次を使用します。

指定した経路:

```
from("direct:example1")//
        .setHeader("approval.ContextId", simple("${body['contextId']}"))
        .setHeader("approval.NextApproverIds", simple("${body['nextApproverIds']}"))
        .to("salesforce:approval?"//
           + "approval.actionType=Submit"//
           + "&approval.comments=this is a test"//
           + "&approval.processDefinitionNameOrId=Test_Account_Process"//
           + "&approval.skipEntryCriteria=true");
```

以下を使用して、承認のためにレコードを送信できます。

```
final Map<String, String> body = new HashMap<>();
body.put("contextId", accountIds.iterator().next());
body.put("nextApproverIds", userId);

final ApprovalResult result = template.requestBody("direct:example1", body, ApprovalResult.class);
```

268.8. SALESFORCE 最近の項目 API の使用

最近のアイテムを取得するには、**salesforce:recent** オペレーションを使用します。このオペレーションは、**org.apache.camel.component.salesforce.api.dto.RecentItem** オブジェクト (**List<RecentItem>**) の **java.util.List** を返します。これには、**Id**、**Name**、および **Attributes** (**type** および **url** プロパティーを含む) が含まれます。返されるレコードの最大数に設定された **limit** パラメーターを指定することにより、返されるアイテムの数を制限できます。以下に例を示します。

```
from("direct:fetchRecentItems")
    to("salesforce:recent")
        .split().body()
            .log("${body.name} at ${body.attributes.url}");
```

268.9. 承認の操作

すべてのプロパティーの名前は、Salesforce REST API とまったく同じで、**approval.**の接頭辞が付いています。テンプレートとして使用されるエンドポイントの **approval.PropertyName** を設定することで、承認プロパティーを設定できます。つまり、本文にもヘッダーにも存在しないプロパティーは、エンドポイント設定から取得されます。または、**approval** プロパティーをレジストリー内の Bean への参照に割り当てることで、エンドポイントに承認テンプレートを設定できます。

受信メッセージヘッダーで同じ **approval.PropertyName** を使用してヘッダー値を指定することもできます。

最後に、本文には、バッチとして処理する 1 つの **AprovalRequest** または **ApprovalRequest** オブジェクトの **Iterable** を含めることができます。

覚えておくべき重要なことは、これら 3 つのメカニズムで指定された値の優先度です。

1. 本文の値は他の値よりも優先されます
2. メッセージヘッダーの値はテンプレート値よりも優先されます
3. ヘッダーまたは本文に他の値が指定されていない場合、テンプレートの値が設定されます

たとえば、ヘッダーの値を使用して承認のために 1 つのレコードを送信するには、次を使用します。

指定した経路:

```
from("direct:example1")//
        .setHeader("approval.ContextId", simple("${body['contextId']}"))
        .setHeader("approval.NextApproverIds", simple("${body['nextApproverIds']}"))
        .to("salesforce:approval?"//
           + "approvalActionType=Submit"//
           + "&approvalComments=this is a test"//
           + "&approvalProcessDefinitionNameOrId=Test_Account_Process"//
           + "&approvalSkipEntryCriteria=true");
```

以下を使用して、承認のためにレコードを送信できます。

```
final Map<String, String> body = new HashMap<>();
body.put("contextId", accountIds.iterator().next());
body.put("nextApproverIds", userId);

final ApprovalResult result = template.requestBody("direct:example1", body, ApprovalResult.class);
```

268.10. SALESFORCE COMPOSITE API を使用して SOBJECT ツリーを送信する

親子関係を含む最大 200 件のレコードを作成するには、**salesforce:composite-tree** オペレーションを使用します。これには、入力メッセージで **org.apache.camel.component.salesforce.api.dto.composite.SObjectTree** のインスタンスが必要であり、出力メッセージでオブジェクトの同じツリーを返します。ツリー内の **org.apache.camel.component.salesforce.api.dto.AbstractSObjectBase** インスタンスは、識別子の値 (**Id** プロパティー) または対応する **org.apache.camel.component.salesforce.api.dto.composite.SObjectNode** で更新されます。失敗すると **errors** が表示されます。

一部のレコード操作は成功する場合と失敗する場合があることに注意してください。そのため、手動でエラーをチェックする必要があります。

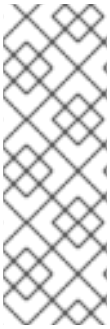
この機能を使用する最も簡単な方法は、**camel-salesforce-maven-plugin** によって生成された DTO を使用することですが、データベースの主キーなど、ツリー内の各オブジェクトを識別する参照をカスタマイズするオプションもあります。

例を見てみましょう:

```
Account account = ...
Contact president = ...
Contact marketing = ...

Account anotherAccount = ...
Contact sales = ...
Asset someAsset = ...

// build the tree
SObjectTree request = new SObjectTree();
request.addObject(account).addChildren(president, marketing);
request.addObject(anotherAccount).addChild(sales).addChild(someAsset);

final SObjectTree response = template.requestBody("salesforce:composite-tree", tree,
SObjectTree.class);
final Map<Boolean, List<SObjectNode>> result = response.allNodes()
                                        .collect(Collectors.groupingBy(SObjectNode::hasErrors));

final List<SObjectNode> withErrors = result.get(true);
final List<SObjectNode> succeeded = result.get(false);

final String firstId = succeeded.get(0).getId();
```

268.11. SALESFORCE COMPOSITE API を使用して複数のリクエストをバッチで送信する

Composite API のバッチ操作 (**composite-batch**) を使用すると、複数のリクエストをバッチに蓄積して一度に送信できるため、複数の個別のリクエストの往復コストを節約できます。次に、各応答は、順序が保持された応答のリストで受信されるため、n 番目の要求の応答は応答の n 番目の場所になります。



注記

結果は API ごとに異なる可能性があるため、リクエストの結果は **java.lang.Object** として提供されます。ほとんどの場合、結果は文字列のキーと値を持つ **java.util.Map** か、値として他の **java.util.Map** になります。JSON 形式で作成されたリクエストは、いくつかの型情報を保持するため (つまり、どの値が文字列で、どの値が数値であるかがわかっている)、一般的にそれらはより型に適しています。XML と JSON では応答が異なることに注意してください。これは、Salesforce API からの応答が異なるためです。したがって、応答処理コードを変更せずにフォーマットを切り替える場合は注意してください。

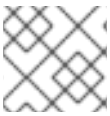
例を見てみましょう:

```
final String acountId = ...
final SObjectBatch batch = new SObjectBatch("38.0");

final Account updates = new Account();
updates.setName("NewName");
batch.addUpdate("Account", accountId, updates);

final Account newAccount = new Account();
newAccount.setName("Account created from Composite batch API");
batch.addCreate(newAccount);
```

```
batch.addGet("Account", accountId, "Name", "BillingPostalCode");

batch.addDelete("Account", accountId);

final SObjectBatchResponse response = template.requestBody("salesforce:composite-batch?
format=JSON", batch, SObjectBatchResponse.class);

boolean hasErrors = response.hasErrors(); // if any of the requests has resulted in either 4xx or 5xx
HTTP status
final List<SObjectBatchResult> results = response.getResults(); // results of three operations sent in
batch

final SObjectBatchResult updateResult = results.get(0); // update result
final int updateStatus = updateResult.getStatusCode(); // probably 204
final Object updateResultData = updateResult.getResult(); // probably null

final SObjectBatchResult createResult = results.get(1); // create result
@SuppressWarnings("unchecked")
final Map<String, Object> createData = (Map<String, Object>) createResult.getResult();
final String newAccountId = createData.get("id"); // id of the new account, this is for JSON, for XML it
would be createData.get("Result").get("id")

final SObjectBatchResult retrieveResult = results.get(2); // retrieve result
@SuppressWarnings("unchecked")
final Map<String, Object> retrieveData = (Map<String, Object>) retrieveResult.getResult();
final String accountName = retrieveData.get("Name"); // Name of the retrieved account, this is for
JSON, for XML it would be createData.get("Account").get("Name")
final String accountBillingPostalCode = retrieveData.get("BillingPostalCode"); // Name of the retrieved
account, this is for JSON, for XML it would be createData.get("Account").get("BillingPostalCode")

final SObjectBatchResult deleteResult = results.get(3); // delete result
final int updateStatus = deleteResult.getStatusCode(); // probably 204
final Object updateResultData = deleteResult.getResult(); // probably null
```

268.12. SALESFORCE COMPOSITE API を使用して、チェーン化された複数の要求を送信する

composite 操作では、連鎖可能な最大 25 個のリクエストを送信できます。たとえば、前のリクエストで生成された識別子を後続のリクエストで使用できます。個々のリクエストとレスポンスは、提供された **参照** にリンクされています。



注記

複合 API は JSON ペイロードのみをサポートします。

注記

バッチ API と同様に、結果は API ごとに異なる可能性があるため、リクエストの結果は **java.lang.Object** として提供されます。ほとんどの場合、結果は文字列のキーと値を持つ **java.util.Map** か、値として他の **java.util.Map** になります。JSON 形式で作成されたリクエストは、いくつかの型情報を保持するため (つまり、どの値が文字列で、どの値が数値であるかがわかっている)、一般的にそれらはより型に適しています。

例を見てみましょう:

```
SObjectComposite composite = new SObjectComposite("38.0", true);

// first insert operation via an external id
final Account updateAccount = new TestAccount();
updateAccount.setName("Salesforce");
updateAccount.setBillingStreet("Landmark @ 1 Market Street");
updateAccount.setBillingCity("San Francisco");
updateAccount.setBillingState("California");
updateAccount.setIndustry(Account_IndustryEnum.TECHNOLOGY);
composite.addUpdate("Account", "001xx000003DIpcAAG", updateAccount, "UpdatedAccount");

final Contact newContact = new TestContact();
newContact.setLastName("John Doe");
newContact.setPhone("1234567890");
composite.addCreate(newContact, "NewContact");

final AccountContactJunction__c junction = new AccountContactJunction__c();
junction.setAccount__c("001xx000003DIpcAAG");
junction.setContactId__c("@{NewContact.id}");
composite.addCreate(junction, "JunctionRecord");

final SObjectCompositeResponse response = template.requestBody("salesforce:composite?
format=JSON", composite, SObjectCompositeResponse.class);
final List<SObjectCompositeResult> results = response.getCompositeResponse();

final SObjectCompositeResult accountUpdateResult = results.stream().filter(r ->
"UpdatedAccount".equals(r.getReferenceId())).findFirst().get()
final int statusCode = accountUpdateResult.getHttpStatusCode(); // should be 200
final Map<String, ?> accountUpdateBody = accountUpdateResult.getBody();

final SObjectCompositeResult contactCreationResult = results.stream().filter(r ->
"JunctionRecord".equals(r.getReferenceId())).findFirst().get()
```

268.13. CAMEL SALESFORCE MAVEN プラグイン

この Maven プラグインは、Camel Salesforce の DTO を生成します。

268.14. オプション

Salesforce コンポーネントは、以下に示す 29 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
authenticationType (security)	使用する明示的な認証方法で、USERNAME_PASSWORD、REFRESH_TOKEN、または JWT のいずれかです。Salesforce コンポーネントは、プロパティーセットから使用する認証方法を自動決定できます。このプロパティーを設定してあいまいさを排除します。		AuthenticationType

名前	説明	デフォルト	タイプ
loginConfig (security)	1つのネストされた Bean 内のすべての認証設定、そこに設定されたすべてのプロパティーは、コンポーネントでも直接設定できます		SalesforceLoginConfig
instanceUrl (security)	認証後に使用される Salesforce インスタンスの URL。デフォルトでは、認証の成功時に Salesforce から受信されます		String
loginUrl (security)	認証に使用される Salesforce インスタンスの **必須** URL。デフォルトでは https://login.salesforce.com に設定されています	https://login.salesforce.com	String
clientId (security)	Salesforce インスタンス設定で設定された接続アプリケーションの **必須** OAuth コンシューマーキー。通常、接続アプリケーションを設定する必要がありますが、パッケージをインストールすることで提供できます。		String
clientSecret (security)	Salesforce インスタンス設定で設定された接続アプリケーションの OAuth コンシューマーシークレット。		String
keystore (security)	OAuth JWT フローで使用する KeyStore パラメーター。KeyStore には、秘密鍵と証明書を含むエントリーを1つだけ含める必要があります。Salesforce は証明書チェーンを検証しないため、これは簡単に自己署名証明書になる可能性があります。対応する接続アプリケーションに証明書をアップロードしていることを確認してください。		KeyStoreParameters
refreshToken (security)	リフレッシュトークン OAuth フローですでに取得されているリフレッシュトークン。Web アプリケーションをセットアップして、リフレッシュトークンを受け取るコールバック URL を設定するか、https://login.salesforce.com/services/oauth2/success または https://test.salesforce.com/services/oauth2/success で組み込みのコールバックを使用して設定し、フローの最後で URL から refresh_token を取得する必要があります。開発組織では、Salesforce がコールバック Web アプリケーションを localhost でホストすることを許可していることに注意してください。		String
userName (security)	アクセストークンにアクセスするために OAuth フローで使用されるユーザー名。パスワード OAuth フローは簡単に開始できますが、他のフローよりも安全性が低いと見なされるため、一般的には避けるべきです。		String

名前	説明	デフォルト	タイプ
password (security)	アクセストークンにアクセスするために OAuth フローで使用されるパスワード。パスワード OAuth フローは簡単に開始できますが、他のフローよりも安全性が低いと見なされるため、一般的には避けるべきです。セキュリティートークンを使用する場合は、パスワードの末尾にセキュリティートークンを追加してください。		String
lazyLogin (security)	true に設定すると、コンポーネントの開始時にコンポーネントが Salesforce に対して認証されなくなります。通常、これを (デフォルトの) false に設定し、早期に認証して、認証の問題をすぐに認識します。	false	boolean
config (common)	グローバルエンドポイント設定 - すべてのエンドポイントに共通の値を設定するために使用します		SalesforceEndpoint 設定
httpClientProperties (common)	基になる HTTP クライアントで設定できるプロパティーを設定するために使用されます。利用可能なすべてのオプションについては、SalesforceHttpClient と Jetty HttpClient のプロパティーを参照してください。		Map
longPollingTransport Properties (common)	ストリーミング API によって使用される BayeuxClient (CometD) によって使用される LongPollingTransport で設定できる任意のプロパティーを設定するために使用されます		Map
sslContextParameters (security)	使用する SSL パラメーター。使用可能なすべてのオプションについては、SSLContextParameters クラスを参照してください。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします	false	boolean
httpProxyHost (proxy)	使用する HTTP プロキシーサーバーのホスト名。		String
httpProxyPort (proxy)	使用する HTTP プロキシーサーバーのポート番号。		Integer
httpProxyUsername (security)	HTTP プロキシーサーバーに対する認証に使用するユーザー名。		String

名前	説明	デフォルト	タイプ
httpProxyPassword (security)	HTTP プロキシーサーバーに対する認証に使用するパスワード。		String
isHttpProxySocks4 (proxy)	true に設定すると、SOCKS4 プロキシーとして使用するように HTTP プロキシーが設定されます。	false	boolean
isHttpProxySecure (security)	false に設定すると、HTTP プロキシーへのアクセス時に TLS の使用が無効になります。	true	boolean
httpProxyIncluded Addresses (proxy)	HTTP プロキシーサーバーを使用するアドレスのリスト。		Set
httpProxyExcluded Addresses (proxy)	HTTP プロキシーサーバーを使用しないアドレスのリスト。		Set
httpProxyAuthUri (security)	HTTP プロキシーサーバーに対する認証で使用されます。httpProxyUsername と httpProxyPassword を認証に使用するには、プロキシーサーバーの URI と一致する必要があります。		String
httpProxyRealm (security)	HTTP プロキシーサーバーに対するプリエンプティブ Basic/Digest 認証方式で使用される、プロキシーサーバーのレルム。		String
httpProxyUseDigest Auth (security)	true に設定すると、HTTP プロキシーへの認証時にダイジェスト認証が使用されます。それ以外の場合は、基本認証方法が使用されます	false	boolean
packages (common)	生成された DTO クラスが含まれているパッケージ。通常、クラスは camel-salesforce-maven-plugin を使用して生成されます。生成された DTO を使用して、パラメーター/ヘッダー値で短い SObject 名を使用する利点を得る場合に設定します。		String[]
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Salesforce エンドポイントは、URI 構文を使用して設定されます。

```
salesforce:operationName:topicName
```

パスおよびクエリーパラメーターを使用します。

268.14.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
operationName	使用する操作		OperationName
topicName	使用するトピックの名前		文字列

268.14.2. クエリーパラメーター(44 個のパラメーター):

名前	説明	デフォルト	タイプ
apexMethod (Common)	APEX メソッド名		String
apexQueryParams (common)	APEX メソッドのクエリーパラメーター		Map
apexUrl (Common)	APEX メソッドの URL		String
apiVersion (Common)	Salesforce API バージョン、デフォルトは SalesforceEndpointConfig.DEFAULT_VERSION		String
backoffIncrement (Common)	CometD 自動再接続を超えた失敗に対するストリーミング接続の再起動を試みるバックオフ間隔の増分。		long
batchId (Common)	Bulk API バッチ ID		String
contentType (common)	Bulk API コンテンツタイプ。XML、CSV、ZIP_XML、ZIP_CSV のいずれかです		ContentType
defaultReplayId (Common)	リンク initialReplayIdMap に値が見つからない場合のデフォルトの replayId 設定		Long
format (Common)	Salesforce API 呼び出しに使用するペイロード形式 (JSON または XML) のデフォルトは JSON です		PayloadFormat
httpClient (Common)	Salesforce への接続に使用するカスタム Jetty Http クライアント。		SalesforceHttpClient
includeDetails (Common)	Salesforce1 Analytics レポートに詳細を含めます。デフォルトは false です。		Boolean

名前	説明	デフォルト	タイプ
initialReplayIdMap (Common)	チャンネル名ごとに開始する Replay ID。		Map
instanceId (Common)	Salesforce1 Analytics レポート実行インスタンス ID		String
jobId (Common)	Bulk API ジョブ ID		文字列
limit (Common)	返されるレコード数の制限。一部の API に適用されます。Salesforce のドキュメントを確認してください。		Integer
maxBackoff (Common)	CometD 自動再接続を超えた障害に対するストリーミング接続の再起動試行の最大バックオフ間隔。		long
notFoundBehaviour (Common)	Salesforce API から受信した 404 not found ステータスの動作を設定します。本文を NULL リンク NotFoundBehaviourNULL に設定するか、交換リンク NotFoundBehaviourEXCEPTION で例外を通知する (デフォルト) 必要があります。		NotFoundBehaviour
notifyForFields (Common)	フィールドの通知、オプションは ALL、REFERENCED、SELECT、WHERE です		NotifyForFieldsEnum
notifyForOperationCreate (Common)	作成操作を通知します。デフォルトは false (API バージョン = 29.0) です		Boolean
notifyForOperationDelete (Common)	削除操作を通知します。デフォルトは false (API バージョン = 29.0) です		Boolean
notifyForOperations (Common)	操作を通知します。オプションは ALL、CREATE、EXTENDED、UPDATE (API バージョン 29.0) です		NotifyForOperations 列挙
notifyForOperationUndelete (common)	削除取り消し操作を通知します。デフォルトは false (API バージョン = 29.0) です		Boolean
notifyForOperationUpdate (common)	更新操作を通知します。デフォルトは false (API バージョン = 29.0) です		Boolean
objectMapper (Common)	Salesforce オブジェクトをシリアライズ/デシリアライズするときに使用するカスタム Jackson ObjectMapper。		ObjectMapper

名前	説明	デフォルト	タイプ
rawPayload (Common)	DTO の代わりに、リクエストとレスポンス (形式に応じて JSON または XML) に Raw ペイロード文字列を使用します。デフォルトでは false です	false	boolean
reportId (Common)	Salesforce1 Analytics レポート ID		String
reportMetadata (Common)	フィルタリング用の Salesforce1 Analytics レポートのメタデータ		ReportMetadata
resultId (Common)	Bulk API の結果 ID		String
serializeNulls (Common)	指定された DTO の NULL 値を空 (NULL) 値でシリアル化する必要があります。これは、JSON データ形式のみに影響します。	false	boolean
sObjectBlobFieldName (common)	SObject blob フィールド名		String
sObjectClass (Common)	完全修飾 SObject クラス名。通常は camel-salesforce-maven-plugin を使用して生成されます		String
sObjectFields (Common)	取得する SObject フィールド		String
sObjectId (Common)	API で必要な場合は SObject ID		String
sObjectIdName (Common)	sObject 外部 ID 項目名		String
sObjectIdValue (Common)	sObject 外部 ID 項目の値		String
sObjectName (Common)	API で必要またはサポートされている場合の sObject 名		String
sObjectQuery (Common)	Salesforce SOQL クエリー文字列		String
sObjectSearch (Common)	Salesforce SOSL 検索文字列		String
updateTopic (Common)	ストリーミング API の使用時に既存のプッシュトピックを更新するかどうか。デフォルトは false です	false	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
replayId (consumer)	サブスクライブ時に使用する replayId 値		Long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

明らかなセキュリティー上の理由から、pom.xml では clientId、clientSecret、userName、および password フィールドを設定しないことをお勧めします。プラグインは、残りのプロパティーに対して設定する必要があり、次のコマンドを使用して実行できます。

```
mvn camel-salesforce:generate -DcamelSalesforce.clientId=<clientid> -
DcamelSalesforce.clientSecret=<clientsecret> \
    -DcamelSalesforce.userName=<username> -DcamelSalesforce.password=<password>
```

生成された DTO は、Jackson および XStream アノテーションを使用します。すべての Salesforce フィールドタイプがサポートされています。日時フィールドは Joda DateTime にマップされ、ピックリストフィールドは生成された Java 列挙にマップされます。

268.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第269章 SAP コンポーネント

SAP コンポーネントは、10 個の異なる SAP コンポーネントのスイートで設定されるパッケージです。sRFC、tRFC、および qRFC プロトコルをサポートするリモートファンクションコール (RFC) コンポーネントがあります。また、IDoc 形式のメッセージを使用して通信を容易にする IDoc コンポーネントがあります。このコンポーネントは、SAP Java Connector (SAP JCo) ライブラリーを使用して SAP との双方向通信を容易にし、SAP IDoc ライブラリーを使用して中間ドキュメント (IDoc) 形式でのドキュメントの送信を促進します。

269.1. 概要

依存関係

Maven ユーザーがこのコンポーネントを使用するには、**pom.xml** ファイルに次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.fusesource</groupId>
  <artifactId>camel-sap</artifactId>
  <version>x.x.x</version>
<dependency>
```

SAP コンポーネントの追加のプラットフォーム制限

SAP コンポーネントはサードパーティーの JCo 3.0 および Doc 3.0 ライブラリーに依存しているため、これらのライブラリーがサポートするプラットフォームにのみインストールできます。プラットフォームの制限の詳細は、Red Hat JBoss Fuse サポートされる構成 を参照してください。

SAP JCo および SAP IDoc ライブラリー

SAP コンポーネントを使用するための前提条件は、SAP Java Connector (SAP JCo) ライブラリーと SAP IDoc ライブラリーが Java ランタイムの **lib**/ ディレクトリーにインストールされていることです。ターゲットオペレーティングシステムに適した SAP ライブラリーのセットを SAP Service Marketplace からダウンロードしていることを確認する必要があります。

ライブラリーファイルの名前は、ターゲットオペレーティングシステムによって異なります。表269.1「必要な SAP ライブラリー」

表269.1 必要な SAP ライブラリー

SAP コンポーネント	Linux と UNIX	Windows
SAP JCo 3	**sapjco3.jar** **libsapjco3.so**	**sapjco3.jar** **sapjco3.dll**
SAP IDoc	**sapidoc3.jar**	**sapidoc3.jar**

Fuse OSGi コンテナーへのデプロイ

以下のように、SAP JCo ライブラリーと SAP IDoc ライブラリーを JBoss Fuse OSGi コンテナーにインストールできます。

1. SAP サービスマーケットプレイス (http://service.sap.com/public/connectors) から SAP JCo ライブラリーと SAP IDoc ライブラリーをダウンロードし、オペレーティングシステムに適したバージョンのライブラリーを選択してください。

注記

バージョン 3.0.11 以降の JCo ライブラリーとバージョン 3.0.10 以降の IDoc ライブラリーが必要です。これらのライブラリーをダウンロードして使用するには、**SAP Service Marketplace アカウント** が必要です。

2. **sapjco3.jar**、**libsapjco3.so** (または Windows では **sapjco3.dll**)、および **sapidoc3.jar** ライブラリーファイルを Fuse インストールの **lib/** ディレクトリーにコピーします。

3. 設定プロパティーファイル **etc/config.properties** とカスタムプロパティーファイル **etc/custom.properties** の両方をテキストエディターで開きます。**etc/config.properties** ファイルで、**org.osgi.framework.system.packages.extra** プロパティーを探し、完全なプロパティー設定をコピーします (この設定は、行の継続を示すために使用されるバックスラッシュ文字 \ を使用して、複数の行にまたがります。)。この設定を **etc/custom.properties** ファイルに貼り付けます。
SAP ライブラリーをサポートするために必要な追加パッケージを追加できるようになりました。**etc/custom.properties** ファイルで、次のように必要なパッケージを **org.osgi.framework.system.packages.extra** 設定に追加します。

```
org.osgi.framework.system.packages.extra = \
 ... , \
 com.sap.conn.idoc, \
 com.sap.conn.idoc.jco, \
 com.sap.conn.jco, \
 com.sap.conn.jco.ext, \
 com.sap.conn.jco.monitor, \
 com.sap.conn.jco.rt, \
 com.sap.conn.jco.server
```

リストが適切に続くように、新しいエントリーの前の各行の終わりにコンマとバックスラッシュ、\ を含めることを忘れないでください。

4. これらの変更を有効にするには、コンテナーを再起動する必要があります。

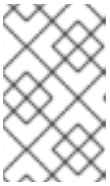
5. コンテナーに **camel-sap** 機能をインストールする必要があります。Karaf コンソールで、次のコマンドを入力します。

```
JBossFuse:karaf@root> features:install camel-sap
```

JBoss EAP コンテナーへのデプロイ

SAP コンポーネントを JBoss EAP コンテナーにデプロイするには、以下の手順を実行します。

1. SAP サービスマーケットプレイス (http://service.sap.com/public/connectors) から SAP JCo ライブラリーと SAP IDoc ライブラリーをダウンロードし、オペレーティングシステムに適したバージョンのライブラリーを選択してください。

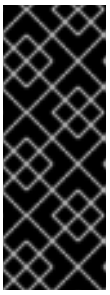


注記

バージョン 3.0.11 以降の JCo ライブラリーとバージョン 3.0.10 以降の IDoc ライブラリーが必要です。これらのライブラリーをダウンロードして使用するには、**SAP Service Marketplace アカウント** が必要です。

2. JCo ライブラリーファイルと IDoc ライブラリーファイルを JBoss EAP インストールの適切なサブディレクトリーにコピーします。たとえば、ホストプラットフォームが 64 ビット Linux (**linux-x86_64**) の場合、次のようにライブラリーファイルをインストールします。

```
cp sapjco3.jar sapidoc3.jar
$JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/
mkdir -p $JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/lib/linux-
x86_64
cp libsapjco3.so
$JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/lib/linux-x86_64/
```



重要

ネイティブライブラリー (**libsapjco3.so** など) を JBoss EAP インストールにインストールする場合、ライブラリーサブディレクトリーの命名には標準化された規則があり、これに従う必要があります。64 ビット Linux の場合、サブディレクトリーは **linux-x86_64** です。他のプラットフォームについては、https://docs.jboss.org/author/display/MODULES/Native+Libraries を参照してください。

3. **$JBOSS_HOME/modules/system/layers/fuse/org/wildfly/camel/extras/main/module.xml** という新しいファイルを作成し、次の内容を **追加** します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.wildfly.camel.extras">

 <dependencies>
  <module name="org.fusesource.camel.component.sap" export="true" services="export" />
 </dependencies>

</module>
```

URI 形式

SAP コンポーネントによって提供されるエンドポイントには、リモートファンクションコール (RFC) エンドポイントと中間ドキュメント (IDoc) エンドポイントの 2 種類があります。

RFC エンドポイントの URI 形式は次のとおりです。

```
sap-srfc-destination:destinationName:rfcName
sap-trfc-destination:destinationName:rfcName
sap-qrfc-destination:destinationName:queueName:rfcName
sap-srfc-server:serverName:rfcName[?options]
sap-trfc-server:serverName:rfcName[?options]
```

IDoc エンドポイントの URI 形式は次のとおりです。

```
sap-idoc-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-idoclist-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-qidoc-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applic
ationRelease]]]
sap-qidoclist-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applic
ationRelease]]]
sap-idoclist-
server:serverName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
[?options]
```

sap- **endpointKind** -destination で始まる URI 形式は、宛先エンドポイント (つまり、Camel プロデューサーエンドポイント) を定義するために使用され、**destinationName** は SAP インスタンスへの特定のアウトバウンド接続の名前です。で説明されているように、アウトバウンド接続はコンポーネントレベルで名前が付けられ、設定されます。「宛先設定」

sap- **endpointKind** -server で始まる URI 形式は、サーバーエンドポイント (つまり、Camel コンシューマーエンドポイント) を定義するために使用され、**serverName** は SAP インスタンスからの特定のインバウンド接続の名前です。インバウンド接続は、コンポーネントレベルで名前が付けられ、設定されます。「サーバー設定」

RFC エンドポイント URI のその他のコンポーネントは次のとおりです。

rfcName

(必須) 宛先エンドポイント URI では、接続された SAP インスタンスのエンドポイントによって呼び出される RFC の名前です。サーバーエンドポイント URI では、接続された SAP インスタンスから呼び出されたときにエンドポイントによって処理される RFC の名前です。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

IDoc エンドポイント URI のその他のコンポーネントは次のとおりです。

idocType

(必須) このエンドポイントによって生成される IDoc の基本 IDoc タイプを指定します。

idocTypeExtension

このエンドポイントによって生成された IDoc の IDoc タイプ拡張 (存在する場合) を指定します。

systemRelease

このエンドポイントによって生成された IDoc に関連付けられている SAP Basis Release がある場合は、それを指定します。

applicationRelease

このエンドポイントによって生成された IDoc の関連付けられたアプリケーションリリースがある場合は、それを指定します。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

RFC 宛先エンドポイントのオプション

RFC 宛先エンドポイント (**sap-srfc-destination**、**sap-trfc-destination**、および **sap-qrfc-destination**) は、次の URI オプションをサポートしています。

名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します
transacted	**false**	**true** の場合、このエンドポイントが SAP トランザクションを開始することを指定します

RFC サーバーエンドポイントのオプション

SAP RFC サーバーエンドポイント (**sap-srfc-server** および **sap-trfc-server**) は、次の URI オプションをサポートしています。

名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します。
propagateExceptions	**false**	**(sap-trfc-server エンドポイントのみ) true** の場合、エクスチェンジの例外ハンドラーではなく、このエンドポイントが SAP の呼び出し元に例外を伝達することを指定します。

IDoc List Server エンドポイントのオプション

SAP IDoc List Server エンドポイント (**sap-idoclist-server**) は、次の URI オプションをサポートしています。

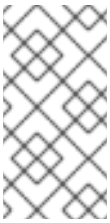
名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します。
propagateExceptions	**false**	**true** の場合、エクスチェンジの例外ハンドラーではなく、このエンドポイントが SAP の呼び出し元に例外を伝搬することを指定します。

RFC および IDoc エンドポイントの概要

SAP コンポーネントパッケージは、次の RFC および IDoc エンドポイントを提供します。

sap-srfc-destination

JBoss Fuse SAP Synchronous Remote Function Call Destination Camel コンポーネント。このエンドポイントは、Camel ルートが SAP システムへのリクエストと SAP システムからのレスポンスの同期配信を必要とする場合に使用する必要があります。



注記

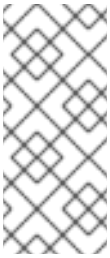
このコンポーネントで使用される sRFC プロトコルは、SAP システムとの間でリクエストとレスポンスを **ベストエフォート** で配信します。リクエストの送信中に通信エラーが発生した場合、受信側の SAP システムでのリモート関数呼び出しの完了ステータスは **不明** のままです。

sap-trfc-destination

JBoss Fuse SAP Transactional Remote Function Call Destination Camel コンポーネント。このエンドポイントは、リクエストを受信側の SAP システムに **最大 1 回** 配信する必要がある場合に使用する必要があります。これを達成するために、コンポーネントはトランザクション ID **tid** を生成します。この ID は、ルートのエクスチェンジでコンポーネントを介して送信されるすべてのリクエストに付随します。受信側の SAP システムは、リクエストを配信する前に、リクエストに付随する **tid** を記録します。SAP システムが同じ **tid** のリクエストを再度受信した場合、リクエストは配信されません。したがって、このコンポーネントのエンドポイントを介してリクエストを送信するときにルートで通信エラーが発生した場合、ルートは同じリクエスト内でリクエストの送信を再試行できますが、配信と実行は 1 回だけです。

注記

このコンポーネントで使用される tRFC プロトコルは非同期であり、レスポンスを返しません。したがって、このコンポーネントのエンドポイントはレスポンスメッセージを返しません。



注記

このコンポーネントは、エンドポイントを介した一連のリクエストの順序を保証しません。これらのリクエストの配信および実行順序は、通信エラーやリクエストの再送信により、受信側の SAP システムで異なる場合があります。配信順序の保証については、JBoss Fuse SAP Queued Remote Function Call Destination Camel コンポーネントを参照してください。

sap-qrfc-destination

JBoss Fuse SAP Queued Remote Function Call Destination Camel コンポーネント。このコンポーネントは、JBoss Fuse Transactional Remote Function Call Destination camel コンポーネントの機能を、そのエンドポイントを介したリクエストの配信に **順番に** 配信保証を追加することで拡張します。このエンドポイントは、一連のリクエストが相互に依存しており、受信側の SAP システムに **最大 1 回、順番に** 配信する必要がある場合に使用する必要があります。このコンポーネントは、JBoss Fuse SAP Transactional Remote Function Call Destination Camel コンポーネントと同じメカニズムを使用して、**最大 1 回の** 配信保証を実現します。順序の保証は、SAP システムが受信した順序でリクエストを **受信キュー** にシリアル化することによって実現されます。受信キューは、SAP 内の **QIN スケジューラー** によって処理されます。インバウンドキューが **アクティブ化される** と、QIN スケジューラーはキューリクエストを順番に実行します。

注記

このコンポーネントで使用される qRFC プロトコルは非同期であり、レスポンスを返しません。したがって、このコンポーネントのエンドポイントはレスポンスメッセージを返しません。

sap-srfc-server

JBoss Fuse SAP Synchronous Remote Function Call Server Camel コンポーネント。このコンポーネントとそのエンドポイントは、SAP システムからの要求と SAP システムへの応答を同期的に処理するために Camel ルートが必要な場合に使用する必要があります。

sap-trfc-server

JBoss Fuse SAP Transactional Remote Function Call Server Camel コンポーネント。このエンドポイントは、送信側の SAP システムがリクエストを Camel ルートに **最大 1 回** 配信する必要がある場合に使用する必要があります。これを実現するために、送信側の SAP システムは、コンポーネントのエンドポイントに送信するすべてのリクエストに付随するトランザクション ID **tid** を生成します。送信側の SAP システムは、**tid** に関連付けられた一連のリクエストを送信する前に、コンポーネントが特定の **tid** を受信したかどうかを最初にチェックします。コンポーネントは、保持している受信 **tid** のリストをチェックし、送信された **tid** がそのリストにない場合は記録し、送信 SAP システムに応答して、**tid** がすでに記録されているかどうかを示します。送信側の SAP システムは、**tid** が以前に記録されていない場合にのみ、一連のリクエストを送信します。これにより、送信側の SAP システムは一連のリクエストを確実に camel ルートに 1 回送信できます。

sap-idoc-destination

JBoss Fuse SAP IDoc Destination Camel コンポーネント。このエンドポイントは、中間ドキュメント (IDoc) のリストを SAP システムに送信するために Camel ルートが必要な場合に使用する必要があります。

sap-idoclist-destination

JBoss Fuse SAP IDoc List Destination Camel コンポーネント。このエンドポイントは、中間ドキュメント (IDoc) リストのリストを SAP システムに送信するために Camel ルートが必要な場合に使用する必要があります。

sap-qidoc-destination

JBoss Fuse SAP Queued IDoc Destination Camel コンポーネント。このコンポーネントとそのエンドポイントは、中間ドキュメント (IDoc) のリストを順番に SAP システムに送信するために Camel ルートが必要な場合に使用する必要があります。

sap-qidoclist-destination

JBoss Fuse SAP Queued IDoc List Destination Camel コンポーネント。このコンポーネントとそのエンドポイントは、中間ドキュメント (IDoc) リストのリストを順番に SAP システムに送信するために camel ルートが必要な場合に使用する必要があります。

sap-idoclist-server

JBoss Fuse SAP IDoc List Server Camel コンポーネント。このエンドポイントは、送信側の SAP システムが中間ドキュメントリストを Camel ルートに配信する必要がある場合に使用する必要があります。このコンポーネントは、**sap-trfc-server-standalone** クイックスタートで説明されているように、tRFC プロトコルを使用して SAP と通信します。

SAP RFC 宛先エンドポイント

RFC 宛先エンドポイントは、SAP へのアウトバウンド通信をサポートします。これにより、これらのエンドポイントは、SAP の ABAP 関数モジュールへの RFC 呼び出しを行うことができます。RFC 宛先エンドポイントは、SAP インスタンスへの特定の接続を介して特定の ABAP 関数への RFC 呼び出しを行うように設定されています。RFC 宛先は、アウトバウンド接続の論理的な指定であり、一意の名前を持っています。RFC 宛先は、**宛先データ** と呼ばれる一連の接続パラメーターによって指定されます。

RFC 宛先エンドポイントは、受信した IN-OUT エクスチェンジの入力メッセージから RFC リクエストを抽出し、そのリクエストを関数呼び出しで SAP にディスパッチします。関数呼び出しからのレスポンスは、エクスチェンジの出力メッセージで返されます。SAP RFC 宛先エンドポイントはアウトバウンド通信のみをサポートするため、RFC 宛先エンドポイントはプロデューサーの作成のみをサポートします。

SAP RFC サーバーエンドポイント

RFC サーバーエンドポイントは、SAP からのインバウンド通信をサポートします。これにより、SAP の ABAP アプリケーションがサーバーエンドポイントに対して RFC 呼び出しを行うことができます。ABAP アプリケーションは、リモート関数モジュールであるかのように RFC サーバーエンドポイントと対話します。RFC サーバーエンドポイントは、SAP インスタンスから特定の接続を介して特定の RFC 関数への RFC 呼び出しを受信するように設定されています。RFC サーバーは、インバウンド接続の論理的な指定であり、一意の名前を持っています。RFC サーバーは、**サーバーデータ** と呼ばれる一連の接続パラメーターによって指定されます。

RFC サーバーエンドポイントは、入力 RFC リクエストを処理し、それを IN-OUT エクスチェンジの入力メッセージとしてディスパッチします。エクスチェンジの出力メッセージは、RFC 呼び出しのレスポンスとして返されます。SAP RFC サーバーエンドポイントはインバウンド通信のみをサポートするため、RFC サーバーエンドポイントはコンシューマーの作成のみをサポートします。

SAP IDoc および IDoc リストの宛先エンドポイント

IDoc 宛先エンドポイントは、SAP へのアウトバウンド通信をサポートします。SAP は、IDoc メッセージに対してさらに処理を実行できます。IDoc ドキュメントはビジネストランザクションを表し、非 SAP システムと簡単にエクスチェンジできます。IDoc 宛先は、**宛先データ** と呼ばれる一連の接続パラメーターによって指定されます。

IDoc リスト宛先エンドポイントは、処理するメッセージが IDoc ドキュメントの **リスト** で設定されることを除いて、IDoc 宛先エンドポイントに似ています。

SAP IDoc リストサーバーエンドポイント

IDoc リストサーバーエンドポイントは、SAP からのインバウンド通信をサポートし、Camel ルートが SAP システムから IDoc ドキュメントのリストを受信できるようにします。IDoc リストサーバーは、**サーバーデータ** と呼ばれる一連の接続パラメーターによって指定されます。

メタデータリポジトリー

メタデータリポジトリーは、次の種類のメタデータを格納するために使用されます。

汎用モジュールのインタフェース説明

このメタデータは、JCo および ABAP ランタイムによって使用され、RFC 呼び出しをチェックして、それらの呼び出しをディスパッチする前に、通信パートナー間でタイプセーフなデータ転送を保証します。リポジトリーには、リポジトリーデータが取り込まれます。リポジトリーデータは、名前付き関数テンプレートのマップです。関数テンプレートには、関数モジュールとの間で渡されるすべてのパラメーターとその入力情報を記述するメタデータが含まれており、関数テンプレートが説明する関数モジュールの一意の名前が付けられています。

IDoc タイプの説明

このメタデータは、IDoc ランタイムによって使用され、IDoc ドキュメントが通信パートナーに送信される前に正しくフォーマットされていることを確認します。基本的な IDoc タイプは、名前、許可されたセグメントのリスト、およびセグメント間の階層関係の説明で設定されます。いくつかの追

加の制約をセグメントに課すことができます。セグメントは必須またはオプションにすることができます。また、各セグメントの最小/最大範囲を指定することができます (そのセグメントの許容反復回数を定義します)。

したがって、SAP 宛先およびサーバーエンドポイントは、RFC 呼び出しを送受信し、IDoc ドキュメントを送受信するために、リポジトリーへのアクセスを必要とします。RFC 呼び出しの場合、エンドポイントによって呼び出されて処理されるすべての機能モジュールのメタデータは、リポジトリー内に存在する必要があります。IDoc エンドポイントの場合、エンドポイントによって処理されるすべての IDoc タイプおよび IDoc タイプ拡張のメタデータは、リポジトリー内に存在する必要があります。宛先およびサーバーエンドポイントによって使用されるリポジトリーの場所は、それぞれの接続の宛先データおよびサーバーデータで指定されます。

SAP 宛先エンドポイントの場合、使用するリポジトリーは通常、SAP システムに存在し、接続先の SAP システムにデフォルト設定されます。このデフォルトでは、宛先データに明示的な設定は必要ありません。さらに、宛先エンドポイントが行うリモート関数呼び出しのメタデータは、それが呼び出す既存の関数モジュールのリポジトリーにすでに存在します。したがって、宛先エンドポイントによって行われる呼び出しのメタデータは、SAP コンポーネントで設定する必要はありません。

一方、サーバーエンドポイントによって処理される関数呼び出しのメタデータは、通常、SAP システムのリポジトリーには存在せず、代わりに SAP コンポーネントに存在するリポジトリーによって提供される必要があります。SAP コンポーネントは、名前付きメタデータリポジトリーのマップを維持します。リポジトリーの名前は、メタデータを提供するサーバーの名前に対応しています。

269.2. 設定

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを格納する 3 つのマップを維持します。**宛先データストア** と **サーバーデータストア** は、特別な設定オブジェクト **SapConnectionConfiguration** で設定され、SAP コンポーネントに (Blueprint XML 設定または Spring XML 設定ファイルのコンテキストで) 自動的に挿入されます。**リポジトリーデータストア** は、関連する SAP コンポーネントで直接設定する必要があります。

269.2.1. 設定の概要

概要

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを格納する 3 つのマップを維持します。コンポーネントのプロパティー **destinationDataStore** は宛先名をキーとする宛先データを格納し、プロパティー **serverDataStore** はサーバー名をキーとするサーバーデータを格納し、プロパティー **repositoryDataStore** はリポジトリー名をキーとするリポジトリーデータを格納します。これらの設定は、初期化中にコンポーネントに渡す必要があります。

例

次の例は、ブループリント XML ファイルでサンプルの宛先データストアとサンプルサーバーデータストアを設定する方法を示しています。**sap-configuration** Bean (タイプ **SapConnectionConfiguration**) は、この XML ファイルで使用されるすべての SAP コンポーネントに自動的に挿入されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
```

```
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

269.2.2. 宛先設定

概要

宛先の設定は、SAP コンポーネントの **destinationDataStore** プロパティーで維持されます。このマップの各エントリーは、SAP インスタンスへの個別のアウトバウンド接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、URI 形式のセクションで説明されているように、宛先エンドポイント URI の **destinationName** コンポーネントで使用されます。

各エントリーの値は、アウトバウンド SAP 接続の設定を指定する宛先データ設定オブジェクト (**org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl**) です。

サンプル宛先設定

次の Blueprint XML コードは、サンプルの宛先を **quickstartDest** という名前で設定する方法を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
```

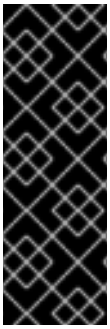
```
<blueprint ... >
   ...
   <!-- Create interceptor to support tRFC processing -->
   <bean id="currentProcessorDefinitionInterceptor"
         class="org.fusesource.camel.component.sap.CurrentProcessorDefinitionInterceptStrategy" />

   <!-- Configures the Inbound and Outbound SAP Connections -->
   <bean id="sap-configuration"
      class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
      <property name="destinationDataStore">
         <map>
            <entry key="quickstartDest" value-ref="quickstartDestinationData" />
         </map>
      </property>
   </bean>

   <!-- Configures an Outbound SAP Connection -->
   <!-- *** Please enter the connection property values for your environment *** -->
   <bean id="quickstartDestinationData"
      class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
      <property name="ashost" value="example.com" />
      <property name="sysnr" value="00" />
      <property name="client" value="000" />
      <property name="user" value="username" />
      <property name="passwd" value="password" />
      <property name="lang" value="en" />
   </bean>

</blueprint>
```

たとえば、前の Blueprint XML ファイルに示されているように宛先を設定した後、次の URI を使用して、**quickstartDest** 宛先で **BAPI_FLCUST_GETLIST** リモート関数呼び出しを呼び出すことができます。

```
sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST
```

tRFC および qRFC 宛先のインターセプター

前のサンプル宛先設定は、**CurrentProcessorDefinitionInterceptStrategy** オブジェクトのインスタンス化を示しています。このオブジェクトは、Camel ランタイムにインターセプターをインストールします。これにより、Camel SAP コンポーネントは、RFC トランザクションの処理中に Camel ルート内の位置を追跡できます。詳細は、「トランザクション RFC 宛先エンドポイント」 を参照してください。



重要

このインターセプターは、トランザクション RFC 宛先エンドポイント (**sap-trfc-destination** や **sap-qrfc-destination** など) にとって非常に重要であり、アウトバウンドトランザクション RFC 通信を適切に管理するには、Camel ランタイムにインストールする必要があります。ストラテジーが実行時に見つからない場合、Destination RFC Transaction Handlers は Camel ログに警告を発行します。この状況では、アウトバウンドトランザクション RFC 通信を適切に管理するために、Camel ランタイムを再プロビジョニングして再起動する必要があります。

ログオンと認証のオプション

次の表に、SAP 宛先データストアで宛先を設定するための **ログオンオプションと認証** オプションを示します。

名前	デフォルト値	説明
クライアント		SAP クライアント、必須ログオンパラメーター
user		ログオンユーザー、パスワードベース認証のログオンパラメーター
aliasUser		ログオンユーザーエイリアス。ログオンユーザーの代わりに使用できます。
userId		ABAP AS へのログオンに使用されるユーザー ID。宛先設定が認証に SSO/アサーションチケット、証明書、現在のユーザー、または SNC 環境を使用する場合、JCo ランタイムによって使用されます。ユーザーもユーザー別名も設定されていない場合、ユーザー ID は必須です。この ID は SAP バックエンドに送信されることはなく、JCo ランタイムによってローカルで使用されます。
passwd		ログオンパスワード、パスワードベースの認証のログオンパラメーター。
lang		ログオン言語。定義されていない場合は、デフォルトのユーザー言語が使用されます。
mysapsso2		指定した SAP Cookie バージョン 2 を SSO ベースの認証のログオンチケットとして使用します。
x509cert		証明書ベースの認証に指定された X509 証明書を使用します。
lcheck		最初の呼び出しまで認証を延期する - 1 (有効)。特別な場合にのみ使用されます。
useSapGui		表示または非表示の SAP GUI を使用するか、SAP GUI を使用しません。

codePage		ログオンパラメーターの変換に使用されるコードページを定義する追加のログオンパラメーター。特別な場合にのみ使用
getsso2		ログオン後に SSO チケットを注文します。取得したチケットは宛先属性で使用できます
denyInitialPassword		**1** に設定すると、初期パスワードを使用すると例外が発生します (デフォルトは **0** です)。

接続オプション

次の表に、SAP 宛先データストアで宛先を設定するための **接続** オプションを示します。

名前	デフォルト値	説明
saprouter		SAP ルーターの背後にあるシステムに接続するための SAP ルーター文字列。SAP ルーター文字列には、一連の SAP ルーターとそのポート番号が含まれており、次の形式になっています: **(/H/<host>[/S/<port>])+**
sysnr		SAP ABAP アプリケーションサーバーのシステム番号、直接接続に必須
ashost		SAP ABAP アプリケーションサーバー、直接接続に必須。
mshost		SAP メッセージサーバー、負荷分散接続の必須プロパティー。
msserv		SAP メッセージサーバーポート。負荷分散接続のオプションプロパティー。サービス名 sapmsXXX を解決するために、**etc/services** のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。

gwhost		アプリケーションサーバーへの接続を確立するために使用する具体的なゲートウェイを指定できます。指定しない場合、アプリケーションサーバーのゲートウェイが使用されます
gwserv		gwhost を使用する場合に設定する必要があります。そのゲートウェイで使用されるポートを指定できます。指定しない場合、アプリケーションサーバーのゲートウェイのポートが使用されます。サービス名 sapgwXXX を解決するために、etc/services のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。
r3name		SAP システムのシステム ID。負荷分散接続の必須プロパティー。
group		SAP アプリケーションサーバーのグループ、負荷分散接続の必須プロパティー

接続プールのオプション

次の表に、SAP 宛先データストアで宛先を設定するための **接続プール** オプションを示します。

名前	デフォルト値	説明
peakLimit	**0**	宛先に対して同時に作成できるアクティブなアウトバウンド接続の最大数。値が **0** の場合、アクティブな接続の数に制限はありません。それ以外の場合、値が **jpoolCapacity** の値よりも小さい場合は、この値に自動的に増加します。デフォルト設定は **poolCapacity** の値です。または、**poolCapacity** も指定されていない場合、デフォルトは **0** (無制限) です。

poolCapacity	**1**	宛先によって開いたままのアイドル状態のアウトバウンド接続の最大数。値 **0** は、接続プーリングがないという効果があります (デフォルトは **1** です)。
expirationTime		宛先によって内部的に保持されている空き接続を閉じることができるまでの時間 (ミリ秒)。
expirationPeriod		宛先が解放された接続の有効期限をチェックするまでのミリ秒単位の期間。
maxGetTime		アプリケーションによって最大許容数の接続がすでに割り当てられている場合に、接続を待機する最大時間 (ミリ秒)。

安全なネットワーク接続オプション

次の表に、SAP 宛先データストアで宛先を設定するための **セキュアなネットワーク** オプションを示します。

名前	デフォルト値	説明
sncMode		セキュアなネットワーク接続 (SNC) モード、**0** (オフ) または **1** (オン)
sncPartnername		SNC パートナー、例: **p:CN=R3, O=XYZ-INC, C=EN**
sncQop		セキュリティーの SNC レベル: **1** ~ **9**
sncMyname		独自の SNC 名。環境設定をオーバーライドします
sncLibrary		SNC サービスを提供するライブラリーへのパス

リポジトリーオプション

次の表に、SAP 宛先データストアで宛先を設定するための **リポジトリー** オプションを示します。

名前	デフォルト値	説明

repositoryDest		リポジトリーとして使用する宛先を指定します。
repositoryUser		リポジトリーの宛先が設定されておらず、このプロパティーが設定されている場合、リポジトリー呼び出しのユーザーとして使用されます。これにより、リポジトリーの検索に別のユーザーを使用できます。
repositoryPasswd		リポジトリーユーザーのパスワード。リポジトリーユーザーを使用する必要がある場合は必須です。
repositorySnc		**(オプション)** この宛先に SNC が使用されている場合、このプロパティーが **0** に設定されていれば、リポジトリー接続に対して SNC をオフにすることができます。デフォルト設定は **jco.client.snc**_mode の値です。特殊な場合のみ。

repositoryRoundtripOptimization		**RFC_METADATA_GET** API を有効にすると、リポジトリーデータが 1 回の往復で提供されます。 **1** ABAP システムで **RFC_METADATA_GET** の使用を有効にします。 **0** ABAP システムで **RFC_METADATA_GET** を無効化します。 プロパティーが設定されていない場合、宛先は最初にリモート呼び出しを実行して、**RFC_METADATA_GET** が使用可能かどうかを確認します。利用可能な場合、宛先はそれを使用します。 **注記:** リポジトリーがすでに初期化されている場合 (たとえば、他の宛先で使用されているため)、このプロパティーは効果がありません。通常、このプロパティーは ABAP システムに関連しており、同じ ABAP システムを指すすべての宛先で同じ値を持つ必要があります。バックエンドの前提条件については、ノート 1456826 を参照してください。

トレース設定オプション

次の表に、SAP 宛先データストアで宛先を設定するための **トレース設定** オプションを示します。

名前	デフォルト値	説明
trace		RFC トレースを有効/無効にする (**0** または **1**)
cpicTrace		CPIC トレースを有効/無効にする **[0..3]**

269.2.3. サーバー設定

概要

サーバーの設定は、SAP コンポーネントの **serverDataStore** プロパティーで維持されます。このマップの各エントリーは、SAP インスタンスからの個別のインバウンド接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、URI 形式のセクションで説明されているように、サーバーエ

ンドポイント URI の **serverName** コンポーネントで使用されます。

各エントリーの値は、インバウンド SAP 接続の設定を定義する **サーバーデータ設定オブジェクト、org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl** です。

例: サーバー設定

次の Blueprint XML コードは、名前が **quickstartServer** のサンプルサーバー設定を作成する方法を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

この例では、サーバーがリモート SAP インスタンスからメタデータを取得するために使用する宛先接続である **quickstartDest** も設定する方法に注目してください。この宛先は、**repositoryDestination** オ

プションを介してサーバーデータで設定されます。このオプションを設定しない場合は、代わりにローカルメタデータリポジトリーを作成する必要があります (「リポジトリーの設定」)。

たとえば、前述の Blueprint XML ファイルに示されているように宛先を設定した後、次の URI を使用して、呼び出し元のクライアントからの **BAPI_FLCUST_GETLIST** リモート関数呼び出しを処理できます。

```
sap-srfc-server:quickstartServer:BAPI_FLCUST_GETLIST
```

必須オプション

サーバーデータ設定オブジェクトに必要なオプションは次のとおりです。

名前	デフォルト値	説明
gwhost		サーバー接続を登録するゲートウェイホスト。
gwserv		登録を行うことができるポートであるゲートウェイサービス。サービス名 **sapgwXXX** を解決するために、**etc/services** のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。
progid		登録が行われたプログラム ID。ゲートウェイおよび ABAP システムの宛先で識別子として機能します。
repositoryDestination		リモート SAP サーバーでホストされているメタデータリポジトリーからメタデータを取得するためにサーバーが使用できる宛先名を指定します。
connectionCount		ゲートウェイに登録する必要がある接続の数。

安全なネットワーク接続オプション

サーバーデータ設定オブジェクトの安全なネットワーク接続オプションは次のとおりです。

名前	デフォルト値	説明

sncMode		セキュアなネットワーク接続 (SNC) モード、**0** (オフ) または **1** (オン)
sncQop		セキュリティーの SNC レベル、**1** ~ **9**
sncMyname		サーバーの SNC 名。デフォルトの SNC 名を上書きします。通常、**p:CN=JCoServer, O=ACompany, C=EN** のようなものです。
sncLib		SNC サービスを提供するライブラリーへのパス。このプロパティーが指定されていない場合は、代わりに **jco.middleware.snc_lib** プロパティーの値が使用されます。

その他のオプション

サーバーデータ設定オブジェクトのその他のオプションは次のとおりです。

名前	デフォルト値	説明
saprouter		その ABAP システムのゲートウェイでサーバーを登録するときに、ファイアウォールによって保護されているため、SAProuter を介してのみアクセスできるシステムに使用する SAP ルーター文字列。一般的なルーター文字列は **/H/firewall.hostname/H/** です。
maxStartupDelay		失敗した場合の 2 回の起動試行間の最大時間 (秒単位)。待機時間は、起動に失敗するたびに最初の 1 秒から 2 倍になり、最大値に達するか、サーバーが正常に起動できるようになります。
trace		RFC トレースを有効/無効にする (**0** または **1**)

workerThreadCount		サーバー接続で使用されるスレッドの最大数。設定されていない場合、**connectionCount** の値が **workerThreadCount** として使用されます。スレッドの最大数は 99 を超えることはできません。
workerThreadMinCount		サーバー接続で使用されるスレッドの最小数。設定されていない場合、**connectionCount** の値が **workerThreadMinCount** として使用されます。

269.2.4. リポジトリーの設定

概要

リポジトリーの設定は、SAP コンポーネントの **repositoryDataStore** プロパティーで維持されます。このマップの各エントリーは、個別のリポジトリーを設定します。各エントリーのキーはリポジトリーの名前であり、このキーはこのリポジトリーが接続されているサーバーの名前にも対応しています。

各エントリーの値は、メタデータリポジトリーのコンテンツを定義するリポジトリーデータ設定オブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl** です。リポジトリーデータオブジェクトは、機能テンプレート設定オブジェクト **org.fuesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl** のマップです。このマップの各エントリーは汎用モジュールのインタフェースを指定し、各エントリーのキーは指定された汎用モジュールの名前です。

リポジトリーデータの例

次のコードは、メタデータリポジトリーを設定する簡単な例を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the sap-srfc-server component -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="repositoryDataStore">
            <map>
                <entry key="nplServer" value-ref="nplRepositoryData" />
            </map>
        </property>
    </bean>

    <!-- Configures a Meta-Data Repository -->
    <bean id="nplRepositoryData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl">
        <property name="functionTemplates">
            <map>
                <entry key="BOOK_FLIGHT" value-ref="bookFlightFunctionTemplate" />
            </map>
```

```
        </property>
    </bean>
    ...
</blueprint>
```

関数テンプレートのプロパティー

汎用モジュールのインタフェースは、RFC コールでデータが汎用モジュールとの間でやり取りされる 4 つのパラメーターリストで設定されています。各パラメーターリストは 1 つ以上のフィールドで設定され、それぞれが RFC コールで転送される名前付きパラメーターです。次のパラメーターリストと例外リストがサポートされています。

- **インポートパラメーターリスト** には、RFC コールで汎用モジュールに送信されるパラメーター値が含まれています。
- **エクスポートパラメーターリスト** には、RFC コールで汎用モジュールによって返されるパラメーター値が含まれています。
- **変更パラメーター一覧** には、RFC コールで汎用モジュールとの間で送受信されるパラメーター値が含まれています。
- **テーブルパラメーター一覧** には、RFC コールで汎用モジュールとの間で送受信される内部テーブル値が含まれています。
- 汎用モジュールのインタフェースは、モジュールが RFC コールで呼び出されたときに発生する可能性のある ABAP 例外の **例外リスト** からも設定されます。

関数テンプレートは、関数インターフェイスの各パラメーターリスト内のパラメーターの名前と型、および関数によって出力される ABAP 例外を記述します。関数テンプレートオブジェクトは、次の表に示すように、メタデータオブジェクトの 5 つのプロパティーリストを保持します。

プロパティー	説明
importParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールへの RFC 呼び出しで送信されるパラメーターを指定します。
changingParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールとの間の RFC 呼び出しで送受信されるパラメーターを指定します。
exportParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールからの RFC 呼び出しで返されるパラメーターを指定します。

tableParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールとの間の RFC 呼び出しで送受信されるテーブルパラメーターを指定します。
exceptionList	ABAP 例外メタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.AbapExceptionImpl**。汎用モジュールの RFC 呼び出しで潜在的に発生する ABAP 例外を指定します。

関数テンプレートの例

次の例は、関数テンプレートを設定する方法の概要を示しています。

```
<bean id="bookFlightFunctionTemplate"
    class="org.fusesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl">
    <property name="importParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="changingParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exportParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="tableParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exceptionList">
        <list>
            ...
        </list>
    </property>
</bean>
```

フィールドのメタデータプロパティーを一覧表示する

リストフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl** は、パラメーターリスト内のフィールドの名前とタイプを指定します。基本パラメーターフィールド

(**CHAR**、**DATE**、**BCD**、**TIME**、**BYTE**、**NUM**、**FLOAT**、**INT**、**INT1**、**INT2**、**DECF16**、**DECF34**、**STRING**、**XSTRING**) の場合、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを次の表に示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。
byteLength	-	非 Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
unicodeByteLength	-	Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
decimals	**0**	フィールド値の小数点以下の桁数。パラメータータイプ BCD および FLOAT にのみ必要です。「RFC のメッセージボディー」 を参照してください。
任意	**false**	**true** の場合、フィールドはオプションであり、RFC 呼び出しで設定する必要はありません

すべての基本パラメーターフィールドでは、フィールドメタデータオブジェクトで **name**、**type**、**byteLength**、および **unicodeByteLength** プロパティーを指定する必要があることに注意してください。さらに、**BCD**、**FLOAT**、**DECF16**、および **DECF34** フィールドでは、フィールドメタデータオブジェクトで decimal プロパティーを指定する必要があります。

タイプ **TABLE** または **STRUCTURE** の複雑なパラメーターフィールドの場合、次の表に、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ

recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** が渡され、構造またはテーブル行のフィールドが指定されます。
任意	**false**	**true** の場合、フィールドはオプションであり、RFC 呼び出しで設定する必要はありません

すべての複雑なパラメーターフィールドでは、フィールドメタデータオブジェクトで **name**、**type**、および **recordMetaData** プロパティーを指定する必要があることに注意してください。**recordMetaData** プロパティーの値は、レコードフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** であり、ネストされた構造の構造またはテーブル行の構造を指定します。

基本リストフィールドのメタデータの例

次のメタデータ設定では、オプションの **TICKET_PRICE** という名前の小数点以下 2 桁の 24 桁のパックされた BCD 数値パラメーターを指定します。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
    <property name="name" value="TICKET_PRICE" />
    <property name="type" value="BCD" />
    <property name="byteLength" value="12" />
    <property name="unicodeByteLength" value="24" />
    <property name="decimals" value="2" />
    <property name="optional" value="true" />
</bean>
```

複雑なリストフィールドのメタデータの例

次のメタデータ設定では、**connectionInfo** レコードメタデータオブジェクトによって行構造が指定された、**CONNINFO** という名前の必須の **TABLE** パラメーターを指定します。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
    <property name="name" value="CONNINFO" />
    <property name="type" value="TABLE" />
    <property name="recordMetaData" ref="connectionInfo" />
</bean>
```

メタデータのプロパティーを記録する

レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** は、ネストされた **STRUCTURE** または **TABLE** パラメーターの行の名前と内容を指定します。レコードメタデータオブジェクトは、ネストされた構造またはテーブル行にあるパラメーターを指定するレコードフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl** のリストを維持します。

次の表に、レコードメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	レコードの名前。
recordFieldMetaData	-	レコードフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**。構造体に含まれるフィールドを指定します。

注記

レコードメタデータオブジェクトのすべてのプロパティーが必要です。

レコードのメタデータの例

次の例は、レコードメタデータオブジェクトを設定する方法を示しています。

```
<bean    id="connectionInfo"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl">
    <property name="name" value="CONNECTION_INFO" />
    <property name="recordFieldMetaData">
        <list>
            ...
        </list>
    </property>
</bean>
```

レコードフィールドのメタデータプロパティー

レコードフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl** は、構造体でパラメーターフィールドの名前とタイプを指定します。

レコードフィールドメタデータオブジェクトは、ネストされた構造またはテーブル行内の個々のフィールド位置のオフセットを追加で指定する必要があることを除いて、パラメーターフィールドメタデータオブジェクトに似ています。個々のフィールドの非 Unicode オフセットと Unicode オフセットは、構造体または行内の前のフィールドの非 Unicode バイト長と Unicode バイト長の合計から計算して指定する必要があります。ネストされた構造およびテーブル行のフィールドのオフセットを適切に指定しないと、基礎となる JCo および ABAP ランタイムのパラメーターのフィールドストレージが重複し、RFC コールでの値の適切な転送が妨げられることに注意してください。

基本パラメーターフィールド (**CHAR**、**DATE**、**BCD**、**TIME**、**BYTE**、**NUM**、**FLOAT**、**INT**、**INT1**、**INT2**、**DECF16**、**DECF34**、**STRING**、**XSTRING**) の場合、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを次の表に示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ
byteLength	-	非 Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
unicodeByteLength	-	Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
byteOffset	-	非 Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲まれている構造内のフィールドのバイト位置です。
unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲まれている構造内のフィールドのバイト位置です。
decimals	**0**	フィールド値の小数点以下の桁数。パラメータータイプ **BCD** および **FLOAT** にのみ必要です。「RFC のメッセージボディー」 を参照してください。

タイプ **TABLE** または **STRUCTURE** の複雑なパラメーターフィールドの場合、次の表に、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ
byteOffset	-	非 Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲まれている構造内のフィールドのバイト位置です。

unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲んでいる構造内のフィールドのバイト位置です。
recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** が渡され、構造またはテーブル行のフィールドが指定されます。

基本レコードフィールドのメタデータの例

次のメタデータ設定は、**ARRDATE** という名前の **DATE** フィールドパラメーターを指定します。これは、非 Unicode レイアウトの場合は囲んでいる構造の 85 バイト、Unicode レイアウトの場合は囲んでいる構造の 170 バイトに配置されています。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
    <property name="name" value="ARRDATE" />
    <property name="type" value="DATE" />
    <property name="byteLength" value="8" />
    <property name="unicodeByteLength" value="16" />
    <property name="byteOffset" value="85" />
    <property name="unicodeByteOffset" value="170" />
</bean>
```

複雑なレコードフィールドのメタデータの例

次のメタデータ設定は、**flightInfo** レコードメタデータオブジェクトによって指定された構造を持つ **FLTINFO** という名前の **STRUCTURE** フィールドパラメーターを指定します。パラメーターは、非 Unicode レイアウトと Unicode レイアウトの両方の場合に、囲んでいる構造の先頭に配置されます。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
    <property name="name" value="FLTINFO" />
    <property name="type" value="STRUCTURE" />
    <property name="byteOffset" value="0" />
    <property name="unicodeByteOffset" value="0" />
    <property name="recordMetaData" ref="flightInfo" />
</bean>
```

269.3. メッセージヘッダー

SAP コンポーネントは、次のメッセージヘッダーをサポートしています。

ヘッダー	説明

CamelSap.scheme	メッセージを処理する最後のエンドポイントの URI スキーム。以下のいずれかの値を使用します。 **sap-srfc-destination** **sap-trfc-destination** **sap-qrfc-destination** **sap-srfc-server** **sap-trfc-server** **sap-idoc-destination** **sap-idoclist-destination** **sap-qidoc-destination** **sap-qidoclist-destination** **sap-idoclist-server**
CamelSap.destinationName	メッセージを処理する最後の宛先エンドポイントの宛先名。
CamelSap.serverName	メッセージを処理する最後のサーバーエンドポイントのサーバー名。
CamelSap.queueName	メッセージを処理する最後のキューエンドポイントのキュー名。
CamelSap.rfcName	メッセージを処理する最後の RFC エンドポイントの RFC 名。
CamelSap.idocType	メッセージを処理する最後の IDoc エンドポイントの IDoc タイプ。
CamelSap.idocTypeExtension	メッセージを処理する最後の IDoc エンドポイントの IDoc タイプ拡張 (存在する場合)。
CamelSap.systemRelease	メッセージを処理する最後の IDoc エンドポイントのシステムリリース (存在する場合)。
CamelSap.applicationRelease	メッセージを処理する最後の IDoc エンドポイントのアプリケーションリリース (存在する場合)。

269.4. エクスチェンジプロパティー

SAP コンポーネントは、次のエクスチェンジプロパティーを追加します。

プロパティー	説明
CamelSap.destinationPropertiesMap	エクスチェンジによって検出された各 SAP 宛先のプロパティーを含むマップ。マップは宛先名によってキー付けされ、各エントリーはその宛先の設定プロパティーを含む **java.util.Properties** オブジェクトです。
CamelSap.serverPropertiesMap	エクスチェンジによって検出された各 SAP サーバーのプロパティーを含むマップ。マップはサーバー名でキー付けされ、各エントリーはそのサーバーの設定プロパティーを含む **java.util.Properties** オブジェクトです。

269.5. RFC のメッセージボディー

リクエストおよびレスポンスオブジェクト

SAP エンドポイントは、SAP リクエストオブジェクトを含むメッセージボディーを含むメッセージを受信することを想定しており、SAP レスポンスオブジェクトを含むメッセージボディーを含むメッセージを返します。SAP のリクエストとレスポンスは、各フィールドが事前定義されたデータ型を持つ名前付きフィールドを含む固定マップデータ構造です。

SAP リクエストとレスポンスの名前付きフィールドは、SAP エンドポイントに固有であり、各エンドポイントが受け入れる SAP リクエストとレスポンスのパラメーターを定義することに注意してください。SAP エンドポイントは、それに固有のリクエストとレスポンスオブジェクトを作成するファクトリーメソッドを提供します。

```
public class SAPEndpoint ... {
    ...
    public Structure getRequest() throws Exception;

    public Structure getResponse() throws Exception;
    ...
}
```

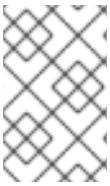
構造物オブジェクト

SAP 要求オブジェクトと応答オブジェクトは両方とも、**org.fusesource.camel.component.sap.model.rfc.Structure** インターフェイスをサポートする構造オブジェクトとして Java で表されます。このインターフェイスは、**java.util.Map** インターフェイスと **org.eclipse.emf.ecore.EObject** インターフェイスの両方を拡張します。

```
public interface Structure extends org.eclipse.emf.ecore.EObject,
                                   java.util.Map<String, Object> {

    <T> T get(Object key, Class<T> type);

}
```

構造体オブジェクトのフィールド値には、マップインターフェイスのフィールドの getter メソッドを介してアクセスします。さらに、構造体インターフェイスは、フィールド値を取得するための型制限されたメソッドを提供します。

構造オブジェクトは、Eclipse モデリングフレームワーク (EMF) を使用してコンポーネントランタイムに実装され、そのフレームワークの **EObject** インターフェイスをサポートします。構造オブジェクトのインスタンスには、それが提供するフィールドのマップの構造と内容を定義および制限するメタデータが添付されています。このメタデータには、EMF が提供する標準的な方法を使用してアクセスし、イントロスペクションすることができます。詳細については、EMF のドキュメントを参照してください。



注記

構造体オブジェクトで定義されていないパラメーターを取得しようとすると、null が返されます。構造体で定義されていないパラメーターを設定しようとすると、例外が出力され、パラメーターの値を正しくない型で設定しようとします。

次のセクションで説明するように、構造体オブジェクトには、複合フィールド型 **STRUCTURE** および **TABLE** の値を含むフィールドを含めることができます。これらの型のインスタンスを作成して構造体に追加する必要はないことに注意してください。これらのフィールド値のインスタンスは、必要に応じて、囲んでいる構造にアクセスするときにオンデマンドで作成されます。

フィールドの種類

SAP リクエストまたはレスポンスの構造オブジェクト内に存在するフィールドは、**基本** または **複合** のいずれかです。基本フィールドには単一のスカラー値が含まれますが、複合フィールドには基本タイプまたは複合タイプのフィールドが 1 つ以上含まれます。

基本フィールドの種類

基本フィールドは、文字、数値、16 進数、または文字列フィールドタイプのいずれかです。次の表は、構造体オブジェクトに存在する可能性のある基本フィールドのタイプをまとめたものです。

フィールドタイプ	対応 Java 型	Byte Length	Unicode バイト長	数 小数 桁	説明
CHAR	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP タイプ 'C': 固定サイズの文字列。
DATE	**java.util.Date**	8	16	-	ABAP タイプ 'D': 日付 (形式: YYYYMMDD)。
BCD	**java.math.BigDecimal**	1 から 16	1 から 16	0 から 14	ABAP タイプ 'P': パックされた BCD 番号。BCD 番号には、1 バイトあたり 2 桁が含まれます。

TIME	**java.util.Date**	6	12	-	ABAP タイプ 'T': 時間 (形式: HHMMSS)。
BYTE	**byte[]**	1 から 65535	1 から 65535	-	ABAP タイプ 'X': 固定サイズのバイト配列。
NUM	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP タイプ 'N': 固定サイズの数値文字列。
FLOAT	**java.lang.Double**	8	8	0 から 15	ABAP タイプ 'F': 浮動小数点数。
INT	**java.lang.Integer**	4	4	-	ABAP タイプ 'I': 4 バイト整数。
INT2	**java.lang.Integer**	2	2	-	ABAP タイプ 'S': 2 バイト整数。
INT1	**java.lang.Integer**	1	1	-	ABAP タイプ 'B': 1 バイト整数。
DECF16	**java.match.BigDecimal**	8	8	16	ABAP タイプ 'decfloat16': 8 バイトの 10 進浮動小数点数。
DECF34	**java.math.BigDecimal**	16	16	34	ABAP タイプ 'decfloat34': 16 バイトの 10 進浮動小数点数。
STRING	**java.lang.String**	8	8	-	ABAP タイプ 'G': 可変長文字列。
XSTRING	**byte[]**	8	8	-	ABAP タイプ 'Y': 可変長バイト配列。

文字フィールドの種類

文字フィールドには、基礎となる JCo および ABAP ランタイムで非 Unicode または Unicode 文字エンコーディングを使用できる固定サイズの文字列が含まれます。非 Unicode 文字列は、1 バイトあたり 1 文字をエンコードします。Unicode 文字列は、UTF-16 エンコーディングを使用して 2 バイトでエンコードされます。文字フィールドの値は、Java では **java.lang.String** オブジェクトとして表され、基礎となる JCo ランタイムが ABAP 表現への変換を担当します。

文字フィールドは、関連する **byteLength** および **unicodeByteLength** プロパティーでそのフィールド長を宣言します。これらのプロパティーは、各エンコーディングシステムでのフィールドの文字列の長さを決定します。

CHAR

CHAR 文字項目は、英数字を含むテキスト項目であり、ABAP タイプ C に対応します。

NUM

NUM 文字フィールドは、数字のみを含む数値テキストフィールドであり、ABAP タイプ N に対応します。

DATE

DATE 文字フィールドは、年、月、日が **YYYYMMDD** としてフォーマットされた 8 文字の日付フィールドであり、ABAP タイプ D に対応します。

TIME

TIME 文字フィールドは、時、分、および秒が **HHMMSS** としてフォーマットされた 6 文字の時間フィールドであり、ABAP タイプ T に対応します。

数値フィールドの種類

数値フィールドには数値が含まれています。次の数値フィールドタイプがサポートされています。

INT

INT 数値フィールドは、基礎となる JCo および ABAP ランタイムで 4 バイトの整数値として格納される整数フィールドであり、ABAP タイプ I に対応します。**INT** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

INT2

INT2 数値フィールドは、基礎となる JCo および ABAP ランタイムに 2 バイトの整数値として格納される整数フィールドであり、ABAP タイプ S に対応します。**INT2** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

INT1

INT1 フィールドは、基になる JCo および ABAP ランタイム値に 1 バイトの整数値として格納される整数フィールドであり、ABAP タイプ B に対応します。**INT1** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

FLOAT

FLOAT フィールドは、基礎となる JCo および ABAP ランタイムに 8 バイトの double 値として格納される 2 進浮動小数点数フィールドであり、ABAP タイプ F に対応します。**FLOAT** フィールドは、フィールドの値が関連する小数プロパティー。**FLOAT** フィールドの場合、この 10 進プロパティーは 1 ~ 15 桁の値を持つことができます。**FLOAT** フィールド値は、Java では **java.lang.Double** オブジェクトとして表されます。

BCD

BCD フィールドは、基礎となる JCo および ABAP ランタイムで 1 から 16 バイトのパック数として保管される 2 進化 10 進フィールドであり、ABAP タイプ P に対応します。パック数は、1 バイトあたり 2 桁の 10 進数を保管します。**BCD** フィールドは、関連する **byteLength** および

unicodeByteLength プロパティーでフィールド長を宣言します。**BCD** フィールドの場合、これらのプロパティーは 1 ~ 16 バイトの値を持つことができ、両方のプロパティーが同じ値になります。**BCD** フィールドは、関連付けられた decimal プロパティーで、フィールドの値に含まれる 10 進数の桁数を宣言します。**BCD** フィールドの場合、この 10 進プロパティーは 1 ~ 14 桁の値を持つことができます。**BCD** フィールド値は、Java では **java.math.BigDecimal** として表されます。

DECF16

DECF16 フィールドは、基礎となる JCo および ABAP ランタイムで 8 バイトの IEEE 754 decimal64 浮動小数点値として格納される 10 進浮動小数点であり、ABAP タイプ **decfloat16** に対応します。**DECF16** フィールドの値は、10 進数で 16 桁です。 **DECF16** フィールドの値は、Java では **java.math.BigDecimal** として表されます。

DECF34

DECF34 フィールドは、基礎となる JCo および ABAP ランタイムで 16 バイトの IEEE 754 decimal128 浮動小数点値として格納される 10 進浮動小数点であり、ABAP タイプ **decfloat34** に対応します。**DECF34** フィールドの値には、34 桁の 10 進数があります。 **DECF34** フィールドの値は、Java では **java.math.BigDecimal** として表されます。

16 進フィールドタイプ

16 進数フィールドには生のバイナリーデータが含まれます。次の 16 進数フィールドタイプがサポートされています。

BYTE

BYTE フィールドは、基礎となる JCo および ABAP ランタイムにバイト配列として格納される固定サイズのバイト文字列であり、ABAP タイプ X に対応します。**BYTE** フィールドは、関連する **byteLength** および **unicodeByteLength** プロパティーでフィールド長を宣言します。**BYTE** フィールドの場合、これらのプロパティーは 1 ~ 65535 バイトの値を持つことができ、両方のプロパティーが同じ値になります。**BYTE** フィールドの値は、Java では **byte** オブジェクトとして表されます。

文字列フィールドの種類

文字列フィールドは、可変長の文字列値を参照します。その文字列値の長さは実行時まで固定されません。文字列値のストレージは、基礎となる JCo および ABAP ランタイムで動的に作成されます。文字列フィールド自体のストレージは固定されており、文字列ヘッダーのみが含まれています。

STRING

STRING フィールドは文字列を参照し、基礎となる JCo および ABAP ランタイムに 8 バイト値として格納されます。ABAP タイプ G に対応します。**STRING** 項目の値は、Java では **java.lang.String** オブジェクトとして表されます。

XSTRING

XSTRING フィールドはバイト文字列を参照し、基礎となる JCo および ABAP ランタイムに 8 バイト値として格納されます。ABAP タイプ Y に対応します。**STRING** 項目の値は、Java では **byte** オブジェクトとして表されます。

複雑なフィールドタイプ

複合フィールドは、構造体またはテーブルフィールドタイプのいずれかです。次の表は、これらの複雑なフィールドタイプをまとめたものです。

フィールドタイプ	対応 Java 型	Byte Length	Unicode バイト長	数 小数 桁	説明

STRUCTURE	**org.fusesource.camel.component.sap.model.rfc.Structure**	個々のフィールドのバイト長の合計	個々のフィールドの Unicode バイト長の合計	-	ABAP タイプ 'u' & 'v': 異種構造
TABLE	**org.fusesource.camel.component.sap.model.rfc.Table**	行構造体のバイト長	行構造体の Unicode バイト長	-	ABAP タイプ 'h': テーブル

構造体フィールドタイプ

STRUCTURE 項目には構造オブジェクトが含まれ、基礎となる JCo および ABAP ランタイムに ABAP 構造レコードとして格納されます。ABAP タイプ **u** または **v** のいずれかに対応します。**STRUCTURE** 項目の値は、Java ではインターフェイス **org.fusesource.camel.component.sap.model.rfc.Structure** を持つ構造オブジェクトとして表されます。

テーブルフィールドタイプ

TABLE フィールドにはテーブルオブジェクトが含まれ、基礎となる JCo および ABAP ランタイムに ABAP 内部テーブルとして格納されます。ABAP タイプ **h** に対応します。フィールドの値は、インタフェース **org.fusesource.camel.component.sap.model.rfc.Table** を持つテーブルオブジェクトによって Java で表されます。

テーブルオブジェクト

テーブルオブジェクトは、同じ構造を持つ構造オブジェクトの行を含む同種のリストデータ構造です。このインターフェイスは、**java.util.List** インターフェイスと **org.eclipse.emf.ecore.EObject** インターフェイスの両方を拡張します。

```
public interface Table<S extends Structure>
    extends org.eclipse.emf.ecore.EObject,
    java.util.List<S> {

    /**
     * Creates and adds table row at end of row list
     */
    S add();

    /**
     * Creates and adds table row at index in row list
     */
    S add(int index);

}
```

テーブルオブジェクト内の行のリストは、リストインターフェイスで定義された標準メソッドを使用してアクセスおよび管理されます。さらに、テーブルインターフェイスは、構造体オブジェクトを作成して行リストに追加するための 2 つのファクトリーメソッドを提供します。

テーブルオブジェクトは、Eclipse Modeling Framework (EMF) を使用してコンポーネントランタイムに実装され、そのフレームワークの EObject インターフェイスをサポートします。テーブルオブジェクトのインスタンスには、それが提供する行の構造と内容を定義および制限するメタデータが添付されています。このメタデータには、EMF が提供する標準的な方法を使用してアクセスし、イントロスペクションすることができます。詳細については、EMF のドキュメントを参照してください。

注記

間違った型の行構造値を追加または設定しようとすると、例外が出力されます。

269.6. IDOC のメッセージ本文

IDoc メッセージタイプ

IDoc Camel SAP エンドポイントの 1 つを使用する場合、メッセージ本文のタイプは、使用している特定のエンドポイントによって異なります。

sap-idoc-destination エンドポイントまたは **sap-qidoc-destination** エンドポイントの場合、メッセージ本文は **Document** タイプです。

```
org.fusesource.camel.component.sap.model.idoc.Document
```

sap-idoclist-destination エンドポイント、**sap-qidoclist-destination** エンドポイント、または **sap-idoclist-server** エンドポイントの場合、メッセージボディーは **DocumentList** タイプです。

```
org.fusesource.camel.component.sap.model.idoc.DocumentList
```

IDoc 文書モデル

Camel SAP コンポーネントの場合、IDoc ドキュメントは、基礎となる SAP IDoc API のラッパー API を提供する Eclipse Modeling Framework (EMF) を使用してモデル化されます。このモデルで最も重要なタイプは次のとおりです。

```
org.fusesource.camel.component.sap.model.idoc.Document
org.fusesource.camel.component.sap.model.idoc.Segment
```

Document タイプは、IDoc ドキュメントインスタンスを表します。概説すると、**Document** インターフェイスは次のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Document extends EObject {
    // Access the field values from the IDoc control record
    String getArchiveKey();
    void setArchiveKey(String value);
    String getClient();
    void setClient(String value);
    ...

    // Access the IDoc document contents
    Segment getRootSegment();
}
```

次の種類のメソッドが **Document** インターフェイスによって公開されます。

制御レコードにアクセスする方法

ほとんどのメソッドは、IDoc 制御レコードのフィールド値にアクセスまたは変更するためのものです。これらのメソッドは、**AttributeName**、**AttributeName** の形式です。ここで、**AttributeName** はフィールド値の名前です (を参照してください)。表269.2「IDoc ドキュメントの属性」)。

ドキュメントコンテンツへのアクセス方法

getRootSegment メソッドは、ドキュメントコンテンツ (IDoc データレコード) へのアクセスを提供し、コンテンツを **Segment** オブジェクトとして返します。各 **Segment** オブジェクトには任意の数の子セグメントを含めることができ、セグメントは任意の程度にネストできます。
ただし、セグメント階層の正確なレイアウトは、文書の特定の **IDoc タイプ** によって定義されることに注意してください。したがって、セグメント階層を作成 (または読み取り) するときは、IDoc タイプによって定義された正確な構造に従う必要があります。

Segment タイプは、IDoc ドキュメントのデータレコードにアクセスするために使用されます。セグメントは、ドキュメントの IDoc タイプによって定義された構造に従って配置されます。概説すると、**Segment** インターフェイスは次のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Segment extends EObject, java.util.Map<String, Object> {
    // Returns the value of the '<em><b>Parent</b></em>' reference.
    Segment getParent();

    // Return a immutable list of all child segments
    <S extends Segment> EList<S> getChildren();

    // Returns a list of child segments of the specified segment type.
    <S extends Segment> SegmentList<S> getChildren(String segmentType);

    EList<String> getTypes();

    Document getDocument();

    String getDescription();

    String getType();

    String getDefinition();

    int getHierarchyLevel();

    String getIdocType();

    String getIdocTypeExtension();

    String getSystemRelease();

    String getApplicationRelease();

    int getNumFields();
```

```
    long getMaxOccurrence();

    long getMinOccurrence();

    boolean isMandatory();

    boolean isQualified();

    int getRecordLength();

    <T> T get(Object key, Class<T> type);
}
```

getChildren (String segmentType) メソッドは、新しい (ネストされた) 子をセグメントに追加する場合に特に便利です。次のように定義されているタイプ **SegmentList** のオブジェクトを返します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface SegmentList<S extends Segment> extends EObject, EList<S> {
    S add();

    S add(int index);
}
```

したがって、**E1SCU_CRE** タイプのデータレコードを作成するには、次のような Java コードを使用できます。

```
Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();
```

IDoc と Document オブジェクトの関係

SAP ドキュメントによると、IDoc ドキュメントは次の主要部分で設定されています。

制御記録

コントロールレコード (IDoc ドキュメントのメタデータを含む) は、**Document** オブジェクトの属性によって表されます。詳細については、表269.2「IDoc ドキュメントの属性」 を参照してください。

データ記録

データレコードは、セグメントのネストされた階層として構築される **Segment** オブジェクトによって表されます。**Document.getRootSegment** メソッドを介してルートセグメントにアクセスできます。

状況記録

Camel SAP コンポーネントでは、ステータスレコードはドキュメントモデルによって表され **ません**。ただし、制御レコードの **status** 属性を介して最新のステータス値にアクセスできます。

Document インスタンスの作成例

例えば、例269.1「Java での IDoc ドキュメントの作成」 Java で IDoc モデル API を使用して、IDoc タイプ **FLCUSTOMER_CREATEFROMDATA01** で IDoc ドキュメントを作成する方法を示します。

例269.1 Java での IDoc ドキュメントの作成

```
// Java
import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.util.IDocUtil;

import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.DocumentList;
import org.fusesource.camel.component.sap.model.idoc.IdocFactory;
import org.fusesource.camel.component.sap.model.idoc.IdocPackage;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.model.idoc.SegmentChildren;
...
//
// Create a new IDoc instance using the modelling classes
//

// Get the SAP Endpoint bean from the Camel context.
// In this example, it's a 'sap-idoc-destination' endpoint.
SapTransactionalIDocDestinationEndpoint endpoint =
    exchange.getContext().getEndpoint(
        "bean:SapEndpointBeanID",
        SapTransactionalIDocDestinationEndpoint.class
    );

// The endpoint automatically populates some required control record attributes
Document document = endpoint.createDocument()

// Initialize additional control record attributes
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
document.setRecipientPartnerNumber("QUICKCLNT");
document.setRecipientPartnerType("LS");
document.setSenderPartnerNumber("QUICKSTART");
document.setSenderPartnerType("LS");

Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();

Segment E1BPSCUNEW_Segment =
E1SCU_CRE_Segment.getChildren("E1BPSCUNEW").add();
E1BPSCUNEW_Segment.put("CUSTNAME", "Fred Flintstone");
E1BPSCUNEW_Segment.put("FORM", "Mr.");
E1BPSCUNEW_Segment.put("STREET", "123 Rubble Lane");
E1BPSCUNEW_Segment.put("POSTCODE", "01234");
E1BPSCUNEW_Segment.put("CITY", "Bedrock");
E1BPSCUNEW_Segment.put("COUNTR", "US");
E1BPSCUNEW_Segment.put("PHONE", "800-555-1212");
E1BPSCUNEW_Segment.put("EMAIL", "fred@bedrock.com");
E1BPSCUNEW_Segment.put("CUSTTYPE", "P");
E1BPSCUNEW_Segment.put("DISCOUNT", "005");
E1BPSCUNEW_Segment.put("LANGU", "E");
```

ドキュメント属性

表269.2「IDoc ドキュメントの属性」 **Document** オブジェクトに設定できる制御レコード属性を示します。

表269.2 IDoc ドキュメントの属性

属性	長さ	SAP フィールド	説明
archiveKey	70	**ARCKEY**	EDI アーカイブキー
クライアント	3	**MANDT**	クライアント
creationDate	8	**CREDAT**	IDoc が作成された日付
creationTime	6	**CRETIM**	IDoc が作成された時間
direction	1	**DIRECT**	方向
eDIMessage	14	**REFMES**	メッセージ参照
eDIMessageGroup	14	**REFGRP**	メッセージグループへの参照
eDIMessageType	6	**STDMES**	EDI メッセージタイプ
eDIStandardFlag	1	**STD**	EDI 規格
eDIStandardVersion	6	**STDVRS**	EDI 規格のバージョン
eDITransmissionFile	14	**REFINT**	エクスチェンジファイルへの参照
iDocCompoundType	8	**DOCTYP**	IDoc タイプ
iDocNumber	16	**DOCNUM**	IDoc 番号
iDocSAPRelease	4	**DOCREL**	IDoc の SAP リリース
iDocType	30	**IDOCTP**	基本 IDoc タイプの名前
iDocTypeExtension	30	**CIMTYP**	拡張タイプの名前
messageCode	3	**MESCOD**	論理メッセージコード
messageFunction	3	**MESFCT**	論理メッセージ機能
messageType	30	**MESTYP**	論理メッセージタイプ

属性	長さ	SAP フィールド	説明
outputMode	1	**OUTMOD**	出力モード
recipientAddress	10	**RCVSAD**	受信者アドレス (SADR)
recipientLogicalAddress	70	**RCVLAD**	受信者の論理アドレス
recipientPartnerFunction	2	**RCVPFC**	受信者のパートナー機能
recipientPartnerNumber	10	**RCVPRN**	受信機のパートナー番号
recipientPartnerType	2	**RCVPRT**	受信者のパートナータイプ
recipientPort	10	**RCVPOR**	受信側ポート (SAP システム、EDI サブシステム)
senderAddress		**SNDSAD**	送信者アドレス (SADR)
senderLogicalAddress	70	**SNDLAD**	送信者の論理アドレス
senderPartnerFunction	2	**SNDPFC**	差出人のパートナー機能
senderPartnerNumber	10	**SNDPRN**	送信者のパートナー番号
senderPartnerType	2	**SNDPRT**	送信者のパートナータイプ
senderPort	10	**SNDPOR**	送信側ポート (SAP システム、EDI サブシステム)
serialization	20	**SERIAL**	EDI/ALE: シリアル化フィールド
status	2	**STATUS**	IDoc のステータス
testFlag	1	**テスト**	テストフラグ

Java でドキュメント属性を設定する

Java で制御レコード属性を設定する場合 (から表269.2「IDoc ドキュメントの属性」)、Java Bean プロパティーの通常の規則に従います。つまり、属性値を取得および設定するために、**getName** および **setName** メソッドを介して **name** 属性にアクセスできます。たとえば、**iDocType**、**iDocTypeExtension**、および **messageType** 属性は、**Document** オブジェクトで次のように設定できます。

```
// Java
document.setIDocType("FLCUSTOMER_CREATEFROMDATA01");
document.setIDocTypeExtension("");
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
```

XML でドキュメント属性を設定する

XML で制御レコード属性を設定する場合、**idoc:Document** 要素に属性を設定する必要があります。たとえば、**iDocType**、**iDocTypeExtension** および **messageType** 属性は次のように設定できます。

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document ...
            iDocType="FLCUSTOMER_CREATEFROMDATA01"
            iDocTypeExtension=""
            messageType="FLCUSTOMER_CREATEFROMDATA" ... >
   ...
</idoc:Document>
```

269.7. トランザクションサポート

BAPI トランザクションモデル

SAP コンポーネントは、SAP とのアウトバウンド通信用に BAPI トランザクションモデルをサポートしています。**true** に設定されたトランザクションオプションを含む URL を持つ宛先エンドポイントは、必要に応じて、エンドポイントのアウトバウンド接続でステートフルセッションを開始し、Camel Synchronization オブジェクトをエクスチェンジに登録します。この Synchronization オブジェクトは、BAPI サービスメソッド **BAPI_TRANSACTION_COMMIT** を呼び出し、メッセージエクスチェンジの処理が完了するとステートフルセッションを終了します。メッセージエクスチェンジの処理が失敗した場合、Synchronization オブジェクトは BAPI サーバーメソッド **BAPI_TRANSACTION_ROLLBACK** を呼び出し、ステートフルセッションを終了します。

RFC トランザクションモデル

tRFC プロトコルは、一意のトランザクション識別子 (TID) で各トランザクション要求を識別することにより、AT-MOST-ONCE の配信と処理の保証を実現します。TID は、プロトコルで送信される各要求に付随します。tRFC プロトコルを使用する送信側アプリケーションは、要求を送信するときに、要求の各インスタンスを一意の TID で識別する必要があります。アプリケーションは、特定の TID を持つリクエストを複数回送信できますが、プロトコルは、リクエストが受信側システムで配信され、処理されるのは多くても 1 回であることを保証します。アプリケーションは、要求の送信時に通信エラーまたはシステムエラーが発生した場合に、指定された TID を使用してリクエストを再送信することを選択できます。したがって、そのリクエストが受信側システムで配信および処理されたかどうかは不明です。通信エラーが発生したときにリクエストを再送信することにより、tRFC プロトコルを使用するクライアントアプリケーションは、そのリクエストの配信と処理を確実に 1 回だけ保証できます。

どのトランザクションモデルを使用するか?

