Red Hat build of Apache Camel 4.4

Red Hat build of Apache Camel for Quarkus リファレンス

Red Hat が提供する Red Hat build of Apache Camel for Quarkus

Last Updated: 2024-06-01

Red Hat build of Apache Camel 4.4 Red Hat build of Apache Camel for Quarkus リファレンス

Red Hat が提供する Red Hat build of Apache Camel for Quarkus

法律上の通知

概要

Red Hat build of Apache Camel for Quarkus は、多くの Camel コンポーネント用の Quarkus エクステンションを提供します。このリファレンスでは、Red Hat がサポートする各エクステンションの設定について説明します。

目次

はじめに

RED HAT BUILD OF APACHE CAMEL ドキュメントに関するフィードバック

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. チケットを作成するには、以下のリンクをクリックします。
2. Summary に、問題の簡単な説明を入力します。
3. 説明 に、問題または拡張機能の詳細情報を提供します。問題があるドキュメントのセクションへの URL を含めてください。
4. Submit をクリックして、問題を適切なドキュメントチームにルーティングします。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 サポートレベルの定義

新しい機能、サービス、およびコンポーネントは、いくつかのサポートレベルを通過してから、実稼働環境での使用に適したフルサポート対象の機能として Red Hat build of Apache Camel for Quarkus に組み込まれます。これは、Red Hat の製品に期待されるエンタープライズグレードの安定性と、新しい Red Hat build of Apache Camel for Quarkus テクノロジーをお客様やパートナーにお試しいただき、今後の開発活動の指針となるフィードバックを得る必要性との間の適切なバランスを確保するためです。

表1.1 Red Hat build of Apache Camel for Quarkus のサポートレベル

タイプ	説明
コミュニティーサポート	Red Hat のアップストリームファーストへの取り組みの一環として、Red Hat build of Apache Camel for Quarkus ディストリビューションへの新しいエクステンションの統合が、アップストリームコミュニティーで始まります。これらのエクステンションはアップストリームでテストされ、文書化されていますが、弊社はこれらのエクステンションの成熟度を確認しておらず、今後の製品リリースで Red Hat によって正式にサポートされない可能性があります。  **注記** コミュニティーエクステンションは、Camel Quarkus コミュニティープロジェクトのエクステンションリファレンスページにリストされています。
テクノロジープレビュー	テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。ただし、これらの機能は Red Hat サブスクリプションレベルアグリーメントでは完全にサポートされておらず、機能的に完全でない可能性があり、実稼働環境での使用を目的としていません。Red Hat ではテクノロジープレビュー機能を今後も繰り返し使用することで一般提供に移行できると考えていることから、お客様がこの機能を使用する際に発生する問題の解決に取り組みます。
製品サポート	製品サポートのエクステンションは、Red Hat の公式リリースに含まれており、完全にサポートされます。ドキュメントにギャップはなく、エクステンションはサポートされるすべての設定でテストされます。

第2章 CAMEL QUARKUS エクステンションの概要

2.1. サポートされるエクステンション

90 のエクステンションがあります。

表2.1 Red Hat build of Apache Camel for Quarkus のサポートマトリックス

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Amqp	camel-quarkus-amqp	製品サポート	製品サポート	Apache QPid クライアントを使用した AMQP プロトコルによるメッセージング。
Attachments	camel-quarkus-attachments	製品サポート	製品サポート	Camel メッセージでの添付のサポート
Aws2-cw	camel-quarkus-aws2-cw	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用してメトリクスを AWS CloudWatch に送信します。
Aws2-ddb	camel-quarkus-aws2-ddb	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して、AWS DynamoDB サービスからデータを保存および取得したり、AWS DynamoDB Stream からメッセージを受信したりします。
Aws2-kinesis	camel-quarkus-aws2-kinesis	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して、AWS Kinesis Streams からレコードを消費および作成します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Aws2-lambda	camel-quarkus-aws2-lambda	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して、AWS Lambda 関数を管理および呼び出します。
Aws2-s3	camel-quarkus-aws2-s3	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して、AWS S3 Storage Service からオブジェクトを保存および取得します。
Aws2-sns	camel-quarkus-aws2-sns	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して AWS Simple Notification Topic にメッセージを送信します。
Aws2-sqs	camel-quarkus-aws2-sqs	製品サポート	製品サポート	AWS SDK バージョン 2.x を使用して AWS SQS サービスを送信先および送信元としてメッセージを送受信します。
Azure-servicebus	camel-quarkus-azure-servicebus	テクノロジープレビュー	テクノロジープレビュー	Azure Service Bus との間でメッセージを送受信します。
Azure-storage-blob	camel-quarkus-azure-storage-blob	テクノロジープレビュー	テクノロジープレビュー	SDK v12 を使用して Azure Storage Blob Service からブロブを保存および取得します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Azure-storage-queue	camel-quarkus-azure-storage-queue	テクノロジープレビュー	テクノロジープレビュー	azure-storage-queue コンポーネントは、Azure SDK v12 を使用して Azure Storage Queue への/からのメッセージを保存および取得するために使用されます。
Bean	camel-quarkus-bean	製品サポート	製品サポート	Java Bean のメソッドを呼び出します。
Bean-validator	camel-quarkus-bean-validator	製品サポート	製品サポート	Java Bean Validation API を使用してメッセージボディーを検証します。
Browse	camel-quarkus-browse	製品サポート	製品サポート	BrowsableEndpoint をサポートするエンドポイントで受信したメッセージを調べます。
Cassandraql	camel-quarkus-cassandraql	製品サポート	製品サポート	CQL3 API (Thrift API 以外) を使用して Cassandra 2.0 と統合します。DataStax が提供する Cassandra Java Driver をベースにしています。
Cli-connector	camel-quarkus-cli-connector	製品サポート	製品サポート	Camel CLI に接続するランタイムアダプター
Controlbus	camel-quarkus-controlbus	製品サポート	製品サポート	Camel ルートを管理および監視します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Core	camel-quarkus-core	製品サポート	製品サポート	Camel コア機能と基本的な Camel 言語/Constant、ExchangeProperty、Header、Ref、Simple および Tokeinze
Crypto	camel-quarkus-crypto	製品サポート	製品サポート	Java Cryptographic Extension (JCE) の署名サービスを使用してエクスチェンジに署名し、検証します。
Cron	camel-quarkus-cron	製品サポート	製品サポート	Unix cron 構文で指定されたタイミングでイベントをトリガーする汎用インターフェイス。
Cxf-soap	camel-quarkus-cxf-soap	製品サポート	製品サポート	Apache CXF を使用して SOAP WebServices を公開するか、CXF WS クライアントを使用して外部 WebServices に接続します。
Dataformat	camel-quarkus-dataformat	製品サポート	製品サポート	Camel Data Format を通常の Camel コンポーネントとして使用します。
Dataset	camel-quarkus-dataset	devSupport	devSupport	Camel アプリケーションの負荷テストおよびソークテスト用のデータを提供します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Direct	camel-quarkus-direct	製品サポート	製品サポート	同じ Camel コンテキストから別のエンドポイントを同期的に呼び出します。
Elasticsearch	camel-quarkus-elasticsearch	製品サポート	なし	Java クライアント API 経由で ElasticSearch にリクエストを送信します。 xtensions-elasticsearch-rest-client
Elasticsearch Rest クライアント	elasticsearch-rest-client	製品サポート	なし	Elasticsearch または OpenSearch でクエリーやその他の操作を実行します (低レベルクライアントを使用します)。
Fhir	camel-quarkus-fhir	製品サポート	製品サポート	FHIR (Fast Healthcare Interoperability Resources) 規格を使用して、ヘルスケアドメインの情報を交換します。JSON との間で FHIR オブジェクトをマーシャリングおよびアンマーシャリングします。XML との間で FHIR オブジェクトをマーシャリングおよびアンマーシャリングします。
Flink	camel-quarkus-flink	テクノロジープレビュー	なし	DataSet ジョブを Apache Flink クラスターに送信します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
File	camel-quarkus-file	製品サポート	製品サポート	ファイルの読み取りと書き込みを行います。
Ftp	camel-quarkus-ftp	製品サポート	製品サポート	SFTP、FTP、または SFTP サーバーとの間でファイルをアップロードおよびダウンロードする
Google-bigquery	camel-quarkus-google-bigquery	製品サポート	製品サポート	SQL クエリーまたは Google Client Services API を使用して Google Cloud BigQuery サービスにアクセスする
Google-pubsub	camel-quarkus-google-pubsub	製品サポート	製品サポート	Google Cloud Platform PubSub サービスとの間でメッセージを送受信します。
Grpc	camel-quarkus-grpc	製品サポート	製品サポート	gRPC エンドポイントを公開し、外部 gRPC エンドポイントにアクセスします。
HTTP	camel-quarkus-http	製品サポート	製品サポート	Apache HTTP Client 5.x を使用して、外部の HTTP サーバーにリクエストを送信します。
Infinispan	camel-quarkus-infinispan	製品サポート	製品サポート	Infinispan の分散キー/値のストアとデータグリッドの読み取りと書き込みを行います。
Jasypt	camel-quarkus-jasypt	製品サポート	製品サポート	Jasypt を使用したセキュリティー

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Java-joor-dsl	camel-quarkus-java-joor-dsl	community	community	実行時の Java ルート定義の解析のサポート
Jdbc	camel-quarkus-jdbc	製品サポート	製品サポート	SQL および JDBC を通じてデータベースにアクセスします。
Jira	camel-quarkus-jira	製品サポート	製品サポート	JIRA 問題トラッカーと対話します。
Jms	camel-quarkus-jms	製品サポート	製品サポート	JMS Queue または Topic との間でメッセージを送受信します。
Jpa	camel-quarkus-jpa	製品サポート	製品サポート	Java Persistence API (JPA) を使用して、データベースから Java オブジェクトを保存し、取得します。
JT400	camel-quarkus-jt400	テクノロジープレビュー	テクノロジープレビュー	データキュー、メッセージキュー、またはプログラム呼び出しを使用して、IBM i システムとメッセージを交換します。IBM i は、AS/400 および iSeries サーバーの代替品です。
Jta	camel-quarkus-jta	製品サポート	製品サポート	Java Transaction API (JTA) および Narayana トランザクションマネージャーを使用して、Camel ルートをトランザクションに含めます。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Jt400	camel-quarkus-jta	製品サポート	なし	データキュー、メッセージキュー、またはプログラム呼び出しを使用して、IBM i システムとメッセージを交換します。IBM i は、AS/400 および iSeries サーバーの代替品です。
Kafka	camel-quarkus-kafka	製品サポート	製品サポート	Apache Kafka ブローカーとの間でメッセージを送受信します。
Kamelet	camel-quarkus-kamelet	製品サポート	製品サポート	ルートテンプレートを具体化する
Kubernetes	camel-quarkus-kubernetes	テクノロジープレビュー	テクノロジープレビュー	Kubernetes API に対して操作を実行する
Kudu	camel-kudu	実稼働サポート (IBM Z および IBM Power ではサポートされません)	実稼働サポート (IBM Z および IBM Power ではサポートされません)	Apache Hadoop エコシステムの無料およびオープンソースの列指向データストアである Apache Kudu と対話します。
Language	camel-quarkus-language	製品サポート	製品サポート	Camel がサポートする任意の言語でスクリプトを実行します。
Ldap	camel-quarkus-ldap	製品サポート	製品サポート	LDAP サーバーで検索を実行します。
LRA	camel-quarkus-lra	テクノロジープレビュー	テクノロジープレビュー	Long-Running-Action フレームワークの Camel saga バインディング。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Log	camel-quarkus-log	製品サポート	製品サポート	基礎となるロギングメカニズムにメッセージをログとして記録します。
Mail	camel-quarkus-mail	製品サポート	製品サポート	imap、pop3、および smtp プロトコルを使用してメールを送受信します。添付のある Camel メッセージを MIME-Multipart メッセージに、またはその逆にマーシャリングします。
管理	camel-quarkus-management	製品サポート	製品サポート	JMX 管理ストラテジーと関連する管理リソース。
Mapstruct	camel-quarkus-mapstruct	製品サポート	製品サポート	Mapstruct を使用した型変換
Master	camel-quarkus-master	製品サポート	製品サポート	クラスター内の単一のコンシューマーのみが特定のエンドポイントから消費するようにします。JVM が停止した場合に自動的にフェイルオーバーします。
Micrometer	camel-quarkus-micrometer	製品サポート	製品サポート	Micrometer ライブラリーを使用して、Camel ルートからさまざまなメトリクスを直接収集します。
Microprofile-fault-tolerance	camel-quarkus-microprofile-fault-tolerance	製品サポート	製品サポート	Microprofile フォールトトレラン を使用した Circuit Breaker EIP

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Microprofile-health	camel-quarkus-microprofile-health	製品サポート	製品サポート	MicroProfile Health による Camel ヘルスチェックの公開
Minio	camel-quarkus-minio	製品サポート	製品サポート	Minio SDK を使用して、Minio Storage Service からオブジェクトを保存および取得します。
Mllp	camel-quarkus-mllp	製品サポート	製品サポート	MLLP プロトコルを使用して外部システムと通信します。
Mybatis	camel-quarkus-mybatis	製品サポート	製品サポート	MyBatis を使用して、リレーショナルデータベースでクエリー、ポーリング、挿入、更新、または削除を実行します。
Mock	camel-quarkus-mock	製品サポート	製品サポート	モックを使用してルートおよび仲介ルールをテストします。
Mongodb	camel-quarkus-mongodb	テクノロジープレビュー	テクノロジープレビュー	MongoDB ドキュメントおよびコレクションの操作を実行します。
Netty	camel-quarkus-netty	製品サポート	製品サポート	Netty 4.x で TCP または UDP を使用するソケットレベルのネットワーク。
Netty-http	camel-quarkus-netty-http	製品サポート	製品サポート	Netty 4.x を使用する Netty HTTP サーバーおよびクライアント。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Openapi-java	camel-quarkus-openapi-java	製品サポート	製品サポート	Camel REST DSLL で定義された OpenAPI リソースを公開する
OpenTelemetry	camel-quarkus-opentelemetry	製品サポート	製品サポート	OpenTelemetry を使用した分散トレース
Quartz	camel-quarkus-quartz	製品サポート	製品サポート	Quartz 2.x スケジューラーを使用してメッセージの送信をスケジュールします。
Paho	camel-quarkus-paho	製品サポート	製品サポート	Eclipse Paho MQTT クライアントを使用して MQTT メッセージブローカーと通信します。
Paho-mqtt5	camel-quarkus-paho-mqtt5	製品サポート	製品サポート	Eclipse Paho MQTT v5 クライアントを使用して MQTT メッセージブローカーと通信します。
Platform-http	camel-quarkus-platform-http	製品サポート	製品サポート	現在のプラットフォームで利用可能な HTTP サーバーを使用して HTTP エンドポイントを公開します。
Ref	camel-quarkus-ref	製品サポート	製品サポート	Camel Registry で名前によって動的に検索されたエンドポイントにメッセージをルーティングします。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Rest	camel-quarkus-rest	製品サポート	製品サポート	REST サービスおよび OpenAPI Specification を公開するか、外部の REST サービスを呼び出します。
Rest-openapi	camel-quarkus-rest-openapi	製品サポート	製品サポート	RestProducerFactory インターフェイスを実装するコンポーネントに委任する OpenAPI 仕様ドキュメントに基づいて REST プロデューサーを設定します。
Salesforce	camel-quarkus-salesforce	製品サポート	製品サポート	Java DTO を使用して Salesforce と通信します。
SAGA	camel-quarkus-saga	製品サポート	製品サポート	Saga EIP を使用して、ルート内でカスタムアクションを実行します。
SAP	camel-quarkus-sap	製品サポート	なし	SAP Camel コンポーネントを提供します。
Saxon	camel-quarkus-saxon	製品サポート	製品サポート	XQuery および Saxon を使用して XML ペイロードをクエリーまたは変換します。
Scheduler	camel-quarkus-scheduler	製品サポート	製品サポート	java.util.concurrent.ScheduledExecutorService を使用して、指定された間隔でメッセージを生成します。
Seda	camel-quarkus-seda	製品サポート	製品サポート	同じ JVM の Camel コンテキストから別のエンドポイントを非同期に呼び出します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Servlet	camel-quarkus-servlet	製品サポート	製品サポート	Servlet によって HTTP リクエストを処理します。
Slack	camel-quarkus-slack	製品サポート	製品サポート	Slack との間でメッセージを送受信します。
Snmp	camel-quarkus-snmp	製品サポート	テクノロジープレビュー	トラップを受信し、SNMP (Simple Network Management Protocol) 対応デバイスをポーリングします。
Splunk	camel-quarkus-splunk	製品サポート	製品サポート	Splunk でイベントを公開または検索します。
Splunk HEC	camel-quarkus-splunk-hec	なし	テクノロジープレビュー	splunk コンポーネントを使用すると、HTTP Event Collector を使用して Splunk でイベントを公開できます。
Sql	camel-quarkus-sql	製品サポート	製品サポート	SQL クエリーを実行します。
Telegram	camel-quarkus-telegram	製品サポート	製品サポート	Telegram Bot API として動作するメッセージを送受信します。
Timer	camel-quarkus-timer	製品サポート	製品サポート	java.util.Timer を使用して、指定された間隔でメッセージを生成します。
Validator	camel-quarkus-validator	製品サポート	製品サポート	XML スキーマと JAXP 検証を使用してペイロードを検証します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Velocity	camel-quarkus-velocity	製品サポート	製品サポート	Velocity テンプレートを使用してメッセージを変換します。
Vertx-http	camel-quarkus-vertx-http	製品サポート	製品サポート	Vert.x による Camel HTTP クライアントのサポート
Vertx-websocket	camel-quarkus-vertx-websocket	製品サポート	製品サポート	Vert.x による Camel WebSocket のサポート
XJ	camel-quarkus-xj	製品サポート	製品サポート	XSLT を使用して、JSON および XML メッセージを変換します。
XML IO DSL	camel-quarkus-xml-io-dsl	製品サポート	製品サポート	XML ルート定義を解析するための XML スタック
XSLT	camel-quarkus-xslt	製品サポート	製品サポート	XSLT テンプレートを使用して XML ペイロードを変換します。
XSLT-Saxon	camel-quarkus-xslt-saxon	製品サポート	製品サポート	Saxon を使用した XSLT テンプレートを使用して XML ペイロードを変換します。
Zipfile	camel-quarkus-zipfile	製品サポート	製品サポート	java.util.zip.ZipStream を使用して、ストリームを圧縮および圧縮解除します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Zip-deflater	camel-quarkus-zip-deflater	製品サポート	製品サポート	java.util.zip.Deflater、java.util.zip.Inflater、または java.util.zip.GZIPStream を使用して、ストリームを圧縮および解凍します。

2.2. サポート言語

7 つの言語があります。

表2.2 Red Hat build of Apache Camel for Quarkus のサポートマトリックス (言語)

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Bean	camel-quarkus-bean	製品サポート	製品サポート	Java Bean のメソッドを呼び出します。
Core	camel-quarkus-core	製品サポート	製品サポート	Camel コア機能と基本的な Camel 言語/Constant、ExchangeProperty、Header、Ref、Simple および Tokeinze
Hl7	camel-quarkus-hl7	製品サポート	製品サポート	HL7 MLLP コーデックを使用して、HL7 (Health Care) モデルオブジェクトをマーシャリングおよびアンマーシャリングします。
Jsonpath	camel-quarkus-jsonpath	製品サポート	製品サポート	JSON メッセージのボディーに対して、JSONPath 式を評価します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Jslt	camel-quarkus-jslt	製品サポート	製品サポート	JSLT を使用して JSON ペイロードをクエリーまたは変換します。
Saxon	camel-quarkus-saxon	製品サポート	製品サポート	XQuery および Saxon を使用して XML ペイロードをクエリーまたは変換します。
Xpath	camel-quarkus-xpath	製品サポート	製品サポート	XML ペイロードに対して XPath 式を評価します。

2.3. サポートされるデータフォーマット

データフォーマットは 10 個あります。

表2.3 Red Hat build of Apache Camel for Quarkus のサポートマトリックス (データフォーマット)

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Avro	camel-quarkus-avro	製品サポート	製品サポート	Apache Avro バイナリーデータフォーマットを使用して、メッセージをシリアライズおよびデシリアライズします。
Bindy	camel-quarkus-bindy	製品サポート	製品サポート	Camel Bindy を使用して、片側の POJO と、反対側のコンマ区切り値 (CSV)、固定フィールド長、またはキーと値のペア (KVP) 形式の間のマーシャリングとアンマーシャル

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Crypto (Java Cryptographic Extension)	camel-quarkus-crypto	製品サポート	製品サポート	Camel のマーシャルおよびアンマーシャルフォーマットメカニズムを使用した対称 (共有鍵) 暗号化および復号化。
Gson	camel-quarkus-gson	製品サポート	製品サポート	Gson を使用して、POJO を JSON に、およびその逆にマーシャリングします。
Hl7	camel-quarkus-hl7	製品サポート	製品サポート	HL7 MLLP コーデックを使用して、HL7 (Health Care) モデルオブジェクトをマーシャリングおよびアンマーシャリングします。
Jackson	camel-quarkus-jackson	製品サポート	製品サポート	Jackson を使用して、POJO を JSON にマーシャリングし、その逆も行います。
Jackson-avro	camel-quarkus-jackson-avro	製品サポート	製品サポート	Jackson を使用して、POJO を Avro にマーシャリングし、その逆も行います。
Jackson-protobuf	camel-quarkus-jackson-protobuf	製品サポート	製品サポート	Jackson を使用して、POJO を Protobuf にマーシャリングし、その逆も行います。
Jacksonxml	camel-quarkus-jacksonxml	製品サポート	製品サポート	Jackson の XMLMapper エクステンションを使用して、XML ペイロードを POJO にアンマーシャリングし、戻します。

エクステンション	アーティファクト	JVM サポートレベル	ネイティブサポートレベル	説明
Jaxb	camel-quarkus-jaxb	製品サポート	製品サポート	JAXB2 XML マーシャリング標準を使用して、XML ペイロードを POJO に、およびその逆にアンマーシャリングします。
Xml-jaxp	camel-quarkus-xml-jaxp	製品サポート	製品サポート	Camel XML JAXP
PGP	camel-quarkus-crypto	製品サポート	製品サポート	Camel のマーシャルおよびアンマーシャルフォーマットメカニズムを使用した対称 (共有鍵) 暗号化および復号化。
Soap	camel-quarkus-soap	製品サポート	製品サポート	Java オブジェクトを SOAP メッセージにマーシャリングし、その逆も行います。

第3章 CAMEL QUARKUS エクステンションリファレンス

本章では、Red Hat build of Apache Camel for Quarkus の使用情報を提供します。

3.1. AMQP

Apache QPid クライアントを使用した AMQP プロトコルによるメッセージング。

3.1.1. 含まれるもの

- AMQP コンポーネント、URI 構文： **amqp:destinationType:destinationName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.1.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.1.3. 使用方法

3.1.3.1. org.w3c.dom.Node を使用したメッセージマッピング

Camel AMQP コンポーネントは、**jakarta.jms.Message** および **org.apache.camel.Message** 間のメッセージマッピングをサポートします。Camel メッセージ本文タイプ **org.w3c.dom.Node** を変換する場合は、**camel-quarkus-xml-jaxp** エクステンションがクラスパスに存在することを確認する必要があります。

3.1.3.2. jakarta.jms.ObjectMessage のネイティブモードのサポート

JMS メッセージペイロードを **jakarta.jms.ObjectMessage** として送信する場合、シリアル化のために登録する関連クラスに **@RegisterForReflection(serialization = true)** でアノテーションを付ける必要があります。このエクステンションは、**quarkus.camel.native.reflection.serialization-enabled = true** を自動的に設定することに注意してください。詳細は、ネイティブモードのユーザーガイド を参照してください。

3.1.3.3. 接続プール

quarkus-pooled-jms エクステンションを使用すると、接続のプーリングサポートを取得できます。詳細は、quarkus-pooled-jms エクステンションドキュメントを参照してください。

以下の依存関係を **pom.xml** に追加するだけで、接続のプーリングサポートを取得できます。

```
<dependency>
  <groupId>io.quarkiverse.messaginghub</groupId>
  <artifactId>quarkus-pooled-jms</artifactId>
```

```
</dependency>
```

プーリングのサポートを有効にするには、以下の設定を **application.properties** に追加する必要があります。

```
quarkus.qpid-jms.wrap=true
```

3.1.4. ネイティブモードの transferException オプション

ネイティブモードで **transferException** オプションを使用するには、オブジェクトのシリアル化のサポートを有効にする必要があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

また、シリアル化する予定の例外クラスのシリアル化を有効にする必要があります。以下に例を示します。

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

3.1.5. 追加の Camel Quarkus 設定

エクステンションは、Quarkus Qpid JMS エクステンションを活用します。ConnectionFactory Bean は自動的に作成され、AMQP コンポーネントに接続されます。接続ファクトリーは、Quarkus Qpid JMS 設定オプション を介して設定できます。

3.2. ATTACHMENTS

Camel メッセージでの添付のサポート

3.2.1. 含まれるもの

- Attachments

使用方法と設定の詳細は、上記リンクを参照してください。

3.2.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.3. AVRO

Apache Avro バイナリーデータフォーマットを使用して、メッセージをシリアライズおよびデシリアライズします。

3.3.1. 含まれるもの

- Avro データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.3.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.3.3. 追加の Camel Quarkus 設定

vanilla Camel から知られている標準的な使用方法以外に、Camel Quarkus では、JVM と Native モードの両方でビルド時に Avro スキーマを解析する機能が追加されています。

Avro スキーマファイルから Avro クラスを生成するアプローチは、**quarkus-avro** エクステンションによって作成されたものです。以下が必要です。

1. ***.avsc** ファイルを **src/main/avro** または **src/test/avro** という名前のフォルダーに保存します
2. **quarkus-maven-plugin** の通常の **ビルド** 目標に加えて、**generate-code** 目標を追加します。

```
<plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>generate-code-and-build</id>
            <goals>
                <goal>generate-code</goal>
                <goal>build</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Camel Quarkus Avro 統合テスト および Quarkus Avro 統合テスト の動作設定を参照してください。

3.4. AWS 2 CLOUDWATCH

メトリクスを AWS CloudWatch に送信します。

3.4.1. 含まれるもの

- AWS CloudWatch コンポーネント, URI 構文： **aws2-cw:namespace**

使用方法と設定の詳細は、上記リンクを参照してください。

3.4.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-aws2-cw</artifactId>
</dependency>
```

3.4.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.5. AWS 2 DYNAMODB

AWS SDK バージョン 2.x を使用して、AWS DynamoDB サービスからデータを保存および取得したり、AWS DynamoDB Stream からメッセージを受信したりします。

3.5.1. 含まれるもの

- AWS DynamoDB コンポーネント, URI 構文： **aws2-ddb:tableName**
- AWS DynamoDB Streams コンポーネント、URI 構文： **aws2-ddbstream:tableName**

使用方法と設定の詳細については、上記リンクを参照してください。

3.5.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-aws2-ddb</artifactId>
</dependency>
```

3.5.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.5.4. 追加の Camel Quarkus 設定

3.5.4.1. Quarkus Amazon DynamoDB とのオプションの統合

必要に応じて、Quarkus Amazon DynamoDB エクステンションを Camel Quarkus AWS 2 DynamoDB と組み合わせて使用することができます。これは完全に任意であり、必須ではないことに注意してください。Quarkus のドキュメント に従ってください。ただし、次の注意事項に注意してください。

1. クライアントタイプ **apache** は、次のプロパティーを設定して選択する必要があります。

```
quarkus.dynamodb.sync-client.type=apache
```

2. **DynamoDbClient** は、Camel Quarkus が実行時に検索できるように、 Quarkus CDI 参照 の意味で削除不可である必要があります。たとえば、**DynamoDbClient** を注入するダミー Bean を追加することで、これに到達できます。

```
import jakarta.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

@ApplicationScoped
@Unremovable
class UnremovableDynamoDbClient {
    @Inject
    DynamoDbClient dynamoDbClient;
}
```

3.6. AWS 2 KINESIS

AWS SDK バージョン 2.x を使用して、AWS Kinesis Streams からレコードを消費および作成します。

3.6.1. 含まれるもの

- AWS Kinesis コンポーネント、URI 構文： **aws2-kinesis:streamName**
- AWS Kinesis Firehose コンポーネント、URI 構文： **aws2-kinesis-firehose:streamName**

使用方法と設定の詳細については、上記リンクを参照してください。

3.6.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-kinesis</artifactId>
</dependency>
```

3.6.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.7. AWS 2 LAMBDA

AWS SDK バージョン 2.x を使用して、AWS Lambda 関数を管理および呼び出します。

3.7.1. 含まれるもの

- AWS Lambda コンポーネント、URI 構文： **aws2-lambda:function**

使用方法と設定の詳細は、上記リンクを参照してください。

3.7.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-aws2-lambda</artifactId>
</dependency>
```

3.7.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.7.4. 追加の Camel Quarkus 設定

3.7.4.1. Camel aws2-lambda エクステンションによって quarkus-amazon-lambda を活用することはできません

quarkus-amazon-lambda エクステンションを使用すると、Quarkus を使用して AWS Lambda をビルドできますが、Camel コンポーネントは既存の関数を管理 (デプロイ、アンデプロイなど) します。したがって、**quarkus-amazon-lambda** を **aws2-lambda** エクステンションのクライアントとして使用することはできません。

3.8. AWS 2 S3 STORAGE SERVICE

AWS S3 Storage Service からオブジェクトを保存および取得します。

3.8.1. 含まれるもの

- AWS S3 Storage Service コンポーネント、URI 構文： **aws2-s3://bucketNameOrArn**

使用方法と設定の詳細は、上記リンクを参照してください。

3.8.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
```

```
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-aws2-s3</artifactId>
</dependency>
```

3.8.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.8.4. 追加の Camel Quarkus 設定

3.8.4.1. Quarkus Amazon S3 とのオプションの統合

必要に応じて、Quarkus Amazon S3 エクステンションを Camel Quarkus AWS 2 S3 Storage Service と組み合わせて使用することができます。これは完全に任意であり、必須ではないことに注意してください。Quarkus のドキュメント に従ってください。ただし、次の注意事項に注意してください。

1. クライアントタイプ **apache** は、次のプロパティーを設定して選択する必要があります。

```
quarkus.s3.sync-client.type=apache
```

2. **S3Client** は、Camel Quarkus が実行時に検索できるように、 Quarkus CDI 参照 の意味で削除不可にする必要があります。たとえば、**S3Client** を注入するダミー Bean を追加することで、これに到達できます。

```
import jakarta.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.s3.S3Client;

@ApplicationScoped
@Unremovable
class UnremovableS3Client {
    @Inject
    S3Client s3Client;
}
```

3.9. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)

AWS Simple Notification Topic にメッセージを送信します。

3.9.1. 含まれるもの

- AWS Simple Notification System (SNS)コンポーネント , URI 構文： **aws2-sns:topicNameOrArn**

使用方法と設定の詳細は、上記リンクを参照してください。

3.9.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-aws2-sns</artifactId>
</dependency>
```

3.9.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.9.4. 追加の Camel Quarkus 設定

3.9.4.1. Quarkus Amazon SNS とのオプションの統合

必要に応じて、Quarkus Amazon SNS エクステンションを Camel Quarkus AWS 2 Simple Notification System (SNS) と組み合わせて使用することができます。これは完全に任意であり、必須ではないことに注意してください。Quarkus のドキュメント に従ってください。ただし、次の注意事項に注意してください。

1. クライアントタイプ **apache** は、次のプロパティーを設定して選択する必要があります。

```
quarkus.sns.sync-client.type=apache
```

2. **SnsClient** は、Camel Quarkus が実行時に検索できるように、 Quarkus CDI 参照 の意味で削除不可にする必要があります。たとえば、**SnsClient** を注入するダミー Bean を追加することで、これに到達できます。

```
import jakarta.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.sns.SnsClient;

@ApplicationScoped
@Unremovable
class UnremovableSnsClient {
    @Inject
    SnsClient snsClient;
}
```

3.10. AWS 2 SIMPLE QUEUE SERVICE (SQS)

AWS SQS サービスとの間でメッセージを送受信します。

3.10.1. 含まれるもの

- AWS Simple Queue Service (SQS)コンポーネント , URI 構文： **aws2-sqs:queueNameOrArn**

使用方法と設定の詳細は、上記リンクを参照してください。

3.10.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-aws2-sqs</artifactId>
</dependency>
```

3.10.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.10.4. 追加の Camel Quarkus 設定

3.10.4.1. Quarkus Amazon SQS とのオプションの統合

必要に応じて、Quarkus Amazon SQS エクステンションを Camel Quarkus AWS 2 Simple Queue Service (SQS) と組み合わせて使用することができます。これは完全に任意であり、必須ではないことに注意してください。Quarkus のドキュメント に従ってください。ただし、次の注意事項に注意してください。

1. クライアントタイプ **apache** は、次のプロパティーを設定して選択する必要があります。

   ```
   quarkus.sqs.sync-client.type=apache
   ```

2. **SqsClient** は、Camel Quarkus が実行時に検索できるように、 Quarkus CDI 参照 の意味で削除不可にする必要があります。たとえば、**SqsClient** を注入するダミー Bean を追加することで、これに到達できます。

   ```
   import jakarta.enterprise.context.ApplicationScoped;
   import io.quarkus.arc.Unremovable;
   import software.amazon.awssdk.services.sqs.SqsClient;

   @ApplicationScoped
   @Unremovable
   class UnremovableSqsClient {
       @Inject
       SqsClient sqsClient;
   }
   ```

3.11. AZURE SERVICEBUS

Azure Service Bus との間でメッセージを送受信します。

3.11.1. 含まれるもの

- Azure ServiceBus コンポーネント, URI 構文： **azure-servicebus:topicOrQueueName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.11.2. Maven コーディネート

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-azure-servicebus</artifactId>
</dependency>
```

3.12. AZURE STORAGE BLOB SERVICE

SDK v12 を使用して Azure Storage Blob Service からブロブを保存および取得します。

3.12.1. 含まれるもの

- Azure Storage Blob Service コンポーネント、URI 構文： **azure-storage-blob:accountName/containerName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.12.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-azure-storage-blob</artifactId>
</dependency>
```

3.12.3. 使用方法

3.12.3.1. Micrometer メトリクスのサポート

Reactor Netty トランスポートの Micrometer メトリクスのコレクションを有効にする場合は、**quarkus-micrometer** の依存関係を宣言して、Quarkus メトリクス HTTP エンドポイント経由で利用できるようにします。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

3.12.4. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.13. AZURE STORAGE QUEUE SERVICE

azure-storage-queue コンポーネントは、Azure SDK v12 を使用して Azure Storage Queue への/からのメッセージを保存および取得するために使用されます。

3.13.1. 含まれるもの

- Azure Storage Queue Service コンポーネント、URI 構文： **azure-storage-queue:accountName/queueName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.13.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-azure-storage-queue</artifactId>
</dependency>
```

3.13.3. 使用方法

3.13.3.1. Micrometer メトリクスのサポート

Reactor Netty トランスポートの Micrometer メトリクスのコレクションを有効にする場合は、**quarkus-micrometer** の依存関係を宣言して、Quarkus メトリクス HTTP エンドポイント経由で利用できるようにします。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

3.13.4. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.14. BEAN バリデーター

Java Bean Validation API を使用してメッセージボディーを検証します。

3.14.1. 含まれるもの

- Bean Validator コンポーネント、URI 構文： **bean-validator:label**

使用方法と設定の詳細は、上記リンクを参照してください。

3.14.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-bean-validator</artifactId>
</dependency>
```

3.14.3. 使用方法

3.14.3.1. ValidatorFactory の設定

このエクステンションの実装は、Quarkus Hibernate Validator エクステンション を利用します。

そのため、Camel のプロパティー (**constraintValidatorFactory**、**messageInterpolator**、**traversableResolver**、**validationProviderResolver**、**validatorFactory**) によって **ValidatorFactory** を設定することはできません。

デフォルトの **ValidatorFactory** (Quarkus によって作成される) に注入される Bean を作成して **ValidatorFactory** を設定できます。詳細は、Quarkus CDI のドキュメント を参照してください。

3.14.3.2. ネイティブモードでのカスタム検証グループ

ネイティブモードでカスタム検証グループを使用する場合は、反映するためにすべてのインターフェイスを登録する必要があります (ドキュメント を参照)。

以下に例を示します。

```
@RegisterForReflection
public interface OptionalChecks {
}
```

3.14.4. Camel Quarkus の制限

(META-INF/validation.xml ファイルを指定して) XML として制約を記述することはできません。サポートされているのは Java アノテーションのみです。これは、Quarkus Hibernate Validator エクステンションの制限によって生じます (issue を参照)。

3.15. BEAN

Java Bean のメソッドを呼び出します。

3.15.1. 含まれるもの

- Bean コンポーネント、URI 構文： **bean:beanName**
- Bean メソッド言語
- Class コンポーネント、URI 構文： **class:beanName**

使用方法と設定の詳細については、上記リンクを参照してください。

3.15.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.15.3. 使用方法

Camel レジストリーで使用できる Bean のメソッドを呼び出す場合を除き、Bean コンポーネントおよび Bean メソッド言語は、Quarkus CDI Bean を呼び出すこともできます。

3.16. BINDY

Camel Bindy を使用して、片側の POJO と、反対側のコンマ区切り値 (CSV)、固定フィールド長、またはキーと値のペア (KVP) 形式の間のマーシャリングとアンマーシャル

3.16.1. 含まれるもの

- Bindy CSV データフォーマット
- Bindy 固定長データフォーマット
- Bindy キー/値ペアデータフォーマット

使用方法と設定の詳細については、上記リンクを参照してください。

3.16.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.16.3. Camel Quarkus の制限

ネイティブモードで camel-quarkus-bindy を使用する場合は、ビルドマシンのロケールのみがサポートされます。

たとえば、ロケールが french のビルドマシンの場合、以下のコード

```
BindyDataFormat dataFormat = new BindyDataFormat();
dataFormat.setLocale("ar");
```

は、JVM モードでは期待どおりに数値をアラビア数字にフォーマットします。ただし、ネイティブモードでは数値をフランス語的にフォーマットします。

さらに調整しないと、ビルドマシンのデフォルトロケールが使用されます。別のロケールは、quarkus.native.user-language および quarkus.native.user-country 設定プロパティーで指定できます。

3.17. BROWSE

BrowsableEndpoint をサポートするエンドポイントで受信したメッセージを調べます。

3.17.1. 含まれるもの

- Browse コンポーネント、URI 構文： **browse:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.17.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.18. CASSANDRA CQL

CQL3 API (Thrift API 以外) を使用して Cassandra 2.0 と統合します。DataStax が提供する Cassandra Java Driver をベースにしています。

3.18.1. 含まれるもの

- Cassandra CQL コンポーネント、URI 構文： **cql:beanRef:hosts:port/keyspace**

使用方法と設定の詳細は、上記リンクを参照してください。

3.18.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.18.3. 追加の Camel Quarkus 設定

3.18.3.1. ネイティブモードでの Cassandra 集約リポジトリー

ネイティブモードで **CassandraAggregationRepository** などの Cassandra 集約リポジトリーを使用するには、ネイティブシリアライズのサポートを有効化 する必要があります。

さらに、エクスチェンジボディがカスタムタイプである場合、クラス宣言に **@RegisterForReflection(serialization = true)** のアノテーションを付けてシリアライズ用に登録する必要があります。

3.19. CLI CONNECTOR

Camel CLI に接続するランタイムアダプター

3.19.1. 含まれるもの

- CLI Connector

使用方法と設定の詳細は、上記リンクを参照してください。

3.19.2. Maven コーディネート

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-cli-connector</artifactId>
</dependency>
```

3.20. CONTROL BUS

Camel ルートを管理および監視します。

3.20.1. 含まれるもの

- Control Bus コンポーネント、URI 構文： **controlbus:command:language**

使用方法と設定の詳細は、上記リンクを参照してください。

3.20.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.20.3. 使用方法

3.20.3.1. 統計

stats コマンドエンドポイントを使用する場合は、JMX を有効にするために、**camel-quarkus-management** エクステンションをプロジェクトの依存関係として追加する必要があります。Maven ユーザーは、**pom.xml** に以下を追加する必要があります。

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

3.20.3.2. 言語

次の言語は、Red Hat build of Apache Camel for Quarkus の Control Bus エクステンションでの使用がサポートされています。

- Bean 言語
- ExchangeProperty 言語
- Header 言語
- Simple 言語

3.20.3.2.1. Bean

Bean 言語を使用して、Bean でメソッドを呼び出し、ルートの状態を制御できます。**org.apache.camel.quarkus:camel-quarkus-bean** エクステンションをクラスパスに追加する必要があります。Maven ユーザーは、次の依存関係を POM に追加する必要があります。

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

ネイティブモードでは、Bean クラスに **@RegisterForReflection** のアノテーションを付ける必要があります。

3.20.3.2.2. Simple

Simple 言語を使用して、ルートの状態を制御できます。次の例では、**ProducerTemplate** を使用して、id **foo** のルートを停止します。

```
template.sendBody(
  "controlbus:language:simple",
  "${camelContext.getRouteController().stopRoute('foo')}"
);
```

OGNL 表記を使用するには、依存関係として **org.apache.camel.quarkus:camel-quarkus-bean** エクステンションを追加する必要があります。

ネイティブモードでは、OGNL 表記で使用されるクラスをリフレクション用に登録する必要があります。上記のコードスニペットでは、**camelContext.getRouteController()** から返される **org.apache.camel.spi.RouteController** クラスを登録する必要があります。これはサードパーティークラスであるため、**@RegisterForReflection** で直接アノテーションを付けることはできません。代わりに、別のクラスにアノテーションを付けて、登録するターゲットクラスを指定することができます。たとえば、Camel ルートを定義するクラスは、**@RegisterForReflection(targets = { org.apache.camel.spi.RouteController.class })** でアノテーションを付けることができます。

または、次の行を **src/main/resources/application.properties** に追加します。

```
quarkus.camel.native.reflection.include-patterns = org.apache.camel.spi.RouteController
```

3.20.4. Camel Quarkus の制限

3.20.4.1. 統計

JMX は GraalVM ではサポートされていないため、**stats** アクションはネイティブモードでは使用できません。そのため、クラスパス上で **camel-quarkus-management** エクステンションを使用してネイティブイメージをビルドしようとすると、ビルドに失敗します。

この機能は、Red Hat build of Apache Camel for Quarkus ではサポートされません。

3.21. CORE

Camel コア機能と基本的な Camel 言語/Constant、ExchangeProperty、Header、Ref、Simple および Tokeinze

3.21.1. 含まれるもの

- Constant 言語
- ExchangeProperty 言語
- File 言語
- Header 言語
- Ref 言語
- Simple 言語
- Tokenize 言語

使用方法と設定の詳細については、上記リンクを参照してください。

3.21.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.21.3. 追加の Camel Quarkus 設定

3.21.3.1. Simple 言語

3.21.3.1.1. OGNL 表記の使用

Simple 言語から OGNL 表記を使用する場合は、**camel-quarkus-bean** エクステンションを使用する必要があります。

たとえば、以下の簡易式は、**Client** 型のメッセージボディーの **getAddress()** メソッドにアクセスします。

```
---
simple("${body.address}")
---
```

このような場合、ここで説明するように、camel-quarkus-bean エクステンションで追加の依存関係を取る必要があり ます。ネイティブモードでは、反映するためにいくつかのクラスを登録する必要がある場合があります。上記の例では、**Client** クラスを 反映するために登録する 必要があります。

3.21.3.1.2. ネイティブモードでの動的型解決の使用

次のような単純な式から型を動的に解決する場合:

- **simple("${mandatoryBodyAs(TYPE)}")**
- **simple("${type:package.Enum.CONSTANT}")**
- **from("...").split(bodyAs(TYPE.class))**
- **simple("${body} is TYPE")**

反映するため、一部のクラスを手動で登録する必要がある場合があります。

たとえば、以下の単純な式は、ランタイム時に **java.nio.ByteBuffer** 型を動的に解決します。

```
---
simple("${body} is 'java.nio.ByteBuffer'")
---
```

したがって、**java.nio.ByteBuffer** クラスを 反映するために登録する 必要があります。

3.21.3.1.3. ネイティブモードでのクラスパスリソースと Simple 言語の使用

次の例のように、ルートがクラスパスから Simple スクリプトをロードすることになっている場合

```
from("direct:start").transform().simple("resource:classpath:mysimple.txt");
```

次に、Quarkus **quarkus.native.resources.includes** プロパティーを使用して、以下に示すようにネイティブ実行可能ファイルにリソースを含める必要があります。