BAPI トランザクションは、SAP データベースで BAPI メソッドまたは RFC 関数によって実行される永

続的なデータ変更に ACID 保証を課すという意味で、アプリケーションレベルのトランザクションです。RFC トランザクションは、BAPI メソッドや RFC 機能への要求に対して配信保証 (AT-MOST-ONCE、EXACTLY-ONCE、EXACTLY-ONCE-IN-ORDER) を課すという意味で、通信トランザクションです。

トランザクション RFC 宛先エンドポイント

次の宛先エンドポイントは、RFC トランザクションをサポートしています。

- **sap-trfc-destination**
- **sap-qrfc-destination**

単一の Camel ルートには、複数のトランザクション RFC 宛先エンドポイントを含めることができ、複数の RFC 宛先にメッセージを送信したり、同じ RFC 宛先に複数回メッセージを送信したりすることもできます。これは、Camel SAP コンポーネントが、ルートを通過する各 **Exchange** オブジェクトの **多くの** トランザクション ID (TID) を追跡する必要がある可能性があることを意味します。ルート処理が失敗し、再試行する必要がある場合、状況は非常に複雑になります。RFC トランザクションセマンティクスでは、ルートに沿った各 RFC 宛先が、最初に使用されたのと **同じ** TID を使用して呼び出される必要があります (各宛先の TID は互いに異なる場合)。つまり、Camel SAP コンポーネントは、ルート上のどのポイントでどの TID が使用されたかを追跡し、この情報を記憶して、TID を正しい順序で再生できるようにする必要があります。

デフォルトでは、Camel は **Exchange** がルートのどこにいるかを知ることができるメカニズムを提供しません。このようなメカニズムを提供するには、**CurrentProcessorDefinitionInterceptStrategy** インターセプターを Camel ランタイムにインストールする必要があります。Camel SAP コンポーネントがルート内の TID を追跡するには、このインターセプターを Camel ランタイムにインストールする必要があります。インターセプターの設定方法の詳細については、「tRFC および qRFC 宛先のインターセプター」 を参照してください。

トランザクション RFC サーバーエンドポイント

次のサーバーエンドポイントは、RFC トランザクションをサポートしています。

- **sap-trfc-server**

トランザクションリクエストを処理する Camel エクスチェンジで処理エラーが発生すると、Camel は標準のエラー処理メカニズムを通じて処理エラーを処理します。エクスチェンジを処理する Camel ルートがエラーを呼び出し元に伝播するように設定されている場合、エクスチェンジを開始した SAP サーバーエンドポイントは失敗を記録し、送信側の SAP システムにエラーが通知されます。送信側の SAP システムは、同じ TID を持つ別のトランザクションリクエストを送信して応答し、リクエストを再度処理できます。

269.8. RFC の XML シリアライゼーション

概要

SAP 要求および応答オブジェクトは、これらのオブジェクトを XML ドキュメントとの間でシリアル化できるようにする XML シリアル化形式をサポートしています。

XML 名前空間

リポジトリー内の各 RFC は、リクエストオブジェクトとレスポンスオブジェクトのシリアル化された形式を設定する要素に対して、特定の XML 名前空間を定義します。この名前空間 URL の形式は次のとおりです。

```
http://sap.fusesource.org/rfc/<Repository Name>/<RFC Name>
```

RFC 名前空間 URL には、共通の **http://sap.fusesource.org/rfc** 接頭辞があり、その後に RFC のメタデータが定義されているリポジトリーの名前が続きます。URL の最後のコンポーネントは、RFC 自体の名前です。

リクエストとレスポンスの XML ドキュメント

SAP リクエストオブジェクトは XML ドキュメントにシリアル化され、そのドキュメントのルート要素は Request という名前で、リクエストの RFC の名前空間によってスコープが設定されます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Request>
```

SAP レスポンスオブジェクトは XML ドキュメントにシリアル化され、そのドキュメントのルート要素は Response という名前で、レスポンス応答の RFC の名前空間によってスコープが設定されます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Response
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Response>
```

構造体フィールド

パラメーターリストまたはネストされた構造体の構造体フィールドは、要素としてシリアル化されます。シリアル化された構造体の要素名は、それが含まれるパラメーターリスト、構造体、またはテーブルの行エントリー内の構造体のフィールド名に対応します。

```
<BOOK_FLIGHT:FLTINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:FLTINFO>
```

次の例のように、RFC 名前空間の構造要素の型名は、構造を定義するレコードメタデータオブジェクトの名前に対応することに注意してください。

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
   ...
   <xs:complexType name="FLTINFO_STRUCTURE”>
   ...
   </xs:complexType>
   ...
</xs:schema>
```

この区別は、「例 3: SAP からのリクエストの処理」 で見られるように、構造体のマーシャリングとアンマーシャリングを行う JAXB Bean を指定するときに重要になります。

テーブルフィールド

パラメーターリストまたはネストされた構造のテーブルフィールドは、要素としてシリアル化されます。シリアル化された構造体の要素名は、それが存在する外側のパラメーターリスト、構造体、またはテーブル行エントリー内のテーブルのフィールド名に対応します。テーブル要素には、テーブルの行エントリーのシリアル化された値を保持する一連の行要素が含まれます。

```
<BOOK_FLIGHT:CONNINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   <row ... > ... </row>
   ...
   <row ... > ... </row>
</BOOK_FLIGHT:CONNINFO>
```

RFC 名前空間の table 要素の型名は、**_TABLE** の接尾辞が付いたテーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応することに注意してください。次の例のように、RFC 名のテーブル行要素の型名は、テーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応します。

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
   ...
   <xs:complextType name="CONNECTION_INFO_STRUCTURE_TABLE”>
      <xs:sequence>
         <xs:element
            name="row”
            minOccures="0”
            maxOccurs="unbounded”
            type="CONNECTION_INFO_STRUCTURE”/>
         ...
         <xs:sequence>
      </xs:sequence>
   </xs:complexType>

   <xs:complextType name="CONNECTION_INFO_STRUCTURE”>
         ...
   </xs:complexType>
   ...
</xs:schema>
```

この区別は、「例 3: SAP からのリクエストの処理」 で見られるように、構造体のマーシャリングとアンマーシャリングを行う JAXB Bean を指定するときに重要になります。

Elementary フィールド

パラメーターリストまたはネストされた構造体の Elementary フィールドは、囲んでいるパラメーターリストまたは構造体の要素の属性としてシリアル化されます。シリアル化されたフィールドの属性名は、次の例のように、それが存在する囲んでいるパラメーターリスト、構造体、またはテーブル行エントリー内のフィールドのフィールド名に対応します。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
     xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
     CUSTNAME="James Legrand"
    PASSFORM="Mr"
    PASSNAME="Travelin Joe"
    PASSBIRTH="1990-03-17T00:00:00.000-0500"
    FLIGHTDATE="2014-03-19T00:00:00.000-0400"
    TRAVELAGENCYNUMBER="00000110"
    DESTINATION_FROM="SFO"
    DESTINATION_TO="FRA"/>
```

日付と時刻の形式

日付と時刻のフィールドは、次の形式を使用して属性値にシリアル化されます。

```
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

日付フィールドは、年、月、日、およびタイムゾーンのコンポーネントセットのみでシリアル化されます。

```
DEPDATE="2014-03-19T00:00:00.000-0400"
```

時間フィールドは、時、分、秒、ミリ秒、およびタイムゾーンコンポーネントセットのみでシリアル化されます。

```
DEPTIME="1970-01-01T16:00:00.000-0500"
```

269.9. IDOC の XML シリアル化

概要

IDoc メッセージ本文は、組み込み型コンバーターを使用して、XML 文字列形式にシリアル化できます。

XML 名前空間

シリアル化された各 IDoc は、次の一般的な形式を持つ XML 名前空間に関連付けられています。

```
http://sap.fusesource.org/idoc/repositoryName/idocType/idocTypeExtension/systemRelease/applicationRelease
```

repositoryName (リモート SAP メタデータリポジトリーの名前) と **idocType** (IDoc ドキュメントタイプ) の両方が必須ですが、名前空間の他のコンポーネントは空白のままにすることができます。たとえば、次のような XML 名前空間を持つことができます。

```
http://sap.fusesource.org/idoc/MY_REPO/FLCUSTOMER_CREATEFROMDATA01///
```

組み込み型コンバーター

Camel SAP コンポーネントには組み込み型コンバーターがあり、**Document** オブジェクトまたは **DocumentList** オブジェクトを String 型に変換したり、**String** 型から変換したりできます。

たとえば、**Document** オブジェクトを XML 文字列にシリアル化するには、次の行を XML DSL のルートに追加するだけです。

```
<convertBodyTo type="java.lang.String"/>
```

このアプローチを使用して、シリアライズされた XML メッセージを **Document** オブジェクトにすることもできます。たとえば、現在のメッセージ本文がシリアル化された XML 文字列である場合、XML DSL のルートに次の行を追加することで、それを **Document** オブジェクトに戻すことができます。

```
<convertBodyTo type="org.fusesource.camel.component.sap.model.idoc.Document"/>
```

XML 形式のサンプル IDoc メッセージ本文

IDoc メッセージを **String** に変換すると、ルート要素が **idoc:Document** (単一のドキュメントの場合) または **idoc:DocumentList** (ドキュメントのリストの場合) のいずれかである XML ドキュメントにシリアライズされます。例269.2「XML の IDoc メッセージボディー」 は、**idoc:Document** 要素にシリアル化された単一の IDoc ドキュメントを示しています。

例269.2 XML の IDoc メッセージボディー

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:FLCUSTOMER_CREATEFROMDATA01---
="http://sap.fusesource.org/idoc/XXX/FLCUSTOMER_CREATEFROMDATA01///"
    xmlns:idoc="http://sap.fusesource.org/idoc"
    creationDate="2015-01-28T12:39:13.980-0500"
    creationTime="2015-01-28T12:39:13.980-0500"
    iDocType="FLCUSTOMER_CREATEFROMDATA01"
    iDocTypeExtension=""
    messageType="FLCUSTOMER_CREATEFROMDATA"
    recipientPartnerNumber="QUICKCLNT"
    recipientPartnerType="LS"
    senderPartnerNumber="QUICKSTART"
    senderPartnerType="LS">
  <rootSegment xsi:type="FLCUSTOMER_CREATEFROMDATA01---:ROOT" document="/">
    <segmentChildren parent="//@rootSegment">
      <E1SCU_CRE parent="//@rootSegment" document="/">
        <segmentChildren parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0">
          <E1BPSCUNEW parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0"
              document="/"
              CUSTNAME="Fred Flintstone" FORM="Mr."
              STREET="123 Rubble Lane"
              POSTCODE="01234"
              CITY="Bedrock"
              COUNTR="US"
              PHONE="800-555-1212"
              EMAIL="fred@bedrock.com"
              CUSTTYPE="P"
              DISCOUNT="005"
              LANGU="E"/>
        </segmentChildren>
```

```
      </E1SCU_CRE>
    </segmentChildren>
  </rootSegment>
</idoc:Document>
```

269.10. 例 1: SAP からのデータの読み取り

概要

この例は、SAP から **FlightCustomer** ビジネスオブジェクトデータを読み取るルートを示しています。ルートは、データを取得するために SAP 同期 RFC 宛先エンドポイントを使用して、**FlightCustomer** BAPI メソッド **BAPI_FLCUST_GETLIST** を呼び出します。

ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
from("direct:getFlightCustomerInfo")
    .to("bean:createFlightCustomerGetListRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST")
    .to("bean:returnFlightCustomerInfo");
```

ルートの XML DSL

また、同じルートの Spring DSL は次のとおりです。

```
<route>
    <from uri="direct:getFlightCustomerInfo"/>
    <to uri="bean:createFlightCustomerGetListRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST"/>
    <to uri="bean:returnFlightCustomerInfo"/>
</route>
```

createFlightCustomerGetListRequest bean

createFlightCustomerGetListRequest Bean は、後続の SAP エンドポイントの RFC 呼び出しで使用される exchange メソッドで SAP 要求オブジェクトを構築するロールを果たします。次のコードスニペットは、リクエストオブジェクトを作成する一連の操作を示しています。

```
public void create(Exchange exchange) throws Exception {

    // Get SAP Endpoint to be called from context.
    SapSynchronousRfcDestinationEndpoint endpoint =
        exchange.getContext().getEndpoint("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST",
                                          SapSynchronousRfcDestinationEndpoint.class);

    // Retrieve bean from message containing Flight Customer name to
    // look up.
    BookFlightRequest bookFlightRequest =
        exchange.getIn().getBody(BookFlightRequest.class);
```

```
    // Create SAP Request object from target endpoint.
    Structure request = endpoint.getRequest();

    // Add Customer Name to request if set
    if (bookFlightRequest.getCustomerName() != null &&
        bookFlightRequest.getCustomerName().length() > 0) {
            request.put("CUSTOMER_NAME",
                    bookFlightRequest.getCustomerName());
        }
    } else {
        throw new Exception("No Customer Name");
    }

    // Put request object into body of exchange message.
    exchange.getIn().setBody(request);
}
```

returnFlightCustomerInfo bean

returnFlightCustomerInfo Bean は、前の SAP エンドポイントから受け取った exchange メソッドで、SAP レスポンスオブジェクトからデータを展開するロールを果たします。次のコードスニペットは、レスポンスオブジェクトからデータを抽出する一連の操作を示しています。

```
public void createFlightCustomerInfo(Exchange exchange) throws Exception {

    // Retrieve SAP response object from body of exchange message.
    Structure flightCustomerGetListResponse =
        exchange.getIn().getBody(Structure.class);

    if (flightCustomerGetListResponse == null) {
        throw new Exception("No Flight Customer Get List Response");
    }

    // Check BAPI return parameter for errors
    @SuppressWarnings("unchecked")
    Table<Structure> bapiReturn =
        flightCustomerGetListResponse.get("RETURN", Table.class);
    Structure bapiReturnEntry = bapiReturn.get(0);
    if (bapiReturnEntry.get("TYPE", String.class) != "S") {
        String message = bapiReturnEntry.get("MESSAGE", String.class);
        throw new Exception("BAPI call failed: " + message);
    }

    // Get customer list table from response object.
    @SuppressWarnings("unchecked")
    Table<? extends Structure> customerList =
        flightCustomerGetListResponse.get("CUSTOMER_LIST", Table.class);

    if (customerList == null || customerList.size() == 0) {
        throw new Exception("No Customer Info.");
    }

    // Get Flight Customer data from first row of table.
    Structure customer = customerList.get(0);
```

```
    // Create bean to hold Flight Customer data.
    FlightCustomerInfo flightCustomerInfo = new FlightCustomerInfo();

    // Get customer id from Flight Customer data and add to bean.
    String customerId = customer.get("CUSTOMERID", String.class);
    if (customerId != null) {
        flightCustomerInfo.setCustomerNumber(customerId);
    }

    ...

    // Put bean into body of exchange message.
    exchange.getIn().setHeader("flightCustomerInfo", flightCustomerInfo);

}
```

269.11. 例 2: SAP へのデータの書き込み

概要

この例は、SAP で **FlightTrip** ビジネスオブジェクトインスタンスを作成するルートを示しています。ルートは、**FlightTrip** BAPI メソッド **BAPI_FLTRIP_CREATE** を呼び出し、宛先エンドポイントを使用してオブジェクトを作成します。

ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
from("direct:createFlightTrip")
    .to("bean:createFlightTripRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true")
    .to("bean:returnFlightTripResponse");
```

ルートの XML DSL

また、同じルートの Spring DSL は次のとおりです。

```
<route>
    <from uri="direct:createFlightTrip"/>
    <to uri="bean:createFlightTripRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true"/>
    <to uri="bean:returnFlightTripResponse"/>
</route>
```

トランザクションサポート

SAP エンドポイントの URL では、**transacted** オプションが **true** に設定されていることに注意してください。「トランザクションサポート」 で説明したように、このオプションが有効になっている場合、エンドポイントは、RFC 呼び出しを呼び出す前に SAP トランザクションセッションが開始されていることを確認します。このエンドポイントの RFC は SAP で新しいデータを作成するため、ルートの変更を SAP で永続的にするには、このオプションが必要です。

リクエストパラメーターの設定

createFlightTripRequest および **returnFlightTripResponse** Bean は、前の例で示したのと同じ一連の操作に従って、リクエストパラメーターを SAP リクエストに入力し、SAP レスポンスからレスポンスパラメーターをそれぞれ展開します。

269.12. 例 3: SAP からのリクエストの処理

概要

この例では、SAP から **BOOK_FLIGHT** RFC へのリクエストを処理するルートを示します。これは、ルートによって実装されます。さらに、JAXB を使用して SAP リクエストオブジェクトとレスポンスオブジェクトをカスタム Bean にアンマーシャリングおよびマーシャリングする、コンポーネントの XML シリアライゼーションサポートを示します。

このルートは、旅行代理店 **FlightCustomer** に代わって **FlightTrip** ビジネスオブジェクトを作成します。ルートは、最初に、SAP サーバーエンドポイントによって受信された SAP リクエストオブジェクトをカスタム JAXB Bean に非整列化します。次に、このカスタム Bean はエクスチェンジで 3 つのサブルートにマルチキャストされ、フライト旅行の作成に必要な旅行代理店、フライト接続、乗客情報が収集されます。最後のサブルートは、前の例で示したように、SAP でフライトトリップオブジェクトを作成します。最後のサブルートは、SAP レスポンスオブジェクトにマーシャリングされ、サーバーエンドポイントによって返されるカスタム JAXB Bean も作成して返します。

ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
DataFormat jaxb = new JaxbDataFormat("org.fusesource.sap.example.jaxb");

from("sap-srfc-server:nplserver:BOOK_FLIGHT")
    .unmarshal(jaxb)
    .multicast()
    .to("direct:getFlightConnectionInfo",
        "direct:getFlightCustomerInfo",
        "direct:getPassengerInfo")
    .end()
    .to("direct:createFlightTrip")
    .marshal(jaxb);
```

ルートの XML DSL

同じルートの XML DSL は次のとおりです。

```
<route>
    <from uri="sap-srfc-server:nplserver:BOOK_FLIGHT"/>
    <unmarshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </unmarshal>
    <multicast>
        <to uri="direct:getFlightConnectionInfo"/>
        <to uri="direct:getFlightCustomerInfo"/>
        <to uri="direct:getPassengerInfo"/>
    </multicast>
```

```
    <to uri="direct:createFlightTrip"/>
    <marshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </marshal>
</route>
```

BookFlightRequest bean

次のリストは、シリアライズされた形式の SAP **BOOK_FLIGHT** リクエストオブジェクトからアンマーシャリングする JAXB Bean を示しています。

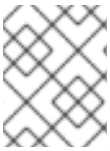
```
@XmlRootElement(name="Request",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightRequest {

    @XmlAttribute(name="CUSTNAME")
    private String customerName;

    @XmlAttribute(name="FLIGHTDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date flightDate;

    @XmlAttribute(name="TRAVELAGENCYNUMBER")
    private String travelAgencyNumber;

    @XmlAttribute(name="DESTINATION_FROM")
    private String startAirportCode;

    @XmlAttribute(name="DESTINATION_TO")
    private String endAirportCode;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlAttribute(name="CLASS")
    private String flightClass;

    ...
}
```

BookFlightResponse Bean

次のリストは、シリアライズされた形式の SAP **BOOK_FLIGHT** レスポンスオブジェクトにマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="Response",
```

```
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightResponse {

    @XmlAttribute(name="TRIPNUMBER")
    private String tripNumber;

    @XmlAttribute(name="TICKET_PRICE")
    private BigDecimal ticketPrice;

    @XmlAttribute(name="TICKET_TAX")
    private BigDecimal ticketTax;

    @XmlAttribute(name="CURRENCY")
    private String currency;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlElement(name="FLTINFO")
    private FlightInfo flightInfo;

    @XmlElement(name="CONNINFO")
    private ConnectionInfoTable connectionInfo;

    ...
}
```



注記

レスポンスオブジェクトの複雑なパラメーターフィールドは、応答の子要素としてシリアル化されます。

FlightInfo ビーン

次のリストは、複雑な構造体パラメーター **FLTINFO** のシリアル化された形式にマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="FLTINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class FlightInfo {

    @XmlAttribute(name="FLIGHTTIME")
    private String flightTime;

    @XmlAttribute(name="CITYFROM")
    private String cityFrom;
```

```
    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureTime;

    @XmlAttribute(name="CITYTO")
    private String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalTime;

    ...
}
```

ConnectionInfoTable Bean

次のリストは、複雑なテーブルパラメーター **CONNINFO** のシリアル化された形式にマーシャリングする JAXB Bean を示しています。JAXB Bean のルート要素タイプの名前は、接尾辞 **_TABLE** が付いた行構造体タイプの名前に対応し、Bean には行要素のリストが含まれていることに注意してください。

```
@XmlRootElement(name="CONNINFO_TABLE",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfoTable {

    @XmlElement(name="row")
    List<ConnectionInfo> rows;

    ...
}
```

ConnectionInfo bean

次のリストは、上記のテーブルの行要素のシリアル化された形式にマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="CONNINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfo {

    @XmlAttribute(name="CONNID")
    String connectionId;

    @XmlAttribute(name="AIRLINE")
```

```
    String airline;

    @XmlAttribute(name="PLANETYPE")
    String planeType;

    @XmlAttribute(name="CITYFROM")
    String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureTime;

    @XmlAttribute(name="CITYTO")
    String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalTime;

    ...
}
```

第270章 SAP NETWEAVER コンポーネント

Camel バージョン 2.12 以降で利用可能

sap-netweaver は、HTTP トランスポートを使用して SAP NetWeaver Gateway と統合します。

このキャメルコンポーネントはプロデューサーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sap-netweaver</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

270.1. URI 形式

SAP Netweaver ゲートウェイコンポーネントの URI スキームは次のとおりです。

```
sap-netweaver:https://host:8080/path?username=foo&password=secret
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

270.2. 前提条件

このコンポーネントを利用するには、SAP NetWeaver システムのアカウントが必要です。SAP は、アカウントを要求できる デモセットアップ を提供します。

このコンポーネントは、SAP NetWeaver へのログインに Basic 認証スキームを使用します。

270.3. SAPNETWEAVER オプション

SAP NetWeaver コンポーネントにはオプションがありません。

SAP NetWeaver エンドポイントは、URI 構文を使用して設定されます。

```
sap-netweaver:url
```

パスおよびクエリーパラメーターを使用します。

270.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
url	**必須** SAP net-weaver ゲートウェイサーバーへの URL。		String

270.3.2. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
flatternMap (producer)	JSON マップに含まれるエントリーが 1 つだけの場合は、その 1 つのエントリー値をメッセージボディーとして格納することで平坦化します。	true	boolean
json (producer)	データを JSON 形式で返すかどうか。このオプションが false の場合、XML は Atom 形式で返されます。	true	boolean
jsonAsMap (producer)	メッセージ本文で JSON を String から Map に変換します。	true	boolean
password (producer)	**必須** アカウントのパスワード。		String
username (producer)	**必須** アカウントのユーザー名。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

270.4. メッセージヘッダー

次のヘッダーは、プロデューサーで使用できます。

名前	タイプ	説明
CamelNetWeaverCommand	**String**	**必須**: MS ADO.Net Data Service 形式で実行するコマンド。

270.5. 例

この例では、インターネット 経由 でオンラインで入手できる SAP のフライトデモの例を使用しています。

以下のルートでは、次の URL を使用して SAP NetWeaver デモサーバーをリクエストします。

> https://*sapes1.sapdevcenter.com/sap/opu/odata/IWBEP/RMTSAMPLEFLIGHT_2/*

そして、次のコマンドを実行したい

```
FlightCollection(AirLineID='AA',FlightConnectionID='0017',FlightDate=datetime'2012-08-
29T00%3A00%3A00')
```

特定のフライトのフライトの詳細を取得します。コマンド構文は、MS ADO.Net Data Service 形式です。

次の Camel ルートがあります

```
from("direct:start")
    .setHeader(NetWeaverConstants.COMMAND, constant(command))
    .toF("sap-netweaver:%s?username=%s&password=%s", url, username, password)
    .to("log:response")
    .to("velocity:flight-info.vm")
```

URL、ユーザー名、パスワード、およびコマンドは次のように定義されます。

```
    private String username = "P1909969254";
    private String password = "TODO";
    private String url =
"https://sapes1.sapdevcenter.com/sap/opu/odata/IWBEP/RMTSAMPLEFLIGHT_2/";
    private String command =
"FlightCollection(AirLineID='AA',FlightConnectionID='0017',FlightDate=datetime'2012-08-
29T00%3A00%3A00')";
```

パスワードが無効です。デモを実行するには、最初に SAP でアカウントを作成する必要があります。

速度テンプレートは、基本的な HTML ページへのレスポンスをフォーマットするために使用されます

```
<html>
  <body>
  Flight information:

  <p/>
  <br/>Airline ID: $body["AirLineID"]
  <br/>Aircraft Type: $body["AirCraftType"]
  <br/>Departure city: $body["FlightDetails"]["DepartureCity"]
  <br/>Departure airport: $body["FlightDetails"]["DepartureAirPort"]
  <br/>Destination city: $body["FlightDetails"]["DestinationCity"]
  <br/>Destination airport: $body["FlightDetails"]["DestinationAirPort"]

  </body>
</html>
```

アプリケーションを実行すると、サンプル出力が得られます。

```
Flight information:
Airline ID: AA
Aircraft Type: 747-400
Departure city: new york
Departure airport: JFK
Destination city: SAN FRANCISCO
Destination airport: SFO
```

270.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- HTTP

第271章 スケジューラーコンポーネント

Camel バージョン 2.15 以降で利用可能

scheduler: コンポーネントは、スケジューラーの起動時にメッセージ交換を生成するために使用されます。このコンポーネントは Timer コンポーネントに似ていますが、スケジュール関連の機能をより多く提供します。また、このコンポーネントは JDK **ScheduledExecutorService** を使用します。タイマーは JDK **Timer** を使用します。

このエンドポイントからのイベントのみを使用できます。

271.1. URI 形式

```
scheduler:name[?options]
```

name は、エンドポイント間で作成および共有されるスケジューラーの名前です。したがって、すべてのタイマーエンドポイントに同じ名前を使用すると、1 つのスケジューラースレッドプールとスレッドのみが使用されますが、より多くの同時スレッドを許可するようにスレッドプールを設定できます。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

注記: 生成されたエクスチェンジの IN ボディーは **null** です。したがって、**exchange.getIn().getBody()** は **null** を返します。

271.2. オプション

Scheduler コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
concurrentTasks (スケジューラー)	スケジューリングスレッドプールによって使用されるスレッドの数。デフォルトでは、単一のスレッドを使用します	1	int
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

スケジューラーエンドポイントは、URI 構文を使用して設定されます。

```
scheduler:name
```

パスおよびクエリーパラメーターを使用します。

271.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** スケジューラーの名前		String

271.2.2. クエリーパラメーター (20 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。つまり、ポーリングにより情報を収集しているときにエラーが発生し (例えばファイルネットワークへのアクセスに失敗した)、Camel はそれにアクセスしてファイルをスキャンできません。デフォルトの実装では、WARN レベルで原因となった例外をログに記録し、無視します。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
concurrentTasks (スケジューラー)	スケジューリングスレッドプールによって使用されるスレッドの数。デフォルトでは、単一のスレッドを使用します	1	int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。デフォルト値は 500 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。デフォルト値は 1000 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。このオプションを使用すると、複数のコンシューマー間でスレッドプールを共有できます。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	カスタムの org.apache.camel.spi.ScheduledPollConsumerScheduler をプラグインして、ポーリングコンシューマーの実行時に起動するスケジューラーとして使用できるようにします。デフォルトの実装では ScheduledExecutorService を使用し、CRON 式をサポートする Quartz2 および Spring ベースのものがあります。注記: カスタムスケジューラーを使用している場合、initialDelay、useFixedDelay、timeUnit、および scheduledExecutorService のオプションが使用されていない可能性があります。Quartz2 スケジューラーの使用を参照するにはテキスト Quartz2 を使用し、Spring ベースを使用するにはテキスト spring を使用し、レジストリー内の ID でカスタムスケジューラーを参照するにはテキスト myScheduler を使用します。例については、Quartz2 ページを参照してください。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

271.3. 補足情報

このコンポーネントはスケジューラーの ポーリングコンシューマー であり、上記のオプションに関する詳細情報と、ポーリングコンシューマー ページで例を見つけることができます。

271.4. エクスチェンジプロパティー

タイマーが起動すると、次の情報がプロパティーとして **Exchange** に追加されます。

名前	タイプ	説明
Exchange.TIMER_NAME	**String**	**name** オプションの値。
Exchange.TIMER_FIRED_TIME	**日付**	コンシューマーが起動した時刻。

271.5. 例

60 秒ごとにイベントを生成するルートを設定するには:

```
from("scheduler://foo?period=60s").to("bean:myBean?method=someMethodName");
```

上記のルートはイベントを生成し、JNDI や Spring などのレジストリーで **myBean** と呼ばれる Bean で **someMethodName** メソッドを呼び出します。

そして、Spring DSL の経路:

```
<route>
  <from uri="scheduler://foo?period=60s"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

271.6. スケジューラーが完了したらすぐにトリガーするように強制する

前のタスクが完了するとすぐにスケジューラーがトリガーされるようにするには、オプション **greedy=true** を設定できます。ただし、スケジューラーは常に起動し続けることに注意してください。したがって、注意して使用してください。

271.7. スケジューラーを強制的にアイドル状態にする

スケジューラーをトリガーして、Greedy な設定にするようなユースケースがあるとします。ただし、ポーリングするタスクがないことをスケジューラーに通知したい場合もあります。これにより、スケジューラーはバックオフオプションを使用してアイドルモードに変更できます。これを行うには、キー **Exchange.SCHEDULER_POLLED_MESSAGES** を使用してエクスチェンジのプロパティーを false のブール値に設定する必要があります。これにより、コンシューマーは、ポーリングされたメッセージがなかったことを示します。

それ以外の場合、コンシューマーは、デフォルトで、コンシューマーがエクスチェンジの処理を完了するたびに、スケジューラーにポーリングされた 1 つのメッセージを返します。

271.8. 関連項目

- Timer

- Quartz

第272章 SCHEMATRON コンポーネント

Camel バージョン 2.15 以降で利用可能

Schematron は、XML インスタンスドキュメントを検証するための XML ベースの言語です。XML ドキュメント内のデータに関するアサーションを作成するために使用され、運用ルールやビジネスルールを表現するためにも使用されます。Schematron は ISO 規格 です。スキーマトロンコンポーネントは、ISO スキーマトロンの主要な 実装 を使用します。XSLT ベースの実装です。schematron ルールは 4 つの XSLT パイプライン を介して実行され、XML ドキュメントに対してアサーションを実行するための基礎として使用される最終的な XSLT が生成されます。コンポーネントは、Schematron ルールがエンドポイントの開始時に (1 回だけ) ロードされるように記述されています。これは、ルールを表す Java テンプレートオブジェクトをインスタンス化するオーバーヘッドを最小限に抑えるためです。

272.1. URI 形式

```
schematron://path?[options]
```

272.2. URI オプション

Schematron コンポーネントにはオプションがありません。

Schematron エンドポイントは、URI 構文を使用して設定されます。

```
schematron:path
```

パスおよびクエリーパラメーターを使用します。

272.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
path	**必須** schematron ルールファイルへのパス。クラスパスまたはファイルシステム内の場所のいずれかにすることができます。		String

272.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
abort (producer)	ルートを中止し、schematron 検証例外を出力するフラグ。	false	boolean
rules (producer)	パスからロードする代わりに、指定された schematron ルールを使用するには		テンプレート

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
uriResolver (advanced)	schematron インクルードの解決に使用する URIResolver をルールファイルに設定します。		URIResolver

272.3. ヘッダー

名前	説明	タイプ	In/Out
CamelSchematronValidationStatus	schematron の検証ステータス: SUCCESS/FAILED	String	IN
CamelSchematronValidationReport	XML 形式の schematron レポート本文。以下の例を参照してください	String	IN

272.4. URI とパスの構文

次の例は、Java DSL で schematron プロセッサーを呼び出す方法を示しています。schematron ルールファイルは、クラスパスから取得されます。

```
from("direct:start").to("schematron://sch/schematron.sch").to("mock:result")
```

次の例は、XML DSL で schematron プロセッサーを呼び出す方法を示しています。schematron ルールファイルは、ファイルシステムから取得されます。

```
<route>
  <from uri="direct:start" />
  <to uri="schematron:///usr/local/sch/schematron.sch" />
```

```
  <log message="Schematron validation status: ${in.header.CamelSchematronValidationStatus}" />
  <choice>
    <when>
      <simple>${in.header.CamelSchematronValidationStatus} == 'SUCCESS'</simple>
      <to uri="mock:success" />
    </when>
    <otherwise>
      <log message="Failed schematron validation" />
      <setBody>
        <header>CamelSchematronValidationReport</header>
      </setBody>
      <to uri="mock:failure" />
    </otherwise>
  </choice>
</route>
```

ヒント

schematron ルールをどこに保存しますか?Schematron ルールはビジネス要件に応じて変更される可能性があるため、これらのルールをファイルシステムのどこかに保存することをお勧めします。schematron コンポーネントエンドポイントが開始されると、ルールは Java テンプレートオブジェクトとして XSLT にコンパイルされます。これは、Java Templates オブジェクトのインスタンス化のオーバーヘッドを最小限に抑えるために 1 回だけ実行されます。これは、大規模なルールセットの場合、プロセスが XSLT 変換 の 4 つのパイプラインを通過することを考えると、コストのかかる操作になる可能性があります。そのため、たまたまルールをファイルシステムに保存した場合、更新が発生した場合は、ルートまたはコンポーネントを再起動するだけで済みます。これらのルールをクラスパスに保存しても害はありませんが、変更を反映するにはコンポーネントをビルドしてデプロイする必要があります。

272.5. SCHEMATRON ルールとレポートのサンプル

Schematron ルールの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Check Sections 12/07</title>
  <pattern id="section-check">
    <rule context="section">
      <assert test="title">This section has no title</assert>
      <assert test="para">This section has no paragraphs</assert>
    </rule>
  </pattern>
</schema>
```

Schematron レポートの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl"
 xmlns:iso="http://purl.oclc.org/dsdl/schematron"
 xmlns:saxon="http://saxon.sf.net/"
 xmlns:schold="http://www.ascc.net/xml/schematron"
 xmlns:xhtml="http://www.w3.org/1999/xhtml"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="" title="">
```

```
    <svrl:active-pattern document="" />
    <svrl:fired-rule context="chapter" />
    <svrl:failed-assert test="title" location="/doc[1]/chapter[1]">
      <svrl:text>A chapter should have a title</svrl:text>
    </svrl:failed-assert>
    <svrl:fired-rule context="chapter" />
    <svrl:failed-assert test="title" location="/doc[1]/chapter[2]">
      <svrl:text>A chapter should have a title</svrl:text>
    </svrl:failed-assert>
    <svrl:fired-rule context="chapter" />
</svrl:schematron-output>
```

ヒント

便利なリンクとリソース * Mulleberry technologies による Schematron の紹介。Schematron の使用を開始するための優れた PDF ドキュメントです。スキーマトロン公式サイト。これには、他のリソースへのリンクが含まれています

第273章 SCP コンポーネント

Camel バージョン 2.10 以降で利用可能

camel-jsch コンポーネントは、Jsch プロジェクトのクライアント API を使用して SCP プロトコル をサポートします。Jsch は、**sftp:** プロトコルの FTP コンポーネントによって camel ですでに使用されています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jsch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

273.1. URI 形式

```
scp://host[:port]/destination[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

ファイル名は、URI の <path> 部分で、またはメッセージの CamelFileName ヘッダーとして指定できます (コードで使用されている場合は **Exchange.FILE_NAME**)。

273.2. オプション

SCP コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
verboseLogging (producer)	JSCH は、すぐに使用できる詳細ログです。したがって、デフォルトでロギングを DEBUG ロギングに下げます。ただし、このオプションを true に設定すると、詳細ログが再びオンになります。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SCP エンドポイントは、URI 構文を使用して設定されます。

```
scp:host:port/directoryName
```

パスおよびクエリーパラメーターを使用します。

273.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** FTP サーバーのホスト名		String
port	FTP サーバーのポート		int
directoryName	開始ディレクトリー		文字列

273.2.2. クエリーパラメーター (20 パラメーター)

名前	説明	デフォルト	タイプ
disconnect (Common)	使用直後にリモート FTP サーバーから切断するかどうか。切断は、FTP サーバーへの現在の接続のみを切断します。停止したいコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
chmod (producer)	保存されたファイルに chmod を設定できます。たとえば、chmod=644 です。	664	String
fileName (producer)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込むファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。(注: ヘッダー自体を式にすることもできます)。式オプションは String タイプと Expression タイプの両方をサポートします。式が String タイプである場合、これは常にファイル言語を使用して評価されます。式が Expression タイプである場合、指定された Expression タイプが使用されます。これにより、たとえば OGNL 式を使用できます。コンシューマーの場合、これを使用してファイル名をフィルターリングできるため、たとえば、ファイル言語構文 mydata-$date:now:yyyyMMdd.txt を使用して今日のファイルを消費できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は一度だけ使用されるヘッダーであり、CamelFileName を一時的に保存して後で復元する必要がなくなるため、簡単になります。		String

名前	説明	デフォルト	タイプ
flatten (producer)	flatten は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に使用できますが、たとえばファイルを別のディレクトリーに書き込む場合、ファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が先頭パスから削除されます。	false	boolean
strictHostKeyChecking (producer)	厳密なホストキーチェックを使用するかどうかを設定します。可能な値は次のとおりです: いいえ、はい	いいえ	String
allowNullBody (producer)	ファイルの書き込み中に null の本文を許可するかどうかを指定するために使用されます。true に設定すると空のファイルが作成され、false に設定して null の本文をファイルコンポーネントに送信しようとすると、Cannot write null body to file.という GenericFileWriteException が出力されます。fileExist オプションを Override に設定するとファイルは切り捨てられ、append に設定するとファイルは変更されません。	false	boolean
disconnectOnBatchComplete (producer)	バッチアップロードが完了した直後にリモート FTP サーバーから切断するかどうか。disconnectOnBatchComplete は、FTP サーバーへの現在の接続のみを切断します。	false	boolean
connectTimeout (advanced)	接続が確立されるのを待つための接続タイムアウトを設定します。FTPClient と JSCH の両方で使用されます	10000	int
soTimeout (advanced)	SO タイムアウトを設定します。FTPClient によってのみ使用されます。	300000	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
timeout (advanced)	応答を待つためのデータタイムアウトを設定します。FTPClient だけが使用します	30000	int
knownHostsFile (security)	jsch エンドポイントがホストキーの検証を実行できるように、known_hosts ファイルを設定します。ファイルシステムの代わりにクラスパスからファイルをロードするには、classpath: を前に付けることができます。		String

名前	説明	デフォルト	タイプ
password (security)	ログインに使用するパスワード		String
preferredAuthentications (security)	優先順に使用される認証のコンマ区切りリストを設定します。可能な認証方法は、JCraft JSCH によって定義されています。いくつかの例を以下に示します: gssapi-with-mic,publickey,keyboard-interactive,password 指定しない場合、JSCH および/またはシステムのデフォルトが使用されます。		String
privateKeyBytes (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵のバイトを設定します。これは、privateKeyFile が設定されていない場合にのみ使用する必要があります。それ以外の場合、ファイルが優先されます。		byte[]
privateKeyFile (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルを設定します。ファイルシステムの代わりにクラスパスからファイルをロードするには、classpath: を前に付けることができます。		String
privateKeyFilePassphrase (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルのパスフレーズを設定します。		String
username (security)	ログインに使用するユーザー名		String
useUserKnownHostsFile (security)	knownHostFile が明示的に設定されていない場合は、System.getProperty (user.home)/.ssh/known_hosts のホストファイルを使用します。	true	boolean
ciphers (security)	優先順に使用される暗号のコンマ区切りリストを設定します。可能な暗号名は、JCraft JSCH によって定義されています。例としては、aes128-ctr、aes128-cbc、3des-ctr、3des-cbc、blowfish-cbc、aes192-cbc、aes256-cbc などがあります。指定しない場合、JSCH のデフォルトリストが使用されます。		文字列

273.3. 制限

現在 camel-jsch は Producer のみをサポートしています (つまり、ファイルを別のホストにコピーします)。

273.4. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント
- エンドポイント
- スタートガイド

第274章 CAMEL SCR (非推奨)

Camel 2.15 以降で利用可能

SCR は Service Component Runtime の略で、OSGi Declarative Services 仕様の実装です。SCR を使用すると、定型コードを使用せずに、プレーンな古い Java オブジェクトで OSGi サービスを公開して使用できます。

OSGi フレームワークは、通常は **org.apache.felix:maven-scr-plugin** などのプラグインによって Java アノテーションから生成されるバンドル内の SCR 記述子ファイルを調べることで、オブジェクトを認識します。

SCR バンドルで Camel を実行することは、Spring DM および Blueprint ベースのソリューションの優れた代替手段であり、OSGi フレームワークとの間のコード行が大幅に少なくなります。SCR を使用すると、バンドルは完全に Java の世界にとどまることができます。XML またはプロパティーファイルを編集する必要はありません。これにより、すべてを完全に制御でき、選択した IDE がプロジェクトで何が起こっているかを正確に認識できることを意味します。

274.1. CAMEL SCR サポート

Camel-scr バンドルは 2.15.0 より前の Apache Camel バージョンには含まれていませんが、アーティファクト自体は 2.12.0 以降のすべての Camel バージョンで使用できます。

org.apache.camel/camel-scr バンドルは、Camel コンテキストを管理する基本クラス **AbstractCamelRunner** と、単体テストで SCR プロパティーを使用するためのヘルパークラス **ScrHelper** を提供します。Apache Karaf の Camel-scr 機能は、SCR バンドルで Camel を実行するために必要なすべての機能とバンドルを定義します。

AbstractCamelRunner クラスは、CamelContext のライフサイクルをサービスコンポーネントのライフサイクルに結び付け、Camel の PropertiesComponent を使用して設定を処理します。Java クラスからサービスコンポーネントを作成するために必要なことは、それを **AbstractCamelRunner** から拡張し、クラスレベルで次の **org.apache.felix.scr.annotations** を追加することだけです。

必要なアノテーションを追加する

```
@Component
@References({
    @Reference(name = "camelComponent",referenceInterface = ComponentResolver.class,
        cardinality = ReferenceCardinality.MANDATORY_MULTIPLE, policy =
ReferencePolicy.DYNAMIC,
        policyOption = ReferencePolicyOption.GREEDY, bind = "gotCamelComponent", unbind =
"lostCamelComponent")
})
```

次に、実行する Camel ルートを返す **getRouteBuilders()** メソッドを実装します。

Implement getRouteBuilders()

```
    @Override
    protected List<RoutesBuilder> getRouteBuilders() {
        List<RoutesBuilder> routesBuilders = new ArrayList<>();
        routesBuilders.add(new YourRouteBuilderHere(registry));
        routesBuilders.add(new AnotherRouteBuilderHere(registry));
        return routesBuilders;
    }
```

最後に、デフォルト設定を次のように指定します。

アノテーションのデフォルト設定

```
@Properties({
  @Property(name = "camelContextId", value = "my-test"),
  @Property(name = "active", value = "true"),
  @Property(name = "...", value = "..."),
  ...
})
```

以上です。そして、**camel-archetype-scr** を使用してプロジェクトを生成した場合、これはすべてすでに処理されています。

以下は、**camel-archetype-scr:** によって生成された完全なサービスコンポーネントクラスの例です。

CamelScrExample.java

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-SNAPSHOT
package example;

import java.util.ArrayList;
import java.util.List;

import org.apache.camel.scr.AbstractCamelRunner;
import example.internal.CamelScrExampleRoute;
import org.apache.camel.RoutesBuilder;
import org.apache.camel.spi.ComponentResolver;
import org.apache.felix.scr.annotations.*;

@Component(label = CamelScrExample.COMPONENT_LABEL, description =
CamelScrExample.COMPONENT_DESCRIPTION, immediate = true, metatype = true)
@Properties({
   @Property(name = "camelContextId", value = "camel-scr-example"),
   @Property(name = "camelRouteId", value = "foo/timer-log"),
   @Property(name = "active", value = "true"),
   @Property(name = "from", value = "timer:foo?period=5000"),
   @Property(name = "to", value = "log:foo?showHeaders=true"),
   @Property(name = "messageOk", value = "Success: {{from}} -> {{to}}"),
   @Property(name = "messageError", value = "Failure: {{from}} -> {{to}}"),
   @Property(name = "maximumRedeliveries", value = "0"),
   @Property(name = "redeliveryDelay", value = "5000"),
   @Property(name = "backOffMultiplier", value = "2"),
   @Property(name = "maximumRedeliveryDelay", value = "60000")
})
@References({
   @Reference(name = "camelComponent",referenceInterface = ComponentResolver.class,
      cardinality = ReferenceCardinality.MANDATORY_MULTIPLE, policy =
ReferencePolicy.DYNAMIC,
      policyOption = ReferencePolicyOption.GREEDY, bind = "gotCamelComponent", unbind =
"lostCamelComponent")
})
public class CamelScrExample extends AbstractCamelRunner {
```

```
    public static final String COMPONENT_LABEL = "example.CamelScrExample";
    public static final String COMPONENT_DESCRIPTION = "This is the description for camel-scr-
example.";

    @Override
    protected List<RoutesBuilder> getRouteBuilders() {
        List<RoutesBuilder> routesBuilders = new ArrayList<>();
        routesBuilders.add(new CamelScrExampleRoute(registry));
        return routesBuilders;
    }
}
```

CamelContextId と **active** プロパティーは、CamelContext の名前 (デフォルトは camel-runner-default) と、それを開始するかどうか (デフォルトは false) をそれぞれ制御します。これらに加えて、好きなだけプロパティーを追加して使用できます。Camel の PropertiesComponent は、再帰的なプロパティーとフォールバックの接頭辞を問題なく処理します。

AbstractCamelRunner は、Camel の PropertiesComponent の助けを借りて、これらのプロパティーを RouteBuilders で利用できるようにし、名前が一致する場合、サービスコンポーネントと RouteBuilder のフィールドにもこれらの値を挿入します。フィールドは任意の可視性レベルで宣言でき、多くの型がサポートされています (String、int、boolean、URL など)。

以下は **camel-archetype-scr** によって生成された RouteBuilder クラスの例です:

CamelScrExampleRoute.java

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-SNAPSHOT
package example.internal;

import org.apache.camel.LoggingLevel;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.impl.SimpleRegistry;
import org.apache.commons.lang.Validate;

public class CamelScrExampleRoute extends RouteBuilder {

    SimpleRegistry registry;

    // Configured fields
    private String camelRouteId;
    private Integer maximumRedeliveries;
    private Long redeliveryDelay;
    private Double backOffMultiplier;
    private Long maximumRedeliveryDelay;

    public CamelScrExampleRoute(final SimpleRegistry registry) {
        this.registry = registry;
    }

    @Override
    public void configure() throws Exception {
        checkProperties();
```

```
        // Add a bean to Camel context registry
        registry.put("test", "bean");

        errorHandler(defaultErrorHandler()
            .retryAttemptedLogLevel(LoggingLevel.WARN)
            .maximumRedeliveries(maximumRedeliveries)
            .redeliveryDelay(redeliveryDelay)
            .backOffMultiplier(backOffMultiplier)
            .maximumRedeliveryDelay(maximumRedeliveryDelay));

        from("{{from}}")
            .startupOrder(2)
            .routeId(camelRouteId)
            .onCompletion()
                .to("direct:processCompletion")
            .end()
            .removeHeaders("CamelHttp*")
            .to("{{to}}");


        from("direct:processCompletion")
            .startupOrder(1)
            .routeId(camelRouteId + ".completion")
            .choice()
                .when(simple("${exception} == null"))
                    .log("{{messageOk}}")
                .otherwise()
                    .log(LoggingLevel.ERROR, "{{messageError}}")
            .end();
        }
    }

    public void checkProperties() {
        Validate.notNull(camelRouteId, "camelRouteId property is not set");
        Validate.notNull(maximumRedeliveries, "maximumRedeliveries property is not set");
        Validate.notNull(redeliveryDelay, "redeliveryDelay property is not set");
        Validate.notNull(backOffMultiplier, "backOffMultiplier property is not set");
        Validate.notNull(maximumRedeliveryDelay, "maximumRedeliveryDelay property is not set");
    }
}
```

CamelScrExampleRoute をさらに詳しく見てみましょう。

```
    // Configured fields
    private String camelRouteId;
    private Integer maximumRedeliveries;
    private Long redeliveryDelay;
    private Double backOffMultiplier;
    private Long maximumRedeliveryDelay;
```

これらのフィールドの値は、名前を一致させることにより、プロパティからの値で設定されます。

```
        // Add a bean to Camel context registry
        registry.put("test", "bean");
```

ルート用にいくつかの Bean を CamelContext のレジストリーに追加する必要がある場合は、次のように実行できます。

```
    public void checkProperties() {
        Validate.notNull(camelRouteId, "camelRouteId property is not set");
        Validate.notNull(maximumRedeliveries, "maximumRedeliveries property is not set");
        Validate.notNull(redeliveryDelay, "redeliveryDelay property is not set");
        Validate.notNull(backOffMultiplier, "backOffMultiplier property is not set");
        Validate.notNull(maximumRedeliveryDelay, "maximumRedeliveryDelay property is not set");
    }
```

ルートの開始を許可する前に、必要なパラメーターが設定されていて、それらに意味のある値があることを確認することをお勧めします。

```
        from("{{from}}")
            .startupOrder(2)
            .routeId(camelRouteId)
            .onCompletion()
                .to("direct:processCompletion")
            .end()
            .removeHeaders("CamelHttp*")
            .to("{{to}}");


        from("direct:processCompletion")
            .startupOrder(1)
            .routeId(camelRouteId + ".completion")
            .choice()
                .when(simple("${exception} == null"))
                    .log("{{messageOk}}")
                .otherwise()
                    .log(LoggingLevel.ERROR, "{{messageError}}")
            .end();
```

ルートのほとんどすべてがプロパティーで設定されていることに注意してください。これにより、基本的に RouteBuilder がテンプレートになります。SCR を使用すると、代替設定を提供するだけで、ルートのインスタンスをさらに作成できます。詳細については、**Camel SCR バンドルをテンプレートとして使用する** セクションを参照してください。

274.2. SCR での ABSTRACTCAMELRUNNER のライフサイクル

1. コンポーネントの設定ポリシーと必須の参照が満たされると、SCR は **activate()** を呼び出します。これにより、**activate()** → **prepare()** → **createCamelContext()** → **setupPropertiesComponent()** → **configure()** → **setupCamelContext()** の呼び出しチェーン

を通じて CamelContext が作成および設定されます。最後に、コンテキストは、**runWithDelay()** を使用して **AbstractCamelRunner.START_DELAY** で定義された遅延の後に開始するようにスケジュールされます。

2. Camel コンポーネント (正確には **ComponentResolver** サービス) が OSGi に登録されると、SCR は **gotCamelComponent**`()` を呼び出します。これは、同じ **AbstractCamelRunner.START_DELAY** によって CamelContext の開始を再スケジュール/遅延させます。これにより、CamelContext は、すべての Camel コンポーネントがロードされるか、それらの間に十分なギャップができるまで待機します。同じロジックにより、Camel コンポーネントをさらに追加するたびに、起動に失敗した CamelContext に再試行するように指示されます。

3. Camel コンポーネントが未登録の場合、SCR は **lostCamelComponent**`()` を呼び出します。この呼び出しは何もしません。

4. **activate() の呼び出しの原因となった要件の 1 つが失われると、SCR は activate()** を呼び出します。これにより、CamelContext がシャットダウンされます。

(非 OSGi) 単体テストでは、よりきめ細かい制御のために、**prepare() → run() → stop()** instead of **activate() → deactivate()** を使用する必要があります。また、これにより、テストで発生する可能性のある SCR 固有の操作を回避できます。

274.3. CAMEL-ARCHETYPE-SCR の使用

Camel SCR バンドルプロジェクトを作成する最も簡単な方法は、**camel-archetype-scr** と Maven を使用することです。

次の手順でプロジェクトを生成できます。

プロジェクトの生成

```
$ mvn archetype:generate -Dfilter=org.apache.camel.archetypes:camel-archetype-scr

Choose archetype:
1: local -> org.apache.camel.archetypes:camel-archetype-scr (Creates a new Camel SCR bundle
project for Karaf)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 1
Define value for property 'groupId': : example
[INFO] Using property: groupId = example
Define value for property 'artifactId': : camel-scr-example
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': example: :
[INFO] Using property: archetypeArtifactId = camel-archetype-scr
[INFO] Using property: archetypeGroupId = org.apache.camel.archetypes
[INFO] Using property: archetypeVersion = 2.15-SNAPSHOT
Define value for property 'className': : CamelScrExample
Confirm properties configuration:
groupId: example
artifactId: camel-scr-example
version: 1.0-SNAPSHOT
package: example
archetypeArtifactId: camel-archetype-scr
archetypeGroupId: org.apache.camel.archetypes
archetypeVersion: 2.15-SNAPSHOT
className: CamelScrExample
Y: :
```

-

完了

実行します:

```
mvn install
```

これで、バンドルをデプロイする準備が整いました。

274.4. 単体テスト CAMEL ルート

Service Component は POJO であり、(OSGi 以外の) 単体テストに関する特別な要件はありません。ただし、Camel SCR に固有の手法や、テストを簡単にする手法がいくつかあります。

以下は **camel-archetype-scr** によって生成された単体テストの例です:

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-SNAPSHOT
package example;

import java.util.List;

import org.apache.camel.scr.internal.ScrHelper;
import org.apache.camel.builder.AdviceWithRouteBuilder;
import org.apache.camel.component.mock.MockComponent;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.model.ModelCamelContext;
import org.apache.camel.model.RouteDefinition;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestName;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

@RunWith(JUnit4.class)
public class CamelScrExampleTest {

    Logger log = LoggerFactory.getLogger(getClass());

    @Rule
    public TestName testName = new TestName();

    CamelScrExample integration;
    ModelCamelContext context;

    @Before
    public void setUp() throws Exception {
        log.info("********************************************************************************");
        log.info("Test: " + testName.getMethodName());
        log.info("********************************************************************************");

        // Set property prefix for unit testing
```

```
        System.setProperty(CamelScrExample.PROPERTY_PREFIX, "unit");

        // Prepare the integration
        integration = new CamelScrExample();
        integration.prepare(null, ScrHelper.getScrProperties(integration.getClass().getName()));
        context = integration.getContext();

        // Disable JMX for test
        context.disableJMX();

        // Fake a component for test
        context.addComponent("amq", new MockComponent());
    }

    @After
    public void tearDown() throws Exception {
        integration.stop();
    }

    @Test
    public void testRoutes() throws Exception {
        // Adjust routes
        List<RouteDefinition> routes = context.getRouteDefinitions();

        routes.get(0).adviceWith(context, new AdviceWithRouteBuilder() {
            @Override
            public void configure() throws Exception {
                // Replace "from" endpoint with direct:start
                replaceFromWith("direct:start");
                // Mock and skip result endpoint
                mockEndpoints("log:*");
            }
        });

        MockEndpoint resultEndpoint = context.getEndpoint("mock:log:foo", MockEndpoint.class);
        // resultEndpoint.expectedMessageCount(1); // If you want to just check the number of messages
        resultEndpoint.expectedBodiesReceived("hello"); // If you want to check the contents

        // Start the integration
        integration.run();

        // Send the test message
        context.createProducerTemplate().sendBody("direct:start", "hello");

        resultEndpoint.assertIsSatisfied();
    }
}
```

それでは、気になる部分を1つずつ見ていきましょう。

プロパティーの接頭辞の使用

```
        // Set property prefix for unit testing
        System.setProperty(CamelScrExample.PROPERTY_PREFIX, "unit");
```

これにより、プロパティーの前に unit.を付けることで、設定の一部をオーバーライドできます。たとえば、**unit.from** は単体テストの **from** をオーバーライドします。

接頭辞を使用して、ルートが実行されるランタイム環境間の違いを処理できます。変更されていないバンドルを開発、テスト、および本番環境に移動することは、典型的な使用例です。

アノテーションからテスト設定を取得する

```
integration.prepare(null, ScrHelper.getScrProperties(integration.getClass().getName()));
```

ここでは、OSGi 環境で使用されるのと同じプロパティーを使用して、テストでサービスコンポーネントを設定します。

テスト用のコンポーネントのモック

```
// Fake a component for test
context.addComponent("amq", new MockComponent());
```

テストで使用できないコンポーネントをこのようにモックして、ルートを開始できるようにすることができます。

テスト用ルートの調整

```
// Adjust routes
List<RouteDefinition> routes = context.getRouteDefinitions();

routes.get(0).adviceWith(context, new AdviceWithRouteBuilder() {
    @Override
    public void configure() throws Exception {
        // Replace "from" endpoint with direct:start
        replaceFromWith("direct:start");
        // Mock and skip result endpoint
        mockEndpoints("log:*");
    }
});
```

Camel の AdviceWith 機能を使用すると、ルートをテスト用に変更できます。

ルーターの起動

```
// Start the integration
integration.run();
```

ここで、サービスコンポーネントを開始し、それとともにルートも開始します。

テストメッセージの送信

```
    // Send the test message
    context.createProducerTemplate().sendBody("direct:start", "hello");
```

ここでは、テスト中のルートにメッセージを送信します。

274.5. APACHE KARAF でバンドルを実行する

バンドルが **mvn install** でビルドされると、デプロイメントする準備が整います。 バンドルを Apache Karaf にデプロイするには、Karaf コマンドラインで次の手順を実行します。

バンドルを Apache Karaf にデプロイする

```
# Add Camel feature repository
karaf@root> features:chooseurl camel 2.15-SNAPSHOT

# Install camel-scr feature
karaf@root> features:install camel-scr

# Install commons-lang, used in the example route to validate parameters
karaf@root> osgi:install mvn:commons-lang/commons-lang/2.6

# Install and start your bundle
karaf@root> osgi:install -s mvn:example/camel-scr-example/1.0-SNAPSHOT

# See how it's running
karaf@root> log:tail -n 10

Press ctrl-c to stop watching the log.
```

274.5.1. デフォルト設定のオーバーライド

デフォルトでは、サービスコンポーネントの設定 PID は、そのクラスの完全修飾名と同じです。Karaf の **config:*** コマンドを使用して、サンプルバンドルのプロパティーを変更できます。

プロパティーをオーバーライドする

```
# Override 'messageOk' property
karaf@root> config:propset -p example.CamelScrExample messageOk "This is better logging"
```

または、Karaf の **etc** フォルダーにあるプロパティーファイルを編集して、設定を変更することもできます。

274.5.2. Camel SCR バンドルをテンプレートとして使用する

たとえば **from → to** など、頻繁に使用する統合パターンを実装する Camel SCR バンドルがあるとします。成功/失敗のログ記録と再配信は、サンプルルートが実装するパターンでもあります。インスタンスごとに個別のバンドルを作成したくない場合があります。心配はいりません。SCR が対応します。

サービスコンポーネントの設定 PID を作成しますが、末尾にダッシュを追加すると、SCR はその設定を使用してコンポーネントの新しいインスタンスを作成します。

新しいサービスコンポーネントインスタンスの作成

```
# Create a PID with a tail
karaf@root> config:edit example.CamelScrExample-anotherone

# Override some properties
karaf@root> config:propset camelContextId my-other-context
karaf@root> config:propset to "file://removeme?fileName=removemetoo.txt"

# Save the PID
karaf@root> config:update
```

これにより、オーバーライドされたプロパティーで新しい CamelContext が開始されます。なんと便利なんでしょう。

274.6. 注記

サービスコンポーネントをテンプレートとして設計する場合、通常、"tailed" 設定なしで、つまり既定の設定で開始することは望ましくありません。

サービスコンポーネントがデフォルトの設定で開始しないようにするには **policy = ConfigurationPolicy.REQUIRE `to the class level `@Component** アノテーションに追加します。

第275章 XML SECURITY DATAFORMAT

Camel バージョン 2.0 以降で利用可能

XMLSecurity データフォーマットは、ドキュメント、エレメント、およびエレメントコンテンツレベルでの XML ペイロードの暗号化と復号化を容易にします (XPath を使用した同時マルチノード暗号化/復号化を含む)。XML 署名仕様を使用してメッセージに署名するには、Camel XML Security コンポーネントを参照してください。

暗号化機能は、Apache XML Security (Santuario) プロジェクトを使用してサポートされる形式に基づいています。対称暗号化/復号化は現在、トリプル DES および AES (128、192、および 256) 暗号化形式を使用してサポートされています。追加フォーマットは、必要に応じて後で簡単に追加できます。この機能により、Camel ユーザーは、ルートに配信または受信されている間にペイロードを暗号化/復号化できます。

Camel 2.9 以降で利用可能
XMLSecurity データ形式は、非対称キー暗号化をサポートしています。この暗号化モデルでは、対称キーが生成され、XML コンテンツの暗号化または復号化を実行するために使用されます。このコンテンツ暗号化キーは、受信者の公開キーをキー暗号化キーとして利用する非対称暗号化アルゴリズムを使用して暗号化されます。非対称鍵暗号化アルゴリズムを使用すると、受信者の秘密鍵の所有者のみが、生成された対称暗号化鍵にアクセスできるようになります。したがって、秘密鍵の所有者だけがメッセージをデコードできます。XMLSecurity データフォーマットは、非対称キー暗号化を使用してメッセージコンテンツと暗号化キーを暗号化および復号化するために必要なすべてのロジックを処理します。

XMLSecurity データ形式では、暗号化するコンテンツを選択する XPath クエリーを処理する際の名前空間のサポートも改善されています。名前空間定義マッピングは、データ形式設定の一部として含めることができます。これにより、XPath クエリーとターゲット xml ドキュメントの接頭辞値が同等の文字列でない場合でも、真の名前空間の一致が可能になります。

275.1. XML セキュリティーオプション

XML セキュリティーデータ形式は、以下に示す 12 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
xmlCipherAlgorithm	**TRIPLEDES**	**String**	XML メッセージコンテンツの暗号化/復号化に使用される暗号アルゴリズム。選択できるのは以下の通りです: XMLCipher.TRIPLEDES XMLCipher.AES_128 XMLCipher.AES_128_GCM XMLCipher.AES_192 XMLCipher.AES_192_GCM XMLCipher.AES_256 XMLCipher.AES_256_GCM XMLCipher.SEED_128 XMLCipher.CAMELLIA_128 XMLCipher.CAMELLIA_192 XMLCipher.CAMELLIA_256 デフォルト値は MLCipher.TRIPLEDES です。
passPhrase		**String**	コンテンツを暗号化/復号化するための passPhrase として使用される文字列。passPhrase を指定する必要があります。passPhrase が指定されていない場合は、デフォルトの passPhrase が使用されます。passPhrase は、適切な暗号化アルゴリズムと組み合わせて使用する必要があります。たとえば、TRIPLEDES を使用すると、passPhase を別の 24 バイトキーのみにすることができます。

名前	デフォルト	Java タイプ	説明
secureTag		String	暗号化/復号化のために選択された XML 要素への XPath 参照。タグが指定されていない場合、ペイロード全体が暗号化/復号化されます。
secureTagContents	false	Boolean	XML 要素を暗号化するか、XML 要素のコンテンツを暗号化するかを指定するブール値 false = 要素レベル true = 要素コンテンツレベル
keyCipherAlgorithm	RSA_OAEP	String	非対称キーの暗号化/復号化に使用される暗号アルゴリズム。選択できるのは以下の通りです: XMLCipher.RSA_v1dot5 XMLCipher.RSA_OAEP XMLCipher.RSA_OAEP_11 デフォルト値は XMLCipher.RSA_OAEP です。
recipientKeyAlias		String	非対称キーの暗号化または復号化を実行するときに、KeyStore から受信者の公開キーまたは秘密キーを取得するときに使用される Key Alias。
keyOrTrustStoreParametersId		String	送信者の trustStore または受信者の keyStore を表す KeyStore インスタンスを作成およびロードするための設定オプションに使用される、レジストリーで検索する KeyStore インスタンスを参照します。
keyPassword		String	KeyStore から秘密鍵を取得するために使用されるパスワード。この鍵は、非対称復号化に使用されます。
digestAlgorithm	SHA1	String	RSA OAEP アルゴリズムで使用するダイジェストアルゴリズム。XMLCipher.SHA1 XMLCipher.SHA256 XMLCipher.SHA512 デフォルト値は XMLCipher.SHA1 です。
mgfAlgorithm	MGF1_SHA1	String	RSA OAEP アルゴリズムで使用する MGF アルゴリズム。選択できるのは以下の通りです: EncryptionConstants.MGF1_SHA1 EncryptionConstants.MGF1_SHA256 EncryptionConstants.MGF1_SHA512 デフォルト値は EncryptionConstants.MGF1_SHA1 です。
addKeyValueForEncryptedKey	true	Boolean	セッション鍵の暗号化に使用される公開鍵を EncryptedKey 構造体の KeyValue として追加するかどうか。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

275.1.1. 鍵暗号アルゴリズム

Camel 2.12.0 の時点で、デフォルトの鍵暗号アルゴリズムは XMLCipher.RSA_v1dot5 ではなく XMLCipher.RSA_OAEP になりました。XMLCipher.RSA_v1dot5 の使用は、さまざまな攻撃を避けるため推奨されていません。鍵暗号アルゴリズムとして RSA v1.5 を使用する要求は、鍵暗号アルゴリズムとして明示的に設定されていない限り拒否されます。

275.2. MARSHAL

ペイロードを暗号化するには、**marshal** プロセッサーをルートに適用し、続いて **secureXML()** タグを適用する必要があります。

275.3. UNMARSHAL

ペイロードを復号化するには、**unmarshal** プロセッサーをルートに適用し、続いて **secureXML()** タグを適用する必要があります。

275.4. 例

以下に、ドキュメント、エレメント、およびコンテンツレベルでマーシャリングを実行する方法の例をいくつか示します。

275.4.1. フルペイロードの暗号化/復号化

```
from("direct:start")
    .marshal().secureXML()
    .unmarshal().secureXML()
    .to("direct:end");
```

275.4.2. 部分的なペイロードコンテンツのみの暗号化/復号化

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

275.4.3. 部分的なマルチノードペイロードコンテンツのみの暗号化/復号化

```
String tagXPATH = "//cheesesites/*/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

275.4.4. passPhrase (password) の選択による部分的なペイロードコンテンツのみの暗号化/復号化

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
String passPhrase = "Just another 24 Byte key";
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .to("direct:end");
```

275.4.5. passPhrase (password) と Algorithm を使用した部分的なペイロードコンテンツのみの暗号化/復号化

```
import org.apache.xml.security.encryption.XMLCipher;
....
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
String passPhrase = "Just another 24 Byte key";
String algorithm= XMLCipher.TRIPLEDES;
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .to("direct:end");
```

275.4.6. 名前空間をサポートする部分的なペイロードコンテンツ

Java DSL

```
final Map<String, String> namespaces = new HashMap<String, String>();
namespaces.put("cust", "http://cheese.xmlsecurity.camel.apache.org/");

final KeyStoreParameters tsParameters = new KeyStoreParameters();
tsParameters.setPassword("password");
tsParameters.setResource("sender.ts");

context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("direct:start")
          .marshal().secureXML("//cust:cheesesites/italy", namespaces, true, "recipient",
                               testCypherAlgorithm, XMLCipher.RSA_v1dot5, tsParameters)
          .to("mock:encrypted");
    }
}
```

-

Spring XML

camelContext 定義の一部として定義されている名前空間接頭辞は、**secureXML** 要素のデータ形式 **secureTag** 属性内のコンテキストで再利用できます。

```
<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
  <route>
    <from uri="direct://start"/>
      <marshal>
        <secureXML secureTag="//cheese:cheesesites/italy"
                   secureTagContents="true"/>
      </marshal>
      ...
```

275.4.7. 対称キーの暗号化

Spring XML Sender

```
<!-- trust store configuration -->
<camel:keyStoreParameters id="trustStoreParams" resource="./sender.ts" password="password"/>

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
  <route>
    <from uri="direct://start"/>
      <marshal>
        <secureXML secureTag="//cheese:cheesesites/italy"
                   secureTagContents="true"
                   xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                   keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                   recipientKeyAlias="recipient"
                   keyOrTrustStoreParametersId="trustStoreParams"/>
      </marshal>
      ...
```

Spring XML 受信者

```
<!-- key store configuration -->
<camel:keyStoreParameters id="keyStoreParams" resource="./recipient.ks" password="password" />

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
  <route>
    <from uri="direct://encrypted"/>
      <unmarshal>
        <secureXML secureTag="//cheese:cheesesites/italy"
                   secureTagContents="true"
                   xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                   keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                   recipientKeyAlias="recipient"
```

```
                keyOrTrustStoreParametersId="keyStoreParams"
                keyPassword="privateKeyPassword" />
        </unmarshal>
        ...
```

275.5. 依存関係

このデータ形式は **camel-xmlsecurity** コンポーネント内で提供されます。

第276章 SEDA コンポーネント

Camel バージョン 1.1 以降で利用可能

seda: コンポーネントは非同期 SEDA 動作を提供するため、BlockingQueue でメッセージがエクスチェンジされ、プロデューサーとは別のスレッドでコンシューマーが呼び出されます。

キューは **単一の** CamelContext 内でのみ表示されることに注意してください。**CamelContext** インスタンス間で通信する場合 (たとえば、Web アプリケーション間の通信) は、仮想マシン コンポーネントを参照してください。

このコンポーネントは、メッセージがまだ処理されていない間に仮想マシンが終了した場合、いかなる種類の永続化または回復も実装しません。永続性、信頼性、または分散型 SEDA が必要な場合は、JMS または ActiveMQ を使用してみてください。

ヒント: *同期* Direct コンポーネントは、プロデューサーがメッセージ交換を送信するときに、すべてのコンシューマーの同期呼び出しを提供します。

276.1. URI 形式

```
seda:someName[?options]
```

someName は、現在の CamelContext 内のエンドポイントを一意に識別する任意の文字列にすることができます。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

276.2. オプション

SEDA コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
queueSize (advanced)	SEDA キューのデフォルトの最大容量 (つまり、保持できるメッセージの数) を設定します。		int
concurrentConsumers (consumer)	交換を処理する同時スレッドのデフォルト数を設定します。	1	int
defaultQueueFactory (advanced)	デフォルトのキューファクトリーを設定します。		Exchange>
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SEDA エンドポイントは、URI 構文を使用して設定されます。

```
seda:name
```

パスおよびクエリーパラメーターを使用します。

276.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** キューの名前		文字列

276.2.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
size (common)	SEDA キューの最大容量 (つまり、保持できるメッセージの数)。	2147483647	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
limitConcurrentConsumers (consumer)	concurrentConsumers の数を最大 500 に制限するかどうか。デフォルトでは、エンドポイントがより大きな数で設定されている場合、例外が出力されます。このチェックを無効にするには、このオプションをオフにします。	true	boolean

名前	説明	デフォルト	タイプ
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe メッセージングに SEDA を使用できます。つまり、メッセージを SEDA キューに送信し、各コンシューマーにメッセージのコピーを受信させることができます。有効にすると、このオプションはすべてのコンシューマーエンドポイントで指定する必要があります。	false	boolean
pollTimeout (consumer)	ポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
purgeWhenStopping (consumer)	コンシューマー/ルートを停止するときにタスクキューをパージするかどうか。これにより、キューに保留中のメッセージが破棄されるため、より迅速に停止できます。	false	boolean
blockWhenFull (producer)	満杯の SEDA キューにメッセージを送信するスレッドが、キューの容量がなくなるまでブロックするかどうか。デフォルトでは、キューがいっぱいであることを示す例外が出力されます。このオプションを有効にすると、呼び出しスレッドは代わりにブロックされ、メッセージが受け入れられるまで待機します。	false	boolean
discardIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーがメッセージを破棄する (メッセージをキューに追加しない) かどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーが例外を出力して失敗するかどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
timeout (producer)	SEDA プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト (ミリ秒単位)。0 または負の値を使用して、タイムアウトを無効にすることができます。	30000	long

名前	説明	デフォルト	タイプ
waitForTaskToComplete (producer)	続行する前に呼び出し元が非同期タスクの完了を待つかどうかを指定するオプション。次の 3 つのオプションがサポートされています: Always、Never、または IfReplyExpected。最初の 2 つの値は一目瞭然です。最後の値 IfReplyExpected は、メッセージが Request Reply ベースの場合にのみ待機します。デフォルトのオプションは IfReplyExpected です。	IfReplyExpected	WaitForTaskToComplete
queue (advanced)	エンドポイントで使用されるキューインスタンスを定義します。このオプションは、カスタムキューインスタンスを使用するまれなユースケースのみを対象としています。		BlockingQueue
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

276.3. BLOCKINGQUEUE 実装の選択

Camel 2.12 以降で利用可能

デフォルトでは、SEDA コンポーネントは常に LinkedBlockingQueue をインスタンス化しますが、別の実装を使用できます。独自の BlockingQueue 実装を参照できます。この場合、サイズオプションは使用されません。

```
<bean id="arrayQueue" class="java.util.ArrayBlockingQueue">
  <constructor-arg index="0" value="10" ><!-- size -->
  <constructor-arg index="1" value="true" ><!-- fairness -->
</bean>

<!-- ... and later -->
<from>seda:array?queue=#arrayQueue</from>
```

または、BlockingQueueFactory 実装を参照できます。LinkedBlockingQueueFactory、ArrayBlockingQueueFactory、および PriorityBlockingQueueFactory の 3 つの実装が提供されています。

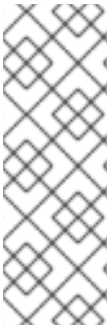
```
<bean id="priorityQueueFactory"
class="org.apache.camel.component.seda.PriorityBlockingQueueFactory">
  <property name="comparator">
    <bean class="org.apache.camel.demo.MyExchangeComparator" />
  </property>
</bean>

<!-- ... and later -->
<from>seda:priority?queueFactory=#priorityQueueFactory&size=100</from>
```

276.4. REQUEST REPLY (リクエストリプライ) の利用

SEDA コンポーネントは、発信者が非同期ルートの完了を待機する Request Reply の使用をサポートしています。たとえば、以下のようになります。

```
from("mina:tcp://0.0.0.0:9876?textline=true&sync=true").to("seda:input");

from("seda:input").to("bean:processInput").to("bean:createResponse");
```

上記の経路では、受信リクエストを受け入れるポート 9876 に TCP リスナーがあります。リクエストは **seda:input** キューにルーティングされます。Request Reply メッセージなので、レスポンスを待ちます。**seda:input** キューのコンシューマーが完了すると、応答を元のメッセージ応答にコピーします。



注記

2.2 まで: 2 つのエンドポイントでのみ動作SEDA または 仮想マシン を介した Request Reply の使用は、2 つのエンドポイントでのみ動作します。A → B → C などに送信してエンドポイントをチェーンする **ことはできません**。A → B の間のみ。理由は、実装ロジックがかなり単純だからです。3 つ以上のエンドポイントをサポートすると、待機中のスレッド間の順序付けと通知を適切に処理するためのロジックがより複雑になります。これは **Camel 2.3** 以降で改善され、好きなだけエンドポイントをチェーンできるようになりました。

276.5. 同時利用者

デフォルトでは、SEDA エンドポイントは単一のコンシューマースレッドを使用しますが、コンシューマースレッドを同時に使用するように設定できます。したがって、スレッドプールの代わりに次を使用できます。

```
from("seda:stageName?concurrentConsumers=5").process(...)
```

この 2 つの違いについては、**スレッドプールは** 実行時に負荷に応じて動的に増減できますが、同時コンシューマーの数は常に固定されていることに注意してください。

276.6. THREAD POOLS

次のようにして、SEDA エンドポイントにスレッドプールを追加することに注意してください。

```
from("seda:stageName").thread(5).process(...)
```

2 つの **BlockQueues** で終わる可能性があります。1 つは SEDA エンドポイントから、もう 1 つはスレッドプールのワークキューからのものですが、これは望ましくない場合があります。代わりに、同期と非同期の両方でメッセージを処理できるスレッドプールを使用して Direct エンドポイントを設定することをお勧めします。以下に例を示します。

```
from("direct:stageName").thread(5).process(...)
```

また、**concurrentConsumers** オプションを使用して、SEDA エンドポイントでメッセージを処理するスレッドの数を直接設定することもできます。

276.7. 例

以下のルートでは、SEDA キューを使用してこの非同期キューにリクエストを送信し、別のスレッドでさらに処理するためにファイアアンドフォーゲットメッセージを送信し、このスレッドで一定の応答を元の呼び出し元に返すことができるようにします。

Hello World メッセージを送信し、応答が OK であることを期待します。

Hello World メッセージは、さらに処理するために別のスレッドの SEDA キューから消費されます。これは単体テストからのものであるため、単体テストでアサーションを実行できる **mock** エンドポイントに送信されます。

276.8. MULTIPLECONSUMERS の使用

Camel 2.2 で利用可能

この例では、2 つのコンシューマーを定義し、Spring Bean として登録しました。

seda foo エンドポイントで **multipleConsumers=true** を指定したので、これら 2 つのコンシューマーに、一種の pub-sub スタイルメッセージングとしてメッセージの独自のコピーを受信させることができます。

Bean は単体テストの一部であるため、単純にメッセージをモックエンドポイントに送信しますが、@Consume を使用して seda キューから消費する方法に注目してください。

276.9. キュー情報の展開

必要に応じて、次の方法で JMX を使用せずにキューサイズなどの情報を取得できます。

```
SedaEndpoint seda = context.getEndpoint("seda:xxxx");
int size = seda.getExchanges().size();
```

276.10. 関連項目

- VM
- Disruptor
- Direct
- 非同期

第277章 JAVA オブジェクトのシリアル化 DATAFORMAT

Camel バージョン 2.12 以降で利用可能

シリアル化は、標準の Java シリアル化メカニズムを使用して、バイナリーペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトをバイナリー BLOB にマーシャリングするデータ形式です。
たとえば、次の例では Java シリアル化を使用してバイナリーファイルを非整列化し、それを ObjectMessage として ActiveMQ に送信します。

```
from("file://foo/bar").
  unmarshal().serialization().
  to("activemq:Some.Queue");
```

277.1. オプション

Java Object Serialization データ形式は、以下にリストされている 1 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

277.2. 依存関係

このデータ形式は **camel-core** で提供されるため、追加の依存関係は必要ありません。

第278章 SERVICENOW コンポーネント

Camel バージョン 2.18 以降で利用可能

ServiceNow コンポーネントは、REST API を介して ServiceNow プラットフォームへのアクセスを提供します。

注記

Camel 2.18.1 から、コンポーネントは ServiceNow プラットフォームの複数のバージョンをサポートし、デフォルトは Helsinki です。対応バージョンは表278.1「API マッピング」と 表278.2「API マッピング」 です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servicenow</artifactId>
  <version>${camel-version}</version>
</dependency>
```

278.1. URI 形式

```
servicenow://instanceName?[options]
```

278.2. オプション

ServiceNow コンポーネントは、以下に示す 14 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
instanceName (advanced)	ServiceNow インスタンス名		String
configuration (advanced)	ServiceNow のデフォルト設定		ServiceNowConfiguration
apiUrl (producer)	ServiceNow REST API URL		String
userName (security)	ServiceNow ユーザーアカウント名		String
password (security)	ServiceNow アカウントのパスワード		String
oauthClientId (security)	OAuth2 ClientID		String

名前	説明	デフォルト	タイプ
oauthClientSecret (security)	OAuth2 ClientSecret		String
oauthTokenUrl (security)	OAuth トークンの URL		String
proxyHost (advanced)	プロキシーホスト名		String
proxyPort (advanced)	プロキシーポート番号		Integer
proxyUserName (security)	プロキシー認証用のユーザー名		String
proxyPassword (security)	プロキシー認証のパスワード		String
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ServiceNow エンドポイントは、URI 構文を使用して設定されます。

```
servicenow:instanceName
```

パスおよびクエリーパラメーターを使用します。

278.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
instanceName	**必須** ServiceNow インスタンス名		String

278.2.2. クエリーパラメーター(44 個のパラメーター):

名前	説明	デフォルト	タイプ
display (producer)	このパラメーターを true に設定すると、インジケーターの 表示 フィールドが選択されているスコアカードのみが返されます。このパラメーターを all に設定すると、任意の表示フィールド値を持つスコアカードが返されます。このパラメーターはデフォルトで true です。	true	String
displayValue (producer)	参照フィールドの表示値 (true)、実際の値 (false)、またはその両方 (すべて) を返します (デフォルト: false)	false	String
excludeReferenceLink (producer)	参照フィールドのテーブル API リンクを除外する場合は True (デフォルト: false)		Boolean
favorites (producer)	このパラメーターを true に設定すると、クエリーを実行しているユーザーのお気に入りのスコアカードのみが返されます。		Boolean
includeAggregates (producer)	このパラメーターを true に設定すると、集計がすでに適用されている場合を含め、インジケーターで使用可能なすべての集計が常に返されます。値が指定されていない場合、このパラメーターはデフォルトで false になり、集計は返されません。		Boolean
includeAvailableAggregates (producer)	このパラメーターを true に設定すると、集計が適用されていない場合に、インジケーターに使用可能なすべての集計が返されます。値が指定されていない場合、このパラメーターはデフォルトで false になり、集計は返されません。		Boolean
includeAvailableBreakdowns (producer)	このパラメーターを true に設定すると、インジケーターで利用可能なすべての内訳が返されます。値が指定されていない場合、このパラメーターはデフォルトで false になり、ブレークダウンは返されません。		Boolean
includeScoreNotes (producer)	スコアに関連付けられたすべての音符を返すには、このパラメーターを true に設定します。note 要素には、メモテキストのほか、メモが追加されたときの作成者とタイムスタンプが含まれます。		Boolean
includeScores (producer)	スコアカードのすべてのスコアを返すには、このパラメーターを true に設定します。値が指定されていない場合、このパラメーターはデフォルトで false に設定され、最新のスコア値のみが返されます。		Boolean
inputDisplayValue (producer)	入力フィールドの生の値を設定する場合は True (デフォルト: false)		Boolean

名前	説明	デフォルト	タイプ
key (producer)	キーインジケーターのスコアカードのみを返すには、このパラメーターを true に設定します。		Boolean
models (producer)	リクエストモデルとレスポンスモデルの両方を定義する		String
perPage (producer)	各クエリーが返すことができるスコアカードの最大数を入力します。デフォルトでは、この値は 10 で、最大値は 100 です。	10	Integer
release (producer)	ターゲットとする ServiceNow リリース、デフォルトは Helsinki https://docs.servicenow.com を参照	HELSINKI	ServiceNowRelease
requestModels (producer)	リクエストモデルを定義します		String
resource (producer)	デフォルトのリソースは、ヘッダー CamelServiceNowResource でオーバーライドできます		String
responseModels (producer)	レスポンスモデルを定義します		String
sortBy (producer)	結果を並べ替えるときに使用する値を指定します。デフォルトでは、クエリーはレコードを値で並べ替えます。		String
sortDir (producer)	昇順または降順のソート方向を指定します。デフォルトでは、クエリーはレコードを降順で並べ替えます。昇順でソートするには、sysparm_sortdir=asc を使用します。		String
suppressAutoSysField (producer)	システムフィールドの自動生成を抑制する場合は True (デフォルト: false)		Boolean
suppressPaginationHeader (producer)	レスポンスから Link ヘッダーを削除するには、この値を true に設定します。Link ヘッダーを使用すると、クエリーに一致するレコードの数がクエリーの制限を超えた場合に、データの追加ページをリクエストできます		Boolean
table (producer)	デフォルトのテーブルは、ヘッダー CamelServiceNowTable でオーバーライドできます		String
target (producer)	ターゲットを持つスコアカードのみを返すには、このパラメーターを true に設定します。		Boolean

名前	説明	デフォルト	タイプ
topLevelOnly (producer)	親がカタログであるカテゴリーのみを取得します。		Boolean
apiVersion (advanced)	ServiceNow REST API バージョン、デフォルトは最新		String
dateFormat (advanced)	Json シリアライゼーション/デシリアライゼーションに使用される日付形式	yyyy-MM-dd	String
dateTimeFormat (advanced)	Json シリアライゼーション/デシリアライゼーションに使用される日時形式	yyyy-MM-dd HH:mm:ss	String
httpClientPolicy (advanced)	http クライアントを設定するには		HTTPClientPolicy
mapper (advanced)	リクエスト/レスポンスに使用する Jackson の ObjectMapper を設定します		ObjectMapper
proxyAuthorizationPolicy (advanced)	プロキシー認証を設定するには		ProxyAuthorization ポリシー
retrieveTargetRecordOn Import (advanced)	インポートセット API を使用する場合に対象レコードを取得するには、このパラメーターを true に設定します。インポートセットの結果は、ターゲットレコードに置き換えられます。	false	Boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
timeFormat (advanced)	Json シリアル化/逆シリアル化に使用される時刻形式	HH:mm:ss	String
proxyHost (proxy)	プロキシーホスト名		String
proxyPort (proxy)	プロキシーポート番号		Integer
apiUrl (security)	ServiceNow REST API URL		String
oauthClientId (security)	OAuth2 ClientID		String

名前	説明	デフォルト	タイプ
oauthClientSecret (security)	OAuth2 ClientSecret		String
oauthTokenUrl (security)	OAuth トークンの URL		String
password (security)	**必須** ServiceNow アカウントのパスワード。入力する必要があります。		String
proxyPassword (security)	プロキシー認証のパスワード		String
proxyUserName (security)	プロキシー認証用のユーザー名		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。See http://camel.apache.org/camel-configuration-utilities.html		SSLContextParameters
userName (security)	**必須** ServiceNow ユーザーアカウント名。入力する必要があります。		文字列

278.3. ヘッダー

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowResource	String	-	-	アクセスするリソース
CamelServiceNowAction	String	-	-	実行する動作

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowActionSubject	-	-	String	アクションが適用されるサブジェクト
CamelServiceNowModel	Class	-	-	データモデル
CamelServiceNowRequestModel	Class	-	-	リクエストデータモデル
CamelServiceNowResponseModel	Class	-	-	レスポンスデータモデル
CamelServiceNowOffsetNext	-	-	-	-
CamelServiceNowOffsetPrev	-	-	-	-
CamelServiceNowOffsetFirst	-	-	-	-
CamelServiceNowOffsetLast	-	-	-	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowContentType	-	-	-	-
CamelServiceNowContentEncoding	-	-	-	-
CamelServiceNowContentMeta	-	-	-	-
CamelServiceNowSysId	String	sys_id	-	-
CamelServiceNowUserSysId	String	user_sysid	-	-
CamelServiceNowUserId	String	user_id	-	-
CamelServiceNowCartItemId	String	cart_item_id	-	-
CamelServiceNowFileName	String	file_name	-	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowTable	String	table_name	-	-
CamelServiceNowTableSysId	String	table_sys_id	-	-
CamelServiceNowEncryptionContext	String	暗号化コンテキスト	-	-
CamelServiceNowCategory	String	sysparm_category	-	-
CamelServiceNowType	String	sysparm_type	-	-
CamelServiceNowCatalog	String	sysparm_catalog	-	-
CamelServiceNowQuery	String	sysparm_query	-	-
CamelServiceNowDisplayValue	String	sysparm_display_value	displayValue	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowInputDisplayValue	Boolean	sysparm_input_display_value	inputDisplayValue	-
CamelServiceNowExcludeReferenceLink	Boolean	sysparm_exclude_reference_link	excludeReferenceLink	-
CamelServiceNowFields	String	sysparm_fields	-	-
CamelServiceNowLimit	Integer	sysparm_limit	-	-
CamelServiceNowText	String	sysparm_text	-	-
CamelServiceNowOffset	Integer	sysparm_offset	-	-
CamelServiceNowView	String	sysparm_view	-	-
CamelServiceNowSuppressAutoSysField	Boolean	sysparm_suppress_auto_sys_field	suppressAutoSysField	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowSuppressPaginationHeader	Boolean	sysparm_suppress_pagination_header	suppressPaginationHeader	-
CamelServiceNowMinFields	String	sysparm_min_fields	-	-
CamelServiceNowMaxFields	String	sysparm_max_fields	-	-
CamelServiceNowSumFields	String	sysparm_sum_fields	-	-
CamelServiceNowAvgFields	String	sysparm_avg_fields	-	-
CamelServiceNowCount	Boolean	sysparm_count	-	-
CamelServiceGroupBy	String	sysparm_group_by	-	-
CamelServiceOrderBy	String	sysparm_order_by	-	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceHaving	String	sysparm_having	-	-
CamelServiceNowUUID	String	sysparm_uuid	-	-
CamelServiceNowBreakdown	String	sysparm_breakdown	-	-
CamelServiceNowIncludeScores	Boolean	sysparm_include_scores	includeScores	-
CamelServiceNowIncludeScoreNotes	Boolean	sysparm_include_score_notes	includeScoreNotes	-
CamelServiceNowIncludeAggregates	Boolean	sysparm_include_aggregates	includeAggregates	-
CamelServiceNowIncludeAvailableBreakdowns	Boolean	sysparm_include_available_breakdowns	includeAvailableBreakdowns	-

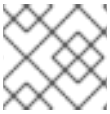
名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowIncludeAvailableAggregates	Boolean	sysparm_include_available_aggregates	includeAvailableAggregates	-
CamelServiceNowFavorites	Boolean	sysparm_favorites	favorites	-
CamelServiceNowKey	Boolean	sysparm_key	key	-
CamelServiceNowTarget	Boolean	sysparm_target	target	-
CamelServiceNowDisplay	String	sysparm_display	display	-
CamelServiceNowPerPage	Integer	sysparm_per_page	perPage	-
CamelServiceNowSortBy	String	sysparm_sortby	sortBy	-
CamelServiceNowSortDir	String	sysparm_sortdit	sortDir	-

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowContains	String	sysparm_contains	-	-
CamelServiceNowTags	String	sysparm_tags	-	-
CamelServiceNowPage	String	sysparm_page	-	-
CamelServiceNowElementsFilter	String	sysparm_elements_filter	-	-
CamelServiceNowBreakdownRelation	String	sysparm_breakdown_relation	-	-
CamelServiceNowDataSource	String	sysparm_data_source	-	-
CamelServiceNowTopLevelOnly	Boolean	sysparm_top_level_only	topLevelOnly	-
CamelServiceNowApiVersion	String	-	-	REST API のバージョン

名前	タイプ	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowResponseMeta	Map	-	-	レスポンスとともに提供されるメタデータ

表278.1 API マッピング

CamelServiceNowResource	CamelServiceNowAction	メソッド	API URI
TABLE	RETRIEVE	GET	/api/now/v1/table/{table_name}/{sys_id}
	CREATE	POST	/api/now/v1/table/{table_name}
	MODIFY	PUT	/api/now/v1/table/{table_name}/{sys_id}
	DELETE	DELETE	/api/now/v1/table/{table_name}/{sys_id}
	UPDATE	PATCH	/api/now/v1/table/{table_name}/{sys_id}
AGGREGATE	RETRIEVE	GET	/api/now/v1/stats/{table_name}
IMPORT	RETRIEVE	GET	/api/now/import/{table_name}/{sys_id}
	CREATE	POST	/api/now/import/{table_name}



注記

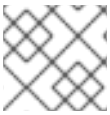
Fuji REST API ドキュメンテーション

表278.2 API マッピング

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
TABLE	RETRIEVE		GET	/api/now/v1/table/{table_name}/{sys_id}
	CREATE		POST	/api/now/v1/table/{table_name}
	MODIFY		PUT	/api/now/v1/table/{table_name}/{sys_id}
	DELETE		DELETE	/api/now/v1/table/{table_name}/{sys_id}
	UPDATE		PATCH	/api/now/v1/table/{table_name}/{sys_id}
AGGREGATE	RETRIEVE		GET	/api/now/v1/stats/{table_name}
IMPORT	RETRIEVE		GET	/api/now/import/{table_name}/{sys_id}
	CREATE		POST	/api/now/import/{table_name}
ATTACHMENT	RETRIEVE		GET	/api/now/api/now/attachment/{sys_id}
	CONTENT		GET	/api/now/attachment/{sys_id}/file
	UPLOAD		POST	/api/now/api/now/attachment/file
	DELETE		DELETE	/api/now/attachment/{sys_id}
SCORECARDS	RETRIEVE	PERFORMANCE_ANALYTICS	GET	/api/now/pa/scorecards

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
MISC	RETRIEVE	USER_ROLE_INHERITANCE	GET	/api/global/user_role_inheritance
	CREATE	IDENTIFY_RECONCILE	POST	/api/now/identifyreconcile
SERVICE_CATALOG	RETRIEVE		GET	/sn_sc/servicecatalog/catalogs/{sys_id}
	RETRIEVE	CATEGORIES	GET	/sn_sc/servicecatalog/catalogs/{sys_id}/categories
SERVICE_CATALOG_ITEMS	RETRIEVE		GET	/sn_sc/servicecatalog/items/{sys_id}
	RETRIEVE	SUBMIT_GUIDE	POST	/sn_sc/servicecatalog/items/{sys_id}/submit_guide
	RETRIEVE	CHECKOUT_GUIDE	POST	/sn_sc/servicecatalog/items/{sys_id}/checkout_guide
	CREATE	SUBJECT_CART	POST	/sn_sc/servicecatalog/items/{sys_id}/add_to_cart
	CREATE	SUBJECT_PRODUCER	POST	/sn_sc/servicecatalog/items/{sys_id}/submit_producer
SERVICE_CATALOG_CARTS	RETRIEVE		GET	/sn_sc/servicecatalog/cart
	RETRIEVE	DELIVERY_ADDRESS	GET	/sn_sc/servicecatalog/cart/delivery_address/{user_id}

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
	RETRIEVE	CHECKOUT	POST	/sn_sc/servicecatalog/cart/checkout
	UPDATE		POST	/sn_sc/servicecatalog/cart/{cart_item_id}
	UPDATE	CHECKOUT	POST	/sn_sc/servicecatalog/cart/submit_order
	DELETE		DELETE	/sn_sc/servicecatalog/cart/{sys_id}/empty
SERVICE_CATALOG_CATEGORIES	RETRIEVE		GET	/sn_sc/servicecatalog/categories/{sys_id}



注記

Helsinki REST API ドキュメント

278.4. 使用例:

インシデントを 10 件取得する

```
context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("direct:servicenow")
            .to("servicenow:{{env:SERVICENOW_INSTANCE}}"
                + "?userName={{env:SERVICENOW_USERNAME}}"
                + "&password={{env:SERVICENOW_PASSWORD}}"
                + "&oauthClientId={{env:SERVICENOW_OAUTH2_CLIENT_ID}}"
                + "&oauthClientSecret={{env:SERVICENOW_OAUTH2_CLIENT_SECRET}}"
            .to("mock:servicenow");
    }
});

FluentProducerTemplate.on(context)
    .withHeader(ServiceNowConstants.RESOURCE, "table")
    .withHeader(ServiceNowConstants.ACTION, ServiceNowConstants.ACTION_RETRIEVE)
    .withHeader(ServiceNowConstants.SYSPARM_LIMIT.getId(), "10")
    .withHeader(ServiceNowConstants.TABLE, "incident")
```

```
    .withHeader(ServiceNowConstants.MODEL, Incident.class)
    .to("direct:servicenow")
    .send();
```

第279章 SERVLET コンポーネント

Camel バージョン 2.0 以降で利用可能

servlet: コンポーネントは、公開されたサーブレットにバインドされた HTTP エンドポイントに到着する HTTP 要求を消費するための HTTP ベースのエンドポイントを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servlet</artifactId>
  <version>x.x.x</version>
  <\!-\- use the same version as your Camel core version \-->
</dependency>
```

情報: **ストリーム**。サーブレットはストリームベースです。つまり、受信した入力はストリームとして Camel に送信されます。つまり、ストリームのコンテンツを **一度** だけ読み取ることができます。もし、メッセージ本文が空のように見える場合や、何度もデータにアクセスする必要がある場合 (例: マルチキャストや再配送エラー処理)、ストリームキャッシュを使用するか、メッセージ本文を何度読んでも安全な **String** に変換する必要があります。

279.1. URI 形式

```
servlet://relative_path[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

279.2. オプション

Servlet コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
servletName (consumer)	使用するサーブレットのデフォルト名。デフォルト名は CamelServlet です。		文字列
httpRegistry (consumer)	カスタム org.apache.camel.component.servlet.HttpRegistry を使用します。		HttpRegistry
attachmentMultipart Binding (consumer)	Camel エクスチェンジで multipart/form-data を添付として自動的にバインドするかどうか。オプション attachmentMultipartBinding=true と disableStreamCache=false は一緒に使用できません。AttachmentMultipartBinding を使用するには、disableStreamCache を削除します。サーブレットの使用時にこれを有効にするには、サーブレット固有の設定が必要になる場合があるため、これはデフォルトでオフになっています。	false	boolean

名前	説明	デフォルト	タイプ
httpBinding (上級)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration を基本設定として使用するには、以下を行います。		HttpConfiguration
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に Java シリアル化を許可するかどうか。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

サーブレットエンドポイントは、URI 構文を使用して設定されます。

```
servlet:contextPath
```

パスおよびクエリーパラメーターを使用します。

279.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
contextPath	**必須** 使用するコンテキストパス		文字列

279.2.2. クエリーパラメーター(21 パラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
disableStreamCache (common)	サーブレットからの生の入力ストリームがキャッシュされるかどうかを決定します (Camel はストリームをメモリー内/ファイルへのオーバーフロー、ストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。ストリームの複数回の読み取りをサポートするためにこのオプションが false の場合、DefaultHttpBinding は要求入力ストリームをストリームキャッシュにコピーし、それをメッセージ本文に入れます。サーブレットを使用してエンドポイントをブリッジ/プロキシーする場合、メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にしてパフォーマンスを向上させることを検討してください。http/http4 プロデューサーは、デフォルトでレスポンスボディーストリームをキャッシュします。このオプションを true に設定すると、プロデューサは応答本文ストリームをキャッシュせず、応答ストリームをそのままメッセージ本文として使用します。	false	boolean
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
httpBinding (common)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		HttpBinding
async (consumer)	非同期モードで動作するようにコンシューマーを設定します	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
chunked (consumer)	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean

名前	説明	デフォルト	タイプ
httpMethodRestrict (consumer)	GET/POST/PUT など、HttpMethod が一致する場合にのみ消費を許可するために使用されます。複数のメソッドをコンマで区切って指定できます。		String
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、コンシューマーが URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse.		Integer
servletName (consumer)	使用するサーブレットの名前	Camel Servlet	String
transferException (consumer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean
attachmentMultipartBinding (consumer)	Camel エクスチェンジで multipart/form-data を添付として自動的にバインドするかどうか。オプション attachmentMultipartBinding=true と disableStreamCache=false は一緒に使用できません。AttachmentMultipartBinding を使用するには、disableStreamCache を削除します。サーブレットの使用時にこれを有効にするには、サーブレット固有の設定が必要になる場合があるため、これはデフォルトでオフになっています。	false	boolean
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 または存在しない場合に、HTTP リクエストにコンテンツがあるかどうかを先行チェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
optionsEnabled (consumer)	このサーブレットコンシューマーに対して HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは、OPTIONS はオフになっています。	false	boolean
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
mapHttpMessageBody (advanced)	このオプションが true の場合、交換の IN exchange ボディは HTTP ボディにマップされます。これを false に設定すると、HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、交換の IN exchange Form Encoded ボディは HTTP にマップされます。これを false に設定すると、HTTP Form Encoded ボディマッピングが回避されます。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、交換の IN exchange ヘッダーは HTTP ヘッダーにマップされます。これを false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

279.3. メッセージヘッダー

Camel は、HTTP コンポーネントと同じメッセージヘッダーを適用します。

Camel は、**すべての request.parameter** および **request.headers** も生成します。たとえば、クライアントリクエストの URL が **http://myserver/myserver?orderid=123** の場合、エクスチェンジには、値が 123 の **orderid** という名前のヘッダーが含まれます。

279.4. 使用方法

サーブレットコンポーネントによって生成されたエンドポイントからのみ使用できます。したがって、Camel ルートへの入力としてのみ使用する必要があります。他の HTTP エンドポイントに対して HTTP リクエストを発行するには、HTTP Component コンポーネントを使用します。

279.5. CAMEL JAR をアプリサーバーのブートクラスパスに配置する

camel-core、**camel-servlet** などの Camel JAR をアプリケーションサーバーのブートクラスパス (例: 通常 lib ディレクトリ) に配置する場合、サーブレットマッピングリストは、アプリケーションサーバーにデプロイされた複数の Camel アプリケーション間で共有されていることに注意してください。

Camel JAR をアプリケーションサーバーのブートクラスパスに配置することは、一般的にベストプラクティスではないことに注意してください。

したがって、このような状況では、各 Camel アプリケーションでカスタムの一意のサーブレット名を定義する **必要があります**。たとえば、**web.xml** で次のように定義します。

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>org.apache.camel.component.servlet.CamelHttpTransportServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

そして、Camel エンドポイントにサーブレット名も含めます。

```
<route>
  <from uri="servlet://foo?servletName=MyServlet"/>
  ...
</route>
```

Camel 2.11 以降、Camel はこの重複を検出し、アプリケーションの起動に失敗します。次のように、サーブレットの init-parameter ignoreDuplicateServletName を true に設定することで、この重複を無視するように制御できます。

```
  <servlet>
    <servlet-name>CamelServlet</servlet-name>
    <display-name>Camel Http Transport Servlet</display-name>
    <servlet-class>org.apache.camel.component.servlet.CamelHttpTransportServlet</servlet-class>
    <init-param>
      <param-name>ignoreDuplicateServletName</param-name>
      <param-value>true</param-value>
    </init-param>
  </servlet>
```

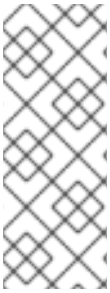
ただし、この重複や予期しない副作用を避けるために、Camel アプリケーションごとに一意のサーブレット名を使用することを **強くお勧めします**。

279.6. 例

情報: Camel 2.7 以降では、Spring Web アプリケーションで サーブレット を使用する方が簡単です。詳細は、Servlet Tomcat Tomcat の例 を参照してください。

このサンプルでは、**http://localhost:8080/camel/services/hello** で HTTP サービスを公開するルートを定義します。
最初に、通常の Web コンテナーまたは OSGi サービスを介して CamelHttpTransportServlet を公開する必要があります。
Web.xml ファイルを使用して、次のように CamelHttpTransportServlet を公開します。

次に、次のようにルートを定義できます。



注記

camel-servlet エンドポイントの相対パスを指定する HTTP トランスポートを公開されたサーブレットにバインドしていて、サーブレットのアプリケーションコンテキストパスがわからないため、**camel-servlet** エンドポイントは相対パスを使用してエンドポイントの URL を指定します。クライアントは、サーブレット公開アドレス **("http://localhost:8080/camel/services") + RELATIVE_PATH("/hello")** を介して、**camel-servlet** エンドポイントにアクセスできます。

279.6.1. Spring 3.x 使用時のサンプル

サーブレット Tomcat の例を参照してください。

279.6.2. Spring 2.x 使用時のサンプル

Camel/Spring アプリケーションで Servlet コンポーネントを使用する場合、多くの場合、Servlet コンポーネントの開始 **後に** Spring ApplicationContext をロードする必要があります。これは、**ContextLoaderListener** の代わりに Spring の **ContextLoaderServlet** を使用することで実現できます。その場合、次のように CamelHttpTransportServlet の後に **ContextLoaderServlet** を開始する必要があります。

```
<web-app>
  <servlet>
      <servlet-name>CamelServlet</servlet-name>
      <servlet-class>
         org.apache.camel.component.servlet.CamelHttpTransportServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
      <servlet-name>SpringApplicationContext</servlet-name>
      <servlet-class>
         org.springframework.web.context.ContextLoaderServlet
      </servlet-class>
      <load-on-startup>2</load-on-startup>
  </servlet>
<web-app>
```

279.6.3. OSGi 使用時のサンプル

Camel 2.6.0 から、このように SpringDM を使用して、CamelHttpTransportServlet を OSGi サービスとして公開できます。

次に、このサービスを camel ルートで次のように使用します。

Camel 2.6 より前のバージョンでは、**Activator** を使用して、OSGi プラットフォームで CamelHttpTransportServlet を公開できます。

279.6.4. Spring-Boot での使用

Camel 2.19.0 以降、**camel-servlet-starter** ライブラリーは "/camel/*" コンテキストパスの下のすべての残りのエンドポイントを自動的にバインドします。次の表は、camel-servlet-starter ライブラリーで使用できる追加の設定プロパティーをまとめたものです。Camel サーブレットの自動マッピングも無効にすることができます。

Spring-Boot プロパティー	デフォルト	説明
camel.component.servlet.mapping.enabled	**true**	サーブレットコンポーネントの Spring Web コンテキストへの自動マッピングを有効にします
camel.component.servlet.mapping.context-path	**/camel/***	自動マッピングのためにサーブレットコンポーネントによって使用されるコンテキストパス
camel.component.servlet.mapping.servlet-name	**CamelServlet**	Camel サーブレットの名前

279.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- サーブレット Tomcat の例
- サーブレット Tomcat No Spring の例
- HTTP
- Jetty

279.8. SERVLETLISTENER COMPONENT

Camel 2.11 から利用可能

このコンポーネントは、Web アプリケーションで Camel アプリケーションをブートストラップするために使用されます。たとえば、事前に Camel をブートストラップする独自の方法を見つけるか、Spring などのサードパーティーフレームワークに依存する必要があります。

注記

サイドバー このコンポーネントは Servlet 2.x 以降をサポートしているため、古い Web コンテナーでも動作します。これがこのコンポーネントの目標です。ただし、Servlet 2.x では web.xml ファイルを設定として使用する必要があります。Servlet 3.x コンテナーの場合、@WebListener を使用してアノテーション駆動型設定を使用して Camel をブーストし、Camel をブーストする独自のクラスを実装できます。それでも、エンドユーザーが Camel を簡単に設定できるようにするにはどうしたらいいかという課題が残りますが、これは昔ながらの web.xml ファイルを使えば無料で手に入ります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servletlistener</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

279.8.1. 使用

抽象クラス **org.apache.camel.component.servletlistener.CamelServletContextListener** の次の実装のいずれかを選択する必要があります。

- **JndiRegistry** を使用してレジストリーに JNDI を活用する **JndiCamelServletContextListener**。
- **SimpleRegistry** を使用して **java.util.Map** をレジストリーとして活用する **SimpleCamelServletContextListener**。

これを使用するには、以下に示すように、**WEB-INF/web.xml** ファイルで **org.apache.camel.component.servletlistener.CamelServletContextListener** を設定する必要があります。

279.8.2. オプション

org.apache.camel.component.servletlistener.CamelServletContextListener は、web.xml ファイルで context-param として設定できる次のオプションをサポートします。

オプション	タイプ	説明
propertyPlaceholder.XXX		Camel でプロパティープレースホルダーを設定するには。オプションの前に propertyPlaceholder .を付ける必要があります。たとえば、場所を設定するには、propertyPlaceholder.location を名前として使用します。プロパティー コンポーネントからすべてのオプションを設定できます。
jmx.XXX		JMX を設定する場合。オプションの前に jmx.を付ける必要があります。たとえば、JMX を無効にするには、jmx.disabled を名前として使用します。**org.apache.camel.spi.ManagementAgent** からすべてのオプションを設定できます。JMX ページに記載されているオプションも同様です。

オプション	タイプ	説明
name	**String**	CamelContext の名前を設定します。
messageHistory	**Boolean**	**Camel 2.12.2:** メッセージ履歴を有効にするか無効にするか (デフォルトで有効)。
streamCache	**Boolean**	ストリームキャッシュを有効にするかどうか。
trace	**Boolean**	トレーサーを有効にするかどうか。
delayer	**Long**	Delay Interceptor の遅延値を設定します。
handleFault	**Boolean**	ハンドルフォルトを有効にするかどうか。
errorHandlerRef	**String**	使用するコンテキストスコープのエラーハンドラーを参照します。
autoStartup	**Boolean**	Camel の起動時にすべてのルートを開始するかどうか。
useMDCLogging	**Boolean**	MDC ロギングを使用するかどうか。
useBreadcrumb	**Boolean**	breadcrumb を使用するかどうか。
managementNamePattern	**String**	JMX MBean のカスタム命名パターンを設定します。
threadNamePattern	**String**	スレッドのカスタム命名パターンを設定するには。
properties.XXX		**CamelContext.getProperties** でカスタムプロパティーを設定します。これはめったに使用されません。
routebuilder.XXX		使用するルートを設定します。詳細は、こちらを参照してください。

オプション	タイプ	説明
CamelContextLifecycle		**org.apache.camel.component.servletlistener.CamelContextLifecycle** の実装の FQN クラス名を参照します。これにより、CamelContext が開始または停止される前後にカスタムコードを実行できます。詳細については、以下を参照してください。
XXX		CamelContext に任意のオプションを設定します。

279.8.3. 例

Servlet Tomcat No Spring の例 を参照してください。

279.8.4. 作成した CamelContext へのアクセス

Camel 2.14/2.13.3/2.12.5 以降で利用可能

作成された **CamelContext** は、キー CamelContext を持つ属性として **ServletContext** に格納されます。以下に示すように、**ServletContext** を取得できる場合は、CamelContext を取得できます。

```
ServletContext sc = ...
CamelContext camel = (CamelContext) sc.getAttribute("CamelContext");
```

279.8.5. ルートの作成

web.xml ファイルで使用するルートを設定する必要があります。これはさまざまな方法で行うことができますが、すべてのパラメーターの前に routeBuilder を付ける必要があります。

279.8.5.1. RouteBuilder クラスの使用

以下に示すように、デフォルトでは、Camel は param-value が Camel RouteBuilder クラスの FQN クラス名であると想定します。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <param-value>org.apache.camel.component.servletlistener.MyRoute</param-value>
</context-param>
```

以下に示すように、同じ param-value で複数のクラスを指定できます。

```
<context-param>
  <param-name>routeBuilder-routes</param-name>
  <!-- we can define multiple values separated by comma -->
  <param-value>
    org.apache.camel.component.servletlistener.MyRoute,
    org.apache.camel.component.servletlistener.routes.BarRouteBuilder
  </param-value>
</context-param>
```

パラメーターの名前は、実行時には意味がありません。一意で、routeBuilder で始まる必要があります。上記の例では、routeBuilder-routes があります。しかし、routeBuilder.foo という名前にすることもできます。

279.8.5.2. パッケージスキャンの使用

また、Camel にパッケージスキャンを使用するように指示することもできます。つまり、指定されたパッケージで RouteBuilder タイプのすべてのクラスを検索し、それらを Camel ルートとして自動的に追加します。これを行うには、以下に示すように、値の前に packagescan: を付ける必要があります。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <!-- define the routes using package scanning by prefixing with packagescan: -->
  <param-value>packagescan:org.apache.camel.component.servletlistener.routes</param-value>
</context-param>
```

279.8.5.3. XML ファイルの使用

XML DSL を使用して Camel ルートを定義することもできますが、Spring または Blueprint を使用していないため、XML ファイルには Camel ルートのみを含めることができます。

web.xml では、以下に示すように、classpath、file、または httpURL から取得できる XML ファイルを参照します。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <param-value>classpath:routes/myRoutes.xml</param-value>
</context-param>
```

XML ファイルは次のとおりです。

routes/myRoutes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- the xmlns="http://camel.apache.org/schema/spring" is needed -->
<routes xmlns="http://camel.apache.org/schema/spring">

  <route id="foo">
    <from uri="direct:foo"/>
    <to uri="mock:foo"/>
  </route>

  <route id="bar">
    <from uri="direct:bar"/>
    <to uri="mock:bar"/>
  </route>

</routes>
```

XML ファイルでは、ルートタグは <routes> であり、名前空間 "http://camel.apache.org/schema/spring" を使用する必要があることに注意してください。この名前空間の名前には Spring が含まれていますが、これは歴史的な理由によるものです。Spring は当時の最初で唯一の XML DSL でした。実行時に Spring JAR は必要ありません。おそらく Camel 3.0 では、名前空間を一般的な名前に変更できます。

279.8.5.4. 適切なプレースホルダーの設定

クラスパスから **myproperties.properties** をロードするプロパティープレースホルダーを設定するための web.xml 設定のスニペットを次に示します。

```
<!-- setup property placeholder to load properties from classpath -->
<!-- we do this by setting the param-name with propertyPlaceholder. as prefix and then any options such as location, cache etc -->
<context-param>
  <param-name>propertyPlaceholder.location</param-name>
  <param-value>classpath:myproperties.properties</param-value>
</context-param>
<!-- for example to disable cache on properties component, you do -->
<context-param>
  <param-name>propertyPlaceholder.cache</param-name>
  <param-value>false</param-value>
</context-param>
```

279.8.5.5. JMX の設定

JMX の無効化など、JMX を設定するための web.xml 設定のスニペットを次に示します。

```
<!-- configure JMX by using names that is prefixed with jmx. -->
<!-- in this example we disable JMX -->
<context-param>
  <param-name>jmx.disabled</param-name>
  <param-value>true</param-value>
</context-param>
```

JNDI または Camel Registry のようなシンプルな ^^^^^^^^^^^^^^^^^^

このコンポーネントは、JNDI または Simple をレジストリーとして使用します。
これにより、JNDI で Bean やその他のサービスを検索したり、独自の Bean をバインドおよびバインド解除したりできます。

これは、**org.apache.camel.component.servletlistener.CamelContextLifecycle** を実装することにより、Java コードから実行されます。

279.8.5.6. カスタム CamelContextLifecycle の使用

以下のコードでは、コールバック **beforeStart** および **afterStop** を使用して、カスタム Bean を Simple Registry に登録し、停止時にクリーンアップします。

次に、以下に示すように、パラメーター名 CamelContextLifecycle を使用して、このクラスを web.xml ファイルに登録する必要があります。値
は、**org.apache.camel.component.servletlistener.CamelContextLifecycle** インターフェイスを実装するクラスを参照する FQN でなければなりません。

```
<context-param>
  <param-name>CamelContextLifecycle</param-name>
  <param-value>org.apache.camel.component.servletlistener.MyLifecycle</param-value>
</context-param>
```

myBean という名前を使用して HelloBean Bean を登録したので、以下に示すように、Camel ルートでこの Bean を参照できます。

```
public class MyBeanRoute extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("seda:foo").routeId("foo")
            .to("bean:myBean")
            .to("mock:foo");
    }
}
```

重要: org.apache.camel.component.servletlistener.JndiCamelServletContextListener を使用する場合、**CamelContextLifecycle** は **JndiRegistry** も使用する必要があります。同様に、サーブレットが **org.apache.camel.component.servletlistener.SimpleCamelServletContextListener** の場合、**CamelContextLifecycle** は **SimpleRegistry** を使用する必要があります

279.8.6. 関連項目

- SERVLET
- サーブレット Tomcat の例
- サーブレット Tomcat No Spring の例

第280章 SFTP コンポーネント

Camel バージョン 1.1 以降で利用可能

このコンポーネントは、FTP および SFTP プロトコルを介したリモートファイルシステムへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細については、FTP コンポーネント を参照してください。

280.1. URI オプション

以下のオプションは、FTPS コンポーネント専用です。

SFTP コンポーネントにはオプションがありません。

SFTP エンドポイントは、URI 構文を使用して設定されます。

```
sftp:host:port/directoryName
```

パスおよびクエリーパラメーターを使用します。

280.1.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** FTP サーバーのホスト名		String
port	FTP サーバーのポート		int
directoryName	開始ディレクトリー		文字列

280.1.2. クエリーパラメーター (111 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
charset (Common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。コンシューマーでこれを使用して、ファイルのエンコーディングを指定できます。これにより、Camel は、ファイルコンテンツがアクセスされている場合にファイルコンテンツをロードする必要がある charset を知ることができます。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む charset を指定できます。ファイルを書き込むとき、Camel はメッセージの内容をメモリーに読み込んで、データを設定された charset に変換できるようにする必要があることに注意してください。つまり、メッセージが大きい場合は、これを使用しないでください。		String
disconnect (Common)	使用直後にリモート FTP サーバーから切断するかどうか。切断は、FTP サーバーへの現在の接続のみを切断します。停止したいコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
doneFileName (Common)	Producer: 指定された場合、元のファイルが書き込まれると、Camel は 2 番目の完了ファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的プレースホルダーを使用することもできます。完了ファイルは、常に元のファイルと同じフォルダーに書き込まれます。Consumer: 指定すると、Camel は完了ファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定の名前を指定できます。または、動的なプレースホルダーを使用できます。完了ファイルは、常に元のファイルと同じフォルダーにあると想定されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		String

名前	説明	デフォルト	タイプ
fileName (Common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込むファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。(注: ヘッダー自体を式にすることもできます)。式オプションは String タイプと Expression タイプの両方をサポートします。式が String タイプである場合、これは常にファイル言語を使用して評価されます。式が Expression タイプである場合、指定された Expression タイプが使用されます。これにより、たとえば OGNL 式を使用できます。コンシューマーの場合、これを使用してファイル名をフィルターリングできるため、たとえば、ファイル言語構文 mydata-$date:now:yyyyMMdd.txt を使用して今日のファイルを消費できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は一度だけ使用されるヘッダーであり、CamelFileName を一時的に保存して後で復元する必要がなくなるため、簡単になります。		String
jschLoggingLevel (common)	JSCH アクティビティーログに使用するログレベル。JSCH はデフォルトで INFO レベルで冗長であるため、しきい値はデフォルトで WARN です。	WARN	LoggingLevel
separator (common)	使用するパス区切りを設定します。UNIX = UNIX スタイルのパス区切りを使用 Windows = Windows スタイルのパス区切りを使用 Auto = (デフォルト) ファイル名に既存のパス区切りを使用します	UNIX	PathSeparator
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用して、ファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示することをサポートしていない可能性があるため、オプションが false の場合、camel-ftp は古い方法を使用してディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。このオプションは、readLock=changed にも影響を与え、ファイル情報を更新するための高速チェックを実行するかどうかを制御します。これは、FTP サーバーに多くのファイルがある場合にプロセスを高速化するために使用できます。	false	boolean

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delete (consumer)	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて move failure 式を設定します。たとえば、ファイルを .error サブディレクトリーに移動するには、.error を使用します。注: ファイルを失敗したロケーションに移動すると、Camel はエラーを処理し、ファイルを再度取得しません。		String
noop (consumer)	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel は idempotent=true も設定し、同じファイルを繰り返し消費しないようにします。	false	boolean
preMove (consumer)	処理前に移動する場合にファイル名を動的に設定するために使用される式 (File 言語など)。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		String
preSort (consumer)	pre-sort が有効になっている場合、コンシューマーはポーリング中に、ファイルシステムから取得されたファイル名とディレクトリー名を並べ替えます。ソートされた順序でファイルを操作する必要がある場合に、これを行うことができます。pre-sort は、コンシューマーがフィルターリングを開始する前に実行され、Camel によって処理されるファイルを受け入れます。このオプション default=false で無効になっています。	false	boolean
recursive (consumer)	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean

名前	説明	デフォルト	タイプ
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streamDownload (consumer)	ローカル作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取られるときにルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーにロードされます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist に似ていますが、これは再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードする必要があるかどうか。このオプションが false に設定されている場合、メッセージ本文は null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルに関する詳細を含む Camel Exchange を引き続きトリガーします。ファイルがダウンロードされないだけです。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
ignoreFileNotFoundOrPermissionError (consumer)	(ディレクトリー内のファイルを一覧表示しようとするとき、またはファイルをダウンロードするとき)、存在しない場合、またはアクセス許可エラーが原因である場合に無視するかどうか。デフォルトでは、ディレクトリーまたはファイルが存在しないか、権限が不十分な場合、例外が出力されます。このオプションを true に設定すると、代わりにそれを無視できます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。in-progress リポジトリーは、現在進行中のファイルが消費されていることを示すために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>

名前	説明	デフォルト	タイプ
localWorkDirectory (consumer)	使用する場合、ローカルの作業ディレクトリーを使用して、リモートファイルのコンテンツをローカルファイルに直接保存し、コンテンツがメモリーに読み込まれないようにできます。これは、非常に大きなリモートファイルを使用している場合に、メモリーを節約するために役立ちます。		String
onCompletionException Handler (consumer)	カスタム org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを実行する完了プロセスのファイル中に出力される例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視します。		ExceptionHandler
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。特別な準備完了ファイルが存在するなど、ファイルを使用する前に特別な条件を満たす必要がある場合にも使用できます。このオプションを設定すると、readLock オプションは適用されません。		GenericFileProcess s Strategy<T>
startingDirectoryMustExist (consumer)	開始ディレクトリーの存在が必要かどうか。autoCreate オプションがデフォルトで有効になっていることに注意してください。これは、開始ディレクトリーが存在しない場合、通常は自動作成されることを意味します。autoCreate を無効にして有効にすると、開始ディレクトリーの存在が必要なことを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean

名前	説明	デフォルト	タイプ
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。場合によっては、特定のファイルをダウンロードする必要があり、LIST コマンドの使用が許可されていない場合があるため、このオプションを false に設定できます。このオプションを使用する場合、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、権限などのメタデータ情報が含まれないことに注意してください。これらの情報は、LIST コマンドを使用している場合にのみ取得できるためです。	true	boolean
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。Override: これがデフォルトで、既存のファイルを置き換えます。append: 既存ファイルにコンテンツを追加します。fail: GenericFileOperationException を出力し、既存ファイルがあることを示します。ignore: 問題を警告なしで無視して既存のファイルは上書きしませんが、問題は発生していないと想定します。move: オプションを設定するには、moveExisting オプションも使用する必要があります。オプション eagerDeleteTargetFile を使用して、ファイルを移動する際に既存ファイルが存在する場合に何をすべきか制御でき、そうでない場合は移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用できます。これにより、存在チェックを実行せずに、一時的なファイル名から実際のファイル名への変更を試みることができます。このチェックは、一部のファイルシステム、特に FTP サーバーでは高速になる場合があります。	オーバーライド	GenericFileExist
flatten (producer)	flatten は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に使用できますが、たとえばファイルを別のディレクトリーに書き込む場合、ファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が先頭パスから削除されます。	false	boolean

名前	説明	デフォルト	タイプ
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式 (File 言語など)。ファイルをバックアップサブディレクトリーに移動するには、backup と入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントでは file:parent がサポートされていないことに注意してください。FTP コンポーネントは、既存のファイルを現在のディレクトリーをベースとする相対ディレクトリーにしか移動できないためです。		String
tempFileName (producer)	tempPrefix オプションと同じですが、ファイル言語を使用するため、一時ファイル名の命名をより細かく制御できます。		String
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込み、書き込みが完了した後に、その名前を実際の名前に変更するために使用されます。書き込み中のファイルを識別し、(排他的読み取りロックを使用せずに) コンシューマーが進行中のファイルを読み取らないようにするために使用できます。大きなファイルをアップロードするときに FTP でよく使用されます。		String
allowNullBody (producer)	ファイルの書き込み中に null の本文を許可するかどうかを指定するために使用されます。true に設定すると空のファイルが作成され、false に設定して null の本文をファイルコンポーネントに送信しようとすると、Cannot write null body to file.という GenericFileWriteException が出力されます。fileExist オプションを Override に設定するとファイルは切り捨てられ、append に設定するとファイルは変更されません。	false	boolean
chmod (producer)	保存されたファイルに chmod を設定できます。たとえば、chmod=640 です。		String
disconnectOnBatchComplete (producer)	バッチアップロードが完了した直後にリモート FTP サーバーから切断するかどうか。disconnectOnBatchComplete は、FTP サーバーへの現在の接続のみを切断します。	false	boolean

名前	説明	デフォルト	タイプ
eagerDeleteTargetFile (producer)	既存のターゲットファイルを先行して削除するかどうか。このオプションは、fileExists=Override および tempFileName オプションを使用している場合にのみ適用されます。これを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除することを無効化 (false に設定) できます。たとえば、大きなファイルを書き込んで、一時ファイルの書き込み中にターゲットファイルを存在させたい場合があります。これにより、一時ファイルの名前がターゲットファイル名に変更される直前まで、ターゲットファイルは削除されません。このオプションは、fileExist=Move が有効で、既存のファイルが存在する場合に、既存のファイルを削除するかどうかを制御するためにも使用されます。このオプション copyAndDeleteOnRenameFails が false の場合、既存のファイルが存在する場合は例外が出力されます。true の場合、移動操作の前に既存のファイルが削除されます。	true	boolean
keepLastModified (producer)	ソースファイル (存在する場合) からの最終変更のタイムスタンプを保持します。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには、java.util.Date またはタイムスタンプ付きの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプが設定されます。注記: このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	ファイルを FTP サーバーにアップロードする前に書き込み前チェックとして noop コマンドを送信するかどうか。接続の検証がまだ有効であるため、これはデフォルトで有効になっています。これにより、サイレントに再接続してファイルをアップロードできるようになります。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
autoCreate (advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	書き込みバッファーのサイズ (バイト単位)。	131072	int

名前	説明	デフォルト	タイプ
bulkRequests (advanced)	一度に未処理のリクエストの数を指定します。この値を大きくすると、ファイル転送速度がわずかに向上する可能性がありますが、メモリー使用量が増加します。		Integer
compression (advanced)	圧縮を使用する場合。1~10 のレベルを指定します。重要: 圧縮をサポートするには、必要な JSCH zlib JAR を手動でクラスパスに追加する必要があります。		int
connectTimeout (advanced)	接続が確立されるのを待つための接続タイムアウトを設定します。FTPClient と JSCH の両方で使用されます	10000	int
maximumReconnectAttempts (advanced)	Camel がリモート FTP サーバーへの接続を試行するときに実行する再接続の最大試行回数を指定します。この動作を無効にするには、0 を使用します。		int
proxy (advanced)	カスタム設定の com.jcraft.jsch.Proxy を使用する場合。このプロキシーは、ターゲット SFTP ホストからのメッセージを消費/送信するために使用されます。		Proxy
reconnectDelay (advanced)	ミリ秒単位の遅延 Camel は、再接続試行を実行する前に待機します。		long
serverAliveCountMax (advanced)	sftp セッションの serverAliveCountMax を設定できます。	1	int
serverAliveInterval (advanced)	sftp セッションの serverAliveInterval を設定できます。		int
soTimeout (advanced)	SO タイムアウトを設定します。FTPClient によってのみ使用されます。	300000	int
stepwise (advanced)	ファイルをダウンロードするとき、またはファイルをディレクトリーにアップロードするときに、ファイル構造をトラバースしながらディレクトリーを段階的に変更するかどうかを設定します。たとえば、セキュリティー上の理由で FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

名前	説明	デフォルト	タイプ
throwExceptionOnConnect Failed (advanced)	接続が失敗した (使い果たされた) 場合に例外を出力する必要があります。デフォルトでは、例外は出力されず、WARN がログに記録されます。これを使用して、例外の出力を有効にし、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから出力された例外を処理できます。	false	boolean
timeout (advanced)	応答を待つためのデータタイムアウトを設定します。FTPClient だけが使用します	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。コンマ区切り形式で複数の除外を指定できます。		String
antFilterCaseSensitive (filter)	ant フィルターに大文字と小文字を区別するフラグを設定します	true	boolean
antInclude (filter)	Ant スタイルフィルターの組み込み。コンマ区切り形式で複数の組み込みを指定できます。		String
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が eager かどうかを制御できます。eager の場合、ファイルのスキャン中に制限されます。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートしてからポーリングを制限できます。ソートのためにすべてのファイルの詳細がメモリー内にあるため、メモリー使用量が大きくなることに注意してください。	true	boolean
exclude (filter)	ファイル名が正規表現パターンに一致する場合にファイルを除外するために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
filter (filter)	org.apache.camel.component.file.GenericFileFilter クラスとしてのプラグ可能なフィルター。フィルターがその accept () メソッドで false を返す場合、ファイルをスキップします。		GenericFileFilter<T>

名前	説明	デフォルト	タイプ
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルターリングします。たとえば、現在の日付でフィルターリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		String
filterFile (filter)	Simple 言語に基づいてファイルをフィルターリングします。たとえば、ファイルサイズでフィルターリングするには、$file:size 5000 を使用できます。		String
idempotent (filter)	Camel が既に処理されたファイルをスキップできるように、Idempotent Consumer EIP パターンを使用するオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	Boolean
idempotentKey (filter)	カスタムのべき等性キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用できます。たとえば、ファイル名とファイルサイズを使用するには idempotentKey=$file:name-$file:size となります。		String
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、プラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository はデフォルトで MemoryMessageIdRepository を使用します。		String>
include (filter)	ファイル名が正規表現パターンに一致する場合にファイルを含めるために使用されます (照合では大文字と小文字を区別します)。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW() 構文を使用して設定する必要があります。詳細はエンドポイント URI の設定を参照してください。		String
maxDepth (filter)	ディレクトリーを再帰的に処理する際にトラバースする最大深度。	2147483647	int

名前	説明	デフォルト	タイプ
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されていません。たとえば制限を 1000 などに設定して、数千のファイルがあるサーバーの起動を回避できます。無効にするには、0 または負の値を設定します。注記: このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 個のファイルがある場合に maxMessagesPerPoll=500 を使用すると、最初の 500 個のファイルのみ選択され、ソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 はベースディレクトリーを意味します。minDepth=2 は最初のサブディレクトリーを意味します。		int
move (filter)	処理後に移動する場合にファイル名を動的に設定するために使用される式 (Simple 言語など)。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		String
exclusiveReadLockStrategy (lock)	org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装としてのプラグ可能な読み取りロック。		GenericFileExclusiveReadLockStrategy<T>

名前	説明	デフォルト	タイプ
readLock (lock)	ファイルに排他的な読み取りロックがある (つまり、ファイルが進行中または書き込み中ではない) 場合にのみファイルをポーリングするために、コンシューマーが使用します。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none: 読み取りロックは使用されていません。markerFile: Camel はマーカーファイル (fileName.camelLock) を作成してロックを保持します。このオプションは、FTP コンポーネント changed では使用できません。changed は、ファイルの長さ/変更のタイムスタンプを使用して、ファイルが現在コピーされているかどうかを検出します。この判断には 1 秒以上かかるため、このオプションは他のオプションほど速くファイルを消費できませんが、JDK IO API はファイルが別のプロセスで使用中かどうか判断できないため、信頼性が高くなります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。ファイルシステムが分散ファイルロックをサポートしていない限り、マウント/共有によりリモートファイルシステムにアクセスする場合、このアプローチは避ける必要があります。rename: 排他的な読み取りロックを取得できるかどうかのテストとしてファイル名の変更を試みるために使用します。idempotent: (ファイルコンポーネントのみ) 読み取りロックとして idempotentRepository を使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-changed: (ファイルコンポーネントのみ) idempotentRepository および changed を結合された read-lock として使用するためのものです。これにより、べき等性リポジトリーの実装がサポートする場合に、クラスターリングをサポートする読み取りロックを使用できます。idempotent-rename: (ファイルコンポーネントのみ) idempotentRepository および rename を結合された読み取りロックとして使用するためのものです。これにより、べき等性リポジトリーの実装がサポートしている場合、クラスターリングをサポートする読み取りロックを使用できます。注記: さまざまな読み取りロックは、異なるノード上の同時コンシューマーが共有ファイルシステム上の同じファイルを求めて競合するクラスターモードでの動作にすべて適しているわけではありません。アトミックに近い操作を使用して空のマーカーファイルを作成する markerFile ですが、クラスターでの動作は保証されていません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等性リ	none	String

名前	説明	デフォルト	タイプ
	ポジトリーが Hazelcast コンポーネントや Infinispan などのクラスターリングに対応している場合、べき等性等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合、読み取りロックの間隔 (ミリ単位)。この間隔は、読み取りロックを取得する試行間のスリープに使用されます。たとえば、changed 読み取りロックを使用する場合、遅い書き込みに対応するために間隔を長く設定できます。デフォルトは 1 秒ですが、プロデューサーによるファイルの書き込みが非常に遅い場合は短すぎる可能性があります。注記: FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	1000	long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされなかった場合 (JVM クラッシュなど)、マーカーファイルを使用した読み取りロックが、ファイルシステムに残っている可能性のある孤立した読み取りロックファイルを起動時に削除する必要があるかどうか。このオプションを false にすると、孤立したロックファイルがあると Camel はそのファイルを取得しようとしなくなります。これは、別のノードが同じ共有ディレクトリーから同時にファイルを読み取っているが原因である可能性もあります。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルを変更できます。たとえば、ログを記録しないように OFF に設定できます。このオプションを適用できる readLock タイプは、changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename のみです。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	changed、rename、exclusive の読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、他のプロセスが同じファイルを取得するのを防ぐために、マーカーファイルも使用されます。このオプションを false に設定すると、この動作をオフにできます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みたくない場合などです。	true	boolean

名前	説明	デフォルト	タイプ
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションは、読み取りロックを取得しようとする前に、ファイルが経過しなければならない最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルに 5 分以上の経過を要求します。これにより、指定された期間以上のファイルの取得を試みるため、changed 読み取りロックが高速化されます。	0	long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小ファイル長を設定できます。デフォルトで Camel はファイルにデータが含まれていると想定するため、デフォルト値は 1 です。このオプションをゼロに設定すると、長さがゼロのファイルを使用できます。	1	long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に成功し、コミットが行われるときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトはファイルは削除されないため、競合状態が発生せず、別のアクティブなノードがファイルを取得しようとする可能性があります。代わりにべき等性リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする場合があります。これにより、競合状態の問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションは、ファイル処理に失敗し、ロールバックが発生するときに、べき等性リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが (ファイルがコミットされたかのように) 確認されます。	true	boolean

名前	説明	デフォルト	タイプ
readLockTimeout (lock)	読み取りロックでサポートされている場合、読み取りロックのオプションのタイムアウト (ミリ秒単位)。読み取りロックを許可できず、タイムアウトがトリガーされた場合、Camel はファイルをスキップします。次のポーリングで、Camel はファイルを再試行します。このときに、読み取りロックが許可される可能性があります。無期限を指定するには、0 以下の値を使用します。現在、fileLock、changed、および rename がタイムアウトに対応しています。注記: FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくする必要がありますが、thumb のルールではタイムアウトは readLockCheckInterval の 2 倍以上にする必要があります。これは、タイムアウトに達する前に読み取りロックプロセスがロックを取得しようとするためのアンブル時間を確保するために必要です。	10000	long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long

名前	説明	デフォルト	タイプ
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルします (ランダムな順序でのソート)	false	boolean
sortBy (sort)	File 言語を使用したビルトインソート。ネストされたソートをサポートしているため、ファイル名でのソートと、2 つ目のグループとして変更日でソートできます。		String
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
ciphers (security)	優先順に使用される暗号のコンマ区切りリストを設定します。可能な暗号名は、JCraft JSCH によって定義されています。例としては、aes128-ctr、aes128-cbc、3des-ctr、3des-cbc、blowfish-cbc、aes192-cbc、aes256-cbc などがあります。指定しない場合、JSCH のデフォルトリストが使用されます。		String

名前	説明	デフォルト	タイプ
keyPair (security)	SFTP エンドポイントが公開鍵/秘密鍵の検証を実行できるように、公開鍵と秘密鍵の鍵ペアを設定します。		KeyPair
knownHosts (security)	SFTP エンドポイントがホストキーの検証を実行できるように、バイト配列から known_hosts を設定します。		byte[]
knownHostsFile (security)	SFTP エンドポイントがホストキーの検証を実行できるように、known_hosts ファイルを設定します。		String
knownHostsUri (security)	SFTP エンドポイントがホストキーの検証を実行できるように、known_hosts ファイル (デフォルトでクラスパスからロードされる) を設定します。		String
password (security)	ログインに使用するパスワード		String
preferredAuthentications (security)	SFTP エンドポイントが使用する優先認証を設定します。例として、パスワード、公開鍵などがあります。指定しない場合、JSCH のデフォルトリストが使用されます。		String
privateKey (security)	SFTP エンドポイントが秘密鍵の検証を行えるように、秘密鍵をバイトとして設定します。		byte[]
privateKeyFile (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルを設定します。		String
privateKeyPassphrase (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルのパスフレーズを設定します。		String
privateKeyUri (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイル (デフォルトでクラスパスからロードされる) を設定します。		String
strictHostKeyChecking (security)	厳密なホストキーチェックを使用するかどうかを設定します。	いいえ	String
username (security)	ログインに使用するユーザー名		String
useUserKnownHostsFile (security)	knownHostFile が明示的に設定されていない場合は、System.getProperty (user.home)/.ssh/known_hosts からホストファイルを使用します。	true	boolean

第281章 SHIRO SECURITY コンポーネント

Camel 2.5 で利用可能

Camel の **shiro-security** コンポーネントは、Apache Shiro セキュリティープロジェクトに基づくセキュリティー重視のコンポーネントです。

Apache Shiro は、認証、承認、エンタープライズセッション管理、および暗号化を適切に処理する、強力で柔軟なオープンソースセキュリティーフレームワークです。Apache Shiro プロジェクトの目的は、最も堅牢で包括的なアプリケーションセキュリティーフレームワークを提供すると同時に、非常に理解しやすく、非常に使いやすいものにすることです。

この camel shiro-security コンポーネントを使用すると、認証と承認のサポートを camel ルートのさまざまなセグメントに適用できます。

Shiro セキュリティーは、Camel ポリシーを使用してルートに適用されます。Camel のポリシーは、Camel プロセッサーにインターセプターを適用するための戦略パターンを利用します。camel ルートのセクション/セグメントに横断的な関心事 (たとえば、セキュリティー、トランザクションなど) を適用する機能を提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-shiro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

281.1. SHIRO セキュリティーの基本

camel ルートで Shiro セキュリティーを使用するには、ShiroSecurityPolicy オブジェクトをセキュリティー設定の詳細 (ユーザー、パスワード、ロールなどを含む) でインスタンス化する必要があります。次に、このオブジェクトを camel ルートに適用する必要があります。この ShiroSecurityPolicy オブジェクトは、Camel レジストリー (JNDI または ApplicationContextRegistry) に登録され、Camel コンテキストの他のルートで使用される場合もあります。

設定の詳細は、Ini ファイル (プロパティーファイル) または Ini オブジェクトを使用して ShiroSecurityPolicy に提供されます。Ini ファイルは、以下に示すように、ユーザー/ロールの詳細を含む標準の Shiro 設定ファイルです。

```
[users]
# user 'ringo' with password 'starr' and the 'sec-level1' role
ringo = starr, sec-level1
george = harrison, sec-level2
john = lennon, sec-level3
paul = mccartney, sec-level3

[roles]
# 'sec-level3' role has all permissions, indicated by the
# wildcard '*'
sec-level3 = *

# The 'sec-level2' role can do anything with access of permission
# readonly (*) to help
```

```
sec-level2 = zone1:*

# The 'sec-level1' role can do anything with access of permission
# readonly
sec-level1 = zone1:readonly:*
```

281.2. SHIROSECURITYPOLICY オブジェクトのインスタンス化

ShiroSecurityPolicy オブジェクトは次のようにインスタンス化されます

```
    private final String iniResourcePath = "classpath:shiro.ini";
    private final byte[] passPhrase = {
        (byte) 0x08, (byte) 0x09, (byte) 0x0A, (byte) 0x0B,
        (byte) 0x0C, (byte) 0x0D, (byte) 0x0E, (byte) 0x0F,
        (byte) 0x10, (byte) 0x11, (byte) 0x12, (byte) 0x13,
        (byte) 0x14, (byte) 0x15, (byte) 0x16, (byte) 0x17};
    List<permission> permissionsList = new ArrayList<permission>();
    Permission permission = new WildcardPermission("zone1:readwrite:*");
    permissionsList.add(permission);

    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy(iniResourcePath, passPhrase, true, permissionsList);
```

281.3. SHIROSECURITYPOLICY OPTIONS

名前	デフォルト値	タイプ	説明
iniResourcePath or ini	**none**	リソース文字列または Ini オブジェクト	iniResourcePath の必須のリソース文字列または Ini オブジェクトのインスタンスをセキュリティーポリシーに渡す必要があります。リソースは、file:、classpath:、または url: という接頭辞をそれぞれ付けた場合、ファイルシステム、クラスパス、または URL から取得できます。例: "classpath:shiro.ini"
passPhrase	**AES 128 ベースのキー**	byte[]	メッセージエクスチェンジとともに送信された ShiroSecurityToken を復号化するパスフレーズ
alwaysReauthenticate	**true**	boolean	個々のリクエストごとに再認証を確実にするための設定。false に設定すると、ユーザーは認証されてロックされ、同じユーザーからのリクエストのみが認証されるようになります。

名前	デフォルト値	タイプ	説明
permissionsList	**none**	List<Permission>	認証されたユーザーがさらにアクションを実行することを許可されるために必要な許可のリスト。つまり、ルートをさらに続行します。ShiroSecurityPolicy オブジェクトに Permissions リストまたは Roles List (以下を参照) が提供されていない場合、承認は不要であると見なされます。デフォルトでは、リスト内のいずれかの権限オブジェクトが適用される場合に承認が付与されることに注意してください。
rolesList	**none**	List<String>	**Camel 2.13:** 認証されたユーザーがさらにアクションを実行することを承認されるために必要なロールのリスト。つまり、ルートをさらに続行します。ShiroSecurityPolicy オブジェクトにロールリストまたは権限リスト (上記を参照) が提供されていない場合、承認は不要と見なされます。デフォルトでは、リスト内のいずれかのロールが適用可能な場合に許可が付与されることに注意してください。
cipherService	**AES**	org.apache.shiro.crypto.CipherService	Shiro には、AES および Blowfish ベースの CipherServices が付属しています。これらのいずれかを使用するか、独自の Cipher 実装に渡すことができます
base64	**false**	**boolean**	**Camel 2.12:** セキュリティートークンヘッダーに base64 エンコードを使用します。これにより、JMS などを介してヘッダーを転送できます。このオプションは、**ShiroSecurityTokenInjector** でも設定する必要があります。
allPermissionsRequired	**false**	**boolean**	**Camel 2.13:** デフォルトでは、permissionsList パラメーターのいずれかのアクセス許可オブジェクトが適用可能である場合に承認が付与されます。すべてのアクセス許可を満たす必要がある場合は、これを true に設定します。
allRolesRequired	**false**	**boolean**	**Camel 2.13:** デフォルトでは、rolesList パラメーターのいずれかのロールが適用可能な場合に承認が付与されます。すべてのロールを満たす必要がある場合は、これを true に設定します。

281.4. CAMEL ROUTE での SHIRO 認証の適用

ShiroSecurityPolicy は、メッセージヘッダーに暗号化された SecurityToken を含む入力メッセージエクスチェンジをテストし、許可して、適切な認証に続いて続行します。SecurityToken オブジェクトには、ユーザーが有効なユーザーであるかどうかを判別するために使用されるユーザー名/パスワードの詳細が含まれています。

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("classpath:shiro.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

281.5. CAMEL ROUTE での SHIRO 承認の適用

アクセス許可リストを ShiroSecurityPolicy に関連付けることで、camel ルートに承認を適用できます。権限リストは、ユーザーがルートセグメントの実行を続行するために必要な権限を指定します。ユーザーが適切な権限セットを持っていない場合、リクエストはそれ以上続行できません。

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("./src/test/resources/securityconfig.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

281.6. SHIROSECURITYTOKEN を作成して MESSAGE EXCHANGE に挿入する

ShiroSecurityToken オブジェクトは、ShiroSecurityTokenInjector と呼ばれる Shiro Processor を使用して作成し、Message Exchange に挿入できます。クライアントで ShiroSecurityTokenInjector を使用して ShiroSecurityToken を注入する例を以下に示します。

```
ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");
ShiroSecurityTokenInjector shiroSecurityTokenInjector =
    new ShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

from("direct:client").
    process(shiroSecurityTokenInjector).
    to("direct:secureEndpoint");
```

281.7. SHIROSECURITYPOLICY で保護されたルートにメッセージを送信する

セキュリティーポリシーが適用される camel ルートで送信されるメッセージとメッセージエクスチェンジには、Exchange ヘッダーに SecurityToken を含める必要があります。SecurityToken は、ユーザー名とパスワードを保持する暗号化されたオブジェクトです。SecurityToken はデフォルトで AES 128 ビットセキュリティーを使用して暗号化されており、任意の暗号に変更できます。

以下は、Camel の ProducerTemplate と SecurityToken を使用してリクエストを送信する方法の例です。

```
@Test
public void testSuccessfulShiroAuthenticationWithNoAuthorization() throws Exception {
    //Incorrect password
    ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "stirr");

    // TestShiroSecurityTokenInjector extends ShiroSecurityTokenInjector
    TestShiroSecurityTokenInjector shiroSecurityTokenInjector =
        new TestShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

    successEndpoint.expectedMessageCount(1);
    failureEndpoint.expectedMessageCount(0);

    template.send("direct:secureEndpoint", shiroSecurityTokenInjector);

    successEndpoint.assertIsSatisfied();
    failureEndpoint.assertIsSatisfied();
}
```

281.8. SHIROSECURITYPOLICY によって保護されたルートへのメッセージの送信 (CAMEL 2.12 以降ではるかに簡単です)

Camel 2.12 以降では、サブジェクトを 2 つの異なる方法で提供できるため、さらに簡単になります。

281.8.1. ShiroSecurityToken の使用

ユーザー名とパスワードを含むタイプ

org.apache.camel.component.shiro.security.ShiroSecurityToken のキー **ShiroSecurityConstants.SHIRO_SECURITY_TOKEN** のヘッダーを持つメッセージを Camel ルートに送信できます。以下に例を示します。

```
        ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");

        template.sendBodyAndHeader("direct:secureEndpoint", "Beatle Mania",
    ShiroSecurityConstants.SHIRO_SECURITY_TOKEN, shiroSecurityToken);
```

以下に示すように、2 つの異なるヘッダーでユーザー名とパスワードを指定することもできます。

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put(ShiroSecurityConstants.SHIRO_SECURITY_USERNAME, "ringo");
        headers.put(ShiroSecurityConstants.SHIRO_SECURITY_PASSWORD, "starr");
        template.sendBodyAndHeaders("direct:secureEndpoint", "Beatle Mania", headers);
```

ユーザー名とパスワードのヘッダーを使用すると、Camel ルートの ShiroSecurityPolicy はそれらをキー ShiroSecurityConstants.SHIRO_SECURITY_TOKEN とトークンを持つ単一のヘッダーに自動的に変換します。次に、token は **ShiroSecurityToken** インスタンス、または文字列としての base64 表現です (後者は、base64=true を設定した場合です)。

第282章 SIMPLE 言語

Camel バージョン 1.1 以降で利用可能

Simple Expression Language は、作成されたときは非常に単純な言語でしたが、その後、より強力になりました。これは主に、新しい依存関係や XPath の知識を必要とせずに、式と述語を評価するための非常に小さくて単純な言語であることを目的としています。そのため、camel-core でのテストに最適です。アイデアは、Camel ルートで式ベースのスクリプトが少し必要な場合に、一般的なユースケースの 95% をカバーすることでした。

ただし、より複雑なユースケースでは、一般に、次のようなより表現力のある強力な言語を選択することをお勧めします。

- SpEL
- Mvel
- Groovy
- JavaScript
- OGNL
- サポートされている スクリプト言語 の 1 つ

単純な言語は、式に定数リテラルが含まれる複雑な式に **${body}** プレースホルダーを使用します。式がトークン自体のみの場合、$\{ } プレースホルダーは省略できます。

ヒント

代替構文 Camel 2.5 以降では、プレースホルダーとして **$simple{ }** を使用する代替構文を使用することもできます。これは、たとえば Spring プロパティーのプレースホルダーを Camel と一緒に使用する場合の競合を回避するために使用できます。

282.1. CAMEL 2.9 以降の SIMPLE 言語の変更

Simple 言語は Camel 2.9 以降で改善され、正確なエラーメッセージのインデックスを作成できる、より優れた構文パーサーを使用するようになりました。たとえば、演算子の 1 つに入力ミスがあると、以前はパーサーがこれを検出できず、評価が true になりました。下位互換性がなくなった構文にいくつかの変更があります。Simple 言語を述語として使用する場合、リテラルテキストを一重引用符または二重引用符で囲む **必要があります**。例: **"${body} == 'Camel'"** .リテラルを一重引用符で囲んでいることに注意してください。**"body"** と **"header.foo"** を使用してメッセージの本文とヘッダーを参照する古いスタイルは @deprecated であり、組み込み関数には常に $\{ } トークンを使用することをお勧めします。範囲演算子では、**${header.zip} between '30000..39999'** のように、範囲を一重引用符で囲む必要があります。

in メッセージの本文を取得するには: **"body"**、または **"in.body"** または **"${body}"**。

複雑な式では、**"Hello ${in.header.name} how are you?"** のように $\{ } プレースホルダーを使用する必要があります。.

同じ式に複数の関数を含めることができます: **"Hello ${in.header.name} this is ${in.header.me} speaking"** .
ただし、Camel 2.8.x 以前では関数をネストでき **ません** (つまり、既存のプレースホルダーに別の $\{ } プレースホルダーを含めることはできません)。
Camel 2.9 以降では、関数をネストできます。

282.2. SIMPLE 言語オプション

Simple 言語は、以下に示す 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
resultType		**String**	結果の型 (出力からの型) のクラス名を設定します
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

282.3. VARIABLES

変数	タイプ	説明
camelId	String	**Camel 2.10:** CamelContext 名
camelContext.**OGNL**	Object	**Camel 2.11:** Camel OGNL 式を使用して呼び出された CamelContext。
exchange	Exchange	**Camel 2.16:** エクスチェンジ
exchange.**OGNL**	Object	**Camel 2.16:** Camel OGNL 式を使用して呼び出された Exchange。
exchangeId	String	**Camel 2.3:** エクスチェンジ ID
id	String	入力メッセージ ID
body	Object	入力ボディー
in.body	Object	入力ボディー
body.**OGNL**	Object	**Camel 2.3:** Camel OGNL 式を使用して呼び出される入力本文
in.body.**OGNL**	Object	**Camel 2.3:** Camel OGNL 式を使用して呼び出される入力本文
bodyAs(**type**)	タイプ	**Camel 2.3:** クラス名で決定される特定の型にボディーを変換します。変換された本文は null にすることができます。

変数	タイプ	説明
bodyAs(**type**).**OGNL**	Object	**Camel 2.18:** クラス名によって決定される特定のタイプにボディーを変換し、Camel OGNL 式を使用してメソッドを呼び出します。変換された本文は null にすることができます。
mandatoryBodyAs(**type**)	タイプ	**Camel 2.5:** クラス名によって決定される特定の型にボディーを変換し、ボディーには null を指定できません。
mandatoryBodyAs(**type**).**OGNL**	Object	**Camel 2.18:** クラス名によって決定される特定のタイプにボディーを変換し、Camel OGNL 式を使用してメソッドを呼び出します。
out.body	Object	出力ボディー
header.foo	Object	入力 foo ヘッダーを参照します
header[foo]	Object	**Camel 2.9.2:** 入力 foo ヘッダーを参照します
headers.foo	Object	入力 foo ヘッダーを参照します
headers[foo]	Object	**Camel 2.9.2:** 入力 foo ヘッダーを参照します
in.header.foo	Object	入力 foo ヘッダーを参照します
in.header[foo]	Object	**Camel 2.9.2:** 入力 foo ヘッダーを参照します
in.headers.foo	Object	入力 foo ヘッダーを参照します
in.headers[foo]	Object	**Camel 2.9.2:** 入力 foo ヘッダーを参照します
header.foo[bar]	Object	**Camel 2.3:** 入力された foo ヘッダーをマップと見なし、bar をキーとしてマップ上でルックアップを実行します

変数	タイプ	説明
in.header.foo[bar]	Object	**Camel 2.3:** 入力された foo ヘッダーをマップと見なし、bar をキーとしてマップ上でルックアップを実行します
in.headers.foo[bar]	Object	**Camel 2.3:** 入力された foo ヘッダーをマップと見なし、bar をキーとしてマップ上でルックアップを実行します
header.foo.**OGNL**	Object	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します
in.header.foo.**OGNL**	Object	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します
in.headers.foo.**OGNL**	Object	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します
out.header.foo	Object	out ヘッダ foo を参照します
out.header[foo]	Object	**Camel 2.9.2:** out ヘッダー foo を参照します
out.headers.foo	Object	out ヘッダ foo を参照します
out.headers[foo]	Object	**Camel 2.9.2:** out ヘッダー foo を参照します
headerAs(**key**, **type**)	タイプ	**Camel 2.5:** クラス名によって決定される、指定のタイプにヘッダーを変換します
ヘッダー	Map	**Camel 2.9:** 入力ヘッダーを参照します
in.headers	Map	**Camel 2.9:** 入力ヘッダーを参照します

変数	タイプ	説明
property.foo	Object	**非推奨:** エクスチェンジの foo プロパティーを参照します
exchangeProperty.foo	Object	**Camel 2.15:** エクスチェンジの foo プロパティーを参照します
property[foo]	Object	**非推奨:** エクスチェンジの foo プロパティーを参照します
exchangeProperty[foo]	Object	**Camel 2.15:** エクスチェンジの foo プロパティーを参照します
property.foo.**OGNL**	Object	**非推奨:** エクスチェンジの foo プロパティーを参照し、Camel OGNL 式を使用してその値を呼び出します。
exchangeProperty.foo.**OGNL**	Object	**Camel 2.15:** エクスチェンジの foo プロパティーを参照し、Camel OGNL 式を使用してその値を呼び出します。
sys.foo	String	システムプロパティーを参照します
sysenv.foo	String	**Camel 2.3:** システム環境を参照します
exception	Object	**Camel 2.4:** エクスチェンジの例外オブジェクトを参照します。クスチェンジに例外が設定されていない場合は **null** です。Exchange に例外がある場合は、フォールバックしてキャッチされた例外 (**Exchange.EXCEPTION_CAUGHT**) を取得します。
exception.**OGNL**	Object	**Camel 2.4:** Camel OGNL 式オブジェクトを使用して呼び出された exchange 例外を参照します。
exception.message	String	エクスチェンジの exception.message を参照してください。エクスチェンジに例外が設定されていない場合は **null** です。Exchange に例外がある場合は、フォールバックしてキャッチされた例外 (**Exchange.EXCEPTION_CAUGHT**) を取得します。
exception.stacktrace	String	**Camel 2.6.**エクスチェンジの exception.stacktrace を参照してください。エクスチェンジに例外が設定されていない場合は **null** です。Exchange に例外がある場合は、フォールバックしてキャッチされた例外 (**Exchange.EXCEPTION_CAUGHT**) を取得します。

変数	タイプ	説明
date:_command_	Date	Date オブジェクトに対して評価されます。サポートされているコマンドは以下の通りです: **now** で現在のタイムスタンプ、**in.header.xxx** または **header.xxx** で IN ヘッダーの Date オブジェクトをキー xxx で使用します。**out.header.xxx** を使用して、OUT ヘッダーでキー xxx を持つ Date オブジェクトを使用します。**property.xxx** を使用して、キー xxx を持つ exchange プロパティーの Date オブジェクトを使用します。ファイルの最終変更タイムスタンプの **file** (ファイルコンシューマーで利用可能)。コマンドは、**now-24h** または **in.header.xxx+1h** または **now+1h30m-100** などを使用できます。
date:_command:pattern_	String	**java.text.SimpleDataFormat** パターンを使用した日付の書式設定。
date-with-timezone:_command:timezone:pattern_	String	**java.text.SimpleDataFormat** タイムゾーンとパターンを使用した日付の書式設定。
bean:_bean expression_	Object	Bean 言語を使用して Bean 式を呼び出します。メソッド名を指定するには、区切り文字としてドットを使用する必要があります。Bean コンポーネントで使用される ?method=methodname 構文もサポートしています。
properties:_locations:key_	String	**非推奨 (代わりに properties-location を使用) Camel 2.3:** 指定されたキーでプロパティーを検索します。**locations** はオプションです。詳しくは、PropertyPlaceholder の使用を参照してください。
properties-location:_http://locationskey[locations:key]_	String	**Camel 2.14.1:** 指定されたキーでプロパティーを検索します。**locations** はオプションです。詳しくは、PropertyPlaceholder の使用を参照してください。
properties:key:default	String	**Camel 2.14.1** : 指定されたキーでプロパティーを検索します。キーが存在しないか値がない場合は、オプションのデフォルト値を指定できます。
routeId	String	**Camel 2.11:** Exchange がルーティングされている現在のルートの ID を返します。

変数	タイプ	説明
threadName	String	**Camel 2.3:** 現在のスレッドの名前を返します。ロギング目的で使用できます。
ref:xxx	Object	**Camel 2.6:** 指定された ID でレジストリーから Bean を検索します。
type:name.field	Object	**Camel 2.11:** FQN 名でタイプまたはフィールドを参照します。フィールドを参照するには、.FIELD_NAME を追加できます。たとえば、Exchange の定数フィールドを **org.apache.camel.Exchange.FILE_NAME** として参照できます。
null	null	**Camel 2.12.3: null** を表します。
random(value)	Integer	*Camel 2.16.0:* 0 (含まれる) から **値** (含まれない) までの間のランダムな整数を返します
random(min,max)	Integer	*Camel 2.16.0:* **min** (含まれる) から **max** (含まれない) までの間のランダムな整数を返します
collate(group)	List	**Camel 2.17:** collate 関数は、メッセージボディーを反復し、データを指定されたサイズのサブリストにグループ化します。これをスプリッター EIP と共に使用して、メッセージボディーを分割し、分割されたサブメッセージを N 個のサブリストのグループにグループ化/バッチ化できます。このメソッドは、Groovy の collate メソッドと同様に機能します。
skip(number)	Iterator	**Camel 2.19:** skip 関数はメッセージのボディーをイテレートし、最初の項目数をスキップします。Splitter EIP と併用することで、メッセージボディーを分割し、最初の N 項目数をスキップすることができます。
messageHistory	String	**Camel 2.17:** ルーティングされた現在の交換のメッセージ履歴。これは、未処理の例外が発生した場合にエラーハンドラーがログに記録するルートスタックトレースメッセージの履歴に似ています。
messageHistory(false)	String	**Camel 2.17:** messageHistory と同じですが、エクスチェンジの詳細はありません (ルート strack-trace のみが含まれます)。これは、メッセージ自体から機密データをログに記録しない場合に使用できます。

282.4. OGNL 式のサポート

Camel 2.3 の時点で利用可能

情報: Camel の OGNL サポートは、メソッドの呼び出しのみを対象としています。フィールドにアクセスできません。**Camel 2.11.1** 以降から、Java 配列の長さフィールドにアクセスするための特別なサポートが追加されました。

Simple および Bean 言語は、チェーンのような方法で Bean を呼び出すための Camel OGNL 表記をサポートするようになりました。Message IN ボディに **getAddress ()** メソッドがある POJO が含まれているとします。

次に、Camel OGNL 表記を使用してアドレスオブジェクトにアクセスできます。

```
simple("${body.address}")
simple("${body.address.street}")
simple("${body.address.zip}")
```

Camel は getter の短縮名を理解しますが、任意のメソッドを呼び出すか、次のような実際の名前を使用できます。

```
simple("${body.address}")
simple("${body.getAddress.getStreet}")
simple("${body.address.getZip}")
simple("${body.doSomething}")
```

たとえば、本文にアドレスがない場合は、Null セーフ演算子 (**?.**) を使用して NPE を回避することもできます。

```
simple("${body?.address?.street}")
```

Map または **List** タイプでインデックスを作成することもできるため、次のことができます。

```
simple("${body[foo].name}")
```

本文が **Map** ベースであると想定し、**foo** をキーとして値を検索し、その値に対して **getName** メソッドを呼び出します。

キーにスペースがある場合は、'foo bar' のようにキーを引用符で囲む **必要** があります。

```
simple("${body['foo bar'].name}")
```

キー名 (ドットありまたはドットなし) を使用して、**Map** オブジェクトまたは **List** オブジェクトに直接アクセスできます。

```
simple("${body[foo]}")
simple("${body[this.is.foo]}")
```

キー **foo** に値がないと仮定すると、null セーフ演算子を使用して、次のように NPE を回避できます。

```
simple("${body[foo]?.name}")
```

リスト 型にアクセスすることもできます。たとえば、次のようにしてアドレスから行を取得できます。

```
simple("${body.address.lines[0]}")
simple("${body.address.lines[1]}")
simple("${body.address.lines[2]}")
```

リストから最後の値を取得するために使用できる特別な **last** キーワードがあります。

```
simple("${body.address.lines[last]}")
```

-

最後から 2 番目の値を取得するには、数値を除算できるので、**last-1** を使用してこれを指定できます。

```
simple("${body.address.lines[last-1]}")
```

また、最後の 3 番目も当然、以下のようになります。

```
simple("${body.address.lines[last-2]}")
```

そして、リストの size メソッドを呼び出すことができます

```
simple("${body.address.lines.size}")
```

Camel 2.11.1 以降から、Java 配列の長さフィールドのサポートも追加しました。

```
String[] lines = new String[]{"foo", "bar", "cat"};
exchange.getIn().setBody(lines);

simple("There are ${body.length} lines")
```

はい、以下に示すように、これを演算子サポートと組み合わせることができます。

```
simple("${body.address.zip} > 1000")
```

282.5. 演算子サポート

パーサーは、単一の演算子のみをサポートするように制限されています。

有効にするには、左の値を $\{ } で囲む必要があります。構文は以下のようになります。

```
${leftValue} OP rightValue
```

rightValue は、**'** で囲まれた文字列リテラル、**null**、定数値、または $\{ } で囲まれた別の式に指定できます。

重要

演算子の前後にはスペースが **必要** です。

Camel は rightValue 型を leftValue 型に自動的に型変換します。文字列を数値に変換して、数値に対して > 比較を使用できるようにします。

次の演算子がサポートされています。

Operator	説明
==	等しい

Operator	説明
=~	**Camel 2.16:** equals は大文字と小文字を区別しません (文字列値を比較するときに大文字と小文字を区別しません)
>	は次の値よりも大きい:
>=	より大きいか等しい
<	は次の値よりも小さい:
⇐	より小さいか等しい
!=	等しくない
contains	文字列ベースの値に含まれているかどうかをテストします
not contains	文字列ベースの値に含まれていないかどうかをテストします
~~	文字列ベースの値で大文字と小文字の区別を無視して含まれているかどうかをテストします
regex	文字列値として定義された特定の正規表現パターンと照合するため
not regex	文字列値として定義された特定の正規表現パターンと一致しない場合
in	一連の値があり、各要素をコンマで区切る必要がある場合に照合するため空の値を含める場合は 2 つのコンマを使用して定義する必要があります。たとえば、',,bronze,silver,gold' のように、空の値と 3 つのメダルを含めた 4 つの値のセットです。
not in	値がセットでなく、各要素をコンマで区切る必要がある場合に照合するため。空の値を含める場合は 2 つのコンマを使用して定義する必要があります。たとえば、',,bronze,silver,gold' のように、空の値と 3 つのメダルを含めた 4 つの値のセットです。
is	左側の型が値のインスタンスである場合に照合するため。
not is	左側の型が値のインスタンスでない場合に照合するため。

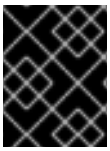
Operator	説明
range	左辺が数値として定義された値の範囲内にある場合に照合するため (**from..to**)。Camel 2.9 以降では、範囲の値を一重引用符で囲む必要があります。
not range	左辺が数値として定義された値の範囲内にない場合に照合するため (**from..to**。Camel 2.9 以降では、範囲の値を一重引用符で囲む必要があります。
starts with	**Camel 2.17.1, 2.18**: 左辺の文字列が右辺の文字列で開始するかどうかをテストします
ends with	**Camel 2.17.1, 2.18**: 左辺の文字列が右辺の文字列で終了するかどうかをテストします

また、次の単項演算子を使用できます。

演算子	説明
++	**Camel 2.9:** 数値を 1 ずつ増やします。左辺は関数でなければならず、それ以外の場合はリテラルとして解析されます。
–	**Camel 2.9:** 数値を 1 ずつ減らします。左辺は関数でなければならず、それ以外の場合はリテラルとして解析されます。
\	**Camel 2.9.3 から 2.10.x** 値をエスケープするには、たとえば \$ で $ 記号を示します。特殊: 改行には \n、タブには \t、キャリッジリターンには \r を使用します。**注意:** File Language を使用したエスケープはサポートされ **ていません**。**注意:** Camel 2.11 以降、エスケープ文字はサポートされなくなりましたが、次の 3 つの特別なエスケープ文字に置き換えられました。
\n	**Camel 2.11:** 改行文字を使用する場合。
\t	**Camel 2.11:** タブ文字を使用する場合。
\r	**Camel 2.11:** キャリッジリターン文字を使用する場合。
\}	**Camel 2.18:** } 文字をテキストとして使用する場合。

また、次の論理演算子を使用して式をグループ化できます。

演算子	説明
および	**非推奨** 代わりに && を使用します。and 論理演算子は、2 つの式のグループ化に使用されます。
または	|| **非推奨** 代わりに || を使用します。or 論理演算子は、2 つの式のグループ化に使用されます。
&&	**Camel 2.9:** and 論理演算子は、2 つの式のグループ化に使用されます。
||	**Camel 2.9:** or 論理演算子は、2 つの式のグループ化に使用されます。



重要

and、or 演算子の使用 Camel 2.4 以前では、and または **or** は simple 言語式で **1回しか** 使用できません。**Camel 2.5** 以降では、これらの演算子を複数回使用できます。

AND の構文は次のとおりです。

```
${leftValue} OP rightValue and ${leftValue} OP rightValue
```

OR の構文は次のとおりです。

```
${leftValue} OP rightValue or ${leftValue} OP rightValue
```

例:

```
// exact equals match
simple("${in.header.foo} == 'foo'")

// ignore case when comparing, so if the header has value FOO this will match
simple("${in.header.foo} =~ 'foo'")

// here Camel will type convert '100' into the type of in.header.bar and if it is an Integer '100' will also be converter to an Integer
simple("${in.header.bar} == '100'")

simple("${in.header.bar} == 100")

// 100 will be converter to the type of in.header.bar so we can do > comparison
simple("${in.header.bar} > 100")
```

282.5.1. さまざまなタイプとの比較

String や int などの異なる型と比較する場合は、少し注意が必要です。Camel は左辺のタイプを優先して使用します。右辺の型に基づいて両方の値を比較できなかった場合は、右辺の型にフォールバックします。
これは、値を反転して特定のタイプを適用できることを意味します。上記のバーの値が文字列であるとします。次に、方程式を反転できます。

```
simple("100 < ${in.header.bar}")
```

これにより、int 型が最優先で使用されます。

これは、Camel チームがバイナリー比較演算を改善して、文字列ベースよりも数値型を優先する場合に、今後変更される可能性があります。数値と比較するときに問題を引き起こすのは、ほとんどの場合 String 型です。

```
// testing for null
simple("${in.header.baz} == null")

// testing for not null
simple("${in.header.baz} != null")
```

また、正しい値が別の式である場合の、もう少し高度な例

```
simple("${in.header.date} == ${date:now:yyyyMMdd}")

simple("${in.header.type} == ${bean:orderService?method=getOrderType}")
```

contains が含まれる例。タイトルに Camel という用語が含まれるかをテストします。

```
simple("${in.header.title} contains 'Camel'")
```

正規表現が含まれる例。数のヘッダーが 4 桁の値かどうかをテストします。

```
simple("${in.header.number} regex '\\d{4}'")
```

最後に、ヘッダーがリスト内のいずれかの値と等しい場合の例を示します。各要素はコンマで区切る必要があり、前後にスペースを入れないでください。
Camel は各要素を左辺の型に変換するため、これは数値などにも機能します。

```
simple("${in.header.type} in 'gold,silver'")
```

また、最後の 3 つについては、not を使用した否定テストもサポートしています。

```
simple("${in.header.type} not in 'gold,silver'")
```

そして、文字列など、タイプが特定のインスタンスであるかどうかをテストできます。

```
simple("${in.header.type} is 'java.lang.String'")
```

次のように記述できるように、すべての **java.lang** タイプの短縮形を追加しました。

```
simple("${in.header.type} is 'String'")
```

範囲もサポートされています。範囲間隔には数字が必要で、最初と最後の両方の値が含まれます。たとえば、値が 100 から 199 の間であるかどうかをテストするには、次のようにします。

```
simple("${in.header.number} range 100..199")
```

スペースなしの範囲で **..** を使用していることに注意してください。Groovy と同じ構文に基づいています。

Camel 2.9 以降では、範囲の値は一重引用符で囲む必要があります

```
simple("${in.header.number} range '100..199'")
```

282.5.2. Spring XML の使用

Spring XML は、さまざまなビルダーメソッドをすべて備えた Java DSL ほど強力ではないため、単純な演算子でテストするには、他の言語を使用する必要があります。これで、Simple 言語でこれを行うことができます。以下のサンプルでは、ヘッダーがウィジェットの注文かどうかをテストします。

```
<from uri="seda:orders">
  <filter>
    <simple>${in.header.type} == 'widget'</simple>
    <to uri="bean:orderService?method=handleWidget"/>
  </filter>
</from>
```

282.6. AND/OR の使用

式が 2 つある場合は、**and** または **or** 演算子で結合できます。

ヒント

Camel 2.9 以降 Camel 2.9 以降は、&& または || を使用します。

たとえば、以下のようになります。

```
simple("${in.header.title} contains 'Camel' and ${in.header.type'} == 'gold'")
```

もちろん **or** もサポートされています。サンプルは次のようになります。

```
simple("${in.header.title} contains 'Camel' or ${in.header.type'} == 'gold'")
```

注意: 現在、**and** または **or** は、simple 言語表現で **1回しか** 使用できません。これは将来変更される可能性があります。
したがって、次のことは **できません**。

```
simple("${in.header.title} contains 'Camel' and ${in.header.type'} == 'gold' and ${in.header.number}
range 100..200")
```

282.7. サンプル

以下の Spring XML サンプルでは、ヘッダー値に基づいてフィルタリングします。

```
<from uri="seda:orders">
  <filter>
    <simple>${in.header.foo}</simple>
```

```
      <to uri="mock:fooOrders"/>
    </filter>
  </from>
```

Simple 言語は、上記のメッセージフィルターパターンの述語テストに使用できます。ここでは、メッセージに **foo** ヘッダーがあるかどうかをテストします (キーが **foo** のヘッダーが存在します)。式が **true** と評価された場合に、メッセージは **mock:fooOrders** エンドポイントにルーティングされます。それ以外の場合、メッセージは破棄されます。

Java DSL での同じ例:

```
from("seda:orders")
    .filter().simple("${in.header.foo}")
        .to("seda:fooOrders");
```

Simple 言語は、次のような単純なテキスト連結にも使用できます。

```
from("direct:hello")
    .transform().simple("Hello ${in.header.user} how are you?")
    .to("mock:reply");
```

Camel が正しく解析できるように、式で $\{ } プレースホルダーを使用する必要があることに注意してください。

このサンプルでは、date コマンドを使用して現在の日付を出力しています。

```
from("direct:hello")
    .transform().simple("The today is ${date:now:yyyyMMdd} and it is a great day.")
    .to("mock:reply");
```

以下のサンプルでは、Bean 言語を呼び出して、返される文字列に含まれる Bean のメソッドを呼び出します。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator}")
    .to("mock:reply");
```

orderIdGenerator は、レジストリーに登録されている Bean の ID です。Spring を使用している場合は、Spring Bean ID です。

注文 ID ジェネレーター Bean で呼び出すメソッドを宣言する場合は、**generateId** メソッドを呼び出す場所に、以下のように **.method name** を追加する必要があります。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator.generateId}")
    .to("mock:reply");
```

Bean コンポーネント自体に慣れている場合に **?method=methodname** オプションを使用できます。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator?method=generateId}")
    .to("mock:reply");
```

また、Camel 2.3 以降では、ボディを特定の型に変換することもできます。

```
<transform>
  <simple>Hello ${bodyAs(String)} how are you?</simple>
</transform>
```

簡略表記を含む型がいくつかあるため、**java.lang.String** の代わりに **String** を使用できます。これらは **byte[], String, Integer, Long** です。他のすべてのタイプは、**org.w3c.dom.Document** などの FQN 名を使用する必要があります。

Camel 2.3 以降では、ヘッダー **Map** から値を検索することもできます。

```
<transform>
  <simple>The gold value is ${header.type[gold]}</simple>
</transform>
```

上記のコードでは、名前 **type** でヘッダーを検索してして、その内容を **java.util.Map** と見なし、その後にキー **gold** で検索して値を返します。ヘッダーが Map に変換できない場合、例外が出力されます。name **型** のヘッダーが存在しない場合は **null** が返されます。

Camel 2.9 以降では、以下に示すように関数をネストできます。

```
<setHeader headerName="myHeader">
  <simple>${properties:${header.someKey}}</simple>
</setHeader>
```

282.8. 定数または列挙型の参照

Camel 2.11 から利用可能

顧客の列挙型があるとします。

また、Content Based Router では、Simple 言語を使用してこの列挙型を参照し、一致する列挙型のメッセージを確認できます。

282.9. XML DSL での改行またはタブの使用

Camel 2.9.3 以降で利用可能

Camel 2.9.3 以降では、値をエスケープできるようになったため、XML DSL で改行またはタブを指定するのが簡単になりました。

```
<transform>
  <simple>The following text\nis on a new line</simple>
</transform>
```

282.10. 先頭と末尾の空白の処理

Camel 2.10.0 以降で利用可能

Camel 2.10.0 以降では、式のトリム属性を使用して、先頭と末尾の空白文字を削除するか保持するかを制御できます。デフォルト値は true で、空白文字が削除されます。

```
<setBody>
  <simple trim="false">You get some trailing whitespace characters.     </simple>
</setBody>
```

282.11. 結果タイプの設定

Camel 2.8 から利用可能

Simple 式に結果の型を指定できるようになりました。これは、評価の結果が目的の型に変換されることを意味します。これは、ブール値、整数などの型を定義するのに最も便利です。

たとえば、ヘッダーをブール型として設定するには、次のようにします。

```
.setHeader("cool", simple("true", Boolean.class))
```

そして XML DSL では

```
<setHeader headerName="cool">
  <!-- use resultType to indicate that the type should be a java.lang.Boolean -->
  <simple resultType="java.lang.Boolean">true</simple>
</setHeader>
```

282.12. 関数の開始トークンと終了トークンの変更

Camel 2.9.1 以降で利用可能

Java コードを使用して、**SimpleLanguage** でセッター **changeFunctionStartToken** および **changeFunctionEndToken** を使用して、関数の開始トークンと終了トークン - $\{ } を設定できます。以下に示すように、Spring XML から、プロパティーで新しい変更されたトークンを使用して <bean> タグを定義できます。

```
<!-- configure Simple to use custom prefix/suffix tokens -->
<bean id="simple" class="org.apache.camel.language.simple.SimpleLanguage">
  <property name="functionStartToken" value="["/>
  <property name="functionEndToken" value="]"/>
</bean>
```

上記の例では、変更されたトークンとしてを使用しています。

start/end トークンを変更すると、クラスパスで同じ **camel-core** を共有するすべての Camel アプリケーションのトークンが変更されることに注意してください。
たとえば、OSGi サーバーでは、これは多くのアプリケーションに影響を与える可能性がありますが、WAR ファイルとしての Web アプリケーションは Web アプリケーションにのみ影響します。

282.13. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").simple("resource:classpath:mysimple.txt")
```

282.14. SPRING BEAN を EXCHANGE プロパティーに設定する

Camel 2.6 以降で利用可能

以下に示すように、Spring Bean を exchange プロパティーに設定できます。

```
<bean id="myBeanId" class="my.package.MyCustomClass" />
...
<route>
  ...
  <setProperty propertyName="monitoring.message">
    <simple>ref:myBeanId</simple>
  </setProperty>
  ...
</route>
```

282.15. 依存関係

Simple 言語は **camel-core** の一部です。

第283章 SIP コンポーネント

Camel バージョン 2.5 以降で利用可能

Camel の **sip** コンポーネントは、Jain SIP 実装 (JCP ライセンスで利用可能) に基づく通信コンポーネントです。

Session Initiation Protocol (SIP) は、IETF で定義されたシグナリングプロトコルであり、インターネットプロトコル (IP) 上で音声やビデオ通話などのマルチメディア通信セッションを制御するために広く使われています。SIP プロトコルは、基盤となるトランスポート層から独立するように設計されたアプリケーション層のプロトコルであり、TCP (送信制御プロトコル)、UDP (ユーザーデータグラムプロトコル)、SCTP (ストリーム制御伝送プロトコル) で実行可能です。

Jain SIP 実装は、TCP と UDP のみをサポートします。

Camel SIP コンポーネント は、RFC3903 - Session Initiation Protocol (SIP) Extension for Event で説明されているように、SIP Publish および Subscribe 機能 **のみ** をサポートします。

この camel コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方をサポートします。

Camel SIP プロデューサー (イベントパブリッシャー) と SIP コンシューマー (イベントサブスクライバー) は、SIP プレゼンスエージェント (ステートフルブローカーエンティティー) と呼ばれる中間エンティティーを使用して、イベントと状態の情報を相互に通信します。

SIP ベースの通信の場合、リスナーを持つ SIP スタックは、SIP プロデューサーとコンシューマーの両方でインスタンス化する **必要があります** (ローカルホストを使用する場合は別のポートを使用)。これは、通信中に SIP スタック間で交換されるハンドシェイクと確認応答をサポートするために必要です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sip</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

283.1. URI 形式

一口エンドポイントの URI スキームは次のとおりです。

```
sip://johndoe@localhost:99999[?options]
sips://johndoe@localhost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーエンドポイントとコンシューマーエンドポイントをサポートします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

283.2. オプション

SIP コンポーネントは、SIP プロトコルを介して状態を伝達するために必要なカスタムステートフルヘッダーを作成するための設定オプションと機能の広範なセットを提供します。

SIP コンポーネントにはオプションがありません。

SIP エンドポイントは、URI 構文を使用して設定されます。

```
sip:uri
```

パスおよびクエリーパラメーターを使用します。

283.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
uri	**必須** 接続する SIP サーバーの URI (john:secretmyserver:9999 のようにユーザー名とパスワードを含めることができます)		URI

283.2.2. クエリーパラメーター(44 個のパラメーター):

名前	説明	デフォルト	タイプ
cacheConnections (common)	接続作成のコストを削減するために、SipStack によって接続をキャッシュする必要があります。これは、接続が長時間の会話に使用される場合に役立ちます。	false	boolean
contentSubType (common)	contentSubType の設定は、任意の有効な MimeSubType に設定できます。	plain	String
contentType (common)	contentType の設定は、任意の有効な MimeType に設定できます。	text	String
eventHeaderName (common)	文字列ベースのイベントタイプの設定。		String
eventId (common)	文字列ベースのイベント ID の設定。レジストリーベースの FromHeader が指定されていない場合は必須の設定		String
fromHost (common)	メッセージ発信者のホスト名。レジストリーベースの FromHeader が指定されていない場合は必須の設定です。		String
fromPort (common)	メッセージ発信元のポート。レジストリーベースの FromHeader が指定されていない場合は必須の設定です。		int

名前	説明	デフォルト	タイプ
fromUser (common)	メッセージ発信者のユーザー名。レジストリーベースのカスタム FromHeader が指定されていない限り、必須の設定です。		String
msgExpiration (common)	エンドポイントで受信したメッセージが有効と見なされる時間です。	3600	int
receiveTimeoutMillis (common)	別の SIP スタックから受信できる応答および/または確認応答を待機する時間を指定するための設定です。	10000	long
stackName (common)	SIP エンドポイントに関連付けられた SIP スタックインスタンスの名前。	NAME_NOT_SET	String
toHost (common)	メッセージ受信者のホスト名。レジストリーベースの ToHeader が指定されていない場合は必須の設定です。		String
toPort (common)	メッセージ受信者のポート名。レジストリーベースの ToHeader が指定されていない場合は必須の設定です。		int
toUser (common)	メッセージ受信者のユーザー名。レジストリーベースのカスタム ToHeader が指定されていない限り、必須の設定です。		String
transport (common)	トランスポートプロトコルの選択の設定。有効な選択肢は tcp または udp です。	tcp	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumer (consumer)	この設定は、このエンドポイント用に作成する必要があるヘッダーの種類 (FromHeader、ToHeader など) を決定するために使用されます。	false	boolean

名前	説明	デフォルト	タイプ
presenceAgent (consumer)	この設定は、Presence Agent とコンシューマーを区別するために使用されます。これは、SIP Camel コンポーネントに基本的なプレゼンスエージェントが付属しているためです (テスト目的のみ)。コンシューマーは、このフラグを true に設定する必要があります。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
addressFactory (advanced)	カスタム AddressFactory を使用する場合。		AddressFactory
callIdHeader (advanced)	通話の詳細を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.CallIdHeader を実装する必要があります。		CallIdHeader
contactHeader (advanced)	詳細な連絡先の詳細 (電子メール、電話番号など) を含むオプションのカスタムヘッダーオブジェクト。タイプ javax.sip.header.ContactHeader を実装する必要があります。		ContactHeader
contentTypeHeader (advanced)	メッセージコンテンツの詳細を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.ContentTypeHeader を実装する必要があります。		ContentTypeHeader
eventHeader (advanced)	イベントの詳細を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.EventHeader を実装する必要があります。		EventHeader
expiresHeader (advanced)	メッセージの有効期限の詳細を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.ExpiresHeader を実装する必要があります。		ExpiresHeader
extensionHeader (advanced)	ユーザー/アプリケーション固有の詳細を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.ExtensionHeader を実装する必要があります。		ExtensionHeader

名前	説明	デフォルト	タイプ
fromHeader (advanced)	メッセージ発信元の設定を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.FromHeader を実装する必要があります。		FromHeader
headerFactory (advanced)	カスタム HeaderFactory を使用する場合。		HeaderFactory
listeningPoint (advanced)	カスタム ListeningPoint 実装を使用する場合。		ListeningPoint
maxForwardsHeader (advanced)	最大プロキシー転送の詳細を含むカスタムヘッダーオブジェクト。このヘッダーは、可能な viaHeaders に制限を課します。タイプ javax.sip.header.MaxForwardsHeader を実装する必要があります。		MaxForwardsHeader
maxMessageSize (advanced)	最大許容メッセージサイズの設定 (バイト単位)。	1048576	int
messageFactory (advanced)	カスタム MessageFactory を使用する場合。		MessageFactory
sipFactory (advanced)	カスタム SipFactory を使用して、使用する SipStack を作成する場合。		SipFactory
sipStack (advanced)	カスタム SipStack を使用する場合。		SipStack
sipUri (advanced)	カスタム SipURI を使用する場合。何も設定されていない場合、オプション toUser toHost:toPort を使用するための SipUri フォールバック。		SipURI
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
toHeader (advanced)	メッセージ受信者の設定を含むカスタムヘッダーオブジェクト。タイプ javax.sip.header.ToHeader を実装する必要があります。		ToHeader
viaHeaders (advanced)	タイプ javax.sip.header.ViaHeader のカスタムヘッダーオブジェクトのリスト。リクエスト転送用のプロキシーアドレスを含む各 ViaHeader。(このヘッダーは、リクエストがリスナーに到着したときに各プロキシーによって自動的に更新されることに注意してください)		List

名前	説明	デフォルト	タイプ
implementationDebugLogFile (logging)	ロギングに使用するクライアントデバッグログファイルの名前。		String
implementationServerLogFile (logging)	ロギングに使用するサーバーログファイルの名前。		String
implementationTraceLevel (logging)	トレースのログレベル。	0	String
maxForwards (proxy)	最大プロキシー転送数。		int
useRouterForAllUris (proxy)	この設定は、要求がプロキシー経由で Presence Agent に送信されるときに使用されます。	false	boolean

283.3. SIP エンドポイントとの間でメッセージを送信する

283.3.1. Camel SIP パブリッシャーの作成

以下の例では、SIP イベントパブリケーションを送信するために SIP パブリッシャーが作成されます。ユーザー agent@localhost:5152。これは、SIP パブリッシャーとサブスクライバーの間のブローカーとして機能する SIP プレゼンスエージェントのアドレスです。

- client という名前の SIP スタックを使用する
- evtHdrName と呼ばれるレジストリーベースの eventHeader を使用する
- evtId と呼ばれるレジストリーベースの eventId を使用する
- リスナーが user2@localhost:3534 として設定された SIP スタックから
- 公開されているイベントは EVENT_A
- REQUEST_METHOD という必須ヘッダーが Request.Publish に設定されているため、エンドポイントがイベントパブリッシャーとして設定される

```
producerTemplate.sendBodyAndHeader(
    "sip://agent@localhost:5152?
stackName=client&eventHeaderName=evtHdrName&eventId=evtid&fromUser=user2&fromHost=localhc
st&fromPort=3534",
    "EVENT_A",
    "REQUEST_METHOD",
    Request.PUBLISH);
```

283.3.2. Camel SIP サブスクライバーの作成

以下の例では、SIP サブスクライバーは、に送信された SIP イベントパブリケーションを受信するために作成されます。
ユーザー johndoe@localhost:5154

- Subscriber という名前の SIP スタックを使用する
- agent@localhost:5152 という Presence Agent ユーザーに登録する
- evtHdrName と呼ばれるレジストリーベースの eventHeader を使用する。evtHdrName には、Event_A に設定されたイベントが含まれている
- evtId と呼ばれるレジストリーベースの eventId を使用する

```
@Override
protected RouteBuilder createRouteBuilder() throws Exception {
    return new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            // Create PresenceAgent
            from("sip://agent@localhost:5152?
stackName=PresenceAgent&presenceAgent=true&eventHeaderName=evtHdrName&eventId=evtid")
                .to("mock:neverland");

            // Create Sip Consumer(Event Subscriber)
            from("sip://johndoe@localhost:5154?
stackName=Subscriber&toUser=agent&toHost=localhost&toPort=5152&eventHeaderName=evtHdrNam
e&eventId=evtid")
                .to("log:ReceivedEvent?level=DEBUG")
                .to("mock:notification");

        }
    };
}
```

Camel SIP コンポーネントには、テストおよびデモ目的でのみ使用することを意図した Presence Agent も同梱されています。プレゼンスエージェントのインスタンス化の例は、上に示されています。

Presence Agent はユーザー agent@localhost:5152 として設定されており、パブリッシャーとサブスクライバーの両方と通信できることに注意してください。パブリッシャーおよびサブスクライバーとは別の SIP stackName があります。Camel コンシューマーとして設定されていますが、実際にはエンドポイント mock:neverland へのルートに沿ってメッセージを送信しません。

第284章 SIMPLE JMS BATCH コンポーネント

Camel バージョン 2.16 以降で利用可能

SJMS Batch は、JMS キューから高パフォーマンスのトランザクションバッチを消費するための特殊なコンポーネントです。これは、コンシューマー専用コンポーネントとアグリゲーターのハイブリッドと考えることができます。

Camel の一般的な使用例は、キューからのメッセージを消費し、集約された状態を別のエンドポイントに送信する前にそれらを一本化することです。処理を実行しているシステムに障害が発生した場合にデータが失われないようにするために、データは通常、トランザクション内でキューから消費され、JDBC Component にあるような永続的な **AggregationRepository** に集約されて格納されます。

アグリゲーターパターンの動作には、受信メッセージが集約される前に **AggregationRepository** からデータをフェッチし、その後結果を書き戻すことが含まれます。本質的に、集約されたアーティファクトの数が増えるにつれて、読み取りと書き込みにかかる時間が徐々に長くなります。これの影響を示す任意の時間単位を使用した大まかな例は次のとおりです。

項目	読み取り時間	書き込み時間	合計時間
0	0	1	1
1	1	2	4
2	2	3	9
3	3	4	16
4	4	5	25
5	5	6	36
6	6	7	49
7	7	8	64
8	8	9	81
9	9	10	100

対照的に、SJMS Batch コンポーネントを使用した消費パフォーマンスは直線的です。各メッセージは、次のメッセージがフェッチされる前に **AggregationStrategy** を使用して消費および集約されます。すべての消費と集約が単一の JMS トランザクションで実行されるため、中間状態を維持するために外部ストレージは必要ありません。これにより、上記の読み取りと書き込みのコストが回避されます。実際には、これにより数桁高いスループットが得られます。

最初のメッセージからのサイズまたは期間によって完了条件が満たされると、集約された **Exchange** がルートに渡されます。この **Exchange** の処理中に例外が出力されるか、システムがシャットダウンすると、元の消費されたすべてのメッセージが最終的にキューに戻されます (または、ブローカーの設定に応じて配信不能キューに配置されます)。

通常のアグリゲーターを使用する場合とは異なり、集約条件の機能はありません。つまり、メッセージを複数のグループにまとめて消費することはできません。消費されたすべてのメッセージは、1 つのバッチにまとめられます。

複数の JMS コンシューマーサポートが利用可能です。これにより、1 つのルートを使用して並行して消費し、同時に JMS メッセージグループなどの機能を使用して関連するメッセージをグループ化できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

284.1. URI 形式

```
sjms:[queue:]destinationName[?options]
```

ここで、**destinationName** はJMS キューです。デフォルトでは、 **destinationName** はキュー名として解釈されます。

```
sjms:FOO.BAR
```

必要に応じて、オプションの **queue:** 接頭辞を含めることができます。

```
sjms:queue:FOO.BAR
```

そのコンテキスト内でバッチ消費を使用する利点がないため、トピック消費はサポートされていません。トピックメッセージは通常非永続的であり、損失は許容されます。失敗したトランザクション内で消費された場合、トピックメッセージはブローカーによって再配信されない可能性があります。このシナリオでは、プレーンな SJMS コンシューマーエンドポイントを通常の非永続性に基づくアグリゲータと組み合わせて使用できます。

284.2. コンポーネントのオプションと設定

シンプル JMS バッチコンポーネントは、以下に示す 5個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionFactory (advanced)	SjmsBatchComponent を有効にするには、ConnectionFactory が必要です。		ConnectionFactory

名前	説明	デフォルト	タイプ
asyncStartListener (advanced)	ルートの開始時に consumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	int
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS Batch エンドポイントは、URI 構文を使用して設定されます。

```
sjms-batch:destinationName
```

パスおよびクエリーパラメーターを使用します。

284.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
destinationName	**必須** 宛先名。キューのみがサポートされており、名前の前に queue: を付けることができます。		String

284.2.2. クエリーパラメーター(23 個のパラメーター):

名前	説明	デフォルト	タイプ
aggregationStrategy (consumer)	**必須** バッチ化されたすべてのメッセージを 1 つのメッセージにマージする、使用する集約ストラテジー。		AggregationStrategy
allowNullBody (consumer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
completionInterval (consumer)	ミリ単位の完了間隔。これにより、間隔ごとにスケジュールされた固定レートでバッチが完了します。タイムアウトがトリガーされ、バッチにメッセージがなかった場合、バッチは空になることがあります。完了タイムアウトと完了間隔の両方を同時に使用することはできません。設定できるのは 1 つだけです。	1000	int
completionPredicate (consumer)	完了述語。述語が true と評価された場合にバッチを完了させます。述語は、文字列構文を使用した単純な言語を使用して設定することもできます。オプション eagerCheckCompletion を true に設定して、述語が受信メッセージとマッチさせることもできます。それ以外の場合は、集約されたメッセージとマッチさせます。		String
completionSize (consumer)	バッチが完了するまでに消費されるメッセージの数。	200	int
completionTimeout (consumer)	バッチが完了するときの最初の最初のメッセージの受信からのタイムアウト (ミリ秒)。タイムアウトがトリガーされ、バッチにメッセージがなかった場合、バッチは空になることがあります。完了タイムアウトと完了間隔の両方を同時に使用することはできません。設定できるのは 1 つだけです。	500	int
consumerCount (consumer)	消費する JMS セッションの数。	1	int

名前	説明	デフォルト	タイプ
eagerCheckCompletion (consumer)	完了の先行チェックを使用します。これは、completionPredicate が受信 Exchange を使用することを意味します。完了の先行チェックなしとは対照的に、completionPredicate は集約された Exchange を使用します。	false	boolean
includeAllJMSXProperties (consumer)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean
mapJmsMessage (consumer)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。詳細については、以下のマッピングの仕組みに関するセクションを参照してください。	true	boolean
pollDuration (consumer)	メッセージの各ポーリングの期間 (ミリ秒)。それが短く、バッチが開始されている場合は、completionTimeout が使用されます。	1000	int
sendEmptyMessageWhenIdle (consumer)	完了タイムアウトまたは間隔を使用する場合、タイムアウトがトリガーされ、バッチにメッセージがなかった場合、バッチが空になることがあります。このオプションが true で、バッチが空の場合、空のメッセージがバッチに追加されるため、空のメッセージがルーティングされます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
asyncStartListener (advanced)	ルートの開始時に consumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
keepAliveDelay (advanced)	有効なセッションを再確立する試行間の遅延 (ミリ秒)。これが正の値の場合、メッセージの消費中に IllegalStateException が発生すると、SjmsBatchConsumer は新しいセッションの作成を試みます。この遅延値を使用すると、ログのスパムを防ぐために試行間で一時停止できます。これが負の値 (デフォルトは -1) の場合、SjmsBatchConsumer は以前と同じように動作します。つまり、IllegalStateException が検出された場合は、ベイルアウトし、ルートがシャットダウンします。	-1	int
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy

名前	説明	デフォルト	タイプ
recoveryInterval (advanced)	リカバリーの試行の間隔を指定します。つまり、接続が更新されるタイミング（ミリ秒単位）を指定します。デフォルトは 5000 ミリ秒、つまり 5 秒です。	5000	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
timeoutCheckerExecutor Service (advanced)	completionInterval オプションを使用すると、完了間隔をトリガーするバックグラウンドスレッドが作成されます。個々のコンシューマーへ新しいスレッドを作成するのではなく、カスタムスレッドプールを提供する場合は、このオプションを設定します。		ScheduledExecutor Service

completionSize エンドポイント属性は、**completionTimeout** と組み合わせて使用されます。最初の条件が満たされると、集約された **Exchange** がルートに放出されます。

第285章 SIMPLE JMS コンポーネント

Camel バージョン 2.11 以降で利用可能

Simple JMS コンポーネント (SJMS) は、JMS クライアントの作成と設定に関してよく知られたベストプラクティスを使用する Camel で使用する JMS クライアントです。SJMS には、Camel 用に明示的に記述されたまったく新しい JMS クライアント API が含まれており、サードパーティーのメッセージング実装を排除して軽量で回復力を維持しています。次の機能が含まれています。

- 標準のキューとトピックのサポート (永続的および非永続的)
- InOnly & InOut MEP のサポート
- 非同期プロデューサーおよびコンシューマー処理
- 内部 JMS トランザクションのサポート

その他の主な機能は次のとおりです。

- Plugable 接続リソース管理
- セッション、コンシューマー、およびプロデューサーのプーリングとキャッシング管理
- バッチコンシューマーとプロデューサー
- トランザクションバッチのコンシューマーとプロデューサー
- カスタマイズ可能なトランザクションコミットストラテジーのサポート (ローカル JMS トランザクションのみ)

注記

SJMS の S の理由

S は Simple and Standard and Springless の略です。また、camel-jms はすでに取得されています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

285.1. URI 形式

```
sjms:[queue:|topic:]destinationName[?options]
```

ここで、**destinationName** は JMS キューまたはトピック名です。デフォルトでは、**destinationName** はキュー名として解釈されます。たとえば、キューに接続するには、**FOO.BAR** を次のように使用します。

```
sjms:FOO.BAR
```

必要に応じて、オプションの **queue:** 接頭辞を含めることができます。

```
sjms:queue:FOO.BAR
```

トピックに接続するには、**topic:** 接頭辞を含める **必要** があります。たとえば、トピック **Stocks.Prices** に接続するには、次を使用します。

```
sjms:topic:Stocks.Prices
```

?option=value&option=value&... の形式を使用して、クエリーオプションを URI に追加します。

285.2. コンポーネントのオプションと設定

Simple JMS コンポーネントは、以下に示す 15 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionFactory (advanced)	SjmsComponent を有効にするには、ConnectionFactory が必要です。直接設定することも、ConnectionResource の一部として設定することもできます。		ConnectionFactory
connectionResource (advanced)	ConnectionResource は、ConnectionFactory のカスタマイズとコンテナー制御を可能にするインターフェイスです。詳細については、プラグ可能な接続リソースの管理を参照してください。		ConnectionResource
connectionCount (common)	このコンポーネントで開始されたエンドポイントで使用可能な接続の最大数。	1	Integer
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel はすぐに使用できる 1 つの実装を提供します: デフォルトです。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
transactionCommit Strategy (transaction)	使用するコミットストラテジーの種類を設定する場合。Camel は、追加設定なしで、default と batch の 2 つの実装を提供します。		TransactionCommit ストラテジー

名前	説明	デフォルト	タイプ
destinationCreationStrategy (advanced)	カスタム DestinationCreationStrategy を使用する場合。		DestinationCreationStrategy
timedTaskManager (advanced)	カスタム TimedTaskManager を使用する場合。		TimedTaskManager
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
connectionTestOnBorrow (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合、プールから返される前に各 javax.jms.Connection をテストする (start を呼び出す) 必要があります。	true	boolean
connectionUsername (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するユーザー名。		String
connectionPassword (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するパスワード。		String
connectionClientId (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するクライアント ID。		String
connectionMaxWait (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合に、プールが使い果たされたときにブロックして空き接続を待機する最大待機時間 (ミリ単位)。	5000	long
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS エンドポイントは、URI 構文を使用して設定されます。

```
sjms:destinationType:destinationName
```

パスおよびクエリーパラメーターを使用します。

285.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
destinationType	使用する宛先の種類	queue	String
destinationName	**必須** DestinationName は、JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。		String

285.2.2. クエリーパラメーター(34 個のパラメーター):

名前	説明	デフォルト	タイプ
acknowledgementMode (common)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	SessionAcknowledgement タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerCount (consumer)	このエンドポイントに使用されるコンシューマーリスナーの数を設定します。	1	int

名前	説明	デフォルト	タイプ
durableSubscriptionId (consumer)	永続トピックに必要な永続サブスクリプション ID を設定します。		String
synchronous (consumer)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
messageSelector (consumer)	JMS メッセージセレクターの構文を設定します。		String
namedReplyTo (producer)	InOut プロデューサエンドポイントに使用される宛先名への返信を設定します。		String
persistent (producer)	メッセージの永続性を有効/無効にするために使用されるフラグ。	true	boolean
producerCount (producer)	このエンドポイントに使用されるプロデューサーの数を設定します。	1	int
ttl (producer)	生成されたメッセージの Time To Live 値を調整するために使用されるフラグ。	-1	long
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean
prefillPool (producer)	起動時にプロデューサ接続プールを事前に埋めるか、必要に応じて遅延接続を作成するか。	true	boolean
responseTimeOut (producer)	InOut 応答がタイムアウトするまでの待機時間を設定します。	5000	long

名前	説明	デフォルト	タイプ
asyncStartListener (advanced)	ルートの開始時に consumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、consumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
connectionCount (advanced)	このエンドポイントで利用可能な接続の最大数。		Integer
connectionFactory (advanced)	エンドポイントの connectionFactory を初期化します。これは、コンポーネントの connectionFactory よりも優先されます (存在する場合)。		ConnectionFactory
connectionResource (advanced)	エンドポイントの connectionResource を初期化します。これは、コンポーネントの connectionResource よりも優先されます (存在する場合)。		ConnectionResource
destinationCreationStrategy (advanced)	カスタム DestinationCreationStrategy を使用する場合。		DestinationCreation Strategy
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean

名前	説明	デフォルト	タイプ
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。詳細については、以下のマッピングの仕組みに関するセクションを参照してください。	true	boolean
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
errorHandlerLoggingLevel (logging)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean
transacted (transaction)	トランザクションモードを使用するかどうかを指定します	false	boolean
transactionBatchCount (transaction)	If transactioned は、トランザクションをコミットする前に処理するメッセージの数を設定します。	-1	int
transactionBatchTimeout (transaction)	バッチトランザクションのタイムアウト (ミリ秒単位) を設定します。値は 1000 以上である必要があります。	5000	long
transactionCommitStrategy (transaction)	コミット戦略を設定します。		TransactionCommit ストラテジー

名前	説明	デフォルト	タイプ
sharedJMSSession (transaction)	JMS セッションを他の SJMS エンドポイントと共有するかどうかを指定します。ルートが複数の JMS プロバイダーにアクセスしている場合は、これをオフにします。複数の JMS プロバイダーに対するトランザクションが必要な場合は、jms コンポーネントを使用して XA トランザクションを利用します。	true	boolean

以下は、必要な **ConnectionFactory** プロバイダーを使用して **SjmsComponent** を設定する方法の例です。デフォルトで単一の接続を作成し、コンポーネントの内部プーリング API を使用してそれを保存し、スレッドセーフな方法でセッション作成リクエストを処理できるようにします。

```
SjmsComponent component = new SjmsComponent();
component.setConnectionFactory(new ActiveMQConnectionFactory("tcp://localhost:61616"));
getContext().addComponent("sjms", component);
```

永続的なサブスクリプションをサポートするために必要な SJMS コンポーネントの場合、デフォルトの **ConnectionFactoryResource** インスタンスをオーバーライドして **clientId** プロパティーを設定できます。

```
ConnectionFactoryResource connectionResource = new ConnectionFactoryResource();
connectionResource.setConnectionFactory(new
ActiveMQConnectionFactory("tcp://localhost:61616"));
connectionResource.setClientId("myclient-id");

SjmsComponent component = new SjmsComponent();
component.setConnectionResource(connectionResource);
component.setMaxConnections(1);
```

285.3. プロデューサーの使用法

285.3.1. InOnly プロデューサー - (デフォルト)

InOnly プロデューサーは、SJMS プロデューサーエンドポイントのデフォルトの動作です。

```
from("direct:start")
    .to("sjms:queue:bar");
```

285.3.2. InOut プロデューサー

InOut 動作を有効にするには、**exchangePattern** 属性を URI に追加します。デフォルトでは、コンシューマーごとに専用の TemporaryQueue を使用します。

```
from("direct:start")
    .to("sjms:queue:bar?exchangePattern=InOut");
```

ただし、**namedReplyTo** を指定すると、より適切な監視ポイントを提供できます。

```
from("direct:start")
    .to("sjms:queue:bar?exchangePattern=InOut&namedReplyTo=my.reply.to.queue");
```

285.4. コンシューマーの使用

285.4.1. InOnly コンシューマー - (デフォルト)

InOnly xonsumer は、SJMS コンシューマーエンドポイントのデフォルトの Exchange 動作です。

```
from("sjms:queue:bar")
    .to("mock:result");
```

285.4.2. InOut コンシューマー

InOut 動作を有効にするには、**exchangePattern** 属性を URI に追加します。

```
from("sjms:queue:in.out.test?exchangePattern=InOut")
    .transform(constant("Bye Camel"));
```

285.5. 高度な使用上の注意

285.5.1. Plugable 接続リソース管理

SJMS は、組み込みの接続プールを通じて JMS **接続** リソース管理を提供します。これにより、サードパーティーの API プーリングロジックに依存する必要がなくなります。ただし、J2EE または OSGi コンテナーによって提供されるものなど、外部接続リソースマネージャーを使用する必要がある場合があります。このため、SJMS は、内部 SJMS 接続プール機能をオーバーライドするために使用できるインターフェイスを提供します。これは、**ConnectionResource** インターフェイスを通じて実現されます。

ConnectionResource は、SJMS コンポーネントに **接続** プールを提供するために使用される契約であり、必要に応じて接続を借用および返却するためのメソッドを提供します。ユーザーは、SJMS を外部接続プーリングマネージャーと統合する必要がある場合に使用する必要があります。

ただし、標準の **ConnectionFactory** プロバイダーの場合は、SJMS で提供される **ConnectionFactoryResource** 実装をそのまま使用するか、このコンポーネント用に最適化して拡張することをお勧めします。

以下は、ActiveMQ **PooledConnectionFactory** でプラグ可能な ConnectionResource を使用する例です。

```
public class AMQConnectionResource implements ConnectionResource {
    private PooledConnectionFactory pcf;

    public AMQConnectionResource(String connectString, int maxConnections) {
        super();
        pcf = new PooledConnectionFactory(connectString);
        pcf.setMaxConnections(maxConnections);
        pcf.start();
    }
```

```
    public void stop() {
        pcf.stop();
    }

    @Override
    public Connection borrowConnection() throws Exception {
        Connection answer = pcf.createConnection();
        answer.start();
        return answer;
    }

    @Override
    public Connection borrowConnection(long timeout) throws Exception {
        // SNIPPED...
    }

    @Override
    public void returnConnection(Connection connection) throws Exception {
        // Do nothing since there isn't a way to return a Connection
        // to the instance of PooledConnectionFactory
        log.info("Connection returned");
    }
}
```

次に、**ConnectionResource** を **SjmsComponent** に渡します。

```
CamelContext camelContext = new DefaultCamelContext();
AMQConnectionResource pool = new AMQConnectionResource("tcp://localhost:33333", 1);
SjmsComponent component = new SjmsComponent();
component.setConnectionResource(pool);
camelContext.addComponent("sjms", component);
```

完全な使用例を確認するには、**ConnectionResourceIT** を参照してください。

285.5.2. バッチメッセージのサポート

SjmsProducer は、**List** をカプセル化する Exchange を作成することにより、メッセージのコレクションの発行をサポートします。この SjmsProducer は、List の内容を繰り返し処理し、各メッセージを個別に公開します。

メッセージのバッチを生成するときに、各メッセージに固有のヘッダーを設定する必要がある場合は、SJMS **BatchMessage** クラスを使用できます。SjmsProducer が **BatchMessage** リストに遭遇すると、各 **BatchMessage** を反復処理し、含まれているペイロードとヘッダーを公開します。

以下は、BatchMessage クラスの使用例です。まず、**BatchMessage** のリストを作成します:

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

次に、リストを公開します。

```
template.sendBody("sjms:queue:batch.queue", messages);
```

285.5.3. カスタマイズ可能なトランザクションコミットストラテジー (ローカル JMS トランザクションのみ)

SJMS は、**TransactionCommitStrategy** インターフェイスを使用して、カスタムでプラグ可能なトランザクションストラテジーを作成する手段を開発者に提供します。これにより、ユーザーは、**SessionTransactionSynchronization** が セッションをいつコミットするかを決定するために使用する一連の固有の状況を定義できます。その使用例は、次のセクションで詳しく説明する **BatchTransactionCommitStrategy** です。

285.5.4. トランザクションバッチのコンシューマーとプロデューサー

SJMS コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方でローカル JMS トランザクションのバッチ処理をサポートするように設計されています。ただし、それらがそれぞれでどのように処理されるかは非常に異なります。

SJMS コンシューマーエンドポイントは、X メッセージを関連するセッションでコミットする前に処理する単純な実装です。コンシューマーでバッチ処理されたトランザクションを有効にするには、まず、**transacted** パラメーターを true に設定してトランザクションを有効にし、次に **transactionBatchCount** を追加して 0 より大きい任意の値に設定します。たとえば、次の設定では、10 メッセージごとにセッションがコミットされます。

```
sjms:queue:transacted.batch.consumer?transacted=true&transactionBatchCount=10
```

コンシューマーエンドポイントでのバッチの処理中に例外が発生した場合、セッションロールバックが呼び出され、メッセージが次に使用可能なコンシューマーに再配信されます。関連するセッションの **BatchTransactionCommitStrategy** のカウンターも 0 にリセットされます。JMSRedelivered ヘッダーが true に設定されたメッセージを監視するために、バッチメッセージのプロセッサーにフックを配置することは、ユーザーの責任です。これは、メッセージがある時点でロールバックされたこと、および処理が成功したことを確認する必要があることを示しています。

トランザクション処理されたバッチコンシューマーには、セッションで開いているトランザクションをコミットする前にメッセージ間で既定の時間 (5000 ミリ秒) 待機する内部タイマーのインスタンスも含まれます。デフォルト値の 5000 ミリ秒 (最小 1000 ミリ秒) は、ほとんどのユースケースに適していますが、さらに調整が必要な場合は、単に **transactionBatchTimeout** パラメーターを設定してください。

```
sjms:queue:transacted.batch.consumer?
transacted=true&transactionBatchCount=10&transactionBatchTimeout=2000
```

受け入れられる最小値は 1000 ミリ秒です。これは、コンテキスト切り替えの量が不要なパフォーマンスへの影響をもたらし、メリットが得られない可能性があるためです。

ただし、プロデューサーエンドポイントの処理方法は大きく異なります。プロデューサでは、各メッセージが宛先に配信された後、Exchange が閉じられ、そのメッセージへの参照がなくなります。再配信可能なすべてのメッセージを利用できるようにするには、BatchMessage をパブリッシュしているプロデューサーエンドポイントでトランザクションを有効にするだけです。トランザクションは、バッチリスト内のすべてのメッセージを含む交換の最後にコミットされます。追加の設定は必要ありません。以下に例を示します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
```

```
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

トランザクションを有効にしてリストを公開します。

```
template.sendBody("sjms:queue:batch.queue?transacted=true", messages);
```

285.6. 追記

285.6.1. メッセージヘッダーの形式

SJMS コンポーネントは、Camel JMS コンポーネントで使用されるのと同じヘッダー形式戦略を使用します。このプラグ可能な戦略により、ネットワーク経由で送信されるメッセージが JMS メッセージ仕様に準拠することが保証されます。

exchange.in.header の場合、次のルールがヘッダーキーに適用されます。

- **JMS** または **JMSX** で始まるキーは予約されています。
- **exchange.in.headers** キーはリテラルで、すべて有効な Java 識別子である必要があります (キー名にドットを使用しないでください)。
- Camel は、JMS メッセージを消費するときにドットとハイフンを置き換え、その逆を行います。
  - Camel がメッセージを消費するときは、**DOT** と逆の置換に置き換えられます。
  - Camel がメッセージを消費するときは、**HYPHEN** と逆の置換に置き換えられます。オプション **jmsKeyFormatStrategy** も参照してください。これにより、キーのフォーマットに独自のカスタム戦略を使用できます。

exchange.in.header の場合、次のルールがヘッダー値に適用されます。

285.6.2. メッセージ内容

ネットワーク経由でコンテンツを配信するには、配信されるメッセージの本文が JMS メッセージ仕様に準拠していることを確認する必要があります。したがって、生成されるものはすべて、プリミティブまたはそのカウンターオブジェクト (**Integer**、**Long**、**Character** など) のいずれかでなければなりません。タイプ **String**、**CharSequence**、**Date**、**BigDecimal**、および **BigInteger** はすべて、それらの **toString()** 表現に変換されます。他のすべてのタイプはドロップされます。

285.6.3. クラスタリング

クラスター化された環境で SJMS で **InOut** を使用する場合は、TemporaryQueue 宛先を使用するか、InOut プロデューサーエンドポイントごとに宛先への一意の名前付きレスポンスを使用する必要があります。メッセージ相関は、ブローカーのメッセージセレクターではなく、エンドポイントによって処理されます。InOut プロデューサエンドポイントは、メッセージ **JMSCorrelationID** によってキャッシュされた Java Concurrency Exchangers を使用します。これにより、パフォーマンスが大幅に向上し、ブローカーのオーバーヘッドが削減されます。これは、すべてのメッセージが、関心のあるコンシューマーによって生成された順序で宛先から消費されるためです。

現在、唯一の相関ストラテジーは **JMSCorrelationId** を使用することです。**InOut** Consumer はこのストラテジーを使用し、含まれている **JMSReplyTo** 宛先へのすべてのレスポンスメッセージにも、リクエストからコピーされた **JMSCorrelationId** が含まれるようにします。

285.7. トランザクションサポート

SJMS は現在、内部 JMS トランザクションの使用のみをサポートしています。Camel Transaction Processor または Java Transaction API (JTA) はサポートされていません。

285.7.1. Springless とは Spring を使用できないということか

そうではまったくありません。以下は、Spring DSL を使用した SJMS コンポーネントの例です。

```
<route
    id="inout.named.reply.to.producer.route">
    <from
        uri="direct:invoke.named.reply.to.queue" />
    <to
        uri="sjms:queue:named.reply.to.queue?
namedReplyTo=my.response.queue&amp;amp;exchangePattern=InOut" />
</route>
```

Springless とは、Spring JMS API への依存関係から離れることを指します。SJMS を強化するために、新しい JMS クライアント API がゼロから開発されています。

第286章 SIMPLE JMS2 コンポーネント

Camel バージョン 2.19 以降で利用可能

Simple JMS 2.0 コンポーネント (SJMS2) は、JMS クライアントの作成と設定に関してよく知られているベストプラクティスを使用する Camel で使用する JMS クライアントです。SJMS2 には、Camel 用に明示的に記述されたまったく新しい JMS 2.0 クライアント API が含まれており、サードパーティーのメッセージング実装を排除して、軽量で回復力を維持しています。次の機能が含まれています。

- 標準のキューとトピックのサポート (永続的および非永続的)
- InOnly & InOut MEP のサポート
- 非同期プロデューサーおよびコンシューマー処理
- 内部 JMS トランザクションのサポート

その他の主な機能は次のとおりです。

- Plugable 接続リソース管理
- セッション、コンシューマー、およびプロデューサーのプーリングとキャッシング管理
- バッチコンシューマーとプロデューサー
- トランザクションバッチのコンシューマーとプロデューサー
- カスタマイズ可能なトランザクションコミットストラテジーのサポート (ローカル JMS トランザクションのみ)

注記

SJMS の S の理由

S は Simple and Standard and Springless の略です。また、camel-jms はすでに取得されています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

286.1. URI 形式

```
sjms2:[queue:|topic:]destinationName[?options]
```

ここで、**destinationName** は JMS キューまたはトピック名です。デフォルトでは、**destinationName** はキュー名として解釈されます。たとえば、キューに接続するには、**FOO.BAR** を次のように使用します。

```
sjms2:FOO.BAR
```

必要に応じて、オプションの **queue:** 接頭辞を含めることができます。

```
sjms2:queue:FOO.BAR
```

トピックに接続するには、**topic:** 接頭辞を含める **必要** があります。たとえば、トピック **Stocks.Prices** に接続するには、次を使用します。

```
sjms2:topic:Stocks.Prices
```

?option=value&option=value&... の形式を使用して、クエリーオプションを URI に追加します。

286.2. コンポーネントのオプションと設定

Simple JMS2 コンポーネントは、以下に示す 15 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionFactory (advanced)	SjmsComponent を有効にするには、ConnectionFactory が必要です。直接設定することも、ConnectionResource の一部として設定することもできます。		ConnectionFactory
connectionResource (advanced)	ConnectionResource は、ConnectionFactory のカスタマイズとコンテナー制御を可能にするインターフェイスです。詳細については、プラグ可能な接続リソースの管理を参照してください。		ConnectionResource
connectionCount (common)	このコンポーネントで開始されたエンドポイントで使用可能な接続の最大数。	1	Integer
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能な戦略。Camel はすぐに使用できる 1 つの実装を提供します: デフォルトです。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
transactionCommit Strategy (transaction)	使用するコミットストラテジーの種類を設定する場合。Camel は、追加設定なしで、default と batch の 2 つの実装を提供します。		TransactionCommit ストラテジー

名前	説明	デフォルト	タイプ
destinationCreation Strategy (advanced)	カスタム DestinationCreationStrategy を使用する場合。		DestinationCreation Strategy
timedTaskManager (advanced)	カスタム TimedTaskManager を使用する場合。		TimedTaskManager
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
connectionTestOnBorrow (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合、プールから返される前に各 javax.jms.Connection をテストする (start を呼び出す) 必要があります。	true	boolean
connectionUsername (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するユーザー名。		String
connectionPassword (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するパスワード。		String
connectionClientId (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用するときに javax.jms.Connection を作成するときに使用するクライアント ID。		String
connectionMaxWait (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合に、プールが使い果たされたときにブロックして空き接続を待機する最大待機時間 (ミリ単位)。	5000	long
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS2 エンドポイントは、URI 構文を使用して設定されます。

```
sjms2:destinationType:destinationName
```

パスおよびクエリーパラメーターを使用します。

286.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
destinationType	使用する宛先の種類	queue	String
destinationName	**必須** DestinationName は、JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。		String

286.2.2. クエリーパラメーター(37 個のパラメーター):

名前	説明	デフォルト	タイプ
acknowledgementMode (common)	JMS 確認応答名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	SessionAcknowledgement タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerCount (consumer)	このエンドポイントに使用されるコンシューマーリスナーの数を設定します。	1	int

名前	説明	デフォルト	タイプ
durable (consumer)	トピックコンシューマーを永続に設定します。	false	boolean
durableSubscriptionId (consumer)	永続トピックに必要な永続サブスクリプション ID を設定します。		String
shared (consumer)	コンシューマーを共有に設定します。	false	boolean
subscriptionId (consumer)	永続トピックまたは共有トピックに必要なサブスクリプション ID を設定します。		String
synchronous (consumer)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
messageSelector (consumer)	JMS メッセージセレクターの構文を設定します。		String
namedReplyTo (producer)	InOut プロデューサエンドポイントに使用される宛先名への返信を設定します。		String
persistent (producer)	メッセージの永続性を有効/無効にするために使用されるフラグ。	true	boolean
producerCount (producer)	このエンドポイントに使用されるプロデューサーの数を設定します。	1	int
ttl (producer)	生成されたメッセージの Time To Live 値を調整するために使用されるフラグ。	-1	long
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false でメッセージボディーが null の場合は、JMSException が出力されます。	true	boolean

名前	説明	デフォルト	タイプ
prefillPool (producer)	起動時にプロデューサ接続プールを事前に埋めるか、必要に応じて遅延接続を作成するか。	true	boolean
responseTimeOut (producer)	InOut 応答がタイムアウトするまでの待機時間を設定します。	5000	long
asyncStartListener (advanced)	ルートの開始時に consumer メッセージリスナーを非同期で開始するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの開始時に Camel がブロックされます。このオプションを true に設定すると、ルートの起動を許可します。一方、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用する場合は、接続を確立できない場合は例外が WARN レベルでログに記録され、コンシューマーはメッセージを受信できず、ルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに、consumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
connectionCount (advanced)	このエンドポイントで利用可能な接続の最大数。		Integer
connectionFactory (advanced)	エンドポイントの connectionFactory を初期化します。これは、コンポーネントの connectionFactory よりも優先されます (存在する場合)。		ConnectionFactory
connectionResource (advanced)	エンドポイントの connectionResource を初期化します。これは、コンポーネントの connectionResource よりも優先されます (存在する場合)。		ConnectionResource
destinationCreationStrategy (advanced)	カスタム DestinationCreationStrategy を使用する場合。		DestinationCreation Strategy
exceptionListener (advanced)	基礎となる JMS 例外の通知を受ける JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
includeAllJMSXProperties (advanced)	JMS から Camel Message へのマッピング時に JMSXxxx プロパティーをすべて含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合、このオプションは適用されません。	false	boolean
jmsKeyFormatStrategy (advanced)	JMS 仕様に準拠できるように、JMS キーをエンコードおよびデコードするためのプラグ可能なストラテジー。Camel は、デフォルトとパススルーの 2 つの実装をそのまま提供します。デフォルトのストラテジーでは、ドットとハイフン（. および -）を安全にマーシャリングします。パススルー戦略では、キーはそのまま残ります。JMS ヘッダーキーに不正な文字が含まれているかどうかは問題にならない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ (javax.jms.TextMessage を文字列など) に自動マップするかどうかを指定します。詳細については、以下のマッピングの仕組みに関するセクションを参照してください。	true	boolean
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信しているときに、Camel が javax.jms.Message オブジェクトの新しいインスタンスを作成するときに呼び出される、指定された MessageCreatedStrategy を使用します。		MessageCreatedStrategy
errorHandlerLoggingLevel (logging)	キャッチされていない例外をログに記録するためのデフォルトの errorHandler ログレベルを設定できます。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler でスタックトレースをログに記録するかどうかを制御できます。	true	boolean
transacted (transaction)	トランザクションモードを使用するかどうかを指定します	false	boolean
transactionBatchCount (transaction)	If transactioned は、トランザクションをコミットする前に処理するメッセージの数を設定します。	-1	int

名前	説明	デフォルト	タイプ
transactionBatchTimeout (transaction)	バッチトランザクションのタイムアウト (ミリ秒単位) を設定します。値は 1000 以上である必要があります。	5000	long
transactionCommitStrategy (transaction)	コミット戦略を設定します。		TransactionCommit ストラテジー
sharedJMSSession (transaction)	JMS セッションを他の SJMS エンドポイントと共有するかどうかを指定します。ルートが複数の JMS プロバイダーにアクセスしている場合は、これをオフにします。複数の JMS プロバイダーに対するトランザクションが必要な場合は、jms コンポーネントを使用して XA トランザクションを利用します。	true	boolean

以下は、必要な **ConnectionFactory** プロバイダーを使用して **Sjms2Component** を設定する方法の例です。デフォルトで単一の接続を作成し、コンポーネントの内部プーリング API を使用してそれを保存し、スレッドセーフな方法でセッション作成リクエストを処理できるようにします。

```
Sjms2Component component = new Sjms2Component();
component.setConnectionFactory(new ActiveMQConnectionFactory("tcp://localhost:61616"));
getContext().addComponent("sjms2", component);
```

永続的なサブスクリプションをサポートするために必要な SJMS2 コンポーネントの場合、デフォルトの **ConnectionFactoryResource** インスタンスをオーバーライドして **clientId** プロパティーを設定できます。

```
ConnectionFactoryResource connectionResource = new ConnectionFactoryResource();
connectionResource.setConnectionFactory(new
ActiveMQConnectionFactory("tcp://localhost:61616"));
connectionResource.setClientId("myclient-id");

Sjms2Component component = new Sjms2Component();
component.setConnectionResource(connectionResource);
component.setMaxConnections(1);
```

286.3. プロデューサーの使用法

286.3.1. InOnly プロデューサー - (デフォルト)

InOnly プロデューサーは、SJMS2 プロデューサーエンドポイントのデフォルトの動作です。

```
from("direct:start")
    .to("sjms2:queue:bar");
```

286.3.2. InOut プロデューサー

InOut 動作を有効にするには、**exchangePattern** 属性を URI に追加します。デフォルトでは、コンシューマーごとに専用の TemporaryQueue を使用します。

```
from("direct:start")
    .to("sjms2:queue:bar?exchangePattern=InOut");
```

ただし、**namedReplyTo** を指定すると、より適切な監視ポイントを提供できます。

```
from("direct:start")
    .to("sjms2:queue:bar?exchangePattern=InOut&namedReplyTo=my.reply.to.queue");
```

286.4. コンシューマーの使用

286.4.1. 永続共有サブスクリプション

1 人以上のコンシューマー間で共有できる永続的なサブスクリプションを作成する場合、JMS 2.0 準拠の接続ファクトリーを使用して、共通の subscriptionId を指定します。次に、サブスクリプションプロパティーの耐久性と共有を true に設定します。

```
from("sjms2:topic:foo?consumerCount=3&subscriptionId=bar&durable=true&shared=true")
    .to("mock:result");

from("sjms2:topic:foo?consumerCount=2&subscriptionId=bar&durable=true&shared=true")
    .to("mock:result");
```

286.4.2. InOnly コンシューマー - (デフォルト)

InOnly コンシューマーは、SJMS2 コンシューマーエンドポイントのデフォルトの Exchange 動作です。

```
from("sjms2:queue:bar")
    .to("mock:result");
```

286.4.3. InOut コンシューマー

InOut 動作を有効にするには、**exchangePattern** 属性を URI に追加します。

```
from("sjms2:queue:in.out.test?exchangePattern=InOut")
    .transform(constant("Bye Camel"));
```

286.5. 高度な使用上の注意

286.5.1. Plugable 接続リソース管理

SJMS2 は、組み込みの接続プールを通じて JMS **Connection** リソース管理を提供します。これにより、サードパーティーの API プーリングロジックに依存する必要がなくなります。ただし、J2EE または OSGi コンテナーによって提供されるものなど、外部接続リソースマネージャーを使用する必要がある場合があります。このため、SJMS2 は、内部 SJMS2 接続プーリング機能をオーバーライドするために使用できるインターフェイスを提供します。これは、**ConnectionResource** インターフェイスを通じて実現されます。

ConnectionResource は、SJMS2 コンポーネントに **Connection** プールを提供するために使用されるコントラクトであり、必要に応じて接続を借用および返却するためのメソッドを提供します。ユーザーは、SJMS2 を外部接続プーリングマネージャーと統合する必要がある場合に使用する必要があります。

ただし、標準の **ConnectionFactory** プロバイダーの場合は、SJMS2 で提供される **ConnectionFactoryResource** 実装をそのまま使用するか、このコンポーネント用に最適化して拡張することをお勧めします。

以下は、ActiveMQ **PooledConnectionFactory** でプラグ可能な ConnectionResource を使用する例です。

```
public class AMQConnectionResource implements ConnectionResource {
    private PooledConnectionFactory pcf;

    public AMQConnectionResource(String connectString, int maxConnections) {
        super();
        pcf = new PooledConnectionFactory(connectString);
        pcf.setMaxConnections(maxConnections);
        pcf.start();
    }

    public void stop() {
        pcf.stop();
    }

    @Override
    public Connection borrowConnection() throws Exception {
        Connection answer = pcf.createConnection();
        answer.start();
        return answer;
    }

    @Override
    public Connection borrowConnection(long timeout) throws Exception {
        // SNIPPED...
    }

    @Override
    public void returnConnection(Connection connection) throws Exception {
        // Do nothing since there isn't a way to return a Connection
        // to the instance of PooledConnectionFactory
        log.info("Connection returned");
    }
}
```

次に、**ConnectionResource** を **Sjms2Component** に渡します。

```
CamelContext camelContext = new DefaultCamelContext();
AMQConnectionResource pool = new AMQConnectionResource("tcp://localhost:33333", 1);
Sjms2Component component = new Sjms2Component();
component.setConnectionResource(pool);
camelContext.addComponent("sjms2", component);
```

完全な使用例を確認するには、**ConnectionResourceIT** を参照してください。

286.5.2. セッション、コンシューマー、およびプロデューサーのプーリングとキャッシング管理

近日公開 ...