```
quarkus.native.resources.includes = mysimple.txt
```

3.21.3.1.4. ネイティブモードのプロパティーを介したカスタム Bean の設定

#class:* や **#type:*** などの設定でネイティブモードのプロパティーを介してカスタム Bean を指定する場合は、リフレクション用にいくつかのクラスを手動で登録する必要がある場合があります。

たとえば、以下のカスタム Bean 定義には、Bean のインスタンス化とセッターの呼び出しにリフレクションを使用することが含まれます。

```
---
camel.beans.customBeanWithSetterInjection =
#class:org.example.PropertiesCustomBeanWithSetterInjection
camel.beans.customBeanWithSetterInjection.counter = 123
---
```

そのため、クラス **PropertiesCustomBeanWithSetterInjection** を リフレクション用に登録 する必要があります。この場合は、フィールドアクセスを省略できることに注意してください。

設定プロパティー	タイプ	デフォルト
quarkus.camel.bootstrap.enabled true に設定すると、**CamelRuntime** は自動的に起動します。	**boolean**	**true**
quarkus.camel.service.discovery.exclude-patterns クラスパスの Camel サービス定義ファイルと一致する Ant-path フォーマットのパターンのコンマ区切りリスト。一致するファイルで定義されたサービスは、**org.apache.camel.spi.FactoryFinder メカニズムでは**検出できません**。 除外は包含よりも優先されます。ここで定義された除外は、Camel Quarkus エクステンションで追加されたサービスの検出性を拒否するのにも使用できます。 値の例: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	
quarkus.camel.service.discovery.include-patterns クラスパスの Camel サービス定義ファイルと一致する Ant-path フォーマットのパターンのコンマ区切りリスト。指定したファイルが **exclude-patterns** で除外されない限り、一致するファイルで定義されたサービスは、**org.apache.camel.spi.FactoryFinder** メカニズムで検出されます。 Camel Quarkus エクステンションには、デフォルトで一部のサービスが含まれる可能性があることに注意してください。ここで選択されそれらのサービスに追加されたサービスおよび **exclude-patterns** で定義された除外は、ユニオンセットに適用されます。 値の例: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	

設定プロパティー	タイプ	デフォルト
quarkus.camel.service.registry.exclude-patterns クラスパスの Camel サービス定義ファイルと一致する Ant-path フォーマットのパターンのコンマ区切りリスト。一致するファイルで定義されたサービスは、アプリケーションの静的初期化中に Camel レジストリーに **追加されません**。 除外は包含よりも優先されます。ここで定義された除外は、Camel Quarkus エクステンションで追加されたサービスの登録を拒否するのにも使用できます。 値の例: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar****	string	
quarkus.camel.service.registry.include-patterns クラスパスの Camel サービス定義ファイルと一致する Ant-path フォーマットのパターンのコンマ区切りリスト。指定したファイルが **exclude-patterns** で除外されない限り、一致するファイルで定義されたサービスは、アプリケーションの静的初期化中に Camel レジストリーに追加されます。 Camel Quarkus エクステンションには、デフォルトで一部のサービスが含まれる可能性があることに注意してください。ここで選択されそれらのサービスに追加されたサービスおよび **exclude-patterns** で定義された除外は、ユニオンセットに適用されます。 値の例: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	string	
quarkus.camel.runtime-catalog.components **true** の場合、アプリケーションに埋め込まれた Runtime Camel Catalog には、アプリケーションで利用可能な Camel コンポーネントの JSON スキーマが含まれます。それ以外の場合は、コンポーネントの JSON スキーマは Runtime Camel Catalog で利用可能ではなく、アクセスしようとすると RuntimeException が発生します。 これを **false** に設定すると、ネイティブイメージのサイズを縮小することができます。JVM モードでは、ネイティブモードとの動作の一貫性を確立する場合を除き、このフラグを **false** に設定することは実際の利点がありません。	boolean	true
quarkus.camel.runtime-catalog.languages **true** の場合、アプリケーションに埋め込まれた Runtime Camel Catalog には、アプリケーションで利用可能な Camel 言語の JSON スキーマが含まれます。それ以外の場合は、言語の JSON スキーマは Runtime Camel Catalog で利用可能ではなく、アクセスしようとすると RuntimeException が発生します。 これを **false** に設定すると、ネイティブイメージのサイズを縮小することができます。JVM モードでは、ネイティブモードとの動作の一貫性を確立する場合を除き、このフラグを **false** に設定することは実際の利点がありません。	boolean	true

設定プロパティー	タイプ	デフォルト
quarkus.camel.runtime-catalog.dataformats **true** の場合、アプリケーションに埋め込まれた Runtime Camel Catalog には、アプリケーションで利用可能な Camel データフォーマットの JSON スキーマが含まれます。それ以外の場合は、データフォーマットの JSON スキーマは Runtime Camel Catalog で利用可能ではなく、アクセスしようとすると RuntimeException が発生します。 これを **false** に設定すると、ネイティブイメージのサイズを縮小することができます。JVM モードでは、ネイティブモードとの動作の一貫性を確立する場合を除き、このフラグを **false** に設定することは実際の利点がありません。	boolean	true
quarkus.camel.runtime-catalog-models **true** の場合、アプリケーションに埋め込まれた Runtime Camel Catalog には、アプリケーションで利用可能な Camel EIP モデルの JSON スキーマが含まれます。それ以外の場合は、EIP モデルの JSON スキーマは Runtime Camel Catalog で利用可能ではなく、アクセスしようとすると RuntimeException が発生します。 これを **false** に設定すると、ネイティブイメージのサイズを縮小することができます。JVM モードでは、ネイティブモードとの動作の一貫性を確立する場合を除き、このフラグを **false** に設定することは実際の利点がありません。	boolean	true
quarkus.camel.routes-discovery.enabled 静的な初期化時のルートの自動検出を有効にします。	boolean	true
quarkus.camel.routes-discovery.exclude-patterns RouteBuilder クラスの除外フィルタースキャンに使用されます。除外フィルタリングは、包含フィルターよりも優先されます。パターンは Ant-path スタイルのパターンを使用しています。複数のパターンをコンマで区切って指定することができます。たとえば、Bar から始まるすべてのクラスを除外するには、**/Bar* を使用します。特定のパッケージからのすべてのルートを除外するには、com/mycompany/bar/* を使用します。特定のパッケージおよびそのサブパッケージからのすべてのルートを除外するには、2 つのワイルドカードを使用します (com/mycompany/bar/**)。特定の 2 つのパッケージからのすべてのルートを除外するには、com/mycompany/bar/*,com/mycompany/stuff/* を使用します。	string	

設定プロパティー	タイプ	デフォルト
quarkus.camel.routes-discovery.include-patterns RouteBuilder クラスの包含フィルタースキャンに使用されます。除外フィルタリングは、包含フィルターよりも優先されます。パターンは Ant-path スタイルのパターンを使用しています。複数のパターンをコンマで区切って指定することができます。たとえば、Foo から始まるすべてのクラスを含めるには、**/Foo* を使用します。特定のパッケージからのすべてのルートを含めるには、com/mycompany/foo/* を使用します。特定のパッケージおよびそのサブパッケージからのすべてのルートを含めるには、2 つのワイルドカードを使用します (com/mycompany/foo/**)。特定の 2 つのパッケージからのすべてのルートを含めるには、com/mycompany/foo/*,com/mycompany/stuff/* を使用します。	**string**	
quarkus.camel.native.reflection.exclude-patterns 反映のために登録から **除外** されるクラス名に一致する Ant-path スタイルパターンのコンマ区切りリスト。**java.lang.Class.getName()** メソッドによって返されるままのクラス名フォーマットを使用します。パッケージセグメントはピリオド **.** で区切られ、内部クラスはドル記号 **$** で区切られます。 このオプションは、**include-patterns** により選択されたセットを絞り込みます。デフォルトでは、クラスは除外されません。 このオプションは、Quarkus エクステンションによって内部で登録されたクラスの登録解除には使用できません。	**string**	

設定プロパティー	タイプ	デフォルト
quarkus.camel.native.reflection.include-patterns 反映のために登録されるクラス名に一致する Ant-path フォーマットのパターンのコンマ区切りリスト。**java.lang.Class.getName()** メソッドによって返されるままのクラス名フォーマットを使用します。パッケージセグメントはピリオド **.** で区切られ、内部クラスはドル記号 **$** で区切られます。 デフォルトでは、クラスは含まれません。このオプションで選択されるセットは、**exclude-patterns** により絞り込むことができます。 Quarkus エクステンションは通常、反映のために必要なクラスを登録することに注意してください。このオプションは、組み込みの機能では十分ではない場合に有用です。 このオプションは、コンストラクター、フィールド、およびメソッドの完全な反映アクセスを有効にします。よりきめ細かい制御が必要な場合は、Java コードで **io.quarkus.runtime.annotations.RegisterForReflection** アノテーションを使用することを検討してください。 このオプションが適切に機能するには、以下の条件のうち少なくとも 1 つが満たされる必要があります。 - パターンにワイルドカード (* または /) はありません。選択したクラスを含むアーティファクトには、Jandex インデックス (**META-INF/jandex.idx**) が含まれます。選択したクラスを含むアーティファクトは、**application.properties** の **quarkus.index-dependency.*** ファミリーのオプションを使用して、インデックス化用に登録されます。以下に例を示します。 ` **quarkus.index-dependency.my-dep.group-id = org.my-group quarkus.index-dependency.my-dep.artifact-id = my-artifact** ` ここで、**my-dep** は、**org.my-group** と **my-artifact** が関連していることを Quarkus に伝えるために選択したラベルになります。	string	
quarkus.camel.native.reflection.serialization-enabled **true** の場合、基本クラスはシリアル化用に登録されます。そうしないと、基本クラスはネイティブモードでのシリアル化のために自動的に登録されません。シリアル化のために自動的に登録されたクラスのリストは、CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES にあります。これを **false** に設定すると、ネイティブイメージのサイズを縮小することができます。JVM モードでは、ネイティブモードとの動作の一貫性を確立する場合を除き、このフラグを **true** に設定することは実際の利点がありません。	boolean	false

設定プロパティー	タイプ	デフォルト
quarkus.camel.expression.on-build-time-analysis-failure ビルド時にルート定義から式を抽出できない場合の対処方法。	org.apache.camel.quarkus.core.CamelConfig.FailureRem	warn

設定プロパティー	タイプ	デフォルト
quarkus.camel.expression.extraction-enabled ビルド時にルート定義から式を抽出する必要があるかどうかを示します。無効にすると、式はランタイムにコンパイルされます。	boolean	true
quarkus.camel.event-bridge.enabled Camel イベントから CDI イベントへのブリッジを有効にするかどうか。 これにより、CDI オブザーバーを Camel イベント用に設定できます。たとえば、**org.apache.camel.quarkus.core.events**、**org.apache.camel.quarkus.main.events**、および **org.apache.camel.impl.event** パッケージに属するもの。 この設定項目は、Camel イベント用に設定されたオブザーバーがアプリケーションに存在する場合にのみ効果があることに注意してください。	boolean	true
quarkus.camel.source-location-enabled camel ソースの場所を有効/無効にするためのビルド時の設定オプション	boolean	false
quarkus.camel.main.shutdown.timeout **CamelMain#stop()** が完了するまで待機するタイムアウト時間 (ミリ秒の精度)	java.time.Duration	PT3S

設定プロパティー	タイプ	デフォルト
quarkus.camel.main.arguments.on-unknown **CamelMain** が不明な引数に遭遇した際のアクション。fail: **CamelMain** の使用状況ステートメントを出力し、**RuntimeException** を出力します。ignore: 警告が解除し、アプリケーションは通常どおりに起動します。warn: **CamelMain** の使用状況ステートメントを出力しますが、アプリケーションが通常どおりに起動するのを許可します。	org.apache.camel.quarkus.core.CamelConfig.FailureRem	warn

設定プロパティー	タイプ	デフォルト


ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.22. CRON

Unix cron 構文で指定されたタイミングでイベントをトリガーする汎用インターフェイス。

3.22.1. 含まれるもの

- Cron コンポーネント、URI 構文： **cron:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.22.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.22.3. 追加の Camel Quarkus 設定

cron コンポーネントは汎用インターフェイスコンポーネントであるため、Camel Quarkus ユーザーは、実装を提供する別のエクステンションと一緒に cron エクステンションを使用する必要があります。

3.23. CRYPTO (JCE)

Java Cryptographic Extension (JCE) の署名サービスを使用してエクスチェンジに署名し、検証します。

3.23.1. 含まれるもの

- Crypto (Java Cryptographic Extension) データフォーマット
- Crypto (JCE)コンポーネント、URI 構文： **crypto:cryptoOperation:name**
- PGP データ形式

使用方法と設定の詳細については、上記リンクを参照してください。

3.23.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.23.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.24. CXF

Apache CXF を使用して SOAP WebServices を公開するか、CXF WS クライアントを使用して外部 WebServices に接続します。

3.24.1. 含まれるもの

- CXF コンポーネント、URI 構文： **cxf:beanId:address**

使用方法と設定の詳細は、上記リンクを参照してください。

3.24.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-cxf-soap</artifactId>
</dependency>
```

3.24.3. 使用方法

3.24.3.1. 全般

camel-quarkus-cxf-soap は、CXF Extensions for Quarkus プロジェクト (**quarkus-cxf**) のエクステンションを使用します。

これは、サポートされるユースケースと WS 仕様のセットの大部分が **quarkus-cxf** によって提供されることを意味します。

重要

== サポートされるエクステンション

現在、次の quarkus-cxf エクステンション **のみ** がサポートされています。

暗黙的に、**camel-quarkus-cxf-soap** の推移的な依存関係として、以下がサポートされます。

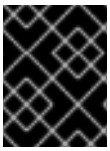
- **quarkus-cxf**
- **quarkus-cxf-rt-features-logging**

WS-Security またはその他の関連機能が必要な場合は、次のサポートされる拡張機能を追加できます。

- **quarkus-cxf-rt-ws-security**
- **quarkus-cxf-services-sts**
- **quarkus-cxf-xjc-plugins**

=== WS-ReliableMessaging

CXF WS-ReliableMessaging の完全なサポートは現在提供されていません。この機能は、バージョン 3.8 ではテクノロジープレビュー機能のままとなります。



重要

サポート対象のユースケースおよび WS 仕様の詳細は、Quarkus CXF の Reference を参照してください。

3.24.3.2. 依存関係の管理

Red Hat build of Apache Camel for Quarkus は、CXF および **quarkus-cxf** のバージョンを 管理 します。これらのプロジェクトと互換性があるバージョンを選択する必要はありません。

3.24.3.3. クライアント

camel-quarkus-cxf-soap (追加の依存関係は不要) を使用すると、CXF クライアントを Camel ルートでプロデューサーとして使用できます。

```
import org.apache.camel.builder.RouteBuilder;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.context.SessionScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class CxfSoapClientRoutes extends RouteBuilder {

    @Override
    public void configure() {

        /* You can either configure the client inline */
        from("direct:cxfUriParamsClient")
```

```
            .to("cxf://http://localhost:8082/calculator-ws?
wsdlURL=wsdl/CalculatorService.wsdl&dataFormat=POJO&serviceClass=org.foo.CalculatorService")
;

        /* Or you can use a named bean produced below by beanClient() method */
        from("direct:cxfBeanClient")
            .to("cxf:bean:beanClient?dataFormat=POJO");

    }

    @Produces
    @SessionScoped
    @Named
    CxfEndpoint beanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("http://localhost:8082/calculator-ws");
        result.setWsdlURL("wsdl/CalculatorService.wsdl"); // a resource in the class path
        return result;
    }
}
```

CalculatorService は以下のようになります。

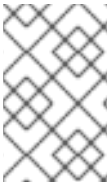
```
import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService(targetNamespace = CalculatorService.TARGET_NS) (1)
public interface CalculatorService {

    public static final String TARGET_NS = "http://acme.org/wscalculator/Calculator";

    @WebMethod (2)
    public int add(int intA, int intB);

    @WebMethod (3)
    public int subtract(int intA, int intB);

    @WebMethod (4)
    public int divide(int intA, int intB);

    @WebMethod (5)
    public int multiply(int intA, int intB);
}
```

(1)(2)(3)(4)(5) 注記: JAX-WS アノテーションが必要です。Simple CXF フロントエンドはサポートされていません。ネイティブモードで適切に機能させるには、複雑なパラメータータイプに JAXB アノテーションが必要です。

ヒント

このサービスエンドポイントインターフェイスを実装する quay.io/l2x6/calculator-ws:1.2 コンテナーに対して、このクライアントアプリケーションをテストできます。

```
$ docker run -p 8082:8080 quay.io/l2x6/calculator-ws:1.2
```



注記

quarkus-cxf は、**@io.quarkiverse.cxf.annotation.CXFClient** アノテーションを使用する SOAP クライアントの注入 をサポートします。詳細は、**quarkus-cxf** ユーザーガイドの SOAP Clients の章を参照してください。

3.24.3.4. Server

camel-quarkus-cxf-soap を使用すると、SOAP エンドポイントを Camel ルートのコンシューマーとして公開できます。このユースケースには、追加の依存関係は必要ありません。

```
import org.apache.camel.builder.RouteBuilder;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class CxfSoapRoutes extends RouteBuilder {

    @Override
    public void configure() {
        /* A CXF Service configured through a CDI bean */
        from("cxf:bean:helloBeanEndpoint")
                .setBody().simple("Hello ${body} from CXF service");

        /* A CXF Service configured through Camel URI parameters */
        from("cxf:///hello-inline?wsdlURL=wsdl/HelloService.wsdl&serviceClass=org.foo.HelloService")
                        .setBody().simple("Hello ${body} from CXF service");
    }

    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        return result;
    }
}
```

これら 2 つのサービスが提供されるパスは、たとえば **application.properties** で設定できる **quarkus.cxf.path** 設定プロパティー の値によって異なります。

application.properties

```
quarkus.cxf.path = /soap-services
```

この設定を適用すると、**http://localhost:8080/soap-services/hello-bean** および **http://localhost:8080/soap-services/hello-inline** で、2 つのサービスにそれぞれアクセスできます。

WSDL には、上記の URL に **?wsdl** を追加することでアクセスできます。

重要

他のエクステンションが HTTP エンドポイントを公開しないと 100% 確信できない限り、アプリケーションで **quarkus.cxf.path =** / を使用しないでください。

quarkus-cxf 2.0.0 より前 (つまり、Red Hat build of Apache Camel for Quarkus 3.0.0 より前)、**quarkus.cxf.path** のデフォルト値は / でした。このデフォルト値は、他の Quarkus エクステンションがさらなる HTTP エンドポイントの公開を阻止していたため、変更されました。とりわけ、RESTEasy、Vert.x、SmallRye Health (health エンドポイントは公開されていません) がこの影響を受けていました。

注記

quarkus-cxf は、SOAP エンドポイントを公開する代替方法をサポートします。詳細は、**quarkus-cxf** ユーザーガイドの SOAP Services の章を参照してください。

3.24.3.5. 要求および応答のロギング

org.apache.cxf.ext.logging.LoggingFeature を使用して、クライアントとサーバーの両方の SOAP メッセージの詳細ロギングを有効にできます。

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.cxf.ext.logging.LoggingFeature;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.context.SessionScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class MyBeans {

    @Produces
    @ApplicationScoped
    @Named("prettyLoggingFeature")
    public LoggingFeature prettyLoggingFeature() {
        final LoggingFeature result = new LoggingFeature();
        result.setPrettyLogging(true);
        return result;
    }

    @Inject
    @Named("prettyLoggingFeature")
    LoggingFeature prettyLoggingFeature;

    @Produces
    @SessionScoped
    @Named
```

```
    CxfEndpoint cxfBeanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("https://acme.org/calculator");
        result.setWsdlURL("wsdl/CalculatorService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }

    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }
}
```

注記

org.apache.cxf.ext.logging.LoggingFeature のサポートは、**camel-quarkus-cxf-soap** 依存関係として **io.quarkiverse.cxf:quarkus-cxf-rt-features-logging** によって提供されます。アプリケーションに明示的に追加する必要はありません。

3.24.3.6. WS 仕様

サポートされる WS 仕様の範囲は、Quarkus CXF プロジェクトによって提供されます。

camel-quarkus-cxf-soap は、**io.quarkiverse.cxf:quarkus-cxf** エクステンションを介して、以下の仕様のみをカバーします。

- JAX-WS
- JAXB
- WS-Addressing
- WS-Policy
- MTOM

アプリケーションが WS-Security や WS-Trust などの他の WS 仕様を必要とする場合は、それをカバーする Quarkus CXF 依存関係を追加する必要があります。どの WS 仕様がどの Quarkus CXF エクステンションでカバーされているかを確認するには、Quarkus CXF の Reference ページを参照してください。

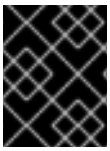
ヒント

Red Hat build of Apache Camel for Quarkus および Quarkus CXF には、さまざまな WS 仕様を実装したアプリケーションの実行可能なサンプルとして役立つ インテグレーション テスト が多数含まれています。

3.24.3.7. ツール

quarkus-cxf は、以下の 2 つの CXF ツールをラップします。

- **wsdl2Java** - WSDL からサービスクラスを生成する 場合
- **java2ws** - Java クラスから WSDL を生成する 場合



重要

wsdl2Java が適切に機能するためには、アプリケーションは **io.quarkiverse.cxf:quarkus-cxf** に直接依存する必要があります。

ヒント

wsdlvalidator はサポート対象外ですが、**application.properties** で以下の設定を指定し、**wsdl2Java** を使用して WSDL を検証できます。

application.properties

```
quarkus.cxf.codegen.wsdl2java.additional-params = -validate
```

3.24.3.8. java.net.http.HttpClientを使用した CXF クライアントでの DoS ベクトルの可能性

CXF クライアントが基礎となる HTTP クライアントとして **java.net.http.HttpClient** を使用している場合、CXF の問題 により、スレッドが終了しないため、CXF の問題により、アプリケーションがクラッシュする可能性があります。

この問題は、要求ごとに CXF クライアントを繰り返し作成されると、**java.net.http.HttpClient** で発生します。アプリケーションのライフサイクル全体でクライアントを保持すると、この問題は発生しません。

Apache Camel for Quarkus]-[および Quarkus CXF 2.2.3 以降、特定の CXF クライアントの HTTP クライアント実装の選択は、**quarkus.cxf.client.yourClient.http-conduit-factory** プロパティーによって制御されます。デフォルトでは、Quarkus CXF によって作成された CXF クライアントは **java.net.HttpURLConnection** を HTTP クライアントとして使用します。そのため、この問題はデフォルトでは発生しません。この問題は、**quarkus.cxf.client.yourClient.http-conduit-factory=HttpClientHTTPConduitFactory** を設定すると発生する可能性があります。

3.24.3.8.1. DoS ベクトルの緩和策

- **java.net.http.HttpClient**- アプリケーションの有効期間中にクライアントが 1 回のみ作成されることが絶対的に確認できる場合は、java.net.http.HttpClient のみを使用します。
- HC5 や **java.net.HttpURLConnection** などのさまざまな HTTP クライアント実装でサポートされる CXF クライアントを使用します。

3.25. DATA FORMAT

Camel Data Format を通常の Camel コンポーネントとして使用します。

Red Hat build of Apache Camel for Quarkus でサポートされるデータフォーマットの詳細は、サポートされるデータフォーマット を参照してください。

3.25.1. 含まれるもの

- Data Format コンポーネント、URI 構文： **dataformat:name:operation**

使用方法と設定の詳細は、上記リンクを参照してください。

3.25.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.26. DATASET

Camel アプリケーションの負荷テストおよびソークテスト用のデータを提供します。

3.26.1. 含まれるもの

- Dataset コンポーネント、URI 構文： **dataset:name**
- DataSet Test コンポーネント、URI 構文： **dataset-test:name**

使用方法と設定の詳細については、上記リンクを参照してください。

3.26.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.27. DIRECT

同じ Camel コンテキストから別のエンドポイントを同期的に呼び出します。

3.27.1. 含まれるもの

- Direct コンポーネント、URI 構文： **direct:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.27.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.28. ELASTICSEARCH

Java クライアント API 経由で ElasticSearch にリクエストを送信します。

3.28.1. 含まれるもの

- Elasticsearch コンポーネント, URI 構文： **elasticsearch:clusterName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.28.2. Maven コーディネート

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

3.29. ELASTICSEARCH 低レベル REST クライアント

Elasticsearch または OpenSearch でクエリーやその他の操作を実行します (低レベルクライアントを使用します)。

3.29.1. 含まれるもの

- Elasticsearch Low level Rest Client コンポーネント, URI 構文： **elasticsearch-rest-client:clusterName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.29.2. Maven コーディネート

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-elasticsearch-rest-client</artifactId>
</dependency>
```

3.30. FHIR

FHIR (Fast Healthcare Interoperability Resources) 規格を使用して、ヘルスケアドメインの情報を交換します。JSON との間で FHIR オブジェクトをマーシャリングおよびアンマーシャリングします。XML との間で FHIR オブジェクトをマーシャリングおよびアンマーシャリングします。

3.30.1. 含まれるもの

- FHIR コンポーネント、URI 構文： **fhir:apiName/methodName**
- FHIR JSon データフォーマット
- FHIR XML データフォーマット

使用方法と設定の詳細については、上記リンクを参照してください。

3.30.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.30.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.30.4. 追加の Camel Quarkus 設定

デフォルトでは、FHIR コンポーネントと DataFormat のデフォルト値であるため、FHIR バージョン **R4** および **DSTU3** のみがネイティブモードで有効になっています。

設定プロパティー	タイプ	デフォルト
quarkus.camel.fhir.enable-dstu2 ネイティブモードで FHIR DSTU2 Specs を有効にします。	**boolean**	**false**
quarkus.camel.fhir.enable-dstu2_hl7org ネイティブモードで FHIR DSTU2_HL7ORG Specs を有効にします。	**boolean**	**false**

設定プロパティー	タイプ	デフォルト
quarkus.camel.fhir.enable-dstu2_1 ネイティブモードで FHIR DSTU2_1 Specs を有効にします。	boolean	false
quarkus.camel.fhir.enable-dstu3 ネイティブモードで FHIR DSTU3 Specs を有効にします。	boolean	false
quarkus.camel.fhir.enable-r4 ネイティブモードで FHIR R4 Specs を有効にします。	boolean	true
quarkus.camel.fhir.enable-r5 ネイティブモードで FHIR R5 Specs を有効にします。	boolean	false

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.31. FILE

ファイルの読み取りと書き込みを行います。

3.31.1. 含まれるもの

- File コンポーネント、URI 構文： **file:directoryName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.31.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.31.3. 追加の Camel Quarkus 設定

3.31.3.1. 特定のエンドポイントから消費するコンシューマがクラスタ内で 1 つだけであること

同じルートが複数の JVM にデプロイされている場合、このエクステンションを マスターのエクステンション と組み合わせて使用すると便利な場合があります。このようなセットアップでは、キャメルマスター namespace 全体で一度に 1 つのコンシューマーがアクティブになります。

たとえば、以下のルートを複数の JVM にデプロイするとします。

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single JVM at a time");
```

以下のようなプロパティーを使用して、ファイルクラスターサービスを有効にすることができます。

```
quarkus.camel.cluster.file.enabled = true
quarkus.camel.cluster.file-root = target/cluster-folder-where-lock-file-will-be-held
```

その結果、1 つのコンシューマーが **ns** camel マスター namespace 全体でアクティブになります。これは、ある時点で、1 つのタイマーだけがすべての JVM 間でエクスチェンジを生成することを意味します。つまり、メッセージは一度に 1 つの JVM で 100 ミリ秒ごとにログに記録されます。

ファイルクラスターサービスは、**quarkus.camel.cluster.file.*** プロパティーを微調整できます。

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.file.enabled quarkus.camel.cluster.file.* 設定に従って、ファイルロッククラスターサービスを自動的に設定するかどうか。	**boolean**	**false**
quarkus.camel.cluster.file-id クラスターサービス ID (デフォルトは null)。	**string**	

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.file-root ルートパス (デフォルトは null)。	string	
quarkus.camel.cluster.file-order サービス検索の順序/優先度 (デフォルトは 2147482647)。	java.lang.Integer	
quarkus.camel.cluster.file.acquire-lock-delay ロックの取得を開始するまでの待機時間 (デフォルトは 1000 ミリ秒)。	string	
quarkus.camel.cluster.file.acquire-lock-interval ロックの取得を試みるまでの待機時間 (デフォルトは 10000 ミリ秒)。	string	

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.file.attributes サービスに関連付けられたカスタム属性 (デフォルトは空のマップ)。	Map<String,String>	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.32. FLINK

DataSet ジョブを Apache Flink クラスターに送信します。

3.32.1. 含まれるもの

- Flink コンポーネント, URI 構文： **flink:endpointType**

使用方法と設定の詳細は、上記リンクを参照してください。

3.32.2. Maven コーディネート

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

3.33. FTP

SFTP、FTP、または SFTP サーバーとの間でファイルをアップロードおよびダウンロードする

3.33.1. 含まれるもの

- FTP コンポーネント、URI 構文： **ftp:host:port/directoryName**
- FTPS コンポーネント、URI 構文： **ftps:host:port/directoryName**

- SFTP コンポーネント、URI 構文： **sftp:host:port/directoryName**

使用方法と設定の詳細については、上記リンクを参照してください。

3.33.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.34. GOOGLE BIGQUERY

SQL クエリーまたは Google Client Services API を使用して Google Cloud BigQuery サービスにアクセスする

3.34.1. 含まれるもの

- Google BigQuery コンポーネント , URI 構文： **google-bigquery:projectId:datasetId:tableId**
- Google BigQuery Standard SQL コンポーネント , URI 構文： **google-bigquery-sql:projectId:queryString**

使用方法と設定の詳細については、上記リンクを参照してください。

3.34.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-google-bigquery</artifactId>
</dependency>
```

3.34.3. 使用方法

ネイティブモードで **google-bigquery-sql** を使用してクラスパスから SQL スクリプトを読み取る場合は、それらが **quarkus.native.resources.includes** 設定プロパティーを介してネイティブイメージに追加されていることを確認する必要があります。詳細については、Quarkus のドキュメント を確認してください。

3.35. GOOGLE PUBSUB

Google Cloud Platform PubSub サービスとの間でメッセージを送受信します。

3.35.1. 含まれるもの

- Google Pubsub コンポーネント、URI 構文： **google-pubsub:projectId:destinationName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.35.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-google-pubsub</artifactId>
</dependency>
```

3.35.3. Camel Quarkus の制限

デフォルトでは、メッセージ本文が **String** または **byte[]** 以外の場合は常に、Camel PubSub コンポーネントは **ObjectOutputStream** を介して JDK オブジェクトのシリアル化を使用します。

このようなシリアル化は GraalVM ではまだサポートされていないため、このエクステンションはカスタムの Jackson ベースのシリアライザーを提供して、複雑なメッセージペイロードを JSON としてシリアル化します。

ペイロードにバイナリーデータが含まれている場合は、カスタムの Jackson シリアライザー/デシリアライザーを作成して処理する必要があります。これを行う方法については、Quarkus Jackson ガイド を参照してください。

3.36. GRPC

gRPC エンドポイントを公開し、外部 gRPC エンドポイントにアクセスします。

3.36.1. 含まれるもの

- gRPC コンポーネント、URI 構文： **grpc:host:port/service**

使用方法と設定の詳細は、上記リンクを参照してください。

3.36.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.36.3. 使用方法

3.36.3.1. Protobuf で生成されるコード

Camel Quarkus gRPC は、**.proto** ファイルの gRPC サービススタブを生成できます。Maven を使用する場合は、プロジェクトビルドで **quarkus-maven-plugin** の **generate-code** ゴールが有効になっていることを確認してください。

```
<build>
  <plugins>
    <plugin>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

この設定を使用すると、サービス定義とメッセージ定義を **src/main/proto** ディレクトリーに配置でき、**quarkus-maven-plugin** が **.proto** ファイルからコードを生成します。

3.36.3.1.1. インポートによる proto ファイルのスキャン

プロトコルバッファー仕様は、proto ファイルをインポートする方法を提供します。設定プロパティーの **quarkus.camel.grpc.codegen.scan-for-imports** プロパティーを **application.properties** に追加することで、スキャンする依存関係の範囲を制御できます。利用可能なオプションの概要を以下に示します。

- **all** - すべての依存関係をスキャンします。
- **none** - 依存関係のスキャンを無効にします。**src/main/proto** または **src/test/proto** で定義された proto 定義のみを使用します。
- **groupId1:artifactId1,groupId2:artifactId2** - **groupId** および **artifactId** リストに一致する依存関係のみをスキャンします。

デフォルト値は **com.google.protobuf:protobuf-java** です。

3.36.3.1.2. 依存関係からの proto ファイルのスキャン

複数の依存関係間で共有されている proto ファイルがある場合は、設定プロパティー **quarkus.camel.grpc.codegen.scan-for-proto** を **application.properties** に追加することで、そのファイルの gRPC サービススタブを生成できます。

まず、proto ファイルを含むアーティファクトの依存関係をプロジェクトに追加します。次に、proto ファイルの依存関係スキャンを有効にします。

```
quarkus.camel.grpc.codegen.scan-for-proto=org.my.groupId1:my-artifact-id-1,org.my.groupId2:my-artifact-id-2
```

設定プロパティーを使用して、特定の proto ファイルを依存関係スキャンに追加または除外することができます。

設定プロパティー名の接尾辞は、追加/除外を設定する依存関係の Maven **groupId**/**artifactId** です。パスは、依存関係に含まれる proto ファイルのクラスパスの場所への相対パスです。パスは、proto ファイルへの明示的なパス、または複数のファイルを追加/除外する glob パターンとして指定できます。

```
quarkus.camel.grpc.codegen.scan-for-proto-includes."<groupId>\:<artifactId>"=foo/**,bar/**,baz/a-proto.proto
quarkus.camel.grpc.codegen.scan-for-proto-excludes."<groupId>\:<artifactId>"=foo/private/**,baz/another-proto.proto
```

注記

プロパティーキー内の **:** 文字は \ でエスケープする必要があります。

3.36.3.2. ネイティブモードでのクラスパスリソースへのアクセス

gRPC コンポーネントには、リソースをクラスパスから解決するさまざまなオプションがあります。

- **keyCertChainResource**
- **keyResource**
- **serviceAccountResource**
- **trustCertCollectionResource**

これらのオプションをネイティブモードで使用する場合は、そのようなリソースがネイティブイメージに含まれていることを確認する必要があります。

これは、設定プロパティー **quarkus.native.resources.includes** を **application.properties** に追加することで実現できます。たとえば、SSL/TLS キーと証明書を含めるには、次のように指定します。

```
quarkus.native.resources.includes = certs/*.pem,certs.*.key
```

3.36.4. Camel Quarkus の制限

3.36.4.1. Quarkus gRPC との統合はサポート対象外

現時点では、Camel Quarkus gRPC と Quarkus gRPC の統合はサポートされていません。クラスパスに **camel-quarkus-grpc** と **quarkus-grpc** エクステンションの両方の依存関係がある場合、アプリケーションをコンパイルすると、ビルド時に問題が発生する可能性があります。

3.36.5. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.grpc.codegen.enabled **true** の場合、Camel Quarkus gRPC のコード生成が、**proto** ディレクトリー、または **scan-for-proto** または **scan-for-imports** オプションで指定された依存関係から検出された .proto ファイルに対して実行されます。**false** の場合、.proto ファイルのコード生成は無効になります。	boolean	true
quarkus.camel.grpc.codegen.scan-for-proto Camel Quarkus gRPC のコード生成が、.proto ファイルのアプリケーションの依存関係をスキャンして、そのファイルから Java スタブを生成できます。このプロパティーは、スキャンする依存関係の範囲を設定します。適用可能な値: - **none** - デフォルト - 依存関係をスキャンしない - スキャンする **groupId:artifactId** コーディネートのコンマ区切りのリスト - **all** - すべての依存関係をスキャンする	string	none
quarkus.camel.grpc.codegen.scan-for-imports Camel Quarkus gRPC のコード生成が、このアプリケーションの proto によってインポートできる .proto ファイルの依存関係をスキャンできます。適用可能な値: - **none** - デフォルト - 依存関係をスキャンしない - スキャンする **groupId:artifactId** コーディネートのコンマ区切りリスト - **all** - すべての依存関係をスキャンする (デフォルトは **com.google.protobuf:protobuf-java**)	string	com.google.protobuf:protobuf-java
quarkus.camel.grpc.codegen.scan-for-proto-includes パッケージパスまたはファイル glob パターンは、追加対象として考慮すべき .proto ファイルを含む依存関係ごとに追加します。	Map<String, List<String>>	

設定プロパティー	タイプ	デフォルト
quarkus.camel.grpc.codegen.scan-for-proto-excludes パッケージパスまたはファイル glob パターンは、除外対象として考慮すべき .proto ファイルを含む依存関係ごとに追加します。	Map<String, List<String>>	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.37. GSON

Gson を使用して、POJO を JSON に、およびその逆にマーシャリングします。

3.37.1. 含まれるもの

- JSON Gson データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.37.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.37.3. 追加の Camel Quarkus 設定

3.37.3.1. ネイティブモードでのオブジェクトのマーシャリング/アンマーシャリング

ネイティブモードでオブジェクトをマーシャリングまたはアンマーシャリングする場合、シリアライズされたクラスはすべて リフレクションのために登録 する必要があります。そのため、**GsonDataFormat.setUnmarshalType(...)**、**GsonDataFormat.setUnmarshalTypeName(...)**、さらに **GsonDataFormat.setUnmarshalGenericType(...)** を使用する場合は、アンマーシャリングタイプとサブフィールドタイプをリフレクション用に登録する必要があります。この インテグレーションテスト で実際の例を参照してください。

3.38. HL7

HL7 MLLP コーデックを使用して、HL7 (Health Care) モデルオブジェクトをマーシャリングおよびアンマーシャリングします。

3.38.1. 含まれるもの

- HL7 データフォーマット
- HL7 Terser 言語

使用方法と設定の詳細については、上記リンクを参照してください。

3.38.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-hl7</artifactId>
</dependency>
```

3.38.3. Camel Quarkus の制限

MLLP と TCP を使用する場合には、Netty が Hl7 MLLP リスナーを実行する唯一のサポートされる手段です。現在、GraalVM ネイティブサポートがないため、Mina はサポートされません。

HL7MLLPNettyEncoderFactory および **HL7MLLPNettyDecoderFactory** コーデックのオプションのサポートは、プロジェクト **pom.xml** の依存関係を **camel-quarkus-netty** に追加することで取得できます。

3.39. HTTP

Apache HTTP Client 5.x を使用して、外部の HTTP サーバーにリクエストを送信します。

3.39.1. 含まれるもの

- HTTP コンポーネント、URI 構文： **http://httpUri**
- HTTPS (Secure)コンポーネント、URI 構文： **https://httpUri**

使用方法と設定の詳細については、上記リンクを参照してください。

3.39.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.39.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.39.4. 追加の Camel Quarkus 設定

- アプリケーションがデフォルト以外のエンコーディングを使用してリクエストを送受信することが想定される場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.40. INFINISPAN

Infinispan の分散キー/値のストアとデータグリッドの読み取りと書き込みを行います。

3.40.1. 含まれるもの

- Infinispan コンポーネント、URI 構文： **infinispan:cacheName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.40.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.40.3. 追加の Camel Quarkus 設定

3.40.3.1. Infinispan クライアント設定

該当する Camel Infinispan コンポーネントおよびエンドポイントオプションで Infinispan クライアントを設定するか、Quarkus Infinispan エクステンション設定プロパティー を使用できます。

Quarkus Infinispan 設定プロパティーを使用することを選択した場合は、Camel Infinispan コンポーネントで検出できるようにするために、**RemoteCacheManager** のインジェクションポイントを **追加する必要がある** ことに注意してください。以下に例を示します。

```
public class Routes extends RouteBuilder {
    // Injects the default unnamed RemoteCacheManager
    @Inject
    RemoteCacheManager cacheManager;

    // If configured, injects an optional named RemoteCacheManager
    @Inject
    @InfinispanClientName("myNamedClient")
    RemoteCacheManager namedCacheManager;

    @Override
    public void configure() {
        // Route configuration here...
    }
}
```

3.40.3.2. ネイティブモードの Camel Infinispan InfinispanRemoteAggregationRepository

InfinispanRemoteAggregationRepository をネイティブモードで使用することを選択した場合は、ネイティブシリアライゼーションサポートを有効 にする必要があります。

3.41. AVRO JACKSON

Jackson を使用して、POJO を Avro にマーシャリングし、その逆も行います。

3.41.1. 含まれるもの

- Avro Jackson データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.41.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-avro</artifactId>
</dependency>
```

3.42. PROTOBUF JACKSON

Jackson を使用して、POJO を Protobuf にマーシャリングし、その逆も行います。

3.42.1. 含まれるもの

- Protobuf Jackson データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.42.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-protobuf</artifactId>
</dependency>
```

3.43. JACKSON

Jackson を使用して、POJO を JSON にマーシャリングし、その逆も行います。

3.43.1. 含まれるもの

- JSON Jackson データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.43.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.43.3. 使用方法

3.43.3.1. Jackson ObjectMapper の設定

JacksonDataFormat が使用する **ObjectMapper** を設定するには、いくつかの方法があります。これらの概要を以下に示します。

3.43.3.1.1. JacksonDataFormat によって内部的に作成された ObjectMapper

デフォルトでは、**JacksonDataFormat** は独自の **ObjectMapper** を作成し、**DataFormat** のさまざまな設定オプションを使用して、追加の Jackson モジュール、きれいな印刷、およびその他の機能を設定します。

3.43.3.1.2. JacksonDataFormat のカスタム ObjectMapper

次のように、カスタム **ObjectMapper** インスタンスを **JacksonDataFormat** に渡すことができます。

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        ObjectMapper mapper = new ObjectMapper();
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

3.43.3.1.3. JacksonDataFormat での Quarkus JacksonObjectMapper の使用

Quarkus Jackson エクステンションは、**JacksonDataFormat** によって検出できる **ObjectMapper** CDI Bean を公開します。

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        // Make JacksonDataFormat discover the Quarkus Jackson `ObjectMapper` from the Camel
registry
        dataFormat.setAutoDiscoverObjectMapper(true);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

Camel REST DSL で JSON バインディングモードを使用していて、Quarkus Jackson **ObjectMapper** を使用したい場合は、次のように実現できます。

```
import org.apache.camel.builder.RouteBuilder;

@ApplicationScoped
public class Routes extends RouteBuilder {
    public void configure() {
        restConfiguration().dataFormatProperty("autoDiscoverObjectMapper", "true");
        // REST definition follows...
    }
}
```

ObjectMapperCustomizer を使用して、Quarkus **ObjectMapper** でカスタマイズを実行できます。

```
import com.fasterxml.jackson.databind.ObjectMapper;
import io.quarkus.jackson.ObjectMapperCustomizer;

@Singleton
public class RegisterCustomModuleCustomizer implements ObjectMapperCustomizer {
    public void customize(ObjectMapper mapper) {
```

```
        mapper.registerModule(new CustomModule());
    }
}
```

Quarkus **ObjectMapper** を **@Inject** して、**JacksonDataFormat** に渡すこともできます。

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

@ApplicationScoped
public class Routes extends RouteBuilder {
    @Inject
    ObjectMapper mapper;

    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

3.44. JACKSONXML

Jackson の XMLMapper エクステンションを使用して、XML ペイロードを POJO にアンマーシャリングし、戻します。

3.44.1. 含まれるもの

- Jackson XML データ形式

使用方法と設定の詳細は、上記リンクを参照してください。

3.44.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.45. JASYPT

Jasypt を使用したセキュリティー

3.45.1. 含まれるもの

- Jasypt

使用方法と設定の詳細は、上記リンクを参照してください。

3.45.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.45.3. 使用方法

Camel Quarkus での Jasypt の設定は、設定プロパティーによって実行されます。

最小化は、設定プロパティー **quarkus.camel.jasypt.password** を使用して Jasypt 復号化のマスターパスワードを提供することです。

以下で説明する **quarkus.camel.jasypt** オプションを使用して、Jasypt 設定の暗号化アルゴリズムおよびその他の側面を選択できます。

デフォルトでは、Camel **JasyptPropertiesParser** または **PropertiesComponent** を設定するためにカスタムコードを記述する必要はありません。これは自動的に行われます。

application.properties に追加されたすべての Camel 設定プロパティーは、Jasypt で保護できます。値を暗号化するために、JBang で実行可能なユーティリティーがあります。