286.5.3. バッチメッセージのサポート

Sjms2Producer は、**List** をカプセル化する Exchange を作成することにより、メッセージのコレクションのパブリッシュをサポートします。この Sjms2Producer は、List の内容を繰り返し、各メッセージを個別にパブリッシュします。

メッセージのバッチを生成するときに、各メッセージに固有のヘッダーを設定する必要がある場合は、SJMS2 **BatchMessage** クラスを使用できます。Sjms2Producer が **BatchMessage** リストに遭遇すると、各 **BatchMessage** を繰り返し、含まれているペイロードとヘッダーを公開します。

以下は、BatchMessage クラスの使用例です。まず、**BatchMessage** のリストを作成します:

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

次に、リストを公開します。

```
template.sendBody("sjms2:queue:batch.queue", messages);
```

286.5.4. カスタマイズ可能なトランザクションコミットストラテジー (ローカル JMS トランザクションのみ)

SJMS2 は、**TransactionCommitStrategy** インターフェイスを使用して、カスタムでプラグ可能なトランザクションストラテジーを作成する手段を開発者に提供します。これにより、ユーザーは、**SessionTransactionSynchronization** が セッションをいつコミットするかを決定するために使用する一連の固有の状況を定義できます。その使用例は、次のセクションで詳しく説明する **BatchTransactionCommitStrategy** です。

286.5.5. トランザクションバッチのコンシューマーとプロデューサー

SJMS2 コンポーネントは、プロデューサーエンドポイントとコンシューマーエンドポイントの両方でローカル JMS トランザクションのバッチ処理をサポートするように設計されています。ただし、それらがそれぞれでどのように処理されるかは非常に異なります。

SJMS2 コンシューマーエンドポイントは、X メッセージを関連するセッションでコミットする前に処理する単純な実装です。コンシューマーでバッチ処理されたトランザクションを有効にするには、まず、**transacted** パラメーターを true に設定してトランザクションを有効にし、次に **transactionBatchCount** を追加して 0 より大きい任意の値に設定します。たとえば、次の設定では、10 メッセージごとにセッションがコミットされます。

```
sjms2:queue:transacted.batch.consumer?transacted=true&transactionBatchCount=10
```

コンシューマーエンドポイントでのバッチの処理中に例外が発生した場合、セッションロールバックが呼び出され、メッセージが次に使用可能なコンシューマーに再配信されます。関連するセッションの **BatchTransactionCommitStrategy** のカウンターも 0 にリセットされます。JMSRedelivered ヘッ

ダーが true に設定されたメッセージを監視するために、バッチメッセージのプロセッサーにフックを配置することは、ユーザーの責任です。これは、メッセージがある時点でロールバックされたこと、および処理が成功したことを確認する必要があることを示しています。

トランザクション処理されたバッチコンシューマーには、セッションで開いているトランザクションをコミットする前にメッセージ間で既定の時間 (5000 ミリ秒) 待機する内部タイマーのインスタンスも含まれます。デフォルト値の 5000 ミリ秒 (最小 1000 ミリ秒) は、ほとんどのユースケースに適していますが、さらに調整が必要な場合は、単に **transactionBatchTimeout** パラメーターを設定してください。

```
sjms2:queue:transacted.batch.consumer?
transacted=true&transactionBatchCount=10&transactionBatchTimeout=2000
```

受け入れられる最小値は 1000 ミリ秒です。これは、コンテキスト切り替えの量が不要なパフォーマンスへの影響をもたらし、メリットが得られない可能性があるためです。

ただし、プロデューサーエンドポイントの処理方法は大きく異なります。プロデューサでは、各メッセージが宛先に配信された後、Exchange が閉じられ、そのメッセージへの参照がなくなります。再配信可能なすべてのメッセージを利用できるようにするには、BatchMessage をパブリッシュしているプロデューサーエンドポイントでトランザクションを有効にするだけです。トランザクションは、バッチリスト内のすべてのメッセージを含む交換の最後にコミットされます。追加の設定は必要ありません。以下に例を示します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

トランザクションを有効にしてリストを公開します。

```
template.sendBody("sjms2:queue:batch.queue?transacted=true", messages);
```

286.6. 追記

286.6.1. メッセージヘッダーの形式

SJMS2 コンポーネントは、Camel JMS コンポーネントで使用されるのと同じヘッダー形式ストラテジーを使用します。このプラグ可能な戦略により、ネットワーク経由で送信されるメッセージが JMS メッセージ仕様に準拠することが保証されます。

exchange.in.header の場合、次のルールがヘッダーキーに適用されます。

- **JMS** または **JMSX** で始まるキーは予約されています。
- **exchange.in.headers** キーはリテラルで、すべて有効な Java 識別子である必要があります (キー名にドットを使用しないでください)。
- Camel は、JMS メッセージを消費するときにドットとハイフンを置き換え、その逆を行います。
    - Camel がメッセージを消費するときは、**DOT** と逆の置換に置き換えられます。

- Camel がメッセージを消費するときは、**HYPHEN** と逆の置換に置き換えられます。オプション **jmsKeyFormatStrategy** も参照してください。これにより、キーのフォーマットに独自のカスタム戦略を使用できます。

exchange.in.header の場合、次のルールがヘッダー値に適用されます。

286.6.2. メッセージ内容

ネットワーク経由でコンテンツを配信するには、配信されるメッセージの本文が JMS メッセージ仕様に準拠していることを確認する必要があります。したがって、生成されるものはすべて、プリミティブまたはそのカウンターオブジェクト (**Integer**、**Long**、**Character** など) のいずれかでなければなりません。タイプ **String**、**CharSequence**、**Date**、**BigDecimal**、および **BigInteger** はすべて、それらの **toString()** 表現に変換されます。他のすべてのタイプはドロップされます。

286.6.3. クラスタリング

クラスター化された環境で SJMS2 と一緒に **InOut** を使用する場合、TemporaryQueue 宛先を使用するか、InOut プロデューサエンドポイントごとに宛先への一意の名前付き応答を使用する必要があります。メッセージ相関は、ブローカーのメッセージセレクターではなく、エンドポイントによって処理されます。InOut プロデューサエンドポイントは、メッセージ **JMSCorrelationID** によってキャッシュされた Java Concurrency Exchangers を使用します。これにより、パフォーマンスが大幅に向上し、ブローカーのオーバーヘッドが削減されます。これは、すべてのメッセージが、関心のあるコンシューマーによって生成された順序で宛先から消費されるためです。

現在、唯一の相関ストラテジーは **JMSCorrelationId** を使用することです。**InOut** Consumer はこのストラテジーを使用し、含まれている **JMSReplyTo** 宛先へのすべてのレスポンスメッセージにも、リクエストからコピーされた **JMSCorrelationId** が含まれるようにします。

286.7. トランザクションサポート

SJMS2 は現在、内部 JMS トランザクションの使用のみをサポートしています。Camel Transaction Processor または Java Transaction API (JTA) はサポートされていません。

286.7.1. Springless とは Spring を使用できないということか

そうではまったくありません。以下は、Spring DSL を使用した SJMS2 コンポーネントの例です。

```
<route
    id="inout.named.reply.to.producer.route">
    <from
        uri="direct:invoke.named.reply.to.queue" />
    <to
        uri="sjms2:queue:named.reply.to.queue?
namedReplyTo=my.response.queue&amp;exchangePattern=InOut" />
</route>
```

Springless とは、Spring JMS API への依存関係から離れることを指します。SJMS2 を強化するために、新しい JMS クライアント API がゼロから開発されています。

第287章 SLACK コンポーネント

Camel バージョン 2.16 以降で利用可能

slack コンポーネントを使用すると、Slack のインスタンスに接続し、事前に確立された Slack 着信 Webhook を介してメッセージボディーに含まれるメッセージを配信できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-slack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

287.1. URI 形式

チャネルにメッセージを送信します。

```
slack:#channel[?options]
```

slackuser にダイレクトメッセージを送信するには。

```
slack:@username[?options]
```

287.2. オプション

Slack コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
webhookUrl (producer)	着信 Webhook URL		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Slack エンドポイントは、URI 構文を使用して設定されます。

```
slack:channel
```

パスおよびクエリーパラメーターを使用します。

287.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
channel	**必須** メッセージをユーザーに直接送信するためのチャネル名 (構文名) または slackuser (構文 userName)。		文字列

287.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
iconEmoji (producer)	Slack 絵文字をアバターとして使用する		String
iconUrl (producer)	チャネルまたはユーザーにメッセージを送信するときにコンポーネントが使用するアバター。		String
username (producer)	これは、チャネルまたはユーザーにメッセージを送信するときにボットが持つユーザー名です。		String
webhookUrl (producer)	着信 Webhook URL		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

287.3. SLACKCOMPONENT

XML を使用する SlackComponent は、統合用の入力 Webhook URL をパラメーターとして含む Spring または Blueprint Bean として設定する必要があります。

```
<bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
    <property name="webhookUrl"
value="https://hooks.slack.com/services/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"/>
</bean>
```

Java の場合、Java コードを使用してこれを設定できます。

287.4. 例

ブループリントを使用した CamelContext は次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" default-activation="lazy">

    <bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
        <property name="webhookUrl"
```

```
value="https://hooks.slack.com/services/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"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="direct:test"/>
            <to uri="slack:#channel?iconEmoji=:camel:&amp;username=CamelTest"/>
        </route>
    </camelContext>

</blueprint>
```

287.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第288章 SMPP コンポーネント

Camel バージョン 2.2 以降で利用可能

このコンポーネントは、SMPP プロトコルを介して SMSC (ショートメッセージサービスセンター) へのアクセスを提供し、SMS を送受信します。プロトコルの実装には JSMPP ライブラリーが使用されます。

Camel コンポーネントは現在、SMSC 自体としてではなく、ESME (External Short Messaging Entity) として動作します。

Camel 2.9 以降では、ReplaceSm、QuerySm、SubmitMulti、CancelSm、および DataSm も実行できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-smpp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

288.1. SMS の制限

SMS は信頼性も安全性もありません。 信頼性の高い安全な配信を必要とするユーザーは、代わりに XMPP または SIP コンポーネントの使用を検討し、選択したプロトコルをサポートするスマートフォンアプリと組み合わせて使用することを検討してください。

- 信頼性: SMPP 規格は、エラー、未配信、および配信の確認を示すさまざまなフィードバックメカニズムを提供していますが、モバイルネットワークがこれらの応答を隠したりシミュレートしたりすることは珍しくありません。 たとえば、一部のネットワークでは、宛先番号が無効であるかスイッチがオンになっていない場合でも、メッセージごとに配信確認が自動的に送信されます。 一部のネットワークは、メッセージがスパムであると判断した場合、メッセージを黙ってドロップします。 ネットワーク内のスパム検出ルールは非常に粗雑で、1 人の送信者からの 1 日あたり 100 件を超えるメッセージがスパムと見なされる場合があります。

- セキュリティー: 電波塔から受信者のハンドセットまでのラストホップには、基本的な暗号化があります。 SMS メッセージは、ネットワークの他の部分では暗号化も認証もされません。 一部の通信事業者は、小売店やコールセンターのスタッフが顧客の SMS メッセージ履歴を閲覧できるようにしています。 メッセージ送信者の ID は簡単に偽造できます。 規制当局や携帯電話業界自体でさえ、2 要素認証方式やその他のセキュリティーが重要な目的での SMS の使用に対して警告を発しています。

Camel コンポーネントは SMS ネットワークへのメッセージの送信を可能な限り簡単にしますが、これらの問題を簡単に解決することはできません。

288.2. データコーディング、アルファベットおよび国際文字セット

データコーディングとアルファベットは、メッセージごとに指定できます。 エンドポイントにはデフォルト値を指定できます。 これらのオプションの関係と、複数の値が設定されている場合のコンポーネントの動作を理解することが重要です。

データコーディングは、SMPP ワイヤフォーマットの 8 ビットフィールドです。

アルファベットは、データコーディングフィールドのビット 0 ~ 3 に対応します。 一部のタイプのメッセージでは、(データコーディングフィールドのビット 5 を設定することによって) メッセージクラスが使用され、データコーディングフィールドの下位 2 ビットはアルファベットとして解釈されず、ビット 2 と 3 のみがアルファベットに影響します。

さらに、現在のバージョンの JSMPP ライブラリーは、ビット 0 と 1 がメッセージクラスに使用されると仮定すると、ビット 2 と 3 のみをサポートしているようです。 これが、JSMPP の Alphabet クラスが、ISO-8859-1 を示す値 3 (バイナリー 0011) をサポートしていない理由です。

JSMPP はメッセージクラスパラメーターの表現を提供しますが、Camel コンポーネントは現在、データコーディングフィールドの対応するビットを手動で設定する以外に、それを設定する方法を提供していません。

送信メッセージでデータコーディングフィールドを設定する場合、Camel コンポーネントは次の値を考慮し、最初に見つかった値を使用します。

- ヘッダーで指定されたデータコーディング
- ヘッダーで指定されたアルファベット
- エンドポイント設定で指定されたデータコーディング (URI パラメーター)

Camel の古いバージョンには、国際文字セットのサポートにバグがありました。 この機能は、すべてのメッセージに単一のエンコーディングが使用されている場合にのみ機能し、ユーザーがメッセージごとにエンコーディングを変更したい場合は面倒でした。 これを機能させる必要があるユーザーは、Camel のバージョンに次の修正が含まれていることを確認する必要があります。

JIRA Issues Macro: com.atlassian.sal.api.net.ResponseStatusException: Unexpected response received. ステータスコード: 404

をクリックします。

Camel コンポーネントは、データコーディング値を SMSC に送信しようとするだけでなく、メッセージボディーを分析し、それを Java 文字列 (Unicode) に変換し、それを対応するアルファベットのバイト配列に変換しようとします。バイト配列で使用すると、Camel SMPP コンポーネントはデータコーディング値 (ヘッダーまたは設定) を考慮せず、指定されたアルファベットのみを考慮します (ヘッダーまたはエンドポイントパラメーターのいずれかから)。

String 内の一部の文字が選択したアルファベットで表現できない場合は、疑問符 (?) 記号に置き換えることができます。 API のユーザーは、メッセージ本文をコンポーネントに渡す前に ISO-8859-1 に変換できるかどうかを確認し、変換できない場合はアルファベットヘッダーを設定して UCS-2 エンコーディングをリクエストすることを検討することをお勧めします。 アルファベットとデータコーディングオプションがまったく指定されていない場合、コンポーネントは必要なエンコーディングを検出し、データコーディングを設定しようとする場合があります。

アルファベットコードのリストは、SMPP 仕様 v3.4 のセクション 5.2.19 で指定されています。 SMPP 仕様の注目すべき制限の 1 つは、GSM 3.38 (7 ビット) 文字セットの使用を明示的に要求するためのアルファベットコードがないことです。 アルファベットに値 0 を選択すると、SMSC の **デフォルト** のアルファベットが選択されます。これは通常、GSM 3.38 を意味しますが、保証されていません。 SMPP ゲートウェイの Nexmo では、コントロールパネルオプションを使用して、デフォルトを他の文字セットにマッピングできます。SMSC オペレータに問い合わせて、デフォルトとして使用されている文字セットを正確に確認することをお勧めします。

288.3. メッセージの分割とスロットリング

メッセージ本文を文字列からバイト配列に変換した後、Camel コンポーネントは、メッセージを JSMPP に渡す前に (140 バイトの SMS サイズ制限内で) 部分に分割する役割も果たします。 これは自動的に完了します。

GSM 3.38 アルファベットが使用されている場合、コンポーネントは最大 160 文字を 140 バイトのメッセージボディーにパックします。 8 ビット文字セットが使用されている場合 (西ヨーロッパの ISO-8859-1 など)、140 バイトのメッセージボディー内で 140 文字が許可されます。 16 ビットの UCS-2 エンコーディングが使用されている場合、140 バイトの各メッセージには 70 文字しか収まりません。

一部の SMSC プロバイダーは、スロットリングルールを実装しています。 分割されたメッセージの各部分は、プロバイダーのスロットリングメカニズムによって個別にカウントされる場合があります。 Camel Throttler コンポーネントは、SMSC に渡す前に SMPP ルートでメッセージをスロットリングするのに役立ちます。

288.4. URI 形式

```
smpp://[username@]hostname[:port][?options]
smpps://[username@]hostname[:port][?options]
```

ユーザー名 が指定されていない場合、Camel はデフォルト値 **smppclient** を提供します。
ポート 番号が指定されていない場合、Camel はデフォルト値 **2775** を提供します。
Camel 2.3: プロトコル名が smpps の場合、camel-smpp は SSLSocket を使用してサーバーへの接続を開始しようとします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

288.5. URI オプション

SMPP コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 SmppConfiguration を設定として使用する場合。		SmppConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SMPP エンドポイントは、URI 構文を使用して設定されます。

```
smpp:host:port
```

パスおよびクエリーパラメーターを使用します。

288.5.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	SMSC サーバーが使用するホスト名。	localhost	String
port	SMSC サーバーが使用するポート番号。	2775	Integer

288.5.2. クエリーパラメーター (38 パラメーター)

名前	説明	デフォルト	タイプ
initialReconnectDelay (common)	接続が失われた後、コンシューマー/プロデューサーが SMSC への再接続を試行した後の初期遅延をミリ秒単位で定義します。	5000	long
maxReconnect (common)	SMSC が否定的なバインドレスポンスを返した場合に、SMSC への再接続を試行する最大回数を定義します。	2147483647	int
reconnectDelay (common)	SMSC への接続が失われ、前の接続が成功しなかった場合の再接続の試行間隔をミリ秒単位で定義します。	5000	long
splittingPolicy (common)	長いメッセージを処理するためのポリシーを指定できます。ALLOW - デフォルトでは、長いメッセージはメッセージごとに 140 バイトに分割されます TRUNCATE - 長いメッセージは分割され、最初のフラグメントのみが SMSC に送信されます。一部の通信事業者は後続のフラグメントをドロップするため、これにより、決して配信されないメッセージの一部を送信する SMPP 接続の負荷が軽減されます。REJECT - メッセージを分割する必要がある場合、メッセージは SMPP NegativeResponseException で拒否され、メッセージを示す理由コードが長すぎます。	ALLOW	SmppSplittingPolicy
systemType (common)	このパラメーターは、SMSC にバインドされている ESME (外部ショートメッセージエンティティー) のタイプを分類するために使用されます (最大 13 文字)。	cp	String
addressRange (consumer)	SMPP 3.4 仕様のセクション 5.2.7 で定義されているように、SmppConsumer のアドレス範囲を指定できます。SmppConsumer は、この範囲内のアドレス (MSISDN または IP アドレス) を対象とする SMSC からのみメッセージを受信します。		String

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
destAddr (producer)	宛先 SME アドレスを定義します。モバイル終了メッセージの場合、これは受信者 MS のディレクトリー番号です。SubmitSm、SubmitMulti、CancelSm、DataSm のみ。	1717	String
destAddrNpi (producer)	SME 宛先アドレスパラメーターで使用される番号のタイプ (TON) を定義します。SubmitSm、SubmitMulti、CancelSm、DataSm のみ。次の NPI 値が定義されています。0: Unknown 1: ISDN (E163/E164) 2: Data (X.121) 3: Telex (F.69) 6: Land Mobile (E.212) 8: National 9: Private 10 : ERMES 13: Internet (IP) 18: WAP Client ID (WAP フォーラムで定義)		byte
destAddrTon (producer)	SME 宛先アドレスパラメーターで使用される番号のタイプ (TON) を定義します。SubmitSm、SubmitMulti、CancelSm、DataSm のみ。次の TON 値が定義されています。0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte
lazySessionCreation (producer)	Camel プロデューサーの開始時に SMSC が使用できない場合は、例外を回避するためにセッションを遅延作成できます。Camel は、最初のエクスチェンジのメッセージヘッダー 'CamelSmppSystemId' と 'CamelSmppPassword' をチェックします。それらが存在する場合、Camel はこれらのデータを使用して SMSC に接続します。	false	boolean

名前	説明	デフォルト	タイプ
numberingPlanIndicator (producer)	SME で使用される数値計画指標 (NPI) を定義します。次の NPI 値が定義されています。0: Unknown 1: ISDN (E163/E164) 2: Data (X.121) 3: Telex (F.69) 6: Land Mobile (E.212) 8: National 9: Private 10 : ERMES 13: Internet (IP) 18: WAP Client ID (WAP フォーラムで定義)		byte
priorityFlag (producer)	発信元 SME がショートメッセージに優先度レベルを割り当てることができるようにします。SubmitSm および SubmitMulti のみ。4 つの優先度レベルがサポートされています。0: レベル 0 (最低) 優先度 1: レベル 1 優先度 2: レベル 2 優先度 3: レベル 3 (最高) 優先度		byte
protocolId (producer)	プロトコル ID		byte
registeredDelivery (producer)	SMSC 配信受領書や SME からの確認応答をリクエストするために使用されます。以下の値が定義されています。0: SMSC 配信受領書はリクエストされていません。1: 最終的な配信結果が成功または失敗である場合にリクエストされた SMSC 配信受信。2: 最終的な配信結果が配信失敗である場合にリクエストされた SMSC 配信受信。		byte
replaceIfPresentFlag (producer)	SMSC に対して、以前に送信された、まだ配信が保留されているメッセージを置き換えるようにリクエストするために使用されます。SMSC は、送信元アドレス、宛先アドレス、およびサービスタイプが新しいメッセージの同じフィールドと一致する場合、既存のメッセージを置き換えます。存在フラグ値が定義されている場合、次の置換: 0: 置換しない 1: 置換する		byte
serviceType (producer)	サービスタイプパラメーターを使用して、メッセージに関連付けられた SMS アプリケーションサービスを示すことができます。次の一般的な service_types が定義されています。CMT: Cellular Messaging CPT: Cellular Paging VMN: Voice Mail Notification VMA: Voice Mail Alerting WAP: Wireless Application Protocol USSD: Unstructured Supplementary Services Data	CMT	String
sourceAddr (producer)	このメッセージを発信した SME (Short Message Entity) のアドレスを定義します。	1616	String

名前	説明	デフォルト	タイプ
sourceAddrNpi (producer)	SME 発信元アドレスパラメーターで使用される数値計画インジケータ (NPI) を定義します。次の NPI 値が定義されています。0: Unknown 1: ISDN (E163/E164) 2: Data (X.121) 3: Telex (F.69) 6: Land Mobile (E.212) 8: National 9: Private 10 : ERMES 13: Internet (IP) 18: WAP Client ID (WAP フォーラムで定義)		byte
sourceAddrTon (producer)	SME 発信元アドレスパラメーターで使用される番号のタイプ (TON) を定義します。次の TON 値が定義されています。0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte
typeOfNumber (producer)	SME で使用される番号のタイプ (TON) を定義します。次の TON 値が定義されています。0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte
enquireLinkTimer (advanced)	信頼性チェックの間隔をミリ秒単位で定義します。信頼度チェックは、ESME と SMSC の間の通信パスをテストするために使用されます。	5000	Integer
sessionStateListener (advanced)	レジストリーの org.jsmpp.session.SessionStateListener を参照して、セッション状態が変化したときにコールバックを受け取ることができます。		SessionStateListener
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transactionTimer (advanced)	SMPP エンティティーがセッションがアクティブでないと見なすまでの、トランザクション後に許可される非アクティブの最大期間を定義します。このタイマーは、通信中の SMPP エンティティー (つまり、SMSC または ESME) のいずれかでアクティブになる場合があります。	10000	Integer
alphabet (codec)	SMPP 3.4 仕様のセクション 5.2.19 に従って、データのエンコードを定義します。0: SMSC デフォルトのアルファベット 4: 8 ビットのアルファベット 8: UCS2 のアルファベット		byte

名前	説明	デフォルト	タイプ
dataCoding (codec)	SMPP 3.4 仕様のセクション 5.2.19 に従ってデータコーディングを定義します。データエンコーディングの例: 0: SMSC デフォルトアルファベット 3: Latin 1 (ISO-8859-1) 4: オクテット未指定 (8 ビットバイナリー) 8: UCS2 (ISO/IEC-10646) 13: 拡張漢字 JIS(X 0212- 1990)		byte
encoding (codec)	ショートメッセージのユーザーデータのエンコード方式を定義します。SubmitSm、ReplaceSm、および SubmitMulti のみ。	ISO-8859-1	String
httpProxyHost (proxy)	HTTP プロキシーを介して SMPP をトンネリングする必要がある場合は、この属性を HTTP プロキシーのホスト名または IP アドレスに設定します。		String
httpProxyPassword (proxy)	HTTP プロキシーに Basic 認証が必要な場合は、この属性を HTTP プロキシーに必要なパスワードに設定します。		String
httpProxyPort (proxy)	HTTP プロキシーを介して SMPP をトンネリングする必要がある場合は、この属性を HTTP プロキシーのポートに設定します。	3128	Integer
httpProxyUsername (proxy)	HTTP プロキシーに Basic 認証が必要な場合は、この属性を HTTP プロキシーに必要なユーザー名に設定します。		String
proxyHeaders (proxy)	これらのヘッダーは、接続の確立中にプロキシーサーバーに渡されます。		Map
password (security)	SMSC サーバーに接続するためのパスワード。		String
systemId (security)	SMSC サーバーに接続するためのシステム ID (ユーザー名)。	smppclient	String
usingSSL (security)	smpps プロトコルで SSL を使用するかどうか	false	boolean

これらのオプションはいくつでも使用できます。

```
smpp://smppclient@localhost:2775?
password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consumer
```

288.6. プロデューサーメッセージヘッダー

次のメッセージヘッダーを使用して、SMPP プロデューサーの動作に影響を与えることができます。

ヘッダー	タイプ	説明
CamelSmppDestAddr	**List/String**	**SubmitSm、SubmitMulti、CancelSm、および DataSm のみ** 宛先の SME アドレスを定義します。モバイル終了メッセージの場合、これは受信者 MS のディレクトリー番号です。SubmitMulti の場合は **List<String>** である必要があり、それ以外の場合は **String** である必要があります。
CamelSmppDestAddrTon	**Byte**	**SubmitSm、SubmitMulti、CancelSm、および DataSm のみ** SME 宛先アドレスパラメーターで使用される番号のタイプ (TON) を定義します。上記で定義された **sourceAddrTon** URI オプション値を使用します。
CamelSmppDestAddrNpi	**Byte**	**SubmitSm、SubmitMulti、CancelSm、および DataSm のみ** SME 宛先アドレスパラメーターで使用される数値計画インジケータ (NPI) を定義します。上記の URI オプション **sourceAddrNpi** の値を使用します。
CamelSmppSourceAddr	**String**	このメッセージを発信した SME (Short Message Entity) のアドレスを定義します。
CamelSmppSourceAddrTon	**Byte**	SME 発信元アドレスパラメーターで使用される番号のタイプ (TON) を定義します。上記で定義された **sourceAddrTon** URI オプション値を使用します。
CamelSmppSourceAddrNpi	**Byte**	SME 発信元アドレスパラメーターで使用される数値計画インジケータ (NPI) を定義します。上記の URI オプション **sourceAddrNpi** の値を使用します。
CamelSmppServiceType	**String**	サービスタイプパラメーターを使用して、メッセージに関連付けられた SMS アプリケーションサービスを示すことができます。上記の URI オプションの **serviceType** 設定を使用します。
CamelSmppRegisteredDelivery	**Byte**	**SubmitSm、ReplaceSm、SubmitMulti、および DataSm のみ SMSC 配信受領書および/または SME からの確認応答をリクエストするために** 使用されます。上記の URI オプション **registeredDelivery** 設定を使用します。

ヘッダー	タイプ	説明
CamelSmppPriorityFlag	**Byte**	**SubmitSm および SubmitMulti のみ** 発信元 SME がショートメッセージに優先度レベルを割り当てることを許可します。上記の URI オプションの **priorityFlag** 設定を使用します。
CamelSmppScheduleDeliveryTime	**Date**	**SubmitSm、SubmitMulti、および ReplaceSm のみ。** このパラメーターは、メッセージ配信を最初に試行する予定時刻を指定します。これは、SMSC がこのメッセージの配信を試みる絶対日時、または現在の SMSC 時刻からの相対時刻のいずれかを定義します。絶対時間形式または相対時間形式で指定できます。時刻形式のエンコーディングは、smpp 仕様 v3.4 の 7.1.1 章に規定されています。
CamelSmppValidityPeriod	**String/Date**	**SubmitSm、SubmitMulti、および ReplaceSm の場合のみ** 有効期間パラメーターは、SMSC の有効期限を示します。その後、宛先に配信されない場合、メッセージは破棄されます。**Date** として指定されている場合は、絶対時間として解釈されます。**Camel 2.9.1 以降:** smpp 仕様 v3.4 の 7.1.1 章で指定されているように**String** として指定すると、絶対時間形式または相対時間形式で定義できます。
CamelSmppReplaceIfPresentFlag	**Byte**	**SubmitSm および SubmitMulti の場合のみ** replace if present フラグパラメーターは、SMSC に対して、以前に送信された、まだ配信が保留されているメッセージを置き換えるように要求するために使用されます。SMSC は、送信元アドレス、宛先アドレス、およびサービスタイプが新しいメッセージの同じフィールドと一致する場合、既存のメッセージを置き換えます。次の値が定義されています: **0**、置換しない、および **1**、置換する
CamelSmppAlphabet/CamelSmppDataCoding	**Byte**	Camel 2.5 SubmitSm、SubmitMulti、および ReplaceSm 用 (Camel 2.9 より前では、**CamelSmppAlphabet** の代わりに **CamelSmppDataCoding を** 使用します。) SMPP 3.4 仕様、セクション 5.2.19 に従ったデータコーディング。上記の URI オプションの **alphabet** 設定を使用します。
CamelSmppOptionalParameters	**Map<String, String>**	**非推奨であり、Camel 2.13.0/3.0.0 で削除されます** **Camel 2.10.5 および 2.11.1 以降、SubmitSm、SubmitMulti、および DataSm のみ** オプションのパラメーターは SMSC によって送り返されます。
CamelSmppOptionalParameter	**Map<Short, Object>**	**Camel 2.10.7 および 2.11.2 以降、SubmitSm、SubmitMulti、および DataSm のみ** SMSC に送信されるオプションのパラメーター。値は次のように変換されます: **String → org.jsmpp.bean.OptionalParameter.COctetString**, **byte[] → org.jsmpp.bean.OptionalParameter.OctetString**, **Byte → org.jsmpp.bean.OptionalParameter.Byte**, **Integer → org.jsmpp.bean.OptionalParameter.Int**, **Short → org.jsmpp.bean.OptionalParameter.Short**, **null → org.jsmpp.bean.OptionalParameter.Null**

ヘッダー	タイプ	説明
CamelSmppEncoding	String	**Camel 2.14.1 および Camel 2.15.0 以降、** *SubmitSm、SubmitMulti、および DataSm のみ*。 メッセージボディーのバイトのエンコード (文字セット名) を指定します。 メッセージボディーが文字列の場合、Java 文字列は常に Unicode であるため、これは関係ありません。 ボディーがバイト配列の場合、このヘッダーを使用して、それが ISO-8859-1 またはその他の値であることを示すことができます。 デフォルト値は、エンドポイント設定パラメーターの **encoding** によって指定されます
CamelSmppSplittingPolicy	String	**Camel 2.14.1 および Camel 2.15.0 以降、** *SubmitSm、SubmitMulti、および DataSm のみ*。 このエクスチェンジのメッセージ分割のポリシーを指定します。 設定可能な値は、エンドポイント設定パラメーター **splittingPolicy** で説明されています

次のメッセージヘッダーは、SMSC からのレスポンスをメッセージヘッダーに設定するために SMPP プロデューサーによって使用されます。

ヘッダー	タイプ	説明
CamelSmppId	**List<String>/String**	後で使用するために送信されたショートメッセージを識別するための ID。**Camel 2.9.0 から**: ReplaceSm、QuerySm、CancelSm、および DataSm の場合、このヘッダー値は **String** です。SubmitSm または SubmitMultiSm の場合、このヘッダー値は **List<String>** です。
CamelSmppSentMessageCount	**Integer**	**Camel 2.9 以降では SubmitSm および SubmitMultiSm のみ** 送信されたメッセージの総数。
CamelSmppError	**Map<String, List<Map<String, Object>>>**	**Camel 2.9 以降では SubmitMultiSm のみ** ショートメッセージを **Map<String, List<Map<String, Object>>>** (messageID : (destAddr : address, error : errorCode)) の形式で送信することによって発生したエラー。
CamelSmppOptionalParameters	**Map<String, String>**	**非推奨で、Camel 2.13.0/3.0.0 で削除されます Camel 2.11.1 以降からは DataSm のみ** メッセージを送信することによって SMSC から返されるオプションのパラメーター。

ヘッダー	タイプ	説明
CamelSmppOptionalParameter	**Map<Short, Object>**	Camel 2.10.7、2.11.2 **以降では DataSm のみ** メッセージを送信することによって SMSC から返されるオプションのパラメーター。キーは、オプションパラメーターの **Short** コードです。値は次のように変換されます: **org.jsmpp.bean.OptionalParameter.COctetString → String、org.jsmpp.bean.OptionalParameter.OctetString → byte[]、org.jsmpp.bean.OptionalParameter.Byte → Byte、org.jsmpp.bean.OptionalParameter.Int → Integer、org.jsmpp.bean.OptionalParameter.Short → Short、org.jsmpp.bean.OptionalParameter.Null →null**

288.7. コンシューマーメッセージヘッダー

次のメッセージヘッダーは、SMSC からのリクエストデータをメッセージヘッダーに設定するために SMPP コンシューマーによって使用されます。

ヘッダー	タイプ	説明
CamelSmppSequenceNumber	**Integer**	**AlertNotification、DeliverSm、および DataSm のみシーケンス番号により、** レスポンス PDU をリクエスト PDU と関連付けることができます。関連する SMPP レスポンス PDU は、このフィールドを保持する必要があります。
CamelSmppCommandId	**Integer**	**AlertNotification、DeliverSm、および DataSm のみ** コマンド ID フィールドは、特定の SMPP PDU を識別します。定義された値の完全なリストについては、smpp 仕様 v3.4 の 5.1.2.1 章を参照してください。
CamelSmppSourceAddr	**String**	**AlertNotification、DeliverSm、および DataSm のみ** このメッセージを発信した SME (Short Message Entity) のアドレスを定義します。
CamelSmppSourceAddrNpi	**Byte**	**AlertNotification および DataSm のみ** SME 発信元アドレスパラメーターで使用される数値計画インジケータ (NPI) を定義します。上記の URI オプション **sourceAddrNpi** の値を使用します。
CamelSmppSourceAddrTon	**Byte**	**AlertNotification および DataSm の場合のみ** SME 発信元アドレスパラメーターで使用される番号のタイプ (TON) を定義します。上記で定義された **sourceAddrTon** URI オプション値を使用します。

ヘッダー	タイプ	説明
CamelSmppEsmeAddr	**String**	**AlertNotification の場合のみ** 宛先 ESME アドレスを定義します。モバイル終了メッセージの場合、これは受信者 MS のディレクトリー番号です。
CamelSmppEsmeAddrNpi	**Byte**	**AlertNotification の場合のみ** ESME 発信元アドレスパラメーターで使用される数値計画インジケータ (NPI) を定義します。上記の URI オプション **sourceAddrNpi** の値を使用します。
CamelSmppEsmeAddrTon	**Byte**	**AlertNotification の場合のみ** ESME 発信元アドレスパラメーターで使用される番号のタイプ (TON) を定義します。上記で定義された **sourceAddrTon** URI オプション値を使用します。
CamelSmppId	**String**	**smsc DeliveryReceipt および DataSm の場合のみ** 最初に送信されたときに SMSC によってメッセージに割り当てられたメッセージ ID。
CamelSmppDelivered	**Integer**	**smsc DeliveryReceipt のみ** 配信されたショートメッセージの数。これは、元のメッセージが配布リストに送信された場合にのみ関係します。必要に応じて、値の先頭にゼロが埋め込まれます。
CamelSmppDoneDate	**Date**	**smsc DeliveryReceipt のみ** ショートメッセージが最終状態に達した日時。形式は次のとおりです。YYMMDDhhmm。
CamelSmppStatus	**DeliveryReceiptState**	**smsc DeliveryReceipt のみ:** メッセージの最終ステータス。次の値が定義されています: **DELIVRD**: メッセージは宛先に配信されます **EXPIRED**: メッセージの有効期限が切れました **DELETED**: メッセージは削除されました **UNDELIV**: メッセージは配信不能です **ACCEPTD**: メッセージは受け入れられた状態です (つまり、カスタマーサービスがサブスクライバーに代わりに手動で読み取りました) **UNKNOWN**: メッセージが無効な状態、**REJECTD**: メッセージが拒否された状態
CamelSmppCommandStatus	**Integer**	**DataSm のみ** メッセージのコマンドステータス。

ヘッダー	タイプ	説明
CamelSmppError	**String**	**smsc DeliveryReceipt の場合のみ** 必要に応じて、ネットワーク固有のエラーコードまたはメッセージの配信試行の SMSC エラーコードを保持できます。これらのエラーはネットワークまたは SMSC 固有のものであり、ここには含まれていません。
CamelSmppSubmitDate	**Date**	**smsc DeliveryReceipt のみ** ショートメッセージが送信された日時。置き換えられたメッセージの場合、これは元のメッセージが置き換えられた日付です。形式は次のとおりです。YYMMDDhhmm。
CamelSmppSubmitted	**Integer**	**smsc DeliveryReceipt のみ** 最初に送信されたショートメッセージの数。これは、元のメッセージが配布リストに送信された場合にのみ関係します。必要に応じて、値の先頭にゼロが埋め込まれます。
CamelSmppDestAddr	**String**	**DeliverSm および DataSm のみ:** 宛先の SME アドレスを定義します。モバイル終了メッセージの場合、これは受信者 MS のディレクトリー番号です。
CamelSmppScheduleDeliveryTime	**String**	**DeliverSm のみ:** このパラメーターは、メッセージ配信を最初に試行する予定時刻を指定します。これは、SMSC がこのメッセージの配信を試みる絶対日時、または現在の SMSC 時刻からの相対時刻のいずれかを定義します。絶対時間形式または相対時間形式で指定できます。時刻形式のエンコーディングは、smpp 仕様 v3.4 のセクション 7.1.1 で規定されています。
CamelSmppValidityPeriod	**String**	**DeliverSm のみ** 有効期間パラメーターは、SMSC の有効期限を示します。その後、宛先に配信されない場合、メッセージは破棄されます。絶対時間形式または相対時間形式で定義できます。絶対時間形式と相対時間形式のエンコードは、smpp 仕様 v3.4 のセクション 7.1.1 で指定されています。
CamelSmppServiceType	**String**	**DeliverSm および DataSm のみ** サービスタイプパラメーターは、メッセージに関連付けられた SMS アプリケーションサービスを示します。
CamelSmppRegisteredDelivery	**Byte**	**DataSm のみ** 配信受領書および/または SME からの確認応答をリクエストするために使用されます。上記のプロデューサーヘッダーリストと同じ値。

ヘッダー	タイプ	説明
CamelSmppDestAddrNpi	**Byte**	DataSm **のみ** 宛先アドレスパラメーターで数値計画インジケータ (NPI) を定義します。上記の URI オプション **sourceAddrNpi** の値を使用します。
CamelSmppDestAddrTon	**Byte**	DataSm **のみ** 宛先アドレスパラメーターで数値のタイプ (TON) を定義します。上記で定義された **sourceAddrTon** URI オプション値を使用します。
CamelSmppMessageType	**String**	Camel 2.6 **以降**: 受信メッセージのタイプを識別する: **AlertNotification**: SMSC のアラート通知、**DataSm**: SMSC のデータショートメッセージ、**DeliveryReceipt**: SMSC の配信レシート、**DeliverSm**: SMSC 配信ショートメッセージ。
CamelSmppOptionalParameters	**Map<String, Object>**	**非推奨であり、Camel 2.13.0/3.0.0 で削除されます。 Camel 2.10.5 以降で、DeliverSm の場合のみ** オプションのパラメーターは SMSC によって送り返されます。
CamelSmppOptionalParameter	**Map<Short, Object>**	Camel 2.10.7、2.11.2 **以降、DeliverSm のみ** オプションのパラメーターは SMSC によって送り返されます。キーは、オプションパラメーターの **Short** コードです。値は次のように変換されます: **org.jsmpp.bean.OptionalParameter.COctetString → String、org.jsmpp.bean.OptionalParameter.OctetString → byte[]、org.jsmpp.bean.OptionalParameter.Byte → Byte、org.jsmpp.bean.OptionalParameter.Int → Integer、org.jsmpp.bean.OptionalParameter.Short → Short、org.jsmpp.bean.OptionalParameter.Null → null**

ヒント

JSMPP ライブラリー 基礎となるライブラリーの詳細については、JSMPP ライブラリー のドキュメントを参照してください。

288.8. 例外処理

このコンポーネントは、一般的な Camel 例外処理機能をサポートしています。

SubmitSm (デフォルトアクション) でメッセージを送信する際にエラーが発生すると、ネストされた例外 org.jsmpp.extra.NegativeResponseException とともに org.apache.camel.component.smpp.SmppException が出力されます。NegativeResponseException.getCommandStatus() を呼び出して、正確な SMPP 否定応答コードを取得します。値については、SMPP 仕様 3.4、セクション 5.1.3 で説明されています。

Camel 2.8 以降: SMPP コンシューマーが **DeliverSm** または **DataSm** ショートメッセージを受信し、これらのメッセージの処理が失敗した場合、失敗を処理する代わりに **ProcessRequestException** を出力することもできます。この場合、この例外は基礎となる JSMPP ライブラリー に転送され、含まれているエラーコードが SMSC に返されます。この機能は、SMSC に後でショートメッセージを再送信するように指示する場合などに便利です。これは、次のコード行で実行できます。

```
from("smpp://smppclient@localhost:2775?
password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consumer")
  .doTry()
    .to("bean:dao?method=updateSmsState")
  .doCatch(Exception.class)
    .throwException(new ProcessRequestException("update of sms state failed", 100))
  .end();
```

エラーコードとその意味の完全なリストについては、SMPP 仕様 を参照してください。

288.9. サンプル

Java DSL を使用して SMS を送信するルート:

```
from("direct:start")
  .to("smpp://smppclient@localhost:2775?
      password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=producer");
```

Spring XML DSL を使用して SMS を送信するルート:

```
<route>
  <from uri="direct:start"/>
  <to uri="smpp://smppclient@localhost:2775?

password=password&amp;enquireLinkTimer=3000&amp;transactionTimer=5000&amp;systemType=pro
ducer"/>
</route>
```

Java DSL を使用して SMS を受信するルート:

```
from("smpp://smppclient@localhost:2775?
password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consumer")
  .to("bean:foo");
```

Spring XML DSL を使用して SMS を受信するルート:

```
  <route>
    <from uri="smpp://smppclient@localhost:2775?

password=password&amp;enquireLinkTimer=3000&amp;transactionTimer=5000&amp;systemType=co
nsumer"/>
    <to uri="bean:foo"/>
  </route>
```

ヒント

SMSC シミュレーター テストに SMSC シミュレーターが必要な場合は、 Logica が提供するシミュレーターを使用できます。

288.10. デバッグロギング

このコンポーネントにはログレベル **DEBUG** があり、問題のデバッグに役立ちます。log4j を使用する場合は、設定に次の行を追加できます。

```
log4j.logger.org.apache.camel.component.smpp=DEBUG
```

288.11. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第289章 SNMP コンポーネント

Camel バージョン 2.1 以降で利用可能

snmp: コンポーネントを使用すると、SNMP 対応デバイスをポーリングしたり、トラップを受信したりできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-snmp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

289.1. URI 形式

```
snmp://hostname[:port][?Options]
```

このコンポーネントは、SNMP 対応デバイスからの OID 値のポーリングとトラップの受信をサポートしています。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

289.2. SNMP プロデューサー

2.18 リリースから利用可能

また、GET メソッドを使用して情報をリクエストするために使用することもできます。

レスポンスボディーのタイプは org.apache.camel.component.snmp.SnmpMessage です。

289.3. オプション

SNMP コンポーネントにはオプションがありません。

SNMP エンドポイントは、URI 構文を使用して設定されます。

```
snmp:host:port
```

パスおよびクエリーパラメーターを使用します。

289.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** SNMP 対応デバイスのホスト名		String
port	**必須** SNMP 対応デバイスのポート番号		Integer

名前	説明	デフォルト	タイプ

289.3.2. クエリーパラメーター(34 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	更新レートを秒単位で設定します。	60000	long
oids (consumer)	関心のある値を定義します。ウィキペディアを見て理解を深めてください。単一の OID またはコンマ区切りの OID リストを指定できます。例: oids=1.3.6.1.2.1.1.3.0,1.3.6.1.2.1.25.3.2.1.5.1,1.3.6.1.2.1.25.3.5.1.1.1,1.3.6.1.2.1.43.5.1.1.11.1		String
protocol (consumer)	ここで、使用するプロトコルを選択できます。udp または tcp のいずれかを使用できます。	udp	String
retries (consumer)	リクエストをキャンセルする前に再試行する頻度を定義します。	2	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
snmpCommunity (consumer)	snmp 要求のコミュニティーオクテット文字列を設定します。	public	String
snmpContextEngineId (consumer)	スコープ PDU のコンテキストエンジン ID フィールドを設定します。		String
snmpContextName (consumer)	このスコープ PDU のコンテキスト名フィールドを設定します。		String

名前	説明	デフォルト	タイプ
snmpVersion (consumer)	リクエストの snmp バージョンを設定します。値 0 は SNMPv1、1 は SNMPv2c、値 3 は SNMPv3 を意味します。	0	int
timeout (consumer)	リクエストのタイムアウト値をミリ秒で設定します。	1500	int
type (consumer)	ポーリング、トラップなど、実行する操作。		SnmpActionType
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

名前	説明	デフォルト	タイプ
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
authenticationPassphrase (security)	認証パスフレーズ。null でない場合、authenticationProtocol も null でない必要があります。RFC3414 11.2 では、パスフレーズの長さを 8 バイト以上にする必要があります。authenticationPassphrase の長さが 8 バイト未満の場合、IllegalArgumentException が出力されます。		String
authenticationProtocol (security)	セキュリティーレベルが認証を有効にするように設定されている場合に使用する認証プロトコル。設定可能な値は次のとおりです。MD5、SHA1		String

名前	説明	デフォルト	タイプ
privacyPassphrase (security)	プライバシーパスフレーズ。null でない場合、privacyProtocol も null でない必要があります。RFC3414 11.2 では、パスフレーズの長さを 8 バイト以上にする必要があります。authenticationPassphrase の長さが 8 バイト未満の場合、IllegalArgumentException が出力されます。		String
privacyProtocol (security)	このユーザーに関連付けるプライバシープロトコル ID。null に設定すると、このユーザーは暗号化されていないメッセージのみをサポートします。		String
securityLevel (security)	このターゲットのセキュリティーレベルを設定します。指定されたセキュリティーレベルは、このターゲットに設定されたセキュリティー名に関連付けられたセキュリティーモデル依存情報によってサポートされている必要があります。値 1 は、認証なし、暗号化なしを意味します。誰でもこのセキュリティーレベルでメッセージを作成および読み取ることができます。値 2 は、認証および暗号化なしを意味します。このセキュリティーレベルでメッセージを作成できるのは、正しい認証キーを持つユーザーだけですが、メッセージの内容は誰でも読み取ることができます。値 3 は、認証と暗号化を意味します。正しい認証キーを持つ人だけがこのセキュリティーレベルでメッセージを作成でき、正しい暗号化/復号化キーを持つ人だけがメッセージの内容を読み取ることができます。	3	int
securityName (security)	このターゲットで使用するセキュリティー名を設定します。		文字列

289.4. アンケートの結果

状況を考慮して、次の OID をポーリングします。

OID

```
1.3.6.1.2.1.1.3.0
1.3.6.1.2.1.25.3.2.1.5.1
1.3.6.1.2.1.25.3.5.1.1.1
1.3.6.1.2.1.43.5.1.1.11.1
```

結果は次のようになります。

toString 変換の結果

```
<?xml version="1.0" encoding="UTF-8"?>
<snmp>
  <entry>
```

```
    <oid>1.3.6.1.2.1.1.3.0</oid>
    <value>6 days, 21:14:28.00</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.25.3.2.1.5.1</oid>
    <value>2</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.25.3.5.1.1.1</oid>
    <value>3</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.43.5.1.1.11.1</oid>
    <value>6</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.1.1.0</oid>
    <value>My Very Special Printer Of Brand Unknown</value>
  </entry>
</snmp>
```

お気づきかもしれませんが、リクエストされたものよりも 1 つ多くの結果があります....1.3.6.1.2.1.1.1.0.
この特殊なケースでは、これはデバイスによって自動的に入力されます。ですから、リクエストした以上のものを受け取ることは絶対に起こるかもしれません... 準備してください.

289.5. 例

リモートデバイスのポーリング:

```
snmp:192.168.178.23:161?protocol=udp&type=POLL&oids=1.3.6.1.2.1.1.5.0
```

トラップレシーバーの設定 (**ここでは OID 情報は必要ないことに注意してください!**):

```
snmp:127.0.0.1:162?protocol=udp&type=TRAP
```

Camel 2.10.0 から、メッセージヘッダー securityName で SNMP TRAP のコミュニティーを取得できます。
メッセージヘッダー peerAddress を持つ SNMP TRAP のピアアドレス。

Java でのルーティングの例: (SNMP PDU を XML 文字列に変換します)

```
from("snmp:192.168.178.23:161?protocol=udp&type=POLL&oids=1.3.6.1.2.1.1.5.0").
convertBodyTo(String.class).
to("activemq:snmp.states");
```

289.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第290章 SOAP DATAFORMAT

Camel バージョン 2.3 以降で利用可能

SOAP は、JAXB2 および JAX-WS アノテーションを使用して SOAP ペイロードをマーシャリングおよびアンマーシャリングするデータ形式です。CXF スタックを必要とせずに、Apache CXF の基本機能を提供します。

サポートされている SOAP バージョン

SOAP 1.1 はデフォルトでサポートされています。SOAP 1.2 は Camel 2.11 以降でサポートされています。

名前空間接頭辞のマッピング

SOAP データ形式を使用してマーシャリングするときに、名前空間接頭辞のマッピングを制御する方法は、JAXB を参照してください。

290.1. SOAP オプション

SOAP データ形式は、以下にリストされている 7 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
contextPath		String	JAXB クラスが配置されているパッケージ名。
encoding		String	特定のエンコーディングを無効にして使用します
elementNameStrategyRef		String	レジストリーから検索する要素ストラテジーを参照します。要素名ストラテジーは 2 つの目的で使用されます。1 つ目は、オブジェクトを SOAP メッセージにマーシャリングするときに、特定のオブジェクトと SOAP アクションの xml 要素名を見つけることで、2 つ目は、指定された SOAP エラー名の Exception クラスを見つけることです。次の 3 つの要素のストラテジークラス名は、そのまま使用できます。QNameStrategy: インスタンス化時に設定される固定の qName を使用します。例外ルックアップはサポートされていません。TypeNameStrategy: 指定された型の XMLType アノテーションの名前と名前空間を使用します。名前空間が設定されていない場合は、package-info が使用されます。例外ルックアップはサポートされていません。ServiceInterfaceStrategy: Web サービスインターフェイスからの情報を使用して、型名を決定し、SOAP 障害の例外クラスを見つけます。3 つのクラスはすべて、パッケージ名 org.apache.camel.dataformat.soap.name にあります。cxf-codegen または同様のツールを使用して Web サービスのスタブコードを生成した場合は、おそらく ServiceInterfaceStrategy を使用する必要があります。アノテーション付きのサービスインターフェイスがない場合は、QNameStrategy または TypeNameStrategy を使用する必要があります。
version	1.1	String	SOAP バージョンは 1.1 または 1.2 のいずれかである必要があります。デフォルトは 1.1 です。

名前	デフォルト	Java タイプ	説明
namespacePrefixRef		String	JAXB または SOAP を使用してマーシャリングする場合、JAXB 実装は、ns2、ns3、ns4 などの名前空間接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では目的のマッピングを含むマップを参照できます。
schema		String	既存のスキーマに対して検証します。接頭辞 classpath:、file:、または http: を使用して、リソースの解決方法を指定できます。「,」文字を使用して、複数のスキーマファイルを区切ることができます。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

290.2. ELEMENTNAMESTRATEGY

要素名ストラテジーは 2 つの目的で使用されます。1 つ目は、オブジェクトを SOAP メッセージにマーシャリングするときに、特定のオブジェクトと SOAP アクションの xml 要素名を見つけることで、2 つ目は、指定された SOAP エラー名の Exception クラスを見つけることです。

ストラテジー	用途
QNameStrategy	インスタンス化時に設定される固定の qName を使用します。例外ルックアップはサポートされていません。
TypeNameStrategy	指定された型の @XMLType アノテーションの名前と名前空間を使用します。名前空間が設定されていない場合は、package-info が使用されます。例外ルックアップはサポートされていません。
ServiceInterfaceStrategy	Web サービスインターフェイスからの情報を使用して、型名を決定し、SOAP エラーの例外クラスを見つけます。

cxf-codegen または同様のツールを使用して Web サービスのスタブコードを生成した場合は、おそらく ServiceInterfaceStrategy を使用するようにしてください。アノテーション付きのサービスインターフェイスがない場合は、QNameStrategy または TypeNameStrategy を使用する必要があります。

290.3. JAVA DSL を使用

次の例では、パッケージ com.example.customerservice で設定された **soap** の名前付き DataFormat を使用して、JAXBContext を初期化します。2 番目のパラメーターは ElementNameStrategy です。ルートは、通常のオブジェクトと例外をマーシャリングできます。(以下は、SOAP エンベロープをキュー

に送信するだけである点に注意してください。Web サービスプロバイダーは、SOAP 呼び出しが実際に発生するために実際にキューをリッスンする必要があります。この場合には、一方向の SOAP 要求になります。リクエストの返信が必要な場合は、次の例を参照してください。)

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```

ヒント

参照内容: SOAP データ形式は JAXB データ形式を継承しているため、ほとんどの設定がここにも適用されます。

290.3.1. SOAP 1.2 の使用

Camel 2.11 から利用可能

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
soap.setVersion("1.2");
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```

XML DSL を使用する場合、<soapjaxb> 要素に設定できる version 属性があります。

```
    <!-- Defining a ServiceInterfaceStrategy for retrieving the element name when marshalling -->
    <bean id="myNameStrategy"
class="org.apache.camel.dataformat.soap.name.ServiceInterfaceStrategy">
        <constructor-arg value="com.example.customerservice.CustomerService"/>
    <constructor-arg value="true"/>
    </bean>
```

そして Camel ルートの場合:

```
<route>
  <from uri="direct:start"/>
  <marshal>
    <soapjaxb contentPath="com.example.customerservice" version="1.2"
elementNameStrategyRef="myNameStrategy"/>
  </marshal>
  <to uri="jms:myQueue"/>
</route>
```

290.4. マルチパートメッセージ

Camel 2.8.1 以降で利用可能

マルチパート SOAP メッセージは ServiceInterfaceStrategy でサポートされています。ServiceInterfaceStrategy は、JAX-WS 2.2 に従ってアノテーションが付けられ、Document Bare スタイルの要件を満たすサービスインターフェイス定義で初期化する必要があります。ターゲットメソッド

は、JAX-WS 仕様に従い、以下の基準を満たす必要があります。1) **in** または **in**/**out** のヘッダー以外のパラメーターを最大 1 つ持っている、2) **void** 以外の戻り型が含まれる場合には、 **in**/**out** または **out** のヘッダー以外のパラメーターを含めることができない、3) **void** の戻り型が含まれる場合には最大 1 つの **in**/**out** または **out** ヘッダー以外のパラメーターを持つ必要がある。

ServiceInterfaceStrategy は、マッピング戦略が要求パラメーターまたは応答パラメーターのどちらに適用されるかを示すブール値パラメーターで初期化する必要があります。

```
ServiceInterfaceStrategy strat =  new
ServiceInterfaceStrategy(com.example.customerservice.multipart.MultiPartCustomerService.class,
true);
SoapJaxbDataFormat soapDataFormat = new
SoapJaxbDataFormat("com.example.customerservice.multipart", strat);
```

290.4.1. マルチパートリクエスト

マルチパートリクエストのペイロードパラメーターは、ターゲット操作の署名を反映する **BeanInvocation** オブジェクトを使用して初期化されます。camel-soap DataFormat は、 **marshal()** プロセッサーが呼び出されると、JAX-WS マッピングに従って、**BeanInvocation** のコンテンツを SOAP ヘッダーおよびボディのフィールドにマッピングします。

```
BeanInvocation beanInvocation = new BeanInvocation();

// Identify the target method
beanInvocation.setMethod(MultiPartCustomerService.class.getMethod("getCustomersByName",
    GetCustomersByName.class, com.example.customerservice.multipart.Product.class));

// Populate the method arguments
GetCustomersByName getCustomersByName = new GetCustomersByName();
getCustomersByName.setName("Dr. Multipart");

Product product = new Product();
product.setName("Multiuse Product");
product.setDescription("Useful for lots of things.");

Object[] args = new Object[] {getCustomersByName, product};

// Add the arguments to the bean invocation
beanInvocation.setArgs(args);

// Set the bean invocation object as the message body
exchange.getIn().setBody(beanInvocation);
```

290.4.2. マルチパートレスポンス

マルチパート SOAP 応答には、SOAP 本文に要素が含まれる場合があり、SOAP ヘッダーに 1 つ以上の要素が含まれます。camel-soap DataFormat は、soap ボディー (存在する場合) の要素をアンマーシャルし、エクスチェンジの out メッセージのボディーに配置します。ヘッダー要素は、JAXB マップオブジェクトタイプにマーシャリング **されません**。代わりに、これらの要素は camel 出力メッセージヘッダー **org.apache.camel.dataformat.soap.UNMARSHALLED_HEADER_LIST** に配置されます。要素は、**ignoreJAXBElement** プロパティーの設定に応じて、要素インスタンス値または JAXBElement 値として表示されます。このプロパティーは camel-jaxb から継承されます。

また、**ignoreUnmarshalledHeaders** 値を **true** に設定することで、camel-soap DataFormate がヘッダーコンテンツをすべて無視するようにすることもできます。

290.4.3. ホルダーオブジェクトのマッピング

JAX-WS は、**In**/**Out** および **Out** パラメーターに対して型パラメーター化された **javax.xml.ws.Holder** オブジェクトの使用を指定します。**BeanInvocation** をビルドするときに **Holder** オブジェクトを使用するか、パラメーター化された型のインスタンスを直接使用することができます。camel-soap DataFormat は、 **Holder の値クラスの JAXB マッピングに従って、Holder 値をマーシャリングします。アンマーシャリングされたレスポンス内の \'Holder** オブジェクトには、マッピングが提供されません。

290.5. 例

290.5.1. Web サービスクライアント

次のルートは、リクエストのマーシャリングと、レスポンスまたはエラーのアンマーシャリングをサポートしています。

```
String WS_URI = "cxf://http://myserver/customerservice?
serviceClass=com.example.customerservice&dataFormat=MESSAGE";
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
from("direct:customerServiceClient")
  .onException(Exception.class)
    .handled(true)
    .unmarshal(soapDF)
  .end()
  .marshal(soapDF)
  .to(WS_URI)
  .unmarshal(soapDF);
```

以下のスニペットは、サービスインターフェイスのプロキシーを作成し、上記のルートへの SOAP 呼び出しを行います。

```
import org.apache.camel.Endpoint;
import org.apache.camel.component.bean.ProxyHelper;
...

Endpoint startEndpoint = context.getEndpoint("direct:customerServiceClient");
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
// CustomerService below is the service endpoint interface, *not* the javax.xml.ws.Service subclass
CustomerService proxy = ProxyHelper.createProxy(startEndpoint, classLoader,
CustomerService.class);
GetCustomersByNameResponse response = proxy.getCustomersByName(new
GetCustomersByName());
```

290.5.2. Web サービスサーバー

次のルートを使用すると、jms キュー customerServiceQueue をリッスンし、クラス CustomerServiceImpl を使用してリクエストを処理する Web サービスサーバーが設定されます。当然、customerServiceImpl はインターフェイス CustomerService を実装する必要があります。サーバークラスを直接インスタンス化する代わりに、Spring コンテキストで通常の Bean として定義できます。

```
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
CustomerService serverBean = new CustomerServiceImpl();
from("jms://queue:customerServiceQueue")
  .onException(Exception.class)
    .handled(true)
    .marshal(soapDF)
  .end()
  .unmarshal(soapDF)
  .bean(serverBean)
  .marshal(soapDF);
```

290.6. 依存関係

camel ルートで SOAP データ形式を使用するには、次の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-soap</artifactId>
  <version>2.3.0</version>
</dependency>
```

第291章 SOLR コンポーネント

Camel バージョン 2.9 以降で利用可能

Solr コンポーネントを 使用すると、Apache Lucene Solr サーバー (SolrJ 3.5.0 ベース) とやり取りできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-solr</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

291.1. URI 形式

注記: solrs と solrCloud は、 **Camel 2.14** から新しく追加されました。

```
solr://host[:port]/solr?[options]
solrs://host[:port]/solr?[options]
solrCloud://host[:port]/solr?[options]
```

291.2. SOLR オプション

Solr コンポーネントにはオプションがありません。

Solr エンドポイントは、URI 構文を使用して設定されます。

```
solr:url
```

パスおよびクエリーパラメーターを使用します。

291.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
url	**必須** solr サーバーのホスト名とポート		String

291.2.2. クエリーパラメーター (13 パラメーター)

名前	説明	デフォルト	タイプ
allowCompression (producer)	これを有効にするには、サーバー側で gzip または deflate をサポートする必要があります		Boolean

名前	説明	デフォルト	タイプ
connectionTimeout (producer)	基になる HttpConnectionManager の connectionTimeout。		Integer
defaultMaxConnectionsPer Host (producer)	基になる HttpConnectionManager の maxConnectionsPerHost。		Integer
followRedirects (producer)	Solr サーバーへのアクセスにリダイレクトを使用するかどうかを示します。		Boolean
maxRetries (producer)	一時的なエラーが発生した場合に試行する最大再試行回数。		Integer
maxTotalConnections (producer)	基になる HttpConnectionManager の maxTotalConnection。		Integer
requestHandler (producer)	使用するリクエストハンドラーを設定します。		String
soTimeout (producer)	基になる HttpConnectionManager の読み取りタイムアウト。これはクエリーには適していますが、おそらくインデックス作成には適していません。		Integer
streamingQueueSize (producer)	StreamingUpdateSolrServer のキューサイズを設定します。	10	int
streamingThreadCount (producer)	StreamingUpdateSolrServer のスレッド数を設定します。	2	int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
コレクション (solrCloud)	solrCloud サーバーが使用できるコレクション名を設定します。		String
zkHost (solrCloud)	zkhost=localhost:8123 など、solrCloud が使用できる ZooKeeper ホスト情報を設定します。		文字列

291.3. メッセージ操作

現在、次の Solr 操作がサポートされています。SolrOperation のキーと次のいずれかに設定された値を使用してエクスチェンジヘッダーを設定するだけです。一部の操作では、メッセージボディーも設定する必要があります。

- INSERT 操作は CommonsHttpSolrServer を使用します
- INSERT_STREAMING 操作は StreamingUpdateSolrServer (**Camel 2.9.2**) を使用します。

操作	メッセージボディー	説明
INSERT/INSERT_STREAMING	該当なし	メッセージヘッダーを使用してインデックスを追加します (SolrField を前に付ける必要があります)。
INSERT/INSERT_STREAMING	File	指定されたファイルを使用してインデックスを追加します (ContentStreamUpdateRequest を使用)
INSERT/INSERT_STREAMING	SolrInputDocument	**Camel 2.9.2** は、指定された SolrInputDocument に基づいてインデックスを更新します
INSERT/INSERT_STREAMING	String XML	**Camel 2.9.2** は、指定された XML に基づいてインデックスを更新します (SolrInputDocument 形式に従う必要があります)。
ADD_BEAN	Bean インスタンス	アノテーション付き Bean の値に基づいてインデックスを追加します
ADD_BEANS	コレクション <bean>	**Camel 2.15** は、アノテーション付き Bean のコレクションに基づいてインデックスを追加します
DELETE_BY_ID	削除するインデックス ID	ID でレコードを削除します。
DELETE_BY_QUERY	query string	クエリーでレコードを削除します。
COMMIT	該当なし	保留中のインデックス変更に対してコミットを実行します。

操作	メッセージボディー	説明
ROLLBACK	該当なし	保留中のインデックス変更に対してロールバックを実行します。
OPTIMIZE	該当なし	保留中のインデックス変更に対してコミットを実行し、最適化コマンドを実行します。

291.4. 例

以下は、簡単な INSERT、DELETE、COMMIT の例です。

```
from("direct:insert")
    .setHeader(SolrConstants.OPERATION, constant(SolrConstants.OPERATION_INSERT))
    .setHeader(SolrConstants.FIELD + "id", body())
    .to("solr://localhost:8983/solr");

from("direct:delete")
    .setHeader(SolrConstants.OPERATION, constant(SolrConstants.OPERATION_DELETE_BY_ID))
    .to("solr://localhost:8983/solr");

from("direct:commit")
    .setHeader(SolrConstants.OPERATION, constant(SolrConstants.OPERATION_COMMIT))
    .to("solr://localhost:8983/solr");
```

```
<route>
    <from uri="direct:insert"/>
    <setHeader headerName="SolrOperation">
        <constant>INSERT</constant>
    </setHeader>
    <setHeader headerName="SolrField.id">
        <simple>${body}</simple>
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
<route>
    <from uri="direct:delete"/>
    <setHeader headerName="SolrOperation">
        <constant>DELETE_BY_ID</constant>
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
<route>
    <from uri="direct:commit"/>
    <setHeader headerName="SolrOperation">
        <constant>COMMIT</constant>
```

```
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
```

クライアントは、本文メッセージを挿入ルートまたは削除ルートに渡してから、コミットルートを呼び出すだけで済みます。

```
template.sendBody("direct:insert", "1234");
template.sendBody("direct:commit", null);
template.sendBody("direct:delete", "1234");
template.sendBody("direct:commit", null);
```

291.5. SOLR のクエリー

現在、このコンポーネントはネイティブでのデータのクエリーをサポートしていません (後で追加される可能性があります)。今のところ、次のように HTTP を使用して Solr にクエリーを実行できます。

```
//define the route to perform a basic query
from("direct:query")
    .recipientList(simple("http://localhost:8983/solr/select/?q=${body}"))
    .convertBodyTo(String.class);
...
//query for an id of '1234' (url encoded)
String responseXml = (String) template.requestBody("direct:query", "id%3A1234");
```

詳細については、これらのリソースを参照してください...

Solr クエリーのチュートリアル

Solr クエリー構文

291.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

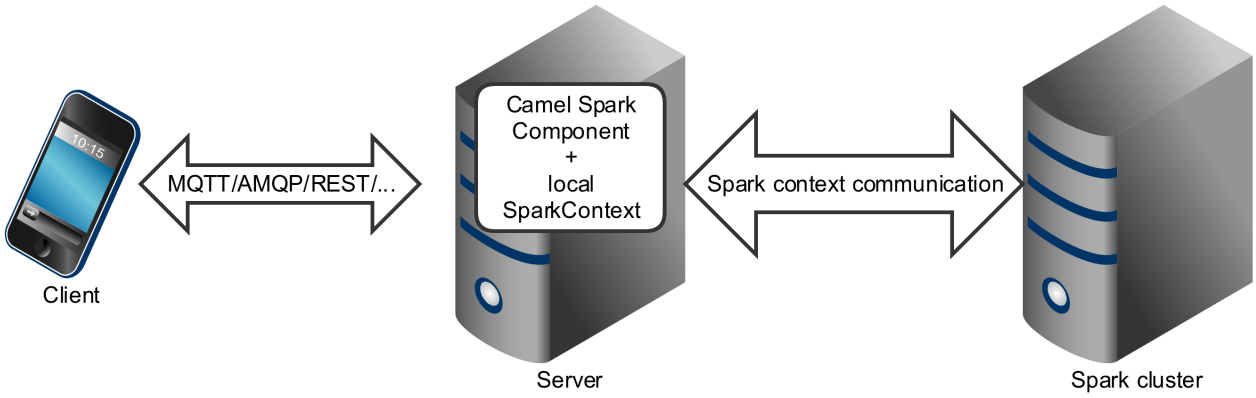
第292章 APACHE SPARK コンポーネント

Camel バージョン 2.17 以降で利用可能

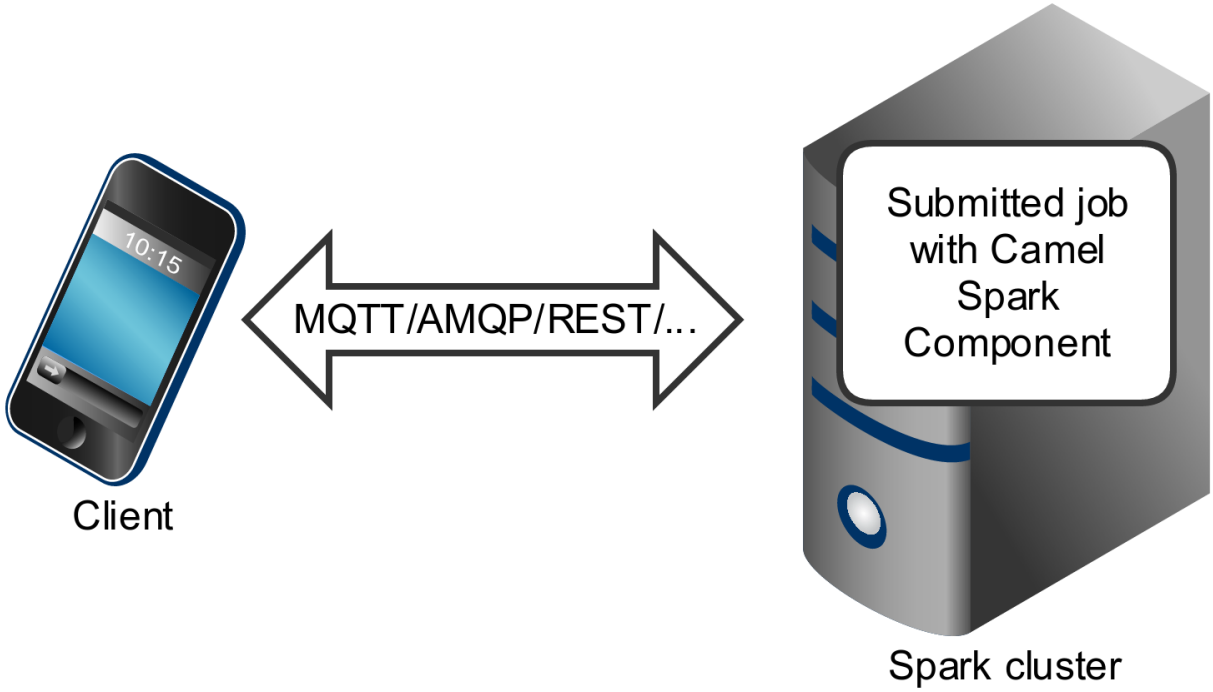
このドキュメントページでは、Apache Camel の Apache Spark コンポーネントについて説明します。Spark と Camel の統合の主な目的は、Camel コネクターと Spark タスクの間のブリッジを提供することです。特に、Camel コネクターは、さまざまなトランスポートからメッセージをルーティングし、実行するタスクを動的に選択し、受信メッセージをそのタスクの入力データとして使用し、最終的に実行結果を Camel パイプラインに戻す方法を提供します。

292.1. サポートされているアーキテクチャースタイル

Spark コンポーネントは、アプリケーションサーバーにデプロイされる (またはファット jar として実行される) ドライバーアプリケーションとして使用できます。



Spark コンポーネントは、Spark クラスターにジョブとして直接送信することもできます。



Spark コンポーネントは主に、Spark クラスターと他のエンドポイント間のブリッジとして機能する **長時間実行ジョブ** として機能するように設計されていますが、**1回限りの** 短いジョブとして使用することもできます。

292.2. OSGI サーバーで SPARK を実行する

現在、Spark コンポーネントは OSGi コンテナーでの実行をサポートしていません。Spark はファット jar として実行されるように設計されており、通常はジョブとしてクラスターに送信されます。これらの理由から、OSGi サーバーで Spark を実行することは、少なくとも困難であり、Camel でもサポートされていません。

292.3. URI 形式

現在、Spark コンポーネントはプロデューサーのみをサポートしています。これは、Spark ジョブを呼び出して結果を返すことを目的としています。RDD、データフレーム、または Hive SQL ジョブを呼び出すことができます。

Spark URI 形式

```
spark:{rdd|dataframe|hive}
```

292.3.1. Spark オプション

Apache Spark コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
rdd (producer)	計算対象の RDD。		JavaRDDLike
rddCallback (producer)	RDD に対してアクションを実行する関数。		RddCallback
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Apache Spark エンドポイントは、URI 構文を使用して設定されます。

```
spark:endpointType
```

パスおよびクエリーパラメーターを使用します。

292.3.2. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
endpointType	**必須** エンドポイントのタイプ (rdd、データフレーム、ハイブ)。		EndpointType

292.3.3. クエリーパラメーター (6 個のパラメーター):

名前	説明	デフォルト	タイプ
collect (producer)	結果を収集またはカウントする必要があるかどうかを示します。	true	boolean
dataFrame (producer)	計算対象の DataFrame。		DataFrame
dataFrameCallback (producer)	DataFrame に対してアクションを実行する関数。		DataFrameCallback
rdd (producer)	計算対象の RDD。		JavaRDDLike
rddCallback (producer)	RDD に対してアクションを実行する関数。		RddCallback
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

RDD ジョブ

RDD ジョブを呼び出すには、次の URI を使用します。

Spark RDD producer

```
spark:rdd?rdd=#testFileRdd&rddCallback=#transformation
```

rdd オプションは Camel レジストリーの RDD インスタンス (**org.apache.spark.api.java.JavaRDDLike** のサブクラス) の名前を参照し、**rddCallback** は **org.apache.camel.component.spark.RddCallback** インターフェイスの実装を参照します。(レジストリーからも)。RDD コールバックは、特定の RDD に対して入力メッセージを適用するために使用される単一のメソッドを提供します。コールバック計算の結果は、ボディーとしてエクスチェンジに保存されます。

Spark RDD コールバック

```
public interface RddCallback<T> {
    T onRdd(JavaRDDLike rdd, Object... payloads);
}
```

次のスニペットは、ジョブへの入力としてメッセージを送信し、結果を返す方法を示しています。

Spark ジョブの呼び出し

```
String pattern = "job input";
long linesCount = producerTemplate.requestBody("spark:rdd?
rdd=#myRdd&rddCallback=#countLinesContaining", pattern, long.class);
```

Spring Bean として登録された上記のスニペットの RDD コールバックは、次のようになります。

Spark RDD コールバック

```
@Bean
RddCallback<Long> countLinesContaining() {
   return new RddCallback<Long>() {
      Long onRdd(JavaRDDLike rdd, Object... payloads) {
         String pattern = (String) payloads[0];
         return rdd.filter({line -> line.contains(pattern)}).count();
      }
   }
}
```

Spring の RDD 定義は次のようになります。

Spark RDD 定義

```
@Bean
JavaRDDLike myRdd(JavaSparkContext sparkContext) {
  return sparkContext.textFile("testrdd.txt");
}
```

292.3.4. RDD コールバックを無効にする

RDD コールバックが Camel パイプラインに値を返さない場合は、**null** 値を返すか、**VoidRddCallback** 基本クラスを使用できます。

Spark RDD 定義

```
@Bean
RddCallback<Void> rddCallback() {
  return new VoidRddCallback() {
      @Override
      public void doOnRdd(JavaRDDLike rdd, Object... payloads) {
         rdd.saveAsTextFile(output.getAbsolutePath());
      }
   };
}
```

292.3.5. RDD コールバックの変換

RDD コールバックに送信される入力データのタイプがわかっている場合は、**ConvertingRddCallback** を使用して、受信メッセージをコールバックに挿入する前に Camel に自動的に変換させることができます。

Spark RDD 定義

```
@Bean
RddCallback<Long> rddCallback(CamelContext context) {
  return new ConvertingRddCallback<Long>(context, int.class, int.class) {
         @Override
         public Long doOnRdd(JavaRDDLike rdd, Object... payloads) {
            return rdd.count() * (int) payloads[0] * (int) payloads[1];
         }
```

```
        };
    };
}
```

292.3.6. アノテーション付き RDD コールバック

おそらく、RDD コールバックを操作する最も簡単な方法は、**@RddCallback** アノテーションでマークされたメソッドをクラスに提供することです。

アノテーション付き RDD コールバック定義

```
import static
org.apache.camel.component.spark.annotations.AnnotatedRddCallback.annotatedRddCallback;

@Bean
RddCallback<Long> rddCallback() {
    return annotatedRddCallback(new MyTransformation());
}

...

import org.apache.camel.component.spark.annotation.RddCallback;

public class MyTransformation {

    @RddCallback
    long countLines(JavaRDD<String> textFile, int first, int second) {
        return textFile.count() * first * second;
    }

}
```

CamelContext をアノテーション付き RDD コールバックファクトリーメソッドに渡す場合、作成されたコールバックは、入力ペイロードを変換して、アノテーション付きメソッドのパラメーターに一致させることができます。

アノテーション付き RDD コールバックのボディー変換

```
import static
org.apache.camel.component.spark.annotations.AnnotatedRddCallback.annotatedRddCallback;

@Bean
RddCallback<Long> rddCallback(CamelContext camelContext) {
    return annotatedRddCallback(new MyTransformation(), camelContext);
}

...


import org.apache.camel.component.spark.annotation.RddCallback;

public class MyTransformation {

    @RddCallback
    long countLines(JavaRDD<String> textFile, int first, int second) {
```

```
        return textFile.count() * first * second;
    }

}

...

// Convert String "10" to integer
long result = producerTemplate.requestBody("spark:rdd?rdd=#rdd&rddCallback=#rddCallback"
Arrays.asList(10, "10"), long.class);
```

292.4. DATAFRAME ジョブ

RDD を使用する代わりに、Spark コンポーネントは DataFrame を使用することもできます。

DataFrame ジョブを呼び出すには、次の URI を使用します。

Spark RDD producer

```
spark:dataframe?dataFrame=#testDataFrame&dataFrameCallback=#transformation
```

dataFrame オプションは、Camel レジストリーの DataFrame インスタンス (**instance of of org.apache.spark.sql.DataFrame**) の名前を参照し、**dataFrameCallback** は **org.apache.camel.component.spark.DataFrameCallback** インターフェイスの実装を参照します (レジストリーからも)。DataFrame コールバックは、指定された DataFrame に対して入力メッセージを適用するために使用される単一のメソッドを提供します。コールバック計算の結果は、ボディーとしてエクスチェンジに保存されます。

Spark RDD コールバック

```
public interface DataFrameCallback<T> {
    T onDataFrame(DataFrame dataFrame, Object... payloads);
}
```

次のスニペットは、ジョブへの入力としてメッセージを送信し、結果を返す方法を示しています。

Spark ジョブの呼び出し

```
String model = "Micra";
long linesCount = producerTemplate.requestBody("spark:dataFrame?
dataFrame=#cars&dataFrameCallback=#findCarWithModel", model, long.class);
```

Spring Bean として登録された上記のスニペットの DataFrame コールバックは、次のようになります。

Spark RDD コールバック

```
@Bean
RddCallback<Long> findCarWithModel() {
    return new DataFrameCallback<Long>() {
        @Override
        public Long onDataFrame(DataFrame dataFrame, Object... payloads) {
            String model = (String) payloads[0];
```

```
            return dataFrame.where(dataFrame.col("model").eqNullSafe(model)).count();
        }
    };
}
```

Spring の DataFrame 定義は次のようになります。

Spark RDD 定義

```
@Bean
DataFrame cars(HiveContext hiveContext) {
    DataFrame jsonCars = hiveContext.read().json("/var/data/cars.json");
    jsonCars.registerTempTable("cars");
    return jsonCars;
}
```

292.5. HIVE ジョブ

RDD または DataFrame Spark コンポーネントを操作する代わりに、Hive SQL クエリーをペイロードとして受け取ることもできます。 Hive クエリーを Spark コンポーネントに送信するには、次の URI を使用します。

Spark RDD producer

```
spark:hive
```

次のスニペットは、ジョブへの入力としてメッセージを送信し、結果を返す方法を示しています。

Spark ジョブの呼び出し

```
long carsCount = template.requestBody("spark:hive?collect=false", "SELECT * FROM cars",
Long.class);
List<Row> cars = template.requestBody("spark:hive", "SELECT * FROM cars", List.class);
```

クエリーを実行するテーブルは、クエリーを実行する前に HiveContext に登録する必要があります。たとえば、Spring では、このような登録は次のようになります。

Spark RDD 定義

```
@Bean
DataFrame cars(HiveContext hiveContext) {
    DataFrame jsonCars = hiveContext.read().json("/var/data/cars.json");
    jsonCars.registerTempTable("cars");
    return jsonCars;
}
```

292.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- スタートガイド

第293章 SPARK REST コンポーネント

Camel バージョン 2.14 以降で利用可能

Spark-rest コンポーネントを使用すると、Rest DSL を使用して Spark REST Java ライブラリー を使用して REST エンドポイントを定義できます。

情報: Spark Java には Java 8 ランタイムが必要です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spark-rest</artifactId>
    <version>${camel-version}</version>
  </dependency>
```

293.1. URI 形式

```
spark-rest://verb:path?[options]
```

293.2. URI オプション

Spark Rest コンポーネントは、以下に示す 12 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
port (consumer)	ポート番号デフォルトでは 4567 を使用します	4567	int
ipAddress (consumer)	Spark がリッスンする IP アドレスを設定します。呼び出されない場合、デフォルトのアドレスは 0.0.0.0 です。	0.0.0.0	String
minThreads (advanced)	Spark スレッドプール内のスレッドの最小数 (グローバルに共有)		int
maxThreads (advanced)	Spark スレッドプール内のスレッドの最大数 (グローバルに共有)		int
timeOutMillis (advanced)	長時間アイドル状態だったスレッドがスレッドプールから終了するスレッドアイドルタイムアウト (ミリ秒)。		int
keystoreFile (security)	キーストアファイルを使用するように接続を安全に設定します。		String
keystorePassword (security)	キーストアのパスワードを使用するように接続を安全に設定します。		String

名前	説明	デフォルト	タイプ
truststoreFile (security)	トラストストアファイルを使用するように接続を安全に設定します。		String
truststorePassword (security)	トラストストアのパスワードを使用するように接続を安全に設定します。		String
sparkConfiguration (advanced)	共有 SparkConfiguration を使用する場合。		SparkConfiguration
sparkBinding (advanced)	カスタム SparkBinding を使用して Camel メッセージとの間でマッピングします。		SparkBinding
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spark Rest エンドポイントは、URI 構文を使用して設定されます。

```
spark-rest:verb:path
```

パスおよびクエリーパラメーターを使用します。

293.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
verb	**必須** get、post、put、patch、delete、head、trace、connect、または options。		String
path	**必須** Spark 構文をサポートするコンテンツパス。		String

293.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
accept (consumer)	text/xml または application/json などのタイプを受け入れます。デフォルトでは、すべての種類のタイプを受け入れます。		String

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
disableStreamCache (consumer)	Spark HttpRequestgetContent() からの raw 入力ストリームがキャッシュされているかどうかを決定します (Camel はストリームを軽量メモリーベースのストリームキャッシュに読み込みます)。デフォルトでは、Camel はサーブレット入力ストリームをキャッシュして複数回の読み取りをサポートし、Camel がストリームからすべてのデータを取得できるようにします。ただし、ファイルやその他の永続ストアに直接ストリーミングするなど、生のストリームにアクセスする必要がある場合は、このオプションを true に設定できます。このオプションを有効にすると、そのままでは Netty ストリームを複数回読み取ることができず、Spark の raw ストリームのリーダーインデックスを手動でリセットする必要があることに注意してください。	false	boolean
mapHeaders (consumer)	このオプションを有効にすると、Spark から Camel Message へのバインド中にヘッダーもマップされます (たとえば、ヘッダーとして Camel Message にも追加されます)。このオプションをオフにして、これを無効にすることができます。ヘッダーには、Spark HTTP リクエストインスタンスを返すメソッド getRequest() を使用して、org.apache.camel.component.sparkrest.SparkMessage メッセージから引き続きアクセスできます。	true	boolean
transferException (consumer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	boolean

名前	説明	デフォルト	タイプ
urlDecodeHeaders (consumer)	このオプションを有効にすると、Spark から Camel Message へのバインド中にヘッダー値が URL デコードされます (たとえば、%20 はスペース文字になります)。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
matchOnUriPrefix (advanced)	完全に一致するものが見つからない場合に、コンシューマーが URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	boolean
sparkBinding (advanced)	カスタム SparkBinding を使用して Camel メッセージとの間でマッピングします。		SparkBinding
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

293.3. SPARK 構文を使用したパス

パスオプションは、パラメーターとスプラットのサポートを使用して REST コンテキストパスを定義する Spark REST 構文を使用して定義されます。詳細は、Spark Java Route の ドキュメントを参照してください。

以下は、固定パスを使用した Camel ルートです。

```
from("spark-rest:get:hello")
  .transform().constant("Bye World");
```

次のルートでは、キー me を持つ Camel ヘッダーにマップされたパラメーターを使用します。

```
from("spark-rest:get:hello/:me")
  .transform().simple("Bye ${header.me}");
```

293.4. CAMEL メッセージへのマッピング

Spark リクエストオブジェクトは、getRequest メソッドを使用して raw Spark リクエストにアクセスできる **org.apache.camel.component.sparkrest.SparkMessage** として Camel メッセージにマップされます。デフォルトでは、Spark ボディーは Camel メッセージボディーにマップされ、すべての HTTP

ヘッダー/Spark パラメーターは Camel メッセージヘッダーにマップされます。キー splat を使用して Camel メッセージヘッダーにマップされる Spark splat 構文の特別なサポートがあります。

たとえば、以下の特定のルートでは、コンテキストパスで Spark スプラット (アスタリスク記号) を使用しており、Simple 言語からヘッダーとしてアクセスして、レスポンスメッセージを作成できます。

```
from("spark-rest:get:/hello/*/to/*")
  .transform().simple("Bye big ${header.splat[1]} from ${header.splat[0]}");
```

293.5. REST DSL

Apache Camel は、REST サービスを優れた REST スタイルで定義できる新しい Rest DSL を提供します。たとえば、以下に示すように REST hello サービスを定義できます。

```
return new RouteBuilder() {
    @Override
    public void configure() throws Exception {
         rest("/hello/{me}").get()
             .route().transform().simple("Bye ${header.me}");
     }
  };
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <rest uri="/hello/{me}">
    <get>
      <route>
        <transform>
          <simple>Bye ${header.me}</simple>
        </transform>
      </route>
    </get>
  </rest>
</camelContext>
```

詳細については、Rest DSL を参照してください。

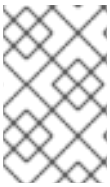
293.6. その他の例

Apache Camel ディストリビューションには **camel-example-spark-rest-tomcat** の例があり、Apache Tomcat または同様の Web コンテナーにデプロイできる Web アプリケーションで camel-spark-rest を使用する方法を示しています。

第294章 SPEL 言語

Camel バージョン 2.7 以降で利用可能

Camel では、Spring Expression Language (SpEL) をDSL または XML 設定で式または述語として使用できます。



注記

Spring ランタイムでは SpEL を使用することをお勧めします。ただし、Camel 2.21 以降では、他のランタイムで SpEL を使用できます (Spring ランタイムで実行されていない場合、SpEL が実行できない機能がある場合があります)。

294.1. VARIABLES

次の変数は、SpEL で記述された式と述語で使用できます。

変数	タイプ	説明
this	Exchange	Exchange はルートオブジェクトです
exchange	Exchange	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)
exchangeId	String	エクスチェンジ ID
fault	メッセージ	Fault メッセージ (ある場合)
body	Object	IN メッセージボディー
request	メッセージ	exchange.in メッセージ
response	メッセージ	exchange.out メッセージ (存在する場合)
properties	Map	エクスチェンジプロパティー
property(name)	Object	指定された名前によるプロパティー

変数	タイプ	説明
property(name, type)	タイプ	指定されたタイプとして指定された名前によるプロパティー

294.2. オプション

SpEL 言語は、以下にリストされている 1 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

294.3. サンプル

294.3.1. 式テンプレート

式のテンプレート化が有効になっているため、SpEL 式は **#{ }** 区切り記号で囲む必要があります。これにより、SpEL 式を通常のテキストと組み合わせて、これを非常に軽量なテンプレート言語として使用できます。

たとえば、次のルートを構築するとします。

```
from("direct:example")
    .setBody(spel("Hello #{request.body}! What a beautiful #{request.headers['dayOrNight']}"))
    .to("mock:result");
```

上記のルートで、spel は **org.apache.camel.language.spel.SpelExpression.spel** からインポートする必要がある静的メソッドであることに注意してください。これは、**setBody** メソッドにパラメーターとして渡される Expression として spel を使用するためです。fluent API を使用する場合は、代わりにこれを行うことができます。

```
from("direct:example")
    .setBody().spel("Hello #{request.body}! What a beautiful #{request.headers['dayOrNight']}")
    .to("mock:result");
```

setBody() メソッドの **spel** メソッドを使用していることに注意してください。そして、これは **org.apache.camel.language.spel.SpelExpression.spel** から spel メソッドを静的にインポートする必要はありません。

そして、本文に文字列 World、ヘッダー dayOrNight、値 day を含むメッセージを送信しました。

```
template.sendBodyAndHeader("direct:example", "World", "dayOrNight", "day");
```

mock:result の出力は、**"Hello World!What a beautiful day"** となります。

294.3.2. Bean インテグレーション

SpEL 式で、レジストリー (ほとんどの場合 **ApplicationContext**) で定義された Bean を参照できます。たとえば、**ApplicationContext** に foo という名前の Bean がある場合、次のようにこの Bean で bar メソッドを呼び出すことができます。

```
#{@foo.bar == 'xyz'}
```

294.3.3. エンタープライズ統合パターンでの SpEL

Recipient List の式として、または Message Filter 内の述語として SpEL を使用できます。

```
<route>
  <from uri="direct:foo"/>
  <filter>
    <spel>#{request.headers['foo'] == 'bar'}</spel>
    <to uri="direct:bar"/>
  </filter>
</route>
```

そして、Java DSL で同等のもの:

```
from("direct:foo")
    .filter().spel("#{request.headers['foo'] == 'bar'}")
    .to("direct:bar");
```

294.4. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").spel("resource:classpath:myspel.txt")
```

第295章 SPLUNK コンポーネント

Camel バージョン 2.13 以降で利用可能

Splunk コンポーネントは、Splunk が提供する クライアント API を使用して Splunk へのアクセスを提供し、Splunk でイベントを公開および検索できるようにします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-splunk</artifactId>
    <version>${camel-version}</version>
  </dependency>
```

295.1. URI 形式

```
splunk://[endpoint]?[options]
```

295.2. プロデューサーエンドポイント:

エンドポイント	説明
stream	指定されていない場合は、名前付きインデックスまたはデフォルトにデータをストリーミングします。ストリームモードを使用する場合、Splunk には、イベントがインデックスに到達する前に内部バッファー (約 1MB 程度) があることに注意してください。リアルタイムが必要な場合は、submit または tcp モードを使用することをお勧めします。
submit	提出モード。Splunk REST API を使用して、指定されていない場合は名前付きインデックスまたはデフォルトにイベントを発行します。
tcp	TCP モード。データを tcp ポートにストリーミングし、Splunk で開いているレシーバーポートを必要とします。

イベントを発行する場合、メッセージボディーに SplunkEvent を含める必要があります。 メッセージボディーの下のコメントを参照してください。

例

```
    from("direct:start").convertBodyTo(SplunkEvent.class)
        .to("splunk://submit?
username=user&password=123&index=myindex&sourceType=someSourceType&source=mySource"
)...
```

この例では、SplunkEvent クラスに変換するためにコンバーターが必要です。

295.3. コンシューマーエンドポイント

エンドポイント	説明
normal	通常の検索を実行し、検索オプションで検索クエリーを必要とします。
savedsearch	splunk に保存された検索クエリーに基づいて検索を実行し、savedSearch オプションでクエリーの名前を必要とします。

例

```
    from("splunk://normal?delay=5s&username=user&password=123&initEarliestTime=-10s&search=search index=myindex sourcetype=someSourcetype")
        .to("direct:search-result");
```

camel-splunk は、ボディーに SplunkEvent を使用して、検索結果ごとにルートエクスチェンジを作成します。

295.4. URI オプション

Splunk コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
splunkConfigurationFactory (advanced)	SplunkConfigurationFactory を使用する場合。		SplunkConfiguration ファクトリー
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Splunk エンドポイントは、URI 構文を使用して設定されます。

```
splunk:name
```

パスおよびクエリーパラメーターを使用します。

295.4.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** 名前には目的がありません		String

295.4.2. クエリーパラメーター (42 個のパラメーター):

名前	説明	デフォルト	タイプ
app (common)	Splunk アプリケーション。		String
connectionTimeout (common)	Splunk サーバーへの接続時の MS でのタイムアウト。	5000	int
host (Common)	Splunk ホスト。	localhost	String
owner (Common)	Splunk 所有者。		String
port (Common)	Splunk ポート。	8089	int
scheme (common)	Splunk スキーム。	https	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
count (consumer)	返されるエンティティーの最大数を示す数値。		int
earliestTime (consumer)	検索時間枠の最も早い時間。		String
initEarliestTime (consumer)	最初の検索の初期開始オフセット		String
latestTime (consumer)	検索時間枠の最新時間。		String
savedSearch (consumer)	実行する Splunk に保存されたクエリーの名前。		String
search (consumer)	実行する Splunk クエリー		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

名前	説明	デフォルト	タイプ
streaming (consumer)	ストリーミングモードを設定します。ストリーミングモードでは、バッチではなく、受信時にエクスチェンジが送信されます。		Boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
eventHost (producer)	デフォルトの Splunk イベントホストフィールドを上書きします。		String
index (producer)	書き込み先の Splunk インデックス。		String
raw (producer)	ペイロードをそのまま挿入する必要があります。	false	boolean
source (producer)	Splunk ソース引数。		String
sourceType (producer)	Splunk ソースタイプ引数。		String
tcpReceiverPort (producer)	Splunk TCP 受信ポート。		int
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int

名前	説明	デフォルト	タイプ
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit

名前	説明	デフォルト	タイプ
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
password (security)	Splunk のパスワード。		String
sslProtocol (security)	使用する SSL プロトコルを設定します。	TLSv1.2	SSLSecurityProtocol
username (security)	Splunk のユーザー名。		String
useSunHttpsHandler (security)	sun.net.www.protocol.https.Handler Https ハンドラーを使用して、Splunk 接続を確立します。アプリを回避するためにアプリケーションサーバーで実行する場合に役立ちます。サーバーの https 処理。	false	boolean

295.5. メッセージボディー

Splunk は、キーと値のペアでデータを操作します。SplunkEvent クラスは、そのようなデータのプレースホルダーであり、プロデューサーのメッセージ本文にある必要があります。同様に、コンシューマーの検索結果ごとに本文で返されます。

Camel 2.16.0 では、プロデューサーエンドポイントで raw オプションを設定することにより、raw データを Splunk に送信できます。これは、たとえば次の場合に役立ちます。json/xml および Splunk がサポートしているその他のペイロード。

295.6. ユースケース

Twitter で音楽付きのツイートを検索し、イベントを Splunk に公開する

```
    from("twitter://search?
type=polling&keywords=music&delay=10&consumerKey=abc&consumerSecret=def&accessToken=hij&
ccessTokenSecret=xxx")
        .convertBodyTo(SplunkEvent.class)
        .to("splunk://submit?username=foo&password=bar&index=camel-
tweets&sourceType=twitter&source=music-tweets");
```

ツイートを SplunkEvent に変換するには、次のようなコンバーターを使用できます。

```
@Converter
public class Tweet2SplunkEvent {
   @Converter
   public static SplunkEvent convertTweet(Status status) {
      SplunkEvent data = new SplunkEvent("twitter-message", null);
```

```
        //data.addPair("source", status.getSource());
        data.addPair("from_user", status.getUser().getScreenName());
        data.addPair("in_reply_to", status.getInReplyToScreenName());
        data.addPair(SplunkEvent.COMMON_START_TIME, status.getCreatedAt());
        data.addPair(SplunkEvent.COMMON_EVENT_ID, status.getId());
        data.addPair("text", status.getText());
        data.addPair("retweet_count", status.getRetweetCount());
        if (status.getPlace() != null) {
            data.addPair("place_country", status.getPlace().getCountry());
            data.addPair("place_name", status.getPlace().getName());
            data.addPair("place_street", status.getPlace().getStreetAddress());
        }
        if (status.getGeoLocation() != null) {
            data.addPair("geo_latitude", status.getGeoLocation().getLatitude());
            data.addPair("geo_longitude", status.getGeoLocation().getLongitude());
        }
        return data;
    }
}
```

Splunk でツイートを検索する

```
    from("splunk://normal?username=foo&password=bar&initEarliestTime=-2m&search=search
index=camel-tweets sourcetype=twitter")
        .log("${body}");
```

295.7. 他のコメント

Splunk には、機械で生成されたデータを分析および表示するための事前構築済みアプリで活用するためのさまざまなオプションが用意されています。
たとえば、jmx アプリケーション。jmx 属性を公開するために使用できます。また jvm メトリクスを Splunk にルーティングし、これをダッシュボードに表示します。

295.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第296章 SPRING サポート

Apache Camel は、さまざまな方法で Spring Framework とうまく連携するように設計されています。

- Camel は、JMS や JPA などのコンポーネントでデフォルトのトランザクション処理として Spring Transactions を使用します
- Camel は、Spring 2 XML 処理で Xml 設定を使用して動作します
- Camel Spring XML スキーマは、Xml リファレンス で定義されています
- Camel は、Spring Remoting の強力なバージョンをサポートしており、トランスポートに利用可能なすべてのコンポーネントを使用するとともに、クライアントとサーバー側の間で強力なルーティングを使用できます
- Camel は、Spring ApplicationContext で定義された任意の Bean との強力な Bean 統合を提供します
- Camel は、さまざまな Spring ヘルパークラスと統合されています。Spring Resources の Type Converter サポートの提供など
- Spring が Component インスタンスまたは CamelContext インスタンス自体に依存性を注入し、Spring Bean をコンポーネントおよびエンドポイントとして自動公開できるようにします
- Spring テストフレームワークを再利用して、エンタープライズ統合パターン と Camel の強力な モック および テスト エンドポイントを使用して、単体テストと統合テストを簡素化できます。
- **Camel 2.15** 以降、Camel は **camel-spring-boot** コンポーネントを使用して Spring Boot をサポートします

296.1. SPRING を使用して CAMELCONTEXT を設定する

CamelContextFactoryBean を使用して、任意の spring.xml 内で CamelContext を設定できます。これにより、参照された Component および Endpoint インスタンスに沿って、参照された Routes とともに CamelContext が自動的に開始されます。

- Camel スキーマの追加
- 次の 2 つの方法でルートを設定します。
  - Java コードの使用
  - Spring XML の使用

296.2. CAMEL スキーマの追加

Camel 1.x の場合、次の名前空間を使用する必要があります。

```
http://activemq.apache.org/camel/schema/spring
```

スキーマの場所は次のとおりです。

```
http://activemq.apache.org/camel/schema/spring/camel-spring.xsd
```

Camel を **schemaLocation** 宣言に追加する必要があります。

```
http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
```

したがって、XML ファイルは次のようになります。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
```

296.2.1. キャメルの使用: 名前空間

または、XML 宣言で camel XSD を参照できます。

```
xmlns:camel="http://camel.apache.org/schema/spring"
```

i. したがって、宣言は次のとおりです。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:camel="http://camel.apache.org/schema/spring"
      xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
```

i. camel: 名前空間接頭辞を使用すると、インライン名前空間宣言を省略できます。

```
<camel:camelContext id="camel5">
  <camel:package>org.apache.camel.spring.example</camel:package>
</camel:camelContext>
```

296.2.2. Spring を使用した高度な設定

詳細は、Spring を使用した CamelContext の高度な設定を 参照してください。

$ # Java コードの使用

Java コードを使用して RouteBuilder 実装を定義できます。これらは、Spring で Bean として定義し、camel コンテキストで参照できます。

296.2.3. <package> の使用

Camel は、特定のパッケージ内のルートの自動検出と初期化を可能にする強力な機能も提供します。これは、Spring コンテキスト定義の camel コンテキストにタグを追加し、RouteBuilder 実装を再帰的に検索するパッケージを指定することによって設定されます。1.X でこの機能を使用するには、検索するパッケージのコンマ区切りリストを指定する <package></package> タグが必要です。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <package>org.apache.camel.spring.config.scan.route</package>
</camelContext>
```

警告: パッケージ名を **org.apache.camel** またはこのサブパッケージとして指定する場合は注意してください。これにより、Camel は独自のパッケージでルートを検索し、問題が発生する可能性があります。

情報:*インスタンス化済みのクラスは無視されます*。<package> と <packageScan> は、Spring などによってすでに作成されているクラスをスキップします。そのため、ルートビルダーを Spring Bean タグとして定義すると、そのクラスはスキップされます。**<routeBuilder ref="theBeanId"/>** または **<contextScan>** 機能を使用して、これらの Bean を含めることができます。

296.2.4. <packageScan> の使用

Camel 2.0 ではこれが拡張され、Ant のようなパスマッチングを使用して、検出されたルートクラスを選択的に含めたり除外したりできるようになりました。Spring には、これは <packageScan/> タグを追加することで指定されます。タグには、1 つ以上の package 要素 (1.x と同様) を含める必要があり、オプションで、検出されたクラスの完全修飾名に適用されるパターンを指定する 1 つ以上の includes または excludes 要素を含める必要があります。例えば

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <packageScan>
    <package>org.example.routes</package>
    <excludes>**.*Excluded*</excludes>
    <includes>**.*</includes>
  </packageScan>
</camelContext>
```

exclude パターンは、include パターンの前に適用されます。include パターンまたは exclude パターンが定義されていない場合、パッケージで検出されたすべての Route クラスが返されます。

上記の例では、camel はすべての org.example.routes パッケージと RouteBuilder クラスのサブパッケージをスキャンします。スキャンで 2 つの RouteBuilders が見つかったとします。1 つは org.example.routes にある MyRoute、もう 1 つはサブパッケージ excluded にある MyExcludedRoute です。各クラスの完全修飾名が抽出され (org.example.routes.MyRoute、org.example.routes.excluded.MyExcludedRoute)、include および exclude パターンが適用されます。

exclude パターン ***.*Excluded は、** fqcn 'org.example.routes.excluded.MyExcludedRoute' に一致し、camel がそれを初期化することを拒否します。

内部では、これは Spring の AntPatternMatcher 実装を使用しており、次のように一致します。

```
? matches one character
* matches zero or more characters
** matches zero or more segments of a fully qualified name
```

以下に例を示します。

***.*Excluded** は、org.simple.Excluded、org.apache.camel.SomeExcludedRoute、または org.example.RouteWhichIsExcluded に一致します。

***.??included** は org.simple.IncludedRoute、org.simple.Excluded と一致しますが、org.simple.PrecludedRoute とは一致しません

296.2.5. contextScan の使用

Camel 2.4 以降で利用可能

Camel がコンテナーコンテキスト (ルートビルダーインスタンスの Spring **ApplicationContext** など) をスキャンできるようにすることができます。これにより、Spring の **<component-scan>** 機能を使用して、スキャンプロセスで Spring によって作成された RouteBuilder インスタンスを Camel にピックアップさせることができます。

これにより、Spring **@Component** を使用してルートにアノテーションを付け、それらのルートを Camel に含めることができます。

```
@Component
public class MyRoute extends SpringRouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start").to("mock:result");
    }
}
```

上記の **<packageScan>** ドキュメントで説明されているように、inclusion と exclusion に ANT スタイルを使用することもできます。

296.3. 他の XML ファイルからルートをインポートする方法

Camel 2.3 の時点で利用可能

Xml 設定 を使用して Camel でルートを定義する場合、他の XML ファイルでいくつかのルートを定義したい場合があります。たとえば、多くのルートがあり、一部のルートが個別の XML ファイルにあると、アプリケーションを維持するのに役立つ場合があります。必要に応じて簡単にインポートできる、共通の再利用可能なルートを他の XML ファイルに保存することもできます。

Camel 2.3 では、新しい **<routeContext/>** タグで **<camelContext/>** の外側でルートを定義できるようになりました。

注意: <routeContext> を使用すると、それらは分離され、既存の <onException>、<intercept>、<dataFormats>、および <camelContext> で定義された同様のクロスカット機能を再利用できません。つまり、<routeContext> は現在分離されています。これは Camel 3.x で変更される可能性があります。

たとえば、次のように、いくつかのルートを含む **myCoolRoutes.xml** という名前のファイルを作成できます。

myCoolRoutes.xml

次に、CamelContext を含む XML ファイルで、Spring を使用して **myCoolRoute.xml** ファイルをインポートできます。
<camelContext/> 内では、以下に示すように、ID で **<routeContext/>** を参照できます。

また、CamelContext 内にルートを持ち、RouteContext で外部化することもできます。

<routeContextRef/> は 好きなだけ持つことができます。

再利用可能なルート

<routeContext/> で定義されたルートは、複数の **<camelContext/>** で再利用できます。ただし、再利用されるのはその定義だけです。実行時に、各 CamelContext は定義に基づいてルートの独自のインスタンスを作成します。

296.3.1. テスト時間の除外。

テスト時に、テストシナリオに適用できない、または有用でない一致するルートを初期化から選択的に除外できることが望ましい場合がよくあります。たとえば、Spring コンテキストファイル routes-context.xml と、org.example.routes パッケージ内の 3 つのルートビルダー RouteA、RouteB、および RouteC を使用できます。packageScan 定義は、これら 3 つのルートすべてを検出して初期化します。

RouteC はテストシナリオには適用できず、テスト中に多くのノイズが発生するとします。この特定のテストからこのルートを除外できると便利です。これを可能にするために、SpringTestSupport クラスが変更されました。単一のクラスまたはクラスの配列を除外するためにオーバーライドできる 2 つのメソッド (excludedRoute および excludeRoutes) を提供します。

```
public class RouteAandRouteBOnlyTest extends SpringTestSupport {
    @Override
    protected Class excludeRoute() {
        return RouteC.class;
    }
}
```

spring による camelContext 初期化にフックして MyExcludedRouteBuilder.class を除外するには、spring コンテキストの作成をインターセプトする必要があります。createApplicationContext をオーバーライドして Spring コンテキストを作成する場合、getRouteExclusiveApplicationContext() メソッドを呼び出して、除外を処理する特別な親 Spring コンテキストを提供します。

```
@Override
protected AbstractXmlApplicationContext createApplicationContext() {
    return new ClassPathXmlApplicationContext(new String[] {"routes-context.xml"},
getRouteExcludingApplicationContext());
}
```

RouteC は初期化から除外されます。同様に、RouteC のみをテストする別のテストでは、RouteB と RouteA をオーバーライドして除外できます。

```
@Override
protected Class[] excludeRoutes() {
    return new Class[]{RouteA.class, RouteB.class};
}
```

296.4. SPRING XML の使用

次の 例 のように、Spring 2.0 XML 設定を使用してルートの Xml 設定を指定できます。

296.5. コンポーネントとエンドポイントの設定

この例 の次のように、Spring XML で Component または Endpoint インスタンスを設定できます。

これにより、何らかの名前 (上記の例では activemq) を使用してコンポーネントを設定でき、次に **activemq:queue:|topic:destinationName** を使用してコンポーネントを参照できます。これは、SpringCamelContext が、エンドポイント URI に使用するスキーム名のスプリングコンテキストからコ

ンポーネントを遅延フェッチすることによって機能します。

詳細は、エンドポイントとコンポーネントの設定を 参照してください。

296.6. CAMELCONTEXTAWARE

POJO に CamelContext を注入したい場合は、CamelContextAware インターフェイス を実装するだけです。次に、Spring が POJO を作成すると、CamelContext が POJO に注入されます。さらなるインジェクションについては、Bean 統合 も参照してください。

296.7. 統合テスト

Spring トランザクションを使用してテストするときにルートがハングしないようにするには、Transactional Client の Spring Integration Testing に関するメモを参照してください。

296.8. その他の参考資料

- Spring JMS チュートリアル
- 新しい Spring ベースの Camel Route の作成
- Spring の例
- Xml Reference
- Spring を使用した CamelContext の高度な設定
- 他の XML ファイルからルートをインポートする方法

第297章 SPRING BATCH コンポーネント

Camel バージョン 2.10 以降で利用可能

spring-batch: コンポーネントおよびサポートクラスは、Camel と Spring Batch インフラストラクチャー間の統合ブリッジを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-batch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

297.1. URI 形式

```
spring-batch:jobName[?options]
```

jobName は、Camel レジストリーにある Spring Batch ジョブの名前を表します。あるいは、JobRegistry が提供されている場合は、代わりにそれを使用してジョブを検索します。

警告: このコンポーネントはプロデューサーエンドポイントの定義にのみ使用できます。つまり、**from()** ステートメントで Spring Batch コンポーネントを使用することはできません。

297.2. オプション

Spring Batch コンポーネントは、以下にリストされている 3 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
jobLauncher (producer)	使用する JobLauncher を明示的に指定します。		JobLauncher
jobRegistry (producer)	使用する JobRegistry を明示的に指定します。		JobRegistry
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spring Batch エンドポイントは、URI 構文を使用して設定されます。

```
spring-batch:jobName
```

パスおよびクエリーパラメーターを使用します。

297.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
jobName	**必須** レジストリーにある Spring Batch ジョブの名前。		String

297.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
jobFromHeader (producer)	jobName を URI ではなくヘッダーから取得する必要があるかどうかを明示的に定義します。	false	boolean
jobLauncher (producer)	使用する JobLauncher を明示的に指定します。		JobLauncher
jobRegistry (producer)	使用する JobRegistry を明示的に指定します。		JobRegistry
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

297.3. 使用方法

Spring Batch コンポーネントがメッセージを受信すると、ジョブの実行がトリガーされます。ジョブは、次のアルゴリズムに従って解決された **org.springframework.batch.core.launch.JobLaucher** インスタンスを使用して実行されます。

- **JobLauncher** がコンポーネントに手動で設定されている場合は、それを使用します。
- コンポーネントに **jobLauncherRef** オプションが設定されている場合、指定された名前の **JobLauncher** の Camel レジストリーを検索します。 **非推奨であり、Camel 3.0 で削除されます!**
- Camel Registry に **jobLauncher** 名で **JobLauncher が** 登録されている場合は、それを使用します。
- 上記のいずれの手順でも **JobLauncher** を解決できず、Camel Registry に **JobLauncher** インスタンスが 1 つだけある場合は、それを使用します。

メッセージで見つかったすべてのヘッダーは、ジョブパラメーターとして **JobLauncher** に渡されます。**String**、**Long**、**Double**、および **java.util.Date** の値は **org.springframework.batch.core.JobParametersBuilder** にコピーされ、その他のデータ型は String に変換されます。

297.4. 例

Spring Batch ジョブの実行をトリガーする:

```
from("direct:startBatch").to("spring-batch:myJob");
```

JobLauncher を明示的に設定して Spring Batch ジョブの実行をトリガーする。

```
from("direct:startBatch").to("spring-batch:myJob?jobLauncherRef=myJobLauncher");
```

Camel 2.11.1 以降、**JobLauncher** によって返される **JobExecution** インスタンスは、**SpringBatchProducer** によって出力メッセージとして転送されます。**JobExecution** インスタンスを使用して、Spring Batch API を直接使用していくつかの操作を実行できます。

```
from("direct:startBatch").to("spring-batch:myJob").to("mock:JobExecutions");
...
MockEndpoint mockEndpoint = ...;
JobExecution jobExecution =
mockEndpoint.getExchanges().get(0).getIn().getBody(JobExecution.class);
BatchStatus currentJobStatus = jobExecution.getStatus();
```

297.5. サポートクラス

コンポーネントとは別に、Camel Spring Batch は、Spring Batch インフラストラクチャーにフックするために使用できるサポートクラスも提供します。

297.5.1. CamelItemReader

CamelItemReader を 使用して、Camel インフラストラクチャーから直接バッチデータを読み取ることができます。

たとえば、以下のスニペットは、JMS キューからデータを読み取るように Spring Batch を設定します。

```
<bean id="camelReader"
class="org.apache.camel.component.spring.batch.support.CamelItemReader">
  <constructor-arg ref="consumerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="camelReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.2. CamelItemWriter

CamelItemWriter は **CamelItemReader** と同様の目的を持っていますが、処理されたデータのチャンクを書き込む専用です。

たとえば、以下のスニペットは、JMS キューからデータを読み取るように Spring Batch を設定します。

```
<bean id="camelwriter" class="org.apache.camel.component.spring.batch.support.CamelItemWriter">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="camelwriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.3. CamelItemProcessor

CamelItemProcessor は、Spring Batch **org.springframework.batch.item.ItemProcessor** インターフェイスの実装です。後者の実装は、Request Reply パターン を中継して、バッチアイテムの処理を Camel インフラストラクチャーに委譲します。処理するアイテムは、メッセージの本文として Camel エンドポイントに送信されます。

たとえば、以下のスニペットは、Direct エンドポイント と Simple 式言語 を使用して、バッチアイテムの簡単な処理を実行します。

```
<camel:camelContext>
  <camel:route>
    <camel:from uri="direct:processor"/>
    <camel:setExchangePattern pattern="InOut"/>
    <camel:setBody>
      <camel:simple>Processed ${body}</camel:simple>
    </camel:setBody>
  </camel:route>
</camel:camelContext>

<bean id="camelProcessor"
class="org.apache.camel.component.spring.batch.support.CamelItemProcessor">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="direct:processor"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" processor="camelProcessor" commit-
interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.4. CamelJobExecutionListener

CamelJobExecutionListener は、ジョブ実行イベントを Camel エンドポイントに送信する **org.springframework.batch.core.JobExecutionListener** インターフェイスの実装です。

Spring Batch によって生成された **org.springframework.batch.core.JobExecution** インスタンスは、メッセージのボディーとして送信されます。コールバックの前後を区別するには、**SPRING_BATCH_JOB_EVENT_TYPE** ヘッダーを **BEFORE** または **AFTER** の 値に設定します。

以下のスニペットの例では、Spring Batch ジョブ実行イベントを JMS キューに送信します。

```
<bean id="camelJobExecutionListener"
class="org.apache.camel.component.spring.batch.support.CamelJobExecutionListener">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:batchEventsBus"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
  <batch:listeners>
    <batch:listener ref="camelJobExecutionListener"/>
  </batch:listeners>
</batch:job>
```

297.6. SPRING CLOUD

Camel 2.19 以降で利用可能

Spring Cloud コンポーネント

Maven ユーザーは、このコンポーネントを使用するために **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
   <groupId>org.apache.camel</groupId>
   <artifactId>camel-spring-cloud</artifactId>
   <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>
```

camel-spring-cloud jar には **spring.factories** ファイルが付属しているため、その依存関係をクラスパスに追加するとすぐに、Spring Boot が自動的に Camel を自動設定します。

297.6.1. Camel Spring Cloud Starter

Camel 2.19 以降で利用可能

スターターを使用するには、Spring Boot pom.xml ファイルに以下を追加します。

```
<dependency>
   <groupId>org.apache.camel</groupId>
   <artifactId>camel-spring-cloud-starter</artifactId>
   <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>
```

297.7. SPRING CLOUD NETFLIX

Camel 2.19 以降で利用可能

Spring Cloud Netflix コンポーネントは Camel Cloud と Spring Cloud Netflix を橋渡しするため、Camel で Spring Cloud Netflix サービスディスカバリーとロードバランス機能を活用したり、Spring Cloud Netflix のリボンロードバランサーの ServerList ソースとして Camel Service Discovery 実装を使用したりできます。

Maven ユーザーは、このコンポーネントを使用するために **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-cloud-netflix</artifactId>
  <version>${camel.version}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-spring-cloud-netflix jar には **spring.factories** ファイルが付属しているため、その依存関係をクラスパスに追加するとすぐに、Spring Boot が自動的に Camel を自動設定します。

次のプロパティーを使用して Camel Spring Cloud Netflix を無効にできます。

```
# Enable/Disable the whole integration, default true
camel.cloud.netflix = true

# Enable/Disable the integration with Ribbon, default true
camel.cloud.netflix.ribbon = true
```

297.8. SPRING CLOUD NETFLIX STARTER

Camel 2.19 以降で利用可能

スターターを使用するには、Spring Boot pom.xml ファイルに以下を追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-cloud-netflix-starter</artifactId>
  <version>${camel.version}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第298章 SPRING EVENT コンポーネント

Camel バージョン 1.4 以降で利用可能

spring-event: コンポーネントは、Spring **ApplicationEvent** オブジェクトへのアクセスを提供します。これにより、**ApplicationEvent** オブジェクトを Spring **ApplicationContext** に発行したり、それらを消費したりできます。その後、Enterprise Integration Patterns を使用して、メッセージフィルター などを処理できます。

298.1. URI 形式

```
spring-event://default[?options]
```

現時点では、このコンポーネントにはオプションがありません。これは将来のリリースで簡単に変更される可能性があるため、もう一度確認してください。

298.2. SPRING EVENT オプション

Spring Event コンポーネントにはオプションがありません。

Spring Event エンドポイントは、URI 構文を使用して設定されます。

```
spring-event:name
```

パスおよびクエリーパラメーターを使用します。

298.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	エンドポイントの名前		String

298.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

298.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第299章 SPRING 統合コンポーネント

Camel バージョン 1.4 以降で利用可能

spring-integration: コンポーネントは、Camel コンポーネントが spring integration endpoint と通信するためのブリッジを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-integration</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

299.1. URI 形式

```
spring-integration:defaultChannelName[?options]
```

defaultChannelName は、Spring Integration Spring コンテキストで使用されるデフォルトのチャネル名を表します。これは、Spring Integration コンシューマーの **inputChannel** 名および Spring Integration プロバイダーの **outputChannel** 名と等しくなります。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

299.2. オプション

Spring Integration コンポーネントにはオプションがありません。

Spring Integration エンドポイントは、URI 構文を使用して設定されます。

```
spring-integration:defaultChannel
```

パスおよびクエリーパラメーターを使用します。

299.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
defaultChannel	**必須** Spring Integration Spring コンテキストで使用されるデフォルトのチャネル名。これは、Spring Integration コンシューマーの inputChannel 名および Spring Integration プロバイダーの outputChannel 名と等しくなります。		String

299.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
inOut (Common)	Spring 統合エンドポイントが使用する交換パターン。inOut=true の場合、Spring Integration Message ヘッダーから、またはエンドポイントで設定された応答チャネルが期待されます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
inputChannel (consumer)	このエンドポイントが Spring 統合から消費したい Spring 統合入力チャネル名。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
outputChannel (producer)	Spring 統合にメッセージを送信するために使用される Spring 統合出力チャネル名。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

299.3. 使用方法

Spring 統合コンポーネントは、Spring 統合の入力チャネルと出力チャネルを介して、Camel エンドポイントを Spring 統合エンドポイントに接続するブリッジです。このコンポーネントを使用して、Camel メッセージを Spring Integration エンドポイントに送信したり、Spring 統合エンドポイントから Camel ルーティングコンテキストでメッセージを受信したりできます。

299.4. 例

299.4.1. Spring 統合エンドポイントの使用

次のように、URI を使用して Spring 統合エンドポイントを設定できます。

または、Spring 統合チャネル名を直接使用します。

299.4.2. ソースとターゲットのアダプター

Spring 統合は、Spring 統合のソースアダプターとターゲットアダプターも提供します。これらは、Spring 統合チャネルから Camel エンドポイントに、または Camel エンドポイントから Spring 統合チャネルにメッセージをルーティングできます。

この例では、次の名前空間を使用します。

次のように、ソースまたはターゲットを Camel エンドポイントにバインドできます。

299.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

299.6. SPRING の JAVA 設定

Spring は、XML Config を使用して Bean を接続することから始まりました。しかし、XML の使用を好まず、むしろ Java コードを使用したいという人もいます。その結果、Spring JavaConfig プロジェクトとともに Guice が作成されました。

Camel では、XML または Java 設定アプローチのいずれかを使用できます。どちらがいいかは自由です。

299.6.1. Spring Java Config の使用

Camel プロジェクトで Spring Java Config を使用するには、pom.xml に以下を追加するのが最も簡単です。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-javaconfig</artifactId>
  <version>${camel-version}</version>
</dependency>
```

これにより、Spring JavaConfig ライブラリーに依存関係が追加され、Spring 内で Camel を設定するためのいくつかのヘルパークラスが追加されます。

このライブラリーは完全にオプションであることに注意してください。Java Config を使用して自分で Camel を配線するだけです。

299.6.2. 設定

Camel で JavaConfig を使用する最も一般的なケースは、ルーターが使用するルートのリストを定義して設定を作成することです。

```
@Configuration
```

```
public class MyRouteConfiguration extends CamelConfiguration {

    @Autowire
    private MyRouteBuilder myRouteBuilder;

    @Autowire
    private MyAnotherRouteBuilder myAnotherRouteBuilder;

    @Override
    public List<RouteBuilder> routes() {
        return Arrays.asList(myRouteBuilder, myAnotherRouteBuilder);
    }

}
```

Camel 2.13.0 以降では、**routes()** 定義をスキップして、Spring コンテキストにある **RouteBuilder** インスタンスにフォールバックできます。

```
@Configuration
@ComponentScan("com.example.routes")
public class MyRouteConfiguration extends CamelConfiguration {
}
```

299.6.3. テスト

Camel 2.11.0 以降、**CamelSpringDelegatingTestContextLoader** で **CamelSpringJUnit4ClassRunner** を使用できます。これは、Java Config と Camel の統合をテストするための推奨される方法です。

RouteBuilder インスタンスのコレクションを作成する場合は、**CamelConfiguration** ヘルパークラスから派生させて、**routes()** メソッドを実装します。(Camel 2.13.0 以降) **routes()** メソッドをオーバーライドしない場合、**CamelConfiguration** は Spring コンテキストで使用可能なすべての **RouteBuilder** インスタンスを使用することに注意してください。

Java Config を使用した次の例は、 Java Config と Camel 2.10 以前との統合をテストする方法を示しています。**JavaConfigContextLoader** は非推奨であり、**CamelSpringDelegatingTestContextLoader** に代わって Camel の将来のバージョンで削除される可能性があることに注意してください。

@ContextConfiguration アノテーションは、使用する設定として **ContextConfig** クラスをロードするように Spring Testing フレームワークに指示します。このクラスは、CamelContext を設定し、作成した RouteBuilder を登録するヘルパー Spring Java Config クラスである **SingleRouteCamelConfiguration** から派生します。

第300章 SPRING LDAP コンポーネント

Camel バージョン 2.11 以降で利用可能

spring-ldap: コンポーネントは、Spring LDAP の Camel ラッパーを提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-ldap</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

300.1. URI 形式

```
spring-ldap:springLdapTemplate[?options]
```

springLdapTemplate は、Spring LDAP テンプレート Bean の名前です。この Bean では、LDAP アクセス用の URL と認証情報を設定します。

300.2. オプション

Spring LDAP コンポーネントにはオプションがありません。

Spring LDAP エンドポイントは、URI 構文を使用して設定されます。

```
spring-ldap:templateName
```

パスおよびクエリーパラメーターを使用します。

300.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
templateName	**必須** Spring LDAP テンプレート Bean の名前		String

300.2.2. クエリーパラメーター(3 個のパラメーター):

名前	説明	デフォルト	タイプ
operation (producer)	**必須** 実行する LDAP 操作。		LdapOperation
scope (producer)	検索操作の範囲。	subtree	String

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

300.3. 使用方法

コンポーネントはプロデューサーエンドポイントのみをサポートします。コンシューマーエンドポイントを作成しようとすると、**UnsupportedOperationException** が発生します。
メッセージの本文はマップ (**java.util.Map** のインスタンス) でなければなりません。ContextSource の設定でベース DN が指定されていない限り、このマップには、実行する LDAP 操作のルートノードを指定するキー **dn** (function_driven 操作には不要) を持つエントリーが少なくとも含まれている必要があります。マップの他のエントリーは操作固有です (以下を参照)。

メッセージの本文は、**BIND** と **unbind** 操作で変更されません。**search** および **function_driven** 操作の場合、本文は検索結果に設定されます。http://static.springsource.org/spring-ldap/site/apidocs/org/springframework/ldap/core/LdapTemplate.html#search%28java.lang.String,%20ja
を参照してください

300.3.1. Search

メッセージボディーには、キー **filter** を含むエントリーが必要です。値は、有効な LDAP フィルターを表す **String** である必要があります。http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Search_and_Compare を参照してください。

300.3.2. バインド

メッセージボディーには、キー **attributes** を持つエントリーが必要です。値は、javax.naming.directory.Attributes のインスタンスである必要があります。このエントリーは、作成する LDAP ノードを指定します。

300.3.3. Unbind

これ以上のエントリーは必要ありません。指定された **dn** を持つノードが削除されます。

300.3.4. 認証

メッセージボディーには、キー **filter** および **password** を含むエントリーが必要です。値は、それぞれ有効な LDAP フィルターとユーザーパスワードを表す **String** のインスタンスである必要があります。

300.3.5. 属性の変更

メッセージボディーには、キー **modificationItems** を持つエントリーが必要です。値は、タイプ javax.naming.directory.ModificationItem の任意の配列のインスタンスである必要があります

300.3.6. ファンクションドリブン

メッセージボディーには、キー **function** および **request** を含むエントリーが必要です。**function** 値は **java.util.function.BiFunction<L, Q, S>** 型である必要があります。**L** 型パラメーターは、型

org.springframework.ldap.core.LdapOperations でなければなりません。**request** 値は、**function** 内の **Q** 型パラメーターと同じ型である必要があり、**function** 内で呼び出される LdapTemplate メソッドによって想定されるパラメーターをカプセル化する必要があります。**S** 型パラメーターは、呼び出された LdapTemplate メソッドによって返されるレスポンスの型を表します。この操作により、上記の操作でカバーされない LdapTemplate メソッドを動的に呼び出すことができます。

キーの定義

スペルミスを避けるために、次の定数が **org.apache.camel.springldap.SpringLdapProducer** で定義されています。

- public static final String DN = "dn"
- public static final String FILTER = "filter"
- public static final String ATTRIBUTES = "attributes"
- public static final String PASSWORD = "password";
- public static final String MODIFICATION_ITEMS = "modificationItems";
- public static final String FUNCTION = "function";
- public static final String REQUEST = "request";

第301章 SPRING REDIS コンポーネント

Camel バージョン 2.11 以降で利用可能

このコンポーネントを使用すると、Redis からメッセージを送受信できます。Redis は、キーに文字列、ハッシュ、リスト、セット、および並べ替えられたセットを含めることができる高度なキー値ストアです。さらに、アプリ間通信のための pub/sub 機能を提供します。
Camel は、コマンドを実行するためのプロデューサー、pub/sub メッセージをサブスクライブするためのコンシューマー、重複メッセージをフィルター処理するためのべき等リポジトリーを提供します。

情報:*前提条件* このコンポーネントを使用するには、Redis サーバーが実行されている必要があります。

301.1. URI 形式

```
spring-redis://host:port[?options]
```

URI には、**?options=value&option2=value&...** という形式でクエリーオプションを追加できます。

301.2. URI オプション

Spring Redis コンポーネントにはオプションがありません。

Spring Redis エンドポイントは、URI 構文を使用して設定されます。

```
spring-redis:host:port
```

パスおよびクエリーパラメーターを使用します。

301.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** Redis サーバーが実行されているホスト。		String
port	**必須** Redis サーバーのポート番号。		Integer

301.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
channels (Common)	サブスクライブするトピック名または名前パターンのリスト。複数の値はコンマで区切ることができます。		String

名前	説明	デフォルト	タイプ
command (common)	デフォルトのコマンド。メッセージヘッダーでオーバーライドできます。コンシューマーは次のコマンドのみをサポートすることに注意してください: PSUBSCRIBE および SUBSCRIBE	SET	コマンド
connectionFactory (common)	使用する事前設定された RedisConnectionFactory インスタンスへの参照。		RedisConnectionFactory
redisTemplate (Common)	使用する事前設定された RedisTemplate インスタンスへの参照。		RedisTemplate
serializer (Common)	使用する事前設定された RedisSerializer インスタンスへの参照。		RedisSerializer
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
listenerContainer (consumer)	使用する事前設定された RedisMessageListenerContainer インスタンスへの参照。		RedisMessageListener コンテナー
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

301.3. 使用方法

https://github.com/apache/camel/tree/master/components/camel-spring-redis/src/test/java/org/apache/camel/component/redis で入手できる単体テストも参照してください。

301.3.1. Redis プロデューサーによって評価されるメッセージヘッダー

プロデューサーはサーバーにコマンドを発行し、各コマンドには特定のタイプの異なるパラメーターセットがあります。コマンド実行の結果は、メッセージボディーで返されます。

ハッシュコマンド	説明	パラメーター	結果
HSET	ハッシュフィールドの文字列値を設定します。	CamelRedis.Key (文字列), CamelRedis.Field (文字列), CamelRedis.Value (オブジェクト型)	void
HGET	ハッシュフィールドの値を取得します。	CamelRedis.Key (文字列), CamelRedis.Field (文字列)	String
HSETNX	フィールドが存在しない場合にのみ、ハッシュフィールドの値を設定します。	CamelRedis.Key (文字列), CamelRedis.Field (文字列), CamelRedis.Value (オブジェクト型)	void
HMSET	複数のハッシュフィールドを複数の値に設定します。	CamelRedis.Key (String), CamelRedis.Values(Map<String, Object>)	void
HMGET	指定されたすべてのハッシュフィールドの値を取得します。	CamelRedis.Key (文字列), CamelRedis.Fields (Collection<String>)	Collection<Object>
HINCRBY	指定された数だけハッシュフィールドの整数値をインクリメントします。	CamelRedis.Key (文字列), CamelRedis.Field (文字列), CamelRedis.Value(Long 型)	Long

ハッシュコマンド	説明	パラメーター	結果
HEXISTS	ハッシュフィールドが存在するかどうかを判断します。	CamelRedis.Key (文字列), CamelRedis.Field (文字列)	Boolean
HDEL	1 つ以上のハッシュフィールドを削除します。	CamelRedis.Key (文字列), CamelRedis.Field (文字列)	void
HLEN	ハッシュ内のフィールド数を取得します。	CamelRedis.Key (文字列)	Long
HKEYS	ハッシュ内のすべてのフィールドを取得します。	CamelRedis.Key (文字列)	Set<String>
HVALS	ハッシュ内のすべての値を取得します。	CamelRedis.Key (文字列)	Collection<Object>
HGETALL	ハッシュ内のすべてのフィールドと値を取得します。	CamelRedis.Key (文字列)	Map<String, Object>

リストコマンド	説明	パラメーター	結果
RPUSH	1 つまたは複数の値をリストに追加します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Long
RPUSHX	リストが存在する場合にのみ、リストに値を追加します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Long

リストコマンド	説明	パラメーター	結果
LPUSH	リストの先頭に1つまたは複数の値を追加します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Long
LLEN	リストの長さを取得します。	CamelRedis.Key (文字列)	Long
LRANGE	リストから要素の範囲を取得します。	CamelRedis.Key (文字列), CamelRedis.Start(Long 型), CamelRedis.End(Long 型)	List<Object>
LTRIM	リストを指定された範囲にトリミングします。	CamelRedis.Key (文字列), CamelRedis.Start(Long 型), CamelRedis.End(Long 型)	void
LINDEX	インデックスでリストから要素を取得します。	CamelRedis.Key (文字列), CamelRedis.Index(Long 型)	String
LINSERT	リスト内の別の要素の前後に要素を挿入します。	CamelRedis.Key (文字列)、CamelRedis.Value (オブジェクト)、CamelRedis.Pivot (文字列)、CamelRedis.Position (文字列)	Long
LSET	リスト内の要素の値をそのインデックスで設定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Index(Long 型)	void

リストコマンド	説明	パラメーター	結果
LREM	リストから要素を削除します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Count(Long 型)	Long
LPOP	リストの最初の要素を削除して取得します。	CamelRedis.Key (文字列)	Object
RPOP	リストの最後の要素を削除して取得します。	CamelRedis.Key (文字列)	String
RPOPLPUSH	リストの最後の要素を削除し、それを別のリストに追加して返します。	CamelRedis.Key (文字列), CamelRedis.Destination (文字列)	Object
BRPOPLPUSH	リストから値をポップし、それを別のリストにプッシュして返します。または利用可能になるまでブロックします。	CamelRedis.Key (文字列), CamelRedis.Destination (文字列), CamelRedis.Timeout(Long 型)	Object
BLPOP	リストの最初の要素を削除して取得するか、使用可能になるまでブロックします。	CamelRedis.Key (文字列), CamelRedis.Timeout(Long 型)	Object

リストコマンド	説明	パラメーター	結果
BRPOP	リストの最後の要素を削除して取得するか、使用可能になるまでブロックします。	CamelRedis.Key (文字列), CamelRedis.Timeout(Long型)	String

コマンドの設定	説明	パラメーター	結果
SADD	1つまたは複数のメンバーをセットに追加します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Boolean
SMEMBERS	セット内のすべてのメンバーを取得します。	CamelRedis.Key (文字列)	Set<Object>
SREM	セットから1つまたは複数のメンバーを削除します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Boolean
SPOP	セットからランダムなメンバーを削除して返します。	CamelRedis.Key (文字列)	String
SMOVE	メンバーをあるセットから別のセットに移動します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Destination (文字列)	Boolean
SCARD	セット内のメンバーの数を取得します。	CamelRedis.Key (文字列)	Long

コマンドの設定	説明	パラメーター	結果
SISMEMBER	指定された値がセットのメンバーであるかどうかを判別します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Boolean
SINTER	複数のセットを交差します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列)	Set<Object>
SINTERSTORE	複数のセットを交差させ、結果のセットをキーに格納します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列), CamelRedis.Destination (文字列)	void
SUNION	複数のセットを追加します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列)	Set<Object>
SUNIONSTORE	複数のセットを追加し、結果のセットをキーに保存します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列), CamelRedis.Destination (文字列)	void
SDIFF	複数のセットを減算します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列)	Set<Object>
SDIFFSTORE	複数のセットを減算し、結果のセットをキーに格納します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列), CamelRedis.Destination (文字列)	void

コマンドの設定	説明	パラメーター	結果
SRANDMEMBER	セットから1つまたは複数のランダムメンバーを取得します。	CamelRedis.Key (文字列)	String

順序集合コマンド	説明	パラメーター	結果
ZADD	並べ替えられたセットに1つ以上のメンバーを追加するか、すでに存在する場合はそのスコアを更新します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Score (Double 型)	Boolean
ZRANGE	並べ替えられたセット内のメンバーの範囲をインデックスで返します。	CamelRedis.Key (文字列)、CamelRedis.Start (ロング)、CamelRedis.End (ロング)、CamelRedis.WithScore (ブール値)	Object
ZREM	ソート済みセットから1つ以上のメンバーを削除します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Boolean
ZINCRBY	ソート済みセットのメンバーのスコアをインクリメントします。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Increment (Double 型)	double

順序集合コマンド	説明	パラメーター	結果
ZRANK	並べ替えられたセット内のメンバーのインデックスを決定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Long
ZREVRANK	スコアが高いものから低いものに並べられた、並べ替えられたセット内のメンバーのインデックスを決定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Long
ZREVRANGE	並べ替えられたセット内のメンバーの範囲をインデックス別に返します。スコアは高いものから低いものへと並べられます。	CamelRedis.Key (文字列)、CamelRedis.Start (ロング)、CamelRedis.End (ロング)、CamelRedis.WithScore (ブール値)	Object
ZCARD	ソート済みセットのメンバー数を取得します。	CamelRedis.Key (文字列)	Long
ZCOUNT	指定された値内のスコアを持つ並べ替えられたセットのメンバーをカウントします。	CamelRedis.Key (文字列), CamelRedis.Min (Double 型), CamelRedis.Max (Double 型)	Long
ZRANGEBYSCORE	ソートされたセット内のメンバーの範囲をスコア別に返します。	CamelRedis.Key (文字列), CamelRedis.Min (Double 型), CamelRedis.Max (Double 型)	Set<Object>

順序集合コマンド	説明	パラメーター	結果
ZREVRANGEBYSCORE	スコアが高いものから低いものへと並べられたスコアで、ソートされたセット内のメンバーの範囲を返します。	CamelRedis.Key (文字列), CamelRedis.Min (Double 型), CamelRedis.Max (Double 型)	Set<Object>
ZREMRANGEBYRANK	指定されたインデックス内の並べ替えられたセットのすべてのメンバーを削除します。	CamelRedis.Key (文字列), CamelRedis.Start(Long 型), CamelRedis.End(Long 型)	void
ZREMRANGEBYSCORE	指定されたスコア内の並べ替えられたセットのすべてのメンバーを削除します。	CamelRedis.Key (文字列), CamelRedis.Start(Long 型), CamelRedis.End(Long 型)	void
ZUNIONSTORE	複数のソート済みセットを追加し、結果のソート済みセットを新しいキーに保存します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列), CamelRedis.Destination (文字列)	void
ZINTERSTORE	複数の並べ替えられたセットを交差させ、結果の並べ替えられたセットを新しいキーに格納します。	CamelRedis.Key (文字列), CamelRedis.Keys (文字列), CamelRedis.Destination (文字列)	void

文字列コマンド	説明	パラメーター	結果
SET	キーの文字列値を設定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	void
GET	キーの値を取得する	CamelRedis.Key (文字列)	Object
STRLEN	キーに格納されている値の長さを取得します。	CamelRedis.Key (文字列)	Long
APPEND	キーに値を追加します。	CamelRedis.Key (文字列), CamelRedis.Value (文字列)	Integer
SETBIT	キーに格納されている文字列値のオフセットのビットを設定またはクリアします。	CamelRedis.Key (文字列), CamelRedis.Offset(Long 型), CamelRedis.Value (ブール型)	void
GETBIT	キーに格納されている文字列値のオフセットのビット値を返します。	CamelRedis.Key (文字列), CamelRedis.Offset(Long 型)	Boolean
SETRANGE	指定されたオフセットから始まるキーの文字列の一部を上書きします。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Offset(Long 型)	void
GETRANGE	キーに格納されている文字列の部分文字列を取得します。	CamelRedis.Key (文字列), CamelRedis.Start(Long 型), CamelRedis.End(Long 型)	String

文字列コマンド	説明	パラメーター	結果
SETNX	キーが存在しない場合にのみ、キーの値を設定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Boolean
SETEX	キーの値と有効期限を設定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型), CamelRedis.Timeout(Long 型), SECONDS	void
DECRBY	指定された数だけキーの整数値を減らします。	CamelRedis.Key (文字列), CamelRedis.Value(Long 型)	Long
DECR	キーの整数値を 1 減らします。	CamelRedis.Key (文字列),	Long
INCRBY	指定された量だけキーの整数値を増やします。	CamelRedis.Key (文字列), CamelRedis.Value(Long 型)	Long
INCR	キーの整数値を 1 ずつ増やします。	CamelRedis.Key (文字列)	Long
MGET	指定されたすべてのキーの値を取得します。	CamelRedis.Fields (Collection<String>)	List<Object>
MSET	複数のキーを複数の値に設定します。	CamelRedis.Values(Map<String, Object>)	void
MSETNX	キーが存在しない場合にのみ、複数のキーを複数の値に設定します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	void

文字列コマンド	説明	パラメーター	結果
GETSET	キーの文字列値を設定し、古い値を返します。	CamelRedis.Key (文字列), CamelRedis.Value (オブジェクト型)	Object

キーコマンド	説明	パラメーター	結果
EXISTS	キーが存在するかどうかを判断します。	CamelRedis.Key (文字列)	Boolean
DEL	キーを削除します。	CamelRedis.Keys (String)	void
TYPE	キーに格納されている型を決定します。	CamelRedis.Key (文字列)	DataType
KEYS	指定されたパターンに一致するすべてのキーを見つけます。	CamelRedis.Pattern (String)	Collection<String>
RANDOMKEY	キースペースからランダムなキーを返します。	CamelRedis.Pattern (String), CamelRedis.Value (String)	String
RENAME	キーの名前を変更します。	CamelRedis.Key (文字列)	void
RENAMENX	新しいキーが存在しない場合にのみ、キーの名前を変更します。	CamelRedis.Key (文字列), CamelRedis.Value (文字列)	Boolean
EXPIRE	キーの存続時間を秒単位で設定します。	CamelRedis.Key (文字列), CamelRedis.Timeout(Long 型)	Boolean

キーコマンド	説明	パラメーター	結果
SORT	リスト、セット、またはソートされたセット内の要素をソートします。	CamelRedis.Key (文字列)	List<Object>
PERSIST	キーから有効期限を削除します。	CamelRedis.Key (文字列)	Boolean
EXPIREAT	キーの有効期限を UNIX タイムスタンプとして設定します。	CamelRedis.Key (文字列), CamelRedis.Timestamp(Long 型)	Boolean
PEXPIRE	キーの存続時間をミリ秒単位で設定します。	CamelRedis.Key (文字列), CamelRedis.Timeout(Long 型)	Boolean
PEXPIREAT	ミリ秒単位で指定された UNIX タイムスタンプとしてキーの有効期限を設定します。	CamelRedis.Key (文字列), CamelRedis.Timestamp(Long 型)	Boolean
TTL	キーの存続時間を取得します。	CamelRedis.Key (文字列)	Long
MOVE	キーを別のデータベースに移動します。	CamelRedis.Key (文字列), CamelRedis.Db (整数)	Boolean

その他のコマンド	説明	パラメーター	結果

その他のコマンド	説明	パラメーター	結果
MULTI	トランザクションブロックの開始をマークします。	none	void
DISCARD	MULTI の後に発行されたすべてのコマンドを破棄します。	none	void
EXEC	MULTI の後に発行されたすべてのコマンドを実行します。	none	void
WATCH	指定されたキーを監視して、MULTI/EXEC ブロックの実行を決定します	CamelRedis.Keys (String)	void
UNWATCH	監視されているすべてのキーを忘れます。	none	void
ECHO	指定された文字列をエコーします。	CamelRedis.Value (String)	String
PING	サーバーに ping を実行します。	none	String
QUIT	接続を閉じます。	none	void
PUBLISH	チャンネルにメッセージを投稿します。	CamelRedis.Channel (文字列), CamelRedis.Message (オブジェクト型)	void

301.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-redis</artifactId>
  <version>${camel-version}</version>
</dependency>
```

${camel-version } は Camel の実際のバージョン (2.11 以降) に置き換える必要があります。

301.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第302章 SPRING SECURITY

Camel 2.3 の時点で利用可能

camel-spring-security コンポーネントは、Camel ルートにロールベースの承認を提供します。これは、Spring Security (以前の Acegi Security) によって提供される認証およびユーザーサービスを活用し、特定のプリンシパルによってルートを実行できるかどうかを制御する宣言型のロールベースのポリシーシステムを追加します。

Spring Security の認証および認可システムに慣れていない場合は、上記のリンク先の SpringSource Web サイトにある最新のリファレンスドキュメントを参照してください。

302.1. 認可ポリシーの作成

ルートへのアクセスは、**SpringSecurityAuthorizationPolicy** オブジェクトのインスタンスによって制御されます。ポリシーオブジェクトには、一連のエンドポイントを実行するために必要な Spring Security 権限 (ロール) の名前と、現在のプリンシパルにそのロールが割り当てられているかどうかを判断するために使用される Spring Security **AuthenticationManager** および **AccessDecisionManager** オブジェクトへの参照が含まれています。ポリシーオブジェクトは、Spring Bean として設定するか、Spring XML の **<authorizationPolicy>** 要素を使用して設定できます。

<authorizationPolicy> 要素には、次の属性を含めることができます。

名前	デフォルト値	説明
id	**null**	ルートでポリシーを参照するために使用される一意の Spring Bean 識別子 (必須)
access	**null**	アクセス Decision Manager に渡される Spring Security 権限名 (必須)
authenticationManager	**authenticationManager**	コンテキスト内の Spring Security **AuthenticationManager** オブジェクトの名前
accessDecisionManager	**accessDecisionManager**	コンテキスト内の Spring Security **AccessDecisionManager** オブジェクトの名前
authenticationAdapter	DefaultAuthenticationAdapter	Camel 2.4 **javax.security.auth.Subject** を Spring Security **Authentication** インスタンスに変換するために使用されるコンテキスト内の **camel-spring-security AuthenticationAdapter** オブジェクトの名前。

名前	デフォルト値	説明
useThreadSecurityContext	**true**	Exchange.AUTHENTICATION の下の In メッセージヘッダーで **javax.security.auth.Subject** が見つからない場合は、**Authentication** オブジェクトの Spring Security **SecurityContextHolder** を確認してください。
alwaysReauthenticate	**false**	true に設定すると、**SpringSecurityAuthorizationPolicy** は、ポリシーにアクセスするたびに常に **AuthenticationManager.authenticate()** を呼び出します。

302.2. CAMEL ルートへのアクセスの制御

このコンポーネントを使用するには、Spring Security **AuthenticationManager** および **AccessDecisionManager** が必要です。Spring Security 名前空間を使用して Spring XML でこれらのオブジェクトを設定する方法の例を次に示します。

基盤となるセキュリティーオブジェクトが設定されたので、それらを使用して承認ポリシーを設定し、そのポリシーを使用してルートへのアクセスを制御できます。

この例では、エンドポイント **mock:end** は、認証済みまたは認証可能で **ROLE_ADMIN** 権限を含む Spring Security **Authentication** オブジェクトが **管理者 SpringSecurityAuthorizationPolicy**によって位置付けられることができなければ、実行されません。

302.3. 認証

認可に使用されるセキュリティー資格証明を取得するプロセスは、このコンポーネントでは指定されていません。必要に応じて、交換から認証情報を取得する独自のプロセッサーまたはコンポーネントを作成できます。たとえば、Jetty コンポーネントで発生した HTTP 要求ヘッダーから認証情報を取得するプロセッサーを作成できます。認証情報がどのように収集されても、Camel Spring Security コンポーネントがアクセスできるように、In メッセージまたは **SecurityContextHolder** に配置する必要があります。

```
import javax.security.auth.Subject;
import org.apache.camel.*;
import org.apache.commons.codec.binary.Base64;
import org.springframework.security.authentication.*;


public class MyAuthService implements Processor {
   public void process(Exchange exchange) throws Exception {
      // get the username and password from the HTTP header
      // http://en.wikipedia.org/wiki/Basic_access_authentication
      String userpass = new
String(Base64.decodeBase64(exchange.getIn().getHeader("Authorization", String.class)));
      String[] tokens = userpass.split(":");

      // create an Authentication object
      UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(tokens[0], tokens[1]);
```

```
        // wrap it in a Subject
        Subject subject = new Subject();
        subject.getPrincipals().add(authToken);

        // place the Subject in the In message
        exchange.getIn().setHeader(Exchange.AUTHENTICATION, subject);

        // you could also do this if useThreadSecurityContext is set to true
        // SecurityContextHolder.getContext().setAuthentication(authToken);
    }
}
```

SpringSecurityAuthorizationPolicy は、必要に応じて **Authentication** オブジェクトを自動的に認証します。

Exchange.AUTHENTICATION ヘッダーの代わりに、または Exchange.AUTHENTICATION ヘッダーに加えて **SecurityContextHolder** を使用する場合、注意すべき 2 つの問題があります。まず、コンテキストホルダーはスレッドローカル変数を使用して **Authentication** オブジェクトを保持します。**seda** や **jms** など、スレッドの境界をまたぐルートは、**Authentication** オブジェクトを失います。次に、Spring Security システムは、コンテキスト内の **Authentication** オブジェクトがすでに認証されており、ロールを持っていることを期待しているように見えます (詳細については、技術概要 セクション 5.3.1 を参照してください)。

camel-spring-security のデフォルトの動作は、**Exchange.AUTHENTICATION** ヘッダーで **Subject** を探すことです。この **Subject** には、**org.springframework.security.core.Authentication** のサブクラスである必要があるプリンシパルが少なくとも 1 つ含まれている必要があります。**<authorizationPolicy>** Bean に **org.apache.camel.component.spring.security.AuthenticationAdapter** の実装を提供することで、**Subject** から **Authentication** オブジェクトへのマッピングをカスタマイズできます。これは、Spring Security を使用しないが **Subject** を提供するコンポーネントを操作している場合に役立ちます。現時点では、CXF コンポーネントのみが **Exchange.AUTHENTICATION** ヘッダーに入力されます。

302.4. 認証および認可エラーの処理

SpringSecurityAuthorizationPolicy で認証または認可が失敗した場合、**CamelAuthorizationException** が出力されます。これは、Exception Clause などの Camel の標準的な例外処理メソッドを使用して処理できます。**CamelAuthorizationException** には、例外を出力したポリシーの ID への参照が含まれているため、ポリシーと例外のタイプに基づいてエラーを処理できます。

```
<onException>
  <exception>org.springframework.security.authentication.AccessDeniedException</exception>
  <choice>
    <when>
      <simple>${exception.policyId} == 'user'</simple>
      <transform>
        <constant>You do not have ROLE_USER access!</constant>
      </transform>
    </when>
    <when>
      <simple>${exception.policyId} == 'admin'</simple>
      <transform>
        <constant>You do not have ROLE_ADMIN access!</constant>
      </transform>
```

```
    </when>
  </choice>
</onException>
```

302.5. 依存関係

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-security</artifactId>
  <version>2.4.0</version>
</dependency>
```

この依存関係は、**org.springframework.security:spring-security-core:3.0.3.RELEASE** および **org.springframework.security:spring-security-config:3.0.3.RELEASE** も取り込みます。

302.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- コンポーネント

第303章 SPRING WEBSERVICE コンポーネント

Camel バージョン 2.6 以降で利用可能

spring-ws: コンポーネントを使用すると、Spring Web Services と統合できます。Web サービスにアクセスするための **クライアント** 側のサポートと、独自のコントラクト優先の Web サービスを作成するための **サーバー** 側のサポートの両方を提供します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-ws</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

情報:*依存関係* Camel 2.8 の時点で、このコンポーネントには Spring-WS 2.0.x が同梱されており、(Camel の残りの部分と同様に) Spring 3.0.x が必要です。以前の Camel バージョンには、Spring 2.5.x および 3.0.x と互換性のある Spring-WS 1.5.9 が同梱されていました。Spring 2.5.x で以前のバージョンの **camel-spring-ws** を実行するには、Spring 2.5.x から **spring-webmvc** モジュールを追加する必要があります。Spring 3.0.x で Spring-WS 1.5.9 を実行するには、Spring 3.0.x から OXM モジュールを除外する必要があります。このモジュールは Spring-WS 1.5.9 にも含まれているためです (この投稿を 参照)。

303.1. URI 形式

このコンポーネントの URI スキームは次のとおりです。

```
spring-ws:[mapping-type:]address[?options]
```

Web サービスを公開するには、**マッピングタイプ** を次のいずれかに設定する必要があります。

マッピングタイプ	説明
rootqname	メッセージに含まれるルート要素の修飾名に基づいて Web サービスリクエストをマップするオプションを提供します。
soapaction	メッセージのヘッダーで指定された SOAP アクションに基づいて Web サービスリクエストをマップするために使用されます。
uri	特定の URI を対象とする Web サービスリクエストをマップする場合。
xpathresult	入力メッセージに対する XPath **式** の評価に基づいて、Web サービスリクエストをマップするために使用されます。評価の結果は、エンドポイント URI で指定された XPath の結果と一致する必要があります。

マッピングタイプ	説明
beanname	**PayloadRootQNameEndpointMapping**、**SoapActionEndpointMapping** などの既存の (レガシー) エンドポイントマッピング と統合するために、**org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher** オブジェクトを参照できます。

コンシューマーとして、**アドレス** には指定されたマッピングタイプ (SOAP アクション、XPath 式など) に関連する値が含まれている必要があります。プロデューサとして、アドレスは呼び出し先の Web サービスの URI に設定する必要があります。

URI には、**?option=value&option=value&...** の形式でクエリー options を追加できます。

303.2. オプション

Spring WebService コンポーネントは、以下にリストされている 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spring WebService エンドポイントは、URI 構文を使用して設定されます。

```
spring-ws:type:lookupKey:webServiceEndpointUri
```

パスおよびクエリーパラメーターを使用します。

303.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
type	エンドポイントマッピングが使用されている場合のエンドポイントマッピングタイプ。rootqname - メッセージに含まれるルート要素の修飾名に基づいて Web サービスリクエストをマップするオプションを提供します。soapaction - メッセージのヘッダーで指定された SOAP アクションに基づいて Web サービスリクエストをマップするために使用されます。uri - 特定の URI を対象とする Web サービスリクエストをマップするため。xpathresult - 入力メッセージに対する XPath 式の評価に基づいて Web サービス要求をマップするために使用されます。評価の結果は、エンドポイント URI で指定された XPath の結果と一致する必要があります。beanname - PayloadRootQNameEndpointMapping、SoapActionEndpointMapping などの既存の (レガシー) エンドポイントマッピングと統合するために、org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher オブジェクトを参照できるようにします。		EndpointMappingType
lookupKey	エンドポイントマッピングが使用されている場合のエンドポイントマッピングキー		String
webServiceEndpointUri	プロデューサーに使用するデフォルトの Web サービスエンドポイント uri。		String

303.2.2. クエリーパラメーター(22 個のパラメーター):

名前	説明	デフォルト	タイプ
messageFilter (common)	カスタム MessageFilter を提供するオプション。たとえば、ヘッダーや添付を自分で処理したい場合などです。		MessageFilter
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
endpointDispatcher (consumer)	Spring org.springframework.ws.server.endpoint.MessageEndpoint は、Spring -WS によって受信されたメッセージを Camel エンドポイントにディスパッチし、PayloadRootQNameEndpointMapping、SoapActionEndpointMapping などの既存の (レガシー) エンドポイントマッピングと統合します。		CamelEndpointDispatcher
endpointMapping (consumer)	Registry/ApplicationContext 内の org.apache.camel.component.spring.ws.bean.CamelEndpointMapping のインスタンスへの参照。すべての Camel/Spring-WS エンドポイントを提供するために、レジストリーで必要な Bean は 1 つだけです。この Bean は MessageDispatcher によって自動検出され、エンドポイントで指定された特性 (ルート QName、SOAP アクションなど) に基づいて要求を Camel エンドポイントにマップするために使用されます。		CamelSpringWSEndpoint マッピング
expression (consumer)	オプション type=xpathresult の場合に使用する XPath 式。次に、このオプションを設定する必要があります。		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
allowResponseAttachment Override (producer)	実際のサービス層からの添付とのイン/アウトエクスチェンジで SOAP レスポンスの添付をオーバーライドするオプション。呼び出されたサービスが SOAP 添付を追加または書き換える場合、このオプションを true に設定すると、変更された SOAP 添付をイン/アウトメッセージ添付で上書きできます	false	boolean
allowResponseHeaderOverride (producer)	実際のサービス層からのヘッダー情報とのイン/アウトエクスチェンジで SOAP レスポンスヘッダーをオーバーライドするオプション。呼び出されたサービスが SOAP ヘッダーを追加または書き換える場合、このオプションを true に設定すると、変更された SOAP ヘッダーをイン/アウトメッセージヘッダーで上書きできます。	false	boolean

名前	説明	デフォルト	タイプ
faultAction (producer)	メソッドによって提供される faultAction レスポンス WS-Addressing Fault Action ヘッダーの値を示します。		URI
faultTo (producer)	メソッドによって提供される faultAction レスポンス WS-Addressing FaultTo ヘッダーの値を示します。		URI
messageFactory (producer)	カスタム WebServiceMessageFactory を提供するオプション。たとえば、SAAJ の代わりに Apache Axiom で Web サービスメッセージを処理する場合などです。		WebServiceMessage ファクトリー
messageIdStrategy (producer)	一意のメッセージ ID の生成を制御するカスタム MessageIdStrategy を提供するオプション。		MessageIdStrategy
messageSender (producer)	カスタム WebServiceMessageSender を提供するオプション。たとえば、認証を実行したり、代替トランスポートを使用したりします		WebServiceMessage 送信元
outputAction (producer)	メソッドによって提供されるレスポンス WS-Addressing Action ヘッダーの値を示します。		URI
replyTo (producer)	メソッドによって提供される、replyTo レスポンス WS-Addressing ReplyTo ヘッダーの値を示します。		URI
soapAction (producer)	リモート Web サービスにアクセスするときに SOAP 要求内に含める SOAP アクション		String

名前	説明	デフォルト	タイプ
timeout (producer)	プロデューサーを使用して Web サービスを呼び出す際のソケット読み取りタイムアウト (ミリ秒単位) を設定します。URLConnection.setReadTimeout() および CommonsHttpMessageSender.setReadTimeout() を参照してください。このオプションは、ビルトインメッセージ送信者の実装である CommonsHttpMessageSender および HttpUrlConnectionMessageSender を使用する場合に機能します。コンポーネントに提供される Spring WS 設定オプションをカスタマイズしない限り、これらの実装の 1 つがデフォルトで HTTP ベースのサービスに使用されます。非標準の送信者を使用している場合は、独自のタイムアウト設定を処理することが想定されています。ビルトインメッセージ送信者 HttpComponentsMessageSender は、非推奨になった CommonsHttpMessageSender の代わりに考慮されます。HttpComponentsMessageSender.setReadTimeout() を参照してください。		int
webServiceTemplate (producer)	カスタム WebServiceTemplate を提供するオプション。これにより、クライアント側の Web サービス処理を完全に制御できます。カスタムインターセプターの追加や、障害リゾルバー、メッセージ送信者、またはメッセージファクトリーの指定など。		WebServiceTemplate
wsAddressingAction (producer)	Web サービスにアクセスするときに含める WS-Addressing 1.0 アクションヘッダー。To ヘッダーは、エンドポイント URI で指定された Web サービスのアドレスに設定されます (デフォルトの Spring-WS 動作)。		URI
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

303.2.3. メッセージヘッダー

名前	タイプ	説明
CamelSpringWebserviceEndpointUri	String	クライアントとしてアクセスする Web サービスの URI は、エンドポイント URI の **アドレス** 部分をオーバーライドします。
CamelSpringWebserviceSoapAction	String	メッセージの SOAP アクションを指定するヘッダー。存在する場合は **soapAction** オプションをオーバーライドします。
CamelSpringWebserviceSoapHeader	Source	**Camel 2.11.1:** このヘッダーを使用して、メッセージの SOAP ヘッダーを指定/アクセスします。
CamelSpringWebserviceAddressingAction	URI	このヘッダーを使用してメッセージの WS-Addressing アクションを指定し、存在する場合は **wsAddressingAction** オプションをオーバーライドします
CamelSpringWebserviceAddressingFaultTo	URI	このヘッダーを使用して WS-Addressing FaultTo を指定し、存在する場合は faultTo オプションをオーバーライドします。
CamelSpringWebserviceAddressingReplyTo	URI	このヘッダーを使用して WS-Addressing ReplyTo を指定し、存在する場合は replyTo オプションをオーバーライドします。

名前	タイプ	説明
CamelSpringWebserviceAddressingOutputAction	URI	このヘッダーを使用して WS-Addressing Action を指定し、存在する場合は outputAction オプションをオーバーライドします。
CamelSpringWebserviceAddressingFaultAction	URI	このヘッダーを使用して WS-Addressing Fault Action を指定します。存在する場合は faultAction オプションをオーバーライドします。

303.3. WEB サービスへのアクセス

http://foo.com/bar で Web サービスを呼び出すには、単純にルートを定義します。

```
from("direct:example").to("spring-ws:http://foo.com/bar")
```

そしてメッセージを送信しました:

```
template.requestBody("direct:example", "<foobar xmlns=\"http://foo.com\"><msg>test
message</msg></foobar>");
```

呼び出しているのが SOAP サービスである場合は、SOAP タグを含める必要がないことに注意してください。Spring-WS は、XML から SOAP へのマーシャリングを実行します。

303.4. SOAP および WS-ADDRESSING アクションヘッダーの送信

リモート Web サービスで SOAP アクションまたは WS-Addressing 標準の使用が必要な場合は、ルートを次のように定義します。

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?soapAction=http://foo.com&wsAddressingAction=http://bar.com")
```

必要に応じて、エンドポイントオプションをヘッダー値でオーバーライドできます。

```
template.requestBodyAndHeader("direct:example",
"<foobar xmlns=\"http://foo.com\"><msg>test message</msg></foobar>",
SpringWebserviceConstants.SPRING_WS_SOAP_ACTION, "http://baz.com");
```

303.5. SOAP ヘッダーの使用

Camel 2.11.1 以降で利用可能

メッセージを spring-ws エンドポイントに送信するときに、SOAP ヘッダーを Camel メッセージヘッダーとして提供できます。

```
String body = ...
String soapHeader = "<h:Header xmlns:h=\"http://www.webserviceX.NET/\">
<h:MessageID>1234567890</h:MessageID><h:Nested><h:NestedID>1111</h:NestedID>
</h:Nested></h:Header>";
```

次のように、Camel メッセージのボディーとヘッダーを設定できます。

```
exchange.getIn().setBody(body);
exchange.getIn().setHeader(SpringWebserviceConstants.SPRING_WS_SOAP_HEADER,
soapHeader);
```

次に、Exchange を **spring-ws** エンドポイントに送信して、Web サービスを呼び出します。

同様に、spring-ws コンシューマーも SOAP ヘッダーを使用して Camel メッセージを強化します。

例については、この 単体テスト を参照してください。

303.6. ヘッダーと添付の伝播

Spring WS Camel は、バージョン 2.10.3 以降、ヘッダーと添付の Spring-WS WebServiceMessage レスポンスへの伝播をサポートしています。エンドポイントは、MessageFilter のいわゆるフックを使用して (デフォルトの実装は BasicMessageFilter によって提供されます)、エクスチェンジヘッダーと添付を WebServiceMessage レスポンスに伝播します。これで使用できます。

```
exchange.getOut().getHeaders().put("myCustom","myHeaderValue")
exchange.getIn().addAttachment("myAttachment", new DataHandler(...))
```

注記: パイプラインのエクスチェンジヘッダーにテキストが含まれている場合、SOAP ヘッダーに Qname (key)=value 属性が生成されます。QName クラスを直接作成し、任意のキーをヘッダーに入れることをお勧めします。

303.7. スタイルシートを使用して SOAP ヘッダーを変換する方法

ヘッダー変換フィルター (HeaderTransformationMessageFilter.java) を使用して、SOAP リクエストの SOAP ヘッダーを変換できます。ヘッダー変換フィルターを使用する場合は、次の例を参照してください。

```
<bean id="headerTransformationFilter"
class="org.apache.camel.component.spring.ws.filter.impl.HeaderTransformationMessageFilter">
    <constructor-arg index="0" value="org/apache/camel/component/spring/ws/soap-header-
transform.xslt"/>
</bean
```

上記で定義した bean を camel エンドポイントで使用する

```
<route>
    <from uri="direct:stockQuoteWebserviceHeaderTransformation"/>
    <to uri="spring-ws:http://localhost?
```

```
webServiceTemplate=#webServiceTemplate&amp;soapAction=http://www.stockquotes.edu/GetQuote&
amp;messageFilter=#headerTransformationFilter"/>
</route>
```

303.8. MTOM アタッチメントの使用方法

BasicMessageFilter は、MTOM メッセージを生成するために Apache Axiom に必要なすべての情報を提供します。Apache Axiom 内で Apache Spring Spring WS を使用する場合の例を次に示します。

```
<bean id="axiomMessageFactory"
class="org.springframework.ws.soap.axiom.AxiomSoapMessageFactory">
<property name="payloadCaching" value="false" />
<property name="attachmentCaching" value="true" />
<property name="attachmentCacheThreshold" value="1024" />
</bean>
```

- 次の依存関係を pom.xml に追加します。

```
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-api</artifactId>
<version>1.2.13</version>
</dependency>
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-impl</artifactId>
<version>1.2.13</version>
<scope>runtime</scope>
</dependency>
```

- たとえば、Processor 実装を使用して、添付をパイプラインに追加します。

```
private class Attachement implements Processor {
public void process(Exchange exchange) throws Exception
{ exchange.getOut().copyFrom(exchange.getIn()); File file = new File("testAttachment.txt");
exchange.getOut().addAttachment("test", new DataHandler(new FileDataSource(file))); }
}
```

- エンドポイント (プロデューサー) を通常どおりに定義します。たとえば、次のようになります。

```
from("direct:send")
.process(new Attachement())
.to("spring-ws:http://localhost:8089/mySoapService?
soapAction=mySoap&messageFactory=axiomMessageFactory");
```

- これで、プロデューサーは最適化された添付を含む MTOM メッセージを生成します。

303.9. カスタムヘッダーと添付のフィルタリング

ヘッダーまたは添付のカスタム処理を提供する必要がある場合は、既存の BasicMessageFilter を拡張して適切なメソッドをオーバーライドするか、MessageFilter インターフェイスのまったく新しい実装を記述します。

カスタムフィルターを使用するには、これを spring コンテキストに追加します。

次のように、グローバルまたはローカルのメッセージフィルターを指定できます。a) すべての Spring-WS エンドポイントのグローバル設定を提供するグローバルカスタムフィルター

```
<bean id="messageFilter" class="your.domain.myMessageFiler" scope="singleton" />
```

または b) 次のように、エンドポイントで直接ローカルの messageFilter を使用します。

```
to("spring-ws:http://yourdomain.com?messageFilter=#myEndpointSpecificMessageFilter");
```

詳細は、CAMEL-5724 を参照してください。

独自の MessageFilter を作成する場合は、クラス BasicMessageFilter の MessageFilter のデフォルト実装で次のメソッドをオーバーライドすることを検討してください。

```
protected void doProcessSoapHeader(Message inOrOut, SoapMessage soapMessage)
{your code /*no need to call super*/ }

protected void doProcessSoapAttachements(Message inOrOut, SoapMessage response)
{ your code /*no need to call super*/ }
```

303.10. カスタム MESSAGESENDER と MESSAGEFACTORY の使用

レジストリー内のカスタムメッセージセンダーまたはファクトリーは、次のように参照できます。

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?
messageFactory=#messageFactory&messageSender=#messageSender")
```

Spring の設定

```
<!-- authenticate using HTTP Basic Authentication -->
<bean id="messageSender"
class="org.springframework.ws.transport.http.HttpComponentsMessageSender">
    <property name="credentials">
        <bean class="org.apache.commons.httpclient.UsernamePasswordCredentials">
            <constructor-arg index="0" value="admin"/>
            <constructor-arg index="1" value="secret"/>
        </bean>
    </property>
</bean>

<!-- force use of Sun SAAJ implementation, http://static.springsource.org/spring-
ws/sites/1.5/faq.html#saaj-jboss -->
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
    <property name="messageFactory">
        <bean
class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl"></bean>
    </property>
</bean>
```

303.11. WEB サービスの公開

このコンポーネントを使用して Web サービスを公開するには、まず MessageDispatcher をセットアップして、Spring XML ファイルでエンドポイントマッピングを探す必要があります。サーブレットコンテナー内で実行する予定がある場合は、**web.xml** で設定された **MessageDispatcherServlet** を使用することをお勧めします。

デフォルトでは、**MessageDispatcherServlet** は **/WEB-INF/spring-ws-servlet.xml** という名前の Spring XML を探します。Spring-WS で Camel を使用するには、その XML ファイルで必須の Bean は **CamelEndpointMapping** のみです。この Bean により、**MessageDispatcher** は Web サービスリクエストをルートにディスパッチできます。

web.xml

```
<web-app>
   <servlet>
      <servlet-name>spring-ws</servlet-name>
      <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>spring-ws</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

spring-ws-servlet.xml

```
<bean id="endpointMapping"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointMapping" />

<bean id="wsdl" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">
   <property name="schema">
      <bean class="org.springframework.xml.xsd.SimpleXsdSchema">
         <property name="xsd" value="/WEB-INF/foobar.xsd"/>
      </bean>
   </property>
   <property name="portTypeName" value="FooBar"/>
   <property name="locationUri" value="/"/>
   <property name="targetNamespace" value="http://example.com/"/>
</bean>
```

Spring-WS のセットアップの詳細については、Writing Contract-First Web Services を参照してください。基本的に、3.6 項エンドポイントの実装はこのコンポーネントによって処理されます (具体的には、3.6.2 項エンドポイントへのメッセージのルーティングは **CamelEndpointMapping** の出番です)。また、Camel ディストリビューションに含まれている Spring Web Services Example も忘れずにチェックしてください。

303.12. ルートのエンドポイントマッピング

XML 設定が整ったので、Camel の DSL を使用して、エンドポイントで処理される Web サービスリクエストを定義できます。

次のルートは、**http://example.com/** 名前空間内に GetFoo という名前のルート要素を持つすべての Web サービスリクエストを受け取ります。

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

次のルートは、**http://example.com/GetFoo** SOAP アクションを含む Web サービスリクエストを受け取ります。

```
from("spring-ws:soapaction:http://example.com/GetFoo?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

次のルートは、**http://example.com/foobar** に送信されたすべてのリクエストを受け取ります。

```
from("spring-ws:uri:http://example.com/foobar?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

以下のルートは、要素 **<foobar>abc</foobar>** をメッセージ内の任意の場所 (およびデフォルトの名前空間) に含むリクエストを受け取ります。

```
from("spring-ws:xpathresult:abc?expression=//foobar&endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

303.13. 既存のエンドポイントマッピングを使用した代替設定

mapping-type **beanname** を持つすべてのエンドポイントに対して、対応する名前を持つタイプ **CamelEndpointDispatcher** の 1 つの Bean が Registry/ApplicationContext に必要です。この Bean は、Camel エンドポイントと **PayloadRootQNameEndpointMapping** などの既存の エンドポイントマッピング の間のブリッジとして機能します。

注記: **beanname** mapping-type の使用は、主に、すでに Spring-WS を使用しており、Spring XML ファイルでエンドポイントマッピングが定義されている (レガシー) 状況を対象としています。**beanname** マッピングタイプを使用すると、Camel ルートを既存のエンドポイントマッピングに接続できます。ゼロから始める場合は、設定が少なくて済み、表現力が高いため、(上の **endpointMapping** で示したように) エンドポイントマッピングを Camel URI として定義することをお勧めします。または、アノテーションを使用してバニラ Spring-WS を使用することもできます。

beanname を使用したルートの例:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="spring-ws:beanname:QuoteEndpointDispatcher" />
        <to uri="mock:example" />
    </route>
</camelContext>

<bean id="legacyEndpointMapping"
class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
    <property name="mappings">
        <props>
            <prop key="{http://example.com/}GetFuture">FutureEndpointDispatcher</prop>
            <prop key="{http://example.com/}GetQuote">QuoteEndpointDispatcher</prop>
        </props>
    </property>
</bean>
```

```
<bean id="QuoteEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
<bean id="FutureEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
```

303.14. POJO (非) マーシャリング

Camel のプラグ可能なデータ形式は、JAXB、XStream、JibX、Castor、XMLBeans などのライブラリーを使用した pojo/xml マーシャリングのサポートを提供します。ルートでこれらのデータ形式を使用して、Web サービスとの間で pojo を送受信できます。

Web サービス **にアクセスする** ときは、リクエストをマーシャリングし、レスポンスメッセージをアンマーシャリングできます。

```
JaxbDataFormat jaxb = new JaxbDataFormat(false);
jaxb.setContextPath("com.example.model");

from("direct:example").marshal(jaxb).to("spring-ws:http://foo.com/bar").unmarshal(jaxb);
```

同様に、Web サービス **を提供する** 場合、POJO への XML リクエストをアンマーシャリングし、レスポンスメッセージを XML にマーシャリングすることができます。

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?
endpointMapping=#endpointMapping").unmarshal(jaxb)
.to("mock:example").marshal(jaxb);
```

303.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第304章 SQL コンポーネント

Camel バージョン 1.4 以降で利用可能

sql: コンポーネントでは、JDBC クエリーを使用してデータベースを操作することができます。このコンポーネントと JDBC コンポーネントの相違点は、SQL の場合、クエリーがエンドポイントのプロパティーであり、クエリーに渡されるパラメーターとしてメッセージペイロードを使用ことです。

このコンポーネントは、実際の SQL 処理のために舞台裏で **spring-jdbc** を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

SQL コンポーネントは以下もサポートしています。

- Idempotent Consumer EIP パターン用の JDBC ベースのリポジトリー。以下を参照してください。
- Aggregator EIP パターンの JDBC ベースのリポジトリー。以下を参照してください。

304.1. URI 形式

警告: Camel 2.11 以降、このコンポーネントはコンシューマー (例: **from()**) とプロデューサーエンドポイント (例 **: to()**) の両方を作成できます。以前のバージョンでは、プロデューサーとしてしか機能できませんでした。

情報: このコンポーネントは、Transactional Client として使用できます。

SQL コンポーネントは、次のエンドポイント URI 表記を使用します。

```
sql:select * from table where id=# order by name[?options]
```

Camel 2.11 以降では、次のように :`#name_of_the_parameter` スタイルを使用して名前付きパラメーターを使用できます。

```
sql:select * from table where id=:#myId order by name[?options]
```

名前付きパラメーターを使用する場合、Camel は指定された優先順位で名前を検索します。
1. **java.util.Map** の場合はメッセージ本文から
2. メッセージヘッダーから

名前付きパラメーターを解決できない場合は、例外が出力されます。

Camel 2.14 以降では、次のように Simple 式をパラメーターとして使用できます。

```
sql:select * from table where id=:#${property.myId} order by name[?options]
```

標準 **?** SQL クエリーのパラメーターを示す記号は **#** 記号に置き換えられます。**?** 記号は、エンドポイントのオプションを指定するために使用されます。**?** シンボルの置換は、エンドポイントごとに設定できます。

Camel 2.17 以降では、次に示すように、SQL クエリーをクラスパスまたはファイルシステム内のファイルに外部化できます。

```
sql:classpath:sql/myquery.sql[?options]
```

そして、myquery.sql ファイルはクラスパスにあり、単なるプレーンテキストです。

```
-- this is a comment
select *
from table
where
  id = :#${property.myId}
order by
  name
```

ファイルでは、複数行を使用して、必要に応じて SQL をフォーマットできます。– 破線などのコメントも使用します。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

304.2. オプション

SQL コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
dataSource (Common)	データベースとの通信に使用する DataSource を設定します。		DataSource
usePlaceholder (advanced)	SQL クエリーでプレースホルダーを使用し、すべてのプレースホルダー文字を符号に置き換えるかどうかを設定します。このオプションのデフォルトは true です	true	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SQL エンドポイントは、URI 構文を使用して設定されます。

```
sql:query
```

パスおよびクエリーパラメーターを使用します。

304.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
query	**必須** 実行する SQL クエリーを設定します。file: または classpath: を接頭辞として使用してクエリーを外部化し、ファイルの場所を指定できます。		String

304.2.2. クエリーパラメーター(45 個のパラメーター):

名前	説明	デフォルト	タイプ
allowNamedParameters (common)	クエリーで名前付きパラメーターの使用を許可するかどうか。	true	boolean
dataSource (Common)	データベースとの通信に使用する DataSource を設定します。		DataSource
dataSourceRef (common)	**非推奨** データベースとの通信に使用するために、レジストリーから参照する DataSource への参照を設定します。		String
outputClass (common)	outputType=SelectOne の場合、変換として使用する完全なパッケージとクラス名を指定します。		String
outputHeader (common)	メッセージ本文ではなく、ヘッダーにクエリー結果を格納します。デフォルトでは、outputHeader == null で、クエリー結果はメッセージ本文に格納され、メッセージ本文の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を格納するヘッダーの名前として使用され、元のメッセージ本文は保持されます。		String
outputType (common)	以下の方法で、コンシューマーまたはプロデューサの出力を Map のリストとして SelectList、または単一の Java オブジェクトとして SelectOne にします。a) クエリーが単一の列を持っている場合、その JDBC 列オブジェクトが返されます。(SELECT COUNT( ) FROM PROJECT のような場合、Long オブジェクトを返します。) b) クエリーが複数の列を持つ場合、その結果の Map を返します。c) 出力クラスが設定されている場合、列名に一致するすべてのセッターを呼び出して、クエリーの結果を Java Bean オブジェクトに変換します。クラスには、インスタンスを作成するためのデフォルトのコンストラクターがあると想定されます。d) クエリーの結果が複数の行になった場合、一意でない結果の例外が出力されます。	SelectList	SqlOutputType

名前	説明	デフォルト	タイプ
separator (common)	パラメーター値がメッセージ本文 (本文が文字列型の場合) から取得されるときに使用するセパレーターで、プレースホルダーに挿入されます。名前付きパラメーターを使用する場合は、代わりに Map タイプが使用されることに注意してください。デフォルト値は、コンマです。	,	char
breakBatchOnConsumeFail (consumer)	onConsume が失敗した場合にバッチを中断するかどうかを設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
expectedUpdateCount (consumer)	onConsume を使用するときに検証する予想更新カウントを設定します。	-1	int
maxMessagesPerPoll (consumer)	ポーリングするメッセージの最大数を設定します		int
onConsume (consumer)	各行を処理した後、エクスチェンジが正常に処理された場合、たとえば行を処理済みとしてマークするために、このクエリーを実行できます。クエリーにはパラメーターを含めることができます。		String
onConsumeBatchComplete (consumer)	バッチ全体を処理した後、このクエリーを実行して行などを一括更新できます。クエリーにパラメーターを含めることはできません。		String
onConsumeFailed (consumer)	各行を処理した後、エクスチェンジが失敗した場合、たとえば、行を失敗としてマークするために、このクエリーを実行できます。クエリーにはパラメーターを含めることができます。		String
routeEmptyResultSet (consumer)	空の結果セットを次のホップに送信できるようにするかどうかを設定します。デフォルトは false です。したがって、空の結果セットは除外されます。	false	boolean

名前	説明	デフォルト	タイプ
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
transacted (consumer)	トランザクションを有効または無効にします。有効にすると、交換の処理が失敗した場合に、コンシューマーはそれ以上の交換の処理を中断して、先行してロールバックを実行させます。	false	boolean
useIterator (consumer)	結果セットをルートに配信する方法を設定します。リストまたは個々のオブジェクトとして配信を示します。デフォルトは真です。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processingStrategy (consumer)	コンシューマーが行/バッチを処理したときに、カスタム org.apache.camel.component.sql.SqlProcessingStrategy を使用してクエリーを実行するプラグインを許可します。		SqlProcessingStrategy
batch (producer)	バッチモードを有効または無効にします	false	boolean
noop (producer)	設定されている場合、SQL クエリーの結果を無視し、既存の IN メッセージを OUT メッセージとして使用して処理を続行します。	false	boolean
useMessageBodyForSql (producer)	メッセージ本文を SQL として使用し、次にパラメーターのヘッダーを使用するかどうか。このオプションを有効にすると、URI の SQL は使用されません。	false	boolean

名前	説明	デフォルト	タイプ
alwaysPopulateStatement (producer)	有効にすると、org.apache.camel.component.sql.SqlPrepareStatementStrategy の populateStatement メソッドが常に呼び出されます。また、準備する必要のあるパラメーターがない場合も同様です。これが false の場合、populateStatement は、1 つ以上の予期されるパラメーターが設定される場合にのみ呼び出されます。たとえば、これにより、パラメーターのない SQL クエリーのメッセージ本文/ヘッダーの読み取りが回避されます。	false	boolean
parametersCount (プロデューサー)	0 より大きい値を設定すると、Camel は JDBC メタデータ API を介してクエリーを実行する代わりに、このパラメーターのカウント値を使用して置き換えます。これは、JDBC ベンダーが正しいパラメーター数を返すことができず、ユーザーが代わりにオーバーライドできる場合に役立ちます。		int
placeholder (advanced)	SQL クエリーで置換される文字を指定します。これは単純な String.replaceAll() 操作であり、SQL 解析が含まれていないことに注意してください (引用符で囲まれた文字列も変更されます)。	#	String
prepareStatementStrategy (advanced)	プラグインでカスタム org.apache.camel.component.sql.SqlPrepareStatementStrategy を使用して、クエリーと準備済みステートメントの準備を制御できるようにします。		SqlPrepareStatement Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
templateOptions (advanced)	マップからのキー/値を使用して Spring JdbcTemplate を設定します		Map
usePlaceholder (advanced)	SQL クエリーでプレースホルダーを使用し、すべてのプレースホルダー文字を符号に置き換えるかどうかを設定します。このオプションのデフォルトは true です	true	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

名前	説明	デフォルト	タイプ
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

304.3. メッセージボディーの扱い

SQL コンポーネントは、メッセージボディーを **java.util.Iterator** 型のオブジェクトに変換しようとし、この反復子を使用してクエリーパラメーターを入力します (各クエリーパラメーターは、エンドポイント URI の **#** 記号 (または設定されたプレースホルダー) で表されます)。.メッセージ本文が配列でもコレクションでもない場合、変換の結果は、本文自体である 1 つのオブジェクトのみを反復処理する反復子になります。

たとえば、メッセージ本文が **java.util.List** のインスタンスである場合、リストの最初の項目は SQL クエリーで最初に出現する **#** に置換され、リストの 2 番目の項目は 2 番目に出現する **#** などに置換されます

batch が **true** に設定されている場合、受信メッセージ本文の解釈がわずかに変更されます。コンポーネントは、パラメーターのイテレーターではなく、パラメーターの反復子を含む反復子を期待します。外側の反復子のサイズによってバッチサイズが決まります。

Camel 2.16 以降では、メッセージ本文を SQL ステートメントとして使用できるようにするオプション useMessageBodyForSql を使用できます。次に、SQL パラメーターをキー SqlConstants.SQL_PARAMETERS を使用してヘッダーに指定する必要があります。これにより、SQL クエリーがメッセージ本文からのものであるため、SQL コンポーネントがより動的に動作できるようになります。

304.4. クエリーの結果

select 操作の場合、結果は JdbcTemplate.queryForList () メソッドによって返される **List<Map<String, Object>>** タイプのインスタンスです。**更新** 操作の場合、結果は更新された行の数であり、**Integer** として返されます。

デフォルトでは、結果はメッセージボディーに配置されます。 outputHeader パラメーターが設定されている場合、結果はヘッダーに配置されます。 これは、完全なメッセージエンリッチメントパターンを使用してヘッダーを追加する代わりの方法であり、シーケンスまたはその他の小さな値をヘッダーにクエリーするための簡潔な構文を提供します。 outputHeader と outputType を一緒に使用すると便利です。

```
from("jms:order.inbox")
    .to("sql:select order_seq.nextval from dual?outputHeader=OrderId&outputType=SelectOne")
    .to("jms:order.booking");
```

304.5. STREAMLIST の使用

Camel 2.18 以降、プロデューサーは、反復子を使用してクエリーの出力をストリーミングする outputType=StreamList をサポートします。これにより、ストリーミング方式でデータを処理できます。たとえば、スプリッター EIP で各行を 1 つずつ処理し、必要に応じてデータベースからデータをロードするために使用できます。

```
from("direct:withSplitModel")
        .to("sql:select * from projects order by id?
outputType=StreamList&outputClass=org.apache.camel.component.sql.ProjectModel")
        .to("log:stream")
        .split(body()).streaming()
            .to("log:row")
            .to("mock:result")
        .end();
```

304.6. ヘッダーの値

update 操作を実行すると、SQL コンポーネントは更新カウントを次のメッセージヘッダーに格納します。

ヘッダー	説明
CamelSqlUpdateCount	**update** 操作によって更新された行の数を **Integer** オブジェクトとして返します。outputType=StreamList を使用する場合、このヘッダーは提供されません。
CamelSqlRowCount	**select** 操作によって返される行の数を **Integer** オブジェクトで返します。outputType=StreamList を使用する場合、このヘッダーは提供されません。
CamelSqlQuery	Camel 2.8: 実行するクエリー。このクエリーは、エンドポイント URI で指定されたクエリーよりも優先されます。ヘッダーのクエリーパラメータは **#** 記号の代わりに **?** 記号で表 **される** ことに注意してください

insert 操作を実行すると、SQL コンポーネントは、生成されたキーとこれらの行の数を含む行を次のメッセージヘッダーに格納します (**Camel 2.12.4、2.13.1 以降で利用可能**)。

ヘッダー	説明
CamelSqlGeneratedKeysRowCount	生成されたキーを含むヘッダー内の行数。
CamelSqlGeneratedKeyRows	生成されたキー (キーのマップのリスト) を含む行。

304.7. 生成されたキー

Camel 2.12.4、2.13.1、および 2.14 で利用可能

SQL INSERT を使用してデータを挿入する場合、RDBMS は自動生成されたキーをサポートしている可能性があります。生成されたキーをヘッダーで返すように SQL プロデューサーに指示できます。
これを行うには、ヘッダー **CamelSqlRetrieveGeneratedKeys=true** を設定します。次に、生成されたキーは、上記の表にリストされているキーを含むヘッダーとして提供されます。

この 単体テスト で詳細を確認できます。

304.8. 設定

URI で直接 **DataSource** への参照を設定できるようになりました。

```
sql:select * from table where id=# order by name?dataSource=myDS
```

304.9. 例

以下のサンプルでは、クエリーを実行し、結果を行の **List** として取得します。ここで、各行は **Map<String, Object>** であり、キーは列名です。

まず、サンプルに使用するテーブルを設定します。これは単体テストに基づいているため、Java で行います。

実行する SQL スクリプト **createAndPopulateDatabase.sql** は、次のようになります。

次に、ルートと **SQL** コンポーネントを設定します。**SQL** エンドポイントの前に **direct** エンドポイントを使用していることに注意してください。これにより、クライアントが長い **sql:** URI よりもはるかに使いやすい URI **direct:simple** を使用して、**direct** エンドポイントにエクスチェンジを送信できます。**DataSource** はレジストリーで検索されるため、標準の Spring XML を使用して **DataSource** を設定できることに注意してください。

次に、メッセージを **direct** エンドポイントに送信します。このエンドポイントは、データベースにクエリーを実行する **SQL** コンポーネントにメッセージをルーティングします。

次のように、Spring XML で **DataSource** を設定できます。

```
<jee:jndi-lookup id="myDS" jndi-name="jdbc/myDataSource"/>
```

304.9.1. 名前付きパラメーターの使用

Camel 2.11 から利用可能

以下の特定のルートでは、projects テーブルからすべてのプロジェクトを取得します。SQL クエリーには、:#lic と :#min という 2 つの名前付きパラメーターがあることに注意してください。
Camel は、メッセージ本文またはメッセージヘッダーからこれらのパラメーターを検索します。上記の例では、2 つのヘッダーに定数値を設定していることに注意してください。
名前付きパラメーターの場合:

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
  .setHeader("min", constant(123))
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

ただし、メッセージボディーが **java.util.Map** の場合、名前付きパラメーターは本文から取得されます。

```
from("direct:projects")
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

304.9.2. 式パラメーターの使用

Camel 2.14 から利用可能

以下の特定のルートでは、データベースからすべてのプロジェクトを取得します。ライセンスを定義するためにエクスチェンジの本文を使用し、プロパティーの値を 2 番目のパラメーターとして使用します。

```
from("direct:projects")
  .setBody(constant("ASF"))
  .setProperty("min", constant(123))
  .to("sql:select * from projects where license = :#${body} and id > :#${property.min} order by id")
```

304.9.3. 動的な値での IN クエリーの使用

Camel 2.17 以降で利用可能

Camel 2.17 以降、SQL プロデューサーは、IN 値が動的に計算される IN ステートメントで SQL クエリーを使用できるようにします。たとえば、メッセージ本文やヘッダーなどから。

IN を使用するには、次のことが必要です。

- パラメーター名の前に **in:**
- パラメーターの前後に **( )** を追加します

例はこれをよりよく説明しています。次のクエリーが使用されます。

```
-- this is a comment
select *
from projects
where project in (:#in:names)
order by id
```

次の経路で:

```
from("direct:query")
    .to("sql:classpath:sql/selectProjectsIn.sql")
    .to("log:query")
    .to("mock:query");
```

次に、IN クエリーは、次のような動的な値を持つキー名を持つヘッダーを使用できます。

```
// use an array
template.requestBodyAndHeader("direct:query", "Hi there!", "names", new String[]{"Camel", "AMQ"});


// use a list
List<String> names = new ArrayList<String>();
names.add("Camel");
names.add("AMQ");

template.requestBodyAndHeader("direct:query", "Hi there!", "names", names);
```

```
// use a string separated values with comma
template.requestBodyAndHeader("direct:query", "Hi there!", "names", "Camel,AMQ");
```

クエリーは、外部化する代わりにエンドポイントで指定することもできます (外部化すると SQL クエリーの維持が容易になることに注意してください)。

```
from("direct:query")
    .to("sql:select * from projects where project in (:#in:names) order by id")
    .to("log:query")
    .to("mock:query");
```

304.10. JDBC ベースのべき等リポジトリーの使用

Camel 2.7 で利用可能: このセクションでは、JDBC ベースのべき等リポジトリーを使用します。

ヒント:*抽象クラス* Camel 2.9 以降から、カスタム JDBC べき等リポジトリーを構築するために拡張できる抽象クラス **org.apache.camel.processor.idempotent.jdbc.AbstractJdbcMessageIdRepository** があります。

最初に、べき等リポジトリーで使用されるデータベーステーブルを作成する必要があります。**Camel 2.7** では、次のスキーマを使用します。

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100) )
```

Camel 2.8 では、createdAt 列を追加しました。

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100), createdAt TIMESTAMP )
```

警告: SQL Server の **TIMESTAMP** 型は、固定長のバイナリー文字列型です。**DATE**、**TIME**、または **TIMESTAMP** のいずれの JDBC 時刻型にもマップされません。

Customize the JdbcMessageIdRepository

Camel 2.9.1 以降、必要に応じて **org.apache.camel.processor.idempotent.jdbc.JdbcMessageIdRepository** を調整するためのいくつかのオプションがあります。

パラメーター	デフォルト値	説明
createTableIfNotExists	true	テーブルが存在しない場合、Camel がテーブルを作成しようとするかどうかを定義します。
tableExistsString	SELECT 1 FROM CAMEL_MESSAGEPROCESSED WHERE 1 = 0	このクエリーは、テーブルがすでに存在するかどうかを確認するために使用されます。テーブルが存在しないことを示すには、例外を出力する必要があります。
createString	CREATE TABLE CAMEL_MESSAGEPROCESSED (processorName VARCHAR(255), messageId VARCHAR(100), createdAt TIMESTAMP)	テーブルの作成に使用されるステートメント。

パラメーター	デフォルト値	説明
queryString	SELECT COUNT(*) FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	メッセージがリポジトリーにすでに存在するかどうかを判断するために使用されるクエリー (結果は 0 に等しくない)。2 つのパラメーターを取ります。この最初のものはプロセッサー名 (**String**) で、2 つ目はメッセージ ID (**String**) です。
insertString	INSERT INTO CAMEL_MESSAGEPROCESSED (processorName, messageId, createdAt) VALUES (?, ?, ?)	エントリーをテーブルに追加するために使用されるステートメント。3 つのパラメーターを取ります。1 つ目はプロセッサー名 (**String**)、2 つ目はメッセージ ID (**String**)、3 つ目はこのエントリーがリポジトリーに追加されたときのタイムスタンプ (**java.sql.Timestamp**) です。

パラメーター	デフォルト値	説明
deleteString	DELETE FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	データベースからエントリーを削除するために使用されるステートメント。2 つのパラメーターを取ります。この最初のものはプロセッサー名 (**String**) で、2 つ目はメッセージ ID (**String**) です。

JDBC ベースの集約リポジトリーの使用

Camel 2.6 以降で利用可能

情報: **Camel 2.6 で JdbcAggregationRepository を使用する**

Camel 2.6 では、JdbcAggregationRepository が **camel-jdbc-aggregator** コンポーネントで提供されます。Camel 2.7 以降では、**JdbcAggregationRepository** が **camel-sql** コンポーネントで提供されます。

JdbcAggregationRepository は **AggregationRepository** であり、その場で集約されたメッセージを永続化します。デフォルトのアグリゲーターはメモリー内のみの **AggregationRepository** を使用するため、これによりメッセージが失われないことが保証されます。**JdbcAggregationRepository** を使用すると、Camel と一緒に Aggregator の永続的なサポートを提供できます。

エクスチェンジが正常に処理された場合にのみ、**AggregationRepository** で **confirm** メソッドが呼び出されたときに完了としてマークされます。これは、同じエクスチェンジが再び失敗した場合、成功するまで再試行されることを意味します。

オプション **maximumRedeliveries** を使用して、復元された特定の Exchange の再配信試行の最大回数を制限できます。**maximumRedeliveries** に達したときに Camel がエクスチェンジの送信先を認識できるように、**deadLetterUri** オプションも設定する必要があります。

このテスト など、camel-sql の単体テストでいくつかの例を確認できます。

データベース

操作可能にするために、各アグリゲーターは 2 つのテーブルを使用します: アグリゲーションと完了したテーブルです。慣例により、完了したものは、**"_COMPLETED"** という接尾辞が付いた集約と同じ名前になります。この名前は、Spring Bean で **RepositoryName** プロパティーを使用して設定する必要があります。次の例では、集約が使用されます。

両方のテーブルのテーブル構造定義は同一です。どちらの場合も、文字列値がキー (id) として使用されますが、ブロブにはバイト配列でシリアル化されたエクスチェンジが含まれます。
ただし、1 つの違いを覚えておく必要があります。テーブルによっては、id フィールドの内容が同じで

はありません。
集約テーブルの **id** は、コンポーネントがメッセージを集約するために使用する相関 ID を保持します。
完成したテーブルの **id** には、対応する ブロブフィールドに格納されているエクスチェンジの ID が保持されます。

テーブルの作成に使用される SQL クエリーを次に示します。**aggregation** をアグリゲーターリポジトリー名に置き換えてください。

```
CREATE TABLE aggregation ( id varchar(255) NOT NULL, exchange blob NOT
NULL, constraint aggregation_pk PRIMARY KEY (id) ); CREATE TABLE
aggregation_completed ( id varchar(255) NOT NULL, exchange blob NOT
NULL, constraint aggregation_completed_pk PRIMARY KEY (id) );
```

本文とヘッダーをテキストとして保存する

Camel 2.11 から利用可能

JdbcAggregationRepository を設定して、メッセージ本文と select (ed) ヘッダーを文字列として別々の列に格納できます。たとえば、本文を保存するには、次の 2 つのヘッダー **companyName** と **accountName** は次の SQL を使用します。

```
CREATE TABLE aggregationRepo3 ( id varchar(255) NOT NULL, exchange blob
NOT NULL, body varchar(1000), companyName varchar(1000), accountName
varchar(1000), constraint aggregationRepo3_pk PRIMARY KEY (id) ); CREATE
TABLE aggregationRepo3_completed ( id varchar(255) NOT NULL, exchange
blob NOT NULL, body varchar(1000), companyName varchar(1000),
accountName varchar(1000), constraint aggregationRepo3_completed_pk
PRIMARY KEY (id) );
```

次に、以下に示すように、この動作を有効にするようにリポジトリーを設定します。

```
<bean id="repo3"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="repositoryName" value="aggregationRepo3"/> <property
name="transactionManager" ref="txManager3"/> <property name="dataSource"
ref="dataSource3"/> <!-- configure to store the message body and
following headers as text in the repo --> <property
name="storeBodyAsText" value="true"/> <property
name="headersToStoreAsText"> <list> <value>companyName</value>
<value>accountName</value> </list> </property> </bean>
```

コーデック (シリアル化)

あらゆるタイプのペイロードを含めることができるため、エクスチェンジは設計上シリアライズできません。データベースのブロブフィールドに格納されるバイト配列に変換されます。これらの変換はすべて **JdbcCodec** クラスによって処理されます。コードの 1 つの詳細に注意する必要があります: **ClassLoadingAwareObjectInputStream** です。

ClassLoadingAwareObjectInputStream は、Apache ActiveMQ プロジェクトから再利用されています。**ObjectInputStream** をラップし、**currentThread** ではなく **ContextClassLoader** で使用します。利点は、他のバンドルによって公開されたクラスをロードできることです。これにより、交換の本文とヘッダーにカスタム型のオブジェクト参照を含めることができます。

Transaction

トランザクションを調整するには、Spring **PlatformTransactionManager** が必要です。

Service (Start/Stop)

start メソッドは、データベースの接続と必要なテーブルの存在を確認します。何か問題があると、起動時に失敗します。

アグリゲーターの設定

対象となる環境によっては、アグリゲーターに何らかの設定が必要になる場合があります。ご存知のように、各アグリゲーターには独自のリポジトリー (対応するテーブルのペアがデータベースに作成されている) とデータソースが必要です。デフォルトの lobHandler がデータベースシステムに適合していない場合は、**lobHandler** プロパティーを挿入できます。

Oracle の宣言は次のとおりです。

```
<bean id="lobHandler"
class="org.springframework.jdbc.support.lob.OracleLobHandler"> <property
name="nativeJdbcExtractor" ref="nativeJdbcExtractor"/> </bean> <bean
id="nativeJdbcExtractor"
class="org.springframework.jdbc.support.nativejdbc.CommonsDbcpNativeJdbcExtractor"/>
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="transactionManager" ref="transactionManager"/> <property
name="repositoryName" value="aggregation"/> <property name="dataSource"
ref="dataSource"/> <!-- Only with Oracle, else use default --> <property
name="lobHandler" ref="lobHandler"/> </bean>
```

Optimistic locking

Camel 2.12 以降では、**optimisticLocking** をオンにして、複数の Camel アプリケーションが集約リポジトリー用の同じデータベースを共有するクラスター化された環境で、この JDBC ベースの集約リポジトリーを使用できます。競合状態がある場合、JDBC ドライバーは、**JdbcAggregationRepository** が対応できるベンダー固有の例外を出力します。JDBC ドライバーからの例外の原因が楽観的ロックエラーと見なされるかを知るには、これを行うマッパーが必要です。したがって、**org.apache.camel.processor.aggregate.jdbc.JdbcOptimisticLockingExceptionMapper** があり、必要に応じてカスタムロジックを実装できます。次のように動作するデフォルトの実装 **org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper** があります。

次のチェックが行われます。

原因となった例外が **SQLException** の場合、SQLState が 23 で始まるかどうかがチェックされます。

原因となった例外が **DataIntegrityViolationException** の場合

発生した例外クラス名に ConstraintViolation が含まれる場合。

クラス名が設定されている場合は、FQN クラス名の一致をオプションでチェックします

さらに、FQN クラス名を追加できます。発生した例外 (またはネストされた例外) のいずれかが FQN クラス名のいずれかと等しい場合は、楽観的ロックエラーになります。

以下は、JDBC ベンダーからの 2 つの追加の FQN クラス名を定義する例です。

```
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="transactionManager" ref="transactionManager"/> <property
name="repositoryName" value="aggregation"/> <property name="dataSource"
ref="dataSource"/> <property name"jdbcOptimisticLockingExceptionMapper"
ref="myExceptionMapper"/> </bean> <!-- use the default mapper with extra
FQN class names from our JDBC driver --> <bean id="myExceptionMapper"
class="org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper">
<property name="classNames"> <util:set>
<value>com.foo.sql.MyViolationExceptoion</value>
<value>com.foo.sql.MyOtherViolationExceptoion</value> </util:set>
</property> </bean>
```

304.11. CAMEL SQL スターター

spring-boot ユーザーは、スターターモジュールを利用できます。スターターを使用する場合、spring-boot プロパティーを使用して **DataSource** を直接設定できます。

```
# Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

この機能を使用するには、Spring Boot pom.xml ファイルに次の依存関係を追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>${spring-boot-version}</version>
</dependency>
```

必要に応じて、特定のデータベースドライバーも含める必要があります。

304.12. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

SQL ストアドプロシージャー

JDBC

第305章 SQL STORED PROCEDURE コンポーネント

Camel バージョン 2.17 以降で利用可能

sql-stored: コンポーネントを使用すると、JDBC Stored Procedure クエリーを使用してデータベースを操作できます。このコンポーネントは SQL コンポーネント の拡張機能ですが、ストアドプロシージャーの呼び出しに特化しています。

このコンポーネントは、実際の SQL 処理のために舞台裏で **spring-jdbc** を使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

305.1. URI 形式

SQL コンポーネントは、次のエンドポイント URI 表記を使用します。

```
sql-stored:template[?options]
```

ここで、template はストアドプロシージャーテンプレートで、ストアドプロシージャーの名前と IN、INOUT、および OUT 引数を宣言します。

次のようなファイルシステムまたはクラスパス上の外部ファイルでテンプレートを参照することもできます。

```
sql-stored:classpath:sql/myprocedure.sql[?options]
```

sql/myprocedure.sql は、次のように、テンプレートを含むクラスパス内のプレーンテキストファイルです。

```
SUBNUMBERS(
  INTEGER ${headers.num1},
  INTEGER ${headers.num2},
  INOUT INTEGER ${headers.num3} out1,
  OUT INTEGER out2
)
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

305.2. オプション

SQL Stored Procedure コンポーネントは、次に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
dataSource (producer)	データベースとの通信に使用する DataSource を設定します。		DataSource
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SQL Stored Procedure エンドポイントは、URI 構文を使用して設定されます。

```
sql-stored:template
```

パスおよびクエリーパラメーターを使用します。

305.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
template	**必須** 実行する StoredProcedure テンプレートを設定します。		String

305.2.2. クエリーパラメーター (7 個のパラメーター):

名前	説明	デフォルト	タイプ
batch (producer)	バッチモードを有効または無効にします	false	boolean
dataSource (producer)	データベースとの通信に使用する DataSource を設定します。		DataSource
function (producer)	この呼び出しが関数に対するものかどうか。	false	boolean
noop (producer)	設定されている場合、テンプレートの結果を無視し、既存の IN メッセージを OUT メッセージとして使用して処理を続行します。	false	boolean

名前	説明	デフォルト	タイプ
outputHeader (producer)	テンプレートの結果をメッセージボディーではなくヘッダーに格納します。デフォルトでは、outputHeader == null で、テンプレートの結果はメッセージ本文に格納され、メッセージ本文の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はテンプレートの結果を格納するヘッダーの名前として使用され、元のメッセージ本文は保持されます。		String
useMessageBodyForTemplate (producer)	メッセージボディーをテンプレートとして使用し、次にパラメーターのヘッダーを使用するかどうか。このオプションを有効にすると、URI のテンプレートは使用されません。	false	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

305.3. ストアドプロシージャーテンプレートの宣言

テンプレートは、Java メソッドシグネチャーに似た構文を使用して宣言されます。ストアドプロシージャーの名前と、括弧で囲まれた引数。例はこれをよく説明しています:

```
<to uri="sql-stored:STOREDSAMPLE(INTEGER ${headers.num1},INTEGER
${headers.num2},INOUT INTEGER ${headers.num3} result1,OUT INTEGER result2)"/>
```

引数は型によって宣言され、単純な式を使用して Camel メッセージにマッピングされます。したがって、この例では、最初の 2 つのパラメーターは INTEGER 型の IN 値であり、メッセージヘッダーにマップされます。3 番目のパラメーターは INOUT です。これは、INTEGER を受け入れてから別の INTEGER 結果を返すことを意味します。最後のパラメーターは OUT 値で、これも INTEGER 型です。

SQL 用語では、ストアドプロシージャーは次のように宣言できます。

```
CREATE PROCEDURE STOREDSAMPLE(VALUE1 INTEGER, VALUE2 INTEGER, INOUT
RESULT1 INTEGER, OUT RESULT2 INTEGER)
```

305.3.1. IN パラメーター

IN パラメーターは、スペースで区切られた 4 つの部分 (パラメーター名、SQL 型 (位取りあり)、型名、および値のソース) を取ります。

パラメーター名はオプションであり、指定されていない場合は自動生成されます。引用符 (') の間に指定する必要があります。

SQL タイプは必須であり、整数 (正または負) または一部のクラスの整数フィールドへの参照にすることができます。SQL 型にドットが含まれている場合、コンポーネントはそのクラスを解決して、指定されたフィールドを読み取ろうとします。たとえば、SQL 型 com.Foo.INTEGER は、クラス com.Foo の

フィールド INTEGER から読み取られます。型にコンマが含まれていない場合、整数値を解決するクラスは java.sql.Types になります。型は位取りによって接尾辞を付けることができます。たとえば、DECIMAL(10) は位取り 10 の java.sql.Types.DECIMAL を意味します。

型名はオプションで、引用符 (') で囲む必要があります。

値のソースが必要です。値のソースは、Exchange からパラメーター値を取り込みます。Simple 式またはヘッダーの場所、つまり :#<header name>. のいずれかです。たとえば、Simple 式 ${header.val} は、パラメーター値がヘッダー val から読み取られることを意味します。ヘッダー位置式 :#val は同じ効果があります。

```
<to uri="sql-stored:MYFUNC('param1' org.example.Types.INTEGER(10) ${header.srcValue})"/>
```

URI は、ストアドプロシージャーがパラメーター名 param1 で呼び出されることを意味します。その SQL タイプはクラス org.example.Types フィールド INTEGER から読み取られ、スケールは 10 に設定されます。パラメーターの入力値は、ヘッダー srcValue から渡されます。

```
<to uri="sql-stored:MYFUNC('param1' 100 'mytypename' ${header.srcValue})"/>
```

URI は、SQL-type が 100 で型名が mytypename であることを除いて、前と同じです。

実際の呼び出しは org.springframework.jdbc.core.SqlParameter を使用して行われます。

305.3.2. OUT パラメーター

OUT パラメーターは IN パラメーターと同様に機能し、SQL 型 (位取り付き)、型名、および出力パラメーター名の 3 つの部分を含みます。

SQL タイプは、IN パラメーターと同じように機能します。

タイプ名はオプションであり、IN パラメーターと同じように機能します。

出力パラメーター名は、OUT パラメーター名と、結果が格納されるヘッダー名に使用されます。

```
<to uri="sql-stored:MYFUNC(OUT org.example.Types.DECIMAL(10) outheader1)"/>
```

URI は、OUT パラメーターの名前が outheader1 であり、結果がヘッダー outheader1 になることを意味します。

```
<to uri="sql-stored:MYFUNC(OUT org.example.Types.NUMERIC(10) 'mytype' outheader1)"/>
```

これは前のものと同じですが、タイプ名は mytype になります。

実際の呼び出しは org.springframework.jdbc.core.SqlOutParameter を使用して行われます。

305.3.3. INOUT パラメーター

INOUT パラメーターは、上記のすべての組み合わせです。エクスチェンジから値を受け取り、結果をメッセージヘッダーとして保存します。唯一の注意点は、IN パラメーターの名前がスキップされることです。代わりに、OUT パラメーターの名前は、SQL パラメーター名と結果ヘッダー名の両方を定義します。

```
<to uri="sql-stored:MYFUNC(INOUT DECIMAL(10) ${headers.inheader} outheader)"/>
```

Actual call will be done using org.springframework.jdbc.core.SqlInOutParameter.