```
jbang org.apache.camel:camel-jasypt:{camel-version} -c encrypt -p secret-password -i "Some secret content"
```

重要

デフォルト(**PBEWithMD5AndDES**)に別の Jasypt アルゴリズムを使用することを選択した場合は、**-a** (algorithm)、**-riga** (IV generator algorithm)および **-rsga** (Salt ジェネレーターアルゴリズム)引数を指定して、暗号化で使用される正しいアルゴリズムを設定する必要があります。そうしないと、アプリケーションは設定値を復号化できなくなります。

または、dev モードで実行している場合は、Dev UI を開き、Camel Jasypt ペインでutilities リンクをクリックします。次に、'Decrypt' または 'Encrypt' アクションのいずれかを選択し、テキストを入力して送信ボタンをクリックします。アクションの結果が、クリップボードにコピーするためのボタンと一緒に出力されます。

設定プロパティーは、**ENC （）** で囲まれた暗号化された値で **application.properties** に追加できます。以下に例を示します。

```
my.secret = ENC(BoDSRQfdBME4V/AcugPOkaR+IcyKufGz)
```

Camel ルートでは、標準のプレースホルダー構文を使用してプロパティー名を参照できます。その値は復号化されます。

```
public class MySecureRoute extends RouteBuilder {
    @Override
    public void configure() {
        from("timer:tick?period=5s")
            .to("{{my.secret}}");
    }
}
```

ヒント

この機能を使用して、プロパティー値の接尾辞に **.secret** を付けることで、Camel のセキュリティー機密設定をマスクできます。また、設定 **camel.main.autoConfigurationLogSummary = false** を使用して起動設定の概要を無効にすることもできます。

3.45.3.1. 暗号化された設定の挿入

@ConfigProperty アノテーションを使用して、暗号化された設定を Camel ルートまたは CDI Bean に注入できます。

```
@ApplicationScoped
public class MySecureRoute extends RouteBuilder {
    @ConfigInject("my.secret")
    String mySecret;

    @Override
    public void configure() {
        from("timer:tick?period=5s")
            .to(mySecret);
    }
}
```

3.45.3.1.1. 代替設定ソースのセキュリティー保護

シークレット設定を **application.properties** とは別のファイルに維持する場合は、**quarkus.config.locations** 設定オプションを使用して追加の設定ファイルを指定できます。

ネイティブモードでは、追加の設定ファイルリソースパスを **quarkus.native.resources.includes** に追加する必要もあります。

3.45.3.1.2. Jasypt 設定をより細かく制御

デフォルト設定で提供されるものよりも Jasypt 設定をより細かく制御する必要がある場合は、以下のオプションを使用できます。

3.45.3.1.2.1. JasyptConfigurationCustomizer

JasyptConfigurationCustomizer クラスを実装して、Jasypt **EnvironmentStringPBEConfig** の要素をカスタマイズします。

```
package org.acme;

import org.apache.camel.quarkus.component.jasypt.JasyptConfigurationCustomizer;
import org.jasypt.encryption.pbe.config.EnvironmentStringPBEConfig;
```

```
import org.jasypt.iv.RandomIvGenerator;
import org.jasypt.salt.RandomSaltGenerator;

public class JasyptConfigurationCustomizer implements JasyptConfigurationCustomizer {
    public void customize(EnvironmentStringPBEConfig config) {
        // Custom algorithms
        config.setAlgorithm("PBEWithHmacSHA256AndAES_256");
        config.setSaltGenerator(new RandomSaltGenerator("PKCS11"));
        config.setIvGenerator(new RandomIvGenerator("PKCS11"));
        // Additional customizations...
    }
}
```

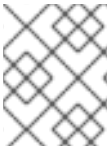
application.properties で、**quarkus.camel.jasypt.configuration-customizer-class-name** 設定プロパティーを追加します。

```
quarkus.camel.jasypt.configuration-customizer-class-name =
org.acme.MyJasyptEncryptorCustomizer
```

3.45.3.1.2.2. 自動 Jasypt 設定の無効化

Camel Jasypt を設定する Classic Java DSL 方式を使用する場合は、**quarkus.camel.jasypt.enabled = false** で自動設定を無効にすることができます。

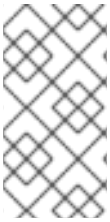
これにより、Camel **JasyptPropertiesParser** と **PropertiesComponent** を手動で設定できます。



注記

このモードでは、**@ConfigProperty** アノテーションを使用して暗号化された設定プロパティーを注入することはできません。

```
import org.apache.camel.CamelContext;
import org.apache.camel.component.jasypt.JasyptPropertiesParser;
import org.apache.camel.component.properties.PropertiesComponent;

public class MySecureRoute extends RouteBuilder {
    @Override
    public void configure() {
        JasyptPropertiesParser jasypt = new JasyptPropertiesParser();
        jasypt.setPassword("secret");

        PropertiesComponent component = (PropertiesComponent)
getContext().getPropertiesComponent();
        jasypt.setPropertiesComponent(component);
        component.setPropertiesParser(jasypt);

        from("timer:tick?period=5s")
            .to("{{my.secret}}");
    }
}
```



注記

PropertiesComponent で **setLocation (...)** を呼び出し、**classpath:** 接頭辞を使用してカスタム設定ファイルの場所を指定する場合は、ファイルをネイティブモードでロードできるように、ファイルを **quarkus.native.resources.includes** に追加する必要があります。

3.45.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.jasypt.enabled このオプションを false に設定すると、Quarkus SmallRye 設定との Jasypt の統合が無効になります。ただし、JasyptPropertiesParser および PropertiesComponent を手動で設定する従来の方法で、Camel を使用して Jasypt を手動で設定できます。詳細は 使用方法のセクションを参照してください。	**boolean**	**true**
quarkus.camel.jasypt.algorithm 復号化に使用するアルゴリズム。復号化に使用するアルゴリズム。	**string**	**PBEWithMD5AndDES**
quarkus.camel.jasypt.password 設定値を復号化するために Jasypt が使用するマスターパスワード。このオプションは、マスターパスワード検索の動作に影響を与える接頭辞をサポートします。 **sys:** JVM システムプロパティーから値を検索します。**sysenv:** 指定されたキーで OS システム環境から値を検索します。	**string**	
quarkus.camel.jasypt.random-iv-generator-algorithm 指定されたアルゴリズムを使用して Jasypt StandardPBEStringEncryptor を RandomIvGenerator で設定します。	**string**	**SHA1PRNG**
quarkus.camel.jasypt.random-salt-generator-algorithm 指定されたアルゴリズムを使用して Jasypt StandardPBEStringEncryptor を RandomSaltGenerator で設定します。	**string**	**SHA1PRNG**
quarkus.camel.jasypt.configuration-customizer-class-name org.apache.camel.quarkus.component.jasypt.JasyptConfigurationCustomizer 実装の完全修飾クラス名。これにより、Jasypt 設定を完全に制御するオプションの機能が提供されます。	**string**	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.45.5. Camel Quarkus の制限

camel-quarkus-jasypt エクステンションでは、システムプロパティーまたは環境変数から提供された場合に設定プロパティーを解決する際に問題が発生します。たとえば、**-Dquarkus.camel.jasypt.enabled=false** または **-Dquarkus.camel.jasypt.password=my-password** を渡すと機能しません。

これを回避するには、**application.properties** で **quarkus.camel.jasypt.enabled** を指定します。

Jasypt の無効化

```
quarkus.camel.jasypt.enabled = false
```

システムプロパティーまたは環境変数から **quarkus.camel.jasypt.password** を上書きするには、以下のように **application.properties** を設定します。

application.properties でパスワードをハードコーディングする方法

ハードコードされたパスワード

```
quarkus.camel.jasypt.password = my-password
```

または、sys または sysenv 接頭辞を使用して、システムプロパティーまたは環境変数からパスワードを解決することもできます。

sys 接頭辞が付いたパスワード

```
quarkus.camel.jasypt.password = sys:jasyptPassword
```

次に、任意のパスワードを使用してアプリケーションをビルドします。

パスワードを使用した構築

```
mvn clean package -DjasyptPassword=my-password
```

そして、アプリケーション JAR を実行する場合。

アプリケーション JAR

```
java -DjasyptPassword=my-password -jar target/quarkus-app/quarkus-run.jar
```

またはネイティブモードの場合：

ネイティブモード

```
target/my-native-application-runner -DjasyptPassword=my-password
```

3.46. JAVA JOOR DSL

実行時の Java ルート定義の解析のサポート

3.46.1. 含まれるもの

- Java DSL (runtime compiled)

使用方法と設定の詳細は、上記リンクを参照してください。

3.46.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-java-joor-dsl</artifactId>
</dependency>
```

3.46.3. Camel Quarkus の制限

コンポーネントによってコンパイルされるクラスに追加されたアノテーションは、Quarkus によって無視されます。このエクステンションで部分的にサポートされている唯一のアノテーションは、ネイティブモードのリフレクションの設定を容易にするアノテーション **RegisterForReflection** です。ただし、要素 **registerFullHierarchy** はサポートされていないことに注意してください。

3.47. JAXB

JAXB2 XML マーシャリング標準を使用して、XML ペイロードを POJO に、およびその逆にアンマーシャリングします。

3.47.1. 含まれるもの

- JAXB データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.47.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.47.3. 使用方法

3.47.3.1. 非 JAXB アノテーション付きクラスのネイティブモード ObjectFactory インスタンス化

JAXB アノテーションを持たない POJO クラスをインスタンス化するために、カスタム **ObjectFactory** で JAXB マーシャル操作を行う場合、それらの POJO クラスをネイティブモードでインスタンス化するために、リフレクションに登録する必要があります。たとえば、**@RegisterForReflection** アノテーションまたは設定プロパティー **quarkus.camel.native.reflection.include-patterns** を介した場合です。

詳細は、ネイティブモード のユーザーガイドを参照してください。

3.48. JDBC

SQL および JDBC を通じてデータベースにアクセスします。

3.48.1. 含まれるもの

- JDBC コンポーネント、URI 構文： **jdbc:dataSourceName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.48.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.48.3. 追加の Camel Quarkus 設定

3.48.3.1. データソースの設定

このエクステンションは、**DataSource** のサポートに Quarkus Agroal を活用します。**DataSource** の設定は、設定プロパティーを介して実行できます。JDBC エンドポイント URI で参照できるように、データソースの名前は明示的に付けることを推奨します。たとえば、**to("jdbc:camel")** のようになります。

```
quarkus.datasource.camel.db-kind=postgresql
quarkus.datasource.camel.username=your-username
quarkus.datasource.camel.password=your-password
quarkus.datasource.camel.jdbc.url=jdbc:postgresql://localhost:5432/your-database
quarkus.datasource.camel.jdbc.max-size=16
```

データソースに名前を付けないことを選択した場合は、**to("jdbc:default")** のようにエンドポイントを定義することでデフォルトの **DataSource** を解決できます。

3.48.3.1.1. Quarkus Dev Services によるゼロ設定

開発モードとテストモードでは、Configuration Free Databases を利用できます。必要なのは、ルート内でデフォルトのデータベースを参照することだけです。たとえば、**to("jdbc:default")** のようになります。

3.49. JIRA

JIRA 問題トラッカーと対話します。

3.49.1. 含まれるもの

- Jira コンポーネント、URI 構文： **jira:type**

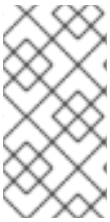
使用方法と設定の詳細は、上記リンクを参照してください。

3.49.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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



注記

camel-quarkus-jira エクステンションを使用するアプリケーションは、追加の Maven リポジトリーを Maven settings.xml ファイルまたはアプリケーションプロジェクトの pom.xml ファイルのいずれかで設定する必要があります。https://packages.atlassian.com/maven-external/

3.49.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.50. JMS

JMS Queue または Topic との間でメッセージを送受信します。

3.50.1. 含まれるもの

- JMS コンポーネント、URI 構文： **jms:destinationType:destinationName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.50.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

```
</dependency>
```

3.50.3. 使用方法

3.50.3.1. org.w3c.dom.Node を使用したメッセージマッピング

Camel JMS コンポーネントは、**jakarta.jms.Message** および **org.apache.camel.Message** 間のメッセージマッピングをサポートします。Camel メッセージ本文タイプ **org.w3c.dom.Node** を変換する場合は、**camel-quarkus-xml-jaxp** エクステンションがクラスパスに存在することを確認する必要があります。

3.50.3.2. jakarta.jms.ObjectMessage のネイティブモードのサポート

JMS メッセージペイロードを **jakarta.jms.ObjectMessage** として送信する場合、シリアル化のために登録する関連クラスに **@RegisterForReflection(serialization = true)** でアノテーションを付ける必要があります。

注記

このエクステンションは、**quarkus.camel.native.reflection.serialization-enabled = true** を自動的に設定します。詳細は、ネイティブモードのユーザーガイド を参照してください。

3.50.3.3. 接続プーリングと X/Open XA 分散トランザクションのサポート

注記

camel-quarkus-jms コンポーネントで接続プールを使用するには、**io.quarkiverse.artemis:quarkus-artemis** および **io.quarkiverse.messaginghub:quarkus-pooled-jms** を pom.xml に追加し、次の設定を指定する必要があります。

```
quarkus.pooled-jms.max-connections = 8
```

quarkus-pooled-jms エクステンションを使用して、JMS 接続のプーリングと XAのサポートを得ることができます。詳細は、quarkus-pooled-jms エクステンションドキュメントを参照してください。現在、このエクステンションは、**quarkus-artemis-jms**、**quarkus-qpid-jms**、および **ibmmq-client** で使用できます。依存関係を **pom.xml** に追加するだけです。

```
<dependency>
    <groupId>io.quarkiverse.messaginghub</groupId>
    <artifactId>quarkus-pooled-jms</artifactId>
</dependency>
```

プーリングはデフォルトで有効になっています。

注記

プーリング接続では、**clientID** と **durableSubscriptionName** はサポートされていません。プールから **reused** された接続で **setClientID** が呼び出されると、**IllegalStateException** が出力されます。次のようなエラーメッセージが表示されます。**Cause: setClientID can only be called directly after the connection is created**

XA を有効にするには、**quarkus-narayana-jta** エクステンションを追加する必要があります。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-narayana-jta</artifactId>
</dependency>
```

続いて、次の設定を **application.properties** に追加します。

```
quarkus.pooled-jms.transaction=xa
quarkus.transaction-manager.enable-recovery=true
```

XA サポートは、**quarkus-artemis-jms** および **ibmmq-client** でのみ利用可能です。

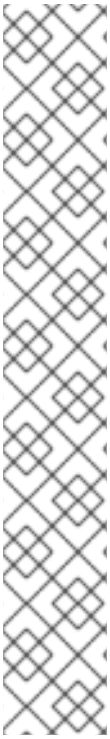
トランザクションリカバリー を有効にすることを強く推奨します。

現在、**ibmmq-client** 用の quarkus エクステンションは存在しないため、独自にカスタム **ConnectionFactory** を作成してラップする必要があります。

以下に例を示します。

ラッパーの例: ibmmq-client 用の ConnectionFactory

```
@Produces
public ConnectionFactory createXAConnectionFactory(PooledJmsWrapper wrapper) {
    MQXAConnectionFactory mq = new MQXAConnectionFactory();
    try {
        mq.setHostName(ConfigProvider.getConfig().getValue("ibm.mq.host", String.class));
        mq.setPort(ConfigProvider.getConfig().getValue("ibm.mq.port", Integer.class));
        mq.setChannel(ConfigProvider.getConfig().getValue("ibm.mq.channel", String.class));
        mq.setQueueManager(ConfigProvider.getConfig().getValue("ibm.mq.queueManagerName",
String.class));
        mq.setTransportType(WMQConstants.WMQ_CM_CLIENT);
        mq.setStringProperty(WMQConstants.USERID,
            ConfigProvider.getConfig().getValue("ibm.mq.user", String.class));
        mq.setStringProperty(WMQConstants.PASSWORD,
            ConfigProvider.getConfig().getValue("ibm.mq.password", String.class));
    } catch (Exception e) {
        throw new RuntimeException("Unable to create new IBM MQ connection factory", e);
    }
    return wrapper.wrapConnectionFactory(mq);
}
```

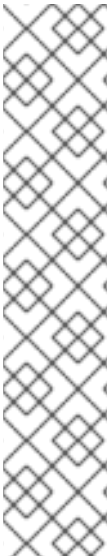


注記

ibmmq-client を使用してメッセージを消費し、XA を有効にする場合は、次のように camel ルートで **TransactionManager** を設定する必要があります。

```
@Inject
TransactionManager transactionManager;

@Override
public void configure() throws Exception {
    from("jms:queue:DEV.QUEUE.XA?transactionManager=#jtaTransactionManager");
}

@Named("jtaTransactionManager")
public PlatformTransactionManager getTransactionManager() {
    return new JtaTransactionManager(transactionManager);
}
```

そうでない場合は、**MQRC_SYNCPOINT_NOT_AVAILABLE** のような例外が発生します。



注記

ibmmq-client を使用してトランザクションをロールバックすると、以下のような WARN メッセージが表示されます。

```
WARN  [com.arj.ats.jta] (executor-thread-1) ARJUNA016045: attempted rollback of <
formatId=131077, gtrid_length=35, bqual_length=36,
tx_uid=0:ffffc0a86510:aed3:650915d7:16, node_name=quarkus,
branch_uid=0:ffffc0a86510:aed3:650915d7:1f, subordinatenodename=null,
eis_name=0 > (com.ibm.mq.jmqi.JmqiXAResource@79786dde) failed with exception
code XAException.XAER_NOTA: javax.transaction.xa.XAException: The method
'xa_rollback' has failed with errorCode '-4'.
```

それを無視して、MQ がトランザクションの作業を破棄したと仮定できます。詳細は、Red Hat ナレッジベース を参照してください。

3.50.4. ネイティブモードの transferException オプション

ネイティブモードで **transferException** オプションを使用するには、オブジェクトのシリアル化のサポートを有効にする必要があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

また、シリアル化する予定の例外クラスのシリアル化を有効にする必要があります。以下に例を示します。

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

3.51. JPA

Java Persistence API (JPA) を使用して、データベースから Java オブジェクトを保存し、取得します。

3.51.1. 含まれるもの

- JPA コンポーネント、URI 構文： **jpa:entityType**

使用方法と設定の詳細は、上記リンクを参照してください。

3.51.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.51.3. 追加の Camel Quarkus 設定

このエクステンションは、Quarkus Hibernate ORM を活用して、Hibernate 経由で JPA 実装を提供します。

Hibernate とデータソースを設定する方法については、Quarkus Hibernate ORM のドキュメントを参照してください。

また、Quarkus TX API を活用して **TransactionStrategy** 実装を提供します。

単一の永続ユニットが使用される場合、Camel Quarkus JPA エクステンションは、自動的に JPA コンポーネントを **EntityManagerFactory** および **TransactionStrategy** で設定します。

3.51.3.1. JpaMessageIdRepository の設定

CDI コンテナーの **EntityManagerFactory** と **TransactionStrategy** を使用して、**JpaMessageIdRepository** を設定する必要があります。

```
@Inject
EntityManagerFactory entityManagerFactory;

@Inject
TransactionStrategy transactionStrategy;

from("direct:idempotent")
    .idempotentConsumer(
        header("messageId"),
        new JpaMessageIdRepository(entityManagerFactory, transactionStrategy,
"idempotentProcessor"));
```

注記

spring-orm 依存関係が除外されるため、**sharedEntityManager**、**transactionManager** などの一部のオプションはサポートされません。

3.52. JSLT

JSLT を使用して JSON ペイロードをクエリーまたは変換します。

3.52.1. 含まれるもの

- JSLT コンポーネント, URI 構文： **jslt:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.52.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.52.3. ネイティブモードの allowContextMapAll オプション

allowContextMapAll オプションはネイティブモードではサポートされていません。これは、**CamelContext** や **Exchange** などのセキュリティーに敏感な Camel コアクラスへのリフレクティブアクセスが必要なためです。これはセキュリティーリスクとみなされるため、この機能へのアクセスはデフォルトでは提供されません。

3.52.4. 追加の Camel Quarkus 設定

3.52.4.1. ネイティブモードでクラスパスから JSLT テンプレートをロードする

通常、このコンポーネントはクラスパスからテンプレートをロードします。ネイティブモードでも機能させるには、**quarkus.native.resources.includes** プロパティーを使用して、テンプレートファイルをネイティブ実行可能ファイルに明示的に埋め込む必要があります。

たとえば、以下のルートは、**transformation.json** という名前のクラスパスリソースから JSLT スキーマを読み込みます。

```
from("direct:start").to("jslt:transformation.json");
```

これら (**.json** ファイルに保存されている可能性のある他のテンプレート) をネイティブイメージに含めるには、**application.properties** ファイルに次のようなものを追加する必要があります。

```
quarkus.native.resources.includes = *.json
```

3.52.4.2. ネイティブモードでの JSLT 関数の使用

camel-quarkus の JSLT 関数をネイティブモードで使用する場合、関数をホストするクラスを リフレクションに登録 する必要があります。ターゲット関数を登録できない場合、以下のようにスタブを書いてしまうことがあります。

```
@RegisterForReflection
public class MathFunctionStub {
```

```
    public static double pow(double a, double b) {
        return java.lang.Math.pow(a, b);
    }
}
```

ターゲット関数 **Math.pow(...)** は、以下のようにコンポーネントに登録できる **MathFunctionStub** クラスを介してアクセスできるようになりました。

```
@Named
JsltComponent jsltWithFunction() throws ClassNotFoundException {
    JsltComponent component = new JsltComponent();
    component.setFunctions(singleton(wrapStaticMethod("power",
"org.apache.cq.example.MathFunctionStub", "pow")));
    return component;
}
```

3.53. JSON PATH

JSON メッセージのボディーに対して、JSONPath 式を評価します。

3.53.1. 含まれるもの

- JSONPath 言語

使用方法と設定の詳細は、上記リンクを参照してください。

3.53.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.54. JTA

Java Transaction API (JTA) および Narayana トランザクションマネージャーを使用して、Camel ルートをトランザクションに含めます。

3.54.1. 含まれるもの

- JTA

使用方法と設定の詳細は、上記リンクを参照してください。

3.54.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.54.3. 使用方法

このエクステンションは、ルーターで **transacted()** EIP を使用する必要がある場合に追加する必要があります。これは、Quarkus の narayana-jta エクステンションによって提供されるトランザクション機能を利用します。

トランザクションサポートの詳細は、Quarkus Transaction guide を参照してください。簡単な使用方法の場合:

```
from("direct:transaction")
    .transacted()
    .to("sql:INSERT INTO A TABLE ...?dataSource=#ds1")
    .to("sql:INSERT INTO A TABLE ...?dataSource=#ds2")
    .log("all data are in the ds1 and ds2")
```

さまざまなトランザクションポリシーのサポートが提供されます。

ポリシー	説明
PROPAGATION_MANDATORY	現在のトランザクションをサポートします。現在のトランザクションが存在しない場合は例外が発生します。
PROPAGATION_NEVER	現在のトランザクションをサポートしません。現在のトランザクションが存在する場合は例外が発生します。
PROPAGATION_NOT_SUPPORTED	現在のトランザクションはサポートせず、常に非トランザクションを実行します。
PROPAGATION_REQUIRED	現在のトランザクションをサポートします。存在しない場合は新しいトランザクションを作成します。
PROPAGATION_REQUIRES_NEW	新しいトランザクションを作成し、現在のトランザクションが存在する場合はそれを一時停止します。
PROPAGATION_SUPPORTS	現在のトランザクションをサポートします。存在しない場合は、非トランザクションを実行します。

3.55. JT400

データキュー、メッセージキュー、またはプログラム呼び出しを使用して、IBM i システムとメッセージを交換します。IBM i は、AS/400 および iSeries サーバーの代替品です。

3.55.1. 含まれるもの

- JT400 コンポーネント, URI 構文：
  jt400:userID:password@systemName/QSYS.LIB/objectPath.type

使用方法と設定の詳細は、上記リンクを参照してください。

3.55.2. Maven コーディネート

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-jt400</artifactId>
</dependency>
```

警告

ネイティブ拡張を使用すると、以下のようなエラーが発生する可能性があります。

> Resource bundle lookup must be loaded during native image generation:

これは、ネイティブ登録がないために発生します
(https://github.com/apache/camel-quarkus/pull/6029)。

回避策として、複数のリソースバンドルを含めることができます。

```
quarkus.native.additional-build-args = -
H:IncludeResourceBundles=com.ibm.as400.access.JDMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.SVMRI_en,-
H:IncludeResourceBundles=com.ibm.as400.access.MRI2,-
H:IncludeResourceBundles=com.ibm.as400.access.JDMRI2,-
H:IncludeResourceBundles=com.ibm.as400.access.SVMRI,-
H:IncludeResourceBundles=com.ibm.as400.data.DAMRI,-
H:IncludeResourceBundles=com.ibm.as400.security.SecurityMRI,-
H:IncludeResourceBundles=com.ibm.as400.util.commtrace.CTMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.CoreMRI,-
H:IncludeResourceBundles=com.ibm.as400.resource.ResourceMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.MRI
```

3.56. KAFKA

Apache Kafka ブローカーとの間でメッセージを送受信します。

3.56.1. 含まれるもの

- Kafka コンポーネント、URI 構文： **kafka:topic**

使用方法と設定の詳細は、上記リンクを参照してください。

3.56.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.56.3. 使用方法

3.56.3.1. Quarkus Kafka Dev Services

Camel Quarkus Kafka は、Quarkus Kafka Dev services を利用して、ローカルのコンテナー化された Kafka ブローカーでの開発およびテストを簡素化できます。

デフォルトで、Kafka Dev Services は開発およびテストモードで有効になっています。Camel Kafka コンポーネントは、**brokers** コンポーネントのオプションがローカルのコンテナー化された Kafka ブローカーを参照するように自動的に設定されます。つまり、このオプションを独自に設定する必要はありません。

この機能は、設定プロパティー **quarkus.kafka.devservices.enabled=false** を使用して無効にできます。

3.56.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.kafka.kubernetes-service-binding.merge-configuration **true** の場合、Quarkus Kubernetes Service Binding エクステンション (設定されている場合) によって検出された Kafka 設定プロパティーは、Camel Kafka コンポーネントまたはエンドポイントオプションを介して設定されたプロパティーとマージされます。**false** の場合、Quarkus Kubernetes Service Binding エクステンションによって検出された Kafka 設定プロパティーはすべて無視され、Kafka コンポーネント設定はすべて Camel によって駆動されます。	**boolean**	**true**

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.57. KAMELET

ルートテンプレートを具体化する

3.57.1. 含まれるもの

- Kamelet コンポーネント、URI 構文： **kamelet:templateId/routeId**

使用方法と設定の詳細は、上記リンクを参照してください。

3.57.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.57.3. 使用方法

3.57.3.1. ビルド時に Kamelets をプリロードします

このエクステンションを使用すると、**quarkus.camel.kamelet.identifiers** プロパティーを使用して、ビルド時に一連の Kamelet をプリロードできます。

3.57.3.2. Kamelet Catalog の使用

/camel-kamelets/latest[Kamelet Catalog] には事前に設定された Kamelets のセットがあります。カタログから Kamelet を使用するには、クラスパスのプロジェクトの yaml 定義 (camel-kamelet リポジトリー にあります) をコピーする必要があります。または、**camel-kamelets-catalog** アーティファクトを **pom.xml** に追加できます。

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

このアーティファクトは、カタログで使用可能なすべての kamelets を Camel Quarkus アプリケーションに追加してビルドタイム処理を行います。アーティファクトがランタイムではなくビルドタイムクラスパスの一部でなければならないという条件 (**provided**) でスコープに含める場合、**quarkus.camel.kamelet.identifiers** プロパティーを介してリストされたすべての kamelets を事前にロードする必要があります。

3.57.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.kamelet.identifiers ビルド時にプリロードする kamelets 識別子のリスト。 個々の識別子は、関連する **org.apache.camel.model.RouteTemplateDefinition** id を設定するために使用されます。	**string**	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.58. KUBERNETES

Kubernetes API に対して操作を実行する

3.58.1. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.58.2. 追加の Camel Quarkus 設定

重要

Red Hat build of Apache Camel for Quarkus のこのリリースでは、**camel-quarkus-kubernetes** エクステンションは、クラスターサービスとして **camel-quarkus-master** エクステンションと併用する場合にのみサポートされます。さらに、**camel-quarkus-kubernetes** エクステンションをサポートするには、アプリケーションで **quarkus-openshift-client** エクステンションへの依存関係を明示的に追加する必要があります。

3.58.2.1. Kubernetes クライアントインスタンスの自動登録

エクステンションは、**kubernetesClient** という名前の Kubernetes クライアント Bean を自動的に登録します。次のように、ルートで Bean を参照できます。

```
from("direct:pods")
    .to("kubernetes-pods:///?kubernetesClient=#kubernetesClient&operation=listPods")
```

デフォルトでは、クライアントはローカルの kubeconfig ファイルから設定されます。**application.properties** 内のプロパティーを使用して、クライアント設定をカスタマイズできます。

```
quarkus.kubernetes-client.master-url=https://my.k8s.host
quarkus.kubernetes-client.namespace=my-namespace
```

設定オプションの完全なセットは、Quarkus Kubernetes Client ガイド に記載されています。

3.58.2.2. 特定のエンドポイントから消費するコンシューマがクラスタ内で 1 つだけであること

同じルートが複数の Pod にデプロイされている場合、このエクステンションを マスターのエクステンション と組み合わせて使用すると便利な場合があります。このようなセットアップでは、キャメルマスター namespace 全体で一度に 1 つのコンシューマーがアクティブになります。

たとえば、以下のルートを複数の Pod にデプロイするとします。

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single pod at a time");
```

以下のようなプロパティーを使用して、kubernetes クラスターサービスを有効にすることができます。

```
quarkus.camel.cluster.kubernetes.enabled = true
```

その結果、1 つのコンシューマーが **ns** camel マスター namespace 全体でアクティブになります。これは、特定の時点で、単一のタイマーのみがクラスター全体でエクスチェンジを生成することを意味します。つまり、メッセージは一度に 1 つの Pod で 100 ミリ秒ごとにログに記録されます。

kubernetes クラスターサービスは、**quarkus.camel.cluster.kubernetes.*** プロパティーをさらに微調整できます。

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.kubernetes.enabled quarkus.camel.cluster.kubernetes.* 設定に従って、Kubernetes クラスターサービスを自動的に設定する必要があるかどうか。	**boolean**	**false**
quarkus.camel.cluster.kubernetes-id クラスターサービス ID (デフォルトは null)。	**string**	
quarkus.camel.cluster.kubernetes.master-url Kubernetes マスターの URL (デフォルトでは Kubernetes クライアントプロパティーから読み取られます)。	**string**	
quarkus.camel.cluster.kubernetes.connection-timeout-millis Kubernetes API サーバーにリクエストを送信するときに使用する接続タイムアウト (ミリ秒単位)。	**java.lang.Integer**	

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.kubernetes.namespace Pod と configmap を含む Kubernetes namespace の名前 (デフォルトで自動検出)。	string	
quarkus.camel.cluster.kubernetes.pod-name 現在の Pod の名前 (デフォルトではコンテナーのホスト名から自動検出されます)。	string	
quarkus.camel.cluster.kubernetes.jitter-factor すべての Pod が同じ瞬間に Kubernetes API を呼び出さないようにするために適用するジッター係数 (デフォルトは 1.2)。	java.lang.Double	
quarkus.camel.cluster.kubernetes.lease-duration-millis 現在のリーダーのデフォルトのリース期間 (デフォルトは 15000)。	java.lang.Long	

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.kubernetes.renew-deadline-millis リーダーシップを失った可能性があるため、リーダーがサービスを停止しなければならない期限 (デフォルトは 10000)。	java.lang.Long	
quarkus.camel.cluster.kubernetes.retry-period-millis リーダーシップを確認して獲得するための後続の 2 つの試みの間の時間。ジッター係数を使用してランダム化されます (デフォルトは 2000)。	java.lang.Long	
quarkus.camel.cluster.kubernetes-order サービス検索の順序/優先度 (デフォルトは 2147482647)。	java.lang.Integer	
quarkus.camel.cluster.kubernetes.resource-name 楽観的ロックを行うために使用されるリースリソースの名前 (デフォルトは leaders)。リソース名は、基になる Kubernetes リソースが単一のロックを管理できる場合に接頭辞として使用されます。	string	

設定プロパティー	タイプ	デフォルト
camel.cluster.kubernetes.lease-resource-type Kubernetes で使用されるリースリソースタイプ。config-map または lease のいずれかです (デフォルトは lease)。	org.apache.camel.component.kubernetes.cluster.LeaseResource	

設定プロパティー	タイプ	デフォルト
quarkus.camel.cluster.kubernetes.rebalancing camel マスター namespace リーダーをクラスター内のすべての camel コンテキストに均等に分散する必要があるかどうか。	boolean	true
quarkus.camel.cluster.kubernetes-labels クラスターを設定する Pod を識別するために使用されるラベルのキー/値。デフォルトは空のマップです。	Map<String,String>	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.59. KUDU

Apache Hadoop エコシステムの無料およびオープンソースの列指向データストアである Apache Kudu と対話します。

3.59.1. 含まれるもの

- Kudu コンポーネント、URI 構文： **kudu:host:port/tableName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.59.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
```

```
    <artifactId>camel-quarkus-kudu</artifactId>
</dependency>
```

3.59.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

注記

Kudu は IBM Z および IBM Power ではサポートされていません。

3.60. LANGUAGE

Camel がサポートする任意の言語でスクリプトを実行します。

3.60.1. 含まれるもの

- Language コンポーネント, URI 構文： **language:languageName:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.60.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.60.3. 使用方法

3.60.3.1. 必要な依存関係

言語エクステンションは、実行するスクリプトに Exchange を渡すことのみを処理します。言語を実装するエクステンションは依存関係として追加する必要があります。次のリストの言語が Core に実装されています。

- Constant
- ExchangeProperty
- File
- Header
- Ref
- Simple

- Tokenize

他の言語を使用するには、対応する依存関係を追加する必要があります。詳細は、言語ガイド を参照してください。

3.60.3.2. ネイティブモード

ネイティブモードでクラスパスからスクリプトをロードする場合、スクリプトファイルへのパスを **application.properties** ファイルの **quarkus.native.resources.includes** プロパティーで指定する必要があります。以下に例を示します。

```
quarkus.native.resources.includes=script.txt
```

3.60.4. ネイティブモードの allowContextMapAll オプション

allowContextMapAll オプションはネイティブモードではサポートされていません。これは、**CamelContext** や **Exchange** などのセキュリティーに敏感な Camel コアクラスへのリフレクティブアクセスが必要なためです。これはセキュリティーリスクとみなされるため、この機能へのアクセスはデフォルトでは提供されません。

3.61. LDAP

LDAP サーバーで検索を実行します。

3.61.1. 含まれるもの

- LDAP コンポーネント、URI 構文： **ldap:dirContextName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.61.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.61.3. 使用方法

3.61.3.1. ネイティブモードでの SSL の使用

カスタム **SSLSocketFactory** をネイティブモードで使用する場合(SSL の設定 セクションのものなど)、リフレクション用にクラスを登録する必要があります。登録しないと、クラスパスでクラスが利用可能になりません。次のように、クラス定義の上に **@RegisterForReflection** アノテーションを追加します。

```
@RegisterForReflection
public class CustomSSLSocketFactory extends SSLSocketFactory {
```

```
    // The class definition is the same as in the above link.
}
```

3.62. LRA

Long-Running-Action フレームワークの Camel saga バインディング

3.62.1. 含まれるもの

- LRA

使用方法と設定の詳細は、上記リンクを参照してください。

3.62.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.63. LOG

基礎となるロギングメカニズムにメッセージをログとして記録します。

3.63.1. 含まれるもの

- Log コンポーネント、URI 構文： **log:loggerName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.63.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.64. MAIL

imap、pop3、および smtp プロトコルを使用してメールを送受信します。添付のある Camel メッセージを MIME-Multipart メッセージに、またはその逆にマーシャリングします。

3.64.1. 含まれるもの

- IMAP コンポーネント、URI 構文： **imap:host:port**
- IMAPS (Secure)コンポーネント、URI 構文： **imaps:host:port**
- MIME Multipart データ形式
- POP3 コンポーネント、URI 構文： **pop3:host:port**
- POP3S コンポーネント、URI 構文： **pop3s:host:port**
- SMTP コンポーネント、URI 構文： **smtp:host:port**
- SMTPS コンポーネント、URI 構文： **smtps:host:port**

使用方法と設定の詳細については、上記リンクを参照してください。

3.64.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.65. 管理

JMX 管理ストラテジーと関連する管理リソース。

3.65.1. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.65.2. 使用方法

Camel で管理対象 Bean を使用する方法については、Camel Manual の JMX セクション を参照してください。

3.65.2.1. JMX の有効化と無効化

JMX は、次のいずれかの方法で Camel-Quarkus で有効または無効にすることができます。

1. **camel-quarkus-management** エクステンションを追加または削除する。
2. **camel.main.jmxEnabled** 設定プロパティーをブール値に設定する。

3. システムプロパティー **-Dorg.apache.camel.jmx.disabled** をブール値に設定する。

3.65.2.2. ネイティブモード

JDK 17/20/Mandrel 23.0 の GraalVM のネイティブ実行可能ファイルに **実験的な** JMX サポートが追加されました。この機能を有効にするには、以下の設定プロパティーを **application.properties** に追加します。

```
quarkus.native.monitoring=jmxserver
```

ネイティブアプリケーションを JConsole や VisualVM などのツールで検出できるようにするには、上記の設定に **jvmstat** オプションを追加します。

詳細は、Quarkus native guide を参照してください。

3.66. MAPSTRUCT

Mapstruct を使用した型変換

3.66.1. 含まれるもの

- Mapstruct コンポーネント、URI 構文： **mapstruct:className**

使用方法と設定の詳細は、上記リンクを参照してください。

3.66.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.66.3. 使用方法

3.66.3.1. アノテーションプロセッサー

MapStruct を使用するには、アノテーションプロセッサーを使用するようにビルドを設定する必要があります。

3.66.3.1.1. Maven

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
            <annotationProcessorPaths>
                <path>
```

```
                <groupId>org.mapstruct</groupId>
                <artifactId>mapstruct-processor</artifactId>
                <version>{mapstruct-version}</version>
            </path>
        </annotationProcessorPaths>
    </configuration>
  </plugin>
</plugins>
```

3.66.3.1.2. Gradle

```
dependencies {
    annotationProcessor 'org.mapstruct:mapstruct-processor:{mapstruct-version}'
    testAnnotationProcessor 'org.mapstruct:mapstruct-processor:{mapstruct-version}'
}
```

3.66.3.2. マッパー定義の検出

デフォルトでは、Red Hat build of Apache Camel for Quarkus は、**@Mapper** アノテーション付きインターフェイスまたは抽象クラスのパッケージパスを自動的に検出し、それらを Camel MapStruct コンポーネントに渡します。

スキャンされる特定のパッケージをより細かく制御する必要がある場合は、**application.properties** で設定プロパティーを設定できます。

```
camel.component.mapstruct.mapper-package-name = com.first.package,org.second.package
```

3.67. MASTER

クラスター内の単一のコンシューマーのみが特定のエンドポイントから消費するようにします。JVM が停止した場合に自動的にフェイルオーバーします。

3.67.1. 含まれるもの

- Master コンポーネント、URI 構文： **master:namespace:delegateUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.67.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.67.3. 追加の Camel Quarkus 設定

このエクステンションは、以下のエクステンションと組み合わせて使用できます。

- Camel Quarkus ファイル
- Camel Quarkus Kubernetes

3.68. MICROMETER

Micrometer ライブラリーを使用して、Camel ルートからさまざまなメトリクスを直接収集します。

3.68.1. 含まれるもの

- Micrometer コンポーネント、URI 構文： **micrometer:metricsType:metricsName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.68.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.68.3. 使用方法

このエクステンションは Quarkus Micrometer を活用しています。Quarkus は、さまざまな Micrometer メトリクスレジストリー実装をサポートしています。

アプリケーションでは、使用する監視ソリューションに応じて、以下の依存関係、または quarkiverse ドキュメント に記載されている依存関係の 1 つを宣言する必要があります。

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

依存関係が宣言されていない場合、Micrometer エクステンションは主にテストに適した **SimpleMeterRegistry** インスタンスを作成します。

3.68.4. Camel Quarkus の制限

3.68.4.1. JMX での Micrometer 統計情報の公開

quarkus-micrometer-registry-jmx には現在ネイティブサポートがないため、JMX での Micrometer 統計の公開はネイティブモードでは利用できません。

3.68.4.2. Counter のデクリメントヘッダーは Prometheus によって無視されます

Prometheus バックエンドは、Counter メトリクスのインクリメント中に負の値を無視します。

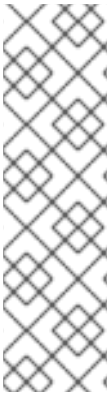
3.68.4.3. JMX での統計の公開

Red Hat build of Apache Camel for Quarkus では、**JmxMeterRegistry** の登録が簡素化されています。**io.quarkiverse.micrometer.registry:quarkus-micrometer-registry-jmx** の依存関係を追加すると、**JmxMeterRegistry** が自動的に作成されます。

3.68.5. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.metrics.enable-route-policy ルート処理時間のメトリクスをキャプチャーするために MicrometerRoutePolicyFactory を有効化するかどうかを設定します。	boolean	true
quarkus.camel.metrics.enable-message-history 個々のルートノード処理時間のメトリクスをキャプチャーするために、MicrometerMessageHistoryFactory を有効化するかどうかを設定します。設定されたルートノードの数によっては、大量のメトリクスが作成される可能性があります。したがって、このオプションはデフォルトで無効になります。	boolean	false
quarkus.camel.metrics.enable-exchange-event-notifier エクスチェンジ処理時間のメトリクスをキャプチャーするために、MicrometerExchangeEventNotifier を有効化するかどうかを設定します。	boolean	true
quarkus.camel.metrics.enable-route-event-notifier ルートの合計数と実行中のルートの合計数のメトリクスをキャプチャーするために、MicrometerRouteEventNotifier を有効化するかどうかを設定します。	boolean	true
quarkus.camel.metrics.enable-instrumented-thread-pool-factory InstrumentedThreadPoolFactory を注入して Camel Thread Pool に関するパフォーマンス情報を収集するかどうかを設定します。	boolean	false

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。



注記

smallrye-metrics から **micrometer** に移行する場合は、一部の Bean をスコープ指定された Bean として手動で定義する必要がある場合があります。

smallrye-metrics では、メトリクス (**@COUNTED**、**@METRIC** など) として登録されているが、スコープ指定された Bean として登録されていないクラスが、自動的に登録されます。**micrometer** ではこのようなことは起こりません。

micrometer では、**@Dependent** アノテーションを追加するなどして、CDI を通じてアクセスされる Bean を手動で登録する必要があります。

3.69. MICROPROFILE フォールトトレランス

Microprofile フォールトトレラン を使用した Circuit Breaker EIP

3.69.1. 含まれるもの

- MicroProfile フォールトトレランス

使用方法と設定の詳細は、上記リンクを参照してください。

3.69.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-fault-tolerance</artifactId>
</dependency>
```

3.70. MICROPROFILE HEALTH

MicroProfile Health による Camel ヘルスチェックの公開

3.70.1. 含まれるもの

- Microprofile Health

使用方法と設定の詳細は、上記リンクを参照してください。

3.70.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-microprofile-health</artifactId>
</dependency>
```

3.70.3. 使用方法

デフォルトでは、**AbstractHealthCheck** を拡張するクラスは liveness および readiness チェックの両方として登録されます。**isReadiness** メソッドを上書きして、この動作を制御できます。

アプリケーションによって提供されるチェックは自動的に検出され、Camel レジストリーにバインドされます。これらは、Quarkus ヘルスエンドポイント **/q/health/live** および **/q/health/ready** から利用できます。

カスタムの **HealthCheckRepository** 実装も提供でき、これらの実装も自動的に検出され、Camel レジストリーにバインドされます。

詳細は、Quarkus health guide を参照してください。

3.70.3.1. 提供されるヘルスチェック

一部のチェックはアプリケーションに自動的に登録されます。

3.70.3.1.1. Camel Context Health

Camel Context のステータスを検査して、ステータスが Started 以外の場合にヘルスチェックのステータスを **DOWN** にします。

3.70.3.1.2. Camel Route Health

各ルートのステータスを検査して、いずれかのルートのステータスが Started 以外の場合にヘルスチェックのステータスを **DOWN** にします。

3.70.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.health.enabled Camel ヘルスチェックを有効にするかどうかを設定します	**boolean**	**true**

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.71. MINIO

Minio SDK を使用して、Minio Storage Service からオブジェクトを保存および取得します。

3.71.1. 含まれるもの

- Minio コンポーネント、URI 構文： **minio:bucketName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.71.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.71.3. 追加の Camel Quarkus 設定

Minio の設定によっては、このエクステンションでは接続に SSL 暗号化が必要になる場合があります。このような場合、**quarkus.ssl.native=true** を **application.properties** に追加する必要があります。Quarkus native SSL guide および Camel Quarkus ユーザーガイドの ネイティブモード セクションも参照してください。

次の 2 つの異なる設定アプローチがあります。

- Minio クライアントは、Quarkiverse Minio を利用する quarkus プロパティーを通じて定義できます (ドキュメント を参照)。Camel はクライアントを Minio コンポーネントに自動接続します。この設定では、minio クライアントを 1 つだけ定義できるため、一緒に実行される、異なる minio エンドポイントを複数定義できません。
- Camel レジストリーのクライアント (CDI プロデューサ/Bean など) を提供し、エンドポイントから参照します。

```
minio:foo?minioClient=#minioClient
```

3.72. MLLP

MLLP プロトコルを使用して外部システムと通信します。

3.72.1. 含まれるもの

- MLLP コンポーネント, URI 構文： **mllp:hostname:port**

使用方法と設定の詳細は、上記リンクを参照してください。

3.72.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.72.3. 追加の Camel Quarkus 設定

- **defaultCharset** コンポーネントオプションを使用する場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.73. MOCK

モックを使用してルートおよび仲介ルールをテストします。

3.73.1. 含まれるもの

- Mock コンポーネント、URI 構文： **mock:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.73.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.73.3. 使用方法

テストで camel-mock 機能を使用するには、MockEndpoint インスタンスへのアクセスを取得する必要があります。

CDI の注入は、インスタンスへのアクセスに使用できます (Quarkus ドキュメント を参照してください)。**@Inject** アノテーションを使用して camelContext をテストに注入できます。その後、Camel コンテキストを使用してモックエンドポイントを取得できます。以下の例を参照してください。

```
import jakarta.inject.Inject;

import org.apache.camel.CamelContext;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.component.mock.MockEndpoint;
import org.junit.jupiter.api.Test;

import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class MockJvmTest {

    @Inject
```

```
    CamelContext camelContext;

    @Inject
    ProducerTemplate producerTemplate;

    @Test
    public void test() throws InterruptedException {

        producerTemplate.sendBody("direct:start", "Hello World");

        MockEndpoint mockEndpoint = camelContext.getEndpoint("mock:result", MockEndpoint.class);
        mockEndpoint.expectedBodiesReceived("Hello World");

        mockEndpoint.assertIsSatisfied();
    }
}
```

サンプルテストに使用するルート:

```
import jakarta.enterprise.context.ApplicationScoped;

import org.apache.camel.builder.RouteBuilder;

@ApplicationScoped
public class MockRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start").to("mock:result");
    }
}
```

3.73.4. Camel Quarkus の制限

(使用法で説明した) CDI Bean の注入は、ネイティブモードでは機能しません。

ネイティブモードでは、テストとテスト中のアプリケーションが 2 つの異なるプロセスで実行され、それらの間でモック Bean を共有することはできません (Quarkus ドキュメント を参照)。

3.74. MONGODB

MongoDB ドキュメントおよびコレクションの操作を実行します。

3.74.1. 含まれるもの

- MongoDB コンポーネント、URI 構文： **mongodb:connectionBean**

使用方法と設定の詳細は、上記リンクを参照してください。

3.74.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.74.3. 追加の Camel Quarkus 設定

エクステンションは Quarkus MongoDB Client エクステンションを活用します。Mongo クライアントは、Quarkus MongoDB Client の 設定オプション を使用して設定できます。

Camel Quarkus MongoDB エクステンションは、**camelMongoClient** という名前の MongoDB クライアント Bean を自動的に登録します。これは、mongodb エンドポイント URI の **connectionBean** パスパラメーターで参照できます。以下に例を示します。

```
from("direct:start")
.to("mongodb:camelMongoClient?database=myDb&collection=myCollection&operation=findAll")
```

アプリケーションが複数の MongoDB サーバーと連携する必要がある場合は、Quarkus MongoDB extension client injection で説明するように、特定の名前のクライアントを作成し、クライアントおよび関連する設定を注入することで、ルートで参照できます。以下に例を示します。

```
//application.properties
quarkus.mongodb.mongoClient1.connection-string = mongodb://root:example@localhost:27017/
```

```
//Routes.java

  @ApplicationScoped
  public class Routes extends RouteBuilder {
    @Inject
    @MongoClientName("mongoClient1")
    MongoClient mongoClient1;

    @Override
    public void configure() throws Exception {
      from("direct:defaultServer")
        .to("mongodb:camelMongoClient?
database=myDb&collection=myCollection&operation=findAll")

      from("direct:otherServer")
        .to("mongodb:mongoClient1?
database=myOtherDb&collection=myOtherCollection&operation=findAll");
    }
  }
```

指定されたクライアントを使用する場合、デフォルトの **camelMongoClient** Bean は引き続き生成されます。詳細は、複数の MongoDB クライアント に関する Quarkus ドキュメントを参照してください。

3.75. MYBATIS

MyBatis を使用して、リレーショナルデータベースでクエリー、ポーリング、挿入、更新、または削除を実行します。

3.75.1. 含まれるもの

- MyBatis コンポーネント、URI 構文： **mybatis:statement**
- MyBatis Bean コンポーネント、URI 構文： **mybatis-bean:beanName:methodName**

使用方法と設定の詳細については、上記リンクを参照してください。

3.75.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.75.3. 追加の Camel Quarkus 設定

設定については、Quarkus MyBatis を参照してください。以下のオプションを有効にする必要があります。

```
quarkus.mybatis.xmlconfig.enable=true
quarkus.mybatis.xmlconfig.path=SqlMapConfig.xml
```

ヒント

quarkus.mybatis.xmlconfig.path は mybatis エンドポイントの **configurationUri** パラメーターと同じである必要があります。

3.76. NETTY HTTP

Netty HTTP エクステンションは、Netty エクステンションに加えて HTTP トランスポートを提供します。

3.76.1. 含まれるもの

- Netty HTTP コンポーネント、URI 構文： **netty-http:protocol://host:port/path**

使用方法と設定の詳細は、上記リンクを参照してください。

3.76.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-netty-http</artifactId>
</dependency>
```

3.76.3. ネイティブモードの transferException オプション

ネイティブモードで **transferException** オプションを使用するには、オブジェクトのシリアル化のサポートを有効にする必要があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

また、シリアル化する予定の例外クラスのシリアル化を有効にする必要があります。以下に例を示します。

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

3.76.4. 追加の Camel Quarkus 設定

- アプリケーションがデフォルト以外のエンコーディングを使用してリクエストを送受信することが想定される場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.77. NETTY

Netty 4.x で TCP または UDP を使用するソケットレベルのネットワーク。

3.77.1. 含まれるもの

- Netty コンポーネント、URI 構文： **netty:protocol://host:port**

使用方法と設定の詳細は、上記リンクを参照してください。

3.77.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.78. OPENAPI JAVA

Camel REST DSLL で定義された OpenAPI リソースを公開する

3.78.1. 含まれるもの

- Openapi Java

使用方法と設定の詳細は、上記リンクを参照してください。

3.78.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-openapi-java</artifactId>
</dependency>
```

3.78.3. 使用方法

このエクステンションを使用して、REST DSL サービスを Quarkus OpenAPI に公開できます。**quarkus-smallrye-openapi** を使用すると、**/q/openapi?format=json** でアクセスできます。

詳細については、Quarkus OpenAPI ガイド を参照してください。

これは実験的な機能です。それを有効にすることができます

```
quarkus.camel.openapi.expose.enabled=true
```

警告

@RegisterForReflection を使用してすべてのモデルクラスをリフレクションに登録するのは、ユーザーの責任です。

現在、**org.apache.camel.builder.LambdaRouteBuilder** で使用されている残りのサービスはサポートされていません。また、CDI が利用できない間にビルド時に残りの定義を取得するため、RouteBuilder の **configure()** で CDI 注入を使用することはできません。

3.79. OPENTELEMETRY

OpenTelemetry を使用した分散トレース

3.79.1. 含まれるもの

- OpenTelemetry

使用方法と設定の詳細は、上記リンクを参照してください。

3.79.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.79.3. 使用方法

エクステンションは Camel **OpenTelemetryTracer** を自動的に作成し、それを Camel レジストリーにバインドします。

キャプチャされたトレースをトレースシステムに送信するには、以下のような **application.properties** 内のいくつかのプロパティーを設定する必要があります。

```
# Identifier for the origin of spans created by the application
quarkus.application.name=my-camel-application

# OTLP exporter endpoint
quarkus.opentelemetry.tracer.exporter.otlp.endpoint=http://localhost:4317
```

設定オプションの完全なリストは、Quarkus OpenTelemetry ガイド を参照してください。

application.properties で **quarkus.camel.opentelemetry.exclude-patterns** という名前のプロパティーを設定することで、ルートエンドポイントをトレースから除外できます。以下に例を示します。

```
# Exclude all direct & netty-http endpoints from tracing
quarkus.camel.opentelemetry.exclude-patterns=direct:*,netty-http:*
```

3.79.3.1. エクスポーター

Quarkus OpenTelemetry のデフォルトは、OpenTelemetry で定義された標準の OTLP エクスポーターです。追加のエクスポーターは、Quarkiverse quarkus-opentelemetry-exporter プロジェクトで利用可能になります。

3.79.3.2. CDI Bean メソッドの実行の追跡

Camel ルートから CDI Bean メソッドの実行をインストルメント化する場合、そのようなメソッドに **io.opentelemetry.extension.annotations.WithSpan** のアノテーションを付ける必要があります。**@WithSpan** アノテーションが付けられたメソッドは、新しい Span を作成し、現在のトレースコンテキストと必要な関係を確立します。

たとえば、Camel ルートから CDI Bean をインストルメント化するには、まず適切なメソッドに **@WithTrace** のアノテーションが付けられていることを確認します。

```
@ApplicationScoped
@Named("myBean")
public class MyBean {
    @WithSpan
    public String greet() {
        return "Hello World!";
    }
}
```

次に、Camel ルートで Bean を使用します。

重要

録画したスパンのシーケンスが正しいことを確認するには、短縮された **.bean()** EIP DSL メソッドではなく、完全な **to("bean:")** エンドポイント URI を使用する必要があります。

```
public class MyRoutes extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:executeBean")
                .to("bean:myBean?method=greet");
    }
}
```

CDI インストルメンテーションの詳細は、Quarkus OpenTelemetry ガイドを参照してください。

3.79.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.opentelemetry.encoding ヘッダー名をエンコードする必要があるかどうかを設定します。OpenTelemetry プロパゲーターが、ターゲットシステムと互換性のない形式でヘッダー名の値を設定する可能性がある状況で役立ちます。たとえば、JMS の場合、仕様ではヘッダー名が有効な Java 識別子であることが義務付けられています。	**boolean**	**false**
quarkus.camel.opentelemetry.exclude-patterns 指定されたコンマ区切りのパターンに一致するエンドポイント URI のトレースを無効にするかどうかを設定します。パターンは次の形式を取ることができます。 1.エンドポイント URI の完全一致。例: platform-http:/some/path 2.ワイルドカードマッチ。E.g platform-http:* 3.エンドポイント URI に一致する正規表現。例: platform-http:/prefix/.*	**string**	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.80. PAHO MQTT5

Eclipse Paho MQTT v5 クライアントを使用して MQTT メッセージブローカーと通信します。

3.80.1. 含まれるもの

- Paho MQTT 5 コンポーネント、URI 構文： **paho-mqtt5:topic**

使用方法と設定の詳細は、上記リンクを参照してください。

3.80.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-paho-mqtt5</artifactId>
</dependency>
```

3.81. PAHO

Eclipse Paho MQTT クライアントを使用して MQTT メッセージブローカーと通信します。

3.81.1. 含まれるもの

- Paho コンポーネント、URI 構文： **paho:topic**

使用方法と設定の詳細は、上記リンクを参照してください。

3.81.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.82. PLATFORM HTTP

このエクステンションにより、HTTP リクエストを使用するために HTTP エンドポイントを作成できます。

これは、**quarkus-vertx-http** エクステンションによって提供される Eclipse Vert.x HTTP サーバー上にビルドされます。

3.82.1. 含まれるもの

- Platform HTTP コンポーネント、URI 構文： **platform-http:path**

使用方法と設定の詳細は、上記リンクを参照してください。

3.82.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-platform-http</artifactId>
</dependency>
```

3.82.3. 使用方法

3.82.3.1. 基本的な使用方法

/hello エンドポイントですべての HTTP メソッドを提供します。

```
from("platform-http:/hello").setBody(simple("Hello ${header.name}"));
```

/hello エンドポイントで GET リクエストのみを提供します。

```
from("platform-http:/hello?httpMethodRestrict=GET").setBody(simple("Hello ${header.name}"));
```

3.82.3.2. Camel REST DSL 経由の platform-http の使用

platform-http コンポーネントで Camel REST DSL を使用できるようにするには、 **pom.xml** に **camel-quarkus-rest** を追加します。

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

その後、Camel REST DSL を使用できます。

```
rest()
    .get("/my-get-endpoint")
        .to("direct:handleGetRequest");

    .post("/my-post-endpoint")
        .to("direct:handlePostRequest");
```

3.82.3.3. multipart/form-data ファイルのアップロードの処理

ホワイトリストに登録して、アップロードを特定のファイル拡張子に制限することができます。

```
from("platform-http:/upload/multipart?fileNameExtWhitelist=html,txt&httpMethodRestrict=POST")
    .to("log:multipart")
    .process(e -> {
        final AttachmentMessage am = e.getMessage(AttachmentMessage.class);
        if (am.hasAttachments()) {
            am.getAttachments().forEach((fileName, dataHandler) -> {
                try (InputStream in = dataHandler.getInputStream()) {
                    // do something with the input stream
                } catch (IOException ioe) {
                    throw new RuntimeException(ioe);
                }
            });
        }
    });
```

3.82.3.4. platform-http エンドポイントのセキュリティー保護

Quarkus は、**platform-http** エンドポイントのセキュリティー保護に使用できるさまざまなセキュリティーおよび認証メカニズムを提供します。詳細は、Quarkus Security のドキュメント を参照してください。

ルート内で、認証されたユーザーとその関連する **SecurityIdentity** および **Principal** を取得できます。

```
from("platform-http:/secure")
    .process(e -> {
        Message message = e.getMessage();
        QuarkusHttpUser user =
message.getHeader(VertxPlatformHttpConstants.AUTHENTICATED_USER,
QuarkusHttpUser.class);
        SecurityIdentity securityIdentity = user.getSecurityIdentity();
        Principal principal = securityIdentity.getPrincipal();
        // Do something useful with SecurityIdentity / Principal. E.g check user roles etc.
    });
```

Quarkus ドキュメント で **quarkus.http.body.*** 設定オプション (特に次の項目) も確認してください。**quarkus.http.body.handle-file-uploads**、**quarkus.http.body.uploads-directory** および **quarkus.http.body.delete-uploaded-files-on-end**。

3.82.3.5. リバースプロキシーの実装

プラットフォーム HTTP コンポーネントはリバースプロキシーとして機能できます。その場合、**Exchange.HTTP_URI**、**Exchange.HTTP_HOST** ヘッダーは、HTTP 要求のリクエスト行で受信した絶対 URL から入力されます。

エクスチェンジを元のサーバーに単純にリダイレクトする HTTP プロキシーの例を次に示します。

```
from("platform-http:proxy")
    .toD("http://"
        + "${headers." + Exchange.HTTP_HOST + "}");
```

3.82.4. 追加の Camel Quarkus 設定

3.82.4.1. プラットフォーム HTTP サーバー設定

プラットフォーム HTTP サーバーの設定は Quarkus によって管理されます。設定オプションの完全なリストについては、Quarkus HTTP 設定ガイド を参照してください。

Platform HTTP サーバーの SSL を設定するには、SSL ガイドを使用した安全な接続 に従ってください。**SSLContextParameters** を使用した SSL 用のサーバーの設定は現在サポートされていないことに注意してください。

3.82.4.2. 文字エンコーディング

アプリケーションがデフォルト以外のエンコーディングを使用してリクエストを送受信することが想定される場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.83. QUARTZ

Quartz 2.x スケジューラーを使用してメッセージの送信をスケジュールします。

3.83.1. 含まれるもの

- Quartz コンポーネント、URI 構文： **quartz:groupName/triggerName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.83.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.83.3. 使用方法

3.83.3.1. クラスタリング

Quartz クラスタリングのサポートは、Quarkus Quartz エクステンション によって提供されます。次の手順では、Camel で使用する Quarkus Quartz の設定方法を説明します。

1. Quartz クラスター化モードを有効にし、**DataSource** を永続 Quartz ジョブストアとして設定します。設定例を以下に示します。

```
# Quartz configuration
quarkus.quartz.clustered=true
quarkus.quartz.store-type=jdbc-cmt
quarkus.scheduler.start-mode=forced

# Datasource configuration
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=quarkus_test
quarkus.datasource.password=quarkus_test
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost/quarkus_test

# Optional automatic creation of Quartz tables
quarkus.flyway.connect-retries=10
quarkus.flyway.table=flyway_quarkus_history
quarkus.flyway.migrate-at-start=true
quarkus.flyway.baseline-on-migrate=true
quarkus.flyway.baseline-version=1.0
quarkus.flyway.baseline-description=Quartz
```

2. **quarkus.datasource.db-kind** の値に対応するアプリケーションに、正しい JDBC ドライバーエクステンションを追加します。上記の例では **postgresql** が使用されるため、以下の JDBC 依存関係が必要になります。必要に応じて調整します。Agroal は、**DataSource** サポートにも必要になります。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-jdbc-postgresql</artifactId>
```

```
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-agroal</artifactId>
</dependency>
```

3. Quarkus Flyway は、必要な Quartz データベーステーブルを自動的に作成できます。 **quarkus-flyway** をアプリケーションに追加します (オプション)。

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-flyway</artifactId>
</dependency>
```

   また、選択したデータベースの種類に Quartz データベース作成スクリプトを追加します。Quartz プロジェクトは、こちら からコピーできる既製のスクリプトを提供します。SQL スクリプトを **src/main/resources/db/migration/V1.0.0__QuarkusQuartz.sql** に追加します。Quarkus Flyway は起動時にそれを検出し、Quartz データベーステーブルの作成に進みます。

4. Quarkus Quartz スケジューラーを使用するように Camel Quartz コンポーネントを設定します。

```
@Produces
@Singleton
@Named("quartz")
public QuartzComponent quartzComponent(Scheduler scheduler) {
    QuartzComponent component = new QuartzComponent();
    component.setScheduler(scheduler);
    return component;
}
```

Quartz スケジューラーの追加のカスタマイズは、さまざまな設定プロパティーで実行できます。詳細は、Quarkus Quartz Configuration ガイドを参照してください。

3.84. REF

Camel Registry で名前によって動的に検索されたエンドポイントにメッセージをルーティングします。

3.84.1. 含まれるもの

- Ref コンポーネント、URI 構文： **ref:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.84.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.84.3. 使用方法

CDI プロデューサーメソッドを利用してエンドポイントを Camel レジストリーにバインドできるため、Camel ルートで **ref** URI スキームを使用してエンドポイントを解決できます。

たとえば、エンドポイント Bean を生成するには、次のようにします。

```
@ApplicationScoped
public class MyEndpointProducers {
    @Inject
    CamelContext context;

    @Singleton
    @Produces
    @Named("endpoint1")
    public Endpoint directStart() {
        return context.getEndpoint("direct:start");
    }

    @Singleton
    @Produces
    @Named("endpoint2")
    public Endpoint logEnd() {
        return context.getEndpoint("log:end");
    }
}
```

ref: を使用して、Camel レジストリーにバインドされた CDI Bean の名前を参照します。

```
public class MyRefRoutes extends RouteBuilder {
    @Override
    public void configure() {
        // direct:start -> log:end
        from("ref:endpoint1")
            .to("ref:endpoint2");
    }
}
```

3.85. REST OPENAPI

RestProducerFactory インターフェイスを実装するコンポーネントに委任する OpenAPI 仕様ドキュメントに基づいて REST プロデューサーを設定します。

3.85.1. 含まれるもの

- REST OpenApi コンポーネント、URI 構文： **rest-openapi:specificationUri#operationId**

使用方法と設定の詳細は、上記リンクを参照してください。

3.85.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-rest-openapi</artifactId>
</dependency>
```

3.85.3. 使用方法

3.85.3.1. 必要な依存関係

rest-openapi エクステンションを使用する場合は、**RestProducerFactory** 実装が使用可能である必要があります。現在知られている拡張子は次のとおりです。

- camel-quarkus-http
- camel-quarkus-netty-http

Maven ユーザーは、これらの依存関係のいずれかを **pom.xml** に追加する必要があります。次に例を示します。

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

OpenApi 仕様の読み込みに使用されるメカニズムによっては、追加の依存関係が必要になる場合があります。**file** リソースロケーターを使用する場合、**org.apache.camel.quarkus:camel-quarkus-file** 拡張子をプロジェクトの依存関係として追加する必要があります。**ref** または **Bean** を使用して仕様をロードする場合、**org.apache.camel.quarkus:camel-quarkus-bean** 依存関係を追加するだけでなく、Bean 自体に **@RegisterForReflection** のアノテーションを付ける必要があります。

ネイティブコードで **classpath** リソースロケーターを使用する場合、OpenAPI 仕様へのパスを **application.properties** ファイルの **quarkus.native.resources.includes** プロパティーで指定する必要があります。以下に例を示します。

```
quarkus.native.resources.includes=openapi.json
```

3.86. REST

REST サービスおよび OpenAPI Specification を公開するか、外部の REST サービスを呼び出します。

3.86.1. 含まれるもの

- REST コンポーネント、URI 構文： **rest:method:path:uriTemplate**
- REST API コンポーネント、URI 構文： **rest-api:path**

使用方法と設定の詳細については、上記リンクを参照してください。

3.86.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.86.3. 追加の Camel Quarkus 設定

このエクステンションは、Platform HTTP エクステンションに依存し、REST トランスポートを提供するコンポーネントとして設定します。

3.86.3.1. platform-http 付きの特殊文字を含むパスパラメーター

platform-http REST トランスポートを使用する場合、一部の文字はパスパラメーター名内で許可されません。これには、- および $ 文字が含まれます。

以下の例の REST **/dashed/param** ルートを正しく機能させるには、システムプロパティーを **io.vertx.web.route.param.extended-pattern=true** にする必要があります。

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        rest("/api")
            // Dash '-' is not allowed by default
            .get("/dashed/param/{my-param}")
            .to("direct:greet")

            // The non-dashed path parameter works by default
            .get("/undashed/param/{myParam}")
            .to("direct:greet");

            from("direct:greet")
                .setBody(constant("Hello World"));
    }
}
```

Vert.x Web ドキュメント には、これに関するいくつかの背景があります。

3.86.3.2. 代替 REST トランスポートプロバイダーの設定

netty-http や **servlet** 等の別の REST トランスポートプロバイダーを使用するには、それぞれのエクステンションを依存関係としてプロジェクトへの追加し、**RouteBuilder** でプロバイダーを設定する必要があります。たとえば、**servlet** の場合、**org.apache.camel.quarkus:camel-quarkus-servlet** 依存関係を追加し、プロバイダーを次のように設定する必要があります。

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        restConfiguration()
```

```
                .component("servlet");
        ...
    }
}
```

3.87. SALESFORCE

Java DTO を使用して Salesforce と通信します。

3.87.1. 含まれるもの

- Salesforce コンポーネント、URI 構文： **salesforce:operationName:topicName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.87.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.87.3. 使用方法

3.87.3.1. salesforce-maven-plugin を使用した Salesforce DTO の生成

テスト内容。

プロジェクトの Salesforce DTO を生成するには、**salesforce-maven-plugin** を使用します。以下のサンプルコードスニペットは、**Account** オブジェクトの単一の DTO を作成します。

```
<plugin>
    <groupId>org.apache.camel.maven</groupId>
    <artifactId>camel-salesforce-maven-plugin</artifactId>
    <version>{camel-version}</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <clientId>${env.SALESFORCE_CLIENTID}</clientId>
                <clientSecret>${env.SALESFORCE_CLIENTSECRET}</clientSecret>
                <userName>${env.SALESFORCE_USERNAME}</userName>
                <password>${env.SALESFORCE_PASSWORD}</password>
                <loginUrl>https://login.salesforce.com</loginUrl>

<packageName>org.apache.camel.quarkus.component.salesforce.generated</packageName>
                <outputDirectory>src/main/java</outputDirectory>
```

```
            <includes>
              <include>Account</include>
            </includes>
          </configuration>
        </execution>
      </executions>
    </plugin>
```

3.87.3.2. POJO pubSubDeserializeType を使用した Pub / Sub API のネイティブモードサポート

Camel Salesforce Pub / Sub API を使用し、**pubSubDeserializeType** が **POJO** として設定されている場合は、リフレクション用に **pubSubPojoClass** オプションで設定されたクラスを登録する必要があります。

たとえば、次のルートがあるとします。

```
from("salesforce:pubSubSubscribe:/event/TestEvent__e?
pubSubDeserializeType=POJO&pubSubPojoClass=org.foo.TestEvent")
    .log("Received Salesforce POJO topic message: ${body}");
```

リフレクションのためにクラス **org.foo.TestEvent** を登録する必要があります。

```
package org.foo;

import io.quarkus.runtime.annotations.RegisterForReflection;

@RegisterForReflection
public class TestEvent {
    // Getters / setters etc
}
```

詳細は、ネイティブモード のユーザーガイドを参照してください。

3.87.4. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.88. SAGA

Saga EIP を使用して、ルート内でカスタムアクションを実行します。

3.88.1. 含まれるもの

- Saga コンポーネント、URI 構文： **saga:action**

使用方法と設定の詳細は、上記リンクを参照してください。

3.88.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.89. SAP

SAP Camel コンポーネントを提供します。

3.89.1. 含まれるもの

SAP エクステンションは、10 個の異なる SAP コンポーネントで構成されるパッケージです。sRFC、tRFC、および qRFC プロトコルをサポートするリモートファンクションコール (RFC) コンポーネントがあります。また、IDoc 形式のメッセージを使用して通信を容易にする IDoc コンポーネントがあります。このコンポーネントは、SAP Java Connector (SAP JCo) ライブラリーを使用して SAP との双方向通信を促進し、SAP IDoc ライブラリーを使用してドキュメントを中間ドキュメント (IDoc) 形式で送信します。

詳細は以下を参照してください。

3.89.2. Maven コーディネート

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

3.89.2.1. SAP エクステンションの Camel Quarkus の制限事項

SAP エクステンションはパッケージタイプ **uber-jar** をサポートしていないため、アプリケーションが次のような実行時例外を出力します。

```
Caused by: java.lang.ExceptionInInitializerError: JCo initialization failed with
java.lang.ExceptionInInitializerError: Illegal JCo archive "sap-1.0.0-SNAPSHOT-runner.jar". It is not
allowed to rename or repackage the original archive "sapjco3.jar".
```

3.89.2.2. SAP コンポーネントの追加のプラットフォーム制限

SAP コンポーネントはサードパーティーの JCo 3 および IDoc 3 ライブラリーに依存しているため、これらのライブラリーがサポートするプラットフォームにのみインストールできます。

3.89.2.3. SAP JCo および SAP IDoc ライブラリー

SAP コンポーネントを使用するための前提条件は、SAP Java Connector (SAP JCo) ライブラリーと SAP IDoc ライブラリーが Java ランタイムの **lib**/ ディレクトリーにインストールされていることです。ターゲットオペレーティングシステムに適した SAP ライブラリーのセットを SAP Service Marketplace からダウンロードしていることを確認する必要があります。

ライブラリーファイルの名前は、次に示すように、対象のオペレーティングシステムによって異なります。

表3.1 必要な SAP ライブラリー

SAP コンポーネント	Linux と UNIX	Windows
SAP JCo 3	**sapjco3.jar** **libsapjco3.so**	**sapjco3.jar** **sapjco3.dll**
SAP IDoc	**sapidoc3.jar**	**sapidoc3.jar**

注記

正しい SAP ライブラリーを取得するには、**pom.xml** に次の依存関係を含める必要があります。

```
<dependency>
      <groupId>com.sap.conn.jco</groupId>
      <artifactId>sapjco3</artifactId>
      <version>3.1.4</version>
      <scope>system</scope>
      <systemPath>${basedir}/lib/sapjco3.jar</systemPath>
    </dependency>
    <dependency>
      <groupId>com.sap.conn.idoc</groupId>
      <artifactId>sapidoc3</artifactId>
      <version>3.1.1</version>
      <scope>system</scope>
      <systemPath>${basedir}/lib/sapidoc3.jar</systemPath>
    </dependency>
```

3.89.3. URI 形式

SAP コンポーネントによって提供されるエンドポイントには、リモートファンクションコール (RFC) エンドポイントと中間ドキュメント (IDoc) エンドポイントの 2 種類があります。

RFC エンドポイントの URI 形式は次のとおりです。

```
sap-srfc-destination:destinationName:rfcName
sap-trfc-destination:destinationName:rfcName
sap-qrfc-destination:destinationName:queueName:rfcName
sap-srfc-server:serverName:rfcName[?options]
sap-trfc-server:serverName:rfcName[?options]
```

IDoc エンドポイントの URI 形式は次のとおりです。

```
sap-idoc-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-idoclist-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-qidoc-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applic
```

> **ationRelease**]]]
> sap-qidoclist-destination:**destinationName**:**queueName**:**idocType**[:**idocTypeExtension**[:**systemRelease**[:**applicationRelease**]]]
> sap-idoclist-server:**serverName**:**idocType**[:**idocTypeExtension**[:**systemRelease**[:**applicationRelease**]]]
> [?**options**]

sap- **endpointKind** -destination で始まる URI 形式は、宛先エンドポイント (つまり、Camel producer エンドポイント) を定義するために使用され、**destinationName** は SAP インスタンスへの特定のアウトバウンド接続の名前です。アウトバウンド接続は、コンポーネントレベルで名前が付けられ、設定されます。

sap- **endpointKind** -server で始まる URI 形式は、サーバーエンドポイント (つまり、Camel consumer エンドポイント) を定義するために使用され、**serverName** は SAP インスタンスからの特定のインバウンド接続の名前です。インバウンド接続は、コンポーネントレベルで名前が付けられ、設定されます。

RFC エンドポイント URI のその他のコンポーネントは次のとおりです。

rfcName

(必須) 宛先エンドポイント URI では、接続された SAP インスタンスのエンドポイントによって呼び出される RFC の名前です。サーバーエンドポイント URI では、接続された SAP インスタンスから呼び出されたときにエンドポイントによって処理される RFC の名前です。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

IDoc エンドポイント URI のその他のコンポーネントは次のとおりです。

idocType

(必須) このエンドポイントによって生成される IDoc の基本 IDoc タイプを指定します。

idocTypeExtension

このエンドポイントによって生成された IDoc の IDoc タイプ拡張 (存在する場合) を指定します。

systemRelease

このエンドポイントによって生成された IDoc に関連付けられている SAP Basis Release がある場合は、それを指定します。

applicationRelease

このエンドポイントによって生成された IDoc の関連付けられたアプリケーションリリースがある場合は、それを指定します。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

3.89.3.1. RFC 宛先エンドポイントのオプション

RFC 宛先エンドポイント (**sap-srfc-destination**、**sap-trfc-destination**、および **sap-qrfc-destination**) は、次の URI オプションをサポートしています。

名前	デフォルト	説明

名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します
transacted	**false**	**true** の場合、このエンドポイントが SAP トランザクションを開始することを指定します

3.89.3.2. RFC サーバーエンドポイントのオプション

SAP RFC サーバーエンドポイント (**sap-srfc-server** および **sap-trfc-server**) は、次の URI オプションをサポートしています。

名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します。
propagateExceptions	**false**	(sap-trfc-server **エンドポイントのみ**) **true** の場合、エクスチェンジの例外ハンドラーではなく、このエンドポイントが SAP の呼び出し元に例外を伝達することを指定します。

3.89.3.3. IDoc List Server エンドポイントのオプション

SAP IDoc List Server エンドポイント (**sap-idoclist-server**) は、次の URI オプションをサポートしています。

名前	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始することを指定します。
propagateExceptions	**false**	**true** の場合、エクスチェンジの例外ハンドラーではなく、このエンドポイントが SAP の呼び出し元に例外を伝搬することを指定します。

3.89.3.4. RFC および IDoc エンドポイントの概要

SAP コンポーネントパッケージは、次の RFC および IDoc エンドポイントを提供します。

sap-srfc-destination

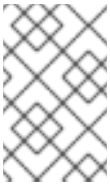
Camel SAP Synchronous Remote Function Call Destination Camel コンポーネント。このエンドポイントは、Camel ルートが SAP システムへのリクエストと SAP システムからのレスポンスの同期配信を必要とする場合に使用する必要があります。

注記

このコンポーネントで使用される sRFC プロトコルは、SAP システムとの間でリクエストとレスポンスを **ベストエフォート** で配信します。リクエストの送信中に通信エラーが発生した場合、受信側の SAP システムでのリモート関数呼び出しの完了ステータスは **不明** のままです。

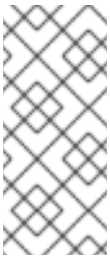
sap-trfc-destination

Camel SAP Transactional Remote Function Call Destination Camel コンポーネント。このエンドポイントは、リクエストを受信側の SAP システムに **最大 1 回** 配信する必要がある場合に使用する必要があります。これを達成するために、コンポーネントはトランザクション ID **tid** を生成します。この ID は、ルートのエクスチェンジでコンポーネントを介して送信されるすべてのリクエストに付随します。受信側の SAP システムは、リクエストを配信する前に、リクエストに付随する **tid** を記録します。SAP システムが同じ **tid** のリクエストを再度受信した場合、リクエストは配信されません。したがって、このコンポーネントのエンドポイントを介してリクエストを送信するときにルートで通信エラーが発生した場合、ルートは同じリクエスト内でリクエストの送信を再試行できますが、配信と実行は 1 回だけです。



注記

このコンポーネントで使用される tRFC プロトコルは非同期であり、レスポンスを返しません。したがって、このコンポーネントのエンドポイントはレスポンスメッセージを返しません。

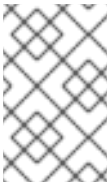


注記

このコンポーネントは、エンドポイントを介した一連のリクエストの順序を保証しません。これらのリクエストの配信および実行順序は、通信エラーやリクエストの再送信により、受信側の SAP システムで異なる場合があります。配送順序の保証については、Camel SAP Queued Remote Function Call Destination Camel コンポーネントを参照してください。

sap-qrfc-destination

Camel SAP Queued Remote Function Call Destination Camel コンポーネント。このコンポーネントは、Transactional Remote Function Call Destination camel コンポーネントの機能を、そのエンドポイントを介したリクエストの配信に **順番に** 配信保証を追加することで拡張します。このエンドポイントは、一連のリクエストが相互に依存しており、受信側の SAP システムに **最大 1 回、順番に** 配信する必要がある場合に使用する必要があります。このコンポーネントは、Camel SAP Transactional Remote Function Call Destination Camel コンポーネントと同じメカニズムを使用して、**最大 1 回の** 配信保証を実現します。順序の保証は、SAP システムが受信した順序でリクエストを **受信キュー** にシリアル化することによって実現されます。受信キューは、SAP 内の **QIN スケジューラー** によって処理されます。インバウンドキューが **アクティブ化される** と、QIN スケジューラーはキューリクエストを順番に実行します。



注記

このコンポーネントで使用される qRFC プロトコルは非同期であり、レスポンスを返しません。したがって、このコンポーネントのエンドポイントはレスポンスメッセージを返しません。

sap-srfc-server

Camel SAP Synchronous Remote Function Call Server Camel コンポーネント。このコンポーネントとそのエンドポイントは、SAP システムからの要求と SAP システムへの応答を同期的に処理するために Camel ルートが必要な場合に使用する必要があります。

sap-trfc-server

Camel SAP Transactional Remote Function Call Server Camel コンポーネント。このエンドポイントは、送信側の SAP システムがリクエストを Camel ルートに **最大 1 回** 配信する必要がある場合に使用する必要があります。これを実現するために、送信側の SAP システムは、コンポーネントのエンドポイントに送信するすべてのリクエストに付随するトランザクション ID **tid** を生成します。送信側の SAP システムは、**tid** に関連付けられた一連のリクエストを送信する前に、コンポーネントが特定の **tid** を受信したかどうかを最初にチェックします。コンポーネントは、保持している受信 **tid** のリストをチェックし、送信された **tid** がそのリストにない場合は記録し、送信 SAP システムに応答して、**tid** がすでに記録されているかどうかを示します。送信側の SAP システムは、**tid** が以前に記録されていない場合にのみ、一連のリクエストを送信します。これにより、送信側の SAP システムは一連のリクエストを確実に camel ルートに 1 回送信できます。

sap-idoc-destination

Camel SAP IDoc Destination Camel コンポーネント。このエンドポイントは、Camel ルートが Intermediate Documents (IDoc) のリストを SAP システムに送信する場合に使用する必要があります。

sap-idoclist-destination

Camel SAP IDoc List Destination Camel コンポーネント。このエンドポイントは、Camel ルートが Intermediate Documents (IDoc) リストのリストを SAP システムに送信する場合に使用する必要があります。

sap-qidoc-destination

Camel SAP Queued IDoc Destination Camel コンポーネント。このコンポーネントとそのエンドポイントは、中間ドキュメント (IDoc) のリストを順番に SAP システムに送信するために Camel ルートが必要な場合に使用する必要があります。

sap-qidoclist-destination

Camel SAP Queued IDoc List Destination Camel コンポーネント。このコンポーネントとそのエンドポイントは、キャメルルートが Intermediate Documents (IDoc) リストを順番に SAP システムに送信する場合に使用されます。

sap-idoclist-server

Camel SAP IDoc List Server Camel コンポーネント。このエンドポイントは、送信側の SAP システムが中間ドキュメントリストを Camel ルートに配信する必要がある場合に使用する必要があります。このコンポーネントは、**sap-trfc-server-standalone** クイックスタートで説明されているように、tRFC プロトコルを使用して SAP と通信します。

3.89.3.5. SAP RFC 宛先エンドポイント

RFC 宛先エンドポイントは、SAP へのアウトバウンド通信をサポートします。これにより、これらのエンドポイントは、SAP の ABAP 関数モジュールへの RFC 呼び出しを行うことができます。RFC 宛先エンドポイントは、SAP インスタンスへの特定の接続を介して特定の ABAP 関数への RFC 呼び出しを行うように設定されています。RFC 宛先は、アウトバウンド接続の論理的な指定であり、一意の名前を持っています。RFC 宛先は、**宛先データ** と呼ばれる一連の接続パラメーターによって指定されます。

RFC 宛先エンドポイントは、受信した IN-OUT エクスチェンジの入力メッセージから RFC リクエストを抽出し、そのリクエストを関数呼び出しで SAP にディスパッチします。関数呼び出しからのレスポンスは、エクスチェンジの出力メッセージで返されます。SAP RFC 宛先エンドポイントはアウトバウンド通信のみをサポートするため、RFC 宛先エンドポイントは producer の作成のみをサポートします。

3.89.3.6. SAP RFC サーバーエンドポイント

RFC サーバーエンドポイントは、SAP からのインバウンド通信をサポートします。これにより、SAP の ABAP アプリケーションがサーバーエンドポイントに対して RFC 呼び出しを行うことができます。ABAP アプリケーションは、リモート関数モジュールであるかのように RFC サーバーエンドポイントと対話します。RFC サーバーエンドポイントは、SAP インスタンスから特定の接続を介して特定の RFC 関数への RFC 呼び出しを受信するように設定されています。RFC サーバーは、インバウンド接続の論理的な指定であり、一意の名前を持っています。RFC サーバーは、**サーバーデータ** と呼ばれる一連の接続パラメーターによって指定されます。

RFC サーバーエンドポイントは、入力 RFC リクエストを処理し、それを IN-OUT エクスチェンジの入力メッセージとしてディスパッチします。エクスチェンジの出力メッセージは、RFC 呼び出しのレスポンスとして返されます。SAP RFC サーバーエンドポイントはインバウンド通信のみをサポートするため、RFC サーバーエンドポイントは consumer の作成のみをサポートします。

3.89.3.7. SAP IDoc および IDoc リストの宛先エンドポイント

IDoc 宛先エンドポイントは、SAP へのアウトバウンド通信をサポートします。SAP は、IDoc メッセージに対してさらに処理を実行できます。IDoc ドキュメントはビジネストランザクションを表し、非 SAP システムと簡単にエクスチェンジできます。IDoc 宛先は、**宛先データ** と呼ばれる一連の接続パラメーターによって指定されます。

IDoc リスト宛先エンドポイントは、処理するメッセージが IDoc ドキュメントの **リスト** で構成されていることを除けば、IDoc 宛先エンドポイントと似ています。

3.89.3.8. SAP IDoc リストサーバーエンドポイント

IDoc リストサーバーエンドポイントは、SAP からのインバウンド通信をサポートし、Camel ルートが SAP システムから IDoc ドキュメントのリストを受信できるようにします。IDoc リストサーバーは、**サーバーデータ** と呼ばれる一連の接続パラメーターによって指定されます。

3.89.3.9. メタデータリポジトリー

メタデータリポジトリーは、次の種類のメタデータを格納するために使用されます。

汎用モジュールのインタフェース説明

このメタデータは、JCo および ABAP ランタイムによって使用され、RFC 呼び出しをチェックして、それらの呼び出しをディスパッチする前に、通信パートナー間でタイプセーフなデータ転送を保証します。リポジトリーには、リポジトリーデータが取り込まれます。リポジトリーデータは、名前付き関数テンプレートのマップです。関数テンプレートには、関数モジュールとの間で渡されるすべてのパラメーターとその入力情報を記述するメタデータが含まれており、関数テンプレートが説明する関数モジュールの一意の名前が付けられています。

IDoc タイプの説明

このメタデータは、IDoc ランタイムによって使用され、IDoc ドキュメントが通信パートナーに送信される前に正しくフォーマットされていることを確認します。基本的な IDoc タイプは、名前、許可されたセグメントのリスト、およびセグメント間の階層関係の説明で構成されます。いくつかの追

加の制約をセグメントに課すことができます。セグメントは必須またはオプションにすることができます。また、各セグメントの最小/最大範囲を指定することができます (そのセグメントの許容反復回数を定義します)。

したがって、SAP 宛先およびサーバーエンドポイントは、RFC 呼び出しを送受信し、IDoc ドキュメントを送受信するために、リポジトリーへのアクセスを必要とします。RFC 呼び出しの場合、エンドポイントによって呼び出されて処理されるすべての機能モジュールのメタデータは、リポジトリー内に存在する必要があります。IDoc エンドポイントの場合、エンドポイントによって処理されるすべての IDoc タイプおよび IDoc タイプ拡張のメタデータは、リポジトリー内に存在する必要があります。宛先およびサーバーエンドポイントによって使用されるリポジトリーの場所は、それぞれの接続の宛先データおよびサーバーデータで指定されます。

SAP 宛先エンドポイントの場合、使用するリポジトリーは通常、SAP システムに存在し、接続先の SAP システムにデフォルト設定されます。このデフォルトでは、宛先データに明示的な設定は必要ありません。さらに、宛先エンドポイントが行うリモート関数呼び出しのメタデータは、それが呼び出す既存の関数モジュールのリポジトリーにすでに存在します。したがって、宛先エンドポイントによって行われる呼び出しのメタデータは、SAP コンポーネントで設定する必要はありません。

一方、サーバーエンドポイントによって処理される関数呼び出しのメタデータは、通常、SAP システムのリポジトリーには存在せず、代わりに SAP コンポーネントに存在するリポジトリーによって提供される必要があります。SAP コンポーネントは、名前付きメタデータリポジトリーのマップを維持します。リポジトリーの名前は、メタデータを提供するサーバーの名前に対応しています。

3.89.4. 設定

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを格納する 3 つのマップを維持します。**宛先データストア** と **サーバーデータストア** は、SAP コンポーネントに自動的に注入される特別な設定オブジェクト **SapConnectionConfiguration** で設定されます。**リポジトリーデータストア** は、関連する SAP コンポーネントで直接設定する必要があります。

3.89.4.1. 設定の概要

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを格納する 3 つのマップを維持します。コンポーネントのプロパティー **destinationDataStore** は宛先名をキーとする宛先データを格納し、プロパティー **serverDataStore** はサーバー名をキーとするサーバーデータを格納し、プロパティー **repositoryDataStore** はリポジトリー名をキーとするリポジトリーデータを格納します。これらの設定は、初期化中にコンポーネントに渡す必要があります。