305.4. CAMEL SQL スターター

spring-boot ユーザーは、スターターモジュールを利用できます。スターターを使用する場合、spring-boot プロパティーを使用して **DataSource** を直接設定できます。

```
# Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

この機能を使用するには、Spring Boot pom.xml ファイルに次の依存関係を追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>${spring-boot-version}</version>
</dependency>
```

必要に応じて、特定のデータベースドライバーも含める必要があります。

305.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- SQL コンポーネント

第306章 SSH コンポーネント

Camel バージョン 2.10 以降で利用可能

SSH コンポーネントを使用すると、SSH コマンドを送信してレスポンスを処理できるように、SSH サーバーにアクセスできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ssh</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

306.1. URI 形式

```
ssh:[username[:password]@]host[:port][?options]
```

306.2. オプション

SSH コンポーネントは、以下に示す 12 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 SSH 設定を使用するには		SshConfiguration
host (Common)	リモート SSH サーバーのホスト名を設定します。		String
port (Common)	リモート SSH サーバーのポート番号を設定します。		int
username (security)	リモート SSH サーバーへのログインに使用するユーザー名を設定します。		String
password (security)	リモート SSH サーバーへの接続に使用するパスワードを設定します。keyPairProvider を null に設定する必要があります。		String
pollCommand (Common)	各ポーリングサイクル中にリモート SSH サーバーに送信するコマンド文字列を設定します。コンシューマーとして使用されている camel-ssh コンポーネント、つまり from (ssh://...) でのみ機能します。コマンドを改行で終了する必要がある場合があり、それは URL エンコードされた %0A でなければなりません		String

名前	説明	デフォルト	タイプ
keyPairProvider (security)	証明書を使用してリモート SSH サーバーに接続するときに使用する KeyPairProvider 参照を設定します。		KeyPairProvider
keyType (security)	認証の一部として KeyPairProvider に渡すキータイプを設定します。KeyPairProvider.loadKey(...) にこの値が渡されます。デフォルトは ssh-rsa です。		String
timeout (common)	リモート SSH サーバー接続を確立する際に待機するタイムアウトをミリ秒単位で設定します。デフォルトは 30000 ミリ秒 (ms) です。		long
certFilename (security)	**非推奨** 認証に使用する証明書のリソースパスを設定します。		String
certResource (security)	認証に使用する証明書のリソースパスを設定します。ResourceHelperKeyPairProvider を使用してファイルベースの証明書を解決し、keyType 設定に依存します。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SSH エンドポイントは、URI 構文を使用して設定されます。

```
ssh:host:port
```

パスおよびクエリーパラメーターを使用します。

306.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
host	**必須** リモート SSH サーバーのホスト名を設定します。		String
port	リモート SSH サーバーのポート番号を設定します。	22	int

306.2.2. クエリーパラメーター(28 パラメーター):

名前	説明	デフォルト	タイプ
failOnUnknownHost (Common)	不明なホストへの接続が失敗するかどうかを指定します。この値は、プロパティー knownHosts が設定されている場合にのみチェックされます。	false	boolean
knownHostsResource (Common)	known_hosts ファイルのリソースパスを設定します		String
timeout (common)	リモート SSH サーバー接続を確立する際に待機するタイムアウトをミリ秒単位で設定します。デフォルトは 30000 ミリ秒 (ms) です。	30000	long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollCommand (consumer)	各ポーリングサイクル中にリモート SSH サーバーに送信するコマンド文字列を設定します。コンシューマーとして使用されている camel-ssh コンポーネント、つまり from(ssh://...) でのみ機能します。コマンドの最後に改行が必要な場合がありますが、これは URL エンコードされた %0A でなければなりません。		String
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
certResource (security)	認証に使用する証明書のリソースパスを設定します。ResourceHelperKeyPairProvider を使用してファイルベースの証明書を解決し、keyType 設定に依存します。		String
keyPairProvider (security)	証明書を使用してリモート SSH サーバーに接続するときに使用する KeyPairProvider 参照を設定します。		KeyPairProvider
keyType (security)	認証の一部として KeyPairProvider に渡すキータイプを設定します。KeyPairProvider.loadKey(...) にこの値が渡されます。デフォルトは ssh-rsa です。	ssh-rsa	String
password (security)	リモート SSH サーバーへの接続に使用するパスワードを設定します。keyPairProvider を null に設定する必要があります。		String
username (security)	リモート SSH サーバーへのログインに使用するユーザー名を設定します。		文字列

306.3. PRODUCER エンドポイントとしての使用

SSH コンポーネントがプロデューサー (**.to ("ssh://...")**) として使用される場合、リモート SSH サーバーで実行するコマンドとしてメッセージボディーが送信されます。

XML DSL 内での例を次に示します。コマンドには、XML エンコードされた改行 (**
**) があることに注意してください。

```
<route id="camel-example-ssh-producer">
```

```
  <from uri="direct:exampleSshProducer"/>
  <setBody>
    <constant>features:list&amp;#10;</constant>
  </setBody>
  <to uri="ssh://karaf:karaf@localhost:8101"/>
  <log message="${body}"/>
</route>
```

306.4. 認証

SSH コンポーネントは、公開鍵証明書またはユーザー名/パスワードの 2 つのメカニズムのいずれかを使用して、リモート SSH サーバーに対して認証できます。SSH コンポーネントが認証を行う方法の設定は、どのオプションがどのように設定されているかに基づいています。

1. 最初に、**certResource** オプションが設定されているかどうかを確認し、設定されている場合は、それを使用して参照されている公開鍵証明書を見つけ、それを認証に使用します。

2. **certResource** が設定されていない場合は、**keyPairProvider が** 設定されているかどうかを確認し、設定されている場合はそれを証明書ベースの認証に使用します。

3. **certResource** も **keyPairProvider** も設定されていない場合、認証には **username** と **password** のオプションが使用されます。**username** と **password** がエンドポイント設定で提供され、ヘッダーが **SshConstants.USERNAME_HEADER** (**CamelSshUsername**) および **SshConstants.PASSWORD_HEADER** (**CamelSshPassword**) で設定されている場合でも、エンドポイント設定が優先され、ヘッダーに設定された認証情報が使用されます。

次のルートフラグメントは、クラスパスからの証明書を使用する SSH ポーリングコンシューマーを示しています。

XML DSL では、

```
<route>
  <from uri="ssh://scott@localhost:8101?
certResource=classpath:test_rsa&amp;useFixedDelay=true&amp;delay=5000&amp;pollCommand=features:list%0A"/>
  <log message="${body}"/>
</route>
```

Java DSL では、

```
from("ssh://scott@localhost:8101?
certResource=classpath:test_rsa&useFixedDelay=true&delay=5000&pollCommand=features:list%0A"
)
    .log("${body}");
```

公開鍵認証の使用例は、**examples/camel-example-ssh-security** で提供されています。

証明書の依存関係

証明書ベースの認証を使用する場合は、ランタイムの依存関係をいくつか追加する必要があります。表示されている依存バージョンは Camel 2.11 のものです。使用している Camel のバージョンによっては、それ以降のバージョンを使用する必要がある場合があります。

```
<dependency>
  <groupId>org.apache.sshd</groupId>
```

```
  <artifactId>sshd-core</artifactId>
  <version>0.8.0</version>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcpg-jdk15on</artifactId>
  <version>1.47</version>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcpkix-jdk15on</artifactId>
  <version>1.47</version>
</dependency>
```

306.5. 例

Camel ディストリビューションの **examples/camel-example-ssh** と **examples/camel-example-ssh-security** を参照してください。

306.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第307章 STAX コンポーネント

Camel バージョン 2.9 以降で利用可能

StAX コンポーネントを使用すると、SAX ContentHandler を介してメッセージを処理できます。
このコンポーネントのもう 1 つの機能は、Splitter EIP などを使用して、StAX を使用して JAXB レコードを反復処理できるようにすることです。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stax</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

307.1. URI 形式

```
stax:content-handler-class
```

例

```
stax:org.superbiz.FooContentHandler
```

Camel 2.11.1 以降では、次のように # 構文を使用してレジストリーから **org.xml.sax.ContentHandler** Bean を検索できます。

```
stax:#myHandler
```

307.2. オプション

StAX コンポーネントにはオプションがありません。

StAX エンドポイントは、URI 構文を使用して設定されます。

```
stax:contentHandlerClass
```

パスおよびクエリーパラメーターを使用します。

307.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
contentHandlerClass	**必須** ContentHandler 実装が使用する FQN クラス名。		String

307.2.2. クエリーパラメーター(1 個のパラメーター):

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

307.3. STAX パーサーとしてのコンテンツハンドラーの使用

処理後のメッセージボディーはハンドラーそのものです。

例は次のとおりです:

```
from("file:target/in")
  .to("stax:org.superbiz.handler.CountingHandler")
  // CountingHandler implements org.xml.sax.ContentHandler or extends
org.xml.sax.helpers.DefaultHandler
  .process(new Processor() {
    @Override
    public void process(Exchange exchange) throws Exception {
        CountingHandler handler = exchange.getIn().getBody(CountingHandler.class);
        // do some great work with the handler
    }
  });
```

307.4. JAXB と STAX を使用してコレクションを反復処理する

まず、JAXB オブジェクトがあるとします。

たとえば、ラッパーオブジェクト内のレコードのリスト:

```
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "records")
public class Records {
    @XmlElement(required = true)
    protected List<Record> record;

    public List<Record> getRecord() {
        if (record == null) {
            record = new ArrayList<Record>();
        }
        return record;
    }
}
```

および

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "record", propOrder = { "key", "value" })
public class Record {
    @XmlAttribute(required = true)
    protected String key;

    @XmlAttribute(required = true)
    protected String value;

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

次に、処理する XML ファイルを取得します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<records>
  <record value="v0" key="0"/>
  <record value="v1" key="1"/>
  <record value="v2" key="2"/>
  <record value="v3" key="3"/>
  <record value="v4" key="4"/>
  <record value="v5" key="5"/>
</record>
```

StAX コンポーネントは、Camel Splitter で XML 要素を反復するときに使用できる **StAXBuilder** を提供します。

```
from("file:target/in")
    .split(stax(Record.class)).streaming()
        .to("mock:records");
```

stax は、Java コードで静的にインポートできる **org.apache.camel.component.stax.StAXBuilder** の静的メソッドです。stax ビルダーは、デフォルトで、それが使用する XMLReader の名前空間を認識します。**Camel 2.11.1** 以降では、以下に示すように、boolean パラメーターを false に設定することでこれ

をオフにすることができます。

```
from("file:target/in")
    .split(stax(Record.class, false)).streaming()
        .to("mock:records");
```

307.4.1. XML DSL を使用した前の例

上記の例は、XML DSL で次のように実装できます。

307.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第308章 STOMP コンポーネント

Camel バージョン 2.12 以降で利用可能

stomp: コンポーネントは、Apache ActiveMQ や ActiveMQ Apollo などの Stomp 準拠のメッセージブローカーと通信するために使用されます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stomp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

308.1. URI 形式

```
stomp:queue:destination[?options]
```

destination はキューの名前です。

308.2. オプション

Stomp コンポーネントは、以下に示す 8 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 stomp 設定を使用する場合。		StompConfiguration
brokerURL (common)	接続する Stomp ブローカーの URI。		String
login (security)	ユーザー名		String
passcode (security)	パスワード		String
host (Common)	仮想ホスト		String
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Stomp エンドポイントは、URI 構文を使用して設定されます。

```
stomp:destination
```

パスおよびクエリーパラメーターを使用します。

308.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
destination	**必須** キューの名前		String

308.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
brokerURL (common)	**必須** 接続先の Stomp ブローカーの URI。	tcp://localhost:61613	String
host (Common)	仮想ホスト名。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
login (security)	ユーザー名		String
passcode (security)	パスワード		String
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

308.3. サンプル

メッセージの送信:

```
from("direct:foo").to("stomp:queue:test");
```

メッセージの消費:

```
from("stomp:queue:test").transform(body().convertToString()).to("mock:result")
```

308.4. エンドポイント

Camel は、エンドポイント インターフェイスを使用してメッセージエンドポイントパターンをサポートします。通常、エンドポイントはコンポーネントによって作成され、エンドポイントは通常、URI を介して DSL で参照されます。

エンドポイントから、次のメソッドを使用できます

* createProducer() は、メッセージエクスチェンジをエンドポイントに送信するための プロデューサー

を作成します * createConsumer() は、コンシューマー の作成時に プロセッサー を介してエンドポイントからのメッセージエクスチェンジを消費するためのイベントドリブンコンシューマーパターンを実装します * createPollingConsumer() は、PollingConsumer を介したエンドポイントからのメッセージエクスチェンジ消費のためのポーリングコンシューマーパターンを実装します。

308.5. 関連項目

- Configuring Camel (Camel の設定)
- Message Endpoint パターン
- URI
- コンポーネントの作成

第309章 STREAM コンポーネント

Camel バージョン 1.3 以降で利用可能

stream: コンポーネントは、**System.in**、**System.out**、および **System.err** ストリームへのアクセスを提供し、ファイルと URL のストリーミングを可能にします。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

309.1. URI 形式

```
stream:in[?options]
stream:out[?options]
stream:err[?options]
stream:header[?options]
```

さらに、**file** と **URL** のエンドポイント URI がサポートされています。

```
stream:file?fileName=/foo/bar.txt
stream:url[?options]
```

stream:header URI が指定されている場合、**stream** ヘッダーを使用して書き込み先のストリームを検索します。このオプションは、ストリームプロデューサーでのみ使用できます (つまり、**from()** には表示されません)。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

309.2. オプション

Stream コンポーネントにはオプションがありません。

Stream エンドポイントは、URI 構文を使用して設定されます。

```
stream:kind
```

パスおよびクエリーパラメーターを使用します。

309.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
kind	**必須** System.in や System.out など、使用するストリームの種類。		文字列

309.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
encoding (common)	エンコーディング (文字セット名) を設定して、テキストベースのストリームを使用できます (たとえば、メッセージボディーは String オブジェクトです)。指定しない場合、Camel は JVM のデフォルトの文字セットを使用します。		String
fileName (Common)	stream:file URI 形式を使用する場合、このオプションはストリーミング先またはストリーミング元のファイル名を指定します。		String
url (common)	stream:url URI 形式を使用する場合、このオプションは、ストリーミング先/ストリーミング元の URL を指定します。入出力ストリームは、JDK URLConnection 機能を使用して開かれます。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
groupLines (consumer)	コンシューマーで X 行をグループ化します。たとえば、10 行をグループ化して、行ごとに 1 つの Exchange ではなく、10 行の Exchange のみを出力する場合。		int
groupStrategy (consumer)	カスタム GroupStrategy を使用して、行をグループ化する方法を制御できます。		GroupStrategy
initialPromptDelay (consumer)	メッセージプロンプトを表示するまでのミリ秒単位の初期遅延。この遅延は 1 回だけ発生します。システムの起動時に使用して、システム出力への他のロギングが行われている間にメッセージプロンプトが書き込まれるのを回避できます。	2000	long
promptDelay (consumer)	メッセージプロンプトを表示するまでのオプションの遅延 (ミリ秒単位)。		long
promptMessage (consumer)	stream:in; からの読み取り時に使用するメッセージプロンプト。たとえば、コマンドを入力するように設定できます。		String

名前	説明	デフォルト	タイプ
retry (consumer)	上書きすると、tail --retry のようにファイルを開くことを再試行します。	false	boolean
scanStream (consumer)	unix tail コマンドなどのストリームを連続して読み取るために使用します。	false	boolean
scanStreamDelay (consumer)	scanStream 使用時の読み取り試行間のミリ秒単位の遅延。		long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
autoCloseCount (producer)	プロデューサー側でストリームを閉じる前に処理するメッセージの数。デフォルトではストリームを閉じない (プロデューサーが停止している場合のみ)。さらにメッセージが送信されると、別の autoCloseCount バッチのためにストリームが再度開かれます。		int
closeOnDone (producer)	このオプションは、Splitter および同じファイルへのストリーミングと組み合わせて使用されます。そのアイデアは、パフォーマンスを向上させるために、ストリームを開いたままにし、スプリッターが完了したときにのみ閉じることです。これには、2 つ以上のファイルではなく、同じファイルにのみストリーミングする必要があることに注意してください。	false	boolean
delay (producer)	ストリームを生成する前のミリ秒単位の初期遅延。		long
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

309.3. メッセージ内容

stream: コンポーネントは、ストリームへの書き込み用に **String** または **byte[]** のいずれかをサポートします。**String** または **byte[]** コンテンツを **message.in.body** に追加するだけです。**ストリーム** に送信されるメッセージ: バイナリーモードのプロデューサーは、(**String** メッセージとは対照的に) 改行文字が続きません。**null** 本文のメッセージは、出力ストリームに追加されません。

カスタム出力ストリームには、特別な **stream:header** URI が使用されます。キー **header** の **message.in.header** に **java.io.OutputStream** オブジェクトを追加するだけです。
例については、サンプルを参照してください。

309.4. サンプル

次のサンプルでは、メッセージを **direct:in** エンドポイントから **System.out** ストリームにルーティングします。

```
// Route messages to the standard output.
from("direct:in").to("stream:out");

// Send String payload to the standard output.
// Message will be followed by the newline.
template.sendBody("direct:in", "Hello Text World");

// Send byte[] payload to the standard output.
// No newline will be added after the message.
template.sendBody("direct:in", "Hello Bytes World".getBytes());
```

次のサンプルは、使用するストリームを決定するためにヘッダーの種類を使用する方法を示しています。サンプルでは、独自の出力ストリーム **MyOutputStream** を使用します。

次のサンプルは、ファイルストリームを連続して読み取る方法を示しています (UNIX の **tail** コマンドに似ています)。

```
from("stream:file?
fileName=/server/logs/server.log&scanStream=true&scanStreamDelay=1000").to("bean:logService?
method=parseLogLine");
```

scanStream (Camel 2.7 より前) または scanStream + retry の 1 つの落とし穴は、ファイルが再度開かれ、scanStreamDelay の反復ごとにスキャンされることです。NIO2 が利用可能になるまで、ファイルが削除/再作成されたことを確実に検出することはできません。

309.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第310章 STRING ENCODING DATAFORMAT

Camel バージョン 2.12 以降で利用可能

文字列データ形式は、エンコーディングをサポートするテキストベースの形式です。

310.1. オプション

String Encoding データ形式は、以下に示す 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
charset		String	使用するエンコーディングを設定します。デフォルトでは、JVM プラットフォームのデフォルトの文字セットを使用します。
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

310.2. MARSHAL

この例では、ファイルの内容を UTF-8 エンコーディングの String オブジェクトにマーシャリングします。

```
from("file://data.csv").marshal().string("UTF-8").to("jms://myqueue");
```

310.3. UNMARSHAL

この例では、newOrder プロセッサーによって処理される前に、UTF-8 エンコーディングを使用して JMS キューから String オブジェクトにペイロードをアンマーシャリングします。

```
from("jms://queue/order").unmarshal().string("UTF-8").processRef("newOrder");
```

310.4. 依存関係

このデータ形式は **camel-core** で提供されるため、追加の依存関係は必要ありません。

第311章 文字列テンプレートコンポーネント

Camel バージョン 1.2 以降で利用可能

string-template: コンポーネントを使用すると、String Template を使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stringtemplate</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

311.1. URI 形式

```
string-template:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI です。またはリモートテンプレートの完全な URL。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

311.2. オプション

String Template コンポーネントにはオプションがありません。

文字列テンプレートエンドポイントは、URI 構文を使用して設定されます。

```
string-template:resourceUri
```

パスおよびクエリーパラメーターを使用します。

311.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		文字列

311.2.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
delimiterStart (producer)	可変開始区切り文字	<	char
delimiterStop (producer)	可変終了区切り文字	>	char
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

311.3. ヘッダー

Camel は、キー **org.apache.camel.stringtemplate.resource** を使用してメッセージヘッダーにリソースへの参照を格納します。Resource は **org.springframework.core.io.Resource** オブジェクトです。

311.4. ホットリロード

文字列テンプレートリソースは、デフォルトで、ファイルリソースとクラスパスリソース (拡張 jar) の両方でホットリロード可能です。**contentCache=true** を設定すると、Camel はリソースを 1 回だけロードし、ホットリロードはできません。このシナリオは、リソースが変更されない場合に実稼働で使用できます。

311.5. STRINGTEMPLATE 属性

Camel 2.14 以降、以下のコードのようにメッセージヘッダー " **CamelStringTemplateVariableMap** " を設定することで、カスタムコンテキストマップを定義できます。

```
Map<String, Object> variableMap = new HashMap<String, Object>();
Map<String, Object> headersMap = new HashMap<String, Object>();
headersMap.put("name", "Willem");
variableMap.put("headers", headersMap);
variableMap.put("body", "Monday");
variableMap.put("exchange", exchange);
exchange.getIn().setHeader("CamelStringTemplateVariableMap", variableMap);
```

311.6. サンプル

たとえば、メッセージへの応答を作成するために、次のように文字列テンプレートを使用できます。

```
from("activemq:My.Queue").
  to("string-template:com/acme/MyResponse.tm");
```

311.7. 電子メールのサンプル

このサンプルでは、文字列テンプレートを使用して注文確認メールを送信します。電子メールテンプレートは **StringTemplate** で次のようにレイアウトされます。この例は **camel 2.11.0** で機能します。camel バージョンが 2.11.0 未満の場合、変数は $ で開始および終了する必要があります。

```
Dear <headers.lastName>, <headers.firstName>

Thanks for the order of <headers.item>.

Regards Camel Riders Bookstore
<body>
```

Java コードは次のとおりです。

311.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第312章 スタブコンポーネント

Camel バージョン 2.10 以降で利用可能

stub: コンポーネントでは、開発中またはテスト中に物理エンドポイントを簡単にスタブ化する方法を提供します。これにより、特定の SMTP または Http エンドポイントなどに実際に接続する必要なくルートを実行できます。エンドポイント URI の前に **stub:** を追加するだけで、エンドポイントをスタブ化できます。

内部的に、Stub コンポーネントは 仮想マシン エンドポイントを作成します。Stub と 仮想マシン の主な違いは、仮想マシン は指定した URI とパラメーターを検証するため、クエリー引数を含む一般的な URI の前に vm: を配置すると、通常は失敗することです。ただし、Stub はそうではありません。基本的にすべてのクエリーパラメーターを無視して、ルート内の 1 つ以上のエンドポイントを一時的にすばやく stub 化できるためです。

312.1. URI 形式

```
stub:someUri
```

someUri は、任意のクエリーパラメーターを持つ任意の URI にすることができます。

312.2. オプション

Stub コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
queueSize (advanced)	SEDA キューのデフォルトの最大容量 (つまり、保持できるメッセージの数) を設定します。		int
concurrentConsumers (consumer)	交換を処理する同時スレッドのデフォルト数を設定します。	1	int
defaultQueueFactory (advanced)	デフォルトのキューファクトリーを設定します。		Exchange>
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Stub エンドポイントは、URI 構文を使用して設定されます。

```
stub:name
```

パスおよびクエリーパラメーターを使用します。

312.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** キューの名前		文字列

312.2.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
size (common)	SEDA キューの最大容量 (つまり、保持できるメッセージの数)。	2147483647	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
limitConcurrentConsumers (consumer)	concurrentConsumers の数を最大 500 に制限するかどうか。デフォルトでは、エンドポイントがより大きな数で設定されている場合、例外が出力されます。このチェックを無効にするには、このオプションをオフにします。	true	boolean
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe メッセージングに SEDA を使用できます。つまり、メッセージを SEDA キューに送信し、各コンシューマーにメッセージのコピーを受信させることができます。有効にすると、このオプションはすべてのコンシューマーエンドポイントで指定する必要があります。	false	boolean

名前	説明	デフォルト	タイプ
pollTimeout (consumer)	ポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
purgeWhenStopping (consumer)	コンシューマー/ルートを停止するときにタスクキューをパージするかどうか。これにより、キューに保留中のメッセージが破棄されるため、より迅速に停止できます。	false	boolean
blockWhenFull (producer)	満杯の SEDA キューにメッセージを送信するスレッドが、キューの容量がなくなるまでブロックするかどうか。デフォルトでは、キューがいっぱいであることを示す例外が出力されます。このオプションを有効にすると、呼び出しスレッドは代わりにブロックされ、メッセージが受け入れられるまで待機します。	false	boolean
discardIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーがメッセージを破棄する (メッセージをキューに追加しない) かどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーが例外を出力して失敗するかどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
timeout (producer)	SEDA プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト (ミリ秒単位)。0 または負の値を使用して、タイムアウトを無効にすることができます。	30000	long
waitForTaskToComplete (producer)	続行する前に呼び出し元が非同期タスクの完了を待つかどうかを指定するオプション。次の 3 つのオプションがサポートされています: Always、Never、または IfReplyExpected。最初の 2 つの値は一目瞭然です。最後の値 IfReplyExpected は、メッセージが Request Reply ベースの場合にのみ待機します。デフォルトのオプションは IfReplyExpected です。	IfReplyExpected	WaitForTaskToComplete

名前	説明	デフォルト	タイプ
queue (advanced)	エンドポイントで使用されるキューインスタンスを定義します。このオプションは、カスタムキューインスタンスを使用するまれなユースケースのみを対象としています。		BlockingQueue
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

312.3. 例

エンドポイント URI の stub 化のサンプルをいくつか示します。

```
stub:smtp://somehost.foo.com?user=whatnot&something=else
stub:http://somehost.bar.com/something
```

第313章 SWAGGER JAVA コンポーネント

Camel 2.16 以降で利用可能

Rest DSL は、Swagger を使用して REST サービスとその API を公開するために使用される **camel-swagger-java** モジュールと統合できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

Camel 2.16 以降、swagger コンポーネントは純粋に Java ベースであり、

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-swagger-java</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-swagger-java モジュールは、REST コンポーネントから使用できます (サーブレットは必要ありません)。

例については、Apache Camel ディストリビューションのサンプルディレクトリーにある **camel-example-swagger-cdi** を参照してください。

313.1. REST-DSL での SWAGGER の使用

以下に示すように、**apiContextPath** dsl を設定して、rest-dsl から swagger api を有効にすることができます。

```
public class UserRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {
    // configure we want to use servlet as the component for the rest DSL
    // and we enable json binding mode
    restConfiguration().component("netty4-http").bindingMode(RestBindingMode.json)
      // and output using pretty print
      .dataFormatProperty("prettyPrint", "true")
      // setup context path and port number that netty will use
      .contextPath("/").port(8080)
      // add swagger api-doc out of the box
      .apiContextPath("/api-doc")
        .apiProperty("api.title", "User API").apiProperty("api.version", "1.2.3")
        // and enable CORS
        .apiProperty("cors", "true");

    // this user REST service is json only
    rest("/user").description("User rest service")
      .consumes("application/json").produces("application/json")
      .get("/{id}").description("Find user by id").outType(User.class)
        .param().name("id").type(path).description("The id of the user to
get").dataType("int").endParam()
        .to("bean:userService?method=getUser(${header.id})")
      .put().description("Updates or create a user").type(User.class)
        .param().name("body").type(body).description("The user to update or create").endParam()
        .to("bean:userService?method=updateUser")
```

```
            .get("/findAll").description("Find all users").outTypeList(User.class)
                .to("bean:userService?method=listUsers");
    }
}
```

313.2. オプション

swagger モジュールは、次のオプションを使用して設定できます。サーブレットを使用して設定するには、上記のように init-param を使用します。rest-dsl で直接設定する場合は、**enableCORS**、**host,contextPath**、dsl などの適切なメソッドを使用します。**api.xxx** のオプションは、**apiProperty** を使用して設定されます。

オプション	タイプ	説明
cors	Boolean	CORS を有効にするかどうか。これにより、REST サービスへの実際のアクセスではなく、API ブラウザーの CORS のみが有効になることに注意してください。デフォルトは false です。このオプションを使用する代わりに、CorsFilte を使用することを推奨します。詳細は以下を参照してください。
swagger.version	String	Swagger spec バージョン。デフォルトは 2.0 です。
host	String	ホスト名を設定します。設定されていない場合、camel-swagger-java は名前を localhost ベースとして計算します。
schemas	String	使用するプロトコルスキーム。複数の値は、"http,https" のようにコンマで区切ることができます。デフォルト値は「http」です。このオプションは、スキームという名前にする必要があったため、Camel 2.17 以降では **非推奨です**。
schemes	String	**Camel 2.17:** 使用するプロトコルスキーム。複数の値は、"http,https" のようにコンマで区切ることができます。デフォルト値は「http」です。
base.path	String	**必須**: REST サービスを利用できるベースパスを設定します。パスは相対パス (例: http/https で始まらない) であり、camel-swagger-java は実行時に絶対ベースパスを計算します (**protocol://host:port/context-path/base.path**)。
api.path	String	API が利用可能なパスを設定します (例:/api-docs)。パスは相対パス (例: http/https で始まらない) であり、camel-swagger-java は実行時に絶対ベースパスを計算します。このパスは **protocol://host:port/context-path/api.path** で、相対パスを使用するほうがはるかに簡単です。例については、上記を参照してください。
api.version	String	API のバージョン。デフォルトは 0.0.0 です。
api.title	String	アプリケーションの名前。

オプション	タイプ	説明
api.description	String	アプリケーションの簡単な説明。
api.termsOfService	String	API の利用規約への URL。
api.contact.name	String	連絡先に使用する個人または組織の名前
api.contact.email	String	API 関連の連絡に使用するメール。
api.contact.url	String	API 関連の問い合わせ先の Web サイトへの URL。
api.license.name	String	API に適用されるライセンス名。
api.license.url	String	API に使用されるライセンスへの URL。
apiContextIdListing	boolean	REST サービスを持つ JVM ですべての CamelContext 名のリストを許可するかどうか。有効にすると、api-doc のルートパスにすべてのコンテキストが一覧表示されます。無効にすると、コンテキスト ID はリストされず、api-doc のルートパスに現在の CamelContext がリストされます。デフォルトは false です。
apiContextIdPattern	String	コンテキストリストに表示される CamelContext 名をフィルタリングできるパターン。パターンは正規表現と * をワイルドカードとして使用しています。Intercept で使用されるのと同じパターンマッチングです。

313.3. CONTEXTIDLISTING が有効

contextIdListing が有効になっている場合、同じ JVM で実行中のすべての CamelContext が検出されます。これらのコンテキストは、**/api-docs** などのルートパスに、json 形式の名前の単純なリストとしてリストされます。swagger ドキュメントにアクセスするには、context-path に Camel コンテキスト ID (**api-docs/myCamel** など) を追加する必要があります。オプション apiContextIdPattern を使用して、このリスト内の名前をフィルタリングできます。

313.4. JSON または YAML

Camel 2.17 以降で利用可能

camel-swagger-java モジュールは、そのままで JSON と Yaml の両方をサポートします。

/swagger.json または/swagger.yaml のいずれかを使用して、返されるものを要求 URL で指定できます。何も指定されていない場合は、HTTP Accept ヘッダーを使用して、json または yaml を受け入れることができるかどうかを検出します。両方が受け入れられるか、いずれも受け入れられるように設定されていない場合、json がデフォルト形式として返されます。

313.5. 例

Apache Camel ディストリビューションでは、この Swagger コンポーネントの使用方法を示す **camel-example-swagger-cdi** と **camel-example-swagger-java** を同梱しています。

第314章 SYSLOG DATAFORMAT

Camel バージョン 2.6 以降で利用可能

syslog データ形式は、RFC3164 および RFC5424 メッセージの操作に使用されます。

このコンポーネントは以下をサポートします。

- syslog メッセージの UDP 消費
- プレーンな String オブジェクトまたは SyslogMessage モデルオブジェクトのいずれかを使用する非依存的なデータ形式。
- SyslogMessage と文字列との間の型コンバーター
- camel-mina コンポーネントとの統合。
- camel-netty コンポーネントとの統合。
- **Camel 2.14:** camel-netty コンポーネントのエンコーダーとデコーダー。
- **Camel 2.14:** RFC5424 もサポート。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-syslog</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

314.1. RFC3164 SYSLOG プロトコル

Syslog は、基礎となるトランスポート層メカニズムとしてユーザーデータグラムプロトコル (UDP) 1 を使用します。syslog に割り当てられている UDP ポートは 514 です。

Syslog リスナーサービスを公開するには、既存の camel-mina コンポーネントまたは camel-netty を再利用します。ここでは、**Rfc3164SyslogDataFormat** を使用してメッセージをマーシャリングおよびアンマーシャリングします。**Camel 2.14** 以降、syslog データ形式の名前が **SyslogDataFormat** に変更されていることに注意してください。

314.2. オプション

Syslog データ形式は、以下にリストされている 1 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明

名前	デフォルト	Java タイプ	説明
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

314.3. RFC5424 SYSLOG プロトコル

Camel 2.14 から利用可能

Syslog リスナーサービスを公開するには、既存の camel-mina コンポーネントまたは camel-netty を再利用します。ここでは、**SyslogDataFormat** を使用してメッセージをマーシャリングおよびアンマーシャリングします。

314.3.1. Syslog リスナーの公開

Spring XML ファイルでは、ポート 10514 で udp メッセージをリッスンするようにエンドポイントを設定します。netty では defaultCodec を無効にしていることに注意してください。NettyTypeConverter へのフォールバックを許可し、メッセージを InputStream として配信します。

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

    <dataFormats>
        <syslog id="mySyslog"/>
    </dataFormats>

    <route>
        <from uri="netty:udp://localhost:10514?sync=false&amp;allowDefaultCodec=false"/>
        <unmarshal ref="mySyslog"/>
        <to uri="mock:stop1"/>
    </route>

</camelContext>
```

camel-mina を使った同じルート

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

    <dataFormats>
        <syslog id="mySyslog"/>
    </dataFormats>

    <route>
        <from uri="mina:udp://localhost:10514"/>
        <unmarshal ref="mySyslog"/>
        <to uri="mock:stop1"/>
    </route>

</camelContext>
```

314.3.2. リモート宛先への syslog メッセージの送信

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

  <dataFormats>
    <syslog id="mySyslog"/>
  </dataFormats>

  <route>
    <from uri="direct:syslogMessages"/>
    <marshal ref="mySyslog"/>
    <to uri="mina:udp://remotehost:10514"/>
  </route>

</camelContext>
```

314.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第315章 TAR ファイルのデータ形式

Camel バージョン 2.16 以降で利用可能

Tar ファイルデータ形式は、メッセージの圧縮および展開形式です。メッセージはエントリーを 1 つ含む Tar ファイルにマーシャリング (圧縮) でき、エントリーを 1つ含む Tar ファイルは元のファイルの内容にアンマーシャリング (展開) できます。

複数のメッセージを 1 つの Tar ファイルに集約できる集約ストラテジーもあります。

315.1. TAR ファイルのオプション

Tar ファイルのデータ形式は、以下に示す 4 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
usingIterator	**false**	**Boolean**	Tar ファイルに複数のエントリーがある場合には、このオプションを true に設定すると、スプリッター EIP を使用して、ストリーミングモードで反復子を使用してデータを分割できます。
allowEmptyDirectory	**false**	**Boolean**	Tar ファイルに複数のエントリーがある場合、このオプションを true に設定すると、ディレクトリーが空であっても反復子を取得できます。
preservePathElements	**false**	**Boolean**	ファイル名にパス要素が含まれている場合、このオプションを true に設定すると、Tar ファイルでパスを維持できます。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

315.2. MARSHAL

この例では、Tar ファイル圧縮を使用して通常のテキスト/XML ペイロードを圧縮ペイロードにマーシャリングし、それを MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().tarFile().to("activemq:queue:MY_QUEUE");
```

作成された Tar ファイル内の Tar エントリーの名前は、受信した **CamelFileName** メッセージヘッダーに基づいています。これは、ファイルコンポーネントによって使用される標準のメッセージヘッダーです。さらに、送信 **CamelFileName** メッセージヘッダーは、受信 **CamelFileName** メッセージヘッダーの値に ".tar" 接尾辞を付けて自動的に設定されます。たとえば、次のルートで入力ディレクトリーに test.txt という名前のファイルが見つかった場合には、出力は test.txt という名前の単一の Tar エントリーを含む test.txt.tar という名前の Tar ファイルになります。

```
from("file:input/directory?antInclude=*/.txt").marshal().tarFile().to("file:output/directory");
```

着信 **CamelFileName** メッセージヘッダーがない場合 (ファイルコンポーネントがコンシューマーでな

い場合など)、メッセージ ID がデフォルトで使用されます。メッセージ ID は通常、一意に生成された ID であるため、**ID-MACHINENAME-2443-1211718892437-1-0.tar** のようなファイル名になります。この動作をオーバーライドする場合は、ルートで **CamelFileName** ヘッダーの値を明示的に設定できます。

```
from("direct:start").setHeader(Exchange.FILE_NAME,
constant("report.txt")).marshal().tarFile().to("file:output/directory");
```

このルートにより、出力ディレクトリーに report.txt.tar という名前の Tar ファイルが作成され、report.txt という名前の単一の Tar エントリーが含まれます。

315.3. UNMARSHAL

この例では、MY_QUEUE という ActiveMQ キューから Tar ファイルペイロードを元の形式にアンマーシャリングして **UnTarpedMessageProcessor** に転送し、処理します。

```
from("activemq:queue:MY_QUEUE").unmarshal().tarFile().process(new
UnTarpedMessageProcessor());
```

Tar ファイルに複数のエントリーがある場合には、TarFileDataFormat の usingIterator オプションを true にすると、スプリッターを使用してさらに作業を行うことができます。

```
  TarFileDataFormat tarFile = new TarFileDataFormat();
  tarFile.setUsingIterator(true);
  from("file:src/test/resources/org/apache/camel/dataformat/tarfile/?
consumer.delay=1000&noop=true")
    .unmarshal(tarFile)
    .split(body(Iterator.class))
        .streaming()
         .process(new UnTarpedMessageProcessor())
    .end();
```

または、このように TarSplitter をスプリッターの式として直接使用できます

```
   from("file:src/test/resources/org/apache/camel/dataformat/tarfile?
consumer.delay=1000&noop=true")
     .split(new TarSplitter())
       .streaming()
       .process(new UnTarpedMessageProcessor())
     .end();
```

315.4. AGGREGATE

情報: この集約ストラテジーを適切に機能させるには、完了の先行チェックが必要であることに注意してください。

この例では、入力ディレクトリーで見つかったすべてのテキストファイルを、出力ディレクトリーに格納される 1 つの Tar ファイルに集約します。

```
   from("file:input/directory?antInclude=*/.txt")
     .aggregate(new TarAggregationStrategy())
       .constant(true)
```

```
        .completionFromBatchConsumer()
        .eagerCheckCompletion()
    .to("file:output/directory");
```

発信 **CamelFileName** メッセージヘッダーは、java.io.File.createTempFile を使用して作成され、".tar" 接尾辞が付きます。 この動作をオーバーライドする場合は、ルートで **CamelFileName** ヘッダーの値を明示的に設定できます。

```
  from("file:input/directory?antInclude=*/.txt")
    .aggregate(new TarAggregationStrategy())
      .constant(true)
      .completionFromBatchConsumer()
      .eagerCheckCompletion()
    .setHeader(Exchange.FILE_NAME, constant("reports.tar"))
  .to("file:output/directory");
```

315.5. 依存関係

camel ルートで Tar ファイルを使用するには、このデータ形式を実装する **camel-tarfile** に依存関係を追加する必要があります。

Maven を使用する場合は、**pom.xml** に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

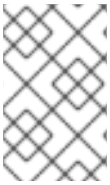
```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tarfile</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第316章 TELEGRAM コンポーネント

Camel バージョン 2.18 以降で利用可能

Telegram コンポーネントは、Telegram Bot API へのアクセスを提供します。これにより、Camel ベースのアプリケーションは、ボットとして機能し、通常のユーザー、プライベートおよびパブリックグループまたはチャネルとの直接の会話に参加して、メッセージを送受信できます。

このコンポーネントを使用する前に、Telegram Bot 開発者 ホームの指示に従って、Telegram Bot を作成する必要があります。新しいボットが作成されると、BotFather は ボットに対応する **認証トークン** を提供します。認証トークンは、camel-telegram エンドポイントの必須パラメーターです。



注記

Bot がグループまたはチャネル内で交換されるすべてのメッセージ (/文字で始まるメッセージだけでなく) を受信できるようにするには、**/setprivacy** コマンドを使用して、BotFather に **プライバシーモードを無効にするよう** に依頼します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-telegram</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

316.1. URI 形式

```
telegram:type/authorizationToken[?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

316.2. オプション

Telegram コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
authorizationToken (security)	エンドポイントで情報が提供されない場合に使用されるデフォルトの Telegram 認証トークン。		String
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Telegram エンドポイントは、URI 構文を使用して設定されます。

```
telegram:type/authorizationToken
```

パスおよびクエリーパラメーターを使用します。

316.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
type	**必須** エンドポイントのタイプ。現在、ボットタイプのみがサポートされています。		String
authorizationToken	**必須** ボットを使用するための認証トークン (BotFather に問い合わせてください)。たとえば、654321531:HGF_dTra456323dHuOedsE343211fqr3t-H.		String

316.2.2. クエリーパラメーター(22 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
limit (consumer)	1 回のポーリングリクエストで受信できる更新の数を制限します。	100	Integer
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
timeout (consumer)	ロングポーリングのタイムアウト (秒単位)。ショートポーリングの場合は 0 を、ロングポーリングの場合はより大きな数を入力します。ポーリングが長いと、応答時間が短くなります。	30	Integer

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
chatId (producer)	生成されたメッセージを受け取るチャットの識別子。チャット ID は、最初に着信メッセージから取得できます (たとえば、テレグラムユーザーがボットとの会話を開始すると、そのクライアントはチャット ID を含む/start メッセージを自動的に送信します)。チャット ID は送信メッセージごとに (本文またはヘッダーを使用して) 動的に設定できるため、オプションのパラメーターです。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long

名前	説明	デフォルト	タイプ
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

316.3. メッセージヘッダー

名前	説明

名前	説明
CamelTelegramChatId	このヘッダーは、メッセージを受信するチャット ID を解決するためにプロデューサーエンドポイントによって使用されます。受信者のチャット ID は、メッセージ本文、**CamelTelegramChatId** ヘッダー、またはエンドポイント設定 (**chatId** オプション) に (優先度順に) 配置できます。このヘッダーは、すべての受信メッセージにも存在します。
CamelTelegramMediaType	このヘッダーは、送信メッセージが純粋なバイナリーデータで設定されている場合にメディアタイプを識別するために使用されます。可能な値は、**org.apache.camel.component.telegram.TelegramMediaType** 列挙に属する文字列または列挙値です。
CamelTelegramMediaTitleCaption	このヘッダーは、発信バイナリーメッセージのキャプションまたはタイトルを提供するために使用されます。
CamelTelegramParseMode	このヘッダーは、HTML または Markdown を使用してテキストメッセージをフォーマットするために使用されます (**org.apache.camel.component.telegram.TelegramParseMode** を参照)。

316.4. 用途

Telegram コンポーネントは、コンシューマーエンドポイントとプロデューサーエンドポイントの両方をサポートします。また、**リアクティブチャットボットモード** で使用することもできます (メッセージを消費してから生成するため)。

316.5. プロデューサーの例

以下は、Telegram Bot API を介して Telegram チャットにメッセージを送信する方法の基本的な例です。

Java DSL で

```
from("direct:start").to("telegram:bots/123456789:insertAuthorizationTokenHere");
```

または Spring XML で

```
<route>
  <from uri="direct:start"/>
  <to uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
<route>
```

コード **123456789:insertAuthorizationTokenHere** ボットに対応する **認証トークン** です。

チャット ID オプションを指定せずにプロデューサーエンドポイントを使用すると、メッセージの本文またはヘッダーに含まれる情報を使用してターゲットチャットが識別されます。次のメッセージ本文はプロデューサーエンドポイントで許可されます (タイプ **OutgoingXXXMessage** のメッセージはパッケージ **org.apache.camel.component.telegram.model** に属します)

Java タイプ	説明
OutgoingTextMessage	チャットにテキストメッセージを送信するには
OutgoingPhotoMessage	写真 (JPG、PNG) をチャットに送信するには
OutgoingAudioMessage	mp3 オーディオをチャットに送信するには
OutgoingVideoMessage	mp4 ビデオをチャットに送信するには
OutgoingDocumentMessage	チャットにファイルを送信するには (任意のメディアタイプ)
byte[]	サポートされている任意のメディアタイプを送信します。**CamelTelegramMediaType** ヘッダーを適切なメディアタイプに設定する必要があります。
String	チャットにテキストメッセージを送信します。自動的に **OutgoingTextMessage** に変換されます

316.6. コンシューマーの例

以下は、テレグラムユーザーが設定済みボットに送信するすべてのメッセージを受信する方法の基本的な例です。Java DSL で

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.bean(ProcessorBean.class)
```

または Spring XML で

```
<route>
    <from uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
    <bean ref="myBean" />
<route>

<bean id="myBean" class="com.example.MyBean"/>
```

MyBean は、メッセージを受け取る単純な Bean です。

```
public class MyBean {

    public void process(String message) {
        // or Exchange, or org.apache.camel.component.telegram.model.IncomingMessage (or both)

        // do process
    }

}
```

着信メッセージでサポートされているタイプは次のとおりです。

Java タイプ	説明
IncomingMessage	着信メッセージの完全なオブジェクト表現
String	メッセージの内容 (テキストメッセージのみ)

316.7. リアクティブなチャットボットの例

リアクティブチャットボットモードは、Camel コンポーネントを使用して、Telegram ユーザーから受信したチャットメッセージに直接返信する単純なチャットボットを構築する簡単な方法です。

以下は、Java DSL でのチャットボットの基本設定です。

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.bean(ChatBotLogic.class)
.to("telegram:bots/123456789:insertAuthorizationTokenHere");
```

または Spring XML で

```
<route>
   <from uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
   <bean ref="chatBotLogic" />
   <to uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
<route>

<bean id="chatBotLogic" class="com.example.ChatBotLogic"/>
```

ChatBotLogic は、汎用の String-to-String メソッドを実装する単純な Bean です。

```
public class ChatBotLogic {

   public String chatBotProcess(String message) {
      if( "do-not-reply".equals(message) ) {
         return null; // no response in the chat
      }

      return "echo from the bot: " + message; // echoes the message
   }

}
```

chatBotProcess メソッドによって返されるすべての非 null 文字列は、リクエストを発信したチャットに自動的にルーティングされます (メッセージのルーティングには **CamelTelegramChatId** ヘッダーが使用されるため)。

316.8. チャット ID の取得

イベントが発生したときに特定のテレグラムチャットにメッセージをプッシュする場合は、対応するチャット ID を取得する必要があります。チャット ID は現在テレグラムクライアントに表示されていませんが、簡単なルートを使用して取得できます。

まず、メッセージをプッシュするチャットにボットを追加してから、次のようなルートを実行します。

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.to("log:INFO?showHeaders=true");
```

ボットが受信したメッセージは、チャットに関する情報 (**CamelTelegramChatId** ヘッダー) と共にログにダンプされます。

チャット ID を取得したら、次のサンプルルートを使用してメッセージをプッシュできます。

```
from("timer:tick")
.setBody().constant("Hello")
to("telegram:bots/123456789:insertAuthorizationTokenHere?chatId=123456")
```

対応する URI パラメーターは単純に **chatId** であることに注意してください。

第317章 TEST コンポーネント

Camel バージョン 1.3 以降で利用可能

分散処理と非同期処理のテストは、非常に難しいことで知られています。Mock、Test、および DataSet エンドポイントは Camel テストフレームワークとうまく連携し、エンタープライズ統合パターン と Camel の幅広いコンポーネントを強力な Bean 統合と共に使用して、ユニットと統合のテストを簡素化します。

test コンポーネントは、Mock コンポーネントを拡張して、起動時に別のエンドポイントからメッセージをプルし、基になる Mock エンドポイントに予期されるメッセージボディーを設定することをサポートします。つまり、ルートでテストエンドポイントを使用すると、そこに到着するメッセージが、他の場所から抽出された予期されるメッセージと暗黙的に比較されます。

したがって、たとえば、想定されるメッセージボディーのセットをファイルとして使用できます。これにより、適切に設定された Mock エンドポイントが設定されます。これは、受信したメッセージが予想されるメッセージの数と一致し、メッセージペイロードが等しい場合にのみ有効です。

Maven ユーザーは、**Camel 2.8** 以前を使用する場合、このコンポーネントの **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel 2.9 以降、Test コンポーネントは camel-core で直接提供されます。

317.1. URI 形式

```
test:expectedMessagesEndpointUri
```

expectedMessagesEndpointUri は、テストを開始する前に予想されるメッセージボディーがプルされる他のコンポーネント URI を参照します。

317.2. URI オプション

Test コンポーネントにはオプションがありません。

Test エンドポイントは、URI 構文を使用して設定されます。

```
test:name
```

パスおよびクエリーパラメーターを使用します。

317.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** テストに使用されるメッセージのポーリングに使用するためにレジストリーで検索するエンドポイントの名前。		String

317.2.2. クエリーパラメーター (14 パラメーター)

名前	説明	デフォルト	タイプ
anyOrder (producer)	想定されるメッセージが同じ順序で到着するか、任意の順序で到着できるか。	false	boolean
assertPeriod (producer)	暫定的なアサーションがまだ有効であることを確認するために、モックエンドポイントが再アサートするまでの猶予期間を設定します。これは、たとえば、正確な数のメッセージが到着したことをアサートするために使用されます。たとえば、リンク expectedMessageCount (int) が 5 に設定されている場合、5 つ以上のメッセージが到着するとアサーションが満たされます。正確に 5 つのメッセージが到着するようにするには、それ以上メッセージが到着しないように少し待つ必要があります。これが、この setAssertPeriod (long) メソッドを使用できる目的です。デフォルトでは、この期間は無効になっています。	0	long
delimiter (producer)	分割が有効な場合に使用する分割区切り文字。デフォルトでは、区切り文字は改行ベースです。デリミターには正規表現を使用できます。		String
expectedCount (producer)	このエンドポイントが受信するメッセージ交換の予想数を指定します。注意: 0 のメッセージを期待したい場合は、特別な注意が必要です。0 はテストの開始時に一致するため、アサート期間を設定して、テストをしばらく実行し、まだメッセージが到着していないことを確認する必要があります。;そのためにはリンク setAssertPeriod (long) を使用します。別の方法として、NotifyBuilder を使用し、モックで assertIsSatisfied () メソッドを呼び出す前に、NotifyBuilder を使用して、Camel がいくつかのメッセージのルーティングを完了したことを知ることができます。これにより、固定アサート期間を使用せずにテスト時間を短縮できます。正確に n 番目のメッセージがこのモックエンドポイントに到着することをアサートする場合は、詳細はリンク setAssertPeriod (long) メソッドも参照してください。	-1	int

名前	説明	デフォルト	タイプ
reportGroup (producer)	サイズのグループに基づいてスループットログを有効にするために使用される数値。		int
resultMinimumWaitTime (producer)	ラッチが満たされるまでリンク assertIsSatisfied () が待機する最小予想時間 (ミリ秒単位) を設定します	0	long
resultWaitTime (producer)	ラッチが満たされるまでリンク assertIsSatisfied () が待機する最大時間 (ミリ秒単位) を設定します	0	long
retainFirst (producer)	受信した Exchange の最初の n 番目の数だけを保持するように指定します。これは、ビッグデータでテストするときに使用され、このモックエンドポイントが受信するすべての Exchange のコピーを保存しないことでメモリー消費を削減します。重要: この制限を使用する場合、リンク getReceivedCounter() は受信した Exchange の実際の数を返します。たとえば、5000 の交換を受信し、最初の 10 の交換のみを保持するように設定した場合、リンク getReceivedCounter() は引き続き 5000 を返しますが、リンク getExchanges() およびリンク getReceivedExchanges() メソッドには最初の 10 の交換しかありません。このメソッドを使用する場合、他の期待値メソッドの一部はサポートされません。たとえば、リンク expectedBodiesReceived(Object...) は、受信した最初の数のボディに期待値を設定します。リンク setRetainFirst(int) メソッドとリンク setRetainLast(int) メソッドの両方を設定して、最初と最後の受信の両方を制限できます。	-1	int
retainLast (producer)	受信した Exchange の最後の n 番目の数だけを保持するように指定します。これは、ビッグデータでテストするときに使用され、このモックエンドポイントが受信するすべての Exchange のコピーを保存しないことでメモリー消費を削減します。重要: この制限を使用する場合、リンク getReceivedCounter() は受信した Exchange の実際の数を返します。たとえば、5000 の交換を受信し、最後の 20 の交換のみを保持するように設定した場合、リンク getReceivedCounter() は引き続き 5000 を返しますが、リンク getExchanges() およびリンク getReceivedExchanges() メソッドには最後の 20 の交換しかありません。このメソッドを使用する場合、他の期待値メソッドの一部はサポートされません。たとえば、リンク expectedBodiesReceived(Object...) は、受信した最初の数のボディに期待値を設定します。リンク setRetainFirst(int) メソッドとリンク setRetainLast(int) メソッドの両方を設定して、最初と最後の受信の両方を制限できます。	-1	int

名前	説明	デフォルト	タイプ
sleepForEmptyTest (producer)	リンク expectedMessageCount (int) がゼロで呼び出されたときに、このエンドポイントが実際に空であることを確認するために待機するスリープを指定できるようにします	0	long
split (producer)	有効にすると、テストエンドポイントからロードされたメッセージは改行区切り文字を使用して分割されるため、各行は想定されるメッセージになります。たとえば、ファイルエンドポイントを使用して、各行が想定されるメッセージであるファイルをロードします。	false	boolean
timeout (producer)	URI からメッセージボディーをポーリングするときに使用するタイムアウト。	2000	long
copyOnExchange (producer)	このモックエンドポイントで受信したときに受信 Exchange のディープコピーを作成するかどうかを設定します。デフォルトでは true です。	true	boolean
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

317.3. 例

たとえば、次のようにテストケースを記述できます。

```
from("seda:someEndpoint").
  to("test:file://data/expectedOutput?noop=true");
```

その後、テストが MockEndpoint.assertIsSatisfied (camelContext) メソッド を呼び出すと、テストケースは必要なアサーションを実行します。

テストエンドポイントで他の期待値を設定する方法については、Mock コンポーネントを参照してください。

317.4. 関連項目

- Spring のテスト

第318章 THRIFT コンポーネント

Camel バージョン 2.20 以降で利用可能

Thrift コンポーネントを使用すると、Apache Thrift バイナリー通信プロトコルとシリアル化メカニズムを使用して、リモートプロシージャーコール (RPC) サービスを呼び出したり、公開したりできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-thrift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

318.1. URI 形式

```
thrift://service[?options]
```

318.2. エンドポイントオプション

Thrift コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
useGlobalSslContext Parameters (security)	Thrift コンポーネントがグローバル SSL コンテキストパラメーターを使用しているかどうかを判断する	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Thrift エンドポイントは、URI 構文を使用して設定されます。

```
thrift:host:port/service
```

パスおよびクエリーパラメーターを使用します。

318.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
host	Thrift サーバーのホスト名。コンシューマーの場合は localhost または 0.0.0.0 (定義されていない場合)、プロデューサーを使用する場合はリモートサーバーのホスト名です。		String
port	**必須** Thrift サーバーポート。		int
service	**必須** thrift 記述子ファイルからの完全修飾サービス名 (パッケージドットサービス定義名)。		String

318.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
compressionType (Common)	プロトコル圧縮メカニズムのタイプ。	NONE	ThriftCompressionType
exchangeProtocol (Common)	Exchange プロトコルのシリアル化の種類。	BINARY	ThriftExchangeProtocol
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clientTimeout (consumer)	コンシューマーのクライアントタイムアウト。		int
maxPoolSize (consumer)	Thrift サーバーコンシューマーの最大スレッドプールサイズ。	10	int
poolSize (consumer)	Thrift サーバーコンシューマーの初期スレッドプールサイズ。	1	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
method (producer)	Thrift が呼び出すメソッド名。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
negotiationType (security)	セキュリティーネゴシエーションタイプ。	PLAIN TEXT	ThriftNegotiationType
sslParameters (security)	SSL/TLS セキュリティーネゴシエーションの設定パラメーター。		SSLContextParameters

318.3. THRIFT メソッドのパラメーターマッピング

呼び出されたプロシージャのパラメーターは、メッセージボディー内のオブジェクトのリストとして渡す必要があります。プリミティブは、オンザフライでオブジェクトから変換されます。対応するメソッドを正しく見つけるには、値に関係なくすべての型を送信する必要があります。以下の例を参照してください。Camel 本文を使用してメソッドにさまざまなパラメーターを渡す方法

```
List requestBody = new ArrayList();

requestBody.add((boolean)true);
requestBody.add((byte)THRIFT_TEST_NUM1);
requestBody.add((short)THRIFT_TEST_NUM1);
requestBody.add((int)THRIFT_TEST_NUM1);
requestBody.add((long)THRIFT_TEST_NUM1);
requestBody.add((double)THRIFT_TEST_NUM1);
requestBody.add("empty"); // String parameter
requestBody.add(ByteBuffer.allocate(10)); // binary parameter
requestBody.add(new Work(THRIFT_TEST_NUM1, THRIFT_TEST_NUM2, Operation.MULTIPLY));
// Struct parameter
requestBody.add(new ArrayList<Integer>()); // list paramater
requestBody.add(new HashSet<String>()); // set parameter
requestBody.add(new HashMap<String, Long>()); // map parameter

Object responseBody = template.requestBody("direct:thrift-alltypes", requestBody);
```

サービスコンシューマーの着信パラメーターも、オブジェクトのリストとしてメッセージボディーに渡されます。

318.4. THRIFT コンシューマーヘッダー (コンシューマーの呼び出し後にインストールされます)

ヘッダー名	説明	使用できる値
CamelThriftMethodName	コンシューマーサービスが扱うメソッド名	

318.5. 例

以下は、ホストとポートのパラメーターを使用した単純な同期メソッドの呼び出しです。

```
from("direct:thrift-calculate")
.to("thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator?
method=calculate&synchronous=true");
```

以下は、XML DSL 設定の単純な同期メソッド呼び出しです。

```
<route>
    <from uri="direct:thrift-add" />
    <to uri="thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator?
method=add&synchronous=true"/>
</route>
```

非同期通信による Thrift サービスコンシューマー

```
from("thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator")
.to("direct:thrift-service");
```

thrift-maven-plugin を使用して .thrift ファイルの Java コード生成を自動化することは可能ですが、開始する前に、オペレーティングシステムの thrift コンパイラーバイナリーディストリビューションが実行中のホストに存在している必要があります。

318.6. 詳細については、これらのリソースを参照してください

Thrift プロジェクト GitHub https://thrift.apache.org/tutorial/java Apache Thrift Java チュートリアル

318.7. 関連項目

- スタートガイド
- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

第319章 THRIFT DATAFORMAT

Camel バージョン 2.20 以降で利用可能

Camel は、Java と Apache Thrift の間でシリアル化するためのデータ形式を提供します。プロジェクトのサイトには、https://thrift.apache.org/ を使用する理由が詳しく説明されています。Apache Thrift は言語とプラットフォームに中立であるため、Camel ルートによって生成されたメッセージは、他の言語の実装によって消費される可能性があります。

Apache Thrift の実装

319.1. THRIFT オプション

Thrift データフォーマットは、以下にリストされている 3 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
instanceClass		**String**	アンアームシャリング時に使用するクラスの名前
contentTypeFormat	**binary**	**String**	thrift メッセージが Java から (to) シリアライズ/デシリアライズされるコンテンツタイプ形式を定義します。形式は、ネイティブバイナリー thrift、json、または単純な json フィールド表現のネイティブまたは json のいずれかです。デフォルト値はバイナリーです。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

319.2. コンテンツタイプの形式

JSON メッセージを解析して Thrift 形式に変換し、ネイティブの util コンバーターを使用して元に戻すことができます。このオプションを使用するには、contentTypeFormat 値を json に設定するか、2 番目のパラメーターで thrift を呼び出します。デフォルトのインスタンスが指定されていない場合は、常にネイティブバイナリーの Thrift 形式を使用します。単純な JSON 形式は書き込み専用 (マーシャル) であり、スクリプト言語による解析に適した単純な出力形式を生成します。サンプルコードを以下に示します。

```
from("direct:marshal")
    .unmarshal()
    .thrift("org.apache.camel.dataformat.thrift.generated.Work", "json")
    .to("mock:reverse");
```

319.3. THRIFT の概要

Thrift の使用方法の簡単な概要です。詳細については、完全なチュートリアル を参照してください

319.4. THRIFT フォーマットの定義

最初のステップは、エクスチェンジのボディーの形式を定義することです。これは .thrift ファイルで次のように定義されています。

tutorial.thrift

```
namespace java org.apache.camel.dataformat.thrift.generated

enum Operation {
  ADD = 1,
  SUBTRACT = 2,
  MULTIPLY = 3,
  DIVIDE = 4
}

struct Work {
  1: i32 num1 = 0,
  2: i32 num2,
  3: Operation op,
  4: optional string comment,
}
```

319.5. JAVA クラスの生成

Apache Thrift は、.thrift ファイルで定義した形式の Java クラスを生成するコンパイラーを提供します。

手動で必要な追加のサポート対象言語に対してコンパイラーを実行することもできます。

thrift -r --gen java -out ../java/ ./tutorial-dataformat.thrift

これにより、.thrift ファイルで定義された型ごとに個別の Java クラスが生成されます。つまり、構造体または列挙型です。生成されたクラスは、直列化メカニズムに必要な org.apache.thrift.TBase を実装します。このため、これらのクラスのみがエクスチェンジのボディーで使用されることが重要です。org.apache.thrift.TBase を実装していないクラスを使用するようデータ形式に指示しようとすると、Camel はルート作成時に例外を出力します。

319.6. JAVA DSL

このように、ThriftDataFormat インスタンスを作成し、それを Camel DataFormat マーシャリングおよびアンマーシャリング API に渡すことができます。

```
ThriftDataFormat format = new ThriftDataFormat(new Work());

from("direct:in").marshal(format);
from("direct:back").unmarshal(format).to("mock:reverse");
```

または、非整列化デフォルトインスタンスまたはデフォルトインスタンスクラス名を次のように渡す DSL thrift() を使用します。

```
// You don't need to specify the default instance for the thrift marshaling
from("direct:marshal").marshal().thrift();
from("direct:unmarshalA").unmarshal()
```

```
        .thrift("org.apache.camel.dataformat.thrift.generated.Work")
        .to("mock:reverse");

    from("direct:unmarshalB").unmarshal().thrift(new Work()).to("mock:reverse");
```

319.7. SPRING DSL

次の例は、Thrift データ型を設定する Spring を使用して非整列化するために Thrift を使用する方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
      <thrift instanceClass="org.apache.camel.dataformat.thrift.generated.Work" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

319.8. 依存関係

camel ルートで Thrift を使用するには、このデータ形式を実装する **camel-thrift** に依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-thrift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第320章 TIDYMARKUP DATAFORMAT

Camel バージョン 2.0 以降で利用可能

TidyMarkup は、TagSoup を使用して HTML を整理するデータ形式です。醜い HTML を解析し、整形式の HTML として返すために使用できます。

正式リリース前の SOAP をテストに使用

PDF マニュアルに奇妙な記号が含まれている問題がありました。そのため、Jonathan はこのデータ形式を使用して、pdf マニュアルをレンダリングするためのベースとして使用される wiki html ページを整形しました。そして奇妙な記号を消すことができました。

TidyMarkup は **非整列化** 操作のみをサポートします。これは、整形式の HTML を醜い HTML に変換したくないためです。

320.1. TIDYMARKUP OPTIONS

TidyMarkup データ形式は、以下に示す 3 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
dataObjectType	**org.w3c.dom.Node**	**String**	非整列化するデータ型は、org.w3c.dom.Node または java.lang.String のいずれかです。デフォルトでは org.w3c.dom.Node です
omitXmlDeclaration	**false**	**Boolean**	文字列を返す場合、先頭の XML 宣言を省略しますか。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

320.2. JAVA DSL の例

コンシューマーが HTML を提供する例

```
from("file://site/inbox").unmarshal().tidyMarkup().to("file://site/blogs");
```

320.3. SPRING XML の例

次の例は、TidyMarkup を使用して Spring を使用して非整列化する方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file://site/inbox"/>
    <unmarshal>
```

```
      <tidyMarkup/>
    </unmarshal>
    <to uri="file://site/blogs"/>
  </route>
</camelContext>
```

320.4. 依存関係

camel ルートで TidyMarkup を使用するには、このデータ形式を実装する **camel-tagsoup** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tagsoup</artifactId>
  <version>x.x.x</version>
</dependency>
```

第321章 TIKA コンポーネント

Camel バージョン 2.19 以降で利用可能

Tika: コンポーネントは、Apache Tika を使用してドキュメントを検出および解析する機能を提供します。このコンポーネントは、ドキュメントを操作するための基礎となるライブラリーとして Apache Tika を使用します。

Tika コンポーネントを使用するには、Maven ユーザーは次の依存関係を **pom.xml** に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tika</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

TIKA コンポーネントはプロデューサーエンドポイントのみをサポートします。

321.1. オプション

Tika コンポーネントにはオプションがありません。

Tika エンドポイントは、URI 構文を使用して設定されます。

```
tika:operation
```

パスおよびクエリーパラメーターを使用します。

321.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	**必須** Tika 演算。解析または検出。		TikaOperation

321.1.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
tikaConfig (producer)	Tika Config。		TikaConfig
tikaConfigUri (producer)	Tika Config の URI: tika-config.xml の URI。		String

名前	説明	デフォルト	タイプ
tikaParseOutputEncoding (producer)	Tika Parse Output Encoding - 解析された出力の文字エンコーディングを指定するために使用されます。Defaults to Charset.defaultCharset() .		String
tikaParseOutputFormat (producer)	Tika 出力フォーマット。サポートされている出力形式。xml: 解析されたコンテンツを XML として返します。html: 解析されたコンテンツを HTML として返します。text: 解析されたコンテンツをテキストとして返します。textMain: ボイラーパイプライブラリーを使用して、Web ページからメインコンテンツを自動的に抽出します。	xml	TikaParseOutputFormat
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

321.2. ファイルの MIME タイプを検出するには

ファイルは Body に配置する必要があります。

```
from("direct:start")
        .to("tika:detect");
```

321.3. ファイルを解析するには

ファイルは Body に配置する必要があります。

```
from("direct:start")
        .to("tika:parse");
```

第322章 TIMER コンポーネント

Camel バージョン 1.0 以降で利用可能

timer: コンポーネントは、タイマーが起動したときにメッセージエクスチェンジを生成するために使用されます。このエンドポイントからのイベントのみを使用できます。

322.1. URI 形式

```
timer:name[?options]
```

name は、エンドポイント間で作成および共有される **Timer** オブジェクトの名前です。したがって、すべてのタイマーエンドポイントに同じ名前を使用すると、**Timer** オブジェクトとスレッドは 1 つだけ使用されます。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

注記: 生成されたエクスチェンジの IN ボディーは **null** です。したがって、**exchange.getIn().getBody()** は **null** を返します。

ヒント:*高度なスケジューラー* 詳細にわたるスケジューリングをサポートする Quartz コンポーネントも参照してください。

ヒント:*人間にわかりやすい形式で時間を指定する* **Camel 2.3** 以降では、人間にわかりやすい構文 で時間を指定できます。

322.2. オプション

Timer コンポーネントにはオプションがありません。

Timer エンドポイントは、URI 構文を使用して設定されます。

```
timer:timerName
```

パスおよびクエリーパラメーターを使用します。

322.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
timerName	**必須** タイマーの名前		String

322.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	最初のイベントが生成されるまで待機するミリ秒数。time オプションと一緒に使用しないでください。デフォルト値は 1000 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
fixedRate (consumer)	イベントは、指定された期間で区切られたほぼ一定の間隔で発生します。	false	boolean
period (consumer)	0 より大きい場合は、期間ミリ秒ごとに定期的なイベントを生成します。デフォルト値は 1000 です。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
repeatCount (consumer)	実行の最大数を指定します。したがって、1 に設定すると、タイマーは 1 回だけ起動します。これを 5 に設定した場合、5 回だけ実行されます。0 または負の値を設定すると、無制限に実行されます。	0	long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
daemon (advanced)	タイマーエンドポイントに関連付けられたスレッドがデーモンとして実行されるかどうかを指定します。デフォルト値は true です。	true	boolean
pattern (advanced)	URI 構文を使用して時間オプションを設定するために使用するカスタムの日付パターンを指定できます。		String

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
time (advanced)	最初のイベントが生成される java.util.Date。URI を使用する場合、予想されるパターンは次のとおりです: yyyy-MM-dd HH:mm:ss または yyyy-MM-dd'T'HH:mm:ss。		日付
timer (advanced)	カスタムタイマーを使用するには		Timer

322.3. エクスチェンジプロパティー

タイマーが起動すると、次の情報がプロパティーとして **Exchange** に追加されます。

名前	タイプ	説明
Exchange.TIMER_NAME	**String**	**name** オプションの値。
Exchange.TIMER_TIME	**日付**	**time** オプションの値。
Exchange.TIMER_PERIOD	**long**	**period** オプションの値。
Exchange.TIMER_FIRED_TIME	**日付**	コンシューマーが起動した時刻。
Exchange.TIMER_COUNTER	**Long**	Camel 2.8: 現在の射撃カウンター。1 から始まります。

322.4. 例

60 秒ごとにイベントを生成するルートを設定するには:

```
from("timer://foo?fixedRate=true&period=60000").to("bean:myBean?method=someMethodName");
```

ヒント

60000 の代わりに、より読みやすい period=60s を使用できます。

上記のルートはイベントを生成し、JNDI や Spring などのレジストリーで **myBean** と呼ばれる Bean で **someMethodName** メソッドを呼び出します。

そして、Spring DSL の経路:

```
<route>
  <from uri="timer://foo?fixedRate=true&amp;period=60000"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

322.5. できるだけ早く起動

Camel 2.17 以降で利用可能

Camel ルートでできるだけ早くメッセージを送信したい場合は、負の遅延を使用できます。

```
<route>
  <from uri="timer://foo?delay=-1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

このようにして、タイマーはすぐにメッセージを送信します。

また、repeatCount パラメーターを負の遅延と組み合わせて指定し、固定数に達した後にメッセージの送信を停止することもできます。

repeatCount を指定しない場合、タイマーはルートが停止するまでメッセージを発し続けます。

322.6. 起動は 1 回のみ

Camel 2.8 から利用可能

ルートの開始時など、Camel ルートでメッセージを 1 回だけ起動したい場合があります。これを行うには、次のように repeatCount オプションを使用します。

```
<route>
  <from uri="timer://foo?repeatCount=1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

322.7. 関連項目

- スケジューラー

- Quartz

第323章 TWILIO コンポーネント

Camel バージョン 2.20 以降で利用可能

Twilio コンポーネントは、Twilio Java SDK を使用してアクセス可能な Twilio REST API のバージョン 2010-04-01 へのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twilio</artifactId>
  <version>${camel-version}</version>
</dependency>
```

323.1. TWILIO オプション

Twilio コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有設定を使用する場合。		TwilioConfiguration
restClient (advanced)	共有 REST クライアントを使用する場合。		TwilioRestClient
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Twilio エンドポイントは、URI 構文を使用して設定されます。

```
twilio:apiName/methodName
```

パスおよびクエリーパラメーターを使用します。

323.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
apiName	**必須** 操作の種類		TwilioApiName
methodName	**必須**: 選択した操作に使用するサブ操作		文字列

323.1.2. クエリーパラメーター (8 つのパラメーター):

名前	説明	デフォルト	タイプ
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
accountSid (security)	使用するアカウント SID。		文字列
password (security)	アカウントの認証トークン。		文字列
username (security)	使用するアカウント。		String

323.2. URI 形式

```
twilio://endpoint-prefix/endpoint?[options]
```

エンドポイント 接頭辞は次のいずれかです。

- アカウント
- address
- address-dependent-phone-number

- application
- available-phone-number-country
- available-phone-number-country-local
- available-phone-number-country-mobile
- available-phone-number-country-toll-free
- call
- call-feedback
- call-feedback-summary
- call-notification
- call-recording
- conference
- conference-participant
- connect-app
- incoming-phone-number
- incoming-phone-number-local
- incoming-phone-number-mobile
- incoming-phone-number-toll-free
- key
- message
- message-feedback
- message-media
- new-key
- new-signing-key
- notification
- outgoing-caller-id
- queue
- queue-member
- recording
- recording-add-on-result

- recording-add-on-result-payload
- recording-transcription
- short-code
- signing-key
- sip-credential-list
- sip-credential-list-credential
- sip-domain
- sip-domain-credential-list-mapping
- sip-domain-ip-access-control-list-mapping
- sip-ip-access-control-list
- sip-ip-access-control-list-ip-address
- token
- transcription
- usage-record
- usage-record-all-time
- usage-record-daily
- usage-record-last-month
- usage-record-monthly
- usage-record-this-month
- usage-record-today
- usage-record-yearly
- usage-record-yesterday
- usage-trigger
- validation-request

323.3. プロデューサーエンドポイント:

プロデューサーエンドポイントは、エンドポイント 接頭辞の後にエンドポイント名と次に説明する関連オプションを使用できます。すべてのエンドポイントに省略形のエイリアスを使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は **CamelTwilio.<option>** の形式である必要があります。**inBody** オプションはメッセージヘッダーをオーバーライドすることに注意してください。つまり、エンドポイントオプショ

ン **inBody=option** は**CamelTwilio.option** ヘッダーをオーバーライドします。

エンドポイントは次のいずれかです。

エンドポイント	短縮形エイリアス	説明
creator	create	作成を実行するために Twilio API にリクエストを行います。
deleter	delete	削除を実行するために Twilio API にリクエストを行います。
fetcher	fetch	フェッチを実行するために Twilio API にリクエストを行います。
reader	read	読み取りを実行するために Twilio API にリクエストを行います。
アップデーター	update	更新を実行するために Twilio API にリクエストを行います

使用可能なエンドポイントは、エンドポイント 接頭辞によって異なります。

エンドポイントとオプションの詳細については、https://www.twilio.com/docs/libraries/reference/twilio-java/index.html の API ドキュメントを参照してください。

323.4. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Options と **consumer.** の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。

Twilio から通話やメッセージを受信し、Camel コンシューマーエンドポイントを使って応答する場合は、**camel-servlet**、**camel-undertow**、**camel-jetty**、**camel-netty-http** などのHTTPベースのコンポーネントを使用して、TwiML で応答することができます。

323.5. メッセージヘッダー

CamelTwilio 接頭辞を使用してプロデューサーエンドポイントのメッセージヘッダーに任意のオプションを指定できます。

323.6. メッセージボディー

すべての結果メッセージボディーは、Twilio Java SDK によって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

第324章 TWITTER コンポーネント

Camel バージョン 2.10 以降で利用可能

camel-twitter は 4 つのコンポーネントで設定されています。

- Twitter のダイレクトメッセージ
- Twitter 検索
- Twitter Streaming
- Twitter Timeline

Twitter コンポーネントは、Twitter4J をカプセル化することにより、Twitter API の最も便利な機能を有効にします。タイムライン、ユーザー、トレンド、およびダイレクトメッセージを直接、ポーリング、またはイベント駆動型で使用できます。また、ステータス更新またはダイレクトメッセージとしてのメッセージの作成もサポートしています。

Twitter では、すべてのクライアントアプリケーション認証に OAuth を使用する必要があります。アカウントで camel-twitter を使用するには、https://dev.twitter.com/apps/new で Twitter 内に新しいアプリケーションを作成し、アプリケーションにアカウントへのアクセスを許可する必要があります。最後に、アクセストークンとシークレットを生成します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twitter</artifactId>
  <version>${camel-version}</version>
</dependency>
```

324.1. コンシューマーエンドポイント

エンドポイントが 1 つのルートエクスチェンジで List を返すのではなく、camel-twitter は、返されたオブジェクトごとに 1 つのルートエクスチェンジを作成します。例として、timeline/home の結果が 5 つのステータスの場合、ルートは 5 回 (ステータスごとに 1 回) 実行されます。

エンドポイント	コンテキスト	ボディタイプ	注意
twitter-directmessage	直接、ポーリング	twitter4j.DirectMessage	
twitter検索	直接、ポーリング	twitter4j.Status	

エンドポイント	コンテキスト	ボディタイプ	注意
twitter-streaming	イベント、ポーリング	twitter4j.Status	
twitter-timeline	直接、ポーリング	twitter4j.Status	

324.2. プロデューサーエンドポイント

エンドポイント	ボディタイプ	注意
twitter-directmessage	String	
twitter検索	List<twitter4j.Status>	
twitter-timeline	String	プロデューサーではユーザーのタイムラインタイプのみがサポートされています

324.3. メッセージヘッダー

名前	説明
CamelTwitterKeywords	このヘッダーは、検索キーワードを動的に変更するために検索プロデューサーによって使用されます。
CamelTwitterSearchLanguage	Camel 2.11.0: このヘッダーは、検索エンドポイントの検索言語を動的に設定する **lang** のオプションをオーバーライドできます。
CamelTwitterCount	Camel 2.11.0 このヘッダーは、返される最大の twitter を設定する **count** のオプションをオーバーライドできます。
CamelTwitterNumberOfPages	Camel 2.11.0 このヘッダーは、Twitter に返すページ数を設定する **numberOfPages** オプションをオーバーライドできます。

324.4. メッセージボディー

すべてのメッセージボディーは、Twitter4J API によって提供されるオブジェクトを利用します。

324.5. ユースケース

注記

API レート制限: Twitter4J によってカプセル化された Twitter REST API は、 API レート制限 の対象となります。メソッドごとの制限については、API レート制限の ドキュメントを参照してください。そのページにリストされていないエンドポイント/リソースは、ウィンドウごとに割り当てられたユーザーごとに 15 リクエストにデフォルト設定されていることに注意してください。

324.5.1. Twitter プロファイル内でステータス更新を作成するには、このプロデューサーに String ボディーを送信する:

```
from("direct:foo")
  .to("twitter-timeline://user?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]);
```

324.5.2. ホームタイムラインのすべてのステータスを 60 秒ごとにポーリングするには:

```
from("twitter-timeline://home?type=polling&delay=60&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

324.5.3. キーワード camel ですべてのステータスを 1 回だけ検索するには:

```
from("twitter-search://foo?type=polling&keywords=camel&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

324.5.4. 静的キーワードを持つプロデューサーを使用した検索:

```
from("direct:foo")
  .to("twitter-search://foo?keywords=camel&consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

324.5.5. ヘッダーからの動的キーワードを含むプロデューサーを使用した検索:

bar ヘッダーには検索するキーワードがあるため、この値を **CamelTwitterKeywords** ヘッダーに割り当てることができます。

```
from("direct:foo")
  .setHeader("CamelTwitterKeywords", header("bar"))
  .to("twitter-search://foo?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

324.6. 例

Twitter Websocket の例 も参照してください。

324.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Twitter Websocket の例

第325章 TWITTER DIRECT MESSAGE コンポーネント

Camel バージョン 2.10 以降で利用可能

Twitter Direct Message コンポーネントは、ユーザーのダイレクトメッセージを消費/生成します。

325.1. コンポーネントオプション

Twitter Direct Message コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
accessToken (security)	アクセストークン		String
accessTokenSecret (security)	アクセストークンのシークレット		String
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。		String
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。		int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

325.2. エンドポイントオプション

Twitter Direct Message エンドポイントは、URI 構文を使用して設定されます。

```
twitter-directmessage:user
```

パスおよびクエリーパラメーターを使用します。

325.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
user	**必須** ダイレクトメッセージを送信するためのユーザー名。これはコンシューマーには無視されます。		String

325.2.2. クエリーパラメーター (42 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
type (consumer)	使用するエンドポイントタイプ。ストリーミングのみがイベントタイプをサポートします。	ポーリング	EndpointType
distanceMetric (consumer)	非ストリーム地理検索で使用され、設定されたメトリックを使用して半径で検索します。単位は、マイルを表す mi またはキロメートルを表す km のいずれかです。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。	km	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
extendedMode (consumer)	Twitter から全文を有効にするために使用されます (例: 140 文字を超えるツイートを受信する)。	true	boolean

名前	説明	デフォルト	タイプ
latitude (consumer)	非ストリーム地理検索で緯度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
locations (consumer)	緯度/経度のペアによって作成される境界ボックス。ストリーミング/フィルターに使用できます。ペアは緯度、経度として定義されます。また、複数のペアーはセミコロンで区切ることができます。		String
longitude (consumer)	非ストリーム地理検索で経度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
radius (consumer)	非ストリーム地理検索で半径で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
twitterStream (consumer)	TwitterStream のカスタムインスタンスを使用する場合。		TwitterStream
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
count (filter)	ページあたりの結果数を制限します。		Integer
filterOld (filter)	以前にポーリングされた古いツイートを除外します。この状態はメモリーにのみ保存され、最後のツイート ID に基づいています。	true	boolean
lang (filter)	検索に使用される言語文字列 ISO_639-1		String
numberOfPages (filter)	camel-twitter が消費する結果のページ数。	1	Integer

名前	説明	デフォルト	タイプ
sinceId (filter)	ツイートをプルするために使用される最後のツイート ID。長時間実行後に camel ルートを再開するときに役立ちます。	1	long
userIds (filter)	ストリーミング/フィルターのユーザー ID でフィルター処理します。複数の値はコンマで区切ることができます。		String
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	30000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler

名前	説明	デフォルト	タイプ
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
sortById (sort)	ID で並べ替えます。古いものが最初に、新しいものが最後になります。	true	boolean
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。代わりに TwitterComponent レベルで設定することもできます。		Integer
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。代わりに TwitterComponent レベルで設定することもできます。		String
accessToken (security)	アクセストークン。代わりに TwitterComponent レベルで設定することもできます。		String
accessTokenSecret (security)	アクセスシークレット。代わりに TwitterComponent レベルで設定することもできます。		String
consumerKey (security)	コンシューマーキー。代わりに TwitterComponent レベルで設定することもできます。		String

名前	説明	デフォルト	タイプ
consumerSecret (security)	コンシューマーシークレット。代わりに TwitterComponent レベルで設定することもできます。		文字列

第326章 TWITTER SEARCH コンポーネント

Camel バージョン 2.10 以降で利用可能

Twitter Search コンポーネントは、検索結果を使用します。

326.1. コンポーネントオプション

Twitter Search コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
accessToken (security)	アクセストークン		String
accessTokenSecret (security)	アクセストークンのシークレット		String
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。		String
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。		int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

326.2. エンドポイントオプション

Twitter Search エンドポイントは、URI 構文を使用して設定されます。

```
twitter-search:keywords
```

パスおよびクエリーパラメーターを使用します。

326.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
keywords	**必須** 検索キーワード。複数の値はコンマで区切ることができます。		String

326.2.2. クエリーパラメーター (42 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
type (consumer)	使用するエンドポイントタイプ。ストリーミングのみがイベントタイプをサポートします。	ポーリング	EndpointType
distanceMetric (consumer)	非ストリーム地理検索で使用され、設定されたメトリックを使用して半径で検索します。単位は、マイルを表す mi またはキロメートルを表す km のいずれかです。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。	km	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
extendedMode (consumer)	Twitter から全文を有効にするために使用されます (例: 140 文字を超えるツイートを受信する)。	true	boolean

名前	説明	デフォルト	タイプ
latitude (consumer)	非ストリーム地理検索で緯度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
locations (consumer)	緯度/経度のペアによって作成される境界ボックス。ストリーミング/フィルターに使用できます。ペアは緯度、経度として定義されます。また、複数のペアーはセミコロンで区切ることができます。		String
longitude (consumer)	非ストリーム地理検索で経度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
radius (consumer)	非ストリーム地理検索で半径で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
twitterStream (consumer)	TwitterStream のカスタムインスタンスを使用する場合。		TwitterStream
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
count (filter)	ページあたりの結果数を制限します。		Integer
filterOld (filter)	以前にポーリングされた古いツイートを除外します。この状態はメモリーにのみ保存され、最後のツイート ID に基づいています。	true	boolean
lang (filter)	検索に使用される言語文字列 ISO_639-1		String
numberOfPages (filter)	camel-twitter が消費する結果のページ数。	1	Integer

名前	説明	デフォルト	タイプ
sinceId (filter)	ツイートをプルするために使用される最後のツイート ID。長時間実行後に camel ルートを再開するときに役立ちます。	1	long
userIds (filter)	ストリーミング/フィルターのユーザー ID でフィルター処理します。複数の値はコンマで区切ることができます。		String
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	30000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler

名前	説明	デフォルト	タイプ
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
sortById (sort)	ID で並べ替えます。古いものが最初に、新しいものが最後になります。	true	boolean
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。代わりに TwitterComponent レベルで設定することもできます。		Integer
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。代わりに TwitterComponent レベルで設定することもできます。		String
accessToken (security)	アクセストークン。代わりに TwitterComponent レベルで設定することもできます。		String
accessTokenSecret (security)	アクセスシークレット。代わりに TwitterComponent レベルで設定することもできます。		String
consumerKey (security)	コンシューマーキー。代わりに TwitterComponent レベルで設定することもできます。		String

名前	説明	デフォルト	タイプ
consumerSecret (security)	コンシューマーシークレット。代わりに TwitterComponent レベルで設定することもできます。		文字列

第327章 TWITTER STREAMING コンポーネント

Camel バージョン 2.10 以降で利用可能

Twitter Streaming コンポーネントは、ストリーミング API を使用して Twitter ステータスを消費します。

327.1. コンポーネントオプション

Twitter Streaming コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
accessToken (security)	アクセストークン		String
accessTokenSecret (security)	アクセストークンのシークレット		String
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。		String
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。		int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

327.2. エンドポイントオプション

Twitter Streaming エンドポイントは、URI 構文を使用して設定されます。

```
twitter-streaming:streamingType
```

パスおよびクエリーパラメーターを使用します。

327.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
streamingType	**必須** 使用するストリーミングタイプ。		StreamingType

327.2.2. クエリーパラメーター(43 個のパラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
type (consumer)	使用するエンドポイントタイプ。ストリーミングのみがイベントタイプをサポートします。	ポーリング	EndpointType
distanceMetric (consumer)	非ストリーム地理検索で使用され、設定されたメトリックを使用して半径で検索します。単位は、マイルを表す mi またはキロメートルを表す km のいずれかです。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。	km	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
extendedMode (consumer)	Twitter から全文を有効にするために使用されます (例: 140 文字を超えるツイートを受信する)。	true	boolean

名前	説明	デフォルト	タイプ
latitude (consumer)	非ストリーム地理検索で緯度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
locations (consumer)	緯度/経度のペアによって作成される境界ボックス。ストリーミング/フィルターに使用できます。ペアは緯度、経度として定義されます。また、複数のペアーはセミコロンで区切ることができます。		String
longitude (consumer)	非ストリーム地理検索で経度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
radius (consumer)	非ストリーム地理検索で半径で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
twitterStream (consumer)	TwitterStream のカスタムインスタンスを使用する場合。		TwitterStream
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
count (filter)	ページあたりの結果数を制限します。		Integer
filterOld (filter)	以前にポーリングされた古いツイートを除外します。この状態はメモリーにのみ保存され、最後のツイート ID に基づいています。	true	boolean
keywords (filter)	ストリーミングフィルターに使用できます。複数の値はコンマで区切ることができます。		String
lang (filter)	検索に使用される言語文字列 ISO_639-1		String

名前	説明	デフォルト	タイプ
numberOfPages (filter)	camel-twitter が消費する結果のページ数。	1	Integer
sinceId (filter)	ツイートをプルするために使用される最後のツイート ID。長時間実行後に camel ルートを再開するときに役立ちます。	1	long
userIds (filter)	ストリーミング/フィルターのユーザー ID でフィルター処理します。複数の値はコンマで区切ることができます。		String
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	30000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

名前	説明	デフォルト	タイプ
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
sortById (sort)	ID で並べ替えます。古いものが最初に、新しいものが最後になります。	true	boolean
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。代わりに TwitterComponent レベルで設定することもできます。		Integer
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。代わりに TwitterComponent レベルで設定することもできます。		String
accessToken (security)	アクセストークン。代わりに TwitterComponent レベルで設定することもできます。		String
accessTokenSecret (security)	アクセスシークレット。代わりに TwitterComponent レベルで設定することもできます。		String
consumerKey (security)	コンシューマーキー。代わりに TwitterComponent レベルで設定することもできます。		String

名前	説明	デフォルト	タイプ
consumerSecret (security)	コンシューマーシークレット。代わりに TwitterComponent レベルで設定することもできます。		文字列

第328章 TWITTER TIMELINE コンポーネント

Camel バージョン 2.10 以降で利用可能

Twitter Timeline コンポーネントは、Twitter タイムラインを使用するか、特定のユーザーのステータスを更新します。

328.1. コンポーネントオプション

Twitter Timeline コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
accessToken (security)	アクセストークン		String
accessTokenSecret (security)	アクセストークンのシークレット		String
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。		String
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。		int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

328.2. エンドポイントオプション

Twitter Timeline エンドポイントは、URI 構文を使用して設定されます。

```
twitter-timeline:timelineType
```

パスおよびクエリーパラメーターを使用します。

328.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
timelineType	**必須** 生成/消費するタイムラインタイプ。		TimelineType

328.2.2. クエリーパラメーター(43 個のパラメーター):

名前	説明	デフォルト	タイプ
user (Common)	timelineType=user を使用する場合のユーザー名。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
type (consumer)	使用するエンドポイントタイプ。ストリーミングのみがイベントタイプをサポートします。	ポーリング	EndpointType
distanceMetric (consumer)	非ストリーム地理検索で使用され、設定されたメトリックを使用して半径で検索します。単位は、マイルを表す mi またはキロメートルを表す km のいずれかです。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。	km	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
extendedMode (consumer)	Twitter から全文を有効にするために使用されます (例: 140 文字を超えるツイートを受信する)。	true	boolean
latitude (consumer)	非ストリーム地理検索で緯度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
locations (consumer)	緯度/経度のペアによって作成される境界ボックス。ストリーミング/フィルターに使用できます。ペアは緯度、経度として定義されます。また、複数のペアーはセミコロンで区切ることができます。		String
longitude (consumer)	非ストリーム地理検索で経度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
radius (consumer)	非ストリーム地理検索で半径で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
twitterStream (consumer)	TwitterStream のカスタムインスタンスを使用する場合。		TwitterStream
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
count (filter)	ページあたりの結果数を制限します。		Integer
filterOld (filter)	以前にポーリングされた古いツイートを除外します。この状態はメモリーにのみ保存され、最後のツイート ID に基づいています。	true	boolean
lang (filter)	検索に使用される言語文字列 ISO_639-1		String
numberOfPages (filter)	camel-twitter が消費する結果のページ数。	1	Integer

名前	説明	デフォルト	タイプ
sinceId (filter)	ツイートをプルするために使用される最後のツイート ID。長時間実行後に camel ルートを再開するときに役立ちます。	1	long
userIds (filter)	ストリーミング/フィルターのユーザー ID でフィルター処理します。複数の値はコンマで区切ることができます。		String
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	30000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler

名前	説明	デフォルト	タイプ
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
sortById (sort)	ID で並べ替えます。古いものが最初に、新しいものが最後になります。	true	boolean
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。代わりに TwitterComponent レベルで設定することもできます。		Integer
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。代わりに TwitterComponent レベルで設定することもできます。		String
accessToken (security)	アクセストークン。代わりに TwitterComponent レベルで設定することもできます。		String
accessTokenSecret (security)	アクセスシークレット。代わりに TwitterComponent レベルで設定することもできます。		String
consumerKey (security)	コンシューマーキー。代わりに TwitterComponent レベルで設定することもできます。		String
consumerSecret (security)	コンシューマーシークレット。代わりに TwitterComponent レベルで設定することもできます。		文字列

第329章 TWITTER コンポーネント (非推奨)

Camel バージョン 2.10 以降で利用可能

重要

複合 twitter コンポーネントは廃止されました。ダイレクトメッセージ、検索、ストリーミング、タイムラインに個別のコンポーネントを使用します。

- Twitter コンポーネント
  - Twitter のダイレクトメッセージ
  - Twitter 検索
  - Twitter Streaming
  - Twitter Timeline

Twitter コンポーネントは、Twitter4J をカプセル化することにより、Twitter API の最も便利な機能を有効にします。タイムライン、ユーザー、トレンド、およびダイレクトメッセージを直接、ポーリング、またはイベント駆動型で使用できます。また、ステータス更新またはダイレクトメッセージとしてのメッセージの作成もサポートしています。

Twitter では、すべてのクライアントアプリケーション認証に OAuth を使用する必要があります。アカウントで camel-twitter を使用するには、https://dev.twitter.com/apps/new で Twitter 内に新しいアプリケーションを作成し、アプリケーションにアカウントへのアクセスを許可する必要があります。最後に、アクセストークンとシークレットを生成します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twitter</artifactId>
  <version>${camel-version}</version>
</dependency>
```

329.1. URI 形式

```
twitter://endpoint[?options]
```

329.2. TWITTER コンポーネント

twitter コンポーネントは、使用前に必須の Twitter アカウント設定で設定できます。

Twitter コンポーネントは、以下に示す 9 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
accessToken (security)	アクセストークン		String

名前	説明	デフォルト	タイプ
accessTokenSecret (security)	アクセストークンのシークレット		String
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。		String
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。		int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

これらのオプションは、エンドポイントで直接設定することもできます。

329.3. コンシューマーエンドポイント

エンドポイントが 1 つのルートエクスチェンジで List を返すのではなく、camel-twitter は、返されたオブジェクトごとに 1 つのルートエクスチェンジを作成します。例として、timeline/home の結果が 5 つのステータスの場合、ルートは 5 回 (ステータスごとに 1 回) 実行されます。

エンドポイント	コンテキスト	ボディタイプ	注意
directmessage	直接、ポーリング	twitter4j.DirectMessage	
search	直接、ポーリング	twitter4j.Status	

エンドポイント	コンテキスト	ボディタイプ	注意
streaming/filter	イベント、ポーリング	twitter4j.Status	
streaming/sample	イベント、ポーリング	twitter4j.Status	
streaming/user	イベント、ポーリング	twitter4j.Status	**Camel 2.16**: 保護されたユーザーとアカウントからのツイートを受信します。
timeline/home	直接、ポーリング	twitter4j.Status	
timeline/mentions	直接、ポーリング	twitter4j.Status	
timeline/public	**直接、ポーリング**	**twitter4j.Status**	**@deprecated.代わりに、timeline/home または direct/home を使用してください。*Camel 2.11 から削除**以降。*
timeline/retweetsofme	直接、ポーリング	twitter4j.Status	
timeline/user	直接、ポーリング	twitter4j.Status	
trends/daily	***Camel 2.10.1: 直接、ポーリング***	**twitter4j.Status**	**@deprecated.Camel 2.11 以降から削除されました。**
trends/weekly	***Camel 2.10.1: 直接、ポーリング***	**twitter4j.Status**	**@deprecated.Camel 2.11 以降から削除されました。**

329.4. プロデューサーエンドポイント

エンドポイント	ボディタイプ
directmessage	String
search	List<twitter4j.Status>
timeline/user	String

329.5. URI オプション

Twitter エンドポイントは、URI 構文を使用して設定されます。

```
twitter:kind
```

パスおよびクエリーパラメーターを使用します。

329.5.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
kind	**必須** エンドポイントの種類		String

329.5.2. クエリーパラメーター(44 個のパラメーター):

名前	説明	デフォルト	タイプ
user (Common)	ユーザーのタイムラインの消費、ダイレクトメッセージの作成などに使用されるユーザー名。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

名前	説明	デフォルト	タイプ
type (consumer)	使用するエンドポイントタイプ。ストリーミングのみがイベントタイプをサポートします。	ポーリング	EndpointType
distanceMetric (consumer)	非ストリーム地理検索で使用され、設定されたメトリックを使用して半径で検索します。単位は、マイルを表す mi またはキロメートルを表す km のいずれかです。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。	km	String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
extendedMode (consumer)	Twitter から全文を有効にするために使用されます (例: 140 文字を超えるツイートを受信する)。	true	boolean
latitude (consumer)	非ストリーム地理検索で緯度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
locations (consumer)	緯度/経度のペアによって作成される境界ボックス。ストリーミング/フィルターに使用できます。ペアは緯度、経度として定義されます。また、複数のペアーはセミコロンで区切ることができます。		String
longitude (consumer)	非ストリーム地理検索で経度で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy

名前	説明	デフォルト	タイプ
radius (consumer)	非ストリーム地理検索で半径で検索するために使用されます。longitude、latitude、radius、および distanceMetric のオプションをすべて設定する必要があります。		double
twitterStream (consumer)	TwitterStream のカスタムインスタンスを使用する場合。		TwitterStream
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
count (filter)	ページあたりの結果数を制限します。		Integer
filterOld (filter)	以前にポーリングされた古いツイートを除外します。この状態はメモリーにのみ保存され、最後のツイート ID に基づいています。	true	boolean
keywords (filter)	検索とストリーミング/フィルターに使用できます。複数の値はコンマで区切ることができます。		String
lang (filter)	検索に使用される言語文字列 ISO_639-1		String
numberOfPages (filter)	camel-twitter が消費する結果のページ数。	1	Integer
sinceId (filter)	ツイートをプルするために使用される最後のツイート ID。長時間実行後に camel ルートを再開するときに役立ちます。	1	long
userIds (filter)	ストリーミング/フィルターのユーザー ID でフィルター処理します。複数の値はコンマで区切ることができます。		String
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

名前	説明	デフォルト	タイプ
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。	30000	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

名前	説明	デフォルト	タイプ
sortById (sort)	ID で並べ替えます。古いものが最初に、新しいものが最後になります。	true	boolean
httpProxyHost (proxy)	camel-twitter に使用できる http プロキシーホスト。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPassword (proxy)	camel-twitter に使用できる http プロキシーパスワード。代わりに TwitterComponent レベルで設定することもできます。		String
httpProxyPort (proxy)	camel-twitter に使用できる http プロキシーポート。代わりに TwitterComponent レベルで設定することもできます。		Integer
httpProxyUser (proxy)	camel-twitter に使用できる http プロキシーユーザー。代わりに TwitterComponent レベルで設定することもできます。		String
accessToken (security)	アクセストークン。代わりに TwitterComponent レベルで設定することもできます。		String
accessTokenSecret (security)	アクセスシークレット。代わりに TwitterComponent レベルで設定することもできます。		String
consumerKey (security)	コンシューマーキー。代わりに TwitterComponent レベルで設定することもできます。		String
consumerSecret (security)	コンシューマーシークレット。代わりに TwitterComponent レベルで設定することもできます。		文字列

329.6. メッセージヘッダー

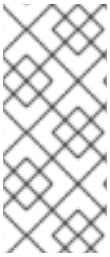
名前	説明
CamelTwitterKeywords	このヘッダーは、検索キーワードを動的に変更するために検索プロデューサーによって使用されます。
CamelTwitterSearchLanguage	Camel 2.11.0: このヘッダーは、検索エンドポイントの検索言語を動的に設定する **lang** のオプションをオーバーライドできます。
CamelTwitterCount	Camel 2.11.0 このヘッダーは、返される最大の twitter を設定する **count** のオプションをオーバーライドできます。

名前	説明
CamelTwitterNumberOfPages	Camel 2.11.0 このヘッダーは、Twitter に返すページ数を設定する **numberOfPages** オプションをオーバーライドできます。

329.7. メッセージボディー

すべてのメッセージボディーは、Twitter4J API によって提供されるオブジェクトを利用します。

329.8. ユースケース



注記

API レート制限: Twitter4J によってカプセル化された Twitter REST API は、 API レート制限 の対象となります。メソッドごとの制限については、API レート制限の ドキュメントを参照してください。そのページにリストされていないエンドポイント/リソースは、ウィンドウごとに割り当てられたユーザーごとに 15 リクエストにデフォルト設定されていることに注意してください。

329.8.1. Twitter プロファイル内でステータス更新を作成するには、このプロデューサーに String ボディーを送信する:

```
from("direct:foo")
  .to("twitter://timeline/user?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]);
```

329.8.2. ホームタイムラインのすべてのステータスを 60 秒ごとにポーリングするには:

```
from("twitter://timeline/home?type=polling&delay=60&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

329.8.3. キーワード camel ですべてのステータスを 1 回だけ検索するには:

```
from("twitter://search?type=polling&keywords=camel&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

329.8.4. 静的キーワードを持つプロデューサーを使用した検索:

```
from("direct:foo")
  .to("twitter://search?keywords=camel&consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

329.8.5. ヘッダーからの動的キーワードを含むプロデューサーを使用した検索:

bar ヘッダーには検索するキーワードがあるため、この値を **CamelTwitterKeywords** ヘッダーに割り当てることができます。

```
from("direct:foo")
  .setHeader("CamelTwitterKeywords", header("bar"))
  .to("twitter://search?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

329.9. 例

Twitter Websocket の例 も参照してください。

329.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- Twitter Websocket の例

第330章 UNDERTOW コンポーネント

Camel バージョン 2.16 以降で利用可能

undertow コンポーネントは、HTTP/WebSocket リクエストを消費および生成するための HTTP および WebSocket ベースのエンドポイントを提供します。

つまり、Undertow コンポーネントは単純な Web サーバーとして動作します。Undertow は http クライアントとしても使用できます。つまり、Camel をプロデューサーとして使用することもできます。

ヒント

Camel バージョン 2.21 以降、**undertow** コンポーネントは WebSocket 接続もサポートするため、Camel Websocket コンポーネントまたは Atmium-Websocket コンポーネントのドロップイン置換として機能できます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-undertow</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

330.1. URI 形式

```
undertow:http://hostname[:port][/resourceUri][?options]
undertow:https://hostname[:port][/resourceUri][?options]
undertow:ws://hostname[:port][/resourceUri][?options]
undertow:wss://hostname[:port][/resourceUri][?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

330.2. オプション

Undertow コンポーネントは、以下にリストされている 5 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
undertowHttpBinding (advanced)	カスタム HttpBinding を使用して、Camel メッセージと HttpClient との間のマッピングを制御します。		UndertowHttpBinding
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean

名前	説明	デフォルト	タイプ
hostOptions (advanced)	スレッドプールなどの一般的なオプションを設定する場合。		UndertowHostOptions
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Undertow エンドポイントは、URI 構文を使用して設定されます。

```
undertow:httpURI
```

パスおよびクエリーパラメーターを使用します。

330.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
httpURI	**必須** 使用する HTTP エンドポイントの url。		URI

330.2.2. クエリーパラメーター(21 パラメーター):

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
httpMethodRestrict (consumer)	GET/POST/PUT など、HttpMethod が一致する場合にのみ消費を許可するために使用されます。複数のメソッドをコンマで区切って指定できます。		String
matchOnUriPrefix (consumer)	完全に一致するものが見つからない場合に、コンシューマーが URI 接頭辞を照合してターゲットコンシューマーを見つけようとするかどうか。	false	Boolean