例

次の例は、サンプルの宛先データストアとサンプルのサーバーデータストアを設定する方法を示しています。**sap-configuration** Bean (**SapConnectionConfiguration** 型) は、このアプリケーションで使用される SAP コンポーネントに自動的に注入されます。

```
public class SAPRouteBuilder extends RouteBuilder {
 @BindToRegistry("sap-configuration")
 public SapConnectionConfiguration sapConfiguration() {
  SapConnectionConfiguration configuration = new SapConnectionConfiguration();
  configuration.setDestinationDataStore(destinationData());
  configuration.setServerDataStore(serverData());
  return configuration;
 }

 /**
  * Configures an Inbound SAP Connection
  * Please enter the connection property values for your environment
```

```
    */
    private Map<String, ServerData> serverData() {
     ServerData data = new ServerDataImpl();
     data.setGwhost("example.com");
     data.setGwserv("3300");
     data.setProgid("QUICKSTART");
     data.setRepositoryDestination("quickstartDest");
     data.setConnectionCount("2");
     return Map.of("quickstartServer", data);
    }

    /**
     * Configures an Outbound SAP Connection
     * Please enter the connection property values for your environment
     */
    private Map<String, DestinationData> destinationData() {
     DestinationData data = new DestinationDataImpl();
     data.setAshost("example.com");
     data.setSysnr("00");
     data.setClient("000");
     data.setUser("username");
     data.setPasswd("password");
     data.setLang("en");
     return Map.of("quickstartDest", data);
    }

    @Override
    public void configure() throws Exception {
     // Routes definitions
    }
}
```

注記

値は、**application.properties** ファイルから指定できます。その場合、ハードコードされた値の代わりにプロパティー名を使用できます。

以下に例を示します。

ConfigProvider.getConfig().getValue("<property name>", String.class)

3.89.4.2. 宛先設定

宛先の設定は、SAP コンポーネントの **destinationDataStore** プロパティーで維持されます。このマップの各エントリーは、SAP インスタンスへの個別のアウトバウンド接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、URI 形式のセクションで説明されているように、宛先エンドポイント URI の **destinationName** コンポーネントで使用されます。

各エントリーの値は、アウトバウンド SAP 接続の設定を指定する宛先データ設定オブジェクト (**org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl**) です。

サンプル宛先設定

次のコードは、**quickstartDest** という名前のサンプルの宛先を設定する方法を示しています。

```
    @BindToRegistry("sap-configuration")
```

```
    public SapConnectionConfiguration sapConfiguration() {
        SapConnectionConfiguration configuration = new SapConnectionConfiguration();
        configuration.setDestinationDataStore(destinationData());
        return configuration;
    }

    private Map<String, DestinationData> destinationData() {
        DestinationData data = new DestinationDataImpl();
        data.setAshost("example.com");
        data.setSysnr("00");
        data.setClient("000");
        data.setUser("username");
        data.setPasswd("password");
        data.setLang("en");
        return Map.of("quickstartDest", data);
    }

    @Override
    public void configure() throws Exception {
        ((DefaultCamelContext) getCamelContext()).addInterceptStrategy(new
CurrentProcessorDefinitionInterceptStrategy());
        // Routes definitions
    }
```

上記のように宛先を設定した後、次の URI を使用して、**quickstartDest** 宛先で **BAPI_FLCUST_GETLIST** リモート関数呼び出しを呼び出すことができます。

```
sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST
```

3.89.4.2.1. tRFC および qRFC 宛先のインターセプター

前のサンプル宛先設定は、**CurrentProcessorDefinitionInterceptStrategy** オブジェクトのインスタンス化を示しています。このオブジェクトは、Camel ランタイムにインターセプターをインストールします。これにより、Camel SAP コンポーネントは、RFC トランザクションの処理中に Camel ルート内の位置を追跡できます。

重要

このインターセプターは、トランザクション RFC 宛先エンドポイント (**sap-trfc-destination** や **sap-qrfc-destination** など) にとって非常に重要であり、アウトバウンドトランザクション RFC 通信を適切に管理するには、Camel ランタイムにインストールする必要があります。ストラテジーが実行時に見つからない場合、Destination RFC Transaction Handlers は Camel ログに警告を発行します。この状況では、アウトバウンドのトランザクション RFC 通信を適切に管理するために、Camel ランタイムを再プロビジョニングして再起動する必要があります。

3.89.4.2.2. ログオンと認証オプション

次の表に、SAP 宛先データストアで宛先を設定するための **ログオンオプションと認証** オプションを示します。

名前	デフォルト値	説明

client		SAP クライアント、必須ログオンパラメーター
user		ログオンユーザー、パスワードベースの認証用のログオンパラメーター。
aliasUser		ログオンユーザーエイリアスは、ログオンユーザーの代わりに使用できます。
userId		ABAP AS へのログオンに使用されるユーザー ID。宛先設定が認証に SSO/アサーションチケット、証明書、現在のユーザー、または SNC 環境を使用する場合、JCo ランタイムによって使用されます。ユーザーもユーザー別名も設定されていない場合、ユーザー ID は必須です。この ID は SAP バックエンドに送信されることはなく、JCo ランタイムによってローカルで使用されます。
passwd		ログオンパスワード、パスワードベースの認証用のログオンパラメーター。
lang		ログオン言語。定義されていない場合は、デフォルトのユーザー言語が使用されます。
mysapsso2		指定された SAP Cookie バージョン 2 を、SSO ベースの認証のログオンチケットとして使用します。
x509cert		指定された X509 証明書を証明書ベースの認証に使用します。
lcheck		最初の呼び出しまで認証を延期する - 1 (有効)。特別な場合にのみ使用されます。
useSapGui		表示または非表示の SAP GUI を使用するか、SAP GUI を使用しません。

codePage		ログオンパラメーターの変換に使用されるコードページを定義する追加のログオンパラメーター。特別な場合にのみ使用されます。
getsso2		ログオン後に SSO チケットを注文すると、取得したチケットは宛先属性で使用できます。
denyInitialPassword		**1** に設定すると、初期パスワードを使用すると例外が発生します (デフォルトは **0** です)。

3.89.4.2.3. 接続オプション

次の表に、SAP 宛先データストアで宛先を設定するための **接続** オプションを示します。

名前	デフォルト値	説明
saprouter		SAP ルーターの背後にあるシステムに接続するための SAP ルーター文字列。SAP ルーター文字列には、一連の SAP ルーターとそのポート番号が含まれており、**(/H/<host>/S/<port>)+** という形式になっています。
sysnr		SAP ABAP アプリケーションサーバーのシステム番号、直接接続に必須。
ashost		SAP ABAP アプリケーションサーバー、直接接続に必須。
mshost		SAP メッセージサーバー、負荷分散接続の必須プロパティー。
msserv		SAP メッセージサーバーポート。負荷分散接続のオプションプロパティー。サービス名 sapmsXXX を解決するために、**etc/services** のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。

gwhost		アプリケーションサーバーへの接続を確立するために使用する具体的なゲートウェイを指定できます。指定しない場合、アプリケーションサーバーのゲートウェイが使用されます。
gwserv		gwhost を使用する場合に設定する必要があります。そのゲートウェイで使用されるポートを指定できます。指定しない場合、アプリケーションサーバーのゲートウェイのポートが使用されます。サービス名 sapgwXXX を解決するために、etc/services のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。
r3name		SAP システムのシステム ID。負荷分散接続の必須プロパティー。
group		SAP アプリケーションサーバーのグループ、負荷分散接続の必須プロパティー。

network	**LAN**	パフォーマンスを最適化するには、JCo とターゲットシステム間のネットワーク品質に応じてこの値を設定します。有効な値は **LAN** または **WAN** です (これは高速シリアル化にのみ関連します)。**network** 設定オプションを **WAN** に設定すると、速度は遅くなりますがより効率的な圧縮アルゴリズムが使用され、データはさらに圧縮オプションを得るために分析されます。**network** 設定を **LAN** に設定すると、非常に高速な圧縮アルゴリズムが使用され、データ分析は非常に基本的なレベルでのみ実行されます。**LAN** オプションを設定すると、圧縮率はそれほど効率的ではありませんが、ネットワーク転送時間はそれほど重要ではないと見なされます。デフォルト設定は **LAN** です。
serializationFormat	**rowBased**	有効な値は、**rowBased** または **columnBased** です。シリアル化を高速化するには、**columnBased** を設定する必要があります。デフォルトのシリアル化設定は **rowBased** です。

3.89.4.2.4. 接続プールのオプション

次の表に、SAP 宛先データストアで宛先を設定するための **接続プール** オプションを示します。

名前	デフォルト値	説明
peakLimit	**0**	宛先に対して同時に作成できるアクティブなアウトバウンド接続の最大数。値を **0** にすると、無制限の数のアクティブな接続が許可されます。それ以外の場合、値が **jpoolCapacity** の値よりも小さい場合は、この値まで自動的に増加されます。デフォルト設定は **poolCapacity** の値です。または、**poolCapacity** も指定されていない場合、デフォルトは **0** (無制限) です。

poolCapacity	**1**	宛先によって開いたままのアイドル状態のアウトバウンド接続の最大数。値 **0** は、接続プーリングがないという効果があります (デフォルトは **1** です)。
expirationTime		宛先によって内部的に保持されている空き接続を閉じることができるまでの時間 (ミリ秒)。
expirationPeriod		宛先が解放された接続の有効期限をチェックするまでのミリ秒単位の期間。
maxGetTime		アプリケーションによって最大許容数の接続がすでに割り当てられている場合に、接続を待機する最大時間 (ミリ秒)。

3.89.4.2.5. 安全なネットワーク接続オプション

次の表に、SAP 宛先データストアで宛先を設定するための **セキュアなネットワーク** オプションを示します。

名前	デフォルト値	説明
sncMode		セキュアなネットワーク接続 (SNC) モード、**0** (オフ) または **1** (オン)。
sncPartnername		SNC パートナー、例: **p:CN=R3, O=XYZ-INC, C=EN**
sncQop		セキュリティーの SNC レベル、**1** から **9**
sncMyname		独自の SNC 名。環境設定をオーバーライドします。
sncLibrary		SNC サービスを提供するライブラリーへのパス。

3.89.4.2.6. リポジトリーオプション

次の表に、SAP 宛先データストアで宛先を設定するための **リポジトリー** オプションを示します。

名前	デフォルト値	説明
repositoryDest		リポジトリーとして使用される宛先を指定します。
repositoryUser		リポジトリーの宛先が設定されておらず、このプロパティーが設定されている場合、リポジトリー呼び出しのユーザーとして使用されます。これにより、リポジトリーの検索に別のユーザーを使用できます。
repositoryPasswd		リポジトリーユーザーのパスワード。リポジトリーユーザーを使用する場合は必須です。
repositorySnc		**(オプション)** この宛先に SNC が使用されている場合、このプロパティーが **0** に設定されていれば、リポジトリー接続に対して SNC をオフにすることができます。デフォルト設定は **jco.client.snc**_mode の値です。特殊な場合のみ。

repositoryRoundtripOptimization		**RFC_METADATA_GET** API を有効にすると、リポジトリーデータが 1 回の往復で提供されます。 **1** ABAP システムで **RFC_METADATA_GET** の使用を有効にします。 **0** ABAP システムで **RFC_METADATA_GET** を無効化します。 プロパティーが設定されていない場合、宛先は最初にリモート呼び出しを実行して、**RFC_METADATA_GET** が使用可能かどうかを確認します。利用可能な場合、宛先はそれを使用します。 **注記:** リポジトリーがすでに初期化されている場合 (たとえば、他の宛先で使用されているため)、このプロパティーは効果がありません。通常、このプロパティーは ABAP システムに関連しており、同じ ABAP システムを指すすべての宛先で同じ値を持つ必要があります。バックエンドの前提条件については、ノート 1456826 を参照してください。

3.89.4.2.7. トレース設定オプション

次の表に、SAP 宛先データストアで宛先を設定するための **トレース設定** オプションを示します。

名前	デフォルト値	説明
trace		RFC トレースを有効/無効にする (**0** または **1**)
cpicTrace		CPIC トレースを有効/無効にする **[0..3]**。

3.89.4.3. サーバー設定

サーバーの設定は、SAP コンポーネントの **serverDataStore** プロパティーで維持されます。このマップの各エントリーは、SAP インスタンスからの個別のインバウンド接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、URI 形式のセクションで説明されているように、サーバーエンドポイント URI の **serverName** コンポーネントで使用されます。

各エントリーの値は、インバウンド SAP 接続の設定を定義する **サーバーデータ設定オブジェクト、org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl** です。

サンプルサーバー設定

次のコードは、**quickstartServer** という名前のサンプルのサーバー設定を作成する方法を示しています。

```
@BindToRegistry("sap-configuration")
 public SapConnectionConfiguration sapConfiguration() {
  SapConnectionConfiguration configuration = new SapConnectionConfiguration();
  configuration.setDestinationDataStore(destinationData());
  configuration.setServerDataStore(serverData());
  return configuration;
 }

 /**
  * Configures an Inbound SAP Connection
  * Please enter the connection property values for your environment
  */
 private Map<String, ServerData> serverData() {
  ServerData data = new ServerDataImpl();
  data.setGwhost("example.com");
  data.setGwserv("3300");
  data.setProgid("QUICKSTART");
  data.setRepositoryDestination("quickstartDest");
  data.setConnectionCount("2");
  return Map.of("quickstartServer", data);
 }

 /**
  * Configures an Outbound SAP Connection
  * Please enter the connection property values for your environment
  */
 private Map<String, DestinationData> destinationData() {
  DestinationData data = new DestinationDataImpl();
  data.setAshost("example.com");
  data.setSysnr("00");
  data.setClient("000");
  data.setUser("username");
  data.setPasswd("password");
  data.setLang("en");
  return Map.of("quickstartDest", data);
 }
```

注記

この例では、サーバーがリモート SAP インスタンスからメタデータを取得するために使用する宛先接続 **quickstartDest** も設定します。この宛先は、**repositoryDestination** オプションを介してサーバーデータで設定されます。このオプションを設定しない場合は、代わりにローカルメタデータリポジトリーを作成する必要があります。

上記のように宛先を設定した後、次の URI を使用して、呼び出し元クライアントからの **quickstartDest** リモート関数呼び出しで **BAPI_FLCUST_GETLIST** リモート関数呼び出しを処理できます。

```
sap-srfc-server:quickstartServer:BAPI_FLCUST_GETLIST
```

3.89.4.3.1. 必須オプション

サーバーデータ設定オブジェクトに必要なオプションは次のとおりです。

名前	デフォルト値	説明
gwhost		サーバー接続を登録するゲートウェイホスト。
gwserv		登録を行うことができるポートであるゲートウェイサービス。サービス名 **sapgwXXX** を解決するために、**etc/services** のルックアップがオペレーティングシステムのネットワーク層によって実行されます。シンボリックサービス名の代わりにポート番号を使用する場合、ルックアップは実行されず、追加のエントリーは必要ありません。
progid		登録が行われたプログラム ID。ゲートウェイおよび ABAP システムの宛先で ID として機能します。
repositoryDestination		リモート SAP サーバーでホストされているメタデータリポジトリーからメタデータを取得するためにサーバーが使用できる宛先名を指定します。
connectionCount		ゲートウェイに登録する必要がある接続の数。

3.89.4.3.2. 安全なネットワーク接続オプション

サーバーデータ設定オブジェクトの安全なネットワーク接続オプションは次のとおりです。

名前	デフォルト値	説明
sncMode		セキュアなネットワーク接続 (SNC) モード、**0** (オフ) または **1** (オン)。
sncQop		セキュリティーの SNC レベル、**1** から **9**。

sncMyname		サーバーの SNC 名。デフォルトの SNC 名を上書きします。通常、**p:CN=JCoServer, O=ACompany, C=EN** のようなものです。
sncLib		SNC サービスを提供するライブラリーへのパス。このプロパティーが指定されていない場合は、代わりに **jco.middleware.snc_lib** プロパティーの値が使用されます。

3.89.4.3.3. その他のオプション

サーバーデータ設定オブジェクトのその他のオプションは次のとおりです。

名前	デフォルト値	説明
saprouter		その ABAP システムのゲートウェイでサーバーを登録するときに、ファイアウォールによって保護されているため、SAProuter を介してのみアクセスできるシステムに使用する SAP ルーター文字列。一般的なルーター文字列は **/H/firewall.hostname/H/** です。
maxStartupDelay		失敗した場合の 2 回の起動試行間の最大時間 (秒単位)。待機時間は、起動に失敗するたびに最初の 1 秒から 2 倍になり、最大値に達するか、サーバーが正常に起動できるようになります。
trace		RFC トレースを有効/無効にする (**0** または **1**)
workerThreadCount		サーバー接続で使用されるスレッドの最大数。設定されていない場合、**connectionCount** の値が **workerThreadCount** として使用されます。スレッドの最大数は 99 を超えることはできません。

workerThreadMinCount		サーバー接続で使用されるスレッドの最小数。設定されていない場合、**connectionCount** の値が **workerThreadMinCount** として使用されます。

3.89.4.4. リポジトリーの設定

リポジトリーの設定は、SAP コンポーネントの **repositoryDataStore** プロパティーで維持されます。このマップの各エントリーは、個別のリポジトリーを設定します。各エントリーのキーはリポジトリーの名前であり、このキーはこのリポジトリーが接続されているサーバーの名前にも対応しています。

各エントリーの値は、メタデータリポジトリーのコンテンツを定義するリポジトリーデータ設定オブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl** です。リポジトリーデータオブジェクトは、機能テンプレート設定オブジェクト **org.fuesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl** のマップです。このマップの各エントリーは汎用モジュールのインタフェースを指定し、各エントリーのキーは指定された汎用モジュールの名前です。

リポジトリーデータの例

次のコードは、メタデータリポジトリーを設定する簡単な例を示しています。

```
@BindToRegistry("sap-configuration")
public SapConnectionConfiguration sapConfiguration() {
    SapConnectionConfiguration configuration = new SapConnectionConfiguration();
    configuration.setRepositoryDataStore(repositoryData());
    return configuration;
}

private Map<String, RepositoryData> repositoryData() {
    RepositoryData data = new RepositoryDataImpl();
    FunctionTemplate bookFlightFunctionTemplate = new FunctionTemplateImpl();
    data.setFunctionTemplates(Map.of("BOOK_FLIGHT", bookFlightFunctionTemplate));
    return Map.of("nplServer", data);
}
```

3.89.4.4.1. 関数テンプレートのプロパティー

汎用モジュールのインタフェースは、RFC コールでデータが汎用モジュールとの間でやり取りされる 4 つのパラメーターリストで構成されています。各パラメーターリストは 1 つ以上のフィールドで構成され、それぞれが RFC コールで転送される名前付きパラメーターです。次のパラメーターリストと例外リストがサポートされています。

- **インポートパラメーターリスト** には、RFC コールで汎用モジュールに送信されるパラメーター値が含まれています。
- **エクスポートパラメーターリスト** には、RFC コールで汎用モジュールによって返されるパラメーター値が含まれています。
- **変更パラメーター一覧** には、RFC コールで汎用モジュールとの間で送受信されるパラメーター値が含まれています。

- **テーブルパラメーター一覧** には、RFC コールで汎用モジュールとの間で送受信される内部テーブル値が含まれています。
- 汎用モジュールのインタフェースは、モジュールが RFC コールで呼び出されたときに発生する可能性のある ABAP 例外の **例外リスト** からも設定されます。

関数テンプレートは、関数インターフェイスの各パラメーターリスト内のパラメーターの名前と型、および関数によって出力される ABAP 例外を記述します。関数テンプレートオブジェクトは、次の表に示すように、メタデータオブジェクトの 5 つのプロパティーリストを保持します。

プロパティー	説明
importParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールへの RFC 呼び出しで送信されるパラメーターを指定します。
changingParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールとの間の RFC 呼び出しで送受信されるパラメーターを指定します。
exportParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールからの RFC 呼び出しで返されるパラメーターを指定します。
tableParameterList	リストフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**。汎用モジュールとの間の RFC 呼び出しで送受信されるテーブルパラメーターを指定します。
exceptionList	ABAP 例外メタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.AbapExceptionImpl**。汎用モジュールの RFC 呼び出しで発生する可能性がある ABAP 例外を指定します。

関数テンプレートの例

次の例は、関数テンプレートを設定する方法の概要を示しています。

```
FunctionTemplate bookFlightFunctionTemplate = new FunctionTemplateImpl();

List<ListFieldMetaData> metaDataList = new ArrayList<>();
ListFieldMetaData metaData = new ListFieldMetaDataImpl();

// configure values
metaData.setName("example");
```

```
metaDataList.add(metaData);
bookFlightFunctionTemplate.setImportParameterList(metaDataList);

// in the same way you can configure other parameters
bookFlightFunctionTemplate.setExportParameterList(...);
bookFlightFunctionTemplate.setChangingParameterList(...);
bookFlightFunctionTemplate.setExceptionList(...);
bookFlightFunctionTemplate.setTableParameterList(...);
```

3.89.4.4.2. リストフィールドのメタデータプロパティー

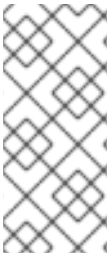
リストフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl** は、パラメーターリスト内のフィールドの名前とタイプを指定します。基本パラメーターフィールド (**CHAR**、**DATE**、**BCD**、**TIME**、**BYTE**、**NUM**、**FLOAT**、**INT**、**INT1**、**INT2**、**DECF16**、**DECF34**、**STRING**、**XSTRING**) の場合、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを次の表に示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。
byteLength	-	非 Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。
unicodeByteLength	-	Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。
decimals	**0**	フィールド値の 10 進数の数。パラメータータイプ BCD および FLOAT の場合は必須です。
任意	**false**	**true** の場合、フィールドはオプションであり、RFC 呼び出しで設定する必要はありません。

すべての基本パラメーターフィールドでは、フィールドメタデータオブジェクトで **name**、**type**、**byteLength**、および **unicodeByteLength** プロパティーを指定する必要があることに注意してください。さらに、**BCD**、**FLOAT**、**DECF16**、および **DECF34** フィールドでは、フィールドメタデータオブジェクトで decimal プロパティーを指定する必要があります。

タイプ **TABLE** または **STRUCTURE** の複雑なパラメーターフィールドの場合、次の表に、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。
recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** が渡され、構造またはテーブル行のフィールドが指定されます。
任意	**false**	**true** の場合、フィールドはオプションであり、RFC 呼び出しで設定する必要はありません。



注記

すべての複合パラメーターフィールドでは、フィールドメタデータオブジェクトで **name**、**type**、および **recordMetaData** プロパティーを指定する必要があります。**recordMetaData** プロパティーの値は、レコードフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** であり、ネストされた構造の構造またはテーブル行の構造を指定します。

基本リストフィールドのメタデータの例

次のメタデータ設定では、オプションの **TICKET_PRICE** という名前の小数点以下 2 桁の 24 桁のパックされた BCD 数値パラメーターを指定します。

```
ListFieldMetaData metaData = new ListFieldMetaDataImpl();
metaData.setName("TICKET_PRICE");
metaData.setType(DataType.BCD);
metaData.setByteLength(12);
metaData.setUnicodeByteLength(24);
metaData.setDecimals(2);
metaData.setOptional(true);
```

複雑なリストフィールドのメタデータの例

次のメタデータ設定では、**connectionInfo** レコードメタデータオブジェクトによって行構造が指定された、**CONNINFO** という名前の必須の **TABLE** パラメーターを指定します。

```
ListFieldMetaData metaData = new ListFieldMetaDataImpl();
metaData.setName("CONNINFO");
metaData.setType(DataType.TABLE);
RecordMetaData connectionInfo = new RecordMetaDataImpl();
metaData.setRecordMetaData(connectionInfo);
```

3.89.4.4.3. レコードのメタデータのプロパティー

レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** は、ネストされた **STRUCTURE** または **TABLE** パラメーターの行の名前と内容を指定します。レコードメタデータオブジェクトは、レコードフィールドメタデータオブジェクトのリスト **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl** を維持します。これは、ネストされた構造またはテーブル行に存在するパラメーターを指定します。

次の表に、レコードメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	レコードの名前。
recordFieldMetaData	-	レコードフィールドメタデータオブジェクトのリスト、**org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**。構造体に含まれるフィールドを指定します。

注記

レコードメタデータオブジェクトのすべてのプロパティーが必要です。

レコードメタデータの例

次の例は、レコードメタデータオブジェクトを設定する方法を示しています。

```
RecordMetaData connectionInfo = new RecordMetaDataImpl();
connectionInfo.setName("CONNECTION_INFO");
connectionInfo.setRecordFieldMetaData(...);
```

3.89.4.4.4. レコードフィールドのメタデータプロパティー

レコードフィールドメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl** は、構造体でパラメーターフィールドの名前とタイプを指定します。

レコードフィールドメタデータオブジェクトは、ネストされた構造またはテーブル行内の個々のフィールド位置のオフセットを追加で指定する必要があることを除いて、パラメーターフィールドメタデータオブジェクトに似ています。個々のフィールドの非 Unicode オフセットと Unicode オフセットは、構造体または行内の前のフィールドの非 Unicode バイト長と Unicode バイト長の合計から計算して指定する必要があります。

注記

ネストされた構造およびテーブル行のフィールドのオフセットを適切に指定しないと、基礎となる JCo および ABAP ランタイムのパラメーターのフィールドストレージが重複し、RFC コールでの値の適切な転送が妨げられるます。

基本パラメーターフィールド
(**CHAR**、**DATE**、**BCD**、**TIME**、**BYTE**、**NUM**、**FLOAT**、**INT**、**INT1**、**INT2**、**DECF16**、**DECF34**、**STRING**、**XSTRING**) の場合、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを次の表に示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。
byteLength	-	非 Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。
unicodeByteLength	-	Unicode レイアウトのバイト単位のフィールド長。この値は、パラメーターのタイプによって異なります。
byteOffset	-	非 Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲んでいる構造内のフィールドのバイト位置です。
unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲んでいる構造内のフィールドのバイト位置です。
decimals	**0**	フィールド値の小数点以下の桁数。パラメータータイプ **BCD** および **FLOAT** にのみ必要です。

タイプ **TABLE** または **STRUCTURE** の複雑なパラメーターフィールドの場合、次の表に、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを示します。

名前	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。
byteOffset	-	非 Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲んでいる構造内のフィールドのバイト位置です。

unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット (バイト単位)。このオフセットは、囲んでいる構造内のフィールドのバイト位置です。
recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** が渡され、構造またはテーブル行のフィールドが指定されます。

基本レコードフィールドのメタデータの例

次のメタデータ設定は、**ARRDATE** という名前の **DATE** フィールドパラメーターを指定します。これは、非 Unicode レイアウトの場合は囲んでいる構造の 85 バイト、Unicode レイアウトの場合は囲んでいる構造の 170 バイトに配置されています。

```
FieldMetaData fieldMetaData = new FieldMetaDataImpl();
fieldMetaData.setName("FLTINFO");
fieldMetaData.setType(DataType.STRUCTURE);
fieldMetaData.setByteOffset(0);
fieldMetaData.setUnicodeByteOffset(0);
RecordMetaData flightInfo = new RecordMetaDataImpl();
fieldMetaData.setRecordMetaData(flightInfo);
```

複雑なレコードフィールドのメタデータの例

次のメタデータ設定は、**flightInfo** レコードメタデータオブジェクトによって指定された構造を持つ **FLTINFO** という名前の **STRUCTURE** フィールドパラメーターを指定します。パラメーターは、非 Unicode レイアウトと Unicode レイアウトの両方の場合に、囲んでいる構造の先頭に配置されます。

```
FieldMetaData fieldMetaData = new FieldMetaDataImpl();
fieldMetaData.setName("FLTINFO");
fieldMetaData.setType(DataType.STRUCTURE);
fieldMetaData.setByteOffset(0);
fieldMetaData.setUnicodeByteOffset(0);
RecordMetaData flightInfo = new RecordMetaDataImpl();
fieldMetaData.setRecordMetaData(flightInfo);
```

3.89.5. メッセージヘッダー

SAP コンポーネントは、次のメッセージヘッダーをサポートしています。

ヘッダー	説明

CamelSap.scheme	メッセージを処理する最後のエンドポイントの URI スキーム。次の値のいずれかを使用します。**sap-srfc-destination** **sap-trfc-destination** **sap-qrfc-destination** **sap-srfc-server** **sap-trfc-server** **sap-idoc-destination** **sap-idoclist-destination** **sap-qidoc-destination** **sap-qidoclist-destination** **sap-idoclist-server**
CamelSap.destinationName	メッセージを処理する最後の宛先エンドポイントの宛先名。
CamelSap.serverName	メッセージを処理する最後のサーバーエンドポイントのサーバー名。
CamelSap.queueName	メッセージを処理する最後のキューエンドポイントのキュー名。
CamelSap.rfcName	メッセージを処理する最後の RFC エンドポイントの RFC 名。
CamelSap.idocType	メッセージを処理する最後の IDoc エンドポイントの IDoc タイプ。
CamelSap.idocTypeExtension	メッセージを処理する最後の IDoc エンドポイントの IDoc タイプ拡張 (存在する場合)。
CamelSap.systemRelease	メッセージを処理する最後の IDoc エンドポイントのシステムリリース (存在する場合)。
CamelSap.applicationRelease	メッセージを処理する最後の IDoc エンドポイントのアプリケーションリリース (存在する場合)。

3.89.6. エクスチェンジプロパティー

SAP コンポーネントは、次のエクスチェンジプロパティーを追加します。

プロパティー	説明

CamelSap.destinationPropertiesMap	エクスチェンジによって検出された各 SAP 宛先のプロパティーを含むマップ。マップは宛先名によってキー付けされ、各エントリーはその宛先の設定プロパティーを含む **java.util.Properties** オブジェクトです。
CamelSap.serverPropertiesMap	エクスチェンジによって検出された各 SAP サーバーのプロパティーを含むマップ。マップはサーバー名でキー付けされ、各エントリーはそのサーバーの設定プロパティーを含む **java.util.Properties** オブジェクトです。

3.89.7. RFC のメッセージボディー

3.89.7.1. リクエストおよびレスポンスオブジェクト

SAP エンドポイントは、SAP リクエストオブジェクトを含むメッセージボディーを含むメッセージを受信することを想定しており、SAP レスポンスオブジェクトを含むメッセージボディーを含むメッセージを返します。SAP のリクエストとレスポンスは、各フィールドが事前定義されたデータ型を持つ名前付きフィールドを含む固定マップデータ構造です。

SAP リクエストとレスポンスの名前付きフィールドは、SAP エンドポイントに固有であり、各エンドポイントが受け入れる SAP リクエストとレスポンスのパラメーターを定義することに注意してください。SAP エンドポイントは、それに固有のリクエストとレスポンスオブジェクトを作成するファクトリーメソッドを提供します。

```
public class SAPEndpoint ... {
    ...
    public Structure getRequest() throws Exception;

    public Structure getResponse() throws Exception;
    ...
}
```

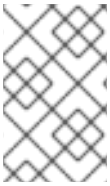
3.89.7.2. 構造物オブジェクト

SAP 要求オブジェクトと応答オブジェクトは両方とも、**org.fusesource.camel.component.sap.model.rfc.Structure** インターフェイスをサポートする構造オブジェクトとして Java で表されます。このインターフェイスは、**java.util.Map** インターフェイスと **org.eclipse.emf.ecore.EObject** インターフェイスの両方を拡張します。

```
public interface Structure extends org.eclipse.emf.ecore.EObject,
                                   java.util.Map<String, Object> {

    <T> T get(Object key, Class<T> type);

}
```

構造体オブジェクトのフィールド値には、マップインターフェイスのフィールドの getter メソッドを介してアクセスします。さらに、構造体インターフェイスは、フィールド値を取得するための型制限されたメソッドを提供します。

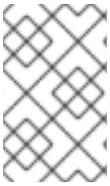
構造オブジェクトは、Eclipse モデリングフレームワーク (EMF) を使用してコンポーネントランタイムに実装され、そのフレームワークの **EObject** インターフェイスをサポートします。構造オブジェクトのインスタンスには、それが提供するフィールドのマップの構造と内容を定義および制限するメタデータが添付されています。このメタデータには、EMF が提供する標準的な方法を使用してアクセスし、イントロスペクションすることができます。詳細は、EMF のドキュメントを参照してください。



注記

構造体オブジェクトで定義されていないパラメーターを取得しようとすると、null が返されます。構造体で定義されていないパラメーターを設定しようとすると、例外が出力され、パラメーターの値を正しくない型で設定しようとします。

次のセクションで説明するように、構造体オブジェクトには、複合フィールド型 **STRUCTURE** および **TABLE** の値を含むフィールドを含めることができます。



注記

これらの型のインスタンスを作成して構造体に追加する必要はありません。これらのフィールド値のインスタンスは、必要に応じて、囲んでいる構造にアクセスするときにオンデマンドで作成されます。

3.89.7.3. フィールドの種類

SAP リクエストまたはレスポンスの構造オブジェクト内に存在するフィールドは、**基本** または **複合** のいずれかです。基本フィールドには単一のスカラー値が含まれますが、複合フィールドには基本タイプまたは複合タイプのフィールドが 1 つ以上含まれます。

3.89.7.3.1. 基本フィールドの種類

基本フィールドは、文字、数値、16 進数、または文字列のフィールドタイプです。次の表は、構造体オブジェクトに存在する可能性のある基本フィールドのタイプをまとめたものです。

フィールドタイプ	対応 Java 型	Byte Length	Unicode バイト長	数 小数 桁	説明
CHAR	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP タイプ 'C': 固定サイズの文字列。
DATE	**java.util.Date**	8	16	-	ABAP タイプ 'D': 日付 (形式: YYYYMMDD)。
BCD	**java.math.BigDecimal**	1 から 16	1 から 16	0 から 14	ABAP タイプ 'P': パックされた BCD 番号。BCD 番号には、1 バイトあたり 2 桁が含まれます。

TIME	**java.util.Date**	6	12	-	ABAP タイプ 'T': 時間 (形式: HHMMSS)。
BYTE	**byte[]**	1 から 65535	1 から 65535	-	ABAP タイプ 'X': 固定サイズのバイト配列。
NUM	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP タイプ 'N': 固定サイズの数値文字列。
FLOAT	**java.lang.Double**	8	8	0 から 15	ABAP タイプ 'F': 浮動小数点数。
INT	**java.lang.Integer**	4	4	-	ABAP タイプ 'I': 4 バイト整数。
INT2	**java.lang.Integer**	2	2	-	ABAP タイプ 'S': 2 バイト整数。
INT1	**java.lang.Integer**	1	1	-	ABAP タイプ 'B': 1 バイト整数。
DECF16	**java.match.BigDecimal**	8	8	16	ABAP タイプ 'decfloat16': 8 バイトの 10 進浮動小数点数。
DECF34	**java.math.BigDecimal**	16	16	34	ABAP タイプ 'decfloat34': 16 バイトの 10 進浮動小数点数。
STRING	**java.lang.String**	8	8	-	ABAP タイプ 'G': 可変長文字列。
XSTRING	**byte[]**	8	8	-	ABAP タイプ 'Y': 可変長バイト配列。

3.89.7.3.2. 文字フィールドの種類

文字フィールドには、基礎となる JCo および ABAP ランタイムで非 Unicode または Unicode 文字エンコーディングを使用できる固定サイズの文字列が含まれます。非 Unicode 文字列は、1 バイトあたり 1 文字をエンコードします。Unicode 文字列は、UTF-16 エンコーディングを使用して 2 バイトでエンコードされます。文字フィールドの値は、Java では **java.lang.String** オブジェクトとして表され、基礎となる JCo ランタイムが ABAP 表現への変換を担当します。

文字フィールドは、関連する **byteLength** および **unicodeByteLength** プロパティーでそのフィールド長を宣言します。これらのプロパティーは、各エンコーディングシステムでのフィールドの文字列の長さを決定します。

CHAR

CHAR 文字項目は、英数字を含むテキスト項目であり、ABAP タイプ C に対応します。

NUM

NUM 文字フィールドは、数字のみを含む数値テキストフィールドであり、ABAP タイプ N に対応します。

DATE

DATE 文字フィールドは、年、月、日が **YYYYMMDD** としてフォーマットされた 8 文字の日付フィールドであり、ABAP タイプ D に対応します。

TIME

TIME 文字フィールドは、時、分、および秒が **HHMMSS** としてフォーマットされた 6 文字の時間フィールドであり、ABAP タイプ T に対応します。

3.89.7.3.3. 数値フィールドの種類

数値フィールドには数値が含まれています。次の数値フィールドタイプがサポートされています。

INT

INT 数値フィールドは、基礎となる JCo および ABAP ランタイムで 4 バイトの整数値として格納される整数フィールドであり、ABAP タイプ I に対応します。**INT** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

INT2

INT2 数値フィールドは、基礎となる JCo および ABAP ランタイムに 2 バイトの整数値として格納される整数フィールドであり、ABAP タイプ S に対応します。**INT2** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

INT1

INT1 フィールドは、基になる JCo および ABAP ランタイム値に 1 バイトの整数値として格納される整数フィールドであり、ABAP タイプ B に対応します。**INT1** フィールド値は、Java では **java.lang.Integer** オブジェクトとして表されます。

FLOAT

FLOAT フィールドは、基礎となる JCo および ABAP ランタイムに 8 バイトの double 値として格納される 2 進浮動小数点数フィールドであり、ABAP タイプ F に対応します。**FLOAT** フィールドは、フィールドの値が関連する小数プロパティー。**FLOAT** フィールドの場合、この 10 進プロパティーは 1 ~ 15 桁の値を持つことができます。**FLOAT** フィールド値は、Java では **java.lang.Double** オブジェクトとして表されます。

BCD

BCD フィールドは、基礎となる JCo および ABAP ランタイムで 1 から 16 バイトのパック数として保管される 2 進化 10 進フィールドであり、ABAP タイプ P に対応します。パック数は、1 バイトあたり 2 桁の 10 進数を保管します。**BCD** フィールドは、関連する **byteLength** および **unicodeByteLength** プロパティーでフィールド長を宣言します。**BCD** フィールドの場合、これら

のプロパティーは 1 ~ 16 バイトの値を持つことができ、両方のプロパティーが同じ値になります。**BCD** フィールドは、関連付けられた decimal プロパティーで、フィールドの値に含まれる 10 進数の桁数を宣言します。**BCD** フィールドの場合、この 10 進プロパティーは 1 ~ 14 桁の値を持つことができます。**BCD** フィールド値は、Java では **java.math.BigDecimal** として表されます。

DECF16

DECF16 フィールドは、基礎となる JCo および ABAP ランタイムで 8 バイトの IEEE 754 decimal64 浮動小数点値として格納される 10 進浮動小数点であり、ABAP タイプ **decfloat16** に対応します。**DECF16** フィールドの値は、10 進数で 16 桁です。 **DECF16** フィールドの値は、Java では **java.math.BigDecimal** として表されます。

DECF34

DECF34 フィールドは、基礎となる JCo および ABAP ランタイムで 16 バイトの IEEE 754 decimal128 浮動小数点値として格納される 10 進浮動小数点であり、ABAP タイプ **decfloat34** に対応します。**DECF34** フィールドの値には、34 桁の 10 進数があります。 **DECF34** フィールドの値は、Java では **java.math.BigDecimal** として表されます。

3.89.7.3.4. 16 進フィールドタイプ

16 進数フィールドには生のバイナリーデータが含まれます。次の 16 進数フィールドタイプがサポートされています。

BYTE

BYTE フィールドは、基礎となる JCo および ABAP ランタイムにバイト配列として格納される固定サイズのバイト文字列であり、ABAP タイプ X に対応します。**BYTE** フィールドは、関連する **byteLength** および **unicodeByteLength** プロパティーでフィールド長を宣言します。**BYTE** フィールドの場合、これらのプロパティーは 1 ~ 65535 バイトの値を持つことができ、両方のプロパティーが同じ値になります。**BYTE** フィールドの値は、Java では **byte** オブジェクトとして表されます。

3.89.7.3.5. 文字列フィールドの種類

文字列フィールドは、可変長の文字列値を参照します。その文字列値の長さは実行時まで固定されません。文字列値のストレージは、基礎となる JCo および ABAP ランタイムで動的に作成されます。文字列フィールド自体のストレージは固定されており、文字列ヘッダーのみが含まれています。

STRING

STRING フィールドは、基礎となる JCo および ABAP ランタイムに 8 バイト値として格納された文字列を参照します。ABAP タイプ G に対応します。**STRING** 項目の値は、Java では **java.lang.String** オブジェクトとして表されます。

XSTRING

XSTRING フィールドは、基礎となる JCo および ABAP ランタイムに 8 バイト値として格納されたバイト文字列を参照します。ABAP タイプ Y に対応します。**STRING** 項目の値は、Java では **byte** オブジェクトとして表されます。

3.89.7.3.6. 複雑なフィールドタイプ

複合フィールドは、構造体またはテーブルフィールドタイプのいずれかです。次の表は、これらの複雑なフィールドタイプをまとめたものです。

フィールドタイプ	対応 Java 型	Byte Length	Unicode バイト長	数 小数 桁	説明

STRUCTURE	**org.fusesource.camel.component.sap.model.rfc.Structure**	個々のフィールドのバイト長の合計	個々のフィールドの Unicode バイト長の合計	-	ABAP タイプ 'u' & 'v': 異種構造
TABLE	**org.fusesource.camel.component.sap.model.rfc.Table**	行構造体のバイト長	行構造体の Unicode バイト長	-	ABAP タイプ 'h': テーブル

3.89.7.3.7. 構造体フィールドタイプ

STRUCTURE 項目には構造オブジェクトが含まれ、基礎となる JCo および ABAP ランタイムに ABAP 構造レコードとして格納されます。ABAP タイプ **u** または **v** のいずれかに対応します。**STRUCTURE** 項目の値は、Java ではインターフェイス **org.fusesource.camel.component.sap.model.rfc.Structure** を持つ構造オブジェクトとして表されます。

3.89.7.3.8. テーブルフィールドタイプ

TABLE フィールドにはテーブルオブジェクトが含まれ、基礎となる JCo および ABAP ランタイムに ABAP 内部テーブルとして格納されます。ABAP タイプ **h** に対応します。フィールドの値は、インタフェース **org.fusesource.camel.component.sap.model.rfc.Table** を持つテーブルオブジェクトによって Java で表されます。

3.89.7.3.9. テーブルオブジェクト

テーブルオブジェクトは、同じ構造を持つ構造オブジェクトの行を含む同種のリストデータ構造です。このインターフェイスは、**java.util.List** インターフェイスと **org.eclipse.emf.ecore.EObject** インターフェイスの両方を拡張します。

```
public interface Table<S extends Structure>
    extends org.eclipse.emf.ecore.EObject,
    java.util.List<S> {

    /**
     * Creates and adds a table row at the end of the row list
     */
    S add();

    /**
     * Creates and adds a table row at the index in the row list
     */
    S add(int index);

}
```

テーブルオブジェクト内の行のリストは、リストインターフェイスで定義された標準メソッドを使用してアクセスおよび管理されます。さらに、テーブルインターフェイスは、構造体オブジェクトを作成して行リストに追加するための 2 つのファクトリーメソッドを提供します。

テーブルオブジェクトは、Eclipse Modeling Framework (EMF) を使用してコンポーネントランタイムに実装され、そのフレームワークの EObject インターフェイスをサポートします。テーブルオブジェクトのインスタンスには、それが提供する行の構造と内容を定義および制限するメタデータが添付されています。このメタデータには、EMF が提供する標準的な方法を使用してアクセスし、イントロスペクションすることができます。詳細は、EMF のドキュメントを参照してください。

注記

間違った型の行構造値を追加または設定しようとすると、例外が出力されます。

3.89.8. IDoc のメッセージ本文

3.89.8.1. IDoc メッセージタイプ

IDoc Camel SAP エンドポイントの 1 つを使用する場合、メッセージ本文のタイプは、使用している特定のエンドポイントによって異なります。

sap-idoc-destination エンドポイントまたは **sap-qidoc-destination** エンドポイントの場合、メッセージ本文は **Document** タイプです。

```
org.fusesource.camel.component.sap.model.idoc.Document
```

sap-idoclist-destination エンドポイント、**sap-qidoclist-destination** エンドポイント、または **sap-idoclist-server** エンドポイントの場合、メッセージボディーは **DocumentList** タイプです。

```
org.fusesource.camel.component.sap.model.idoc.DocumentList
```

3.89.8.2. IDoc 文書モデル

Camel SAP コンポーネントの場合、IDoc ドキュメントは、基礎となる SAP IDoc API のラッパー API を提供する Eclipse Modeling Framework (EMF) を使用してモデル化されます。このモデルで最も重要なタイプは次のとおりです。

```
org.fusesource.camel.component.sap.model.idoc.Document
org.fusesource.camel.component.sap.model.idoc.Segment
```

Document タイプは、IDoc ドキュメントインスタンスを表します。概説すると、**Document** インターフェイスは次のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Document extends EObject {
   // Access the field values from the IDoc control record
   String getArchiveKey();
   void setArchiveKey(String value);
   String getClient();
   void setClient(String value);
   ...

   // Access the IDoc document contents
   Segment getRootSegment();
}
```

次の種類のメソッドが **Document** インターフェイスによって公開されます。

制御レコードにアクセスする方法

ほとんどのメソッドは、IDoc 制御レコードのフィールド値にアクセスまたは変更するためのものです。これらのメソッドの形式は **AttributeName** です。ここで、**AttributeName** はフィールド値の名前です。

ドキュメントコンテンツへのアクセス方法

getRootSegment メソッドは、ドキュメントコンテンツ (IDoc データレコード) へのアクセスを提供し、コンテンツを **Segment** オブジェクトとして返します。各 **Segment** オブジェクトには任意の数の子セグメントを含めることができ、セグメントは任意の程度にネストできます。
ただし、セグメント階層の正確なレイアウトは、文書の特定の **IDoc タイプ** によって定義されることに注意してください。したがって、セグメント階層を作成 (または読み取り) するときは、IDoc タイプによって定義された正確な構造に従う必要があります。

Segment タイプは、IDoc ドキュメントのデータレコードにアクセスするために使用されます。セグメントは、ドキュメントの IDoc タイプによって定義された構造に従って配置されます。概説すると、**Segment** インターフェイスは次のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Segment extends EObject, java.util.Map<String, Object> {
   // Returns the value of the '<em><b>Parent</b></em>' reference.
   Segment getParent();

   // Return an immutable list of all child segments
   <S extends Segment> EList<S> getChildren();

   // Returns a list of child segments of the specified segment type.
   <S extends Segment> SegmentList<S> getChildren(String segmentType);

   EList<String> getTypes();

   Document getDocument();

   String getDescription();

   String getType();

   String getDefinition();

   int getHierarchyLevel();

   String getIdocType();

   String getIdocTypeExtension();

   String getSystemRelease();

   String getApplicationRelease();

   int getNumFields();

   long getMaxOccurrence();
```

```
    long getMinOccurrence();

    boolean isMandatory();

    boolean isQualified();

    int getRecordLength();

    <T> T get(Object key, Class<T> type);
}
```

getChildren (String segmentType) メソッドは、新しい (ネストされた) 子をセグメントに追加する場合に特に便利です。次のように定義されているタイプ **SegmentList** のオブジェクトを返します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface SegmentList<S extends Segment> extends EObject, EList<S&gt; {
    S add();

    S add(int index);
}
```

したがって、**E1SCU_CRE** タイプのデータレコードを作成するには、次のような Java コードを使用できます。

```
Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();
```

3.89.8.3. IDoc と Document オブジェクトの関係

SAP ドキュメントによると、IDoc ドキュメントは次の主要部分で設定されています。

制御記録

コントロールレコード (IDoc ドキュメントのメタデータを含む) は、**Document** オブジェクトの属性によって表されます。

データ記録

データレコードは、セグメントのネストされた階層として構築される **Segment** オブジェクトによって表されます。**Document.getRootSegment** メソッドを介してルートセグメントにアクセスできます。

状況記録

Camel SAP コンポーネントでは、ステータスレコードはドキュメントモデルによって表され **ません**。ただし、制御レコードの **status** 属性を介して最新のステータス値にアクセスできます。

Document インスタンスの作成例

次の例は、Java で IDoc モデル API を使用して、IDoc タイプ **FLCUSTOMER_CREATEFROMDATA01** の IDoc ドキュメントを作成する方法を示しています。

例3.1 Java での IDoc ドキュメントの作成

```
// Java
import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.util.IDocUtil;

import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.DocumentList;
import org.fusesource.camel.component.sap.model.idoc.IdocFactory;
import org.fusesource.camel.component.sap.model.idoc.IdocPackage;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.model.idoc.SegmentChildren;
...
//
// Create a new IDoc instance using the modeling classes
//

// Get the SAP Endpoint bean from the Camel context.
// In this example, it's a 'sap-idoc-destination' endpoint.
SapTransactionalIDocDestinationEndpoint endpoint =
    exchange.getContext().getEndpoint(
        "bean:SapEndpointBeanID",
        SapTransactionalIDocDestinationEndpoint.class
    );

// The endpoint automatically populates some required control record attributes
Document document = endpoint.createDocument()

// Initialize additional control record attributes
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
document.setRecipientPartnerNumber("QUICKCLNT");
document.setRecipientPartnerType("LS");
document.setSenderPartnerNumber("QUICKSTART");
document.setSenderPartnerType("LS");

Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();

Segment E1BPSCUNEW_Segment =
E1SCU_CRE_Segment.getChildren("E1BPSCUNEW").add();
E1BPSCUNEW_Segment.put("CUSTNAME", "Fred Flintstone");
E1BPSCUNEW_Segment.put("FORM", "Mr.");
E1BPSCUNEW_Segment.put("STREET", "123 Rubble Lane");
E1BPSCUNEW_Segment.put("POSTCODE", "01234");
E1BPSCUNEW_Segment.put("CITY", "Bedrock");
E1BPSCUNEW_Segment.put("COUNTR", "US");
E1BPSCUNEW_Segment.put("PHONE", "800-555-1212");
E1BPSCUNEW_Segment.put("EMAIL", "fred@bedrock.com");
E1BPSCUNEW_Segment.put("CUSTTYPE", "P");
E1BPSCUNEW_Segment.put("DISCOUNT", "005");
E1BPSCUNEW_Segment.put("LANGU", "E");
```

3.89.9. ドキュメント属性

IDoc ドキュメント属性の表に、**Document** オブジェクトに設定できる制御レコード属性を示します。

表3.2 IDoc ドキュメントの属性

属性	長さ	SAP フィールド	説明
archiveKey	70	**ARCKEY**	EDI アーカイブキー
client	3	**MANDT**	クライアント
creationDate	8	**CREDAT**	IDoc が作成された日付
creationTime	6	**CRETIM**	IDoc が作成された時間
direction	1	**DIRECT**	方向
eDIMessage	14	**REFMES**	メッセージ参照
eDIMessageGroup	14	**REFGRP**	メッセージグループへの参照
eDIMessageType	6	**STDMES**	EDI メッセージタイプ
eDIStandardFlag	1	**STD**	EDI 規格
eDIStandardVersion	6	**STDVRS**	EDI 規格のバージョン
eDITransmissionFile	14	**REFINT**	エクスチェンジファイルへの参照
iDocCompoundType	8	**DOCTYP**	IDoc タイプ
iDocNumber	16	**DOCNUM**	IDoc 番号
iDocSAPRelease	4	**DOCREL**	IDoc の SAP リリース
iDocType	30	**IDOCTP**	基本 IDoc タイプの名前
iDocTypeExtension	30	**CIMTYP**	拡張タイプの名前
messageCode	3	**MESCOD**	論理メッセージコード
messageFunction	3	**MESFCT**	論理メッセージ機能
messageType	30	**MESTYP**	論理メッセージタイプ
outputMode	1	**OUTMOD**	出力モード

属性	長さ	SAP フィールド	説明
recipientAddress	10	**RCVSAD**	受信者アドレス (SADR)
recipientLogicalAddress	70	**RCVLAD**	受信者の論理アドレス
recipientPartnerFunction	2	**RCVPFC**	受信者のパートナー機能
recipientPartnerNumber	10	**RCVPRN**	受信機のパートナー番号
recipientPartnerType	2	**RCVPRT**	受信者のパートナータイプ
recipientPort	10	**RCVPOR**	受信側ポート (SAP システム、EDI サブシステム)
senderAddress		**SNDSAD**	送信者アドレス (SADR)
senderLogicalAddress	70	**SNDLAD**	送信者の論理アドレス
senderPartnerFunction	2	**SNDPFC**	差出人のパートナー機能
senderPartnerNumber	10	**SNDPRN**	送信者のパートナー番号
senderPartnerType	2	**SNDPRT**	送信者のパートナータイプ
senderPort	10	**SNDPOR**	送信側ポート (SAP システム、EDI サブシステム)
serialization	20	**SERIAL**	EDI/ALE: シリアル化フィールド
status	2	**STATUS**	IDoc のステータス
testFlag	1	**テスト**	テストフラグ

3.89.9.1. Java でドキュメント属性を設定する

Java で制御レコード属性を設定する場合、Java Bean プロパティーの通常の規則に従います。つま

り、属性値を取得および設定するために、**getName** および **setName** メソッドを介して **name** 属性にアクセスできます。たとえば、**iDocType**、**iDocTypeExtension**、および **messageType** 属性は、**Document** オブジェクトで次のように設定できます。

```
// Java
document.setIDocType("FLCUSTOMER_CREATEFROMDATA01");
document.setIDocTypeExtension("");
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
```

3.89.9.2. XML でドキュメント属性を設定する

XML で制御レコード属性を設定する場合、**idoc:Document** 要素に属性を設定する必要があります。たとえば、**iDocType**、**iDocTypeExtension** および **messageType** 属性は次のように設定できます。

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document ...
            iDocType="FLCUSTOMER_CREATEFROMDATA01"
            iDocTypeExtension=""
            messageType="FLCUSTOMER_CREATEFROMDATA" ... >
   ...
</idoc:Document>
```

3.89.10. トランザクションサポート

3.89.10.1. BAPI トランザクションモデル

SAP コンポーネントは、SAP とのアウトバウンド通信用に BAPI トランザクションモデルをサポートしています。**true** に設定されたトランザクションオプションを含む URL を持つ宛先エンドポイントは、必要に応じて、エンドポイントのアウトバウンド接続でステートフルセッションを開始し、Camel Synchronization オブジェクトをエクスチェンジに登録します。この Synchronization オブジェクトは、BAPI サービスメソッド **BAPI_TRANSACTION_COMMIT** を呼び出し、メッセージエクスチェンジの処理が完了するとステートフルセッションを終了します。メッセージエクスチェンジの処理が失敗した場合、Synchronization オブジェクトは BAPI サーバーメソッド **BAPI_TRANSACTION_ROLLBACK** を呼び出し、ステートフルセッションを終了します。

3.89.10.2. RFC トランザクションモデル

tRFC プロトコルは、一意のトランザクション識別子 (TID) で各トランザクション要求を識別することにより、AT-MOST-ONCE の配信と処理の保証を実現します。TID は、プロトコルで送信される各要求に付随します。tRFC プロトコルを使用する送信側アプリケーションは、要求を送信するときに、要求の各インスタンスを一意の TID で識別する必要があります。アプリケーションは、特定の TID を持つリクエストを複数回送信できますが、プロトコルは、リクエストが受信側システムで配信され、処理されるのは多くても 1 回であることを保証します。アプリケーションは、要求の送信時に通信エラーまたはシステムエラーが発生した場合に、指定された TID を使用してリクエストを再送信することを選択できます。したがって、そのリクエストが受信側システムで配信および処理されたかどうかは不明です。通信エラーが発生したときにリクエストを再送信することにより、tRFC プロトコルを使用するクライアントアプリケーションは、そのリクエストの配信と処理を確実に 1 回だけ保証できます。

3.89.10.3. どのトランザクションモデルを使用するか?

BAPI トランザクションは、SAP データベースで BAPI メソッドまたは RFC 関数によって実行される永続的なデータ変更に ACID 保証を課すという意味で、アプリケーションレベルのトランザクションです。RFC トランザクションは、BAPI メソッドや RFC 機能への要求に対して配信保証 (AT-MOST-

ONCE、EXACTLY-ONCE、EXACTLY-ONCE-IN-ORDER) を課すという意味で、通信トランザクションです。

3.89.10.4. トランザクション RFC 宛先エンドポイント

次の宛先エンドポイントは、RFC トランザクションをサポートしています。

- **sap-trfc-destination**
- **sap-qrfc-destination**

単一の Camel ルートには、複数のトランザクション RFC 宛先エンドポイントを含めることができ、複数の RFC 宛先にメッセージを送信したり、同じ RFC 宛先に複数回メッセージを送信したりすることもできます。これは、Camel SAP コンポーネントが、ルートを通過する各 **Exchange** オブジェクトの **多くの** トランザクション ID (TID) を追跡する必要がある可能性があることを意味します。ルート処理が失敗し、再試行する必要がある場合、状況は非常に複雑になります。RFC トランザクションセマンティクスは、ルート上の各 RFC 宛先が、最初に使用されたのと **同じ** TID を使用して呼び出されることを要求します (各宛先の TID が互いに異なる場合)。つまり、Camel SAP コンポーネントは、ルート上のどのポイントでどの TID が使用されたかを追跡し、この情報を記憶して、TID を正しい順序で再生できるようにする必要があります。

デフォルトでは、Camel は **Exchange** がルートのどこにいるかを知ることができるメカニズムを提供しません。このようなメカニズムを提供するには、**CurrentProcessorDefinitionInterceptStrategy** インターセプターを Camel ランタイムにインストールする必要があります。Camel SAP コンポーネントがルート内の TID を追跡するには、このインターセプターを Camel ランタイムにインストールする必要があります。

3.89.10.5. トランザクション RFC サーバーエンドポイント

次のサーバーエンドポイントは、RFC トランザクションをサポートしています。

- **sap-trfc-server**

トランザクションリクエストを処理する Camel エクスチェンジで処理エラーが発生すると、Camel は標準のエラー処理メカニズムを通じて処理エラーを処理します。エクスチェンジを処理する Camel ルートがエラーを呼び出し元に伝播するように設定されている場合、エクスチェンジを開始した SAP サーバーエンドポイントは失敗を記録し、送信側の SAP システムにエラーが通知されます。送信側の SAP システムは、同じ TID を持つ別のトランザクションリクエストを送信して応答し、リクエストを再度処理できます。

3.89.11. RFC の XML シリアライゼーション

SAP 要求および応答オブジェクトは、これらのオブジェクトを XML ドキュメントとの間でシリアル化できるようにする XML シリアル化形式をサポートしています。

3.89.11.1. XML 名前空間

リポジトリー内の各 RFC は、リクエストオブジェクトとレスポンスオブジェクトのシリアル化された形式を設定する要素に対して、特定の XML 名前空間を定義します。この名前空間 URL の形式は次のとおりです。

```
http://sap.fusesource.org/rfc>/<Repository Name>/<RFC Name>
```

RFC 名前空間 URL には、共通の **http://sap.fusesource.org/rfc** 接頭辞があり、その後に RFC のメタデータが定義されているリポジトリーの名前が続きます。URL の最後のコンポーネントは、RFC 自体の名前です。

3.89.11.2. リクエストとレスポンスの XML ドキュメント

SAP リクエストオブジェクトは XML ドキュメントにシリアル化され、そのドキュメントのルート要素は Request という名前で、リクエストの RFC の名前空間によってスコープが設定されます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Request>
```

SAP レスポンスオブジェクトは XML ドキュメントにシリアル化され、そのドキュメントのルート要素は Response という名前で、レスポンス応答の RFC の名前空間によってスコープが設定されます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Response
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Response>
```

3.89.11.3. 構造体フィールド

パラメーターリストまたはネストされた構造体の構造体フィールドは、要素としてシリアル化されます。シリアル化された構造体の要素名は、それが含まれるパラメーターリスト、構造体、またはテーブルの行エントリー内の構造体のフィールド名に対応します。

```
<BOOK_FLIGHT:FLTINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:FLTINFO>
```

次の例のように、RFC 名前空間の構造要素の型名は、構造を定義するレコードメタデータオブジェクトの名前に対応することに注意してください。

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
   ...
   <xs:complexType name="FLTINFO_STRUCTURE”>
   ...
   </xs:complexType>
   ...
</xs:schema>
```

この区別は、JAXB Bean を指定して構造をマーシャリングおよびアンマーシャリングする場合に重要になります。

3.89.11.4. テーブルフィールド

パラメーターリストまたはネストされた構造のテーブルフィールドは、要素としてシリアル化されま

す。シリアル化された構造体の要素名は、それが存在する外側のパラメーターリスト、構造体、またはテーブル行エントリー内のテーブルのフィールド名に対応します。テーブル要素には、テーブルの行エントリーのシリアル化された値を保持する一連の行要素が含まれます。

```
<BOOK_FLIGHT:CONNINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   <row ... > ... </row>
   ...
   <row ... > ... </row>
</BOOK_FLIGHT:CONNINFO>
```

RFC 名前空間の table 要素の型名は、**_TABLE** の接尾辞が付いたテーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応することに注意してください。次の例のように、RFC 名のテーブル行要素の型名は、テーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応します。

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
   ...
   <xs:complextType name="CONNECTION_INFO_STRUCTURE_TABLE”>
      <xs:sequence>
         <xs:element
            name="row”
            minOccures="0”
            maxOccurs="unbounded”
            type="CONNECTION_INFO_STRUCTURE”/>
         ...
         <xs:sequence>
      </xs:sequence>
   </xs:complexType>

   <xs:complextType name="CONNECTION_INFO_STRUCTURE”>
         ...
   </xs:complexType>
   ...
</xs:schema>
```

この区別は、JAXB Bean を指定して構造をマーシャリングおよびアンマーシャリングする場合に重要になります。

3.89.11.5. Elementary フィールド

パラメーターリストまたはネストされた構造体の Elementary フィールドは、囲んでいるパラメーターリストまたは構造体の要素の属性としてシリアル化されます。シリアル化されたフィールドの属性名は、次の例のように、それが存在する囲んでいるパラメーターリスト、構造体、またはテーブル行エントリー内のフィールドのフィールド名に対応します。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
    CUSTNAME="James Legrand"
   PASSFORM="Mr"
   PASSNAME="Travelin Joe"
   PASSBIRTH="1990-03-17T00:00:00.000-0500"
```

```
    FLIGHTDATE="2014-03-19T00:00:00.000-0400"
    TRAVELAGENCYNUMBER="00000110"
    DESTINATION_FROM="SFO"
    DESTINATION_TO="FRA"/>
```

3.89.11.6. 日付と時刻の形式

日付と時刻のフィールドは、次の形式を使用して属性値にシリアル化されます。

```
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

日付フィールドは、年、月、日、およびタイムゾーンのコンポーネントセットのみでシリアル化されます。

```
DEPDATE="2014-03-19T00:00:00.000-0400"
```

時間フィールドは、時、分、秒、ミリ秒、およびタイムゾーンコンポーネントセットのみでシリアル化されます。

```
DEPTIME="1970-01-01T16:00:00.000-0500"
```

3.89.12. IDoc の XML シリアル化

IDoc メッセージ本文は、組み込み型コンバーターを使用して、XML 文字列形式にシリアル化できます。

3.89.12.1. XML 名前空間

シリアル化された各 IDoc は、次の一般的な形式を持つ XML 名前空間に関連付けられています。

```
http://sap.fusesource.org/idoc/repositoryName/idocType/idocTypeExtension/systemRelease/applicationRelease
```

repositoryName (リモート SAP メタデータリポジトリーの名前) と **idocType** (IDoc ドキュメントタイプ) の両方が必須ですが、名前空間の他のコンポーネントは空白のままにすることができます。たとえば、次のような XML 名前空間を持つことができます。

```
http://sap.fusesource.org/idoc/MY_REPO/FLCUSTOMER_CREATEFROMDATA01///
```

3.89.12.2. 組み込み型コンバーター

Camel SAP コンポーネントには組み込み型コンバーターがあり、**Document** オブジェクトまたは **DocumentList** オブジェクトを String 型に変換したり、**String** 型から変換したりできます。

たとえば、**Document** オブジェクトを XML 文字列にシリアル化するには、次の行を XML DSL のルートに追加するだけです。

```
<convertBodyTo type="java.lang.String">;
```

このアプローチを使用して、シリアライズされた XML メッセージを **Document** オブジェクトにすることもできます。たとえば、現在のメッセージ本文がシリアル化された XML 文字列である場合、XML DSL のルートに次の行を追加することで、それを **Document** オブジェクトに戻すことができます。

```
<convertBodyTo type="org.fusesource.camel.component.sap.model.idoc.Document">
```

3.89.12.3. XML 形式のサンプル IDoc メッセージ本文

IDoc メッセージを **String** に変換すると、ルート要素が **idoc:Document** (単一のドキュメントの場合) または **idoc:DocumentList** (ドキュメントのリストの場合) のいずれかである XML ドキュメントにシリアライズされます。これは、**idoc:Document** 要素にシリアライズされた単一の IDoc ドキュメントであることを示しています。

例3.2 XML の IDoc メッセージボディー

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:FLCUSTOMER_CREATEFROMDATA01--
-="http://sap.fusesource.org/idoc/XXX/FLCUSTOMER_CREATEFROMDATA01///"
    xmlns:idoc="http://sap.fusesource.org/idoc"
    creationDate="2015-01-28T12:39:13.980-0500"
    creationTime="2015-01-28T12:39:13.980-0500"
    iDocType="FLCUSTOMER_CREATEFROMDATA01"
    iDocTypeExtension=""
    messageType="FLCUSTOMER_CREATEFROMDATA"
    recipientPartnerNumber="QUICKCLNT"
    recipientPartnerType="LS"
    senderPartnerNumber="QUICKSTART"
    senderPartnerType="LS">
  <rootSegment xsi:type="FLCUSTOMER_CREATEFROMDATA01---:ROOT" document="/">
    <segmentChildren parent="//@rootSegment">
      <E1SCU_CRE parent="//@rootSegment" document="/">
        <segmentChildren parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0">
          <E1BPSCUNEW parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0"
              document="/"
              CUSTNAME="Fred Flintstone" FORM="Mr."
              STREET="123 Rubble Lane"
              POSTCODE="01234"
              CITY="Bedrock"
              COUNTR="US"
              PHONE="800-555-1212"
              EMAIL="fred@bedrock.com"
              CUSTTYPE="P"
              DISCOUNT="005"
              LANGU="E"/>
        </segmentChildren>
      </E1SCU_CRE>
    </segmentChildren>
  </rootSegment>
</idoc:Document>
```

3.89.13. 例 1: SAP からのデータの読み取り

この例は、SAP から **FlightCustomer** ビジネスオブジェクトデータを読み取るルートを示しています。ルートは、データを取得するために SAP 同期 RFC 宛先エンドポイントを使用して、**FlightCustomer** BAPI メソッド **BAPI_FLCUST_GETLIST** を呼び出します。

3.89.13.1. ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
from("direct:getFlightCustomerInfo")
    .to("bean:createFlightCustomerGetListRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST")
    .to("bean:returnFlightCustomerInfo");
```

3.89.13.2. ルートの XML DSL

また、同じルートの Spring DSL は次のとおりです。

```
<route>
    <from uri="direct:getFlightCustomerInfo"/>
    <to uri="bean:createFlightCustomerGetListRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST"/>
    <to uri="bean:returnFlightCustomerInfo"/>
</route>
```

3.89.13.3. createFlightCustomerGetListRequest bean

createFlightCustomerGetListRequest Bean は、後続の SAP エンドポイントの RFC 呼び出しで使用される exchange メソッドで SAP 要求オブジェクトを構築するロールを果たします。次のコードスニペットは、リクエストオブジェクトを作成する一連の操作を示しています。

```
public void create(Exchange exchange) throws Exception {

    // Get SAP Endpoint to be called from context.
    SapSynchronousRfcDestinationEndpoint endpoint =
        exchange.getContext().getEndpoint("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST",
                                          SapSynchronousRfcDestinationEndpoint.class);

    // Retrieve bean from message containing Flight Customer name to
    // look up.
    BookFlightRequest bookFlightRequest =
        exchange.getIn().getBody(BookFlightRequest.class);

    // Create SAP Request object from target endpoint.
    Structure request = endpoint.getRequest();

    // Add Customer Name to request if set
    if (bookFlightRequest.getCustomerName() != null &&
        bookFlightRequest.getCustomerName().length() > 0) {
            request.put("CUSTOMER_NAME",
                        bookFlightRequest.getCustomerName());
        }
    } else {
        throw new Exception("No Customer Name");
    }
```

```
    // Put request object into body of exchange message.
    exchange.getIn().setBody(request);
}
```

3.89.13.4. returnFlightCustomerInfo bean

returnFlightCustomerInfo Bean は、前の SAP エンドポイントから受け取った exchange メソッドで、SAP レスポンスオブジェクトからデータを展開するロールを果たします。次のコードスニペットは、レスポンスオブジェクトからデータを抽出する一連の操作を示しています。

```
public void createFlightCustomerInfo(Exchange exchange) throws Exception {

    // Retrieve SAP response object from body of exchange message.
    Structure flightCustomerGetListResponse =
        exchange.getIn().getBody(Structure.class);

    if (flightCustomerGetListResponse == null) {
        throw new Exception("No Flight Customer Get List Response");
    }

    // Check BAPI return parameter for errors
    @SuppressWarnings("unchecked")
    Table<Structure> bapiReturn =
        flightCustomerGetListResponse.get("RETURN", Table.class);
    Structure bapiReturnEntry = bapiReturn.get(0);
    if (bapiReturnEntry.get("TYPE", String.class) != "S") {
        String message = bapiReturnEntry.get("MESSAGE", String.class);
        throw new Exception("BAPI call failed: " + message);
    }

    // Get customer list table from response object.
    @SuppressWarnings("unchecked")
    Table<? extends Structure> customerList =
        flightCustomerGetListResponse.get("CUSTOMER_LIST", Table.class);

    if (customerList == null || customerList.size() == 0) {
        throw new Exception("No Customer Info.");
    }

    // Get Flight Customer data from first row of table.
    Structure customer = customerList.get(0);

    // Create bean to hold Flight Customer data.
    FlightCustomerInfo flightCustomerInfo = new FlightCustomerInfo();

    // Get customer id from Flight Customer data and add to bean.
    String customerId = customer.get("CUSTOMERID", String.class);
    if (customerId != null) {
        flightCustomerInfo.setCustomerNumber(customerId);
    }

    ...

    // Put bean into body of exchange message.
```

```
    exchange.getIn().setHeader("flightCustomerInfo", flightCustomerInfo);

}
```

3.89.14. 例 2: SAP へのデータの書き込み

この例は、SAP で **FlightTrip** ビジネスオブジェクトインスタンスを作成するルートを示しています。ルートは、**FlightTrip** BAPI メソッド **BAPI_FLTRIP_CREATE** を呼び出し、宛先エンドポイントを使用してオブジェクトを作成します。

3.89.14.1. ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
from("direct:createFlightTrip")
    .to("bean:createFlightTripRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true")
    .to("bean:returnFlightTripResponse");
```

3.89.14.2. ルートの XML DSL

また、同じルートの Spring DSL は次のとおりです。

```
<route>
    <from uri="direct:createFlightTrip"/>
    <to uri="bean:createFlightTripRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true"/>
    <to uri="bean:returnFlightTripResponse"/>
</route>
```

3.89.14.3. トランザクションサポート

SAP エンドポイントの URL では、**transacted** オプションが **true** に設定されていることに注意してください。このオプションを有効にすると、エンドポイントは、RFC 呼び出しを呼び出す前に SAP トランザクションセッションが開始されていることを確認します。このエンドポイントの RFC は SAP で新しいデータを作成するため、ルートの変更を SAP で永続的にするには、このオプションが必要です。

3.89.14.4. リクエストパラメーターの設定

createFlightTripRequest および **returnFlightTripResponse** Bean は、前の例で示したのと同じ一連の操作に従って、リクエストパラメーターを SAP リクエストに入力し、SAP レスポンスからレスポンスパラメーターをそれぞれ展開します。

3.89.15. 例 3: SAP からのリクエストの処理

この例では、SAP から **BOOK_FLIGHT** RFC へのリクエストを処理するルートを示します。これは、ルートによって実装されます。さらに、JAXB を使用して SAP リクエストオブジェクトとレスポンスオブジェクトをカスタム Bean にアンマーシャリングおよびマーシャリングする、コンポーネントの XML シリアライゼーションサポートを示します。

このルートは、旅行代理店 **FlightCustomer** に代わって **FlightTrip** ビジネスオブジェクトを作成します。ルートは、最初に、SAP サーバーエンドポイントによって受信された SAP リクエストオブジェクトをカスタム JAXB Bean に非整列化します。次に、このカスタム Bean はエクスチェンジで 3 つのサ

ブルートにマルチキャストされ、フライト旅行の作成に必要な旅行代理店、フライト接続、乗客情報が収集されます。最後のサブルートは、前の例で示したように、SAP でフライトトリップオブジェクトを作成します。最後のサブルートは、SAP レスポンスオブジェクトにマーシャリングされ、サーバーエンドポイントによって返されるカスタム JAXB Bean も作成して返します。

3.89.15.1. ルートの Java DSL

サンプルルートの Java DSL は次のとおりです。

```
DataFormat jaxb = new JaxbDataFormat("org.fusesource.sap.example.jaxb");

from("sap-srfc-server:nplserver:BOOK_FLIGHT")
    .unmarshal(jaxb)
    .multicast()
    .to("direct:getFlightConnectionInfo",
        "direct:getFlightCustomerInfo",
        "direct:getPassengerInfo")
    .end()
    .to("direct:createFlightTrip")
    .marshal(jaxb);
```

3.89.15.2. ルートの XML DSL

同じルートの XML DSL は次のとおりです。

```
<route>
    <from uri="sap-srfc-server:nplserver:BOOK_FLIGHT"/>
    <unmarshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </unmarshal>
    <multicast>
        <to uri="direct:getFlightConnectionInfo"/>
        <to uri="direct:getFlightCustomerInfo"/>
        <to uri="direct:getPassengerInfo"/>
    </multicast>
    <to uri="direct:createFlightTrip"/>
    <marshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </marshal>
</route>
```

3.89.15.3. BookFlightRequest bean

次のリストは、シリアライズされた形式の SAP **BOOK_FLIGHT** リクエストオブジェクトからアンマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="Request",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightRequest {

    @XmlAttribute(name="CUSTNAME")
    private String customerName;
```

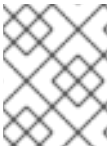
```
    @XmlAttribute(name="FLIGHTDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date flightDate;

    @XmlAttribute(name="TRAVELAGENCYNUMBER")
    private String travelAgencyNumber;

    @XmlAttribute(name="DESTINATION_FROM")
    private String startAirportCode;

    @XmlAttribute(name="DESTINATION_TO")
    private String endAirportCode;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlAttribute(name="CLASS")
    private String flightClass;

    ...
}
```

3.89.15.4. BookFlightResponse Bean

次のリストは、シリアライズされた形式の SAP **BOOK_FLIGHT** レスポンスオブジェクトにマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="Response",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightResponse {

    @XmlAttribute(name="TRIPNUMBER")
    private String tripNumber;

    @XmlAttribute(name="TICKET_PRICE")
    private BigDecimal ticketPrice;

    @XmlAttribute(name="TICKET_TAX")
    private BigDecimal ticketTax;

    @XmlAttribute(name="CURRENCY")
    private String currency;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
```

```
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlElement(name="FLTINFO")
    private FlightInfo flightInfo;

    @XmlElement(name="CONNINFO")
    private ConnectionInfoTable connectionInfo;

    ...
}
```



注記

レスポンスオブジェクトの複雑なパラメーターフィールドは、応答の子要素としてシリアル化されます。

3.89.15.5. FlightInfo ビーン

次のリストは、複雑な構造体パラメーター **FLTINFO** のシリアル化された形式にマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="FLTINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class FlightInfo {

    @XmlAttribute(name="FLIGHTTIME")
    private String flightTime;

    @XmlAttribute(name="CITYFROM")
    private String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureTime;

    @XmlAttribute(name="CITYTO")
    private String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalTime;
```

```
    ...
}
```

3.89.15.6. ConnectionInfoTable Bean

次のリストは、複雑なテーブルパラメーター **CONNINFO** のシリアル化された形式にマーシャリングする JAXB Bean を示しています。JAXB Bean のルート要素タイプの名前は、接尾辞 **_TABLE** が付いた行構造体タイプの名前に対応し、Bean には行要素のリストが含まれていることに注意してください。

```
@XmlRootElement(name="CONNINFO_TABLE",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfoTable {

    @XmlElement(name="row")
    List<ConnectionInfo> rows;

    ...
}
```

3.89.15.7. ConnectionInfo bean

次のリストは、上記のテーブルの行要素のシリアル化された形式にマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="CONNINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfo {

    @XmlAttribute(name="CONNID")
    String connectionId;

    @XmlAttribute(name="AIRLINE")
    String airline;

    @XmlAttribute(name="PLANETYPE")
    String planeType;

    @XmlAttribute(name="CITYFROM")
    String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureTime;

    @XmlAttribute(name="CITYTO")
    String cityTo;
```

```
    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalTime;

    ...
}
```

3.90. XQUERY

XQuery および Saxon を使用して XML ペイロードをクエリーまたは変換します。

3.90.1. 含まれるもの

- XQuery コンポーネント, URI 構文： **xquery:resourceUri**
- XQuery 言語

使用方法と設定の詳細については、上記リンクを参照してください。

3.90.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.90.3. 追加の Camel Quarkus 設定

このコンポーネントは、クラスパスから XQuery 定義をロードできます。ネイティブモードでも機能させるには、**quarkus.native.resources.includes** プロパティーを使用して、クエリーをネイティブ実行可能ファイルに明示的に埋め込む必要があります。

たとえば、以下の 2 つのルートは、それぞれ **myxquery.txt** および **another-xquery.txt** という名前の 2 つのクラスパスリソースから XQuery スクリプトをロードします。

```
from("direct:start").transform().xquery("resource:classpath:myxquery.txt", String.class);
from("direct:start").to("xquery:another-xquery.txt");
```

これら (**.txt** ファイルに保存されている可能性のある他のクエリー) をネイティブイメージに含めるには、**application.properties** ファイルに次のようなものを追加する必要があります。

```
quarkus.native.resources.includes = *.txt
```

3.91. SCHEDULER

java.util.concurrent.ScheduledExecutorService を使用して、指定された間隔でメッセージを生成します。

3.91.1. 含まれるもの

- Scheduler コンポーネント、URI 構文： **scheduler:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.91.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.92. SEDA

同じ JVM の Camel コンテキストから別のエンドポイントを非同期に呼び出します。

3.92.1. 含まれるもの

- SEDA コンポーネント、URI 構文： **seda:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.92.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.93. SERVLET

Servlet によって HTTP リクエストを処理します。

3.93.1. 含まれるもの

- Servlet コンポーネント、URI 構文： **servlet:contextPath**

使用方法と設定の詳細は、上記リンクを参照してください。

3.93.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.93.3. ネイティブモードの transferException オプション

ネイティブモードで **transferException** オプションを使用するには、オブジェクトのシリアル化のサポートを有効にする必要があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

また、シリアル化する予定の例外クラスのシリアル化を有効にする必要があります。例:

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

3.93.4. 追加の Camel Quarkus 設定

設定プロパティー	タイプ	デフォルト
quarkus.camel.servlet.url-patterns CamelServlet にアクセスできるパスパターンのコンマ区切りリスト。パスパターンの例： /*、/**services**/*	string	
quarkus.camel.servlet.servlet-class **url-patterns**に一致するパスを提供するサーブレットクラスの完全修飾名	string	**org.apache.camel.component.servlet.CamelHttpTransportServlet**
quarkus.camel.servlet.servlet-name **web.xml** ファイルまたは **jakarta.servlet.annotation.WebServlet#name （）アノテーションで定義されている servlet** Name。	string	**CamelServlet**
quarkus.camel.servlet."named-servlets".url-patterns CamelServlet にアクセスできるパスパターンのコンマ区切りリスト。パスパターンの例： /*、/**services**/*	string	

設定プロパティー	タイプ	デフォルト
quarkus.camel.servlet."named-servlets".servlet-class **url-patterns**に一致するパスを提供するサーブレットクラスの完全修飾名	string	org.apache.camel.component.servlet.CamelHttpTransportServlet
quarkus.camel.servlet."named-servlets".servlet-name **web.xml** ファイルまたは **jakarta.servlet.annotation.WebServlet#name （）アノテーションで定義されている servlet** Name。	string	CamelServlet

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.94. SLACK

Slack との間でメッセージを送受信します。

3.94.1. 含まれるもの

- Slack コンポーネント、URI 構文： **slack:channel**

使用方法と設定の詳細は、上記リンクを参照してください。

3.94.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.94.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.95. SNMP

トラップを受信し、SNMP (Simple Network Management Protocol) 対応デバイスをポーリングします。

3.95.1. 含まれるもの

- SNMP コンポーネント、URI 構文： **snmp:host:port**

使用方法と設定の詳細は、上記リンクを参照してください。

3.95.2. Maven コーディネート

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

3.96. SOAP DATAFORMAT

Java オブジェクトを SOAP メッセージにマーシャリングし、その逆も行います。

3.96.1. 含まれるもの

- SOAP データフォーマット

使用方法と設定の詳細は、上記リンクを参照してください。

3.96.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.97. SPLUNK

Splunk でイベントを公開または検索します。

3.97.1. 含まれるもの

- Splunk コンポーネント、URI 構文： **splunk:name**

使用方法と設定の詳細は、上記リンクを参照してください。

3.97.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.97.3. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.98. SPLUNK HEC

splunk コンポーネントを使用すると、HTTP Event Collector を使用して Splunk でイベントを公開できます。

3.98.1. 含まれるもの

- Splunk HEC コンポーネント、URI 構文： **splunk-hec:splunkURL/token**

使用方法と設定の詳細は、上記リンクを参照してください。

3.98.2. Maven コーディネート

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-splunk-hec</artifactId>
</dependency>
```

3.99. SQL

SQL クエリーを実行します。

3.99.1. 含まれるもの

- SQL コンポーネント、URI 構文： **sql:query**
- SQL Stored Procedure コンポーネント、URI 構文： **sql-stored:template**

使用方法と設定の詳細については、上記リンクを参照してください。

3.99.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.99.3. 追加の Camel Quarkus 設定

3.99.3.1. データソースの設定

このエクステンションは、**DataSource** のサポートに Quarkus Agroal を活用します。**DataSource** の設定は、設定プロパティーを介して実行できます。

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=your-username
quarkus.datasource.password=your-password
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/your-database
quarkus.datasource.jdbc.max-size=16
```

Camel SQL コンポーネントは、レジストリーから **DataSource** Bean を自動的に解決します。複数のデータソースを設定する場合、URI オプション **datasource** または **dataSourceRef** を使用して、SQL エンドポイントで使用するデータソースを指定できます。詳細については、SQL コンポーネントのドキュメントを参照してください。

3.99.3.1.1. Quarkus Dev Services によるゼロ設定

開発モードとテストモードでは、Configuration Free Databases を利用できます。Camel SQL コンポーネントは、選択した JDBC ドライバータイプに一致するデータベースのローカルコンテナー化インスタンスを指す **DataSource** を使用するように自動的に設定されます。

3.99.3.2. SQL スクリプト

クラスパスからスクリプトファイルを参照するために **sql** または **sql-stored** エンドポイントを設定する場合は、以下の設定プロパティーを設定して、それらがネイティブモードで利用できるようにします。

```
quarkus.native.resources.includes = queries.sql, sql/*.sql
```

3.99.3.3. ネイティブモードでの SQL 集約リポジトリー

ネイティブモードで **JdbcAggregationRepository** などの SQL 集約リポジトリーを使用するには、ネイティブのシリアライズサポートを有効化 する必要があります。

さらに、エクスチェンジボディがカスタムタイプである場合、クラス宣言に **@RegisterForReflection(serialization = true)** のアノテーションを付けてシリアライズ用に登録する必要があります。

3.100. TELEGRAM

Telegram Bot API として動作するメッセージを送受信します。

3.100.1. 含まれるもの

- Telegram コンポーネント、URI 構文： **telegram:type**

使用方法と設定の詳細は、上記リンクを参照してください。

3.100.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.100.3. 使用方法

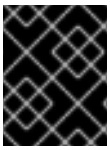
3.100.4. Webhook モード

Telegram エクステンションは、Webhook モードでの使用をサポートしています。

webhook モードを有効化するには、アプリケーションに REST 実装を追加する必要があります。たとえば、Maven ユーザーは、**pom.xml** ファイルに **camel-quarkus-rest** エクステンションを追加できます。

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

3.100.4.1. Webhook



重要

Red Hat build of Apache Camel for Quarkus のこのリリースでは、Webhook モードはサポートされていません。

3.100.5. ネイティブモードの SSL

このエクステンションは、ネイティブモードでの SSL サポートを自動的に有効にします。したがって、自分で **quarkus.ssl.native=true** を **application.properties** に追加する必要はありません。Quarkus SSL ガイド も参照してください。

3.101. TIMER

java.util.Timer を使用して、指定された間隔でメッセージを生成します。

3.101.1. 含まれるもの

- Timer コンポーネント、URI 構文： **timer:timerName**

使用方法と設定の詳細は、上記リンクを参照してください。

3.101.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.102. VALIDATOR

XML スキーマと JAXP 検証を使用してペイロードを検証します。

3.102.1. 含まれるもの

- Validator コンポーネント、URI 構文： **validator:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.102.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.103. VELOCITY

Velocity テンプレートを使用してメッセージを変換します。

3.103.1. 含まれるもの

- velocity コンポーネント、URI 構文： **velocity:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.103.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.103.3. 使用方法

3.103.3.1. ネイティブモードのドメインオブジェクトとしてのカスタムボディ

カスタムオブジェクトをメッセージ本文として使用し、ネイティブモードのテンプレートでそのプロパティーを参照する場合、すべてのクラスをリフレクション用に登録する必要があります (ドキュメントを参照)。

以下に例を示します。

```
@RegisterForReflection
public interface CustomBody {
}
```

3.103.4. ネイティブモードの allowContextMapAll オプション

allowContextMapAll オプションはネイティブモードではサポートされていません。これは、**CamelContext** や **Exchange** などのセキュリティーに敏感な Camel コアクラスへのリフレクティブアクセスが必要なためです。これはセキュリティーリスクとみなされるため、この機能へのアクセスはデフォルトでは提供されません。

3.103.5. 追加の Camel Quarkus 設定

通常、このコンポーネントはクラスパスから Velocity テンプレートをロードします。ネイティブモードでも機能させるには、**quarkus.native.resources.includes** プロパティーを使用して、テンプレートをネイティブ実行可能ファイルに明示的に埋め込む必要があります。

たとえば、以下のルートは、**template/simple.vm** という名前のクラスパスリソースから Velocity テンプレートをロードします。

```
from("direct:start").to("velocity://template/simple.vm");
```

これ (**template** ディレクトリーの **.vm** ファイルに保存されている可能性のある他のテンプレート) をネイティブイメージに含めるには、次のようなものを **application.properties** ファイルに追加する必要があります。

```
quarkus.native.resources.includes = template/*.vm
```

3.104. VERT.X HTTP クライアント

Vert.x による Camel HTTP クライアントのサポート

3.104.1. 含まれるもの

- Vert.x HTTP Client コンポーネント, URI 構文： **vertx-http:httpUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.104.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-vertx-http</artifactId>
```

```
</dependency>
```

3.104.3. ネイティブモードの transferException オプション

ネイティブモードで **transferException** オプションを使用するには、オブジェクトのシリアル化のサポートを有効にする必要があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

また、シリアル化する予定の例外クラスのシリアル化を有効にする必要があります。例:

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

3.104.4. 追加の Camel Quarkus 設定

3.104.5. ネイティブモードの allowJavaSerializedObject オプション

ネイティブモードで **allowJavaSerializedObject** オプションを使用する場合、シリアル化のサポートを有効にする必要がある場合があります。詳細は、ネイティブモードのユーザーガイド を参照してください。

3.104.5.1. 文字エンコーディング

アプリケーションがデフォルト以外のエンコーディングを使用してリクエストを送受信することが予想される場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.105. VERT.X WEBSOCKET

このエクステンションを使用すると、WebSocket サーバーとして、または既存の WebSocket に接続するクライアントとして機能する WebSocket エンドポイントを作成できます。

これは、**quarkus-vertx-http** エクステンションによって提供される Eclipse Vert.x HTTP サーバー上にビルドされます。

3.105.1. 含まれるもの

- Vert.x WebSocket コンポーネント, URI 構文： **vertx-websocket:host:port/path**

使用方法と設定の詳細は、上記リンクを参照してください。

3.105.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-vertx-websocket</artifactId>
</dependency>
```

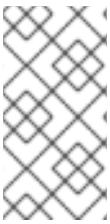
3.105.3. 使用方法

3.105.3.1. Vert.x WebSocket コンシューマー

Vert.x WebSocket コンシューマーを (**from ("vertx-websocket")** などを使用して) 作成する場合 、WebSocket は常に Quarkus HTTP サーバー上でホストされるため、URI 内のホストとポートの設定が冗長になります。

コンシューマーの設定は、簡素化して WebSocket のリソースパスだけを含めることができます。例:

```
from("vertx-websocket:/my-websocket-path")
    .setBody().constant("Hello World");
```



注記

vertx-websocket コンシューマーでホスト/ポートを明示的に設定する必要はありません。設定する場合、ホストとポートが、**quarkus.http.host** および **quarkus.http.port** の Quarkus HTTP サーバー設定値の値と正確に一致する必要があります。一致しないと、実行時に例外が出力されます。

3.105.3.2. Vert.x WebSocket プロデューサー

上記と同様に、内部 Vert.x WebSocket コンシューマーへのメッセージを生成する場合は、エンドポイント URI からホストとポートを省略できます。

```
from("vertx-websocket:/my-websocket-path")
    .log("Got body: ${body}");

from("direct:sendToWebSocket")
    .log("vertx-websocket:/my-websocket-path");
```

あるいは、Quarkus HTTP サーバーのホストとポートの完全な設定を参照することもできます。

```
from("direct:sendToWebSocket")
    .log("vertx-websocket:{{quarkus.http.host}}:{{quarkus.http.port}}/my-websocket-path");
```

外部 WebSocket サーバーへのメッセージを生成する場合は、常にホスト名とポート (必要な場合) を指定する必要があります。

3.105.4. 追加の Camel Quarkus 設定

3.105.4.1. Vert.x WebSocket サーバーの設定

Vert.x WebSocket サーバーの設定は Quarkus によって管理されます。設定オプションの完全なリストについては、Quarkus HTTP 設定ガイド を参照してください。

Vert.x WebSocket サーバーの SSL を設定するには、SSL によるセキュアな接続に関するガイド に従ってください。**SSLContextParameters** を使用した SSL 用のサーバーの設定は現在サポートされていないことに注意してください。

3.105.4.2. 文字エンコーディング

アプリケーションがデフォルト以外のエンコーディングを使用してリクエストを送受信することが想定される場合は、ネイティブモードガイドの 文字エンコーディング のセクションを確認してください。

3.106. XJ

XSLT を使用して、JSON および XML メッセージを変換します。

3.106.1. 含まれるもの

- XJ コンポーネント, URI 構文： **xj:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.106.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.107. XML IO DSL

XML ルート定義を解析するための XML スタック

3.107.1. 含まれるもの

- XML DSL

使用方法と設定の詳細は、上記リンクを参照してください。

3.107.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-xml-io-dsl</artifactId>
</dependency>
```

3.107.3. 追加の Camel Quarkus 設定

3.107.3.1. XML ファイルのエンコーディング

デフォルトでは、一部の XML ファイルエンコーディングはそのままではネイティブモードで機能しない場合があります。修正方法は、文字エンコーディング のセクションを確認してください。

3.108. XML JAXP

XML JAXP タイプコンバーターおよびパーサー

3.108.1. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-xml-jaxp</artifactId>
</dependency>
```

3.109. XPATH

XML ペイロードに対して XPath 式を評価します。

3.109.1. 含まれるもの

- XPath 言語

使用方法と設定の詳細は、上記リンクを参照してください。

3.109.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.109.3. 追加の Camel Quarkus 設定

このコンポーネントは、クラスパスリソースから xpath 式をロードできます。ネイティブモードでも機能させるには、**quarkus.native.resources.includes** プロパティーを使用して、式ファイルをネイティブ実行可能ファイルに明示的に埋め込む必要があります。

たとえば、以下のルートは、**myxpath.txt** という名前のクラスパスリソースから XPath 表現を読み込みます。

```
from("direct:start").transform().xpath("resource:classpath:myxpath.txt");
```

これら (**.txt** ファイルに保存されている可能性のある他の式) をネイティブイメージに含めるには、**application.properties** ファイルに次のようなものを追加する必要があります。

```
quarkus.native.resources.includes = *.txt
```

3.110. XSLT SAXON

Saxon を使用した XSLT テンプレートを使用して XML ペイロードを変換します。

3.110.1. 含まれるもの

- XSLT Saxon コンポーネント、URI 構文： **xslt-saxon:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.110.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-xslt-saxon</artifactId>
</dependency>
```

3.111. XSLT

XSLT テンプレートを使用して XML ペイロードを変換します。

3.111.1. 含まれるもの

- XSLT コンポーネント、URI 構文： **xslt:resourceUri**

使用方法と設定の詳細は、上記リンクを参照してください。

3.111.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

3.111.3. 追加の Camel Quarkus 設定

XSLT 処理を最適化するには、ビルド時に XSLT テンプレートの場所をエクステンションが認識している必要があります。XSLT ソース URI は、**quarkus.camel.xslt.sources** プロパティーを介して渡す必要があります。複数の URI はコンマで区切ることができます。

```
quarkus.camel.xslt.sources = transform.xsl, classpath:path/to/my/file.xsl
```

スキームのない URI は **classpath:** URI として解釈されます。

classpath: のみ: URI は Quarkus ネイティブモードでサポートされます。 **file:**、**http:**、およびその他の種類の URI は、JVM モードでのみ使用できます。

<xsl:include> および **<xsl:messaging>** XSLT 要素も、現在 JVM モードでのみサポートされています。

aggregate DSL を使用する場合は、次のように **XsltSaxonAggregationStrategy** を使用する必要があります。

```
from("file:src/test/resources?noop=true&sortBy=file:name&antInclude=*.xml")
   .routeId("aggregate").noAutoStartup()
   .aggregate(new XsltSaxonAggregationStrategy("xslt/aggregate.xsl"))
   .constant(true)
   .completionFromBatchConsumer()
   .log("after aggregate body: ${body}")
   .to("mock:transformed");
```

また、JVM モードでのみサポートされています。

3.111.3.1. 設定

TransformerFactory 機能は、次のプロパティーを使用して設定できます。

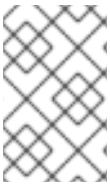
```
quarkus.camel.xslt.features."http\://javax.xml.XMLConstants/feature/secure-processing"=false
```

3.111.3.2. エクステンション機能のサポート

Xalan のエクステンション機能 は、次の場合にのみ正しく機能します。

1. 安全な処理が無効になっています
2. 関数は別の jar で定義されています
3. 関数は、ネイティブビルドフェーズ中に拡張されます。たとえば、リフレクション用に登録できます。

```
@RegisterForReflection(targets = { my.Functions.class })
public class FunctionsConfiguration {
}
```



注記

XSLT ソース URI のコンテンツは、ビルド時に解析され、Java クラスにコンパイルされます。これらの Java クラスは、実行時の XSLT 情報の唯一のソースです。XSLT ソースファイルは、アプリケーションアーカイブにまったく含まれていない場合があります。

設定プロパティー	タイプ	デフォルト
quarkus.camel.xslt.sources コンパイルするテンプレートのコンマ区切りリスト。	string	
quarkus.camel.xslt.package-name 生成されたクラスのパッケージ名。	string	org.apache.camel.quarkus.component.xslt.generated
quarkus.camel.xslt.features TransformerFactory の機能。	Map<String, Boolean>	

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

3.112. YAML DSL

YAML ルート定義を解析するための YAML スタック

3.112.1. 含まれるもの

- YAML DSL

使用方法と設定の詳細は、上記リンクを参照してください。

3.112.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-yaml-dsl</artifactId>
</dependency>
```

3.112.3. 使用方法

3.112.3.1. ネイティブモード

以下の構造を Camel YAML DSL マークアップ内で定義する場合は、リフレクション用のクラスを登録する必要があります。詳細は、ネイティブモード のガイドを参照してください。

3.112.3.1.1. Bean 定義

YAML DSL は、次のように Bean を定義する機能を提供します。

```
- beans:
    - name: "greetingBean"
      type: "org.acme.GreetingBean"
      properties:
        greeting: "Hello World!"
- route:
    id: "my-yaml-route"
    from:
      uri: "timer:from-yaml?period=1000"
      steps:
        - to: "bean:greetingBean"
```

この例では、**GreetingBean** クラスをリフレクション用に登録する必要があります。これは、YAML ルートの **beans** キーで参照するすべての型に適用されます。

```
@RegisterForReflection
public class GreetingBean {
}
```

3.112.3.1.2. 例外処理

Camel は、例外を処理するさまざまな方法を提供します。これらの一部では、DSL 定義で参照されている例外クラスがリフレクション用に登録されていることが必要です。

on-exception

```
- on-exception:
    handled:
      constant: "true"
    exception:
      - "org.acme.MyHandledException"
    steps:
      - transform:
          constant: "Sorry something went wrong"
```

```
@RegisterForReflection
public class MyHandledException {
}
```

throw-exception

```
- route:
    id: "my-yaml-route"
    from:
      uri: "direct:start"
      steps:
        - choice:
            when:
              - simple: "${body} == 'bad value'"
                steps:
                  - throw-exception:
                      exception-type: "org.acme.ForcedException"
                      message: "Forced exception"
            otherwise:
              steps:
                - to: "log:end"
```

```
@RegisterForReflection
public class ForcedException {
}
```

do-catch

```
- route:
    id: "my-yaml-route2"
    from:
      uri: "direct:tryCatch"
      steps:
        - do-try:
            steps:
              - to: "direct:readFile"
            do-catch:
              - exception:
                  - "java.io.FileNotFoundException"
                steps:
                  - transform:
                      constant: "do-catch caught an exception"
```

```
@RegisterForReflection(targets = FileNotFoundException.class)
public class MyClass {
}
```

3.113. ZIP デフレート圧縮

java.util.zip.Deflater、java.util.zip.Inflater、または java.util.zip.GZIPStream を使用して、ストリームを圧縮および解凍します。

3.113.1. 含まれるもの

- GZip デフレーターのデータ形式
- Zip デフレーターのデータ形式

使用方法と設定の詳細については、上記リンクを参照してください。

3.113.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-zip-deflater</artifactId>
</dependency>
```

3.114. ZIP ファイル

java.util.zip.ZipStream を使用して、ストリームを圧縮および圧縮解除します。

3.114.1. 含まれるもの

- Zip File データ形式

使用方法と設定の詳細は、上記リンクを参照してください。

3.114.2. Maven コーディネート

code.quarkus.redhat.com でこのエクステンションを使用して新しいプロジェクトの作成

または、既存のプロジェクトに座標を追加します。

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

第4章 QUARKUS CXF ユーザーガイド

この章では、Quarkus CXF の使用方法と設定について説明します。

4.1. ユーザーガイド

この ユーザーガイド では、Quarkus CXF の典型的なユースケースを説明します。

4.2. 新しいプロジェクトを作成する

このガイドでは、CXF クライアントまたはサーバーをホストする Quarkus アプリケーションの新しいプロジェクト、またはその両方を設定する方法について説明します。

4.2.1. 前提条件

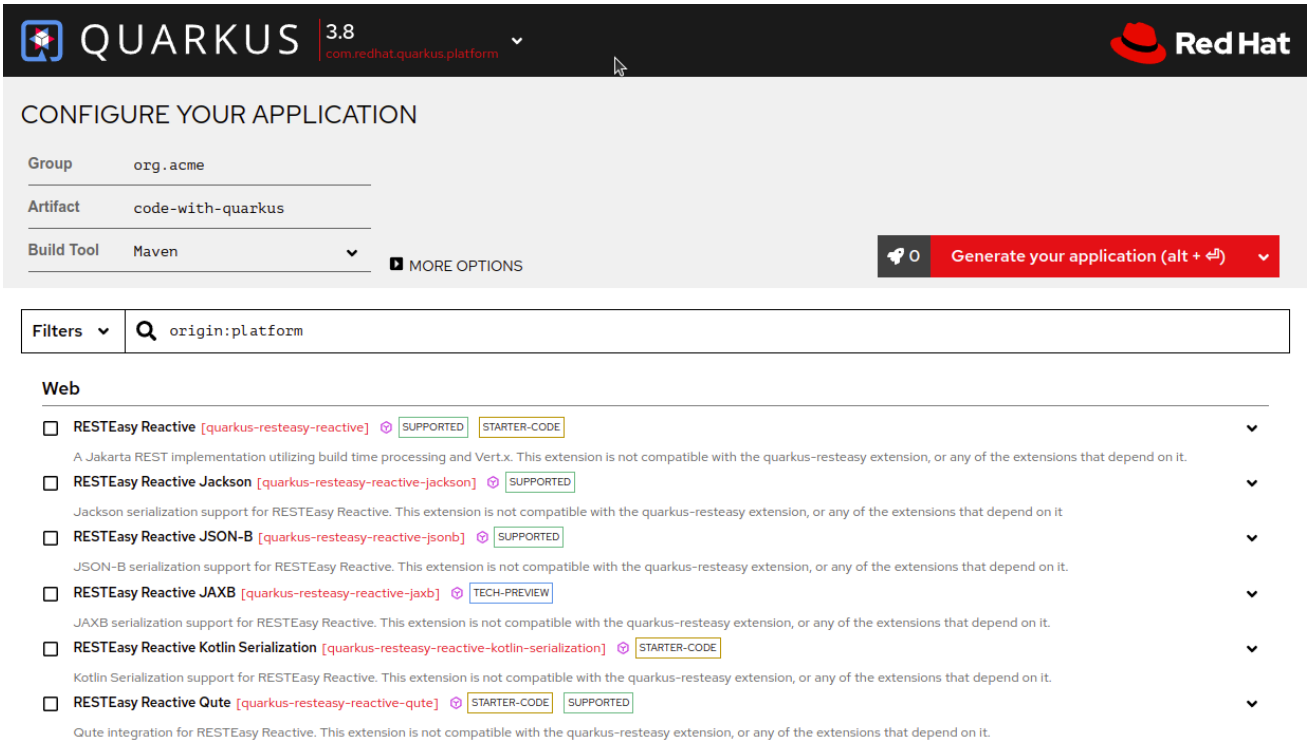
Quarkus スタートガイド の プロジェクトの前提条件 セクション を参照してください。

それに加えて、必要なもの

- **native-image** コマンドをインストールし、**GRAALVM_HOME** 環境変数が設定された GraalVM。Quarkus ドキュメントの Building a native executable セクションを参照してください。
- Linux を使用している場合、ネイティブモードでは **docker** などのコンテナーランタイムも十分です。このオプションを選択した場合は、**-Pnative** の代わりに **-Pnative -Dquarkus.native.container-build=true** を使用します。

4.2.2. プロジェクトの作成

新規プロジェクトスケルトンは、code.quarkus.redhat.com を使用して生成できます。

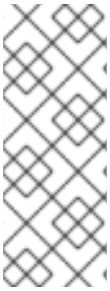


- ここで、作業するエクステンションを選択できます。

- 単純な Hello world Web サービスまたはクライアントの場合は、**quarkus-cxf** エクステンションで十分です。
- 青い **Generate your application** ボタンをクリックして、基本的なスケルトンプロジェクトをダウンロードします。
- zip ファイルを展開し、お気に入りの IDE にプロジェクトをインポートします。

4.2.3. 依存関係の管理

Quarkus CXF は、Quarkus Platform バージョン 3.1.0.Final 以降の Quarkus Platform の一部です。特に、これは code.quarkus.redhat.com およびその他の Quarkus 開発ツール が、適切な依存関係管理でプロジェクトを生成することを意味します。

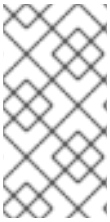


注記

quarkus-cxf プロジェクトの場合は、**com.redhat.quarkus.platform:quarkus-camel-bom** または **com.redhat.quarkus.platform:quarkus-cxf-bom** のいずれかを使用できます(**quarkus-camel-bom** は **quarkus-cxf-bom**のスーパーセットであるため)。

camel-quarkus-cxf-soap プロジェクトの場合は、**quarkus-camel-bom** を使用できます。

```
<project ...>
  ...
  <properties>
    ...
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.version><!-- Check the latest
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-cxf-bom/ --
></quarkus.platform.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>quarkus-cxf-bom</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
...
```



注記

常に同じバージョンの **com.redhat.quarkus.platform:quarkus-bom** および **com.redhat.quarkus.platform:quarkus-cxf-bom** をプロジェクトに追加する必要があります。互換性のあるバージョンの Quarkus、CXF、Quarkus CXF、およびそれらのすべての推移的な依存関係を取得することが最も信頼性の高い方法です。

4.2.4. 次の場所

次の章のいずれかに進むことが推奨されます。

- 最初の SOAP Web サービス
- 最初の SOAP クライアント

4.3. QUARKUS の最初の SOAP WEB サービス

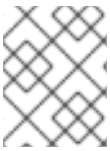
このガイドでは、簡単な SOAP Web サービスを公開する Quarkus アプリケーションを作成する方法を説明します。

最初にプロジェクトを作成します。

「新しいプロジェクトを作成する」 ガイドに従ってから、次に進んでください。

4.3.1. hello world!Web サービス

pom.xml を配置すると、簡単な Hello world! を追加できます。 **src/main/java** の Web サービス



コード例

このセクションで使用されるサンプルコードスニペットは、Quarkus CXF のソースツリーの サーバー統合テスト から来ています。

まず、サービスインターフェイスを追加します。

HelloService.java

```
package io.quarkiverse.cxf.it.server;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

/**
 * The simplest Hello service.
 */
@WebService(name = "HelloService", serviceName = "HelloService")
public interface HelloService {

    @WebMethod
    String hello(String text);

}
```

そして実装は次のとおりです。

HelloServiceImpl.java

```
package io.quarkiverse.cxf.it.server;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

/**
 * The simplest Hello service implementation.
 */
@WebService(serviceName = "HelloService")
public class HelloServiceImpl implements HelloService {

    @WebMethod
    @Override
    public String hello(String text) {
        return "Hello " + text + "!";
    }

}
```

実装を特定のパスで公開するには、以下の設定を **application.properties** に追加する必要があります。

```
# The context path under which all services will be available
quarkus.cxf.path = /soap

# Publish "HelloService" under the context path /${quarkus.cxf.path}/hello
quarkus.cxf.endpoint."/hello".implementor = io.quarkiverse.cxf.it.server.HelloServiceImpl
quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature
```

ヒント

すべての設定プロパティーは、設定 プロパティーリファレンス に記載されています。

これらのファイルが導入されると、Quarkus を開発 モードで**起動** できます。

```
$ mvn quarkus:dev
```

これにより、プロジェクトがコンパイルされ、バックグラウンドでアプリケーションが起動します。

curl またはその他の SOAP クライアントを使用してサービスをテストできます。

最初に、http://localhost:8080/soap/hello?wsdl の自動生成 WSDL を見てみましょう。

```
$ curl http://localhost:8080/soap/hello?wsdl
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://server.it.cxf.quarkiverse.io/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
    name="HelloService" targetNamespace="http://server.it.cxf.quarkiverse.io/">
  <wsdl:types>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://server.it.cxf.quarkiverse.io/" attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://server.it.cxf.quarkiverse.io/">
  <xsd:element name="hello" type="tns:hello"/>
  <xsd:complexType name="hello">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="arg0" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="helloResponse" type="tns:helloResponse"/>
  <xsd:complexType name="helloResponse">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="return" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
  </wsdl:types>
  <wsdl:message name="helloResponse">
    <wsdl:part element="tns:helloResponse" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="hello">
    <wsdl:part element="tns:hello" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="HelloService">
    <wsdl:operation name="hello">
      <wsdl:input message="tns:hello" name="hello">
    </wsdl:input>
      <wsdl:output message="tns:helloResponse" name="helloResponse">
    </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloServiceSoapBinding" type="tns:HelloService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="hello">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="hello">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="helloResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloService">
    <wsdl:port binding="tns:HelloServiceSoapBinding" name="HelloServicePort">
      <soap:address location="http://localhost:8080/soap/hello"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

次に、SOAP リクエストをサービスに送信してみましょう。

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
```

```
    '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body><ns2:hello xmlns:ns2="http://server.it.cxf.quarkiverse.io/"><arg0>World</arg0>
</ns2:hello></soap:Body>
     </soap:Envelope>' \
   http://localhost:8080/soap/hello
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:helloResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <return>Hello World!</return>
    </ns1:helloResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP 応答で予期される < **return>Hello World!</return&** gt; を確認できます。

4.3.2. dev モードの実行中にロギング機能を追加します。

サーバーまたはクライアントによって送受信された SOAP メッセージを検査できるのが便利な場合があります。これは、**quarkus-cxf-rt-features-logging** エクステンションを **pom.xml** に追加することで簡単に実行できます。

ヒント

これは、Quarkus dev モードの実行中に行います。ソースツリーに変更を保存すると、アプリケーションが再コンパイルされ、再デプロイされます。

これを pom.xmlに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-features-logging</artifactId>
</dependency>
```

application.propertiesで SOAP ペイロードのロギングを有効にする

```
quarkus.cxf.endpoint."/hello".features=org.apache.cxf.ext.logging.LoggingFeature
```

その後、新しい SOAP リクエストを送信し、アプリケーションコンソールで SOAP ペイロードを確認できます。

```
2023-01-11 22:12:21,315 INFO  [org.apa.cxf.ser.Hel.REQ_IN] (vert.x-worker-thread-0) REQ_IN
    Address: http://localhost:8080/soap/hello
    HttpMethod: POST
    Content-Type: text/xml;charset=UTF-8
    ExchangeId: af10747a-8477-4c17-bf5f-2a4a3a95d61c
    ServiceName: HelloService
    PortName: HelloServicePort
    PortTypeName: HelloService
    Headers: {Accept=*/*, User-Agent=curl/7.79.1, content-type=text/xml;charset=UTF-8,
Host=localhost:8080, Content-Length=203, x-quarkus-hot-deployment-done=true}
    Payload: <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body><ns2:hello xmlns:ns2="http://server.it.cxf.quarkiverse.io/"><arg0>World</arg0>
</ns2:hello></soap:Body>
```

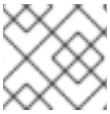
```
</soap:Envelope>


2023-01-11 22:12:21,327 INFO  [org.apa.cxf.ser.Hel.RESP_OUT] (vert.x-worker-thread-0)
RESP_OUT
    Address: http://localhost:8080/soap/hello
    Content-Type: text/xml
    ResponseCode: 200
    ExchangeId: af10747a-8477-4c17-bf5f-2a4a3a95d61c
    ServiceName: HelloService
    PortName: HelloServicePort
    PortTypeName: HelloService
    Headers: {}
    Payload: <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns1:helloResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/"><return>Hello World!</return>
</ns1:helloResponse></soap:Body></soap:Envelope>
```

4.3.3. その他の手順

JVM で実行するか、ネイティブに実行するためにアプリケーションのパッケージ化を続行したい場合があります。

4.4. QUARKUS での最初の SOAP クライアントの作成

このガイドでは、リモート Web サービスのクライアントとして機能する簡単な Quarkus アプリケーションを作成する方法を説明します。



最初にプロジェクトを作成します。

「新しいプロジェクトを作成する」 ガイドに従ってから、次に進んでください。

4.4.1. テスト用のリモート Web サービス

まず、接続するにはいくつかのリモート Web サービスが必要です。この目的のためにコンテナーで実行される単純な Calculator Web サービス を使用できます。

```
$ docker run -p 8082:8080 quay.io/l2x6/calculator-ws:1.0
```

コンテナーを起動して実行したら、その WSDLを検査できます。

```
$ curl -s http://localhost:8082/calculator-ws/CalculatorService?wsdl
<?xml version="1.0" ?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="CalculatorService"
targetNamespace="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">

  ...

  <wsdl:binding name="CalculatorServiceSoapBinding" type="tns:CalculatorService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http">
```

```
</soap:binding>
    <wsdl:operation name="add">
      <soap:operation soapAction="" style="document"></soap:operation>
      <wsdl:input name="add">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="addResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="subtract">
      <soap:operation soapAction="" style="document"></soap:operation>
      <wsdl:input name="subtract">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="subtractResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
    </wsdl:operation>

    ...

  </wsdl:binding>

  ...

</wsdl:definitions>
```

WSDL でわかるように、サービスは **追加**、**減** 算などの基本的な算術演算を提供します。

curl でテストします。

```
$ curl -s \
    -X POST \
    -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
       '<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
          <Body>
             <add xmlns="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">
                <arg0 xmlns="">7</arg0> (1)
                <arg1 xmlns="">4</arg1>
             </add>
          </Body>
       </Envelope>' \
    http://localhost:8082/calculator-ws/CalculatorService
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:addResponse xmlns:ns2="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">
      <return>11</return> (2)
    </ns2:addResponse>
  </soap:Body>
</soap:Envelope>
```

(1) **7** および **4**を追加する要求

2 11 - 操作の戻り値

4.4.2. SOAP クライアント

次に、Quarkus アプリケーション内でクライアントを取得する方法を見てみましょう。

まず、サービスエンドポイントインターフェイス(SEI)と、それが必要とする他のすべてのモデルクラスが必要です。

それらを取得するには、いくつかの方法があります。

- 手動による書き込み
- Java で記述されている場合は、Web Sevice プロジェクトからコピーします。
- Service プロジェクトによって提供されるなど、モデルクラスを含む Maven アーティファクトがある
- WSDL からモデルクラスを生成します

最後のオプションは、クライアントアプリケーションにとって最も簡単で柔軟性が高い傾向にあります。

ヒント

このアプローチを使用する場合は、まず WSDL からの Java の生成 に従ってから、次の手順に進みます。

4.4.3. SEI のクライアントとしての使用

この場合、Service Endpoint Interface (SEI)は **org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService** です。

通常どおり、Quarkus では CDI を介してインスタンスを取得できます。

テストを簡単にするために、REST サービスにラップします。

CxfClientResource.java

```
package io.quarkiverse.cxf.client.it;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;

import org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService;

import io.quarkiverse.cxf.annotation.CXFClient;

@Path("/cxf/calculator-client")
public class CxfClientRestResource {
```

```
    @CXFClient("myCalculator") 1
    CalculatorService myCalculator;

    @GET
    @Path("/add")
    @Produces(MediaType.TEXT_PLAIN)
    public int add(@QueryParam("a") int a, @QueryParam("b") int b) {
        return myCalculator.add(a, b); 2
    }

}
```

1 CDI コンテナーがクライアントのインスタンスを挿入させます。**@CXFClient ("myCalculator")** は **@Inject @CXFClient ("myCalculator")**と同等です。

2 **追加** の操作を呼び出し、リモート Web サービスを呼び出します。

クライアントはこれだけですか ?: 上記に加えて、**application.properties** の CXF Quarkus エクステンションに他のものを伝える必要があります。

application.properties

```
cxf.it.calculator.baseUri=http://localhost:8082
quarkus.cxf.client.myCalculator.wsdl = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService?wsdl
quarkus.cxf.client.myCalculator.client-endpoint-url = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService
quarkus.cxf.client.myCalculator.service-interface = org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService
```

ヒント

すべてのクライアント設定プロパティーは、設定 プロパティーリファレンス に記載されています。

上記のファイルをすべて配置すると、Quarkus **dev** モードでアプリケーションを起動できるようになります。

```
$ mvn quarkus:dev
...
INFO  [io.quarkus] (Quarkus Main Thread) ... Listening on: http://localhost:8080
```

そして、それにいくつかのリクエストを送信することによりテストします：

```
$ curl -s 'http://localhost:8080/cxf/calculator-client/add?a=5&b=6'
11
```

ここで、**11** は、**5** と **6** を追加したときに正しい結果です。

4.4.4. その他の手順

続行したいかもしれません

- JVM およびネイティブ用のパッケージ。
- 高度な SOAP クライアントトピック

4.5. プロジェクト設定オプション

Quarkus CXF は、多数の設定オプションを公開します。

設定オプションは、**application.properties** ファイルまたは環境変数を使用して設定できます。詳細は、Quarkus configuration reference を参照してください。

4.5.1. Bean 参照

Quarkus CXF の複数の設定オプションでは、Quarkus CDI コンテナーに存在する Bean を参照できます。機能およびインターセプター はこれらの典型的な例です。

設定で Bean 参照を設定する方法は、型または Bean 名の 2 つがあります。

4.5.1.1. 型別の Bean 参照

以下に例を示します。

application.properties

```
# bean reference by type
quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature
```

タイプ名別の参照を使用する場合、解像度は以下のようになります。

- Bean は、タイプ別に Quarkus CDI コンテナーで検索されます。
- Bean が利用可能な場合は、それが使用されます。
- 複数の Bean が指定の型に割り当てることができる場合、例外が発生します。
- 一致する Bean がない場合は、クラスがロードされ、デフォルトのコンストラクターを使用してインスタンス化するために試行されます。

4.5.1.2. Bean 名による Bean 参照

以下に例を示します。

application.properties

```
# bean reference by bean name
quarkus.cxf.endpoint."/fruit".features = #myCustomLoggingFeature
```

Bean 名による参照を使用する場合、意図せずに Bean は名前で Quarkus CDI コンテナーで検索されます。**myCustomLoggingFeature** という名前付き Bean は、以下のように定義できます。

```
import org.apache.cxf.ext.logging.LoggingFeature;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Produces;
```

```
class Producers {

    @Produces
    @ApplicationScoped
    @Named("myCustomLoggingFeature")
    LoggingFeature myCustomLoggingFeature() {
        LoggingFeature loggingFeature = new LoggingFeature();
        loggingFeature.setPrettyLogging(true);
        return loggingFeature;
    }
}
```

4.6. JVM またはネイティブで実行するためのパッケージ

この章では、JVM で実行するため、またはネイティブに実行するために Quarkus CXF アプリケーションをパッケージ化する方法を説明します。

4.6.1. JVM モード

SOAP クライアントおよび SOAP サービスの概要ガイドでは、Quarkus **dev** mode でのみ機能しました。Quarkus ツールはバックグラウンドで実行されており、ワークスペースの変更を監視し、必要に応じてアプリケーションを再コンパイルおよび再読み込みしました。

開発が完了したら、JVM でアプリケーションを実行する方法。

まず、Maven でパッケージ化する必要があります。

```
$ mvn package
```

JVM でアプリケーションを実行するために必要なライブラリーは、**target/quarkus-app** ディレクトリーにあります。

```
$ ls -lh target/quarkus-app
drwxr-xr-x. 2 ppalaga ppalaga 4.4K Jan 12 22:29 app
drwxr-xr-x. 4 ppalaga ppalaga 4.4K Jan 12 22:29 lib
drwxr-xr-x. 2 ppalaga ppalaga 4.4K Jan 12 22:29 quarkus
-rw-r-r--. 1 ppalaga ppalaga 6.1K Jan 12 22:29 quarkus-app-dependencies.txt
-rw-r-r--. 1 ppalaga ppalaga  678 Jan 12 22:29 quarkus-run.jar
```

アプリケーションは次のように起動できます。

```
$ java -jar target/quarkus-app/quarkus-run.jar
```

curl を使用して一部の SOAP リクエストを送信し、アプリケーションが機能することを確認できます。

4.6.2. ネイティブモード

Quarkus は GraalVM ネイティブイメージを構築するための最初のクラスサポートを提供します。また、Quarkus CXF も約束を尊重します。

イメージ

GraalVM ネイティブイメージは、JVM なしで直接実行できるプラットフォーム固有の実行可能ファイルです。JVM モードで同じアプリケーションを実行するよりも、起動時間が速くなり、メモリーの使用量が少なくなります。

code.quarkus.redhat.com によって生成された **pom.xml** ファイルには、**ネイティブ** イメージのビルドに必要なネイティブプロファイルが含まれます。

```
<profile>
  <id>native</id>
  <activation>
    <property>
      <name>native</name>
    </property>
  </activation>
  <properties>
    <skipITs>false</skipITs>
    <quarkus.package.type>native</quarkus.package.type>
  </properties>
</profile>
```

さらに、プロジェクトの作成 セクションで説明されているように、GraalVM **ネイティブイメージ** ツールが必要です。

これをローカルにインストールするか、または **GRAALVM_HOME** 環境を適切に設定するか、または docker (Linux ネイティブ実行可能ファイル)を生成するだけでよい場合は、**docker** を使用できます。

GraalVM のローカルインストールの場合

```
# Make sure $GRAALVM_HOME is set properly
$ echo $GRAALVM_HOME
/home/{user}/.sdkman/candidates/java/{major}.{minor}.r{java-version}-grl

# Produce the native executable
mvn package -Pnative
```

ヒント

Quarkus は、GraalVM のバージョンについて非常に選択できます。ローカルインストールを使用する場合は、Quarkus の推奨バージョンを使用するようにしてください。そのためには、**pom.xml** にインポートした **quarkus-bom** を開き、そこで **graalvm** を検索します。Docker を使用する場合、Quarkus は適切なバージョンをプルするために注意します。

dockerあり

```
# Produce the native executable
mvn package -Pnative -Dquarkus.native.container-build=true
```

これには、簡単なアプリケーションには 1 分以上かかる場合があります。

ビルドが完了したら、ネイティブ実行可能ファイルが **ターゲット** ディレクトリーで利用できるようにする必要があります。

```
$ ls -l target
...
-rwxr-xr-x. 1 ppalaga ppalaga  71M Jan 11 22:42 quarkus-cxf-integration-test-server-1.8.0-
SNAPSHOT-runner
...
```

ご覧のとおり、サイズが 71 MB のみで、実行可能です。

これは、以下のように実行できます。

```
$ target/*-runner
...
INFO  [io.quarkus] (main) quarkus-cxf-integration-test-server 1.8.0-SNAPSHOT native (powered by
Quarkus
2.15.2.Final) started in 0.042s. Listening on: http://0.0.0.0:8080
...
```

ここでも、**curl** を使用して一部の SOAP リクエストを送信し、ネイティブ実行可能ファイルが機能することを確認できます。

メモリー使用量、最初のリクエスト時間、およびその他のパフォーマンスメトリクスを以前使用したスタックと比較し、結果を共有することを忘れないでください。

4.6.3. ネイティブイメージ：関連情報

以下のリンクを参照することもできます。これには、ネイティブイメージの操作方法のヒントが含まれています。

- Quarkus: ネイティブ実行可能ファイルのビルド
- Quarkus: ネイティブアプリケーション作成のヒント
- Quarkus: ネイティブリファレンスガイド
- GraalVM: ネイティブイメージでのリソースへのアクセス
- GraalVM: ネイティブイメージでの使用を検討
- GraalVM: Tracing Agent

4.6.4. コンテナーイメージの作成

Quarkus Container image guide を参照してください。

4.7. LOGGING

Quarkus のロギングに関する基本情報は、Quarkus Logging ガイド を参照してください。

- アプリケーションコードでの ロガーの取得
- ログレベル
- Categories
- 形式

- JSON 形式

4.7.1. ペイロードロギング

履歴

Quarkus CXF 2.6.0 以降、ペイロードのロギング機能は **io.quarkiverse.cxf:quarkus-cxf** エクステンション で利用できます。2.6.0 より前のバージョンでは、別の拡張 **io.quarkiverse.cxf:quarkus-cxf-rt-features-logging** を通じて利用できました。これは現在非推奨となり、今後削除される予定です。

ペイロードのロギング機能は、主に **org.apache.cxf.ext.logging.LoggingFeature** クラスを使用して実装されます。

クライアントまたはサービスエンドポイントで機能を設定する方法はいくつかあります。

4.7.2. 設定プロパティーによるペイロードログの設定

4.7.2.1. グローバル設定

グローバルロギングオプションは、Quarkus CXF 2.6.0 以降に存在します。**quarkus.cxf.logging.enabled = true** で有効にすると、Quarkus CXF はグローバルの **LoggingFeature** インスタンスを作成し、アプリケーションのすべてのクライアントおよびサービスエンドポイントにこれを設定します。グローバル設定は、クライアントまたはサービスエンドポイントレベルで 上書き できます。

application.properties

```
# Global settings
quarkus.cxf.logging.enabled = true
quarkus.cxf.logging.pretty = true
```

すべてのロギング設定オプションは、**quarkus-cxf** リファレンスページ にリストされています。

ヒント

このページに記載されているすべてのロギングプロパティーは **ランタイム** 設定オプションです。したがって、アプリケーションを起動しなくても、アプリケーションの起動時に渡すことができます。これには、システムプロパティーをコマンドライン（例： **-Dquarkus.cxf.logging.enabled=true**）に渡すか、環境変数を設定します（例 **: QUARKUS_CXF_LOGGING_ENABLED=true をエクスポート**する）。

4.7.2.2. クライアントごとおよびサービスごとのエンドポイント設定

Quarkus CXF 2.5.0 以降、**application.properties** で適切なオプションを設定することにより、**LoggingFeature** を設定してクライアントまたはサービスエンドポイントに宣言的に割り当てることができます。

application.properties

```
# For a service:
quarkus.cxf.endpoint."/hello".logging.enabled = true
```

```
quarkus.cxf.endpoint."/hello".logging.pretty = true
# For a client:
quarkus.cxf.client.hello.logging.enabled = true
quarkus.cxf.client.hello.logging.pretty = true
```

すべてのロギング設定オプションは、**quarkus-cxf** リファレンスページに記載されています。

- クライアントの場合
- サービスエンドポイントの場合

4.7.3. LoggingFeature をクライアントまたはサービスに追加する代替方法

デフォルト設定でインスタンスを接続するには、以下のいずれかを行います。

1. **application.properties** で以下を行います。

```
# For a service:
quarkus.cxf.endpoint."/hello".features=org.apache.cxf.ext.logging.LoggingFeature
# For a client:
quarkus.cxf.client."myClient".features=org.apache.cxf.ext.logging.LoggingFeature
```

ヒント

ユーザーガイドの Your first SOAP Web サービス の章に例を示します。

または、

2. CXF の **@Features** アノテーションを使用します。

```
@org.apache.cxf.feature.Features (features = {"org.apache.cxf.ext.logging.LoggingFeature"})
@WebService(endpointInterface = "org.acme.SayHi", targetNamespace = "uri:org.acme")
public class SayHiImplementation implements SayHi {
  public long sayHi(long arg) {
     return arg;
  }
  //...
}
```

4.7.3.1. カスタム LoggingFeature Bean の生成

LoggingFeature のセットアップにカスタムロジックが必要な場合は、名前付きの **LoggingFeature** Bean を生成できます。

```
import org.apache.cxf.ext.logging.LoggingFeature;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;

class Producers {

  @Produces
  @ApplicationScoped
  @Named("limitedLoggingFeature") // "limitedLoggingFeature" is redundant if the name of the
```

```
method is the same
    LoggingFeature limitedLoggingFeature() {
        LoggingFeature loggingFeature = new LoggingFeature();
        loggingFeature.setPrettyLogging(true);
        loggingFeature.setLimit(1024);
        return loggingFeature;
    }
}
```

次に、**application.properties** で接頭辞 **#** が付いた名前で参照できます。

```
# For a service:
quarkus.cxf.endpoint."/hello".features = #limitedLoggingFeature
# For a client:
quarkus.cxf.client.hello.features = #limitedLoggingFeature
```

4.8. JAXB を使用した複雑な SOAP ペイロード

Quarkus SOAP クライアントの概要ガイドと、最初の SOAP Web サービスは、プリミティブパラメーターのみを持つサービスと、整数や文字列などの戻り値だけを持つサービスを処理します。より複雑なオブジェクトの受け渡しと受信について見てみましょう。

たとえば、果物を管理するためのアプリケーションを作成します。

注記

このセクションで使用されるサンプルコードスニペットは、Quarkus CXF のソースツリーの サーバー統合テスト から来ています。

fruit の表現は複雑であると仮定されているため、いくつかの属性を持つ Java Bean としてモデル化します。

```
package io.quarkiverse.cxf.it.server;

import java.util.Objects;

import jakarta.xml.bind.annotation.XmlElement;
import jakarta.xml.bind.annotation.XmlRootElement;
import jakarta.xml.bind.annotation.XmlType;

@XmlType(name = "Fruit")
@XmlRootElement
public class Fruit {

    private String name;

    private String description;

    public Fruit() {
    }

    public Fruit(String name, String description) {
        this.name = name;
        this.description = description;
```

```
    }

    public String getName() {
        return name;
    }

    @XmlElement
    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    @XmlElement
    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Fruit)) {
            return false;
        }

        Fruit other = (Fruit) obj;

        return Objects.equals(other.getName(), this.getName());
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.getName());
    }
}
```

ご覧のとおり、**@XmlElement**、**@XmlRootElement** および **@XmlType** などの一部の JAXB アノテーションが使用されました。これは、Bean のシリアル化と XML との間のシリアル化と逆シリアル化を制御するためです。

4.8.1. リフレクションの自動登録

JAXB はリフレクションベースのシリアル化フレームワークです。GraalVM ネイティブイメージについて学習する場合、通常、ビルド時に反映するにはクラス、フィールド、およびメソッドを登録する必要があります。plain GraalVM では、**reflection-config.json** を介して手動でこれを行う必要があります。少なくとも、自分で書き込んだクラスのために。Quarkus ではそうではありません。**quarkus-jaxb** エクステンション(**quarkus-cxf** が依存する)は、アプリケーションのクラスパスで JAXB アノテーションが付けられたクラスのクラスパスをスキャンし、反映のために自動的に登録できます。

したがって、Quarkus で複雑なペイロードを操作することは、ストック CXF とは変わりません。JAXB シリアル化と逆シリアル化は、追加設定なしで使用できます。

4.8.2. SEI と実装

果物を管理するためのサービスエンドポイントインターフェイス(SEI)は以下のようになります。

```
package io.quarkiverse.cxf.it.server;

import java.util.Set;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService
public interface FruitService {

    @WebMethod
    Set<Fruit> list();

    @WebMethod
    Set<Fruit> add(Fruit fruit);

    @WebMethod
    Set<Fruit> delete(Fruit fruit);
}
```

SEI は、できるだけ簡単に実装できます。

```
package io.quarkiverse.cxf.it.server;

import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.Set;

import jakarta.jws.WebService;

@WebService(serviceName = "FruitService")
public class FruitServiceImpl implements FruitService {

    private Set<Fruit> fruits = Collections.synchronizedSet(new LinkedHashSet<>());

    public FruitServiceImpl() {
        fruits.add(new Fruit("Apple", "Winter fruit"));
        fruits.add(new Fruit("Pineapple", "Tropical fruit"));
    }

    @Override
    public Set<Fruit> list() {
        return fruits;
    }

    @Override
    public Set<Fruit> add(Fruit fruit) {
        fruits.add(fruit);
        return fruits;
    }

    @Override
    public Set<Fruit> delete(Fruit fruit) {
```

```
        fruits.remove(fruit);
        return fruits;
    }
}
```

4.8.3. application.properties

この実装は非常にシンプルで、**application.properties** を使用してエンドポイントを定義する必要があります。

```
quarkus.cxf.endpoint."/fruits".implementor = io.quarkiverse.cxf.it.server.FruitServiceImpl
quarkus.cxf.endpoint."/fruits".features = org.apache.cxf.ext.logging.LoggingFeature
```

4.8.4. Quarkus 開発モードおよび curlでのテスト

上記のファイルを配置すると、Quarkus ツールを dev モード で起動できます。

```
$ mvn quarkus:dev
...
INFO  [io.quarkus] (Quarkus Main Thread) ... Listening on: http://localhost:8080
```

次に、その リスト 操作を呼び出して、サービスが機能しているかどうかを確認します。

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
   -d \
    '<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <soapenv:Body>
         <ns1:list/>
      </soapenv:Body>
   </soapenv:Envelope>' \
   http://localhost:8080/soap/fruits
...
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:listResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Winter fruit</description>
        <name>Apple</name>
      </return>
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Tropical fruit</description>
        <name>Pineapple</name>
      </return>
    </ns1:listResponse>
  </soap:Body>
</soap:Envelope>
```

ご覧のとおり、エンドポイントはデフォルトで 2 つの fruits Apple と Pineapple を返します。

次に、Orange があると、別の fruit を追加してみましょう。

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
   -d \
    '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
       <ns2:add xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <arg0>
         <description>Mediterranean fruit</description>
         <name>Orange</name>
        </arg0>
       </ns2:add>
     </soap:Body></soap:Envelope>' \
   http://localhost:8080/soap/fruits
...
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <ns1:addResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
   <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
    <description>Winter fruit</description>
    <name>Apple</name>
   </return>
   <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
    <description>Tropical fruit</description>
    <name>Pineapple</name>
   </return>
   <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
    <description>Mediterranean fruit</description>
    <name>Orange</name>
   </return>
  </ns1:addResponse>
 </soap:Body>
</soap:Envelope>
```

返されたリストに Orange が期待どおりに追加されたことがわかります。

4.8.5. その他の手順

JVM で実行するか、ネイティブに実行するためにアプリケーションのパッケージ化を続行したい場合があります。

4.9. SSL

本章では、SSL、TLS、および HTTPS に関するさまざまなユースケースを説明します。

注記

このセクションで使用されるサンプルコードスニペットは、Quarkus CXF のソースツリーの WS-SecurityPolicy 統合テスト のものです。

4.9.1. クライアント SSL 設定

クライアントがクライアントのオペレーティングシステムによって SSL 証明書を信頼しないサーバーと通信する場合は、クライアントにカスタムトラストストアを設定する必要があります。openssl や Java keytool などのツールは、これを行うために一般的に使用されます。Quarkus CXF ソースツリーで いくつかの例 を確認することを推奨します。