名前	説明	デフォルト	タイプ
optionsEnabled (consumer)	このサーブレットコンシューマーに対して HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは、OPTIONS はオフになっています。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cookieHandler (producer)	HTTP セッションを維持するようにクッキーハンドラーを設定します。		CookieHandler
keepAlive (producer)	非アクティブのためにソケットが閉じられないようにするための設定	true	Boolean
options (producer)	追加のチャネルオプションを設定します。使用できるオプションは org.xnio.Options で定義されています。エンドポイント uri から設定するには、各オプションの前に option. を付けます (例: option.close-abort=true&option.send-buffer=8192)。		Map
reuseAddresses (producer)	ソケットの多重化を容易にするための設定	true	Boolean
tcpNoDelay (producer)	TCP プロトコルのパフォーマンスを向上させるための設定	true	Boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に HttpOperationFailedException を出力することを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	Boolean

名前	説明	デフォルト	タイプ
transferException (producer)	有効にすると、エクスチェンジがコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合に、例外がシリアライズされました。プロデューサー側では、例外がデシリアライズされ、HttpOperationFailedException ではなくそのまま出力されます。原因となった例外はシリアライズする必要があります。これは、デフォルトでオフになっています。これを有効にすると、Java が受信データをリクエストから Java にデシリアライズし、セキュリティー上のリスクが生じる可能性があることに注意してください。	false	Boolean
headerFilterStrategy (advanced)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
undertowHttpBinding (advanced)	カスタムの UndertowHttpBinding を使用して、Camel メッセージと undertow の間のマッピングを制御します。		UndertowHttpBinding
fireWebSocketChannelEvents (websocket)	true の場合、コンシューマーは、新しい WebSocket ピアが接続、切断などしたときにルートに通知を送信します。UndertowConstants.EVENT_TYPE および EventType を参照してください。	false	boolean
sendTimeout (websocket)	Websocket チャネルに送信するときのミリ秒単位のタイムアウト。デフォルトのタイムアウトは 30000 (30 秒) です。	30000	Integer
sendToAll (websocket)	すべての websocket サブスクライバーに送信します。メッセージで UndertowConstants.SEND_TO_ALL ヘッダーを使用する代わりに、エンドポイントレベルで設定するために使用できます。		Boolean
useStreaming (websocket)	true の場合、WebSocket 経由のテキストメッセージとバイナリーメッセージは、Exchange に渡される前にそれぞれ java.io.Reader と java.io.InputStream としてラップされます。それ以外の場合は、それぞれ String および byte として渡されます。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

330.3. メッセージヘッダー

Camel は、HTTP コンポーネントと同じメッセージヘッダーを使用します。Camel 2.2 からは、**Exchange.HTTP_CHUNKED,CamelHttpChunked** ヘッダーも使用して、camel-undertow コンシューマーで chuched エンコーディングをオンまたはオフにします。

Camel は **すべての** request.parameter と request.headers にもデータを取り込みます。たとえば、クライアントリクエストの URL が **http://myserver/myserver?orderid=123** の場合、エクスチェンジには、値が 123 の **orderid** という名前のヘッダーが含まれます。

330.4. HTTP プロデューサーの例

以下は、HTTP 要求を既存の HTTP エンドポイントに送信する方法の基本的な例です。

Java DSL で

```
from("direct:start")
    .to("undertow:http://www.google.com");
```

または XML です。

```
<route>
   <from uri="direct:start"/>
   <to uri="undertow:http://www.google.com"/>
<route>
```

330.5. HTTP コンシューマーの例

このサンプルでは、**http://localhost:8080/myapp/myservice** で HTTP サービスを公開するルートを定義します。

```
<route>
  <from uri="undertow:http://localhost:8080/myapp/myservice"/>
  <to uri="bean:myBean"/>
</route>
```

330.6. WEBSOCKET の例

このサンプルでは、**http://localhost:8080/myapp/mysocket** で WebSocket サービスを公開し、同じチャネルにレスポンスを返すルートを定義します。

```
<route>
  <from uri="undertow:ws://localhost:8080/myapp/mysocket"/>
  <transform><simple>Echo ${body}</simple></transform>
  <to uri="undertow:ws://localhost:8080/myapp/mysocket"/>
</route>
```

330.7. LOCALHOST をホストとして使用する

URL で **localhost** を指定すると、Camel はローカルの TCP/IP ネットワークインターフェイスでのみエンドポイントを公開するため、動作するマシンの外部からはアクセスできません。

特定のネットワークインターフェイスで Jetty エンドポイントを公開する必要がある場合は、このインターフェイスの数値 IP アドレスをホストとして使用する必要があります。すべてのネットワークインターフェイスで Jetty エンドポイントを公開する必要がある場合は、**0.0.0.0** アドレスを使用する必要があります。

URI 接頭辞全体をリッスンするには、Jetty でワイルドカードをマッチさせるには を参照してください。

実際に HTTP でルートを公開する必要があり、すでにサーブレットがある場合は、代わりに サーブレットトランスポート を参照する必要があります。

330.8. {WILDFLY} の UNDERTOW コンシューマー

{wildfly} の camel-undertow コンシューマーの設定は、スタンドアロンの Camel の設定とは異なります。プロデューサーエンドポイントは通常どおりに機能します。

{wildfly} では、camel-undertow コンシューマーは、コンテナーによって提供されるデフォルトの Undertow HTTP サーバーを利用します。サーバーは undertow サブシステム設定内で定義されます。以下は、standalone.xml からのデフォルト設定の抜粋です。

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default" />
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true" />
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true" />
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content" />
            <filter-ref name="server-header" />
            <filter-ref name="x-powered-by-header" />
            <http-invoker security-realm="ApplicationRealm" />
        </host>
    </server>
</subsystem>
```

この例では、Undertow は、http および https ソケットバインディングで指定されたインターフェイス/ポートをリッスンするように設定されています。デフォルトでは、これは http の場合はポート 8080、https の場合は 8443 です。

これには次の意味があります。

- camel-undertow コンシューマーは localhost:8080 または localhost:8443 にのみバインドします
- これらの設定は {wildfly} コンテナーによって管理されるため、一部のエンドポイントコンシューマー設定オプションは効果がありません (以下を参照)。

たとえば、異なるホストまたはポートの組み合わせを使用してエンドポイントコンシューマーを設定すると、サーバーログファイル内に警告が表示されます。たとえば、次のホストとポートの設定は無視されます。

```
from("undertow:http://somehost:1234/path/to/resource")
```

```
[org.wildfly.extension.camel] (pool-2-thread-1) Ignoring configured host:
http://somehost:1234/path/to/resource
```

ただし、コンシューマーはデフォルトのホストとポート localhost:8080 または localhost:8443 で引き続き利用できます。

330.8.1. 代替ポートの設定

代替ポートを受け入れる場合は、{wildfly} サブシステム設定を介してこれらを設定する必要があります。これは、サーバーのドキュメントで説明されています。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html/configuration_guide/configuring_the_web_serv

330.8.2. {wildfly} で無視された camel-undertow コンシューマー設定オプション

hostOptions

サーバーホストオプションの設定方法については、{wildfly} undertow 設定ガイドを参照してください。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html-single/how_to_configure_server_security/#configure_one_way_and_two_way_ssl_tls_for_application

sslContextParameters

SSL を設定するには、{wildfly} SSL 設定ガイドを参照してください。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html-single/how_to_configure_server_security/#configure_one_way_and_two_way_ssl_tls_for_application

第331章 UNIVOCITY CSV DATAFORMAT

Camel バージョン 2.15 以降で利用可能

この データ形式 は、次の 3 種類の表形式データテキストファイルの読み取りと書き込みに uniVocity パーサー を使用します。

- 値が記号 (通常はコンマ) で区切られている CSV (Comma Separated Values)
- 値が既知のサイズである固定幅
- TSV (Tabular Separated Values)、フィールドが表で区切られている場合

したがって、uniVocity パーサーに基づく 3 つのデータ形式があります。

Maven を使用している場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページ を参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

331.1. オプション

uniVocity パーサーのほとんどの設定オプションは、データ形式で利用できます。特定のオプションの詳細については、ドキュメントページ を参照してください。

3 つのデータ形式には共通のオプションがあり、専用のオプションがあります。このセクションでは、それらすべてを紹介します。

331.2. オプション

uniVocity CSV データ形式は、以下に示す 18 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
quoteAllFields	**false**	**Boolean**	値を記述するときにすべての値を引用符で囲む必要があるかどうか。
quote	"	**String**	引用記号。
quoteEscape	"	**String**	引用エスケープ記号。
delimiter	,	**String**	値の区切り文字。
nullValue		**String**	null 値の文字列表現。デフォルト値は null です。

名前	デフォルト	Java タイプ	説明
skipEmptyLines	**true**	**Boolean**	空行を無視するかどうか。デフォルト値は true です。
ignoreTrailingWhitespaces	**true**	**Boolean**	末尾の空白を無視する必要があるかどうか。デフォルト値は true です。
ignoreLeadingWhitespaces	**true**	**Boolean**	先頭の空白を無視する必要があるかどうか。デフォルト値は true です。
headersDisabled	**false**	**Boolean**	ヘッダーが無効になっているかどうか。このオプションを定義すると、ヘッダーが null として明示的に設定されます。これは、ヘッダーがないことを示します。デフォルト値は false です。
headerExtractionEnabled	**false**	**Boolean**	テストドキュメントの最初の行でヘッダーを読み取る必要があるかどうか。既定値は false です。
numberOfRecordsToRead		**Integer**	読み取るレコードの最大数。
emptyValue		**String**	空の値の文字列表現。
lineSeparator		**String**	ファイルの行区切りデフォルト値では、JVM プラットフォームの行区切りが使用されます。
normalizedLineSeparator		**String**	ファイルの正規化された行区切り。デフォルト値は改行文字です。
comment	**#**	**String**	コメント記号。デフォルト値:
lazyLoad	**false**	**Boolean**	アンマーシャリングで、その場で行を読み取る反復子を生成するか、またはすべての行を一度に読み取る必要があるか。デフォルト値は false です。
asMap	**false**	**Boolean**	アンマーシャリングで、リストではなく行の値のマップを生成するかどうか。ヘッダー (定義または収集) が必要です。デフォルト値は false です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

331.3. マーシャリングの使用法

マーシャリングは次のいずれかを受け入れます。

- マップのリスト (L`ist<Map<String, ?>>`)、各行に 1 つ
- 1 行に 1 つのマップ (**Map<String, ?>**)

他のボディは例外を出力します。

331.3.1. 使用例: Map を CSV 形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-csv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

331.3.2. 使用例: Map を固定幅形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-fixed padding="_">
      <univocity-header length="5"/>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
    </univocity-fixed>
  </marshal>
  <to uri="mock:result"/>
</route>
```

331.3.3. 使用例: Map を TSV 形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-tsv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

331.4. 用途のアンマーシャリング

アンマーシャリングは、データを読み取るために **InputStream** を使用します。

各行は次のいずれかを生成します。

- すべての値を含むリスト (**asMap** オプションを **false** に設定);
- ヘッダーによってインデックス付けされたすべての値を含むマップ (**asMap** オプションを **true** に設定)。

すべての行で次のいずれかを実行できます。

- 一度にリストに収集されます (**lazyLoad** オプションを **false** に設定);
- イテレーター (**lazyLoad** オプションを **true** に指定) を使用してオンザフライで読み取られます。

331.4.1. 使用例: CSV 形式を自動ヘッダー付きのマップにアンマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <unmarshal>
    <univocity-csv headerExtractionEnabled="true" asMap="true"/>
  </unmarshal>
  <to uri="mock:result"/>
</route>
```

331.4.2. 使用例: 固定幅形式をリストにアンマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <unmarshal>
    <univocity-fixed>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
    </univocity-fixed>
  </unmarshal>
  <to uri="mock:result"/>
</route>
```

第332章 UNIVOCITY FIXED LENGTH DATAFORMAT

Camel バージョン 2.15 以降で利用可能

この データ形式 は、次の 3 種類の表形式データテキストファイルの読み取りと書き込みに uniVocity パーサー を使用します。

- 値が記号 (通常はコンマ) で区切られている CSV (Comma Separated Values)
- 値が既知のサイズである固定幅
- TSV (Tabular Separated Values)、フィールドが表で区切られている場合

したがって、uniVocity パーサーに基づく 3 つのデータ形式があります。

Maven を使用している場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページ を参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

332.1. オプション

uniVocity パーサーのほとんどの設定オプションは、データ形式で利用できます。特定のオプションの詳細については、ドキュメントページ を参照してください。

3 つのデータ形式には共通のオプションがあり、専用のオプションがあります。このセクションでは、それらすべてを紹介します。

332.2. オプション

uniVocity 固定長データ形式は、以下に示す 17 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
skipTrailingCharsUntilNewline	**false**	**Boolean**	改行までの末尾の文字を無視する必要があるかどうか。デフォルト値は false です。
recordEndsOnNewline	**false**	**Boolean**	レコードが改行で終了するかどうか。デフォルト値は false です。
padding		**String**	パディング文字。デフォルト値はスペースです。
nullValue		**String**	null 値の文字列表現。デフォルト値は null です。
skipEmptyLines	**true**	**Boolean**	空行を無視するかどうか。デフォルト値は true です。

名前	デフォルト	Java タイプ	説明
ignoreTrailingWhitespaces	**true**	**Boolean**	末尾の空白を無視する必要があるかどうか。デフォルト値は true です。
ignoreLeadingWhitespaces	**true**	**Boolean**	先頭の空白を無視する必要があるかどうか。デフォルト値は true です。
headersDisabled	**false**	**Boolean**	ヘッダーが無効になっているかどうか。このオプションを定義すると、ヘッダーが null として明示的に設定されます。これは、ヘッダーがないことを示します。デフォルト値は false です。
headerExtractionEnabled	**false**	**Boolean**	テストドキュメントの最初の行でヘッダーを読み取る必要があるかどうか。既定値は false です。
numberOfRecordsToRead		**Integer**	読み取るレコードの最大数。
emptyValue		**String**	空の値の文字列表現。
lineSeparator		**String**	ファイルの行区切りデフォルト値では、JVM プラットフォームの行区切りが使用されます。
normalizedLineSeparator		**String**	ファイルの正規化された行区切り。デフォルト値は改行文字です。
comment	**#**	**String**	コメント記号。デフォルト値:
lazyLoad	**false**	**Boolean**	アンマーシャリングで、その場で行を読み取る反復子を生成するか、またはすべての行を一度に読み取る必要があるか。デフォルト値は false です。
asMap	**false**	**Boolean**	アンマーシャリングで、リストではなく行の値のマップを生成するかどうか。ヘッダー (定義または収集) が必要です。デフォルト値は false です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

332.3. マーシャリングの使用法

マーシャリングは次のいずれかを受け入れます。

- マップのリスト (L`ist<Map<String, ?>>`)、各行に 1 つ

- 1行に1つのマップ (**Map<String, ?>**)

他のボディは例外を出力します。

332.3.1. 使用例: Map を CSV 形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-csv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

332.3.2. 使用例: Map を固定幅形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-fixed padding="_">
      <univocity-header length="5"/>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
    </univocity-fixed>
  </marshal>
  <to uri="mock:result"/>
</route>
```

332.3.3. 使用例: Map を TSV 形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-tsv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

332.4. 用途のアンマーシャリング

アンマーシャリングは、データを読み取るために **InputStream** を使用します。

各行は次のいずれかを生成します。

- すべての値を含むリスト (**asMap** オプションを **false** に設定);
- ヘッダーによってインデックス付けされたすべての値を含むマップ (**asMap** オプションを **true** に設定)。

すべての行で次のいずれかを実行できます。

- 一度にリストに収集されます (**lazyLoad** オプションを **false** に設定);

- イテレーター (**lazyLoad** オプションを **true** に指定) を使用してオンザフライで読み取られます。

332.4.1. 使用例: CSV 形式を自動ヘッダー付きのマップにアンマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <unmarshal>
    <univocity-csv headerExtractionEnabled="true" asMap="true"/>
  </unmarshal>
  <to uri="mock:result"/>
</route>
```

332.4.2. 使用例: 固定幅形式をリストにアンマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <unmarshal>
    <univocity-fixed>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
    </univocity-fixed>
  </unmarshal>
  <to uri="mock:result"/>
</route>
```

第333章 UNIVOCITY TSV DATAFORMAT

Camel バージョン 2.15 以降で利用可能

この データ形式 は、次の 3 種類の表形式データテキストファイルの読み取りと書き込みに uniVocity パーサー を使用します。

- 値が記号 (通常はコンマ) で区切られている CSV (Comma Separated Values)
- 値が既知のサイズである固定幅
- TSV (Tabular Separated Values)、フィールドが表で区切られている場合

したがって、uniVocity パーサーに基づく 3 つのデータ形式があります。

Maven を使用している場合は、pom.xml に以下を追加して、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページ を参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

333.1. オプション

uniVocity パーサーのほとんどの設定オプションは、データ形式で利用できます。特定のオプションの詳細については、ドキュメントページ を参照してください。

3 つのデータ形式には共通のオプションがあり、専用のオプションがあります。このセクションでは、それらすべてを紹介します。

333.2. オプション

uniVocity TSV データ形式は、以下に示す 15 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
escapeChar	\	String	エスケープ文字。
nullValue		String	null 値の文字列表現。デフォルト値は null です。
skipEmptyLines	true	Boolean	空行を無視するかどうか。デフォルト値は true です。
ignoreTrailingWhitespaces	true	Boolean	末尾の空白を無視する必要があるかどうか。デフォルト値は true です。
ignoreLeadingWhitespaces	true	Boolean	先頭の空白を無視する必要があるかどうか。デフォルト値は true です。

名前	デフォルト	Java タイプ	説明
headersDisabled	**false**	**Boolean**	ヘッダーが無効になっているかどうか。このオプションを定義すると、ヘッダーが null として明示的に設定されます。これは、ヘッダーがないことを示します。デフォルト値は false です。
headerExtractionEnabled	**false**	**Boolean**	テストドキュメントの最初の行でヘッダーを読み取る必要があるかどうか。既定値は false です。
numberOfRecordsToRead		**Integer**	読み取るレコードの最大数。
emptyValue		**String**	空の値の文字列表現。
lineSeparator		**String**	ファイルの行区切りデフォルト値では、JVM プラットフォームの行区切りが使用されます。
normalizedLineSeparator		**String**	ファイルの正規化された行区切り。デフォルト値は改行文字です。
comment	**#**	**String**	コメント記号。デフォルト値:
lazyLoad	**false**	**Boolean**	アンマーシャリングで、その場で行を読み取る反復子を生成するか、またはすべての行を一度に読み取る必要があるか。デフォルト値は false です。
asMap	**false**	**Boolean**	アンマーシャリングで、リストではなく行の値のマップを生成するかどうか。ヘッダー (定義または収集) が必要です。デフォルト値は false です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

333.3. マーシャリングの使用法

マーシャリングは次のいずれかを受け入れます。

- マップのリスト (L`ist<Map<String, ?>>`)、各行に 1 つ
- 1 行に 1 つのマップ (**Map<String, ?>**)

他のボディは例外を出力します。

333.3.1. 使用例: Map を CSV 形式にマーシャリングする

■

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-csv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

333.3.2. 使用例: Map を固定幅形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-fixed padding="_">
      <univocity-header length="5"/>
      <univocity-header length="5"/>
      <univocity-header length="5"/>
    </univocity-fixed>
  </marshal>
  <to uri="mock:result"/>
</route>
```

333.3.3. 使用例: Map を TSV 形式にマーシャリングする

```
<route>
  <from uri="direct:input"/>
  <marshal>
    <univocity-tsv/>
  </marshal>
  <to uri="mock:result"/>
</route>
```

333.4. 用途のアンマーシャリング

アンマーシャリングは、データを読み取るために **InputStream** を使用します。

各行は次のいずれかを生成します。

- すべての値を含むリスト (**asMap** オプションを **false** に設定);
- ヘッダーによってインデックス付けされたすべての値を含むマップ (**asMap** オプションを **true** に設定)。

すべての行で次のいずれかを実行できます。

- 一度にリストに収集されます (**lazyLoad** オプションを **false** に設定);
- イテレーター (**lazyLoad** オプションを **true** に指定) を使用してオンザフライで読み取られます。

333.4.1. 使用例: CSV 形式を自動ヘッダー付きのマップにアンマーシャリングする

```
<route>
```

```
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-csv headerExtractionEnabled="true" asMap="true"/>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

333.4.2. 使用例: 固定幅形式をリストにアンマーシャリングする

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-fixed>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

第334章 VALIDATOR コンポーネント

Camel バージョン 1.1 以降で利用可能

Validation コンポーネントは、JAXP Validation API を使用し、サポートされている XML スキーマ言語のいずれかに基づいて、メッセージ本文の XML 検証を実行します。デフォルトは XML スキーマ です。

Jing コンポーネントは、次の便利なスキーマ言語もサポートしていることに注意してください。

- RelaxNG Compact 構文
- RelaxNG XML 構文

MSV コンポーネントは、RelaxNG XML 構文もサポートしています。

334.1. URI 形式

```
validator:someLocalOrRemoteResource
```

someLocalOrRemoteResource は、クラスパス上のローカルリソースへの URL、または検証対象の XSD を含むファイルシステム上のリモートリソースまたはリソースへの完全な URL です。以下に例を示します。

- **msv:org/foo/bar.xsd**
- **msv:file:../foo/bar.xsd**
- **msv:http://acme.com/cheese.xsd**
- **validator:com/mypackage/myschema.xsd**

Maven ユーザーは、**Camel 2.8** 以前を使用する場合、このコンポーネントの **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel 2.9 以降、Validation コンポーネントは camel-core で直接提供されます。

334.2. オプション

Validator コンポーネントは、以下に示す 2 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
resourceResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタム LSResourceResolver を使用する場合		ValidatorResourceResolverFactory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Validator エンドポイントは URI 構文を使用して設定されます。

```
validator:resourceUri
```

パスおよびクエリーパラメーターを使用します。

334.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	クラスパス上のローカルリソースへの **必須** URL、またはレジストリー内の Bean を検索するための参照、または検証対象の XSD を含むファイルシステム上のリモートリソースまたはリソースへの完全な URL。		String

334.2.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
failOnNullBody (producer)	本文が存在しない場合に失敗するかどうか。	true	boolean
failOnNullHeader (producer)	ヘッダーに対して検証するときに、ヘッダーが存在しない場合に失敗するかどうか。	true	boolean
headerName (producer)	メッセージボディーではなくヘッダーに対して検証します。		String
errorHandler (advanced)	カスタム org.apache.camel.processor.validation.ValidatorErrorHandler を使用するには。デフォルトのエラーハンドラーはエラーをキャプチャし、例外を出力します。		ValidatorErrorHandler

名前	説明	デフォルト	タイプ
resourceResolver (advanced)	カスタム LSResourceResolver を使用するには。リンク setResourceResolverFactory(ValidatorResourceResolverFactory) も併せて参照してください。		LSResourceResolver
resourceResolverFactory (advanced)	エンドポイントリソース URI に依存するリソースリゾルバーを作成するため。メソッドリンク setResourceResolver (LSResourceResolver) と組み合わせて使用しないでください。設定されていない場合は、DefaultValidatorResourceResolverFactory が使用されます		ValidatorResourceResolverFactory
schemaFactory (advanced)	カスタム javax.xml.validation.SchemaFactory を使用する場合		SchemaFactory
schemaLanguage (advanced)	W3C XML スキーマの namespace URI を設定します。	http://www.w3.org/2001/XMLSchema	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
useDom (advanced)	バリデーターが DOMSource/DOMResult または SaxSource/SaxResult を使用するかどうか。	false	boolean
useSharedSchema (advanced)	Schema インスタンスを共有するかどうか。このオプションは、JDK 1.6.x のバグを回避するために導入されました。Xerces にはこの問題はありません。	true	boolean

334.3. 例

次の 例 は、エンドポイント **direct:start** からのルートを設定する方法を示しており、このルートは、XML が指定されたスキーマ (クラスパスで提供される) と一致するかどうかに基づいて、**mock:valid** または **mock:invalid** の 2 つのエンドポイントのいずれかに移動します。.

334.4. 高度: JMX メソッド CLEARCACHEDSCHEMA

Camel 2.17 以降、JMX 操作を使用して、バリデーターエンドポイントのキャッシュされたスキーマを強制的にクリアし、**clearCachedSchema. `You can also use this method to programmatically clear the cache.This method is available on the `ValidatorEndpoint `class**.` で次のプロセス呼び出しで再読み込みします。

第335章 VELOCITY コンポーネント

Camel バージョン 1.2 以降で利用可能

velocity: コンポーネントを使用すると、Apache Velocity テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

335.1. URI 形式

```
velocity:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI です。またはリモートテンプレートの完全な URL (例: file://folder/myfile.vm) です。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

335.2. オプション

Velocity コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
velocityEngine (advanced)	VelocityEngine を使用するには、それ以外の場合は新しいエンジンが作成されます。		VelocityEngine
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Velocity エンドポイントは、URI 構文を使用して設定されます。

```
velocity:resourceUri
```

パスおよびクエリーパラメーターを使用します。

335.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** リソースへのパス。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)。		String

335.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding (producer)	リソースコンテンツの文字エンコード。		String
loaderCache (producer)	デフォルトで有効になっている速度リソースローダーキャッシュを有効または無効にします。	true	boolean
propertiesFile (producer)	VelocityEngine の初期化に使用されるプロパティーファイルの URI。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

335.3. メッセージヘッダー

velocity コンポーネントは、メッセージにいくつかのヘッダーを設定します (これらを自分で設定することはできず、Camel 2.1 からvelocity コンポーネントはこれらのヘッダーを設定しないため、動的テンプレートサポートに副作用が生じます)。

ヘッダー	説明
CamelVelocityResourceUri	**文字列** オブジェクトとしての templateName。

ヘッダー	説明
CamelVelocitySupplementalContext	Camel 2.16: 使用される VelocityContext に追加情報を追加します。このヘッダーの値は、追加されるキー/値を持つ **Map** である必要があります (同じ名前の既存のキーを上書きします)。 これは、ベロシティエンドポイントで再利用する一般的なキー/値を事前に設定するために使用できます。

Velocity 評価中に設定されたヘッダーはメッセージに返され、ヘッダーとして追加されます。次に、Velocity から Message に値を返すことができます。

たとえば、Velocity テンプレート **.tm** で **fruit** のヘッダー値を設定するには、次のようにします。

```
$in.setHeader("fruit", "Apple")
```

fruit ヘッダーは、**message.out.headers** からアクセスできるようになりました。

335.4. VELOCITY コンテキスト

Camel は Velocity コンテキスト (単なる **Map**) でエクスチェンジ情報を提供します。**Exchange** は次のように転送されます。

key	value
exchange	**Exchange** 自体。
exchange.properties	**Exchange** プロパティー。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキストインスタンス。
request	IN メッセージ
in	IN メッセージ
body	In メッセージボディー

key	value
out	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。
response	Out メッセージ (InOut メッセージエクスチェンジパターンのみ)。

Camel-2.14 以降、メッセージヘッダー *CamelVelocityContext * をこのように設定することで、独自の Velocity Context をセットアップできます。

```
VelocityContext velocityContext = new VelocityContext(variableMap);
exchange.getIn().setHeader("CamelVelocityContext", velocityContext);
```

335.5. ホットリロード

Velocity テンプレートリソースは、デフォルトで、ファイルリソースとクラスパスリソース (拡張 jar) の両方でホットリロード可能です。**contentCache=true** を設定すると、Camel はリソースを 1 回だけロードするため、ホットリロードはできません。このシナリオは、リソースが変更されない場合、実稼働環境で使用できます。

335.6. 動的テンプレート

Camel 2.1 以降で利用可能
Camel は、テンプレートまたはテンプレートコンテンツ自体の異なるリソースの場所を定義できる 2 つのヘッダーを提供します。これらのヘッダーのいずれかが設定されている場合、Camel はエンドポイントで設定されたリソースに対してこれを使用します。これにより、実行時に動的なテンプレートを提供できます。

ヘッダー	タイプ	説明
CamelVelocityResourceUri	String	**Camel 2.1:** 設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。
CamelVelocityTemplate	String	**Camel 2.1:** 設定されたエンドポイントの代わりに使用するテンプレート。

335.7. サンプル

たとえば、次のようなものを使用できます

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm");
```

Velocity テンプレートを使用して、InOut メッセージエクスチェンジ (**JMSReplyTo** ヘッダーがある場合) のメッセージへの応答を作成します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、次のルートを使用できます。

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm").
  to("activemq:Another.Queue");
```

また、**.vm** テンプレートが変更されない本番環境などでコンテンツキャッシュを使用するには、次のようにします。

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm?contentCache=true").
  to("activemq:Another.Queue");
```

そしてファイルベースのリソース:

```
from("activemq:My.Queue").
  to("velocity:file://myfolder/MyResponse.vm?contentCache=true").
  to("activemq:Another.Queue");
```

Camel 2.1 では、コンポーネントがヘッダーを介して動的に使用するテンプレートを指定できます。たとえば、次のようになります。

```
from("direct:in").
  setHeader("CamelVelocityResourceUri").constant("path/to/my/template.vm").
  to("velocity:dummy");
```

Camel 2.1では、ヘッダーを介してコンポーネントが動的に使用するヘッダーとして、テンプレートを直接指定することが可能です:

```
from("direct:in").
  setHeader("CamelVelocityTemplate").constant("Hi this is a velocity template that can do templating ${body}").
  to("velocity:dummy");
```

335.8. 電子メールのサンプル

このサンプルでは、注文確認メールに Velocity テンプレートを使用します。電子メールテンプレートは、Velocity で次のようにレイアウトされます。

```
Dear ${headers.lastName}, ${headers.firstName}

Thanks for the order of ${headers.item}.

Regards Camel Riders Bookstore
${body}
```

そして Java コード:

335.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第336章 VERT.X コンポーネント

Camel バージョン 2.12 以降で利用可能

vertx コンポーネントは、Vertx EventBus を操作するためのものです。

vertx EventBus は JSON イベントを送受信します。

情報: **Camel 2.16** 以降では、実行時に Java 1.8 を必要とする vertx 3 が使用されています。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-vertx</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

336.1. URI 形式

```
vertx:channelName[?options]
```

336.2. オプション

Vert.x コンポーネントは、以下に示す 7 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
vertxFactory (advanced)	カスタム VertxFactory 実装を使用するには		VertxFactory
host (Common)	組み込みのクラスター化された EventBus を作成するためのホスト名		String
port (Common)	組み込みのクラスター化された EventBus を作成するためのポート		int
vertxOptions (Common)	vertx の作成に使用するオプション		VertxOptions
vertx (Common)	新しい埋め込み EventBus を作成する代わりに、指定された vertx EventBus を使用するには		Vertx
timeout (common)	クラスター化された Vertx EventBus の準備が整うまで待機するタイムアウト (秒単位)。デフォルト値は 60 です。	60	int

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Vert.x エンドポイントは、URI 構文を使用して設定されます。

```
vertx:address
```

パスおよびクエリーパラメーターを使用します。

336.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
address	**必須** 通信に使用するイベントバスアドレスを設定		String

336.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
pubSub (Common)	vertx エンドポイントに送信するときに、ポイントツーポイントの代わりにパブリッシュ/サブスクライブを使用するかどうか。		Boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

名前	説明	デフォルト	タイプ
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

Camel 2.12.3: 頂点エンドポイントに送信するときに、ポイントツーポイントの代わりにパブリッシュ/サブスクライブを使用するかどうか。

> You can append query options to the URI in the following format, ?option=value&option=value&...

336.3. 既存の VERT.X インスタンスへの接続

JVM にすでに存在する Vert.x インスタンスに接続する場合は、コンポーネントレベルでインスタンスを設定できます。

```
Vertx vertx = ...;
VertxComponent vertxComponent = new VertxComponent();
vertxComponent.setVertx(vertx);
camelContext.addComponent("vertx", vertxComponent);
```

336.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第337章 VM コンポーネント

Camel バージョン 1.1 以降で利用可能

vm: コンポーネントは非同期 SEDA 動作を提供し、BlockingQueue でメッセージを交換し、別のスレッドプールでコンシューマーを呼び出します。

このコンポーネントは、VM が CamelContext インスタンス間の通信をサポートするという点で Seda コンポーネントとは異なります。そのため、このメカニズムを使用して Web アプリケーション間で通信できます (**camel-core.jar** が **system/boot** クラスパスにある場合)。

VM は Seda コンポーネントの拡張機能です。

337.1. URI 形式

```
vm:queueName[?options]
```

queueName は、JVM 内 (または少なくとも camel-core.jar をロードしたクラ出力ダー内) でエンドポイントを一意に識別する任意の文字列にすることができます。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます

プロデューサーエンドポイントとコンシューマーエンドポイントの両方に、まったく同じ VM エンドポイント URI を使用する **必要があります**。それ以外の場合、URI の **queueName** 部分が同一であっても、Camel は 2 番目の VM エンドポイントを作成します。以下に例を示します。

```
from("direct:foo").to("vm:bar?concurrentConsumers=5");

from("vm:bar?concurrentConsumers=5").to("file://output");
```

プロデューサーとコンシューマーの両方のオプションを含む完全な URI を使用する必要があることに注意してください。

Camel 2.4 ではこれが修正され、キュー名のみが一致する必要があります。キュー名 **bar** を使用すると、前の例を次のように書き換えることができます。

```
from("direct:foo").to("vm:bar");

from("vm:bar?concurrentConsumers=5").to("file://output");
```

337.2. オプション

仮想マシンコンポーネントは、以下に記載される 4 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
queueSize (advanced)	SEDA キューのデフォルトの最大容量 (つまり、保持できるメッセージの数) を設定します。		int
concurrentConsumers (consumer)	交換を処理する同時スレッドのデフォルト数を設定します。	1	int

名前	説明	デフォルト	タイプ
defaultQueueFactory (advanced)	デフォルトのキューファクトリーを設定します。		Exchange>
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

VM エンドポイントは、URI 構文を使用して設定されます。

```
vm:name
```

パスおよびクエリーパラメーターを使用します。

337.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** キューの名前		文字列

337.2.2. クエリーパラメーター (16 個のパラメーター):

名前	説明	デフォルト	タイプ
size (common)	SEDA キューの最大容量 (つまり、保持できるメッセージの数)。	2147483647	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトのエクスチェンジパターンを設定します。		ExchangePattern
limitConcurrentConsumers (consumer)	concurrentConsumers の数を最大 500 に制限するかどうか。デフォルトでは、エンドポイントがより大きな数で設定されている場合、例外が出力されます。このチェックを無効にするには、このオプションをオフにします。	true	boolean
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe メッセージングに SEDA を使用できます。つまり、メッセージを SEDA キューに送信し、各コンシューマーにメッセージのコピーを受信させることができます。有効にすると、このオプションはすべてのコンシューマーエンドポイントで指定する必要があります。	false	boolean
pollTimeout (consumer)	ポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に対応できるようになります。	1000	int
purgeWhenStopping (consumer)	コンシューマー/ルートを停止するときにタスクキューをパージするかどうか。これにより、キューに保留中のメッセージが破棄されるため、より迅速に停止できます。	false	boolean
blockWhenFull (producer)	満杯の SEDA キューにメッセージを送信するスレッドが、キューの容量がなくなるまでブロックするかどうか。デフォルトでは、キューがいっぱいであることを示す例外が出力されます。このオプションを有効にすると、呼び出しスレッドは代わりにブロックされ、メッセージが受け入れられるまで待機します。	false	boolean

名前	説明	デフォルト	タイプ
discardIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーがメッセージを破棄する (メッセージをキューに追加しない) かどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーが例外を出力して失敗するかどうか。discardIfNoConsumers オプションと failIfNoConsumers オプションのうち、同時に有効にできるのは 1 つだけです。	false	boolean
timeout (producer)	SEDA プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト (ミリ秒単位)。0 または負の値を使用して、タイムアウトを無効にすることができます。	30000	long
waitForTaskToComplete (producer)	続行する前に呼び出し元が非同期タスクの完了を待つかどうかを指定するオプション。次の 3 つのオプションがサポートされています: Always、Never、または IfReplyExpected。最初の 2 つの値は一目瞭然です。最後の値 IfReplyExpected は、メッセージが Request Reply ベースの場合にのみ待機します。デフォルトのオプションは IfReplyExpected です。	IfReplyExpected	WaitForTaskToComplete
queue (advanced)	エンドポイントで使用されるキューインスタンスを定義します。このオプションは、カスタムキューインスタンスを使用するまれなユースケースのみを対象としています。		BlockingQueue
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

同じルールが Vm コンポーネントに適用されるため、オプションやその他の重要な使用方法の詳細については、Seda コンポーネントを参照してください。

337.3. サンプル

以下のルートでは、CamelContext インスタンス間の交換を **order.email** という名前の VM キューに送信します。

```
from("direct:in").bean(MyOrderBean.class).to("vm:order.email");
```

そして、他の Camel コンテキスト (別の **.war** アプリケーションにデプロイされたものなど) で交換を受け取ります。

```
from("vm:order.email").bean(MyOrderEmailSender.class);
```

337.4. 関連項目

- Seda

第338章 WEATHER コンポーネント

Camel バージョン 2.12 以降で利用可能

weather: コンポーネントは、Open Weather Map (無料の世界の天気予報情報を提供するサイト) から天気情報をポーリングするために使用されます。情報は json String オブジェクトとして返されます。

Camel は、デフォルトで 1 時間に 1 回、現在の天気と予報の更新をポーリングします。また、プロデューサーとして使用されるエンドポイントで定義されたパラメーターに基づいて、weather API を照会するために使用することもできます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-weather</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

338.1. URI 形式

```
weather://<unused name>[?options]
```

338.2. REMARK

10 月 9 日以降、openweather サービスにアクセスするには Api キーが必要です。このキーは、appid param ! を使用して、weather エンドポイントの URI 定義にパラメーターとして渡されます。

338.3. 位置情報プロバイダー

2018 年 7 月以降、FreegeoIP は利用できなくなりました。camel-weather コンポーネントはこの API を使用していました。IPstack に切り替えるため、API を使用している場所からアクセスキーと IP を指定する必要があります。

338.4. オプション

Weather コンポーネントは、以下にリストされている 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
geolocationAccessKey (Common)	位置情報サービスを使用するには accessKey が必要になりました		String
geolocationRequestHost IP (Common)	位置情報サービスは、使用している accessKey に関連付けられた IP を指定する必要があります		String

名前	説明	デフォルト	タイプ
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Weather エンドポイントは、URI 構文を使用して設定されます。

```
weather:name
```

パスおよびクエリーパラメーターを使用します。

338.4.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** 名前の値は使用されません。		String

338.4.2. クエリーパラメーター(45 個のパラメーター):

名前	説明	デフォルト	タイプ
appid (common)	**必須** APPID API サーバーに接続しているユーザーの認証に使用される ID		String
headerName (common)	メッセージボディーではなく、このヘッダーに天気の結果を格納します。これは、現在のメッセージボディーをそのまま保持したい場合に使用できます。		String
language (common)	応答の言語。	en	WeatherLanguage
mode (common)	気象データの出力形式。	JSON	WeatherMode
period (common)	null の場合、現在の天気が返されます。それ以外の場合は、5、7、14 日の値を使用します。予測期間の数値のみが実際に解析されるため、スペルや期間の大文字化はユーザー次第です (無視されます)。		String
units (common)	温度測定の単位。		WeatherUnits

名前	説明	デフォルト	タイプ
weatherApi (common)	使用する API(潮流、予報/3 時間、日予報、駅)		WeatherApi
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
httpConnectionManager (advanced)	カスタム HttpConnectionManager を使用して接続を管理する場合		HttpConnectionManager
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int

名前	説明	デフォルト	タイプ
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit

名前	説明	デフォルト	タイプ
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
cnt (filter)	検索結果の数		Integer
ids (filter)	都市/駅の ID のリスト。複数の ID をコンマで区切ることができます。		String
lat (filter)	ロケーションの緯度。場所の代わりに緯度と経度のオプションを使用できます。ボックス化されたクエリーの場合、これは下の緯度です。		String
location (filter)	null の場合、Camel は IP アドレスの地理位置情報を使用して現在の場所を特定しようとします。それ以外の場合は、都市、国を指定します。よく知られている都市名については、Open Weather Map が最適なものを判断しますが、複数の結果が返される場合があります。したがって、国も指定すると、より正確なデータが返されます。current を場所として指定すると、コンポーネントは現在の緯度と経度を取得し、それを使用して天気の詳細を取得しようとします。場所の代わりに緯度と経度のオプションを使用できます。		String
lon (filter)	場所の経度。場所の代わりに緯度と経度のオプションを使用できます。ボックス化されたクエリーの場合、これは左経度です。		String
rightLon (filter)	ボックス化されたクエリーの場合、これは正しい経度です。topLat および zoom と組み合わせて使用する必要があります。		String
topLat (filter)	ボックス化されたクエリーの場合、これは最高許容範囲です。rightLon および zoom と組み合わせて使用する必要があります。		String
zip (filter)	郵便番号、例: 94040,us		String
zoom (filter)	ボックス化されたクエリーの場合、これはズームです。rightLon および topLat と組み合わせて使用する必要があります。		Integer
proxyAuthDomain (proxy)	プロキシー NTLM 認証用のドメイン		String
proxyAuthHost (proxy)	プロキシー NTLM 認証用のオプションのホスト		String

名前	説明	デフォルト	タイプ
proxyAuthMethod (proxy)	Basic、Digest、または NTLM のいずれかのプロキシーの認証方法。		String
proxyAuthPassword (proxy)	プロキシー認証のパスワード		String
proxyAuthUsername (proxy)	プロキシー認証用のユーザー名		String
proxyHost (proxy)	プロキシーホスト名		String
proxyPort (proxy)	プロキシーポート番号		Integer
geolocationAccessKey (security)	**必須** 位置情報サービスを使用するには accessKey が必要になりました		String
geolocationRequestHostIP (security)	**必須** 位置情報サービスは、使用している accessKey に関連付けられた IP を指定する必要があります		文字列

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

338.5. EXCHANGE データ形式

Camel はボディを json 形式の java.lang.String として配信します (上記の **mode** オプションを参照してください)。

338.6. メッセージヘッダー

ヘッダー	説明
CamelWeatherQuery	Open Weather Map サイトに送信された元のクエリー URL
CamelWeatherLocation	プロデューサーがエンドポイントの場所をオーバーライドし、代わりにこのヘッダーの場所を使用するために使用します。

338.7. サンプル

このサンプルでは、スペインのマドリッドの 7 日間の天気予報を見つけます。

```
from("weather:foo?location=Madrid,Spain&period=7
days&appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geolocationRequestHostIP=L
OCAL_IP").to("jms:queue:weather");
```

現在の場所の現在の天気を見つけるには、これを使用できます。

```
from("weather:foo?
appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geolocationRequestHostIP=LOCAL_
IP").to("jms:queue:weather");
```

プロデューサーを使用して天気を見つけるには、次のようにします。

```
from("direct:start")
  .to("weather:foo?
location=Madrid,Spain&appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geolocation
equestHostIP=LOCAL_IP");
```

そして、次のようにヘッダー付きのメッセージを送信して、任意の場所の天気を取得できます。

```
  String json = template.requestBodyAndHeader("direct:start", "", "CamelWeatherLocation",
"Paris,France&appid=APIKEY", String.class);
```

現在の場所の天気を取得するには、次のようにします。

```
  String json = template.requestBodyAndHeader("direct:start", "", "CamelWeatherLocation",
"current&appid=APIKEY", String.class);
```

第339章 JETTY WEBSOCKET コンポーネント

Camel バージョン 2.10 以降で利用可能

Websocket コンポーネントは、Websocket を使用してクライアントと通信するための Websocket エンドポイントを提供します。このコンポーネントは、IETF 仕様 (ドラフトおよび RFC 6455) を実装する Eclipse Jetty サーバーを使用します。プロトコル ws://および wss://をサポートしています。wss://プロトコルを使用するには、SSLContextParameters を定義する必要があります。

現在サポートされているバージョン

Camel 2.18 は Jetty 9 を使用します

339.1. URI 形式

```
websocket://hostname[:port][/resourceUri][?options]
```

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

339.2. WEBSOCKET オプション

Jetty Websocket コンポーネントは、以下に示す 14 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
staticResources (consumer)	静的リソース (.html ファイルなど) のリソースパスを設定します。classpath: を接頭辞として付けた場合、リソースはクラスパスからロードできます。それ以外の場合、リソースはファイルシステムまたは JAR ファイルからロードされます。たとえば、ルートクラスパスからロードするには、classpath:. または classpath:WEB-INF/static を使用します。設定されていない場合 (null など)、静的リソースは使用されていません。		String
host (Common)	ホスト名。デフォルト値は 0.0.0.0 です。	0.0.0.0	String
port (Common)	ポート番号。デフォルト値は 9292 です。	9292	Integer
sslKeyPassword (security)	SSL を使用する場合のキーストアのパスワード。		String
sslPassword (security)	SSL 使用時のパスワード。		String
sslKeystore (security)	キーストアへのパス。		String

名前	説明	デフォルト	タイプ
enableJmx (advanced)	このオプションが true の場合、Jetty JMX サポートがこのエンドポイントに対して有効になります。詳細については、Jetty JMX サポートを参照してください。	false	boolean
minThreads (advanced)	サーバースレッドプール内のスレッドの最小数の値を設定します。MaxThreads/minThreads or threadPool fields are required due to switch to Jetty9.minThreads のデフォルト値は 1 です。		Integer
maxThreads (advanced)	サーバースレッドプールのスレッドの最大数の値を設定します。MaxThreads/minThreads or threadPool fields are required due to switch to Jetty9.maxThreads のデフォルト値は 12 noCores です。		Integer
threadPool (advanced)	サーバーのカスタムスレッドプールを使用する場合。MaxThreads/minThreads or threadPool fields are required due to switch to Jetty9.		ThreadPool
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
socketFactory (common)	サブプロトコル用のカスタム WebSocketFactory を含むマップを設定するには。マップのキーはサブプロトコルです。デフォルトのキーは、デフォルトの実装用に予約されています。		Map
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Jetty Websocket エンドポイントは、URI 構文を使用して設定されます。

```
websocket:host:port/resourceUri
```

パスおよびクエリーパラメーターを使用します。

339.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	ホスト名。デフォルト値は 0.0.0.0 です。コンポーネントでこのオプションを設定すると、コンポーネントの設定値がデフォルトとして使用されます。	0.0.0.0	String
port	ポート番号。デフォルト値は 9292 です。コンポーネントでこのオプションを設定すると、コンポーネントの設定値がデフォルトとして使用されます。	9292	Integer
resourceUri	**必須** 使用する Websocket チャネルの名前		String

339.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
maxBinaryMessageSize (common)	websocketServlet によって作成された Websocket が閉じる前に受け入れることができるサイズをバイト単位で設定するために使用できます。(デフォルトは -1 または無制限)	-1	Integer
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sessionSupport (consumer)	各 http 要求に対して HttpSession を有効にするセッションサポートを有効にするかどうか。	false	boolean
staticResources (consumer)	静的リソース (.html ファイルなど) のリソースパスを設定します。classpath: を接頭辞として付けた場合、リソースはクラスパスからロードできます。それ以外の場合、リソースはファイルシステムまたは JAR ファイルからロードされます。たとえば、ルートクラスパスからロードするには、classpath:. または classpath:WEB-INF/static を使用します。設定されていない場合 (null など)、静的リソースは使用されていません。		String

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
sendTimeout (producer)	Websocket チャネルに送信する際のミリ秒単位のタイムアウト。デフォルトのタイムアウトは 30000 (30 秒) です。	30000	Integer
sendToAll (producer)	すべての websocket サブスクライバーに送信します。メッセージで WebsocketConstants.SEND_TO_ALL ヘッダーを使用する代わりに、エンドポイントレベルで設定するために使用できます。		Boolean
bufferSize (advanced)	最大フレームバイトサイズでもある websocketServlet のバッファーサイズを設定します (デフォルトは 8192)。	8192	Integer
maxIdleTime (advanced)	websocketServlet によって作成された Websocket がアイドル状態になってから閉じるまでの時間をミリ秒で設定します。(デフォルトは 300000)	300000	Integer
maxTextMessageSize (advanced)	websocketServlet によって作成された Websocket が閉じる前に受け入れることができるサイズを文字で設定するために使用できます。		Integer
minVersion (advanced)	websocketServlet で受け入れられる最小プロトコルバージョンを設定するために使用できます。(デフォルト 13 - RFC6455 バージョン)	13	Integer
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
allowedOrigins (cors)	CORS はオリジンを許可しました。すべてを許可するために使用します。		String
crossOriginFilterOn (cors)	CORS を有効にするかどうか	false	boolean

名前	説明	デフォルト	タイプ
filterPath (cors)	CORS をフィルタリングするためのコンテキストパス		String
enableJmx (monitoring)	このオプションが true の場合、Jetty JMX サポートがこのエンドポイントに対して有効になります。詳細については、Jetty JMX サポートを参照してください。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定する場合。		SSLContextParameters

339.3. メッセージヘッダー

websocket コンポーネントは、2 つのヘッダーを使用して、メッセージを単一/現在のクライアントに送り返すか、すべてのクライアントに送り返すかを示します。

WebsocketConstants.SEND_TO_ALL	**現在接続されているすべてのクライアントにメッセージを送信します。このヘッダーを使用する代わりに、エンドポイントで sendToAll オプションを使用できます。**
WebsocketConstants.CONNECTION_KEY	指定された接続キーでクライアントにメッセージを送信します。

339.4. 使用方法

この例では、クライアントが通信できる websocket サーバーを Camel に公開させます。Websocket サーバーは、デフォルトのホストとポート (**0.0.0.0:9292**) を使用します。
この例では、入力のエコーを返します。メッセージを送り返すには、変換されたメッセージを同じエンドポイント **"websocket://echo"** に送信する必要があります。これが必要です
デフォルトでは、メッセージングは InOnly であるためです。

この例は単体テストの一部であり、ここで 見つけることができます。クライアントとして、Web ソケットのサポートも提供する AHC ライブラリーを使用します。

Jetty アプリケーションサーバーが WebSocket サーブレットを登録するだけでなく、ブラウザーの Web リソースを公開できるように、webapp リソースの場所が定義されている別の例を次に示します。リソースは、webapp ディレクトリーの下で定義する必要があります。

```
from("activemq:topic:newsTopic")
  .routeId("fromJMStoWebSocket")
  .to("websocket://localhost:8443/newsTopic?sendToAll=true&staticResources=classpath:webapp");
```

339.5. WEBSOCKET コンポーネントの SSL の設定

339.5.1. JSSE 設定ユーティリティーの使用

Camel 2.10 の時点で、WebSocket コンポーネントは Camel JSSE Configuration Utility を介した SSL/TLS 設定をサポートしています。 このユーティリティーは、記述する必要があるコンポーネント固有のコードの量を大幅に削減し、エンドポイントおよびコンポーネントレベルで設定できます。 次の例は、Cometd コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
scp.setTrustManagers(tmp);

CometdComponent commetdComponent = getContext().getComponent("cometds",
CometdComponent.class);
commetdComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:keyManagers>
   <camel:trustManagers>
     <camel:keyStore
        resource="/users/home/server/keystore.jks"
        password="keystorePassword"/>
   </camel:trustManagers>
  </camel:sslContextParameters>...
...
  <to uri="websocket://127.0.0.1:8443/test?sslContextParameters=#sslContextParameters"/>...
```

エンドポイントの Java DSL ベースの設定

```
...
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            public void configure() {

                String uri = "websocket://127.0.0.1:8443/test?
sslContextParameters=#sslContextParameters";

                from(uri)
                    .log(">>> Message received from WebSocket Client : ${body}")
                    .to("mock:client")
                    .loop(10)
                        .setBody().constant(">> Welcome on board!")
                        .to(uri);
...
```

339.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド
- AHC
- Jetty
- Twitter Websocket Example は、Twitter 検索の一定のフィードをポーリングし、Web ソケットを使用してリアルタイムで結果を Web ページに公開する方法を示しています。

第340章 WORDPRESS コンポーネント

Camel バージョン 2.21 以降で利用可能

Wordpress API の Camel コンポーネント。

現在、**投稿** と **ユーザーの** 操作のみがサポートされています。

340.1. オプション

Wordpress コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	Wordpress コンポーネントの設定		Wordpress コンポーネントの設定
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Wordpress エンドポイントは、URI 構文を使用して設定されます。

```
wordpress:operationDetail
```

パスおよびクエリーパラメーターを使用します。

340.1.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
operation	**必須** エンドポイント操作。		String
操作詳細	エンドポイント操作の 2 番目の部分。wordpress:post:delete のように、エンドポイントのセマンティックが十分でない場合にのみ必要です。		String

340.1.2. クエリーパラメーター (11 パラメーター)

名前	説明	デフォルト	タイプ
apiVersion (Common)	Wordpress REST API のバージョン	2	String

名前	説明	デフォルト	タイプ
criteria (Common)	複雑な検索で使用する基準。		Map
force (Common)	ゴミ箱をバイパスして強制的に削除するかどうか。	false	Boolean
id (Common)	エンティティー ID		Integer
password (common)	許可されたユーザーからのパスワード		String
url (common)	**必須** サイトの Wordpress API URL (例: http://myblog.com/wp-json/)		String
user (Common)	書き込み操作を実行する許可ユーザー		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

読み取り操作を実行するときに必要なパラメーターのほとんどは、公式 API から反映されています。検索操作を実行するときの **criteria.** 接尾辞が必要です。次の **Consumer** を例に取ります。

```
wordpress:post?criteria.perPage=10&criteria.orderBy=author&criteria.categories=camel,dozer,json
```

340.1.3. Wordpress コンポーネントの設定

WordpressConfiguration クラスを使用して、クエリーパラメーターとして渡す代わりに、初期プロパティー設定をコンポーネントに設定できます。次のリストは、ルートで使用するコンポーネントを設定する方法を示しています。

```
public void configure() {
    final WordpressConfiguration configuration = new WordpressConfiguration();
    final WordpressComponentConfiguration component = new WordpressComponentConfiguration();
    configuration.setApiVersion("2");
    configuration.setUrl("http://yoursite.com/wp-json/");
    component.setConfiguration(configuration);
    getContext().addComponent("wordpress", component);

    from("wordpress:post?id=1")
      .to("mock:result");
}
```

340.1.4. コンシューマーの例

コンシューマーは、Wordpress のドメインオブジェクトを時々 API からポーリングします。以下は、**Post** 操作を使用した例です。

- **wordpress:post** は投稿を取得します (デフォルトは 10 件の投稿)
- **wordpress:post?id=1** 特定の投稿を検索

340.1.5. プロデューサーの例

プロデューサーは、新しいユーザーの追加や投稿の更新など、Wordpress で書き込み操作を実行します。書き込みを行うには、承認されたユーザー認証情報が必要です (認証を参照)。

- **wordpress:post** は、メッセージ本文の **org.apache.camel.component.wordpress.api.model.Post** クラスから新しい投稿を作成します。
- **wordpress:post?id=1** は、メッセージ本文のデータ **org.apache.camel.component.wordpress.api.model.Post** に基づいて投稿を更新します。
- **wordpress:post:delete?id=1** は、特定の投稿を削除する

340.2. 認証

書き込み操作 (新しい投稿の作成など) を実行するプロデューサーには、その操作を行うための 認証済みユーザーが必要です。Wordpress で使用される標準の認証メカニズムは Cookie です。残念ながら、このメソッドは nonce 内部関数に依存しているため、Wordpress 環境以外ではサポートされていません。

ナンスなしで Wordpress API を使用する代替手段がいくつかありますが、特定のプラグインのインストールが必要です。

現時点では、**camel-wordpress** は Basic 認証のみをサポートしています (さらに追加予定)。これを設定するには、Basic-Auth Wordpress プラグイン をインストールし、認証情報をエンドポイントに渡す必要があります。

from("direct:deletePost").to("wordpress:post:delete?id=9&user=ben&password=password123").to("mock:resultDelete");

本番環境で TLS なしで Basic 認証を使用することはお勧めしません!!

第341章 XCHANGE コンポーネント

Camel バージョン 2.21 以降で利用可能

xchange: コンポーネントは、XChange Java ライブラリーを使用して、60 以上の Bitcoin および Altcoin 取引所へのアクセスを提供します。取引と市場データへのアクセスのための一貫したインターフェイスが付属しています。

Camel は、暗号通貨市場データの取得、履歴データのクエリー、成行注文などを行うことができます。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xchange</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

341.1. URI 形式

```
xchange://exchange?options
```

341.2. オプション

XChange コンポーネントにはオプションがありません。

XChange エンドポイントは、URI 構文を使用して設定されます。

```
xchange:name
```

パスおよびクエリーパラメーターを使用します。

341.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
name	**必須** 接続する取引所		String

341.2.2. クエリーパラメーター (5 つのパラメーター):

名前	説明	デフォルト	タイプ
currency (producer)	通貨。		通貨

名前	説明	デフォルト	タイプ
currencyPair (producer)	通貨ペア。		CurrencyPair
method (producer)	**必須** 実行するメソッド。		XChangeMethod
service (producer)	**必須** 呼び出すサービス。		XChangeService
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

341.3. 認証

このコンポーネントは、REST API を介して、サポートされている暗号通貨交換と通信します。一部の API リクエストは、認証されていない単純な GET リクエストを使用します。ただし、興味深いもののほとんどについては、取引所のアカウントが必要で、API アクセスキーが有効になっている必要があります。

これらの API アクセスキーは厳重に保護する必要があります。特に、引き出し機能も許可している場合はそうです。その場合、あなたの API キーを入手できる人は誰でも、あなたのアカウントから別のアドレスに資金を簡単に送金できます。つまり、あなたのお金を盗むことができます。

API アクセスキーは、SSH ディレクトリー内の交換固有のプロパティーファイルに保存できます。たとえば Binance の場合、これは ~/**.ssh/binance-secret.keys** になります。

```
##
# This file MUST NEVER be commited to source control.
# It is therefore added to .gitignore.
#
apiKey = GuRW0*********
secretKey = nKLki***********
```

341.4. メッセージヘッダー

<TODO><title>サンプル</title>
このサンプルでは、現在のビットコインの市場価格を USDT で示しています。

```
from("direct:ticker").to("xchange:binance?service=market&method=ticker&currencyPair=BTC/USDT")
```

</TODO>

第342章 XML BEANS DATAFORMAT (非推奨)

Camel バージョン 1.2 以降で利用可能

XmlBeans は、XmlBeans ライブラリー を使用して XML ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを XML ペイロードにマーシャリングするデータ形式です。

```
from("activemq:My.Queue").
  unmarshal().xmlBeans().
  to("mqseries:Another.Queue");
```

342.1. オプション

XML Beans データ形式は、次に示す 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
prettyPrint	**false**	**Boolean**	適切にフォーマットされたきれいな印刷出力を有効にします。デフォルトでは false です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

342.2. 依存関係

camel ルートで XmlBeans を使用するには、このデータ形式を実装する **camel-xmlbeans** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmlbeans</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第343章 XML JSON DATAFORMAT (非推奨)

Camel バージョン 2.10 以降で利用可能

Camel は、XML および JSON 関連の変換を実行するための多くのデータ形式をすでにサポートしていますが、それらはすべて入力 (マーシャリング用) として POJO を必要とするか、出力 (アンマーシャリング用) として POJO を生成します。このデータ形式は、中間の POJO を経由せずに、XML から JSON に、またはその逆に直接変換する機能を提供します。

このデータ形式は、Json-lib ライブラリーを利用して直接変換を実現します。このコンテキストでは、XML は高レベルの形式と見なされ、JSON は低レベルの形式と見なされます。したがって、整列化/非整列化セマンティクスは次のように割り当てられます。

- マーシャリング ⇒ XML から JSON への変換
- アンマーシャリング ⇒ JSON から XML への変換

343.1. オプション

XML JSon データ形式は、以下に示す 13 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
encoding		**String**	エンコーディングを設定します。アンマーシャリングに使用されます(JSON から XML への変換)。
elementName		**String**	各配列要素を表す XML 要素の名前を指定します。アンマーシャリングに使用されます(JSON から XML への変換)。
arrayName		**String**	最上位の XML 要素の名前を指定します。アンマーシャリングに使用されます(JSON から XML への変換)。たとえば、1、2、3 を変換すると、デフォルトでは 123 として出力されます。このオプションまたは rootName を設定すると、要素 a の名前を変更できます。
forceTopLevelObject	**false**	**Boolean**	結果の JSON が、XML ルート要素と名前が一致する最上位の要素から始まるかどうかを決定します。マーシャリングに使用されます (XML から JSon 変換)。無効にすると、XML 文字列 12 は 'x: '1'、'y': '2' に変わります。それ以外の場合は、'a': 'x: '1'、'y': '2' になります。
namespaceLenient	**false**	**Boolean**	不完全な名前空間接頭辞を許容するフラグ。アンマーシャリングに使用されます(JSON から XML への変換)。ほとんどの場合、json-lib は実行時にこのフラグを処理に合わせて自動的に変更します。

名前	デフォルト	Java タイプ	説明
rootName		**String**	最上位要素の名前を指定します。アンマーシャリングに使用されます(JSON から XML への変換)。設定されていない場合、json-lib は arrayName または objectName を使用します (デフォルト値: 'o'、現時点ではこのデータ形式では設定できません)。'root' に設定すると、JSON 文字列 'x': 'value1'、'y' : 'value2' は value1value2 に変わります。それ以外の場合、'root' 要素は 'o' という名前になります。
skipWhitespace	**false**	**Boolean**	XML 要素間の空白をテキスト値と見なすか無視するかを決定します。マーシャリングに使用されます (XML から JSon 変換)。
trimSpaces	**false**	**Boolean**	文字列値から先頭と末尾の空白を省略するかどうかを決定します。マーシャリングに使用されます (XML から JSon 変換)。
skipNamespaces	**false**	**Boolean**	名前空間を無視するかどうかを通知します。デフォルトでは、xmlns 要素を使用して JSON 出力に追加されます。マーシャリングに使用されます (XML から JSon 変換)。
removeNamespacePrefixes	**false**	**Boolean**	結果の JSON 文字列に名前空間接頭辞が含まれないように、XML 修飾要素から名前空間接頭辞を削除します。マーシャリングに使用されます (XML から JSon 変換)。
expandableProperties		**List**	拡張可能なプロパティを使用すると、JSON 配列要素は、ローカル名が JSON キーと等しい反復的な XML 要素のシーケンスとして、XML　に変換されます: 123 (e は elementName の設定により変更可能) に変換されますが、number が拡張可能なプロパティーとして設定されている場合、代わりに 123 に変換されます。アンマーシャリング (JSON からXMLへの変換) のために使用されます。
typeHints		**String**	結果の XML に型ヒントを追加して、JSON への変換を支援します。アンマーシャリングに使用されます(JSON から XML への変換)。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

343.2. JAVA DSL の基本的な使い方

343.2.1. データ形式の明示的なインスタンス化

org.apache.camel.dataformat.xmljson パッケージの **XmlJsonDataFormat** をインスタンス化するだ

けです。**camel-xmljson** 機能 (OSGi で実行している場合) がインストールされていること、または **camel-xmljson-7.2.jar** とその推移的な依存関係がクラスパスに含まれていることを確認してください。デフォルト設定での初期化の例:

```
XmlJsonDataFormat xmlJsonFormat = new XmlJsonDataFormat();
```

上記のオプションに従ってデータ形式の動作を調整するには、適切なセッターを使用します。

```
XmlJsonDataFormat xmlJsonFormat = new XmlJsonDataFormat();
xmlJsonFormat.setEncoding("UTF-8");
xmlJsonFormat.setForceTopLevelObject(true);
xmlJsonFormat.setTrimSpaces(true);
xmlJsonFormat.setRootName("newRoot");
xmlJsonFormat.setSkipNamespaces(true);
xmlJsonFormat.setRemoveNamespacePrefixes(true);
xmlJsonFormat.setExpandableProperties(Arrays.asList("d", "e"));
```

データ形式をインスタンス化したら、次のステップは、**marshal()** および **unmarshal()** DSL 要素内から実際に使用することです。

```
// from XML to JSON
from("direct:marshal").marshal(xmlJsonFormat).to("mock:json");
// from JSON to XML
from("direct:unmarshal").unmarshal(xmlJsonFormat).to("mock:xml");
```

343.2.2. インラインでデータ形式を定義する

または、**xmljson()** DSL 要素を使用してインラインでデータ形式を定義できます。

```
// from XML to JSON - inline dataformat
from("direct:marshalInline").marshal().xmljson().to("mock:jsonInline");
// from JSON to XML - inline dataformat
from("direct:unmarshalInline").unmarshal().xmljson().to("mock:xmlInline");
```

必要に応じて、**Map<String, String>** をインラインメソッドに渡して、カスタムオプションを提供することもできます。

```
Map<String, String> xmlJsonOptions = new HashMap<String, String>();
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.ENCODING, "UTF-8");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.ROOT_NAME,
"newRoot");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.SKIP_NAMESPACES,
"true");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.REMOVE_NAMESPACE
_PREFIXES, "true");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.EXPANDABLE_PROPE
RTIES, "d e");

// from XML to JSON - inline dataformat w/ options
from("direct:marshalInlineOptions").marshal().xmljson(xmlJsonOptions).to("mock:jsonInlineOptions");
// form JSON to XML - inline dataformat w/ options
from("direct:unmarshalInlineOptions").unmarshal().xmljson(xmlJsonOptions).to("mock:xmlInlineOption
s");
```

343.3. SPRING または BLUEPRINT DSL での基本的な使用法

<dataFormats> ブロック内で、**xmljson** 要素を一意の ID で設定するだけです。

```
<dataFormats>
    <xmljson id="xmljson"/>
    <xmljson id="xmljsonWithOptions" forceTopLevelObject="true" trimSpaces="true"
rootName="newRoot" skipNamespaces="true"
            removeNamespacePrefixes="true" expandableProperties="d e"/>
</dataFormats>
```

次に、**<marshal />** および **<unmarshal />** DSL 内でデータ形式オブジェクトを参照するだけです。

```
<route>
    <from uri="direct:marshal"/>
    <marshal ref="xmljson"/>
    <to uri="mock:json" />
</route>

<route>
    <from uri="direct:unmarshalWithOptions"/>
    <unmarshal ref="xmljsonWithOptions"/>
    <to uri="mock:xmlWithOptions"/>
</route>
```

このコンポーネントの XML DSL オートコンプリートを有効にするのは簡単です。Spring DSL と Blueprint DSL のどちらを使用しているかに応じて、適切な スキーマの場所 を参照するだけです。このデータ形式は Camel 2.10 以降で使用できるため、そのバージョン以降のスキーマのみがこれらの新しい XML 要素と属性を含むことに注意してください。

Blueprint の構文は、Spring DSL の構文と同じです。正しい名前空間と schemaLocations が使用されていることを確認してください。

343.4. 名前空間のマッピング

XML には、要素と属性を完全に修飾する名前空間があります。JSON はそうではありません。XML-JSON 変換を実行するときは、これを考慮する必要があります。

このギャップを埋めるために、Json-lib には、アンマーシャリング (つまり、JSON から XML への変換) 中に XML 出力要素への接頭辞と名前空間 URI の形式でバインド名前空間宣言を行うオプションがあります。たとえば、次の JSON 文字列を指定します。

```
{ "pref1:a": "value1", "pref2:b": "value2" }
```

Json-lib に、要素 **pref1:a** および **pref2:b** の名前空間宣言をバインドに出力し、接頭辞 **pref1** および **pref2** を特定の名前空間 URI に出力するように問えます。

この機能を使用するには、単純に **XmlJsonDataFormat.NamespacesPerElementMapping** オブジェクトを作成し、それらを **namespaceMappings** オプション (**List**) に追加します。

XmlJsonDataFormat.NamespacesPerElementMapping は、要素名と 接頭辞 ⇒ 名前空間 URI のマップを保持します。複数の接頭辞と名前空間 URI のマッピングを容易にするために、**NamespacesPerElementMapping(String element, String pipeSeparatedMappings)** コンストラ

クターは、次の方法で [prefix, namespaceURI] ペアの文字列ベースのパイプ区切りシーケンスを取得します: **|ns2|http://camel.apache.org/personalData|ns3|http://camel.apache.org/personalData2|** .

デフォルトの名前空間を定義するには、対応するキーフィールドを空のままにします: **|ns1|http://camel.apache.org/test1||http://camel.apache.org/default|** .

名前空間宣言を要素名 = 空の文字列にバインドすると、それらの名前空間がルート要素にアタッチされます。

完全なコードは次のようになります。

```
XmlJsonDataFormat namespacesFormat = new XmlJsonDataFormat();
List<XmlJsonDataFormat.NamespacesPerElementMapping> namespaces = new
ArrayList<XmlJsonDataFormat.NamespacesPerElementMapping>();
namespaces.add(new XmlJsonDataFormat.
                    NamespacesPerElementMapping("",
"|ns1|http://camel.apache.org/test1||http://camel.apache.org/default|"));
namespaces.add(new XmlJsonDataFormat.
                    NamespacesPerElementMapping("surname",
"|ns2|http://camel.apache.org/personalData|" +
                        "ns3|http://camel.apache.org/personalData2|"));
namespacesFormat.setNamespaceMappings(namespaces);
namespacesFormat.setRootElement("person");
```

そして、Spring DSL でも同じことが実現できます。

343.4.1. 例

次の JSON 文字列で上記の Java スニペットの名前空間バインディングを使用します。

```
{ "name": "Raul", "surname": "Kripalani", "f": true, "g": null}
```

次の XML が生成されます。

```
<person xmlns="http://camel.apache.org/default" xmlns:ns1="http://camel.apache.org/test1">
   <f>true</f>
   <g null="true"/>
   <name>Raul</name>
   <surname xmlns:ns2="http://camel.apache.org/personalData"
xmlns:ns3="http://camel.apache.org/personalData2">Kripalani</surname>
</person>
```

JSON 仕様では、JSON オブジェクトが次のように定義されていることに注意してください。

> オブジェクトは、名前と値のペアの順序付けられていないセットです。[...]。

そのため、出力 XML では要素の順序が異なります。

343.5. 依存関係

camel ルートで XmlJson データ形式を使用するには、次の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmljson</artifactId>
  <version>x.x.x</version>
  <!-- Use the same version as camel-core, but remember that this component is only available from
2.10 onwards -->
</dependency>

<!-- And also XOM must be included. XOM cannot be included by default due to an incompatible
license with ASF; so add this manually -->
<dependency>
  <groupId>xom</groupId>
  <artifactId>xom</artifactId>
  <version>1.2.5</version>
</dependency>
```

343.6. 関連項目

- データ形式
- json-lib

第344章 XML SECURITY コンポーネント

Camel バージョン 2.12 以降で利用可能

この Apache Camel コンポーネントを使用すると、W3C 標準の XML 署名の構文と処理 で説明されているように、または後継 バージョン 1.1 で説明されているように、XML 署名を生成および検証できます。XML 暗号化のサポートについては、XML セキュリティー データ形式 を参照してください。

XML 署名の概要については、こちら を参照してください。コンポーネントの実装は、W3C 標準に対応する Java API である JSR 105 に基づいており、Apache Santuario と JSR 105 の JDK プロバイダーをサポートしています。実装は、最初に Apache Santuario プロバイダーの使用を試みます。Santuario プロバイダーが見つからない場合は、JDK プロバイダーが使用されます。さらに、実装は DOM ベースです。

Camel 2.15.0 以降、署名者エンドポイント用の **XAdES-BES/EPES** もサポートしています。サブセクション署名者エンドポイントの XAdES-BES/EPES を参照してください。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmlsecurity</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

344.1. XML 署名のラッピングモード

XML 署名は、enveloped、enveloping、detached XML 署名で異なります。enveloped XML 署名の場合、XML 署名は署名済み XML ドキュメントによってラップされます。これは、XML 署名要素が、署名済み XML ドキュメントに属する親要素の子要素であることを意味します。enveloping XML 署名の場合、XML 署名には署名されたコンテンツが含まれます。他のすべてのケースは、detached XML 署名と呼ばれます。2.14.0 以降、特定の形式のdetached XML 署名がサポートされています。

enveloped XML 署名 の場合、サポートされる生成された XML 署名は次の構造を持ちます (変数は **[]** で囲まれます)。

```
<[parent element]>
  ... <!-- Signature element is added as last child of the parent element-->
  <Signature Id="generated_unique_signature_id">
    <SignedInfo>
      <Reference URI="">
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to the
transforms -->
        <DigestMethod>
        <DigestValue>
      </Reference>
      (<Reference URI="#[keyinfo_Id]">
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <DigestMethod>
        <DigestValue>
      </Reference>)?
      <!-- further references possible, see option 'properties' below -->
    </SignedInfo>
```

```
    <SignatureValue>
    (<KeyInfo Id="[keyinfo_id]">)?
    <!-- Object elements possible, see option 'properties' below -->
  </Signature>
</[parent element]>
```

enveloping XML 署名 の場合、サポートされている生成された XML 署名の構造は次のとおりです。

```
<Signature Id="generated_unique_signature_id">
  <SignedInfo>
      <Reference URI="#generated_unique_object_id" type="[optional_type_value]">
          (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to the
transforms -->
          <DigestMethod>
          <DigestValue>
      </Reference>
      (<Reference URI="#[keyinfo_id]">
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          <DigestMethod>
          <DigestValue>
      </Reference>)?
       <!-- further references possible, see option 'properties' below  -->
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo Id="[keyinfo_id]">)?
  <Object Id="generated_unique_object_id"/> <!-- The Object element contains the in-message body;
the object ID can either be generated or set by the option parameter "contentObjectId" -->
  <!-- Further Object elements possible, see option 'properties' below -->
</Signature>
```

2.14.0 の時点で、次の構造を持つ **detached XML 署名** がサポートされています (サブチャプターの署名付き要素のシブリングとしての XML 署名も参照してください)。

```
(<[signed element] Id="[id_value]">
<!-- signed element must have an attribute of type ID -->
    ...

</[signed element]>
<other sibling/>*
<!-- between the signed element and the corresponding signature element, there can be other
siblings.
 Signature element is added as last sibling. -->
<Signature Id="generated_unique_ID">
  <SignedInfo>
    <CanonicalizationMethod>
    <SignatureMethod>
    <Reference URI="#[id_value]" type="[optional_type_value]">
    <!-- reference URI contains the ID attribute value of the signed element -->
        (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to the
transforms -->
        <DigestMethod>
        <DigestValue>
    </Reference>
    (<Reference URI="#[generated_keyinfo_Id]">
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

```
        <DigestMethod>
        <DigestValue>
    </Reference>)?
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo Id="[generated_keyinfo_id]">)?
</Signature>)+
```

344.2. URI 形式

camel コンポーネントは、次の URI 形式を持つ 2 個のエンドポイントで設定されます。

```
xmlsecurity:sign:name[?options]
xmlsecurity:verify:name[?options]
```

- 署名者エンドポイントを使用すると、メッセージ内の本文の XML 署名を生成できます。これは、XML ドキュメントまたはプレーンテキストのいずれかになります。enveloped、enveloping、または detached (12.14 以降) の XML 署名は、送信メッセージのボディーに設定されます。
- ベリファイアエンドポイントを使用すると、enveloped または enveloping XML 署名、またはメッセージ内のボディーに含まれるいくつかの detached (2.14.0 の時点で) XML 署名を検証できます。検証が成功すると、元のコンテンツが XML 署名から抽出され、送信メッセージのボディーに設定されます。
- ユーザーは URI の **name** 部分を選択して、camel コンテキスト内の異なる signer/verifier エンドポイントを区別できます。

344.3. 基本例

次の例は、コンポーネントの基本的な使用法を示しています。

```
from("direct:enveloping").to("xmlsecurity:sign://enveloping?keyAccessor=#accessor",
                             "xmlsecurity:verify://enveloping?keySelector=#selector",
                             "mock:result")
```

Spring XML の場合:

```
<from uri="direct:enveloping" />
    <to uri="xmlsecurity:sign://enveloping?keyAccessor=#accessor" />
    <to uri="xmlsecurity:verify://enveloping?keySelector=#selector" />
<to uri="mock:result" />
```

署名プロセスには、秘密鍵が必要です。この秘密鍵を提供するキーアクセサー Bean を指定します。検証には、対応する公開鍵が必要です。この公開鍵を提供するキーセレクター Bean を指定します。

キーアクセサー Bean は **KeyAccessor** インターフェイスを実装する必要があります。パッケージ **org.apache.camel.component.xmlsecurity.api** には、Java キーストアから秘密鍵を読み取るデフォルトの実装クラス **DefaultKeyAccessor** が含まれています。

キーセレクター Bean は、**javax.xml.crypto.KeySelector** インターフェイスを実装する必要があります。パッケージ **org.apache.camel.component.xmlsecurity.api** には、キーストアから公開鍵を読み取るデフォルトの実装クラス **DefaultKeySelector** が含まれています。

この例では、デフォルトの署名アルゴリズム **http://www.w3.org/2000/09/xmldsig#rsa-sha1** が使用されています。オプション **signatureAlgorithm** (以下を参照) によって、選択した署名アルゴリズムを設定できます。署名者エンドポイントは、**enveloping** XML 署名を作成します。**enveloped** XML 署名を作成する場合は、Signature 要素の親要素を指定する必要があります。詳細については、**parentLocalName** オプションを参照してください。

detached XML 署名の作成については、サブチャプター署名済み要素のシブリングとしての Detached XML 署名を参照してください。

344.4. コンポーネントオプション

XML Security コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
signerConfiguration (advanced)	共有 XmlSignerConfiguration 設定を使用して、エンドポイントを設定するためのベースとして使用します。		XmlSignerConfiguration
verifierConfiguration (advanced)	共有 XmlVerifierConfiguration 設定を使用して、エンドポイントを設定するためのベースとして使用します。		XmlVerifier 設定
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

344.5. エンドポイントオプション

XML Security エンドポイントは、URI 構文を使用して設定されます。

```
xmlsecurity:command:name
```

パスおよびクエリーパラメーターを使用します。

344.5.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
command	**必須** 署名するか検証するか。		XmlCommand

名前	説明	デフォルト	タイプ
name	**必須** URI の名前部分はユーザーが選択して、camel コンテキスト内の異なる signer/verifier エンドポイントを区別できます。		String

344.5.2. クエリーパラメーター(35 個のパラメーター):

名前	説明	デフォルト	タイプ
baseUri (common)	URI デリファレンスで使用するベース URI を設定できます。次に、相対 URI がベース URI と連結されます。		String
clearHeaders (common)	署名と検証後に XML 署名固有のヘッダーをクリアするかどうかを決定します。デフォルトは true です。	true	Boolean
cryptoContextProperties (common)	暗号コンテキストプロパティーを設定します。リンク XMLCryptoContextsetProperty (String、Object) を参照してください。設定可能なプロパティーは、XMLSignContext および XMLValidateContext で定義されます (サポートされるプロパティーを参照)。次のプロパティーは、デフォルトで XML 検証用の値リンク BooleanTRUE に設定されています。これらの機能をオフにする場合は、プロパティー値をリンク BooleanFALSE に設定する必要があります。 org.jcp.xml.dsig.validateManifests javax.xml.crypto.dsig.cacheReference		Map
disallowDoctypeDecl (common)	着信 XML ドキュメントに DTD DOCTYPE 宣言が含まれることを禁止します。デフォルト値はリンク BooleanTRUE です。	true	Boolean
omitXmlDeclaration (common)	送信メッセージボディーの XML 宣言を省略するかどうかを示すインジケーター。デフォルト値は false です。ヘッダーリンク XmlSignatureConstantsHEADER_OMIT_XML_DECLARATION で上書きできます。	false	Boolean
outputXmlEncoding (common)	結果の署名付き XML ドキュメントの文字エンコーディング。null の場合、元の XML ドキュメントのエンコーディングが使用されます。		String

名前	説明	デフォルト	タイプ
schemaResourceUri (common)	XML スキーマへのクラスパス。ID 属性を決定するために detached XML 署名ケースで指定する必要があり、enveloped ケースおよび enveloping ケースで設定される場合があります。設定されている場合、XML ドキュメントは指定された XML スキーマで検証されます。スキーマリソース URI は、ヘッダーリンク XmlSignatureConstantsHEADER_SCHEMA_RESOURCE_URI によって上書きできます。		文字列
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
uriDereferencer (advanced)	参照 URI によるリモートアクセスを制限したい場合は、独自の dereferencer を設定できます。オプションのパラメーター。設定されていない場合、URI フラグメント、HTTP、ファイル、および XPpointer URI を解決できるプロバイダーのデフォルトの逆参照子が使用されます。注意: 実装はプロバイダーに依存します!		URIDereferencer
addKeyInfoReference (sign)	KeyInfo 要素を改ざんから保護するために、署名された情報要素への参照を追加して、署名値によって保護されるようにすることができます。デフォルト値は true です。KeyAccessor によって KeyInfo が返される場合にのみ関連します。リンク KeyInfogetId() は null ではありません。	true	Boolean
canonicalizationMethod (sign)	ダイジェストが計算される前に SignedInfo 要素を正規化するために使用される正規化メソッド。ヘルパーメソッド XmlSignatureHelper.getCanonicalizationMethod(String algorithm) または getCanonicalizationMethod(String algorithm, List inclusiveNamespacePrefixes) を使用して、正規化メソッドを作成できます。	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	AlgorithmMethod
contentObjectId (sign)	コンテンツオブジェクト Id 属性値を設定します。デフォルトでは、UUID が生成されます。null 値を設定すると、新しい UUID が生成されます。enveloping ケースのみ使用。		String
contentReferenceType (sign)	コンテンツ参照のタイプ。デフォルト値は null です。この値は、ヘッダーリンク XmlSignatureConstantsHEADER_CONTENT_REFERENCE_TYPE で上書きできます。		文字列

名前	説明	デフォルト	タイプ
contentReferenceUri (sign)	署名するコンテンツの参照 URI。enveloped ケースのみの使用です。参照 URI に ID 属性値が含まれている場合、リソーススキーマ URI (リンク setSchemaResourceUri (String)) も設定する必要があります。これは、スキーマバリデーターが ID 属性である属性を検出するためです。enveloping ケースや detached ケースは無視されます。		String
digestAlgorithm (sign)	ダイジェストアルゴリズムの URI。オプションのパラメーター。このダイジェストアルゴリズムは、入力メッセージのダイジェストを計算するために使用されます。このダイジェストアルゴリズムが指定されていない場合、ダイジェストアルゴリズムは署名アルゴリズムから計算されます。例: http://www.w3.org/2001/04/xmlencsha256		文字列
keyAccessor (sign)	署名プロセスには、秘密鍵が必要です。この秘密鍵を提供するキーアクセサー Bean を指定します。キーアクセサー Bean は KeyAccessor インターフェイスを実装する必要があります。パッケージ org.apache.camel.component.xmlsecurity.api には、Java キーストアから秘密鍵を読み取るデフォルトの実装クラス DefaultKeyAccessor が含まれています。		KeyAccessor
parentLocalName (sign)	XML 署名要素が追加される親要素のローカル名。enveloped XML 署名にのみ関連します。または、リンク setParentXpath (XPathFilterParameterSpec) を使用することもできます。デフォルト値は null です。enveloping および detached された XML 署名の場合、値は null である必要があります。このパラメーターまたは enveloped 署名のパラメーターリンク setParentXpath(XPathFilterParameterSpec) と detached 署名のパラメーターリンク setXpathsToIdAttributes(List) を同じように設定することはできません。パラメーター parentXpath と parentLocalName が同じ設定で指定されている場合、例外が出力されます。		String
parentNamespace (sign)	XML 署名要素が追加される親要素の名前空間。		String

名前	説明	デフォルト	タイプ
parentXpath (sign)	エンベロープケースで親ノードを検索するように XPath を設定します。このメソッドを介して親ノードを指定するか、メソッドリンク setParentLocalName(String) およびリンク setParentNamespace(String) を使用して親のローカル名と名前空間を指定します。デフォルト値は null です。enveloping および detached された XML 署名の場合、値は null である必要があります。パラメーター parentXpath と parentLocalName が同じ設定で指定されている場合、例外が出力されます。		XPathFilterParameter Spec
plainText (sign)	メッセージボディーにプレーンテキストが含まれているかどうかを示します。デフォルト値は false で、メッセージボディーに XML が含まれていることを示します。値は、ヘッダーリンク XmlSignatureConstantsHEADER_MESSAGE_IS_PLAIN_TEXT で上書きできます。	false	Boolean
plainTextEncoding (sign)	プレーンテキストのエンコード。メッセージボディーがプレーンテキストの場合にのみ関連します (パラメーターリンク plainText) を参照してください。デフォルト値は UTF-8 です。	UTF-8	String
prefixForXmlSignature Namespace (sign)	XML 署名名前空間 http://www.w3.org/2000/09/xmldsig の名前空間接頭辞。デフォルト値は ds です。null または空の値が設定されている場合、XML 署名名前空間に接頭辞は使用されません。ベストプラクティスを参照してください http://www.w3.org/TR/xmldsig-bestpractices/signing-xml-	ds	String
properties (sign)	追加のプロパティーを含む XML 署名に追加の参照とオブジェクトを追加するために、XmlSignatureProperties インターフェイスを実装する Bean を提供できます。		XmlSignatureProperties
signatureAlgorithm (sign)	署名アルゴリズム。デフォルト値は http://www.w3.org/2000/09/xmldsigrsa-sha1 です。	http://www.w3.org/2000/09/xmldsig#rsa-sha1	String

名前	説明	デフォルト	タイプ
signatureId (sign)	署名 ID を設定します。このパラメーターが設定されていない場合 (null 値)、署名 ID に対して一意の ID が生成されます (デフォルト)。このパラメーターが (空の文字列) に設定されている場合、署名要素に Id 属性は作成されません。		String
transformMethods (sign)	ダイジェストが計算される前にメッセージボディーで実行される変換。デフォルトでは、C14n が追加され、enveloped 署名の場合 (parentLocalName オプションを参照)、http://www.w3.org/2000/09/xmldsigenveloped-signature もリストの位置 0 に追加されます。XmlSignatureHelper のメソッドを使用して、変換メソッドを作成します。		List
xpathsToIdAttributes (sign)	XPATH 式を介して detached ケースで署名されている要素を ID 属性 (タイプ ID の属性) に定義します。XPATH 式を介して見つかった各要素に対して、detached 署名が作成されます。その参照 URI には、対応する属性値 (先頭に '' が付きます) が含まれます。署名は、署名された要素の最後のシブリングになります。より深い階層レベルを持つ要素が最初に署名されます。ヘッダーリンク XmlSignatureConstantsHEADER_XPATHS_TO_ID_ATTRIBUTES を使用して、XPATH リストを動的に設定することもできます。enveloped 署名のパラメーターリンク setParentLocalName(String) またはリンク setParentXpath(XPathFilterParameterSpec) と、detached 署名のこのパラメーターを同じ設定で設定することはできません。		List
keySelector (verify)	XML 署名を検証するためのキーを提供します。		KeySelector
outputNodeSearch (verify)	出力メッセージ本文に設定する XML 署名ドキュメントからノードを決定するための出力ノード検索値を設定します。値のクラスは、出力ノード検索のタイプによって異なります。出力ノード検索は XmlSignature2Message に転送されます。		String
outputNodeSearchType (verify)	出力メッセージ bodyF にシリアル化される出力ノードを決定するための検索タイプを決定します。リンク setOutputNodeSearch (Object) を参照してください。サポートされている既定の検索の種類は、DefaultXmlSignature2Message にあります。	デフォルト	String

名前	説明	デフォルト	タイプ
removeSignatureElements (verify)	出力メッセージに設定されたドキュメントから XML 署名要素 (ローカル名 Signature および 名前空間 http://www.w3.org/2000/09/xmldsig を持つ要素) を削除するかどうかを示します。通常、これは XML 署名が enveloped の場合にのみ必要です。デフォルト値はリンク BooleanFALSE です。このパラメーターは XmlSignature2Message に転送されます。出力ノード検索のタイプがリンク DefaultXmlSignature2MessageOUTPUT_NODE_SEARCH_TYPE_DEFAULT.F の場合、このインジケーターは効果がありません。	false	Boolean
secureValidation (verify)	安全な検証を有効にします。true の場合、安全な検証が有効になります。	true	Boolean
validationFailedHandler (verify)	さまざまな検証失敗の状況を処理します。デフォルトの実装は、さまざまな状況に対して特定の例外を出力します (すべての例外はパッケージ名 org.apache.camel.component.xmlsecurity.api を持ち、XmlSignatureInvalidException のサブクラスです。署名値の検証に失敗すると、XmlSignatureInvalidValueException が出力されます。参照の検証に失敗すると、XmlSignatureInvalidContentHashException が出力されます。詳細については、JavaDoc を参照してください。		ValidationFailedHandler

名前	説明	デフォルト	タイプ
xmlSignature2Message (verify)	検証後に XML 署名を出力メッセージにマップする Bean。このマッピングを行う方法は、オプション outputNodeSearchType、outputNodeSearch、および removeSignatureElements によって設定できます。デフォルトの実装では、3 つの出力ノード検索タイプ Default、ElementName、および XPath に関連する 3 つの可能性が提供されます。デフォルトの実装では、ノードが決定され、それがシリアライズされて出力メッセージのボディーに設定されます。 検索タイプが ElementName の場合、出力ノード (この場合は要素でなければなりません) は、検索値で定義されたローカル名と名前空間によって決定されます (オプション outputNodeSearch 参照)。検索タイプが XPath の場合、出力ノードは検索値で指定された XPath によって決定されます (この場合、出力ノードのタイプは Element、TextNode、または Document になります)。出力ノード検索タイプがデフォルトの場合、以下のルールが適用されます: enveloped XML 署名の場合 (URI=と変換 http://www.w3.org/2000/09/xmldsigenveloped-signature を持つ参照があります)、署名要素を持たない入力 XML 文書が出力メッセージボディーに設定されます。非 enveloped XML 署名の場合、メッセージボディーは参照されたオブジェクトから決定されます。これについては、Enveloping XML 署名ケースでの出力ノードの決定の章で詳しく説明されています。		XmlSignature2Message
xmlSignatureChecker (verify)	このインターフェイスを使用すると、検証を実行する前に、アプリケーションで XML 署名を確認できます。この手順は http://www.w3.org/TR/xmldsig-bestpractices/check-what-is-signed で推奨されています		XmlSignatureChecker

344.5.3. Enveloping XML 署名ケースでの出力ノードの決定

検証後、XML 署名ドキュメントからノードが展開され、最終的に出力メッセージボディーに返されます。enveloping XML 署名の場合、**XmlSignature2Message** のデフォルト実装 **DefaultXmlSignature2Message** は、ノード検索タイプ **Default** に対して次の方法でこれを行います (オプション **xmlSignature2Message** を参照)。

- 最初にオブジェクト参照が決定されます。
  - 同じドキュメント参照のみが考慮されます (URI は # で始まる必要があります)
  - マニフェストを介したオブジェクトへの間接的な同じドキュメント参照も考慮されます。
  - オブジェクト参照の結果の数は 1 でなければなりません。

- 次に、オブジェクトが逆参照され、オブジェクトには XML 要素が 1 つだけ含まれている必要があります。この要素は、出力ノードとして返されます。

これは、enveloping XML 署名が次のいずれかの構造を持つ必要があることを意味します。

```
<Signature>
    <SignedInfo>
      <Reference URI="#object"/>
       <!-- further references possible but they must not point to an Object or Manifest containing an object reference -->
       ...
    </SignedInfo>

    <Object Id="object">
        <!-- contains one XML element which is extracted to the message body -->
    <Object>
    <!-- further object elements possible which are not referenced-->
    ...
    (<KeyInfo>)?
</Signature>
```

または構造:

```
<Signature>
    <SignedInfo>
      <Reference URI="#manifest"/>
       <!-- further references  are possible but they must not point to an Object or other manifest containing an object reference -->
       ...
    </SignedInfo>

    <Object >
      <Manifest Id="manifest">
        <Reference URI=#object/>
      </Manifest>
    </Objet>
    <Object Id="object">
       <!-- contains the DOM node which is extracted to the message body -->
    </Object>
     <!-- further object elements possible which are not referenced -->
    ...
    (<KeyInfo>)?
</Signature>
```

344.6. 署名済み要素のシブリングとしての DETACHED XML 署名

2.14.0 以降

署名が、署名された要素のシブリングである detached 署名を作成できます。次の例には、2 つの detached 署名が含まれています。最初の署名は要素 **C** 用で、2 番目の署名は要素 **A** 用です。シグネチャーは **ネストされて** います。2 番目の署名は、最初の署名も含む要素 **A** に対するものです。

Detached XML 署名の例

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<root>
   <A ID="IDforA">
      <B>
         <C ID="IDforC">
            <D>dvalue</D>
         </C>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            Id="_6bf13099-0568-4d76-8649-faf5dcb313c0">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod
                  Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
               <ds:SignatureMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
               <ds:Reference URI="#IDforC">
                  ...
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>aUDFmiG71</ds:SignatureValue>
         </ds:Signature>
      </B>
   </A>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"Id="_6b02fb8a-30df-42c6-ba25-
76eba02c8214">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
         <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
         <ds:Reference URI="#IDforA">
            ...
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>q3tvRoGgc8cMUqUSzP6C21zb7tt04riPnDuk=</ds:SignatureValue>
   </ds:Signature>
<root>
```

この例は、複数の要素に署名できること、および要素ごとに署名がシブリングとして作成されることを示しています。署名する要素には、タイプ ID の属性が必要です。属性の ID タイプは、XML スキーマで定義する必要があります (オプション **schemaResourceUri** を参照)。 タイプ ID の属性を指す XPATH 式のリストを指定します (オプション **xpathsToIdAttributes** を参照)。これらの属性は、署名される要素を決定します。要素は、**keyAccessor** Bean によって指定された同じキーによって署名されます。より高い (より深い) 階層レベルを持つ要素が最初に署名されます。この例では、要素 **C** は要素 **A** の前に署名されています。

Java DSL の例

```
from("direct:detached")
  .to("xmlsecurity:sign://detached?
keyAccessor=#keyAccessorBeant&xpathsToIdAttributes=#xpathsToIdAttributesBean&schemaResource
Uri=Test.xsd")
  .to("xmlsecurity:verify://detached?
keySelector=#keySelectorBean&schemaResourceUri=org/apache/camel/component/xmlsecurity/Test.xs
d")
  .to("mock:result");
```

Spring の例

```
<bean id="xpathsToIdAttributesBean" class="java.util.ArrayList">
    <constructor-arg type="java.util.Collection">
      <list>
        <bean
          class="org.apache.camel.component.xmlsecurity.api.XmlSignatureHelper"
          factory-method="getXpathFilter">
          <constructor-arg type="java.lang.String"
            value="/ns:root/a/@ID" />
          <constructor-arg>
            <map key-type="java.lang.String" value-type="java.lang.String">
              <entry key="ns" value="http://test" />
            </map>
          </constructor-arg>
        </bean>
      </list>
    </constructor-arg>
  </bean>
...
 <from uri="direct:detached" />
    <to
      uri="xmlsecurity:sign://detached?
keyAccessor=#keyAccessorBean&amp;xpathsToIdAttributes=#xpathsToIdAttributesBean&amp;schema
ResourceUri=Test.xsd" />
    <to
      uri="xmlsecurity:verify://detached?
keySelector=#keySelectorBean&amp;schemaResourceUri=Test.xsd" />
    <to uri="mock:result" />
```

344.7. 署名者エンドポイントの XADES-BES/EPES

Camel 2.15.0 以降で利用可能

XML Advanced Electronic Signatures (XAdES) は、XML 署名の拡張機能を定義します。この標準は、European Telecommunication Standards Institute によって定義されたもので、電子署名のコミュニティーフレームワークに関する EU 指令 (1999/93/EC) に準拠した署名を作成できます。XAdES は、署名フォームと呼ばれるさまざまな署名プロパティーのセットを定義します。署名者エンドポイントの署名形式として、**基本的な電子署名** (XAdES-BES) と **明示的なポリシーベースの電子署名** (XAdES-EPES) をサポートしています。XAdES-T および XAdES-C **検証データ付き電子** 署名フォームはサポートされていません。

XAdES-EPES フォームの次のプロパティーをサポートしています (? は、0 または 1 回の出現を示します)。

サポートされている XAdES-EPES プロパティー

```
<QualifyingProperties Target>
  <SignedProperties>
    <SignedSignatureProperties>
      (SigningTime)?
      (SigningCertificate)?
      (SignaturePolicyIdentifier)
      (SignatureProductionPlace)?
      (SignerRole)?
    </SignedSignatureProperties>
```

```
        <SignedDataObjectProperties>
          (DataObjectFormat)?
          (CommitmentTypeIndication)?
        </SignedDataObjectProperties>
      </SignedProperties>
    </QualifyingProperties>
```

XAdES-BES フォームのプロパティーは、**SignaturePolicyIdentifier** プロパティーが XAdES-BES の一部ではないことを除いて同じです。

bean **org.apache.camel.component.xmlsecurity.api.XAdESSignatureProperties** から XAdES-BES/EPES を設定できます、または **org.apache.camel.component.xmlsecurity.api.DefaultXAdESSignatureProperties.XAdESSignatureProperties** から **SigningCertificate** プロパティーを除く上述のすべてのプロパティーをサポートします。**SigningCertificate** プロパティーを取得するには、メソッド **XAdESSignatureProperties.getSigningCertificate()** または **XAdESSignatureProperties.getSigningCertificateChain()** を上書きする必要があります。 クラス **DefaultXAdESSignatureProperties** はメソッド **getSigningCertificate()** を上書きし、キーストアとエイリアスを介して署名証明書を指定できるようにします。次の例は、指定できるすべてのパラメーターを示しています。特定のパラメーターが必要ない場合は、それらを省略できます。

Java DSL での XAdES-BES/EPES の例

```
 Keystore keystore = ... // load a keystore
DefaultKeyAccessor accessor = new DefaultKeyAccessor();
accessor.setKeyStore(keystore);
accessor.setPassword("password");
accessor.setAlias("cert_alias"); // signer key alias

DefaultXAdESSignatureProperties props = new DefaultXAdESSignatureProperties();
props.setNamespace("http://uri.etsi.org/01903/v1.3.2#"); // sets the namespace for the XAdES elements; the namspace is related to the XAdES version, default value is "http://uri.etsi.org/01903/v1.3.2#", other possible values are "http://uri.etsi.org/01903/v1.1.1#" and "http://uri.etsi.org/01903/v1.2.2#"
props.setPrefix("etsi"); // sets the prefix for the XAdES elements, default value is "etsi"

// signing certificate
props.setKeystore(keystore));
props.setAlias("cert_alias"); // specify the alias of the signing certificate in the keystore = signer key alias
props.setDigestAlgorithmForSigningCertificate(DigestMethod.SHA256); // possible values for the algorithm are "http://www.w3.org/2000/09/xmldsig#sha1", "http://www.w3.org/2001/04/xmlenc#sha256", "http://www.w3.org/2001/04/xmldsig-more#sha384", "http://www.w3.org/2001/04/xmlenc#sha512", default value is "http://www.w3.org/2001/04/xmlenc#sha256"
props.setSigningCertificateURIs(Collections.singletonList("http://certuri"));

// signing time
props.setAddSigningTime(true);

// policy
props.setSignaturePolicy(XAdESSignatureProperties.SIG_POLICY_EXPLICIT_ID);
// also the values XAdESSignatureProperties.SIG_POLICY_NONE ("None"), and XAdESSignatureProperties.SIG_POLICY_IMPLIED ("Implied")are possible, default value is XAdESSignatureProperties.SIG_POLICY_EXPLICIT_ID ("ExplicitId")
// For "None" and "Implied" you must not specify any further policy parameters
```

```
props.setSigPolicyId("urn:oid:1.2.840.113549.1.9.16.6.1");
props.setSigPolicyIdQualifier("OIDAsURN"); //allowed values are empty string, "OIDAsURI",
"OIDAsURN"; default value is empty string
props.setSigPolicyIdDescription("invoice version 3.1");
props.setSignaturePolicyDigestAlgorithm(DigestMethod.SHA256);// possible values for the algorithm
are "http://www.w3.org/2000/09/xmldsig#sha1", http://www.w3.org/2001/04/xmlenc#sha256",
"http://www.w3.org/2001/04/xmldsig-more#sha384", "http://www.w3.org/2001/04/xmlenc#sha512",
default value is http://www.w3.org/2001/04/xmlenc#sha256"
props.setSignaturePolicyDigestValue("Ohixl6upD6av8N7pEvDABhEL6hM=");
// you can add  qualifiers for the signature policy either by specifying text or an XML fragment with the
root element "SigPolicyQualifier"
props.setSigPolicyQualifiers(Arrays
    .asList(new String[] {
        "<SigPolicyQualifier xmlns=\"http://uri.etsi.org/01903/v1.3.2#\">
<SPURI>http://test.com/sig.policy.pdf</SPURI><SPUserNotice><ExplicitText>display
text</ExplicitText>"
            + "</SPUserNotice></SigPolicyQualifier>", "category B" }));
props.setSigPolicyIdDocumentationReferences(Arrays.asList(new String[]
{"http://test.com/policy.doc.ref1.txt",
    "http://test.com/policy.doc.ref2.txt" }));

// production place
props.setSignatureProductionPlaceCity("Munich");
props.setSignatureProductionPlaceCountryName("Germany");
props.setSignatureProductionPlacePostalCode("80331");
props.setSignatureProductionPlaceStateOrProvince("Bavaria");

//role
// you can add claimed roles either by specifying text or an XML fragment with the root element
"ClaimedRole"
props.setSignerClaimedRoles(Arrays.asList(new String[] {"test",
    "<a:ClaimedRole xmlns:a=\"http://uri.etsi.org/01903/v1.3.2#\"><TestRole>TestRole</TestRole>
</a:ClaimedRole>" }));
props.setSignerCertifiedRoles(Collections.singletonList(new
XAdESEncapsulatedPKIData("Ahixl6upD6av8N7pEvDABhEL6hM=",
    "http://uri.etsi.org/01903/v1.2.2#DER", "IdCertifiedRole")));

// data object format
props.setDataObjectFormatDescription("invoice");
props.setDataObjectFormatMimeType("text/xml");
props.setDataObjectFormatIdentifier("urn:oid:1.2.840.113549.1.9.16.6.2");
props.setDataObjectFormatIdentifierQualifier("OIDAsURN"); //allowed values are empty string,
"OIDAsURI", "OIDAsURN"; default value is empty string
props.setDataObjectFormatIdentifierDescription("identifier desc");
props.setDataObjectFormatIdentifierDocumentationReferences(Arrays.asList(new String[] {
    "http://test.com/dataobject.format.doc.ref1.txt", "http://test.com/dataobject.format.doc.ref2.txt" }));

//commitment
props.setCommitmentTypeId("urn:oid:1.2.840.113549.1.9.16.6.4");
props.setCommitmentTypeIdQualifier("OIDAsURN"); //allowed values are empty string, "OIDAsURI",
"OIDAsURN"; default value is empty string
props.setCommitmentTypeIdDescription("description for commitment type ID");
props.setCommitmentTypeIdDocumentationReferences(Arrays.asList(new String[]
{"http://test.com/commitment.ref1.txt",
    "http://test.com/commitment.ref2.txt" }));
// you can specify a commitment type qualifier either by simple text or an XML fragment with root
```

```
element "CommitmentTypeQualifier"
props.setCommitmentTypeQualifiers(Arrays.asList(new String[] {"commitment qualifier",
    "<c:CommitmentTypeQualifier xmlns:c=\"http://uri.etsi.org/01903/v1.3.2#\"><C>c</C>
</c:CommitmentTypeQualifier>" }));

beanRegistry.bind("xmlSignatureProperties",props);
beanRegistry.bind("keyAccessorDefault",keyAccessor);

// you must reference the properties bean in the "xmlsecurity" URI
from("direct:xades").to("xmlsecurity:sign://xades?
keyAccessor=#keyAccessorDefault&properties=#xmlSignatureProperties")
            .to("mock:result");
```

Spring XML での XAdES-BES/EPES の例

```
...
<from uri="direct:xades" />
    <to
        uri="xmlsecurity:sign://xades?keyAccessor=#accessorRsa&amp;properties=#xadesProperties"
/>
    <to uri="mock:result" />
...
<bean id="xadesProperties"
    class="org.apache.camel.component.xmlsecurity.api.XAdESSignatureProperties">
    <!-- For more properties see the the previous Java DSL example.
         If you want to have a signing certificate then use the bean class
DefaultXAdESSignatureProperties (see the previous Java DSL example). -->
    <property name="signaturePolicy" value="ExplicitId" />
    <property name="sigPolicyId" value="http://www.test.com/policy.pdf" />
    <property name="sigPolicyIdDescription" value="factura" />
    <property name="signaturePolicyDigestAlgorithm"
value="http://www.w3.org/2000/09/xmldsig#sha1" />
    <property name="signaturePolicyDigestValue" value="Ohixl6upD6av8N7pEvDABhEL1hM=" />
    <property name="signerClaimedRoles" ref="signerClaimedRoles_XMLSigner" />
    <property name="dataObjectFormatDescription" value="Factura electrónica" />
    <property name="dataObjectFormatMimeType" value="text/xml" />
</bean>
<bean class="java.util.ArrayList" id="signerClaimedRoles_XMLSigner">
    <constructor-arg>
        <list>
            <value>Emisor</value>
            <value>&lt;ClaimedRole
                xmlns=&quot;http://uri.etsi.org/01903/v1.3.2#&quot;&gt;&lt;test
                xmlns=&quot;http://test.com/&quot;&gt;test&lt;/test&gt;&lt;/ClaimedRole&gt;</value>
        </list>
    </constructor-arg>
</bean>
```

344.7.1. ヘッダー

ヘッダー	タイプ	説明
CamelXmlSignatureXAdESQualifyingPropertiesId	**String**	**QualifyingProperties** 要素の Id 属性値。
CamelXmlSignatureXAdESSignedDataObjectPropertiesId	**String**	**SignedDataObjectProperties** 要素の Id 属性値。
CamelXmlSignatureXAdESSignedSignaturePropertiesId	**String**	**SignedSignatureProperties** 要素の Id 属性値。
CamelXmlSignatureXAdESDataObjectFormatEncoding	**String**	**DataObjectFormat** 要素の Encoding 要素の値。
CamelXmlSignatureXAdESNamespace	**String**	XAdES 名前空間パラメーター値を上書きします。
CamelXmlSignatureXAdESPrefix	**String**	XAdES 接頭辞 パラメーター値を上書きします。

344.7.2. XAdES バージョン 1.4.2 に関する制限事項

- 署名フォーム XAdES-T および XAdES-C はサポートされていません。
- signer 部分のみ実装されています。Verifier パーツは現在利用できません。
- **QualifyingPropertiesReference** 要素はサポートされていません (仕様のセクション 6.3.2 を参照)。
- **SignaturePolicyIdentifier element** に含まれる **SignaturePolicyId** 要素を含有する **Transforms** 要素はサポートされていません。
- **CounterSignature** 要素はサポートされていません → **UnsignedProperties** 要素はサポートされていません。
- 最大 1 つの **DataObjectFormat** 要素。署名されたデータオブジェクトは 1 つしかないため (これは、XML signer エンドポイントへの受信メッセージボディーです)、複数の **DataObjectFormat** 要素は意味がありません。

- 最大で1つの **CommitmentTypeIndication** 要素。署名されたデータオブジェクトは1つしかないため、複数の **CommitmentTypeIndication** 要素は意味がありません (これは、XML 署名者エンドポイントへの入力メッセージボディーです)。
- **CommitmentTypeIndication** 要素には、常に **AllSignedDataObjects** 要素が含まれます。**CommitmentTypeIndication** 要素内の **ObjectReference** 要素はサポートされていません。
- **AllDataObjectsTimeStamp** 要素はサポートされていません
- **IndividualDataObjectsTimeStamp** 要素はサポートされていません

344.8. 関連項目

- ベストプラクティス

第345章 XMPP コンポーネント

Camel バージョン 1.0 以降で利用可能

xmpp: コンポーネントは、XMPP (Jabber) トランスポートを実装します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmpp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

345.1. URI 形式

```
xmpp://[login@]hostname[:port][/participant][?Options]
```

このコンポーネントは、ルームベースの会話と個人的な会話の両方をサポートしています。
このコンポーネントはプロデューサーとコンシューマーの両方をサポートします (XMPP からメッセージを取得したり、XMPP にメッセージを送信したりできます)。コンシューマーモードはルームの開始をサポートします。

URI には、**?option=value&option=value&...** の形式でクエリーオプションを追加できます。

345.2. オプション

XMPP コンポーネントにはオプションがありません。

XMPP エンドポイントは、URI 構文を使用して設定されます。

```
xmpp:host:port/participant
```

パスおよびクエリーパラメーターを使用します。

345.2.1. パスパラメーター (3 パラメーター):

名前	説明	デフォルト	タイプ
host	**必須** チャットサーバーのホスト名。		String
port	**必須** チャットサーバーのポート番号。		int
participant	メッセージを受信する人の JID (Jabber ID)。room パラメーターは参加者よりも優先されます。		String

345.2.2. クエリーパラメーター (18 パラメーター)

名前	説明	デフォルト	タイプ
login (Common)	ユーザーをログインするかどうか。	true	boolean
nickname (common)	ルームに参加するときはニックネームを使用してください。room が指定され、ニックネームが指定されていない場合、user がニックネームとして使用されます。		String
pubsub (Common)	入力時に pubsub パケットを受け入れます。デフォルトは false です	false	boolean
room (Common)	このオプションを指定すると、コンポーネントは MUC (Multi User Chat) に接続します。通常、MUC のドメイン名はログインドメインとは異なります。たとえば、supermanjabber.org でクリプトンルームに参加したい場合、ルーム URL は kryptonconference.jabber.org です。会議の部分に注意してください。部屋全体の JID を提供する必要はありません。room パラメーターにシンボルが含まれていない場合、ドメイン部分は Camel によって検出され、追加されます。		String
serviceName (Common)	接続しているサービスの名前。Google トークの場合、これは gmail.com になります。		String
testConnectionOnStartup (common)	起動時に接続をテストするかどうかを指定します。これは、ルートの開始時に XMPP クライアントが XMPP サーバーへの有効な接続を確保するために使用されます。接続を確立できない場合、Camel は起動時に例外を出力します。このオプションが false に設定されている場合、Camel はプロデューサーが必要とするときに遅延接続を確立しようとし、接続が確立されるまでコンシューマー接続をポーリングします。デフォルトは true です。	true	boolean
createAccount (Common)	true の場合、アカウントの作成が試行されます。デフォルトは false です。	false	boolean
resource (Common)	XMPP リソース。デフォルトは Camel です。	Camel	String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

名前	説明	デフォルト	タイプ
connectionPollDelay (consumer)	XMPP 接続の正常性を確認するためのポーリング (秒単位) 間の時間、または最初のコンシューマー接続を確立するための試行間の時間 (秒単位)。接続が非アクティブになった場合、Camel は接続を再確立しようとします。デフォルトは 10 秒です。	10	int
doc (consumer)	入力パケットの Document フォームを含む IN メッセージに doc ヘッダーを設定します。presence または pubsub が true の場合、デフォルトは true、それ以外の場合は false です。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
connectionConfig (advanced)	既存の接続設定を使用するには。現在、org.jivesoftware.smack.tcp.XMPPTCPConnectionConfiguration のみがサポートされています (XMPP over TCP)。		ConnectionConfiguration
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
headerFilterStrategy (filter)	カスタムの HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルタリングします。		HeaderFilterStrategy
password (security)	ログイン用パスワード		String
ユーザー (security)	ユーザー名 (サーバー名なし)。指定しない場合、匿名ログインが試行されます。		文字列

345.3. ヘッダーとサブジェクトまたは言語の設定

Camel は、メッセージの IN ヘッダーを XMPP メッセージのプロパティーとして設定します。ヘッダーのカスタムフィルタリングが必要な場合は、**HeaderFilterStategy** を設定できます。XMPP メッセージの **Subject** と **Language** も IN ヘッダーとして提供されている場合は設定されます。

345.4. 例

ユーザー **superman** が **jabber** サーバーのルーム **krypton** にパスワード **secret** で参加するには:

```
xmpp://superman@jabber.org/?room=krypton@conference.jabber.org&password=secret
```

joker にメッセージを送信するためのユーザー **superman**:

```
xmpp://superman@jabber.org/joker@jabber.org?password=secret
```

Java でのルーティングの例:

```
from("timer://kickoff?period=10000").
setBody(constant("I will win!\n Your Superman.")).
to("xmpp://superman@jabber.org/joker@jabber.org?password=secret");
```

joker からのすべてのメッセージをキューの **illegal.talk** に書き込むコンシューマー設定。

```
from("xmpp://superman@jabber.org/joker@jabber.org?password=secret").
to("activemq:evil.talk");
```

ルームメッセージをリッスンするコンシューマー設定:

```
from("xmpp://superman@jabber.org/?password=secret&room=krypton@conference.jabber.org").
to("activemq:krypton.talk");
```

ルームの略記 (ドメイン部分なし):

```
from("xmpp://superman@jabber.org/?password=secret&room=krypton").
to("activemq:krypton.talk");
```

Google Chat サービスに接続するときは、**serviceName** と認証情報を指定する必要があります。

```
from("direct:start").
  to("xmpp://talk.google.com:5222/touser@gmail.com?
serviceName=gmail.com&user=fromuser&password=secret").
  to("mock:result");
```

345.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第346章 XPATH 言語

Camel バージョン 1.1 以降で利用可能

Camel は XPath をサポートして、式または述語を DSL または Xml 設定 で使用できるようにします。たとえば、XPath を使用して、メッセージフィルター で述語を作成したり受信者リストの式として使用したりできます。

ストリーム

メッセージボディーがストリームベースの場合、受信した入力はストリームとして Camel に送信されます。つまり、ストリームのコンテンツを **一度** だけ読み取ることができます。そのため、XPath を Message Filter またはコンテンツベースのルーターとして使用する場合に、データに複数回アクセスする必要があることが頻繁にあります。そのため、ストリームキャッシングを使用するか、メッセージボディーを安全に複数回再読み取りできる **String** に変換する必要があります。

```
from("queue:foo").
  filter().xpath("//foo")).
  to("queue:bar")
```

```
from("queue:foo").
  choice().xpath("//foo")).to("queue:bar").
  otherwise().to("queue:others");
```

346.1. XPATH 言語オプション

XPath 言語は、以下に示す 9 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
documentType		**String**	ドキュメントタイプのクラスの名前。デフォルト値は org.w3c.dom.Document です
resultType	**NODESET**	**String**	結果タイプ (出力からのタイプ) のクラス名を設定します。デフォルトの結果タイプは NodeSet です
saxon	**false**	**Boolean**	Saxon を使用するかどうか。
factoryRef		**String**	レジストリーを検索するためのカスタム XPathFactory を参照します
objectModel		**String**	使用する XPath オブジェクトモデル
logNamespaces	**false**	**Boolean**	トラブルシューティング時に役立つ名前空間をログに記録するかどうか
headerName		**String**	メッセージボディーの代わりに入力として使用するヘッダーの名前

名前	デフォルト	Java タイプ	説明
threadSafety	**false**	**Boolean**	xpath 式の返された結果に対してスレッドセーフを有効にするかどうか。これは、NODESET を結果の型として使用し、返されたセットに複数の要素がある場合に適用されます。この状況で、並列処理モードの Camel Splitter EIP などから NODESET を同時に処理すると、スレッドセーフの問題が発生する可能性があります。このオプションは、ノードの防衛的コピーを実行して、同時実行の問題を防ぎます。アプリケーションで camel-saxon または Saxon を使用している場合は、このオプションをオンにすることをお勧めします。Saxon には、このオプションをオンにすることで防止できるスレッドセーフの問題があります。
trim	**true**	**Boolean**	値をトリミングして、先頭および末尾の空白と改行を削除するかどうか

346.2. NAMESPACES

Namespaces ヘルパークラスを使用すると、XPath 式で名前空間を簡単に使用できます。

346.3. VARIABLES

XPath の変数は、異なる名前空間で定義されています。デフォルトの名前空間は **http://camel.apache.org/schema/spring** です。

名前空間 URI	ローカルの部分	タイプ	説明
http://camel.apache.org/xml/in/	in	Message	exchange.in メッセージ
http://camel.apache.org/xml/out/	out	Message	exchange.out メッセージ
http://camel.apache.org/xml/function/	関数	Object	**Camel 2.5:** 追加機能

名前空間 URI	ローカルの部分	タイプ	説明
http://camel.apache.org/xml/variables/environment-variables	env	Object	OS 環境変数
http://camel.apache.org/xml/variables/system-properties	system	Object	Java System プロパティー
http://camel.apache.org/xml/variables/exchange-property		Object	Exchange プロパティー

Camel は次のいずれかに従って変数を解決します。

- 名前空間が指定されている場合
- 名前空間が指定されていない場合

346.3.1. 名前空間が指定されている場合

名前空間が指定されている場合には、Camel には返す内容が正確に指示されます。ただし、**in** または **out** のいずれかで解決する場合、Camel は最初に指定されたローカル部分でヘッダーを解決しようとし、それを返します。ローカル部分の値が **body** の場合には、代わりに body が返されます。

346.3.2. 名前空間が指定されていない場合

名前空間が指定されていない場合、Camel はローカル部分のみに基づいて解決します。Camel は、次の手順で変数を解決しようとします。

- **variable(name, value)** Fulent Builder を使用して設定された **変数** から
- キーが指定されたヘッダーがある場合の message.in.header から
- キーが指定されたプロパティーがある場合の exchange.properties から

346.4. 関数

Camel は、交換へのアクセスに使用できる次の XPath 関数を追加します。

機能	引数	タイプ	説明
in:body	none	Object	**in** メッセージボディーを返します。
in:header	ヘッダー名	Object	**in** メッセージヘッダーを返します。
out:body	none	Object	**out** メッセージボディーを返します。
out:header	ヘッダー名	Object	**out** メッセージヘッダーを返します。
function:properties	プロパティーの鍵	String	**Camel 2.5:** Properties コンポーネント (プロパティープレースホルダー) を使用してプロパティーを検索します。
function:simple	Simple 式	Object	**Camel 2.5:** Simple 式を評価します。

注意

function:properties および **function:simple** は、Splitter EIP で使用する場合など、戻り値の型が **NodeSet** の場合はサポートされません。

これらの関数の使用例を次に示します。

Camel 2.5 で導入された新機能:

346.5. XML 設定の使用

Spring XML ファイルでルートを設定する場合は、次のように XPath 式を使用できます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
```

```
  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring"
xmlns:foo="http://example.com/person">
    <route>
      <from uri="activemq:MyQueue"/>
      <filter>
        <xpath>/foo:person[@name='James']</xpath>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

名前空間接頭辞 (この場合は **foo**) を XPath 式で再利用して、名前空間ベースの XPath 式をより簡単にする方法に注目してください。

xpath で独自の名前空間を使用する方法については、メーリングリストのこのディスカッション も参照してください。

346.6. 結果タイプの設定

XPath 式は、**org.w3c.dom.NodeList** などのネイティブ XML オブジェクトを使用して結果の型を返します。ただし、多くの場合、結果の型を String にすることがあります。これを行うには、使用する結果の型を XPath に指示する必要があります。

Java DSL で:

```
xpath("/foo:person/@id", String.class)
```

Spring DSL では、**resultType** 属性を使用して完全修飾クラス名を提供します。

```
<xpath resultType="java.lang.String">/foo:person/@id</xpath>
```

@XPath:
Camel 2.1 以降で利用可能

```
@XPath(value = "concat('foo-',//order/name/)", resultType = String.class) String name)
```

xpath 関数 concat を使用して、注文名の前に **foo-** を付けます。この場合、結果の型として String が必要であることを指定する必要があるため、concat 関数が機能します。

346.7. ヘッダーでの XPATH の使用

Camel 2.11 から利用可能

一部のユーザーは、XML をヘッダーに格納している場合があります。XPath をヘッダーの値に適用するには、'headerName' 属性を定義することで対応できます。

また、Java DSL では、次に示すように、headerName を 2 番目のパラメーターとして指定します。

```
  xpath("/invoice/@orderType = 'premium'", "invoiceDetails")
```

346.8. 例

Message Filter で XPath 式を述語として使用する簡単な例を次に示します

作業したい名前空間の標準セットがあり、それらをさまざまな XPath 式で共有したい場合は、この例に示すように NamespaceBuilder を使用できます。

このサンプルには、choice コンストラクトがあります。最初の選択肢は、メッセージが値 **Camel** を持つヘッダーキー **タイプ** を持っているかどうかを評価します。
2 番目の選択肢は、メッセージボディーに値が **Kong** である名前タグ **<name>** があるかどうかを評価します。
どちらも真でない場合、メッセージはそれ以外のブロックでルーティングされます。

そして、ルートに相当する Spring XML:

346.9. XPATH の注入

Bean インテグレーションを使用して Bean のメソッドを呼び出す場合、@XPath (他の 言語も使用可) を使用してメッセージから値を展開し、メソッドパラメーターにバインドすることができます。

デフォルトの @XPath アノテーションには、SOAP および XML 名前空間が使用可能です。XPath 式で独自の名前空間 URI を使用する場合は、XPath アノテーション の独自のコピーを使用して、使用したい名前空間接頭辞を作成できます。

つまり、上部のコードを別のパッケージおよび/またはアノテーション名で独自のプロジェクトにカットアンドペーストし、メソッドパラメーターでアノテーションを使用するときにスコープ内に必要な名前空間接頭辞/URI を追加します。次に、メソッドパラメーターでアノテーションを使用すると、必要なすべての名前空間が XPath 式で使用できるようになります。

以下に例を示します。

```
public class Foo {

    @MessageDriven(uri = "activemq:my.queue")
    public void doSomething(@MyXPath("/ns1:foo/ns2:bar/text()") String correlationID, @Body String
body) {
        // process the inbound message here
    }
}
```

346.10. EXCHANGE なしで XPATHBUILDER を使用する

Camel 2.3 の時点で利用可能

Exchange を必要とせずに **org.apache.camel.builder.XPathBuilder** を使用できるようになりました。これは、カスタム xpath 評価を行うためのヘルパーとして使用する場合に便利です。

XPathBuilder 内の可動部分の多くは Camel Type Converter へのアクセスを必要とするため、CamelContext を渡す必要があり、これが CamelContext が必要となる理由です。

たとえば、次のようなことができます。

```
boolean matches = XPathBuilder.xpath("/foo/bar/@xyz").matches(context, "<foo><bar xyz='cheese'/>
</foo>"));
```

これは、指定された述語に一致します。

たとえば、次の 3 つの例に示すように評価することもできます。

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar></foo>",
String.class);
Integer number = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>123</bar></foo>",
Integer.class);
Boolean bool = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>true</bar></foo>",
Boolean.class);
```

String の結果で評価することは一般的な要件であるため、もう少し簡単に行うことができます。

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar></foo>");
```

346.11. XPATHBUILDER での SAXON の使用

Camel 2.3 の時点で利用可能

プロジェクトへの依存関係として **camel-saxon** を追加する必要があります。

XPathBuilder で Saxon を使用する方が簡単になりました。以下に示すように、いくつかの方法で実行できます。
後者は最も簡単なものです。

- ファクトリーを利用する
- ObjectModel の使用

簡単なもの

346.12. システムプロパティーを使用したカスタム XPATHFACTORY の設定

Camel 2.3 の時点で利用可能

Camel は、使用するカスタム XPathFactory を設定するために使用できる JVM システムプロパティー **javax.xml.xpath.XPathFactory** の読み取りをサポートするようになりました。

この単体テストは、代わりに Saxon を使用するために実行する内容を説明しています。

次のようなデフォルト以外の XPathFactory を使用する場合、Camel は **INFO** レベルでログを記録します。

```
XPathBuilder  INFO  Using system property
javax.xml.xpath.XPathFactory:http://saxon.sf.net/jaxp/xpath/om with value:
                net.sf.saxon.xpath.XPathFactoryImpl when creating XPathFactory
```

Apache Xerces を使用するには、システムプロパティーを設定できます。

```
-Djavax.xml.xpath.XPathFactory=org.apache.xpath.jaxp.XPathFactoryImpl
```

346.13. SPRING DSL から SAXON を有効にする

Camel 2.10 以降で利用可能

Java DSL と同様に、Spring DSL から Saxon を有効にするには、次の 3 つのオプションがあります。

ファクトリーの指定

```
<xpath factoryRef="saxonFactory" resultType="java.lang.String">current-dateTime()</xpath>
```

オブジェクトモデルの指定

```
<xpath objectModel="http://saxon.sf.net/jaxp/xpath/om" resultType="java.lang.String">current-dateTime()</xpath>
```

ショートカット

```
<xpath saxon="true" resultType="java.lang.String">current-dateTime()</xpath>
```

346.14. デバッグを支援する名前空間の監査

Camel 2.10 以降で利用可能

ユーザーが頻繁に直面する XPath 関連の問題の多くは、名前空間の使用に関連しています。メッセージに存在する名前空間と、XPath 式が認識または参照している名前空間との間に、多少のずれがある場合があります。XPath の述語または式において、名前空間の問題が原因で XML 要素と属性を見つけることができない場合は、実際には名前空間の定義が不足しているだけにも関わらず、機能していないように見える場合があります。

XML の名前空間は必ず必要で、ある操作を実装して名前空間を自動的に接続することで使用方法を簡素化することができるといいのですが、実際は、このような道筋をたどったアクションは、標準に反してしまい、相互運用性が確保しにくくなります。

したがって、XPath 式言語に 2 つの新機能を追加して、述語と式の両方からアクセスできるようして、このような問題のデバッグを支援することがこちらで最大限提供できる内容です。

#=== XPath 式/述語の名前空間コンテキストのロギング

内部プールで新しい XPath 式が作成されるたびに、Camel は式の名前空間コンテキストを **org.apache.camel.builder.xml.XPathBuilder** ロガーに記録します。Camel は Namespace Context を階層的な方法 (親子関係) で表すため、ツリー全体が次の形式で再帰的に出力されます。

```
[me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}]]]
```

これらのオプションのいずれかを使用して、このログを有効にできます。

1. **org.apache.camel.builder.xml.XPathBuilder** ロガー、または **org.apache.camel** やルートロガーなどの親ロガーで TRACE ロギングを有効にします。
2. Auditing Namespaces に示されているように **logNamespaces** オプションを有効にします。この場合、ログは INFO レベルで発生します。

346.15. 名前空間の監査

Camel は、XPath 式を評価する前に、すべての受信メッセージに存在する全名前空間を検出してダンプすることができ、可能性のある名前空間の問題を分析して特定するために必要な豊富な情報をすべて提供します。

これを実現するために、別の特別に調整された XPath 式を内部的に使用して、メッセージに表示されるすべての名前空間マッピングを抽出し、個々のマッピングごとに接頭辞と完全な名前空間 URI を表示します。

考慮すべき点:

- 暗黙的な XML 名前空間 (xmlns:xml="http://www.w3.org/XML/1998/namespace") は値を追加しないため、出力から除外されます。
- デフォルトの名前空間は、出力の DEFAULT キーワードの下に一覧表示されます
- 名前空間は異なるスコープで再マッピングできることに注意してください。最上位の a 接頭辞を考えてみてください。これは、内部要素で別の名前空間または内部スコープで変わるデフォルトの名前空間を割り当てることができます。検出された接頭辞ごとに、関連付けられているすべての URI が一覧表示されます。

このオプションは、Java DSL および Spring DSL で有効にできます。

Java DSL の場合

```
XPathBuilder.xpath("/foo:person/@id", String.class).logNamespaces()
```

Spring DSL:

```
<xpath logNamespaces="true" resultType="String">/foo:person/@id</xpath>
```

監査の結果は、**org.apache.camel.builder.xml.XPathBuilder** ロガーの下の INFO レベルに表示され、次のようになります。

```
2012-01-16 13:23:45,878 [stSaxonWithFlag] INFO  XPathBuilder  - Namespaces discovered in
message:
{xmlns:a=[http://apache.org/camel], DEFAULT=[http://apache.org/default],
xmlns:b=[http://apache.org/camelA, http://apache.org/camelB]}
```

346.16. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").xpath("resource:classpath:myxpath.txt", String.class)
```

346.17. 依存関係

XPath 言語は camel-core の一部です。

第347章 XQUERY コンポーネント

Camel バージョン 1.0 以降で利用可能

Camel は XQuery をサポートして、式または述語を DSL または Xml 設定 で使用できるようにします。たとえば、XQuery を使用して メッセージフィルター の述語を作成したり、受信者リストの式として使用したりできます。

347.1. オプション

XQuery コンポーネントは、以下に示す 4 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
moduleURIResolver (advanced)	カスタム ModuleURIResolver を使用します		ModuleURIResolver
configuration (advanced)	カスタム Saxon 設定を使用します		設定
configurationProperties (advanced)	カスタムの Saxon 設定プロパティーを設定します		Map
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

XQuery エンドポイントは、URI 構文を使用して設定されます。

```
xquery:resourceUri
```

パスおよびクエリーパラメーターを使用します。

347.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	**必須** クラスパスまたはファイルシステムからロードするテンプレートの名前。		String

347.1.2. Query Parameters (31 parameters):

名前	説明	デフォルト	タイプ
allowStAX (common)	StAX モードの使用を許可するかどうか。	false	boolean
headerName (common)	メッセージ本文の代わりに Camel メッセージヘッダーを入力ソースとして使用します。		String
namespacePrefixes (common)	名前空間マッピングのセットに使用する名前空間接頭辞を制御できます。		Map
resultsFormat (common)	どの出力結果を使用するか。	DOM	ResultFormat
resultType (common)	クラスとして定義されたどの出力結果を使用するか。		Class<?>
stripsAllWhiteSpace (common)	すべての空白を削除するかどうか。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

名前	説明	デフォルト	タイプ
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
configuration (advanced)	カスタム Saxon 設定を使用します		設定
configurationProperties (advanced)	カスタムの Saxon 設定プロパティーを設定します		Map
moduleURIResolver (advanced)	カスタム ModuleURIResolver を使用します		ModuleURIResolver
parameters (advanced)	追加のパラメーター		Map
properties (advanced)	シリアル化パラメーターを設定するためのプロパティー。		Properties
staticQueryContext (advanced)	カスタム Saxon StaticQueryContext を使用する場合。		StaticQueryContext
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
delay (scheduler)	次のポーリングまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	500	long

名前	説明	デフォルト	タイプ
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

347.2. 例

```
from("queue:foo").filter().
  xquery("//foo").
  to("queue:bar")
```

クエリー内で関数を使用することもできます。その場合、Class を 2 番目の引数として **xquery()** メソッドに渡すことにより、明示的な型変換が必要になります (そうしないと、org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR が発生します)。

```
from("direct:start").
  recipientList().xquery("concat('mock:foo.', /person/@city)", String.class);
```

347.3. VARIABLES

IN メッセージボディーは **contextItem** として設定されます。これに加えて、これらの変数もパラメーターとして追加されます。

変数	タイプ	説明
exchange	Exchange	現在のエクスチェンジ
in.body	Object	In メッセージのボディー
out.body	Object	OUT メッセージのボディー (存在する場合)
in.headers.*	Object	名前が in.headers.foo の変数を使用して、キー **foo** で exchange.in.headers の値を使用できます。
out.headers.*	Object	名前が out.headers.foo 変数である変数を使用して、キー **foo** で exchange.out.headers の値を使用できます。
キー名	Object	exchange.properties と exchange.in.headers および **setParameters (Map)** を使用して設定された追加のパラメーター。これらのパラメーターは、独自のキー名で追加されます。たとえば、キー名が **foo** の IN ヘッダーがある場合、**foo** として追加されます。

347.4. XML 設定の使用

Spring XML ファイルでルートを設定する場合は、次のように XPath 式を使用できます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:foo="http://example.com/person"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
      <from uri="activemq:MyQueue"/>
      <filter>
        <xquery>/foo:person[@name='James']</xquery>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
```

```
    </route>
  </camelContext>
</beans>
```

名前空間接頭辞 (この場合は **foo**) を XPath 式で再利用して、名前空間ベースの XQuery 式をより簡単にする方法に注目してください。

XQuery 式で関数を使用する場合は、**@type** 属性を介して xml 設定で行われる明示的な型変換が必要です。

```
    <xquery type="java.lang.String">concat('mock:foo.', /person/@city)</xquery>
```

347.5. XQUERY の変換としての使用

以下に示すように、ルートで transform または setBody を使用してメッセージの変換を行うことができます。

```
from("direct:start").
  transform().xquery("/people/person");
```

xquery はデフォルトで DOMResult を使用することに注意してください。したがって、person ノードの値を取得する場合は、text() を使用して、次のように結果の型として String を使用するよう xquery に指示する必要があります。

```
from("direct:start").
  transform().xquery("/people/person/text()", String.class);
```

347.6. XQUERY をエンドポイントとして使用する

XQuery 式が非常に大きくなる場合があります。本質的にテンプレート化に使用できます。したがって、XQuery テンプレートを使用してルーティングできるように、XQuery エンドポイントを使用することができます。

次の例は、ActiveMQ キュー (MyQueue) のメッセージを取得し、XQuery を使用して変換し、MQSeries に送信する方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:MyQueue"/>
    <to uri="xquery:com/acme/someTransform.xquery"/>
    <to uri="mqseries:SomeOtherQueue"/>
  </route>
</camelContext>
```

現在、XQuery のカスタム関数によって NullPointerException が発生する場合があります (Camel 2.18、2.19、および 2.20)。これは、Camel 2.21 で修正される予定です。

347.7. 例

Message Filter で XQuery 式を述語として使用する簡単な例を次に示します。

この 例では、 メッセージフィルターの述語として名前空間を使用して XQuery を使用します。

347.8. XQUERY の学習

XQuery は非常に強力な言語で、XML の照会、検索、ソート、および結果としての出力が可能です。XQuery については、以下のチュートリアルを試してください。

- Mike Kay の XQuery 入門
- W3Schools XQuery チュートリアル

XQuery 関数のリファレンス も役立つ場合があります。

347.9. 外部リソースからスクリプトを読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、**"classpath:"**、**"file:"**、または **"http:"** などのリソースから Camel に読み込むことができます。
これは、**"resource:scheme:location"** の構文を使用して行われます。たとえば、クラスパス上のファイルを参照するには、以下を実行します。

```
.setHeader("myHeader").xquery("resource:classpath:myxquery.txt", String.class)
```

347.10. 依存関係

camel ルートで XQuery を使用するには、XQuery 言語を実装する **camel-saxon** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-saxon</artifactId>
  <version>x.x.x</version>
</dependency>
```

第348章 XSLT コンポーネント

Camel バージョン 1.3 以降で利用可能

xslt: コンポーネントを使用すると、XSLT テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストに対するレスポンスを生成する場合に理想的です。

348.1. URI 形式

```
xslt:templateName[?options]
```

URI 形式には、次のいずれかの **templateName** が含まれます。

- 呼び出すテンプレートのクラスパ出力ローカル URI
- リモートテンプレートの完全な URL。

URI には、次の形式でクエリーオプションを追加できます。

?option=value&option=value&...

URI 構文の詳細については、Spring のドキュメント を参照してください。

表348.1 URI の例

URI	説明
xslt:com/acme/mytransform.xsl	クラスパス上のファイル com/acme/mytransform.xsl を参照します
xslt:file:///foo/bar.xsl	ファイル /foo/bar.xsl を参照します
xslt:http://acme.com/cheese/foo.xsl	リモート http リソースを参照します

Camel 2.8 以前の場合、Maven ユーザーは、このコンポーネントの **pom.xml** に次の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel 2.9 以降、XSLT コンポーネントは camel-core で直接提供されます。

348.2. オプション

XSLT コンポーネントは、以下に示す 9 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
xmlConverter (advanced)	org.apache.camel.converter.jaxp.XmlConverter のカスタム実装を使用する場合。		XmlConverter
uriResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタム UriResolver を使用するには。オプション uriResolver と一緒に使用しないでください。		XsltUriResolverFactory
uriResolver (advanced)	カスタム UriResolver を使用するには。オプション uriResolverFactory と一緒に使用しないでください。		URIResolver
contentCache (producer)	ロード時のリソースコンテンツ (スタイルシートファイル) のキャッシュ。false に設定すると、Camel は各メッセージ処理でスタイルシートファイルをリロードします。これは開発に適しています。キャッシュされたスタイルシートは、clearCachedStylesheet 操作を使用して、JMX 経由で実行時に強制的に再読み込みできます。	true	boolean
saxon (producer)	TransformerFactoryClass として Saxon を使用するかどうか。saxon を有効にすると、クラス net.sf.saxon.TransformerFactoryImpl となります。クラスパスに Saxon を追加する必要があります。	false	boolean
saxonExtensionFunctions (advanced)	カスタム net.sf.saxon.lib.ExtensionFunctionDefinition を使用できるようにします。クラスパスに camel-saxon を追加する必要があります。関数はレジストリーで検索されます。検索する複数の値をコンマで区切ることができます。		String
saxonConfiguration (advanced)	カスタム Saxon 設定を使用します		Object
saxonConfiguration Properties (advanced)	カスタムの Saxon 設定プロパティーを設定します		Map
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

XSLT エンドポイントは、URI 構文を使用して設定されます。

```
xslt:resourceUri
```

パスおよびクエリーパラメーターを使用します。

348.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
resourceUri	テンプレートへの **必須** パス。以下は、デフォルトの URIResolver でサポートされています。プリフィックスには、classpath、file、http、ref、または bean. classpath、file、http を付けることができます (classpath はデフォルト)。ref は、レジストリーでリソースを検索します。Bean は、リソースとして使用される Bean のメソッドを呼び出します。Bean の場合は、ドットの後にメソッド名を指定できます (例：bean:myBean.myMethod)		String

348.2.2. クエリーパラメーター (17 パラメーター)

名前	説明	デフォルト	タイプ
allowStAX (producer)	StAX を javax.xml.transform.Source として使用できるようにするかどうか。	true	boolean
contentCache (producer)	ロード時のリソースコンテンツ (スタイルシートファイル) のキャッシュ。false に設定すると、Camel は各メッセージ処理でスタイルシートファイルをリロードします。これは開発に適しています。キャッシュされたスタイルシートは、clearCachedStylesheet 操作を使用して、JMX 経由で実行時に強制的に再読み込みできます。	true	boolean
deleteOutputFile (producer)	output=file を指定した場合、このオプションは、エクスチェンジの処理が完了したときに出力ファイルを削除するかどうかを指定します。たとえば、出力ファイルが一時ファイルである場合、使用後に削除することをお勧めします。	false	boolean
failOnNullBody (producer)	入力本文が null の場合に例外を出力するかどうか。	true	boolean
output (producer)	使用する出力タイプを指定するオプション。可能な値は次のとおりです: 文字列、バイト、DOM、ファイル。最初の 3 つのオプションはすべてメモリーベースであり、ファイルは直接 java.io.File にストリーミングされます。ファイルの場合、キー Exchange.XSLT_FILE_NAME を使用して IN ヘッダーにファイル名を指定する必要があります。これは CamelXsltFileName でもあります。また、ファイル名につながるパスは事前に作成する必要があります。そうしないと、実行時に例外が出力されます。	string	XsltOutput

名前	説明	デフォルト	タイプ
saxon (producer)	TransformerFactoryClass として Saxon を使用するかどうか。saxon を有効にすると、クラス net.sf.saxon.TransformerFactoryImpl となります。クラスパスに Saxon を追加する必要があります。	false	boolean
transformerCacheSize (producer)	Template.newTransformer() の呼び出しを避けるため、再利用のためにキャッシュされる javax.xml.transform.Transformer オブジェクトの数。	0	int
converter (advanced)	org.apache.camel.converter.jaxp.XmlConverter のカスタム実装を使用する場合。		XmlConverter
entityResolver (advanced)	javax.xml.transform.sax.SAXSource でカスタム org.xml.sax.EntityResolver を使用するには。		EntityResolver
errorListener (advanced)	カスタム javax.xml.transform.ErrorListener を使用するように設定できます。これを行うときは、エラーまたは致命的なエラーをキャプチャし、プロパティーとして Exchange に情報を格納するデフォルトのエラーリスナが使用されていないことに注意してください。したがって、このオプションは特別なユースケースにのみ使用してください。		ErrorListener
resultHandlerFactory (advanced)	カスタム org.apache.camel.builder.xml.ResultHandlerFactory を使用できるようにします。これは、カスタム org.apache.camel.builder.xml.ResultHandler 型を使用できるようにします。		ResultHandlerFactory
saxonConfiguration (advanced)	カスタム Saxon 設定を使用します		Object
saxonExtensionFunctions (advanced)	カスタム net.sf.saxon.lib.ExtensionFunctionDefinition を使用できるようにします。クラスパスに camel-saxon を追加する必要があります。関数はレジストリーで検索されます。検索する複数の値をコンマで区切ることができます。		String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
transformerFactory (advanced)	カスタム XSLT トランスフォーマーファクトリーを使用するためのものです		TransformerFactory

名前	説明	デフォルト	タイプ
transformerFactoryClass (advanced)	カスタム XSLT トランスフォーマーファクトリーを使用するには、FQN クラス名として指定します		String
uriResolver (advanced)	カスタム javax.xml.transform.URIResolver を使用するためのものです		URIResolver

348.3. XSLT エンドポイントの使用

次の形式は、XSLT テンプレートを使用して、InOut メッセージ交換 (**JMSReplyTo** ヘッダーがある場合) のメッセージに対する応答を作成する例です。

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl");
```

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、次のルートを使用できます。

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl").
  to("activemq:Another.Queue");
```

348.4. 使用可能なパラメーターの XSLT への取り込み

デフォルトでは、すべてのヘッダーがパラメーターとして追加され、XSLT で使用できるようになります。
パラメーターを使用可能にするには、そのパラメーターを宣言する必要があります。

```
<setHeader headerName="myParam"><constant>42</constant></setHeader>
<to uri="xslt:MyTransform.xsl"/>
```

また、パラメーターを使用できるようにするには、XSLT の最上位レベルでパラメーターを宣言する必要もあります。

```
<xsl: ..... >

  <xsl:param name="myParam"/>

   <xsl:template ...>
```

348.5. SPRING XML バージョン

上記の例を Spring XML で使用するには、次のようなコードを使用します。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="xslt:org/apache/camel/spring/processor/example.xsl"/>
```

```
    <to uri="activemq:Another.Queue"/>
  </route>
</camelContext>
```

例を確認するには、Spring XML とともに テストケース を参照してください。

348.6. XSL:INCLUDE の使用

Camel 2.2 以前
Camel 2.2 以前の XSL ファイルで **xsl:include** を使用する場合、デフォルトの **javax.xml.transform.URIResolver** が使用されます。ファイルは、JVM 開始フォルダーに対して相対的に解決されます。

たとえば、次の include ステートメントは、アプリケーションが開始されたフォルダーから開始して、**staff_template.xsl** ファイルを検索します。

```
<xsl:include href="staff_template.xsl"/>
```

Camel 2.3 以降
Camel 2.3 以降の場合、Camel は **URIResolver** の独自の実装を提供します。これにより、Camel はクラスパスからインクルードファイルをロードできます。

たとえば、次のコードのインクルードファイルは、開始エンドポイントの相対パスに配置されます。

```
<xsl:include href="staff_template.xsl"/>
```

つまり、Camel は **org/apache/camel/component/xslt/staff_template.xsl** の **クラスパス** にファイルを配置します。

classpath: または **file:** を使用して、Camel にクラスパスまたはファイルシステムのいずれかを検索するように指示できます。接頭辞を省略した場合には、Camel はエンドポイント設定の接頭辞を使用します。エンドポイント設定で接頭辞が指定されていない場合、デフォルトは **classpath:** です。

インクルードパスで後方参照することもできます。次の例では、xsl ファイルは **org/apache/camel/component** の下で解決されます。

```
<xsl:include href="../staff_other_template.xsl"/>
```

348.7. XSL:INCLUDE とデフォルトの接頭辞の使用

Camel 2.10.3 以前 では、**classpath:** がデフォルトの接頭辞として使用されます。
file: を使用してロードするように開始リソースを設定する場合、後続のすべてのインクルードには **file:** という接頭辞を付ける必要があります。

Camel 2.10.4 から、Camel は、エンドポイント設定の接頭辞をデフォルトの接頭辞として使用します。

file: または **classpath:** の読み込みを明示的に指定できます。必要に応じて、2 つの読み込みタイプを XSLT スクリプトで混在させることができます。

348.8. SAXON 拡張関数の使用

Saxon 9.2 以降、拡張関数の記述は 統合拡張関数 と呼ばれる新しいメカニズムによって補完され、以下の例に示すように簡単に camel を使用できるようになりました。

```
SimpleRegistry registry = new SimpleRegistry();
registry.put("function1", new MyExtensionFunction1());
registry.put("function2", new MyExtensionFunction2());

CamelContext context = new DefaultCamelContext(registry);
context.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
         .to("xslt:org/apache/camel/component/xslt/extensions/extensions.xslt?
saxonExtensionFunctions=#function1,#function2");
   }
});
```

Spring XML の場合:

```
<bean id="function1" class="org.apache.camel.component.xslt.extensions.MyExtensionFunction1"/>
<bean id="function2" class="org.apache.camel.component.xslt.extensions.MyExtensionFunction2"/>

<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:extensions"/>
    <to uri="xslt:org/apache/camel/component/xslt/extensions/extensions.xslt?
saxonExtensionFunctions=#function1,#function2"/>
  </route>
</camelContext>
```

348.9. 動的スタイルシート

実行時に動的スタイルシートを提供するために、動的 URI を定義できます。詳細は to() で動的 URI を使用する方法 を参照してください。

Camel 2.9 以降で利用可能 (2.11.4、2.12.3、2.13.0 で削除)
Camel は **CamelXsltResourceUri** ヘッダーを提供します。これを使用して、エンドポイント URI で設定されたものに代わるスタイルシートを定義できます。これにより、実行時に動的スタイルシートを提供できます。

348.10. XSLT ERRORLISTENER からの警告、エラー、および致命的なエラーへのアクセス

Camel 2.14 から利用可能

Camel 2.14 から、すべての警告/エラーまたは fatalError は、キーが **Exchange.XSLT_ERROR**、**Exchange.XSLT_FATAL_ERROR**、または **Exchange.XSLT_WARNING** のプロパティーとして現在の Exchange に格納されるので、エンドユーザーは変換中に発生したエラーを把握できます。

たとえば、以下のスタイルシートでは、スタッフに Dob フィールドがある場合には終了していきます。また、xsl:message を使用してカスタムエラーメッセージを含めます。

```
<xsl:template match="/">
```

```
  <html>
    <body>
      <xsl:for-each select="staff/programmer">
        <p>Name: <xsl:value-of select="name"/><br />
          <xsl:if test="dob=''">
            <xsl:message terminate="yes">Error: DOB is an empty string!</xsl:message>
          </xsl:if>
        </p>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
```

例外は、キーが **Exchange.XSLT_WARNING.** の警告として Exchange に保存されます。

348.11. XSLT と JAVA バージョンの使用に関する注意事項

以下は、Camel ユーザーである Sameer が共有してくれた、いくつかの意見を紹介します。

XSLT エンドポイントで問題が発生した場合は、これらの点を確認してください。

単純な xsl を使用して、ある xml から別の xml への単純な変換に xslt エンドポイントを使用しようとしていました。出力 xml は、(ルートの xslt プロセッサーの後に) 表示され続け、コンテンツが含まれていない最も外側の xml タグを使用していました。

DEBUG ログに説明が表示されません。ただし、TRACE ログで、XMLConverter Bean を初期化できなかったことを示すエラー/警告が見つかりました。

数時間頭を悩ませた後、それを機能させるために次のことをしなければなりませんでした (手がかりを与えてくれたユーザーフォーラムの投稿のおかげです)。

1. **class="org.apache.xalan.xsltc.trax.TransformerFactoryImpl"** の Spring コンテキストで定義された Bean を持つ **tFactory** Bean を使用して、ルート **("xslt:my-transformer.xsl?transformerFactory=tFactory")** で TransformerFactory オプションを使用します。
2. Xalan jar を maven pom に追加しました。

私の推測では、JDK 内で提供されるデフォルトの xml 解析メカニズム (私は 1.6.0_03 を使用しています) は、このコンテキストでは正しく機能せず、エラーも発生しません。この方法で Xalan に切り替えると、うまくいきます。これは Camel の問題ではありませんが、xslt コンポーネントページで言及する必要があるかもしれません。

別の注意として、jdk 1.6.0_03 には JAXB 2.0 が同梱されていますが、Camel には 2.1 が必要です。1 つの回避策は、jvm の **jre/lib/endorsed** ディレクトリーに 2.1 jar を追加するか、コンテナーで指定されたとおりに追加することです。

この投稿が初心者の Camel 使用者の時間を節約できることを願っています.

348.12. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- スタートガイド

第349章 XSTREAM DATAFORMAT

Camel バージョン 1.3 以降で利用可能

XStream は、XStream ライブラリー を使用して Java オブジェクトを XML との間でマーシャリングおよびアンマーシャリングするデータ形式です。

camel ルートで XStream を使用するには、このデータ形式を実装する **camel-xstream** に依存関係を追加する必要があります。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xstream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

349.1. オプション

XStream データ形式は、以下に示す 10 個のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
permissions		**String**	xml/json から Java Bean へのアンマーシャリング中に使用できる Java パッケージおよびクラス XStream を制御するパーミッションを追加します。パーミッションは、JVM システムプロパティーを使用して、この場所またはグローバルに設定する必要があります。パーミッションは、プラス記号が許可で、マイナス記号が拒否である構文で指定できます。ワイルドカードは . を接頭辞として使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、com.foo を指定します。複数のパーミッションは、com.foo.,-com.foo.bar.MySecretBean のようにコンマで区切ることができます。以下のデフォルトパーミッションは常に、キー org.apache.camel.xstream.permissions で JVM システムプロパティーを指定して上書きされない限り、-,java.lang.,java.util. が含まれます。
encoding		**String**	使用するエンコーディングを設定します
driver		**String**	カスタム XStream ドライバーを使用するには。インスタンスのタイプは com.thoughtworks.xstream.io.HierarchicalStreamDriver でなければなりません
driverRef		**String**	カスタム XStream ドライバーを参照して、レジストリーを検索します。インスタンスのタイプは com.thoughtworks.xstream.io.HierarchicalStreamDriver でなければなりません

名前	デフォルト	Java タイプ	説明
mode		**String**	重複参照を処理するモード。設定可能な値は以下の通りです: NO_REFERENCES ID_REFERENCES XPATH_RELATIVE_REFERENCES XPATH_ABSOLUTE_REFERENCES SINGLE_NODE_XPATH_RELATIVE_REFERENCES SINGLE_NODE_XPATH_ABSOLUTE_REFERENCES
converters		**List**	カスタム XStream コンバーターを使用するためのクラス名のリスト。クラスは com.thoughtworks.xstream.converters.Converter 型でなければなりません
aliases		**Map**	クラスを XML 要素で使用する短い名前にエイリアスします。
omitFields		**Map**	フィールドがシリアライズされないようにします。フィールドを省略するには、宣言する型を常に指定する必要があり、必ずしも変換される型を指定する必要はありません。
implicitCollections		**Map**	マップされていない XML タグに使用される既定の暗黙的なコレクションを追加します。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

349.2. JAVA DSL を使用

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().xstream().
  to("mqseries:Another.Queue");
```

Camel がメッセージ変換に使用する **XStream** インスタンスを設定する場合は、DSL レベルでそのインスタンスへの参照を渡すだけです。

```
XStream xStream = new XStream();
xStream.aliasField("money", PurchaseOrder.class, "cash");
// new Added setModel option since Camel 2.14
xStream.setModel("NO_REFERENCES");
...

from("direct:marshal").
  marshal(new XStreamDataFormat(xStream)).
  to("mock:marshaled");
```

349.3. XMLINPUTFACTORY と XMLOUTPUTFACTORY

XStream ライブラリー は **javax.xml.stream.XMLInputFactory** と **javax.xml.stream.XMLOutputFactory** を使用します。このファクトリーのどの実装を使用するかを制御できます。

Factory は、次のアルゴリズムを使用して検出されます: 1.**javax.xml.stream.XMLInputFactory**、**javax.xml.stream.XMLOutputFactory** システムプロパティーを使用します。2.**JRE_HOME** ディレクトリーにある **lib/xml.stream.properties** ファイルを使用します。3.JRE で使用可能な jar 内の **META-INF/services/javax.xml.stream.XMLInputFactory**、**META-INF/services/javax.xml.stream.XMLOutputFactory** ファイルを調べて、可能であればサービス API を使用してクラス名を決定します。.4.プラットフォームのデフォルトの XMLInputFactory,XMLOutputFactory インスタンスを使用します。

349.4. XSTREAM DATAFORMAT で XML エンコーディングを設定するには?

Camel 2.2.0 から、キー **Exchange.CHARSET_NAME** を使用して Exchange のプロパティーを設定するか、DSL または Spring 設定から Xstream でエンコーディングプロパティーを設定することにより、Xstream DataFormat で XML のエンコーディングを設定できます。

```
from("activemq:My.Queue").
  marshal().xstream("UTF-8").
  to("mqseries:Another.Queue");
```

349.5. XSTREAM DATAFORMAT のタイプ権限の設定

Camel では、ルートで常に独自の処理ステップを使用して、XStream の unmarhall ステップにルーティングされる特定の XML ドキュメントをフィルタリングおよびブロックできます。Camel 2.16.1、2.15.5 から、XStream のタイプのパーミッション を設定して、特定のタイプのインスタンス化を自動的に許可または拒否できます。

Camel で使用されるデフォルトのタイプ権限設定は、java.lang および java.util パッケージのタイプを除くすべてのタイプを拒否します。この設定は、システムプロパティー org.apache.camel.xstream.permissions を設定することで変更できます。その値はコンマで区切られた許可条件の文字列であり、それぞれが許可または拒否されるタイプを表します。これは、用語の前に '' (注記 '' は省略される場合があります) または '-' がそれぞれ付けられているかどうかによって異なります。

各用語にはワイルドカード文字を含めることができます **''。たとえば、値 "-**,java.lang.,**java.util.**" は、java.lang.* および java.util.* クラスを除くすべてのタイプを拒否することを示します。この値を空の文字列 "" に設定すると、ブラックリストに登録された特定のクラスを拒否し、他のクラスを許可する、デフォルトの XStream の型パーミッション処理に戻ります。

タイプパーミッションの設定は、タイプパーミッションプロパティーを設定することにより、個々の XStream DataFormat インスタンスで拡張できます。

```
<dataFormats>
   <xstream id="xstream-default"
         permissions="org.apache.camel.samples.xstream.*"/>
   ...
```

第350章 YAML SNAKEYAML DATAFORMAT

Camel バージョン 2.17 以降で利用可能

YAML は、Java オブジェクトを YAML との間でマーシャリングおよびアンマーシャリングするためのデータ形式です。

YAML からオブジェクトへのマーシャリングのために、Camel は 3 つの一般的な YAML ライブラリーとの統合を提供します。

- SnakeYAML ライブラリー

すべてのライブラリーには、特別な camel コンポーネントを追加する必要があります (依存関係の段落を参照してください)。デフォルトでは、Camel は SnakeYAML ライブラリーを使用します。

350.1. YAML オプション

YAML SnakeYAML データ形式は、以下に示す 11 のオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
library	**SnakeYAML**	**YAMLLibrary**	使用する yaml ライブラリー。デフォルトでは SnakeYAML です。
unmarshalTypeName		**String**	アンアームシャリング時に使用する Java 型のクラス名
コンストラクター		**String**	入力ドキュメントを構築するための BaseConstructor。
representer		**String**	出力オブジェクトを発行するリプレゼンター。
dumperOptions		**String**	出力オブジェクトを設定するための DuperOptions。
resolver		**String**	暗黙の型を検出するリゾルバー。
useApplicationContextClassLoader	**true**	**Boolean**	ApplicationContextClassLoader をカスタム ClassLoader として使用します。
prettyFlow	**false**	**Boolean**	フロースタイルを使用する場合、エミッターがプリティ YAML ドキュメントを生成するように強制します。
allowAnyType	**false**	**Boolean**	任意のクラスの非整列化を許可します。
typeFilter		**List**	SnakeYAML がアンマーシャリングできるタイプを設定します。

名前	デフォルト	Java タイプ	説明
contentTypeHeader	false	Boolean	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

警告

SnakeYAML は YAML 定義から任意のクラスをロードすることができますが、これはセキュリティ侵害につながる可能性があるため、デフォルトでは SnakeYAML DataForma はロードできるオブジェクトを List や Long などの標準の Java オブジェクトに制限します。カスタム POJO をロードする場合は、それらのタイプを SnakeYAML DataFormat 型フィルターリストに追加する必要があります。ソースが信頼できる場合は、プロパティー allowAnyType を true に設定して、SnakeYAML DataForma が型に対してフィルターを実行しないようにすることができます。

350.2. SNAKEYAML ライブラリーで YAML データ形式を使用する

- Object メッセージを yaml に変換してから MQSeries に送信する

```
from("activemq:My.Queue")
  .marshal().yaml()
  .to("mqseries:Another.Queue");
```

```
from("activemq:My.Queue")
  .marshal().yaml(YAMLLibrary.SnakeYAML)
  .to("mqseries:Another.Queue");
```

- クラスを YAML からロードするように制限する

```
// Creat a SnakeYAMLDataFormat instance
SnakeYAMLDataFormat yaml = new SnakeYAMLDataFormat();

// Restrict classes to be loaded from YAML
yaml.addTypeFilters(TypeFilters.types(MyPojo.class, MyOtherPojo.class));

from("activemq:My.Queue")
  .unmarshal(yaml)
  .to("mqseries:Another.Queue");
```

350.3. SPRING DSL での YAML の使用

Spring DSL でデータ形式を使用する場合には、最初にデータ形式を宣言する必要があります。これは **DataFormats** XML タグで行われます。

```
<dataFormats>
  <!--
    here we define a YAML data format with the id snake and that it should use
    the TestPojo as the class type when doing unmarshal. The unmarshalTypeName
    is optional
  -->
  <yaml
    id="snake"
    library="SnakeYAML"
    unmarshalTypeName="org.apache.camel.component.yaml.model.TestPojo"/>

  <!--
    here we define a YAML data format with the id snake-safe which restricts the
    classes to be loaded from YAML to TestPojo and those belonging to package
    com.mycompany
  -->
  <yaml id="snake-safe">
    <typeFilter value="org.apache.camel.component.yaml.model.TestPojo"/>
    <typeFilter value="com.mycompany\..*" type="regexp"/>
  </yaml>
</dataFormats>
```

そして、ルートでこれらの ID を参照できます。

```
  <route>
    <from uri="direct:unmarshal"/>
    <unmarshal>
      <custom ref="snake"/>
    </unmarshal>
    <to uri="mock:unmarshal"/>
  </route>
  <route>
    <from uri="direct:unmarshal-safe"/>
    <unmarshal>
      <custom ref="snake-safe"/>
    </unmarshal>
    <to uri="mock:unmarshal-safe"/>
  </route>
```

350.4. SNAKEYAML の依存関係

camel ルートで YAML を使用するには、このデータ形式を実装する **camel-snakeyaml** に依存関係を追加する必要があります。

Maven を使用する場合は、pom.xml に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-snakeyaml</artifactId>
```

```
  <version>${camel-version}</version>
</dependency>
```

第351章 YAMMER コンポーネント

Camel バージョン 2.12 以降で利用可能

Yammer コンポーネントを使用すると、Yammer エンタープライズソーシャルネットワークと対話できます。新しいメッセージの作成だけでなく、メッセージ、ユーザー、およびユーザー関係の使用もサポートされています。

Yammer は、すべてのクライアントアプリケーション認証に OAuth 2 を使用します。アカウントで camel-yammer を使用するには、Yammer 内で新しいアプリケーションを作成し、アプリケーションにアカウントへのアクセスを許可する必要があります。最後に、アクセストークンを生成します。詳細は https://developer.yammer.com/authentication/ にあります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-yammer</artifactId>
  <version>${camel-version}</version>
</dependency>
```

351.1. URI 形式

```
yammer:[function]?[options]
```

351.2. コンポーネントのオプション

Yammer コンポーネントは、使用前に必須の Yammer アカウント設定で設定できます。

Yammer コンポーネントは、以下に示す 5 個のオプションをサポートします。

名前	説明	デフォルト	タイプ
consumerKey (security)	コンシューマーキー		String
consumerSecret (security)	コンシューマーシークレット		String
accessToken (security)	アクセストークン		String
config (advanced)	共有 yammer 設定を使用する場合。		YammerConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

これらのオプションは、エンドポイントで直接設定することもできます。

351.3. エンドポイントオプション

Yammer エンドポイントは、URI 構文を使用して設定されます。

```
yammer:function
```

パスおよびクエリーパラメーターを使用します。

351.3.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
function	**必須** 使用する機能。		YammerFunctionType

351.3.2. クエリーパラメーター(28 パラメーター):

名前	説明	デフォルト	タイプ
useJson (Common)	POJO に変換するのではなく raw の JSON を使用する場合は、true に設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
delay (consumer)	ミリ秒単位のポーリング間の遅延	5000	long
limit (consumer)	指定された数のメッセージのみを返します。threaded=true および threaded=extended で機能します。	-1	int
newerThan (consumer)	数値文字列として指定されたメッセージ ID よりも新しいメッセージを返します。これは、新しいメッセージをポーリングするときに使用する必要があります。メッセージを表示していて、返された最新のメッセージが 3516 である場合は、パラメーター newerThan=3516 を使用してリクエストを行い、ページにすでにあるメッセージの重複コピーを取得しないようにすることができます。	-1	int

名前	説明	デフォルト	タイプ
olderThan (consumer)	数値文字列として指定されたメッセージ ID よりも古いメッセージを返します。これは、メッセージのページ付けに役立ちます。たとえば、現在 20 件のメッセージを表示していて、最も古いメッセージが 2912 番である場合、リクエストに oldThan=2912 を追加して、表示されているメッセージより前の 20 件のメッセージを取得できます。	-1	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
threaded (consumer)	threaded=true は、各スレッドの最初のメッセージのみを返します。このパラメーターは、メッセージスレッドを折りたたんで表示するアプリケーションを対象としています。threaded=extended は、Yammer Web インターフェースのデフォルトビューで表示されるように、スレッドのスターターメッセージを最近アクティブなメッセージと最新の 2 つのメッセージの順に返します。		String
userId (consumer)	ユーザー id		String
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。60s (60 秒)、5m30s (5 分 30 秒)、1h (1 時間) などの単位を使用して時間値を指定することもできます。	1000	long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントから cron スケジューラーを使用します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2 や Spring ベースのスケジューラーを使用する場合に、追加のプロパティーを設定します。		Map
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean

名前	説明	デフォルト	タイプ
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessToken (security)	**必須** アクセストークン。		String
consumerKey (security)	**必須** コンシューマーキー。		String
consumerSecret (security)	**必須** コンシューマーシークレット。		文字列

351.4. メッセージの消費

Yammer コンポーネントは、メッセージを消費するためのいくつかのエンドポイントを提供します。

URI	説明
yammer:messages?[options]	ユーザー (API 呼び出しを行うためにアクセストークンが使用されている) の Yammer ネットワーク内のすべてのパブリックメッセージ。Yammer Web インターフェイスのすべての会話に対応します。
yammer:my_feed?[options]	フォロー中とトップの会話の間で行った選択に基づく、ユーザーのフィード。
yammer:algo?[options]	上位の会話に対応するユーザーのアルゴリズムフィード。これは、大多数のユーザーが Yammer Web インターフェイスに表示するものです。
yammer:following?[options]	ユーザーがフォローしている人々、グループ、およびトピックに関する会話であるフォロー中フィード。
yammer:sent?[options]	ユーザーが送信したすべてのメッセージ。
yammer:private?[options]	ユーザーが受信したプライベートメッセージ。

URI	説明
yammer:received?[options]	Camel 2.12.1: ユーザーが受信したすべてのメッセージ。

351.4.1. メッセージの形式

デフォルトでは、すべてのメッセージが **org.apache.camel.component.yammer.model** パッケージで提供される POJO モデルに変換されます。yammer からの元のメッセージは JSON です。メッセージを消費および生成するすべてのエンドポイントに対して、**Messages** オブジェクトが返されます。たとえば、次のようなルートを取ります。

```
from("yammer:messages?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAccessToken"
)
    .to("mock:result");
```

そして、yammer サーバーが返すとしましょう:

```
{
   "messages":[
      {
         "replied_to_id":null,
         "network_id":7654,
         "url":"https://www.yammer.com/api/v1/messages/305298242",
         "thread_id":305298242,
         "id":305298242,
         "message_type":"update",
         "chat_client_sequence":null,
         "body":{
            "parsed":"Testing yammer API...",
            "plain":"Testing yammer API...",
            "rich":"Testing yammer API..."
         },
         "client_url":"https://www.yammer.com/",
         "content_excerpt":"Testing yammer API...",
         "created_at":"2013/06/25 18:14:45 +0000",
         "client_type":"Web",
         "privacy":"public",
         "sender_type":"user",
         "liked_by":{
            "count":1,
            "names":[
               {
                  "permalink":"janstey",
                  "full_name":"Jonathan Anstey",
                  "user_id":1499642294
               }

            ]

         },
         "sender_id":1499642294,
```

```
            "language":null,
            "system_message":false,
            "attachments":[

            ],
            "direct_message":false,
            "web_url":"https://www.yammer.com/redhat.com/messages/305298242"
        },
        {
            "replied_to_id":null,
            "network_id":7654,
            "url":"https://www.yammer.com/api/v1/messages/294326302",
            "thread_id":294326302,
            "id":294326302,
            "message_type":"system",
            "chat_client_sequence":null,
            "body":{
                "parsed":"(Principal Software Engineer) has [[tag:14658]] the redhat.com network. Take a
moment to welcome Jonathan.",
                "plain":"(Principal Software Engineer) has #joined the redhat.com network. Take a moment
to welcome Jonathan.",
                "rich":"(Principal Software Engineer) has #joined the redhat.com network. Take a moment
to welcome Jonathan."
            },
            "client_url":"https://www.yammer.com/",
            "content_excerpt":"(Principal Software Engineer) has #joined the redhat.com network. Take a
moment to welcome Jonathan.",
            "created_at":"2013/05/10 19:08:29 +0000",
            "client_type":"Web",
            "sender_type":"user",
            "privacy":"public",
            "liked_by":{
                "count":0,
                "names":[

                ]

            }
        }
        ]

    }
```

Camel はそれを 2 つの **Message** オブジェクトを含む **Messages** オブジェクトにマーシャリングします。以下に示すように、必要な情報を簡単に取得できる豊富なオブジェクトモデルがあります。

```
Exchange exchange = mock.getExchanges().get(0);
Messages messages = exchange.getIn().getBody(Messages.class);

assertEquals(2, messages.getMessages().size());
assertEquals("Testing yammer API...", messages.getMessages().get(0).getBody().getPlain());
assertEquals("(Principal Software Engineer) has #joined the redhat.com network. Take a moment to
welcome Jonathan.", messages.getMessages().get(1).getBody().getPlain());
```

とはいえ、このデータを POJO にマーシャリングするのは無料ではないため、必要な場合は、**useJson=false** オプションを URI に追加して、純粋な JSON の使用に戻すことができます。

351.5. メッセージの作成

現在のユーザーのアカウントで新しいメッセージを作成するには、次の URI を使用できます。

```
yammer:messages?[options]
```

現在の Camel メッセージ本文は、Yammer メッセージのテキストを設定するために使用されるものです。レスポンスボディーには、メッセージを消費するときと同じ方法で (つまり、デフォルトで **Messages** オブジェクトとして) フォーマットされた新しいメッセージが含まれます。

たとえば、次のルートを参照してください。

```
from("direct:start")
    .to("yammer:messages?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAccessToken"
)
    .to("mock:result");
```

direct:start エンドポイントに **Hi from Camel!** を送信します。メッセージボディー:

```
template.sendBody("direct:start", "Hi from Camel!");
```

サーバー上の現在のユーザーのアカウントに新しいメッセージが作成され、この新しいメッセージも Camel に返されて **Messages** オブジェクトに変換されます。メッセージを消費するときと同様に、**Messages** オブジェクトを調べることができます:

```
Exchange exchange = mock.getExchanges().get(0);
Messages messages = exchange.getIn().getBody(Messages.class);

assertEquals(1, messages.getMessages().size());
assertEquals("Hi from Camel!", messages.getMessages().get(0).getBody().getPlain());
```

351.6. ユーザー関係の取得

Yammer コンポーネントは、ユーザーの関係を取得できます。

```
yammer:relationships?[options]
```

351.7. ユーザーの取得

Yammer コンポーネントは、ユーザーを取得するためのいくつかのエンドポイントを提供します。

URI	説明
`yammer:users?[options]`	現在のユーザーの Yammer ネットワーク内のユーザーを取得します。
`yammer:current?[options]`	現在のユーザーに関するデータを表示します。

351.8. エンリッチャーの使用

camel-yammer のポーリングコンシューマーの 1 つで開始されたルートではなく、エンリッチパターンを使用することが役立つ場合があります (ユーザーまたはリレーションシップコンシューマーの場合は常に)。これは、遅延を設定した頻度に関係なく、コンシューマーが繰り返し発動するためです。ユーザーのデータを検索したり、ある時点でメッセージを取得したいだけの場合は、そのコンシューマーを一度呼び出してから、ルートで取得することをお勧めします。

たとえば、ある時点で現在のユーザーのユーザーデータを取得する必要があるルートがあるとします。。このユーザーを何度もポーリングするのではなく、**pollEnrich** DSL メソッドを使用します。

```
from("direct:start")
    .pollEnrich("yammer:current?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAccessToken"
)
    .to("mock:result");
```

これにより、現在のユーザーの **User** オブジェクトが取得され、Camel メッセージボディーとして設定されます。

351.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第352章 YAHOO クエリー言語コンポーネント

Camel バージョン 2.21 以降で利用可能

yql コンポーネントは、Yahoo クエリー言語プラットフォーム にアクセスするために使用されます。

YQL(Yahoo!クエリー言語) プラットフォームを使用すると、単一のインターフェイスを介して Web 全体でデータのクエリー、フィルター処理、結合を行うことができます。これは、開発者にとって馴染みがあり、適切なデータを取得するのに十分な表現力を持つ、SQL に似た構文を公開します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-yql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

352.1. URI 形式

```
yql://query[?options]
```

query はYQL クエリーを表します。

352.2. オプション

Yahoo クエリー言語コンポーネントは、以下に示す 2 つのオプションをサポートしています。

名前	説明	デフォルト	タイプ
connectionManager (producer)	カスタム設定された HttpClientConnectionManager を使用するため。		HttpClientConnection マネージャー
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Yahoo クエリー言語エンドポイントは、URI 構文を使用して設定されます。

```
yql:query
```

パスおよびクエリーパラメーターを使用します。

352.2.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
query	**必須** 実行する YQL ステートメント。		文字列

352.2.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
callback (producer)	JSONP 形式の JavaScript コールバック関数の名前。コールバックが設定されていて、format=json の場合、応答形式は JSON です。JSON の代わりに XML を使用する方法の詳細は、JSONP-X を参照してください。https://developer.yahoo.com/yql/guide/response.html		文字列
crossProduct (producer)	最適化された値を指定すると、応答の個別の item 要素で返される可能性のある SELECT ステートメントの射影されたフィールドは、代わりに単一の item 要素になるように最適化されます。唯一の許容値は optimized です。詳細: https://developer.yahoo.com/yql/guide/response.htmlresponse-optimizing=		文字列
debug (producer)	true で、diagnostic が true に設定されている場合は、デバッグデータが応答と共に返されます。詳細: https://developer.yahoo.com/yql/guide/dev-external_tables.htmlodt-enable-logging=	false	boolean
diagnostics (producer)	true の場合は、応答とともに診断情報が返されます。	false	boolean
env (producer)	YQL 環境ファイルを介して複数の Open Data Table を使用できます。詳細: https://developer.yahoo.com/yql/guide/yql_storage.htmlusing-records-env-files=		文字列
format (producer)	想定される形式。許可された値: xml または json。	json	文字列
jsonCompat (producer)	ロスレス JSON 処理を有効にします。許可される値は new のみです。詳細: https://developer.yahoo.com/yql/guide/response.htmljson-to-json		文字列
throwExceptionOnFailure (producer)	リモートサーバーからの応答が失敗した場合に YqlHttpException の出力を無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean

名前	説明	デフォルト	タイプ
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
https (security)	HTTPS を使用して YQL と通信するオプション。	true	boolean

352.3. EXCHANGE データ形式

Camel は、本文を JSON または XML 形式の java.lang.String として配信します (上記の **format** オプションを参照してください)。

352.4. メッセージヘッダー

ヘッダー	説明
CamelYqlHttpRequest	YQL に送信された元の HTTP 要求。
CamelYqlHttpStatus	応答からのステータスコード。

352.5. サンプル

352.5.1. 例 1

この例では、YQL にクエリーを実行して、シカゴの現在の風と大気のデータを取得します。

```
from("direct:start")
  .to("yql://select wind, atmosphere from weather.forecast where woeid in (select woeid from geo.places(1) where text='chicago, il'");
```

本文を次のように設定します。

```
{
  "query":{
    "count":1,
    "created":"2017-11-01T19:37:26Z",
    "lang":"en-US",
    "results":{
      "channel":{
        "wind":{
```

```
            "chill":"32",
            "direction":"165",
            "speed":"22"
          },
          "atmosphere":{
            "humidity":"71",
            "pressure":"994.0",
            "rising":"0",
            "visibility":"16.1"
          }
        }
      }
    }
}
```


およびヘッダー:

ヘッダー	値
CamelYqlHttpRequest	https://query.yahooapis.com/v1/public/yql?q=select+wind%2C+atmosphere+from+weather.forecast+where+woeid+in+%28select+woeid+from+geo.places%281%29+where+text%3D%22chicago%2C+il%22%29&format=json&callback=&diagnostics=false&debug=false
CamelYqlHttpStatus	200

352.5.2. 例 2

この例では、YQL にクエリーを実行して Google の見積もりを取得します。

```
from("direct:start")
  .to("yql://select symbol, Ask, Bid, AverageDailyVolume from yahoo.finance.quotes where symbol in ('GOOG')?env=store://datatables.org/alltableswithkeys&https=false&callback=yqlCallback");
```

本文を次のように設定します。


```
/**/yqlCallback({
  "query":{
    "count":1,
    "created":"2017-11-01T19:48:17Z",
    "lang":"en-US",
    "results":{
      "quote":{
        "symbol":"GOOG",
        "Bid":"1025.57",
        "Ask":"1025.92",
        "AverageDailyVolume":"1350640"AverageDailyVolume
      }
    }
  }
});
```


およびヘッダー:

ヘッダー	値
CamelYqlHttpRequest	http://query.yahooapis.com/v1/public/yql?q=select+symbol%2C+Ask%2C+Bid%2C+AverageDailyVolume+from+yahoo.finance.quotes+where+symbol+in+%28%27GOOG%27%29&format=json&callback=yqlCallback&diagnostics=false&debug=false&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys
CamelYqlHttpStatus	200

352.5.3. 例 3

この例では、YQL にクエリーを実行して、Barack Obama が書いた 1 冊の本を取得します。

```
from("direct:start")
  .to("yql://select * from google.books where q='barack obama' and maxResults=1?
format=xml&crossProduct=optimized&env=store://datatables.org/alltableswithkeys");
```

本文を次のように設定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng" yahoo:count="1"
yahoo:created="2017-11-01T20:32:22Z" yahoo:lang="en-US">
  <results>
    <json>
      <kind>books#volumes</kind>
      <totalItems>1993</totalItems>
      <items>
        <kind>books#volume</kind>
        <id>HRCHJp-V0QUC</id>
        <etag>SeTJeSgFDzo</etag>
        <selfLink>https://www.googleapis.com/books/v1/volumes/HRCHJp-V0QUC</selfLink>
        <volumeInfo>
          <title>Dreams from My Father</title>
          <subtitle>A Story of Race and Inheritance</subtitle>
          <authors>Barack Obama</authors>
          <publisher>Broadway Books</publisher>
          <publishedDate>2007-01-09</publishedDate>
          ...
        </volumeInfo>
      </items>
    </json>
  </results>
</query>
<!-- total: 646 -->
```

およびヘッダー:

ヘッダー	値

ヘッダー	値
CamelYqlHttpRequest	https://query.yahooapis.com/v1/public/yql?q=select+*+from+google.books+where+q%3D%27barack+obama%27+and+maxResults%3D1&format=xml&callback=&crossProduct=optimized&diagnostics=false&debug=false&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys
CamelYqlHttpStatus	200

352.6. 関連項目

- YQL 公式ガイド

第353章 ZENDESK コンポーネント

Camel バージョン 2.19 以降で利用可能

Zendesk コンポーネントは、zendesk-java-client を使用してアクセス可能なすべての zendesk.com API へのアクセスを提供します。Zendesk チケット、ユーザー、組織などを管理するためのメッセージを生成できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zendesk</artifactId>
  <version>${camel-version}</version>
</dependency>
```

353.1. ZENDESK オプション

Zendesk コンポーネントは、以下に示す 3 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (common)	共有設定を使用する場合。		ZendeskConfiguration
zendesk (advanced)	共有 Zendesk インスタンスを使用するには。		Zendesk
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Zendesk エンドポイントは、URI 構文を使用して設定されます。

```
zendesk:methodName
```

パスおよびクエリーパラメーターを使用します。

353.1.1. パスパラメーター (1 個のパラメーター):

名前	説明	デフォルト	タイプ
methodName	**必須** 使用する操作。		String

353.1.2. クエリーパラメーター (10 パラメーター)

名前	説明	デフォルト	タイプ
inBody (common)	ボディにて交換で渡されるパラメーターの名前を設定します。		String
serverUrl (common)	接続するサーバー URL。		String
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean
oauthToken (security)	OAuth トークン。		String
password (security)	パスワード。		String
token (security)	セキュリティートークン。		String
username (security)	ユーザー名。		文字列

353.2. URI 形式

```
zendesk://endpoint?[options]
```

353.3. プロデューサーエンドポイント:

プロデューサーエンドポイントは、次に説明するエンドポイント名と関連するオプションを使用できます。

353.4. コンシューマーエンドポイント

どのプロデューサーエンドポイントもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、Scheduled Poll Consumer Options と **consumer.** の接頭辞を使用して、エンドポイントの呼び出しをスケジュールすることができます。配列またはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 回実行されます。

353.5. メッセージヘッダー

CamelZendesk. の接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーには、どのようなオプションでも提供することができます。原則として、パラメーター名は元の **org.zendesk.client.v2.Zendesk** クラスの各 API メソッドの引数名と同じです。ただし、camel API コンポーネントフレームワークでの競合を避けるために、一部の名前は別の名前に変更されています。実際のパラメーター名を確認するには、**org.apache.camel.component.zendesk.internal.ZendeskApiMethod** を確認してください。

353.6. メッセージボディー

すべての結果メッセージボディーは、Zendesk Java クライアントによって提供されるオブジェクトを利用します。プロデューサーエンドポイントは、**inBody** エンドポイントパラメーターで受信メッセージボディーのオプション名を指定できます。

第354章 ZIP DEFLATE COMPRESSION DATAFORMAT

Camel バージョン 2.12 以降で利用可能

Zip データ形式は、メッセージの圧縮および展開形式です。Zip 圧縮を使用してマーシャリングされたメッセージは、エンドポイントで消費される直前に、Zip 解凍を使用してアンマーシャリングできます。圧縮機能は、大きな XML およびテキストベースのペイロードを扱う場合に非常に役立ちます。エンドポイントでペイロードを圧縮および解凍するためのわずかなコストを負担しながら、ネットワーク帯域幅のより最適な使用を容易にします。

INFO:*ファイルでの使用について* Zip データ形式は、(まだ) ファイルを特別にサポートしていません。つまり、大きなファイルを使用すると、ファイルの内容全体がメモリーに読み込まれます。これは、ストリーミングベースのソリューションのメモリーフットプリントを小さくできるように、将来的に変更される可能性があります。

354.1. オプション

Zip Deflate 圧縮データ形式は、次に示す 2 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
compressionLevel	**-1**	**Integer**	0 ~ 9 の間で特定の圧縮を指定します。-1 はデフォルトの圧縮、0 は圧縮なし、9 は最適な圧縮です。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

354.2. MARSHAL

この例では、zip 圧縮 **Deflater.BEST_COMPRESSION** を使用して通常のテキスト/XML ペイロードを圧縮ペイロードにマーシャリングし、MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().zip(Deflater.BEST_COMPRESSION).to("activemq:queue:MY_QUEUE");
```

または、デフォルト設定を使用したい場合は、次のように送信できます

```
from("direct:start").marshal().zip().to("activemq:queue:MY_QUEUE");
```

354.3. UNMARSHAL

この例では、MY_QUEUE という ActiveMQ キューから Zip されたペイロードを元の形式にアンマーシャリングして UnZippedMessageProcessor に転送し、処理します。エラーを回避するために、マーシャリング中に使用される圧縮レベルは、アンマーシャリング中に使用される圧縮レベルと同じである必要があることに注意してください。

```
from("activemq:queue:MY_QUEUE").unmarshal().zip().process(new UnZippedMessageProcessor());
```

-

354.4. 依存関係

このデータ形式は **camel-core** で提供されるため、追加の依存関係は必要ありません。

第355章 ZIP ファイルのデータ形式

Camel バージョン 2.11 以降で利用可能

Zip ファイルデータ形式は、メッセージの圧縮および展開形式です。メッセージはエントリーを 1 つ含む Zip ファイルにマーシャリング (圧縮) でき、エントリーを 1つ含む Zip ファイルは元のファイルの内容にアンマーシャリング (展開) できます。このデータ形式は、Java 7 以降が使用されている限り、ZIP64 をサポートします。

355.1. ZIP ファイルのオプション

Zip ファイルのデータ形式は、以下に示す 4 つのオプションをサポートしています。

名前	デフォルト	Java タイプ	説明
usingIterator	**false**	**Boolean**	zip ファイルに複数のエントリーがある場合には、このオプションを true に設定すると、スプリッター EIP を使用して、ストリーミングモードで反復子を使用してデータを分割できます。
allowEmptyDirectory	**false**	**Boolean**	zip ファイルに複数のエントリーがある場合、このオプションを true に設定すると、ディレクトリーが空であっても反復子を取得できます
preservePathElements	**false**	**Boolean**	ファイル名にパス要素が含まれている場合、このオプションを true に設定すると、zip ファイルでパスを維持できます。
contentTypeHeader	**false**	**Boolean**	データフォーマットがデータ形式を実行できる場合は、データフォーマットの型で Content-Type ヘッダーを設定するかどうか。たとえば、XML にマーシャリングするデータ形式の場合は application/xml、JSON にマーシャリングするデータ形式の場合は JSon です。

355.2. MARSHAL

この例では、Zip ファイル圧縮を使用して通常のテキスト/XML ペイロードを圧縮ペイロードにマーシャリングし、それを MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start")
    .marshal().zipFile()
    .to("activemq:queue:MY_QUEUE");
```

作成された Zip ファイル内の Zip エントリーの名前は、受信した **CamelFileName** メッセージヘッダーに基づいています。これは、ファイルコンポーネントによって使用される標準のメッセージヘッダーです。さらに、送信 **CamelFileName** メッセージヘッダーは、受信 **CamelFileName** メッセージヘッダーの値に ".zip" 接尾辞を付けて自動的に設定されます。たとえば、次のルートで入力ディレクトリーに test.txt という名前のファイルが見つかった場合には、出力は test.txt という名前の単一の Zip エントリーを含む test.txt.zip という名前の Zip ファイルになります。

```
from("file:input/directory?antInclude=*/.txt")
    .marshal().zipFile()
    .to("file:output/directory");
```

着信 **CamelFileName** メッセージヘッダーがない場合 (ファイルコンポーネントがコンシューマーでない場合など)、メッセージ ID がデフォルトで使用されます。メッセージ ID は通常、一意に生成された ID であるため、**ID-MACHINENAME-2443-1211718892437-1-0.zip** のようなファイル名になります。この動作をオーバーライドする場合は、ルートで **CamelFileName** ヘッダーの値を明示的に設定できます。

```
from("direct:start")
    .setHeader(Exchange.FILE_NAME, constant("report.txt"))
    .marshal().zipFile()
    .to("file:output/directory");
```

このルートにより、出力ディレクトリーに report.txt.zip という名前の Zip ファイルが作成され、report.txt という名前の Zip エントリーが 1 つ含まれます。

355.3. UNMARSHAL

この例では、MY_QUEUE という ActiveMQ キューから Zip ファイルペイロードを元の形式にアンマーシャリングして **UnZippedMessageProcessor** に転送し、処理します。

```
from("activemq:queue:MY_QUEUE")
    .unmarshal().zipFile()
    .process(new UnZippedMessageProcessor());
```

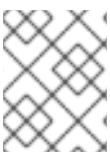
zip ファイルに複数のエントリーがある場合には、ZipFileDataFormat の usingIterator オプションを true にすると、スプリッターを使用してさらに作業を行うことができます。

```
ZipFileDataFormat zipFile = new ZipFileDataFormat();
zipFile.setUsingIterator(true);

from("file:src/test/resources/org/apache/camel/dataformat/zipfile/?consumer.delay=1000&noop=true")
    .unmarshal(zipFile)
    .split(body(Iterator.class)).streaming()
        .process(new UnZippedMessageProcessor())
    .end();
```

または、このように ZipSplitter をスプリッターの式として直接使用できます

```
from("file:src/test/resources/org/apache/camel/dataformat/zipfile?consumer.delay=1000&noop=true")
    .split(new ZipSplitter()).streaming()
        .process(new UnZippedMessageProcessor())
    .end();
```

355.4. AGGREGATE



注記

この集約ストラテジーを適切に機能させるには、完了の先行チェックが必要であることに注意してください。

この例では、入力ディレクトリーで見つかったすべてのテキストファイルを、出力ディレクトリーに格納される 1 つの Zip ファイルに集約します。

```
from("file:input/directory?antInclude=*/.txt")
    .aggregate(constant(true), new ZipAggregationStrategy())
        .completionFromBatchConsumer().eagerCheckCompletion()
        .to("file:output/directory");
```

出力 **CamelFileName** メッセージヘッダーは、java.io.File.createTempFile を使用して作成され、".zip" 接尾辞が付きます。 この動作をオーバーライドする場合は、ルートで **CamelFileName** ヘッダーの値を明示的に設定できます。

```
from("file:input/directory?antInclude=*/.txt")
    .aggregate(constant(true), new ZipAggregationStrategy())
        .completionFromBatchConsumer().eagerCheckCompletion()
        .setHeader(Exchange.FILE_NAME, constant("reports.zip"))
        .to("file:output/directory");
```

355.5. 依存関係

camel ルートで Zip ファイルを使用するには、このデータ形式を実装する **camel-zipfile** に依存関係を追加する必要があります。

Maven を使用する場合は、**pom.xml** に以下を追加するだけで、バージョン番号を最新かつ最高のリリースに置き換えます (最新バージョンのダウンロードページを参照してください)。

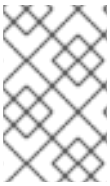
```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zipfile</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第356章 ZIPKIN コンポーネント

Camel 2.18 から利用可能

camel-zipkin コンポーネントは、zipkin を使用して入力および出力 Camel メッセージのトレースとタイミングをとるために使用されます。

イベント (スパン) は、Camel との間で送受信される入力および出力メッセージに対してキャプチャーされます。



注記

camel-zipkin は Camel 2.22.0 でリファクタリングされ、zipkin-scribe を使用せずにデフォルトの http トランスポートを使用する予定です。この作業により、下位互換性が失われる可能性があります。

つまり、どの Camel エンドポイントを zipkin サービス名にマップするかを設定する必要があります。

マッピングは、次を使用して設定できます。

- route id - Camel ルート ID
- endpoint url - Camel エンドポイント URL

どちらの種類でも、Intercept のルールを使用して、ワイルドカードと正規表現を使用して一致させることができます。

すべての Camel メッセージに一致させるには、パターンで * を使用し、それを同じサービス名に設定します。

マッピングが設定されていない場合、Camel はフォールバックし、エンドポイント URI をサービス名として使用します。
ただし、名前に Camel エンドポイント URI の代わりに人間が読める名前を使用できるように、サービスマッピングを設定することをお勧めします。

Camel は、明示的に設定されていないスパンレポーターを自動設定し、Zipkin コレクターへのホスト名とポートが環境変数として設定されている場合

- ZIPKIN_COLLECTOR_HTTP_SERVICE_HOST - The http hostname
- ZIPKIN_COLLECTOR_HTTP_SERVICE_PORT - ポート番号

または

- ZIPKIN_COLLECTOR_THRIFT_SERVICE_HOST - The Scribe (Thrift RPC) hostname
- ZIPKIN_COLLECTOR_THRIFT_SERVICE_PORT - ポート番号

これにより、サービス設定が環境変数として提供される Linux コンテナーでプラットフォームがアプリケーションを実行できるコンテナープラットフォームで camel-zipkin を簡単に使用できます。

356.1. オプション

オプション	デフォルト	説明
rate	1.0f	zipkin によってトレースされるイベントの数を決定するレートを設定します。レートはパーセンテージで表されます (1.0f = 100%、0.5f は 50%、0.1f は 10%)。
spanReporter		**必須:** zipkin スパンイベントを zipkin サーバーに送信するために使用するレポーター。
serviceName		すべての Camel イベントに一致するグローバルサービス名を使用するには
clientServiceMappings		Camel イベントと一致する **クライアント** サービスマッピングを、指定された zipkin サービス名に設定します。内容は Map<String, String> で、キーはパターン、値はサービス名です。このパターンは Intercept のルールを使用します。
serverServiceMappings		Camel イベントと一致する **サーバー** サービスマッピングを、指定された zipkin サービス名に設定します。内容は Map<String, String> で、キーはパターン、値はサービス名です。このパターンは Intercept のルールを使用します。
excludePatterns		パターンに一致する Camel メッセージの zipkin によるトレースを無効にする除外パターンを設定します。設定内容は、キーがパターンである Set<String> です。このパターンは Intercept のルールを使用します。
includeMessageBody	false	zipkin トレースに Camel メッセージボディーを含めるかどうか。これは、本番環境での使用や、ペイロードが大きい場合にはお勧めできません。デバッグログの最大サイズ を設定することで、サイズを制限できます。
includeMessageBodyStreams	false	zipkin トレースにストリームベースのメッセージボディーを含めるかどうか。これには、ルートでストリームキャッシングを有効にするか、CamelContext でグローバルに有効にする必要があります。これは、本番環境での使用や、ペイロードが大きい場合にはお勧めできません。デバッグログの最大サイズ を設定することで、サイズを制限できます。

356.2. 例

camel-zipkin を有効にするには、最初に設定する必要があります

```
ZipkinTracer zipkin = new ZipkinTracer();
// Configure a reporter, which controls how often spans are sent
//   (the dependency is io.zipkin.reporter2:zipkin-sender-okhttp3)
sender = OkHttpSender.create("http://127.0.0.1:9411/api/v2/spans");
zipkin.setSpanReporter(AsyncReporter.create(sender));
// and then add zipkin to the CamelContext
zipkin.init(camelContext);
```

上記の設定は、Camel ルートのすべての入力メッセージと出力メッセージをトレースします。

XML で ZipkinTracer を使用するには、スクライブと zipkin トレーサー Bean を定義するだけです。Camel はそれらを自動的に検出して使用します。

```
<!-- configure how to reporter spans to a Zipkin collector
       (the dependency is io.zipkin.reporter2:zipkin-reporter-spring-beans) -->
<bean id="http" class="zipkin2.reporter.beans.AsyncReporterFactoryBean">
  <property name="sender">
    <bean id="sender" class="zipkin2.reporter.beans.OkHttpSenderFactoryBean">
      <property name="endpoint" value="http://localhost:9411/api/v2/spans"/>
    </bean>
  </property>
  <!-- wait up to half a second for any in-flight spans on close -->
  <property name="closeTimeout" value="500"/>
</bean>

<!-- setup zipkin tracer -->
<bean id="zipkinTracer" class="org.apache.camel.zipkin.ZipkinTracer">
  <property name="serviceName" value="dude"/>
  <property name="spanReporter" ref="http"/>
</bean>
```

356.2.1. ServiceName

ただし、Camel エンドポイントをわかりやすい論理名にマップする場合は、マッピングを追加できます。

- ServiceName *

次のように、すべてのイベントがフォールバックして使用するグローバルサービス名を設定できます。

```
zipkin.setServiceName("invoices");
```

これにより、すべての入力および出力 zipkin トレースに同じサービス名が使用されます。アプリケーションが異なるサービスを使用する場合は、それらをよりきめ細かいクライアント/サーバーサービスマッピングにマッピングする必要があります。

356.2.2. クライアントサービスとサーバーサービスのマッピング

- ClientServiceMappings
- ServerServiceMappings

アプリケーションが他の人が呼び出すことができるサービスをホストしている場合、Camel ルートエンドポイントをサーバーサービスマッピングにマッピングできます。たとえば、Camel アプリケーションに次のルートがあるとします。

```
from("activemq:queue:inbox")
  .to("http:someserver/somepath");
```

これをサーバーサービスとして作成するには、次のマッピングを追加します。

```
zipkin.addServerServiceMapping("activemq:queue:inbox", "orders");
```

次に、メッセージがその受信トレイキューから消費されると、サービス名が orders の zipkin サーバーイベントになります。

ここで、http:someserver/somepath への呼び出しもサービスであり、クライアントサービス名にマップするとします。これは次のように実行できます。

```
zipkin.addClientServiceMapping("http:someserver/somepath", "audit");
```

次に、同じ Camel アプリケーションで、入力エンドポイントと出力エンドポイントを異なる zipkin サービス名にマップしました。

サービスマッピングではワイルドカードを使用できます。すべての出力呼び出しを同じ HTTP サーバーに一致させるには、次のようにします。

```
zipkin.addClientServiceMapping("http:someserver*", "audit");
```

356.3. マッピングルール

サーバーのサービス名のマッピングは、次のルールを使用して発生します

1. from エンドポイントのエンドポイント uri に一致する除外パターンはありますか?はいの場合はスキップします。
2. from エンドポイントのエンドポイント uri と一致する serviceServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
3. 現在のルートのルート ID と一致する serviceServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
4. 交換が開始された元のルート ID と一致する serviceServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
5. サービス名が見つかりませんでした。エクスチェンジは zipkin によって追跡されません。

クライアントのサービス名のマッピングは、次のルールを使用して発生します

1. from エンドポイントのエンドポイント uri に一致する除外パターンはありますか?はいの場合はスキップします。
2. メッセージの送信先のエンドポイントのエンドポイント uri と一致する clientServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
3. 現在のルートのルート ID と一致する clientServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
4. エクスチェンジが開始された元のルート ID と一致する clientServiceMapping に一致がありますか?はいの場合は、見つかったサービス名を使用します。
5. サービス名が見つかりませんでした。エクスチェンジは zipkin によって追跡されません。

356.3.1. クライアントまたはサーバーのマッピングがない

クライアントまたはサーバーサービスマッピングの設定がない場合、CamelZipkin はフォールバックモードで実行され、エンドポイント uris をサービス名として使用します。

上記の例では、次のコードを自分で追加したかのようにサービス名が定義されることを意味します。

```
zipkin.addServerServiceMapping("activemq:queue:inbox", "activemq:queue:inbox");
zipkin.addClientServiceMapping("http:someserver/somepath", "http:someserver/somepath");
```

これは推奨される方法ではありませんが、サービス名のマッピングを行わなくてもすぐに起動して実行できます。ただし、インフラストラクチャー全体に複数のシステムがある場合は、camel エンドポイント uris を使用する代わりに、人間が判読できるサービス名の使用を検討する必要があります。

356.4. CAMEL-ZIPIN-STARTER

Spring Boot を使用している場合は、**camel-zipkin-starter** 依存関係を追加し、メインクラスに **@CamelZipkin** アノテーションを付けて zipkin をオンにすることができます。次に、**application.properties** ファイルで camel-zipkin を設定できます。ここで、Zipkin サーバーのホスト名とポート番号、および上記のオプション表にリストされている他のすべてのオプションを設定できます。

この例は camel-example-zipkin にあります。

第357章 ZOOKEEPER コンポーネント

Camel バージョン 2.9 以降で利用可能

ZooKeeper コンポーネントは、ZooKeeper クラスターとの対話を可能にし、次の機能を Camel に公開します。

1. ZooKeeper 作成モードのいずれかでのノードの作成。
2. 任意のクラスターノードのデータコンテンツを取得および設定します (設定されるデータは **byte[]** に変換可能である必要があります)。
3. 特定のノードに接続された子ノードのリストを作成および取得します。
4. ZooKeeper によって調整されたリーダー選出を利用して、エクスチェンジを処理する必要があるかどうかを判断する分散 **RoutePolicy**。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zookeeper</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

357.1. URI 形式

```
zookeeper://zookeeper-server[:port][/path][?options]
```

URI からのパスは、ZooKeeper サーバー (別名**znode**) のノードを指定します、これは、エンドポイントのターゲットになります。

357.2. オプション

ZooKeeper コンポーネントは、以下に示す 2 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
configuration (advanced)	共有 ZooKeeperConfiguration を使用するには		ZooKeeperConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ZooKeeper エンドポイントは、URI 構文を使用して設定されます。

```
zookeeper:serverUrls/path
```

パスおよびクエリーパラメーターを使用します。

357.2.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
serverUrls	**必須** Zookeeper サーバーホスト (複数のサーバーはコンマで区切ることができます)。		String
path	**必須** ZooKeeper サーバーのノード (別名 znode)。		String

357.2.2. クエリーパラメーター (12 パラメーター)

名前	説明	デフォルト	タイプ
awaitExistence (common)	**非推奨** (使用されていません)	true	boolean
listChildren (common)	ノードの子をリストするかどうか。	false	boolean
timeout (common)	タイムアウトする前に接続を待機する時間間隔。	5000	int
backoff (consumer)	エラー発生後、再試行する前にバックオフする時間間隔。	5000	long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
repeat (consumer)	znode への変更を監視し、繰り返し処理する必要があります。	false	boolean
sendEmptyMessageOnDelete (consumer)	znode の削除時に、空のメッセージをコンシューマーに送信する必要があります	true	boolean

名前	説明	デフォルト	タイプ
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
create (producer)	現在存在しない場合、エンドポイントはノードを作成する必要があります。	false	boolean
createMode (producer)	新しく作成されたノードに使用する作成モード	EPHEMERAL	String
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

357.3. ユースケース

357.3.1. znode からの読み取り

次のスニペットは、**znode /somepath/somenode/** がすでに存在する場合、そこからデータを読み取ります。取得されたデータはエクスチェンジに配置され、ルートの残りの部分に渡されます。

```
from("zookeeper://localhost:39913/somepath/somenode").to("mock:result");
```

ノードがまだ存在しない場合は、エンドポイントがその作成を待機するようにフラグを指定できます。

```
from("zookeeper://localhost:39913/somepath/somenode?awaitCreation=true").to("mock:result");
```

357.3.2. znode からの読み取り (追加の Camel 2.10 以降)

ZooKeeper アンサンブルから受信した **WatchedEvent** が原因でデータが読み取られると、**CamelZookeeperEventType** ヘッダーには、その **WatchedEvent** からの ZooKeeper の **EventType** 値が保持されます。データが最初に読み取られる場合 (**WatchedEvent** によってトリガーされない場合)、**CamelZookeeperEventType** ヘッダーは設定されません。

357.3.3. znode への書き込み

次のスニペットは、エクスチェンジのペイロードを **/somepath/somenode/** の znode に書き込みます (すでに存在する場合)。

```
from("direct:write-to-znode")
    .to("zookeeper://localhost:39913/somepath/somenode");
```

柔軟性を持たせるために、エンドポイントではターゲット **znode** をメッセージヘッダーとして動的に指定することができます。文字列 **CamelZooKeeperNode** をキーとするヘッダーが存在する場合、ヘッダーの値がサーバー上の **znode** へのパスとして使用されます。たとえば、上記と同じルート定義を使用すると、次のコードスニペットは **/somepath/somenode** ではなく、ヘッダー **/somepath/someothernode** からのパスにデータを書き込みます。

警告

ZooKeeper に保存されるデータはバイトベースであるため、**testPayload はbyte[]** に変換可能である必要があります。

```
Object testPayload = ...
template.sendBodyAndHeader("direct:write-to-znode", testPayload, "CamelZooKeeperNode",
"/somepath/someothernode");
```

ノードが存在しない場合にノードも作成するには、**create** オプションを使用する必要があります。

```
from("direct:create-and-write-to-znode")
    .to("zookeeper://localhost:39913/somepath/somenode?create=true");
```

バージョン 2.11 以降では、ヘッダー **CamelZookeeperOperation** を **DELETE** に設定することで、ノードを **削除する** こともできます。

```
from("direct:delete-znode")
    .setHeader(ZooKeeperMessage.ZOOKEEPER_OPERATION, constant("DELETE"))
    .to("zookeeper://localhost:39913/somepath/somenode");
```

または同等になります:

```
<route>
  <from uri="direct:delete-znode" />
  <setHeader headerName="CamelZookeeperOperation">
    <constant>DELETE</constant>
  </setHeader>
  <to uri="zookeeper://localhost:39913/somepath/somenode" />
</route>
```

ZooKeeper ノードはさまざまな型を持つことができます。それらは、Ephemeral または Persistent、Sequenced または Unsequenced のいずれかです。各タイプの詳細については、こちら を参照してください。デフォルトでは、エンドポイントは unsequenced で ephemeral であるノードを作成しますが、タイプは uri 設定パラメーターまたは特別なメッセージヘッダーを介して簡単に操作できます。作成モードに想定される値は、単に **CreateMode** 列挙からの名前です。

- **PERSISTENT**
- **PERSISTENT_SEQUENTIAL**

- **EPHEMERAL**
- **EPHEMERAL_SEQUENTIAL**

たとえば、URI 設定を介して永続的な **znode** を作成するには、次のようにします。

```
from("direct:create-and-write-to-persistent-znode")
    .to("zookeeper://localhost:39913/somepath/somenode?create=true&createMode=PERSISTENT");
```

またはヘッダー **CamelZookeeperCreateMode** を使用します。

警告

ZooKeeper に保存されるデータはバイトベースであるため、**testPayload はbyte[]** に変換可能である必要があります。

```
Object testPayload = ...
template.sendBodyAndHeader("direct:create-and-write-to-persistent-znode", testPayload,
"CamelZooKeeperCreateMode", "PERSISTENT");
```

357.4. ZOOKEEPER 対応のルートポリシー

ZooKeeper では、すぐに使用できる非常にシンプルで効果的なリーダー選出が可能です。このコンポーネントは、**RoutePolicy** でこの選出機能を利用して、ルートを有効にするタイミングと方法を制御します。このポリシーは通常、フェイルオーバーシナリオで使用され、Camel ベースのサーバーのクラスター全体でルートの同一のインスタンスを制御します。非常に一般的なシナリオは、クラスター全体に分散されたルートの複数のインスタンスが存在する単純なマスター - スレーブセットアップですが、それらのうちの 1 つ (マスターのもの) のみが一度に実行される必要があります。マスターに障害が発生した場合、使用可能なスレーブから新しいマスターを選択し、この新しいマスターのルートを開始する必要があります。

ポリシーは、選出に関与する **RoutePolicy** のすべてのインスタンスで共通の **znode** パスを使用します。各ポリシーはその ID をこのノードに書き込み、Zookeeper は書き込みを受け取った順に並べます。次に、ポリシーはノードのリストを読み取り、その ID の位置を確認します。この位置は、ルートを開始するかどうかを決定するために使用されます。ポリシーは起動時に、クラスター全体で開始する必要があるルートインスタンスの数で設定されます。リスト内の位置がこの値よりも小さい場合、そのルートが開始されます。マスター/スレーブシナリオの場合、ルートは 1 つのルートインスタンスで設定され、リストの最初のエントリーのみがそのルートを開始します。すべてのポリシーはリストの更新を監視し、リストが変更された場合、ルートを開始する必要があるかどうかを再計算します。Zookeeper のリーダー選出機能の詳細については、このページ を参照してください。

次の例では、選択にノード **/someapplication/somepolicy** を使用し、ノードリストの上位 '1' エントリーのみを開始するように設定します。つまり、マスターを選択します。

```
ZooKeeperRoutePolicy policy = new
ZooKeeperRoutePolicy("zookeeper:localhost:39913/someapp/somepolicy", 1);
from("direct:policy-controlled")
    .routePolicy(policy)
    .to("mock:controlled");
```

現在、コンポーネントには異なる SLA を持つ 3 つのポリシーが定義されています。

- **ZooKeeperRoutePolicy**
- **CuratorLeaderRoutePolicy** (2.19 以降)
- **MultiMasterCuratorLeaderRoutePolicy** (2.19 以降)

ZooKeeperRoutePolicy は複数のアクティブなノードをサポートしますが、アクティベーションは Camel コンポーネントとそれに対応する Consumer がすでに開始された後にのみ開始されます。これにより、ルートの定義によっては、ノードがアクティブ化されるべきではないというポリシーが確立される前に、コンポーネントがイベントの消費と Exchange の生成をすでに開始できるというリスクが生じます。

CuratorLeaderRoutePolicy は単一のアクティブノードのみをサポートしますが、別の **CamelContext** ライフサイクルメソッドにバインドされています。ルートまたはコンシューマーが開始される前に、このポリシーが開始されるため、ポリシーが決定を下す前に処理されることはありません。

MultiMasterCuratorLeaderRoutePolicy は複数のアクティブノードをサポートし、**CuratorLeaderRoutePolicy** と同じライフサイクルメソッドにバインドされます。ルートまたはコンシューマーが開始される前に、このポリシーが開始されるため、ポリシーが決定を下す前に処理されることはありません。

357.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド

第358章 ZOOKEEPER MASTER コンポーネント

Camel バージョン 2.19 以降で利用可能

zookeeper-master: エンドポイントは、クラスター内の単一のコンシューマーのみが特定のエンドポイントから消費するようにする方法を提供します。その JVM が停止した場合、自動フェイルオーバーします。

これは、同時消費をサポートしていないレガシーバックエンドから消費する必要がある場合、または商業的または安定性の理由により、任意の時点で 1 つの接続しか持てない場合に非常に役立ちます。

358.1. マスターエンドポイントの使用

camel エンドポイントの前に **zookeeper-master:someName:** を付けるだけです。ここで、**someName** は論理名であり、マスターロックを取得するために使用されます。例えば

```
from("zookeeper-master:cheese:jms:foo").to("activemq:wine");
```

上記は、ActiveMQ の [Exclusive Consumers](http://activemq.apache.org/exclusive-consumer.html) タイプの機能をシミュレートしています。しかし、サードパーティーの JMS プロバイダーでは、Exclusive Consumers をサポートしていない場合があります。

358.2. URI 形式

```
zookeeper-master:name:endpoint[?options]
```

endpoint は、マスター/スレーブモードで実行する任意の Camel エンドポイントです。

358.3. オプション

ZooKeeper マスターコンポーネントは、以下に示す 7 個のオプションをサポートしています。

名前	説明	デフォルト	タイプ
containerIdFactory (consumer)	カスタム ContainerIdFactory を使用してコンテナー ID を作成するには。		ContainerIdFactory
zkRoot (consumer)	どのノードがマスター/スレーブであるかなどの情報が保存される Zookeeper で使用するルートパス。デフォルトでは/camel/zookeepermaster/clusters/master を使用します。	/camel/zookeepermaster/clusters/master	String
curator (advanced)	カスタム設定された CuratorFramework を Zookeeper アンサンブルへの接続として使用する場合。		CuratorFramework

名前	説明	デフォルト	タイプ
maximumConnection Timeout (consumer)	Zookeeper アンサンブルへの接続時に使用するミリ秒単位のタイムアウト。	10000	int
zooKeeperUrl (consumer)	Zookeeper アンサンブルの URL。	localhost:2181	String
zooKeeperPassword (security)	Zookeeper アンサンブルに接続するときに使用するパスワード。		String
resolvePropertyPlaceholders (advanced)	起動時にコンポーネントがプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ZooKeeper マスターエンドポイントは、URI 構文を使用して設定されます。

```
zookeeper-master:groupName:consumerEndpointUri
```

パスおよびクエリーパラメーターを使用します。

358.3.1. パスパラメーター (2 個のパラメーター):

名前	説明	デフォルト	タイプ
groupName	**必須** 使用するクラスターグループの名前。		String
consumerEndpointUri	**必須** マスター/スレーブモードで使用するコンシューマーエンドポイント。		String

358.3.2. クエリーパラメーター (4 パラメーター)

名前	説明	デフォルト	タイプ

名前	説明	デフォルト	タイプ
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
synchronous (advanced)	同期処理を厳密に使用するか、Camel が非同期処理を使用できるかどうかを設定します (サポートされている場合)。	false	boolean

358.4. 例

クラスター化された Camel アプリケーションを保護して、1 つのアクティブノードからのファイルのみを消費することができます。

```
// the file endpoint we want to consume from
String url = "file:target/inbox?delete=true";

// use the zookeeper master component in the clustered group named myGroup
// to run a master/slave mode in the following Camel url
from("zookeeper-master:myGroup:" + url)
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

ZooKeeper はデフォルトで **localhost:2181** に接続しますが、これはコンポーネントレベルで設定できます。

```
MasterComponent master = new MasterComponent();
master.setZooKeeperUrl("myzookeeper:2181");
```

ただし、環境変数を使用して ZooKeeper アンサンブルの URL を設定することもできます。

```
export ZOOKEEPER_URL = "myzookeeper:2181"
```

第359章 MASTER ROUTEPOLICY

RoutePolicy を使用して、マスター/スレーブモードでルートを制御することもできます。

その際、ルートポリシーを次のように設定する必要があります。

- Zookeeper アンサンブルへの URL
- クラスターグループの名前
- **重要** であり、ルートを自動起動しないように設定します

ちょっとした例

```
MasterRoutePolicy master = new MasterRoutePolicy();
master.setZooKeeperUrl("localhost:2181");
master.setGroupName("myGroup");

// its import to set the route to not auto startup
// as we let the route policy start/stop the routes when it becomes a master/slave etc
from("file:target/inbox?delete=true").noAutoStartup()
    // use the zookeeper master route policy in the clustered group
    // to run this route in master/slave mode
    .routePolicy(master)
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

359.1. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- スタートガイド