トラストストアの準備が完了したら、それを使用するようにクライアントを設定する必要があります。

4.9.1.1. application.propertiesでクライアントトラストストアを設定する

これは、クライアントトラストストアを設定する最も簡単な方法です。キーロールは以下のプロパティーによって実行されます。

- quarkus.cxf.client."clients".trust-store
- quarkus.cxf.client."clients".trust-store-type
- quarkus.cxf.client."clients".trust-store-password

以下に例を示します。

application.properties

```
keystore.type = jks (1)
quarkus.cxf.client.hello.client-endpoint-url = https://localhost:${quarkus.http.test-ssl-
```

```
port}/services/hello
quarkus.cxf.client.hello.service-interface = io.quarkiverse.cxf.it.security.policy.HelloService
quarkus.cxf.client.hello.trust-store-type = ${keystore.type}
(2)
quarkus.cxf.client.hello.trust-store = client-truststore.${keystore.type}
quarkus.cxf.client.hello.trust-store-password = password
```

(1) pkcs12 トラストストアタイプは、jks の一般的な代替手段です。

(2) 参照される client-truststore.jks ファイルは src/main/resources ディレクトリーで利用可能である必要があります。

4.9.2. サーバー SSL 設定

サーバー SSL 設定は、Quarkus HTTP レイヤー a.k.a によって実行されます。Vert.x.Quarkus HTTP ガイド は、設定オプションに関する情報を提供します。

以下に基本的な例を示します。

application.properties

```
(1)
quarkus.http.ssl.certificate.key-store-file = localhost.${keystore.type}
quarkus.http.ssl.certificate.key-store-password = password
quarkus.http.ssl.certificate.key-store-key-alias = localhost
quarkus.http.ssl.certificate.key-store-key-password = password
```

(1) 参照される localhost.jks ファイルは src/main/resources ディレクトリーで利用可能である必要があります。

4.9.3. WS-SecurityPolicy による SSL の強制

クライアントが HTTPS 経由で接続する要件は、ポリシーで定義できます。

この機能は、quarkus-cxf-rt-ws-security エクステンションによって提供されます。

ポリシーファイルの例を以下に示します。

https-policy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <wsp:ExactlyOne>
      <wsp:All>
         <sp:TransportBinding>
            <wsp:Policy>
               <sp:TransportToken>
                  <wsp:Policy>
                     <sp:HttpsToken RequireClientCertificate="false" />
                  </wsp:Policy>
               </sp:TransportToken>
               <sp:IncludeTimestamp />
               <sp:AlgorithmSuite>
                  <wsp:Policy>
                     <sp:Basic128 />
                  </wsp:Policy>
               </sp:AlgorithmSuite>
            </wsp:Policy>
         </sp:TransportBinding>
      </wsp:All>
   </wsp:ExactlyOne>
</wsp:Policy>
```

ポリシーは、サービスエンドポイントインターフェイス(SEI)から参照する必要があります。

HttpsPolicyHelloService.java

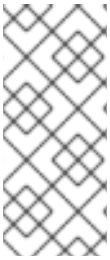
-

```
package io.quarkiverse.cxf.it.security.policy;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

import org.apache.cxf.annotations.Policy;

/**
 * A service implementation with a transport policy set
 */
@WebService(serviceName = "HttpsPolicyHelloService")
@Policy(placement = Policy.Placement.BINDING, uri = "https-policy.xml")
public interface HttpsPolicyHelloService extends AbstractHelloService {

    @WebMethod
    @Override
    public String hello(String text);

}
```

この設定が行われると、HTTP 経由で配信されるリクエストは PolicyVerificationInInterceptor によって拒否されます。

```
ERROR [org.apa.cxf.ws.pol.PolicyVerificationInInterceptor] Inbound policy verification failed:
These policy alternatives can not be satisfied:
 {http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702}TransportBinding: TLS is not
enabled
 ...
```

4.10. 認証および認可



注記

本セクションのサンプルコードスニペットは、Quarkus CXF のソースツリーの クライアントおよびサーバー統合テスト のものです。実行可能な例として使用することもできます。

4.10.1. クライアント HTTP Basic 認証

quarkus-cxf 拡張機能によって提供される以下のクライアント設定オプションを使用して、HTTP Basic 認証のユーザー名とパスワードを渡します。

- quarkus.cxf.client."clients".username
- quarkus.cxf.client."clients".password

以下に例を示します。

application.properties

```
quarkus.cxf.client.basicAuth.wsdl = http://localhost:${quarkus.http.test-port}/soap/basicAuth?wsdl
quarkus.cxf.client.basicAuth.client-endpoint-url = http://localhost:${quarkus.http.test-port}/soap/basicAuth
quarkus.cxf.client.basicAuth.username = bob
quarkus.cxf.client.basicAuth.password = bob234
```

4.10.2. Basic 認証によって保護された WSDL へのアクセス

デフォルトでは、quarkus.cxf.client."clients".secure-wsdl-access を true に設定しない限り、Quarkus CXF によって作成されたクライアントは Authorization ヘッダーを送信しません。

application.properties

```
quarkus.cxf.client.basicAuthSecureWsdl.wsdl = http://localhost:${quarkus.http.test-port}/soap/basicAuth?wsdl
quarkus.cxf.client.basicAuthSecureWsdl.client-endpoint-url = http://localhost:${quarkus.http.test-port}/soap/basicAuthSecureWsdl
quarkus.cxf.client.basicAuthSecureWsdl.username = bob
quarkus.cxf.client.basicAuthSecureWsdl.password = ${client-server.bob.password}
quarkus.cxf.client.basicAuthSecureWsdl.secure-wsdl-access = true
```

4.10.3. サービスエンドポイントのセキュリティー保護

サーバー側の認証および承認は、Quarkus セキュリティー （特に対象である場合）によって実行されます。

- 認証メカニズム
- アイデンティティープロバイダー
- ロールベースアクセス制御 (RBAC)

クライアントとサーバーの統合テスト には、基本的な例があります。主な部分は次のとおりです。

- io.quarkus:quarkus-elytron-security-properties-file 依存関係をアイデンティティープロバイダーとする
- 基本認証が有効になっていること、および application.properties でロールが設定されたユーザー：

application.properties

```
quarkus.http.auth.basic = true
quarkus.security.users.embedded.enabled = true
quarkus.security.users.embedded.plain-text = true
quarkus.security.users.embedded.users.alice = alice123
quarkus.security.users.embedded.roles.alice = admin
quarkus.security.users.embedded.users.bob = bob234
quarkus.security.users.embedded.roles.bob = app-user
```

- ロールベースのアクセス制御は、@RolesAllowed アノテーションを介して行われます。

BasicAuthHelloServiceImpl.java

```
package io.quarkiverse.cxf.it.auth.basic;
```

```
import jakarta.annotation.security.RolesAllowed;
import jakarta.jws.WebService;

import io.quarkiverse.cxf.it.HelloService;

@WebService(serviceName = "HelloService", targetNamespace = HelloService.NS)
@RolesAllowed("app-user")
public class BasicAuthHelloServiceImpl implements HelloService {
    @Override
    public String hello(String person) {
        return "Hello " + person + "!";
    }
}
```

4.11. 高度な SOAP クライアントトピック

4.11.1. client-endpoint-url のデフォルト

application.properties で client-endpoint-url プロパティーを省略した場合、CXF Quarkus エクステンションはサービスが http://localhost:8080/{service-path} で公開されていることを前提としています。ここで、{service-path} は派生しています。

- 設定プロパティー quarkus.cxf.path （指定されている場合）および
- SEI のクラス名（小文字）

quarkus.cxf.path = /ws が指定されると、CalculatorService のデフォルトの有効な client-endpoint-url は http://localhost:8080/ws/org.jboss.eap.quickstarts.wscalculator.calculator.calculatorservice になります。

quarkus.cxf.path が指定されていない場合、client-endpoint-url は http://localhost:8080/org.jboss.eap.quickstarts.wscalculator.calculator.calculatorservice になります。

4.11.2. 複数のクライアントの設定

上記の例では、myCalculator という単一のクライアントのみを設定しました。もちろん、異なる

URL を参照する複数のクライアントや、複数の識別子を使用して異なる SEI を実装する複数のクライアントを設定できます。

application.properties

```
cxf.it.calculator.baseUri = http://localhost:8082
quarkus.cxf.client.myCalculator.wsdl = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService?wsdl
quarkus.cxf.client.myCalculator.client-endpoint-url = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService
quarkus.cxf.client.myCalculator.service-interface = org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService

# another client
quarkus.cxf.client.anotherCalculator.wsdl = https://acme.com/ws/WeatherService?wsdl
quarkus.cxf.client.anotherCalculator.client-endpoint-url = https://acme.com/ws/WeatherService
quarkus.cxf.client.anotherCalculator.service-interface = org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService
```

4.11.3. 高度なクライアント設定

アプリケーションのすべてのクライアントをグローバルに設定するには、以下のスニペット例を使用して HttpConduit を設定します。これにより、クライアントに HTTPClientPolicy、AuthorizationPolicy、ProxyAuthorizationPolicy、または TLSClientParameters を設定できます。

```
void onStart(@Observes StartupEvent ev) {

    HTTPConduitConfigurer httpConduitConfigurer = new HTTPConduitConfigurer() {
        public void configure(String name, String address, HTTPConduit c) {
            AsyncHTTPConduit conduit = (AsyncHTTPConduit)c;
            // use setter to configure client
            conduit.getHttpAsyncClient().getCredentialsProvider().setCredentials(
AuthScope.ANY,
             new NTCredentials( USER,PWD, "", DOM ) );
            conduit.getClient().setAllowChunking( false );
            conduit.getClient().setAutoRedirect( true );
        }
    };

    final Bus bus = BusFactory.getDefaultBus();
    bus.setExtension(httpConduitConfigurer, HTTPConduitConfigurer.class);
}
```

4.11.4. Pure クライアントアプリケーション

Quarkus バッチ（たとえば定期的にスケジュール）やコマンドラインアプリケーションは HTTP サーバーなしで行う場合があります。以下のプロパティーを使用して、起動時に HTTP サーバーが起動しないようにします。

```
quarkus.http.host-enabled = false
```

4.11.5. リソースリークの防止

CXF クライアントプロキシーは java.io.Closeable を実装します。したがって、クライアントが必要なくなったら ((Closeable) proxy).close （） を呼び出し、スレッドなどの関連するすべてのシステムリソースを解放することが重要です。

Quarkus CXF は、CDI コンテナーで破棄されるとすぐに @io.quarkiverse.cxf.annotation.CXFClient で挿入されたクライアントを自動的に閉じます。

手動で作成されたクライアントプロキシーの場合、(Closeable) proxy）.close （）を呼び出す必要があります。

```
import java.net.URL;
import javax.xml.namespace.QName;
import jakarta.xml.ws.Service;
import java.io.Closeable;

final URL serviceUrl = new URL("http://localhost/myService?wsdl");
final QName qName = new QName("http://acme.org/myNamespace", "MyService");
final Service service = jakarta.xml.ws.Service.create(serviceUrl, qName);
final MyService proxy = service.getPort(MyService.class);

try {
    proxy.doSomething();
} finally {
    ((Closeable) proxy).close();
}
```

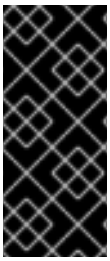
4.12. リバースプロキシーの背後で実行

Quarkus で実行されるサービスに対して目的とする SOAP 要求は、追加のヘッダー(X-Forwarded-Hostなど)を生成するプロキシー経由でルーティングして、関係時に変更または失われるプロキシーサーバーのクライアント側からの情報を保持することができます。このシナリオでは、Quarkus は、これらのヘッダーの値を反映したプロトコル、ホスト、ポート、URI などの情報を自動的に更新するように設定できます。

ヒント

詳細は、Quarkus HTTP リファレンス を参照してください。

さまざまな X-Forwarded ヘッダーの Quarkus CXF サポートは、Quarkus 設定と連携して機能します。



重要

この機能を有効にすると、サーバーはいくつかのセキュリティー問題（情報スプーフィングなど）に晒されます。リバースプロキシーの背後で実行している場合にのみ、アクティベートすることを検討してください。

以下は、関連する Quarkus プロパティーおよびその Quarkus CXF への影響です。

- quarkus.http.proxy.proxy-address-forwarding: リクエスト先部分の書き換えを可能にするメインのスイッチ。
  - 有効にすると、リクエストフィールドの書き換えは、CXF サーバースタック全体で有効になります。
  - 有効にすると、X-Forwarded-Proto および X-Forwarded-Port ヘッダーを介して渡される値を使用して、プロトコル部分と、jakarta.servlet.http.HttpServletRequest.getRequestURL （）によって返された URL のポート部分をそれぞれ設定します。
  - 有効にすると、X-Forwarded-For 経由で渡された値は jakarta.servlet.ServletRequest.getRemoteAddr （）によって返されます。
- quarkus.http.proxy.enable-forwarded-host: jakarta.servlet.http.HttpServletRequest.getRequestURL （）によって返される URL のホスト部分の書き換えを有効にします。実際のホスト名は、quarkus.http.proxy.forwarded-host-header を介して設定されたヘッダーから取得されます（デフォルトは X-Forwarded-Hostです）。
- quarkus.http.proxy.enable-forwarded-prefix:

jakarta.servlet.http.HttpServletRequest.getRequestURL （）によって返される URL のパス部分の書き換えと、jakarta.servlet.http.HttpServletRequest.getRequest URI （）によって返される URI の書き換えを有効にします。実際のパス接頭辞は、quarkus.http.proxy.forwarded-prefix-header を介して設定されたヘッダーから取得されます（デフォルトは X-Forwarded-Prefixです）。

application.properties にコピーする最も一般的なスニペットを次に示します。

```
quarkus.http.proxy.proxy-address-forwarding = true
quarkus.http.proxy.enable-forwarded-host = true
quarkus.http.proxy.enable-forwarded-prefix = true
```

これらの設定で観察可能な影響の 1 つは、http://localhost:8080/services/my-service?wsdl で提供される WSDL の location 値の変更です。たとえば、リクエストに次のヘッダーが含まれる場合

```
X-Forwarded-Proto: https
X-Forwarded-Host: api.example.com
X-Forwarded-Port: 443
X-Forwarded-Prefix: /my-prefix
```

次に、http://localhost:8080/services/my-service?wsdl で提供される WSDL には以下の 場所 が含まれます。

```
...
<soap:address location="https://api.example.com:443/my-prefix/services/my-service"/>
...
```

4.13. 契約の最初のアプローチとコードの最初のアプローチ

コントラクトの最初の開発モードとコードファースト開発モードの両方が、Quarkus CXF によって完全にサポートされています。

4.13.1. コントラクト最初のクライアント

SOAP サービスは WSDL によって記述されます。操作、パラメーター、戻り値などを定義するコントラクトです。WSDL は、完全なクライアントのコードを生成するのに十分なリッチです。CXF は、そのための wsdl2java ユーティリティーを提供します。

Quarkus CXF は wsdl2java を quarkus-cxf エクステンションでラップするため、直接使用する必要はありません。

使用方法の詳細は、ユーザーガイドの WSDL からのモデルクラスの生成 を 参照してください。

一般の紹介として consumer の開発 を確認することもできます。

4.13.2. コントラクトファーストサービス

サービスを実装する場合は、WSDL からの Java コードの生成 も便利な場合があります。wsdl2java は、(JAXB アノテーションを使用して)モデルクラスと(JAX-WS アノテーションを使用して)サービスインターフェイスを生成できます。その後、これらのインターフェイスの実装を提供するタスクになります。

基礎となる概念をよりよく理解するには、CXF ドキュメントの WSDL First Service Development セクションを確認してください。

4.13.3. コード最初のサービス

利用できるもう 1 つの有効なオプションは、JAX-WS と JAXB を使用して、Java でサービスを書き込むことです。次に、WSDL コントラクトを取得する方法は 2 つあります。

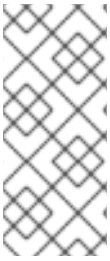
1. サービスを起動し、http://your-host/your-service?wsdlでクライアントを指定します。
2. ビルド時 に Java クラスから WSDL ドキュメントを生成 する

ヒント

詳細は、CXF ドキュメントの コードファースト開発 セクションを確認してください。

4.14. WSDL からモデルクラスを生成します

quarkus-cxf エクステンションは、Quarkus コード生成フェーズ中に WSDL からの Java クラスの生成をサポートします。



コード例

このセクションに示されているコードスニペットは、Quarkus CXF のソースツリーの クライアント統合テスト から取得されます。実行可能な例として確認することを推奨します。

CXF コード生成が機能するには、次の 2 つのことを設定する必要があります。

- プロジェクトに io.quarkiverse.cxf:quarkus-cxf 依存関係がある。
- Maven プロジェクトの場合、 generate-code ゴールが quarkus-maven-plugin の設定に存在する必要があります。

pom.xml

```
<plugin>
  <groupId>com.redhat.quarkus.platform</groupId>
  <artifactId>quarkus-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build</goal>
        <goal>generate-code</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- Gradle プロジェクトには、io.quarkus プラグインの追加設定は必要ありません。
- src/main/resources、src/test/resources またはサブディレクトリーの下に WSDL ファイルを配置します。
- WSDL ファイル名の終了は .wsdlにする必要があります。

- quarkus.cxf.codegen.wsdl2java.includes設定プロパティー を、処理する WSDL ファイルに一致するパターンに設定します。src/main/resources/wsdl または src/test/resources/wsdl 下のすべての WSDL ファイルを処理する場合は、以下のように設定します。

  application.properties

  ```
  quarkus.cxf.codegen.wsdl2java.includes = wsdl/*.wsdl
  ```

これにより、target/generated-sources/wsdl2java または target/generated-test-sources/wsdl2java ディレクトリーで Java クラスが生成されます。そこで自動的にコンパイラプラグインによって選択されます。そのため、アプリケーションやテストコードから自由に参照できます。

quarkus-cxf コード生成は、内部下の CXF の wsdl2Java ユーティリティーを使用することに注意してください。wsdl2Java は、include および excludes によって選択さ れ た各 WSDL ファイルに対して個別に呼び出されます。

4.14.1. カスタムパラメーター

quarkus.cxf.codegen. wsdl2java.additional-params 設定パラメーターにより、カスタムパラメーターを wsdl2java に渡すことができます。

4.14.2. 追加のパラメーター

WSDL ファイルごとに異なる additional-params が必要な場合は、それぞれに異なる名前のパラメーターセットを定義できます。以下に例を示します。

application.properties

```
# Parameters for foo.wsdl
quarkus.cxf.codegen.wsdl2java.foo-params.includes = wsdl/foo.wsdl
quarkus.cxf.codegen.wsdl2java.foo-params.wsdl-location = wsdl/foo.wsdl
# Parameters for bar.wsdl
```

```
quarkus.cxf.codegen.wsdl2java.bar-params.includes = wsdl/bar.wsdl
quarkus.cxf.codegen.wsdl2java.bar-params.wsdl-location = wsdl/bar.wsdl
quarkus.cxf.codegen.wsdl2java.bar-params.xjc = ts
```

ヒント

io.quarkiverse.cxf:quarkus-cxf-xjc-plugins 依存関係をプロジェクトに追加して、-xjc-Xbg,-xjc-Xdv,-xjc-Xjavadoc,-xjc-Xproperty-listener,-xjc-Xts および - xjc-Xwsdlextension wsdl 2java パラメーターを使用できるようにします。

4.15. JAVA から WSDL ドキュメントを生成する

たとえば、Maven アーティファクトとして配布するため、http://your-host/your-service?wsdl でサービスによって提供される WSDL が不十分な場合、java2ws を使用してビルド時に WSDL ドキュメントを生成できます。

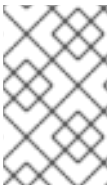
CXF を直接提供する java2ws ツールを呼び出すか、cxf-java2ws-plugin を使用する必要はありません。

quarkus-cxf は java2ws をラップするため、アプリケーションの他の要素として application.properties で設定することができます。

以下に例を示します。

application.properties

```
quarkus.cxf.java2ws.includes =
io.quarkiverse.cxf.it.server.HelloService,io.quarkiverse.cxf.it.server.FaultyHelloService
quarkus.cxf.java2ws.wsdl-name-template =
%TARGET_DIR%/Java2wsTest/%SIMPLE_CLASS_NAME%-from-java2ws.wsdl
```



注記

ANNOTATIONS

注記

4.15.1. 関連項目

-

4.16.

```
@org.apache.cxf.feature.Features (features = {"org.apache.cxf.ext.logging.LoggingFeature"})
@org.apache.cxf.interceptor.InInterceptors (interceptors = {"org.acme.Test1Interceptor" })
@org.apache.cxf.interceptor.InFaultInterceptors (interceptors = {"org.acme.Test2Interceptor"
})
@org.apache.cxf.interceptor.OutInterceptors (interceptors = {"org.acme.Test1Interceptor" })
@org.apache.cxf.interceptor.InFaultInterceptors (interceptors =
{"org.acme.Test2Interceptor","org.acme.Test3Intercetpor" })
@WebService(endpointInterface = "org.acme.SayHi", targetNamespace = "uri:org.acme")
public class SayHiImplementation implements SayHi {
  public long sayHi(long arg) {
     return arg;
```

```
    }
    //...
}
```

```
quarkus.cxf.endpoint."/greeting-service".features=org.apache.cxf.ext.logging.LoggingFeature
quarkus.cxf.endpoint."/greeting-service".in-interceptors=org.acme.Test1Interceptor
quarkus.cxf.endpoint."/greeting-service".out-interceptors=org.acme.Test1Interceptor
quarkus.cxf.endpoint."/greeting-service".in-fault-
interceptors=org.acme.Test2Interceptor,org.acme.Test3Intercetpor
quarkus.cxf.endpoint."/greeting-service".out-fault-interceptors=org.acme.Test1Intercetpor
```

クラスローディング

4.17.

application.properties

```
# A web service endpoint with multiple Handler classes
quarkus.cxf.endpoint."/greeting-
service".handlers=org.acme.MySOAPHandler,org.acme.AnotherSOAPHandler

# A web service client with a single Handler
quarkus.cxf.client."greeting-client".handlers=org.acme.MySOAPHandler
```

```
import jakarta.xml.ws.handler.soap.SOAPHandler;
import jakarta.xml.ws.handler.soap.SOAPMessageContext;
```

```
public class MySOAPHandler implements SOAPHandler<SOAPMessageContext> {

    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
        return true;
    }
    // other methods
}
```

クラスローディング

4.18.

```
import jakarta.xml.transform.stream.StreamSource;
import jakarta.xml.ws.BindingType;
import jakarta.xml.ws.Provider;
import jakarta.xml.ws.Service;
import jakarta.xml.ws.ServiceMode;
import jakarta.xml.ws.WebServiceProvider;
import java.io.StringReader;

@WebServiceProvider
@ServiceMode(value = Service.Mode.PAYLOAD)
public class StreamSourcePayloadProvider implements Provider<StreamSource> {

    public StreamSourcePayloadProvider() {
    }

    public StreamSource invoke(StreamSource request) {
        String payload = StaxUtils.toString(request);

        // Do some interesting things ...

        StreamSource response = new StreamSource(new StringReader(payload));
        return response;
    }
}
```

```
# A web service endpoint with the Provider implementation class
quarkus.cxf.endpoint."/stream-source".implementor=org.acme.StreamSourcePayloadProvider
```

クラスローディング

4.19. 例

4.20.

4.20.1.

```
package org.acme.cxf;

import ...

@Slf4j
@WebService(endpointInterface = "org.acme.cxf.WeatherWebService")
public class WeatherWebServiceImpl implements WeatherWebService {

    @Inject
    BackEndWeatherService backEndWeatherService;

    private Map<String, DailyTemperature> dailyTempByZipCode =
Collections.synchronizedMap(new LinkedHashMap<>());
```

```
    public WeatherWebServiceImpl() {
        this.dailyTempByZipCode.addAll(
          this.backEndWeatherService.getDailyForecast(Instant.now()));
    }

    @Override
    public DailyTemperature estimationTemperatures(String zipCode) {
        log.info("Daily estimation temperatures forecast called with '{}' zip code paramter",
zipCode);
        return this.dailyTempByZipCode.get(zipCode);
    }
}
```

```
quarkus.cxf.path=/soap
quarkus.cxf.endpoint."/weather".implementor=org.acme.cxf.WeatherWebServiceImpl
```

```
package org.acme.reasteasy;

import ...

@Slf4j
@Path("/healthcheck")
public class HealthCheckResource {

 @Inject
    BackEndWeatherService backEndWeatherService;

 @GET
 public Response doHealthCheck() {
  if(this.backEndWeatherService.isAvailable()) {
            return Response.ok().build();
  } else {
            return Response.status(Response.Status.SERVICE_UNAVAILABLE);
  }
 }
}
```

```
quarkus.resteasy.path=/rest
```

- http://localhost:8080/rest/healthcheck for REST
-

4.20.2.

```
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]      compile: 161 459,15 ms,  8,54 GB
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]        image: 158 272,73 ms,  8,43 GB
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]        write:     205,82 ms,  8,43 GB
Fatal error:com.oracle.svm.core.util.VMError$HostedError: java.lang.RuntimeException: oops
: expected ASCII string!
com.oracle.svm.reflect.OperationOrderStatusType_CRÉÉ_f151156b0d42ecdbdfb919501d8a86
dda8733012_1456.hashCode
 at com.oracle.svm.core.util.VMError.shouldNotReachHere(VMError.java:72)
```

```
@XmlType(name = "OperationOrderStatusType")
@XmlEnum
public enum OperationOrderStatusType {

    @XmlEnumValue("Cr\u00e9\u00e9")
    CRÉÉ("Cr\u00e9\u00e9"),
    @XmlEnumValue("A communiquer")
    A_COMMUNIQUER("A communiquer"),
    @XmlEnumValue("En attente de r\u00e9ponse")
    EN_ATTENTE_DE_RÉPONSE("En attente de r\u00e9ponse"),
    @XmlEnumValue("Attribu\u00e9")
    ATTRIBUÉ("Attribu\u00e9"),
    @XmlEnumValue("Clotur\u00e9")
    CLOTURÉ("Clotur\u00e9"),
    @XmlEnumValue("Annul\u00e9")
    ANNULÉ("Annul\u00e9");
    private final String value;

    OperationOrderStatusType(String v) {
        value = v;
    }

    public String value() {
        return value;
```

```
    }

    public static OperationOrderStatusType fromValue(String v) {
        for (OperationOrderStatusType c: OperationOrderStatusType.values()) {
            if (c.value.equals(v)) {
                return c;
            }
        }
        throw new IllegalArgumentException(v);
    }
}
```

```
@XmlType(name = "OperationOrderStatusType")
@XmlEnum
public enum OperationOrderStatusType {

    @XmlEnumValue("Créé")
    CREE("Créé"),
    @XmlEnumValue("A communiquer")
    A_COMMUNIQUER("A communiquer"),
    @XmlEnumValue("En attente de réponse")
    EN_ATTENTE_DE_REPONSE("En attente de réponse"),
    @XmlEnumValue("Attribué")
    ATTRIBUE("Attribué"),
    @XmlEnumValue("Cloturé")
    CLOTURE("Cloturé"),
    @XmlEnumValue("Annulé")
    ANNULE("Annulé");
    private final String value;

    OperationOrderStatusType(String v) {
        value = v;
    }

    public String value() {
        return value;
    }

    public static OperationOrderStatusType fromValue(String v) {
        for (OperationOrderStatusType c: OperationOrderStatusType.values()) {
            if (c.value.equals(v)) {
                return c;
            }
        }
        throw new IllegalArgumentException(v);
    }
}
```

4.21.

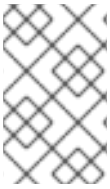
第5章

5.1.

表5.1

	サポートレベル	Since	サポートされる標準
		0.1.0	
		0.14.4	
Quarkus CXF OpenTelemetry **quarkus-cxf-integration-tracing-opentelemetry**		2.7.0	
		0.14.4	
Quarkus CXF WS-ReliableMessaging **quarkus-cxf-rt-ws-rm**		1.5.3	WS-ReliableMessaging
		1.5.3	WS-Trust
		1.1.0	
Quarkus CXF XJC Plugins **quarkus-cxf-xjc-plugins**		1.5.11	

5.2.



注記

5.2.1.

5.2.2.

- JAXB
-

5.2.3. トランスポート

-
-
  - Basic 認証
-
  Asynchronous Client HTTP Transport via quarkus-cxf-rt-transports-http-hc5

5.2.4. ツール

-

-

5.2.5.

バインディング	プロパティー値
	http://schemas.xmlsoap.org/wsdl/soap/http
SOAP 1.2	**http://www.w3.org/2003/05/soap/bindings/HTTP/**
	http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true
	http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true

5.3.

	代替方法
JAX-RS **cxf-rt-frontend-jaxrs** **cxf-rt-rs-client**	
JiBX	
JBI **cxf-rt-transports-jbi** **cxf-rt-bindings-jbi**	

	代替方法
CORBA **cxf-rt-bindings-corba**	
XMLBeans	

5.4.

アノテーション	ステータス
@org.apache.cxf.feature.Features	サポート対象
@org.apache.cxf.interceptor.InInterceptors	サポート対象
@org.apache.cxf.interceptor.OutInterceptors	サポート対象
@org.apache.cxf.interceptor.OutFaultInterceptors	サポート対象
@org.apache.cxf.interceptor.InFaultInterceptors	サポート対象
@org.apache.cxf.annotations.WSDLDocumentation	サポート対象
@org.apache.cxf.annotations.WSDLDocumentationCollection	サポート対象
@org.apache.cxf.annotations.SchemaValidation	サポート対象
@org.apache.cxf.annotations.DataBinding	

アノテーション	ステータス
@org.apache.cxf.ext.logging.Logging	サポート対象
@org.apache.cxf.annotations.GZIP	サポート対象
@org.apache.cxf.annotations.FastInfoset	
@org.apache.cxf.annotations.EndpointProperty	サポート対象
@org.apache.cxf.annotations.EndpointProperties	サポート対象
@org.apache.cxf.annotations.Policy	サポート対象
@org.apache.cxf.annotations.Policies	サポート対象
@org.apache.cxf.annotations.UseAsyncMethod	サポート対象

第6章

6.1.

6.1.1. Maven コーディネート

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf</artifactId>
</dependency>
```

6.1.2. サポートされる標準

- JAX-WS
- JAXB
- WS-Addressing
- WS-Policy
- MTOM

6.1.3. 使用方法

6.1.4. 設定

その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

設定プロパティー	タイプ	デフォルト
quarkus.cxf.path	string	/services
以前のバージョン		
quarkus.cxf.min-chunk-size	int	128
quarkus.cxf.output-buffer-size	int	8191
quarkus.cxf.decoupled-endpoint-base	string	

設定プロパティー	タイプ	デフォルト

```
quarkus.cxf.decoupled-endpoint-base = https://api.example.com:${quarkus.http.ssl-port}${quarkus.cxf.path}
```

```
quarkus.cxf.decoupled-endpoint-base = http://api.example.com:${quarkus.http.port}${quarkus.cxf.path}
```

```
import java.util.Map;
import jakarta.inject.Inject;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.UriInfo;
import jakarta.xml.ws.BindingProvider;
import io.quarkiverse.cxf.annotation.CXFClient;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/my-rest")
public class MyRestEasyResource {

    @Inject
    @CXFClient("hello")
    HelloService helloService;

    @ConfigProperty(name = "quarkus.cxf.path")
                String quarkusCxfPath;

    @POST
    @Path("/hello")
    @Produces(MediaType.TEXT_PLAIN)
       public String hello(String body, @Context UriInfo uriInfo) throws IOException {

       // You may consider doing this only once if you are sure that your service is accessed
       // through a single hostname
       String decoupledEndpointBase = uriInfo.getBaseUriBuilder().path(quarkusCxfPath);
       Map>String, Object< requestContext = ((BindingProvider)
       helloService).getRequestContext();
       requestContext.put("org.apache.cxf.ws.addressing.decoupled.endpoint.base",
       decoupledEndpointBase);

       return wsrmHelloService.hello(body);
    }
}
```

quarkus.cxf.logging.enabled-for		**none**

設定プロパティー	タイプ	デフォルト
quarkus.cxf.logging.pretty	boolean	false
quarkus.cxf.logging.limit	int	49152
quarkus.cxf.logging.in-mem-threshold	long	-1
quarkus.cxf.logging.log-binary	boolean	false
quarkus.cxf.logging.log-multipart	boolean	true
quarkus.cxf.logging.verbose	boolean	true

設定プロパティー	タイプ	デフォルト
	string	
	boolean	**false**
	boolean	**false**
つまり、		
	string	
quarkus.cxf.endpoint."endpoints".wsdl	**string**	
	string	
• **http://schemas.xmlsoap.org/wsdl/soap/http** for SOAP11HTTP_BINDING • **http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true** for SOAP11HTTP_MTOM_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/** for SOAP12HTTP_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true** for SOAP12HTTP_MTOM_BINDING		
	string	

設定プロパティー	タイプ	デフォルト
	boolean	
	int	
	long	
	boolean	
	boolean	
	boolean	

設定プロパティー	タイプ	デフォルト

例 :
quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature
quarkus.cxf.endpoint."/fruit".features = #myCustomLoggingFeature
import org.apache.cxf.ext.logging.LoggingFeature;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Produces;
class Producers {
@Produces
@ApplicationScoped
LoggingFeature myCustomLoggingFeature() {
LoggingFeature loggingFeature = new LoggingFeature();
loggingFeature.setPrettyLogging(true);
return loggingFeature;
}
}		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".wsdl	string	
	string	
• **http://schemas.xmlsoap.org/wsdl/soap/http** for SOAP11HTTP_BINDING • **http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true** for SOAP11HTTP_MTOM_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/** for SOAP12HTTP_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true** for SOAP12HTTP_MTOM_BINDING		
	string	
	string	
	string	
quarkus.cxf.client."clients".username	string	
quarkus.cxf.client."clients".password	string	
	boolean	false

設定プロパティー	タイプ	デフォルト
値がプリティーの場合(2.7.0 以降)、 **pretty** 属性は実質的に **true** に設定されます。デフォルトは quarkus.cxf.logging.enabled-for によって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_ENABLED** **Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.pretty	**boolean**	
true の場合、XML 要素はログでインデントされます。それ以外の場合は、未インデント表示されます。デフォルトは quarkus.cxf.logging.prettyによって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_PRETTY** **以降、 Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.limit	**int**	
ログ内で切り捨てられるメッセージの長さ（バイト単位）。デフォルトは quarkus.cxf.logging.limitによって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_LIMIT** **Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.in-mem-threshold	**long**	
ディスクに書き込まれるメッセージの長さ（バイト単位）。**-1** は無制限です。デフォルトは quarkus.cxf.logging.in-mem-thresholdで指定されます。 **環境変数**:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_IN_MEM_THRESHOLD** (**Quarkus CXF: 2.6.0 以降**)		
quarkus.cxf.client."clients".log-binary	**boolean**	
true の場合、バイナリーペイロードがログに記録されます。それ以外の場合は、ログに記録されません。デフォルトは quarkus.cxf.logging.log-binaryによって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_LOG_BINARY** **Quarkus CXF 以降**: 2.6.0		
quarkus.cxf.client."clients".log-multipart	**boolean**	
true の場合、マルチパートペイロードがログに記録されます。それ以外の場合は、それらはログに記録されません。デフォルトは quarkus.cxf.logging.log-multipartによって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT_CLIENT__CLIENTS__LOGGING_LOG_MULTIPART** 。**Quarkus CXF 以降**: 2.6.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".logging.verbose	boolean	
true の場合、詳細なロギングが有効になります。それ以外の場合は有効になりません。デフォルトは **quarkus.cxf.logging.verbose**によって提供されます。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_VERBOSE** Quarkus CXF: 2.6.0		
quarkus.cxf.client."clients".logging.in-binary-content-media-types	文字列のリスト	
log-binary が **true** でない限り、コンテンツがログに記録されない **LoggingInInterceptor** のデフォルト値に追加する追加のバイナリーメディアタイプのコンマ区切りリスト。デフォルトは **quarkus.cxf.logging.in-binary-content-media-types**によって指定されます。 環境変数:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_IN_BINARY_CONTENT_MEDIA_TYPES**		
quarkus.cxf.client."clients".logging.out-binary-content-media-types	文字列のリスト	
log-binary が **true** でない限り、コンテンツがログに記録されない **LoggingOutInterceptor** のデフォルト値に追加する追加のバイナリーメディアタイプのコンマ区切りリスト。デフォルトは **quarkus.cxf.logging.out-binary-content-media-types**によって指定されます。 環境変数:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_OUT_BINARY_CONTENT_MEDIA_TYPES** （Quarkus CXF 以降: 2.6.0 以降		
quarkus.cxf.client."clients".logging.binary-content-media-types	文字列のリスト	
log-binary が **true** でない限り、**LoggingOutInterceptor** および **LoggingInInterceptor** のデフォルト値に追加する追加のバイナリーメディアタイプのコンマ区切りリスト。デフォルトは **quarkus.cxf.logging.binary-content-media-types**で指定されます。 環境変数:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_BINARY_CONTENT_MEDIA_TYPES** (Quarkus CXF: 2.6.0 以降)		
quarkus.cxf.client."clients".logging.sensitive-element-names	文字列のリスト	

設定プロパティー	タイプ	デフォルト
ログにマスクされる機密情報が含まれる XML 要素のコンマ区切りリスト。デフォルトは **quarkus.cxf.logging.sensitive-element-names**で指定されます。 **環境変数**:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_SENSITIVE_ELEMENT_NAMES** (Quarkus CXF: 2.6.0 **以降**)		
quarkus.cxf.client."clients".logging.sensitive-protocol-header-names	**文字列**のリスト	
ログにマスクされる機密情報が含まれるプロトコルヘッダーのコンマ区切りリスト。デフォルトは **quarkus.cxf.logging.sensitive-protocol-header-names**によって提供されます。 **環境変数**:**QUARKUS_CXF_CLIENT_CLIENT__LOGGING_SENSITIVE_PROTOCOL_HEADER_NAMES** (Quarkus CXF: 2.6.0 **以降**)		
quarkus.cxf.client."clients".features	**文字列**のリスト	
完全修飾 CXF 機能クラス名のコンマ区切りリスト。 以下に例を示します。 `quarkus.cxf.endpoint.myClient.features = org.apache.cxf.ext.logging.LoggingFeature` **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__FEATURES**		
quarkus.cxf.client."clients".handlers	**文字列**のリスト	
ハンドラークラスのコンマ区切りリスト **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__HANDLERS**		
quarkus.cxf.client."clients".in-interceptors	**文字列**のリスト	
InInterceptor クラスのコンマ区切りリスト **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__IN_INTERCEPTORS**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".out-interceptors OutInterceptor クラスのコンマ区切りリスト **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__OUT_INTERCEPTORS**	**文字列**のリスト	
quarkus.cxf.client."clients".out-fault-interceptors OutFaultInterceptor クラスのコンマ区切りリスト **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__OUT_FAULT_INTERCEPTORS**	**文字列**のリスト	
quarkus.cxf.client."clients".in-fault-interceptors InFaultInterceptor クラスのコンマ区切りリスト **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__IN_FAULT_INTERCEPTORS**	**文字列**のリスト	
quarkus.cxf.client."clients".connection-timeout コンシューマーがタイムアウトするまでの接続の確立を試みる期間（ミリ秒単位）を指定します。0 は無限です。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__CONNECTION_TIMEOUT** **Quarkus CXF**: 2.2.3	**long**	**30000**
quarkus.cxf.client."clients".receive-timeout コンシューマーがタイムアウトするまでの応答を待つ期間 (ミリ秒単位) を指定します。0 は無限です。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__RECEIVE_TIMEOUT** **Quarkus CXF**: 2.2.3	**long**	**60000**
quarkus.cxf.client."clients".connection-request-timeout 接続マネージャーからの接続を要求するときに使用される時間をミリ秒単位で指定します。0 は無限です。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__CONNECTION_REQUEST_TIMEOUT** **Quarkus CXF: 2.2.3 以降**	**long**	**60000**
quarkus.cxf.client."clients".auto-redirect	**boolean**	**false**

設定プロパティー	タイプ	デフォルト
コンシューマーが自動的に発行されたリダイレクトに従うかどうかを指定します。（名前は標準の一部ではない） **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__AUTO_REDIRECT** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".max-retransmits	**int**	**-1**
リダイレクトに許可される再送信の最大数を指定します。承認の再送信は再送信数に含まれます。それぞれのリダイレクトにより、UNAUTHORIZED 応答コード（つまり 401）に別の再送信が行われる場合があります。負の数値は、無制限の再送信を示しますが、ループ保護が提供されます。デフォルトは無制限です（名前は標準の一部ではありません）。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__MAX_RETRANSMITS** 。**Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".allow-chunking	**boolean**	**true**
true の場合、クライアントは必要に応じてチャンクストリームを自由に使用できますが、チャンクストリームを使用する必要はありません。false の場合、クライアントはすべてのケースで通常のチャンク化されたリクエストを使用する必要があります。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__ALLOW_CHUNKING** **以降、Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".chunking-threshold	**int**	**4096**
AllowChunking が true の場合、メッセージがチャンクを取得するしきい値を設定します。この制限のメッセージはチャンクにはなりません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__CHUNKING_THRESHOLD** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".chunk-length	**int**	**-1**
HttpURLConnection のチャンクの長さを指定します。この値は java.net.HttpURLConnection.setChunkedStreamingMode (int chunklen)で使用されます。chunklen は、各チャンクに書き込むバイト数を示します。chunklen が 0 以下の場合は、デフォルト値が使用されます。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__CHUNK_LENGTH** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".accept	**string**	

設定プロパティー	タイプ	デフォルト
クライアントが処理する準備ができている MIME タイプを指定します(HTML、JPEG、GIF など)。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".accept-language	**string**	
クライアントが希望する言語を指定します（例：英語、フランス語など）。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT_LANGUAGE**		
quarkus.cxf.client."clients".accept-encoding	**string**	
クライアントが処理する準備されるエンコーディングを指定します(gzip など)。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT_ENCODING** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".content-type	**string**	
ポスト要求で送信されるストリームのコンテンツタイプを指定します（これは Web サービスの場合は text/xml であるか、クライアントがフォームデータを送信している場合は application/x-www-form-urlencoded に設定することができます）。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__CONTENT_TYPE** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".host	**string**	
要求が呼び出されるリソースのインターネットホストおよびポート番号を指定します。これは、デフォルトで URL に基づいて送信されます。DNS のシナリオやアプリケーション設計によっては、この設定を要求する場合がありますが、通常は必須ではありません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__HOST** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".connection	**close、keep-alive**	**keep-alive**
接続の配置。各要求/応答ダイアログの後にサーバーへの接続が閉じられた場合。Keep-Alive クライアントは、サーバーが接続を開いたままにするように要求し、サーバーがキープアライブ要求を受け入れると、接続は再利用されます。多くのサーバーおよびプロキシーはキープアライブリクエストを受け入れません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__CONNECTION** **Quarkus CXF**: 2.2.3		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".cache-control	string	
no-cache の指定に最も一般的に使用されますが、標準では、リクエストがドzen などのリクエストに関連するディレクティブをキャッシュすることがサポートされます。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__CACHE_CONTROL** Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".version	string	auto
接続に使用される HTTP バージョン。デフォルト値 **auto** は、**quarkus.cxf.client.myClient.http-conduit-factory** を介して定義された **HTTPConduit** 実装のデフォルトを使用します。その他の可能な値：**1.1**、**2**。 **HTTPConduit** 実装によっては、これらの値の一部がサポートされない可能性があります。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__VERSION** Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".browser-type	string	
User-Agent HTTP ヘッダーの値。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__BROWSER_TYPE** Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".decoupled-endpoint	string	
URI パス(/で始まる)または別のプロバイダー → コンシューマー接続を介して応答を受信するための完全な URI。値が / で始まる場合は、WS-Addressing **ReplyTo** メッセージヘッダーの値として使用される前に、**quarkus.cxf.client.myClient.decoupled-endpoint-base** を介して設定されたベース URI が接頭辞として付けられます。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__DECOUPLED_ENDPOINT** Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-server	string	
プロキシーサーバーのアドレスが使用されている場合は、それを指定します。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER** Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-server-port	int	
プロキシーサーバーが使用するポート番号を指定します。 環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER_PORT** Quarkus CXF: 2.2.3		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".non-proxy-hosts	string	
プロキシー設定を使用しないホスト名のリストを指定します。例： • **localhost** - 単一のホスト名 • **localhost|www.google.com** - プロキシー設定を使用しない 2 つのホスト名 • **localhost|www.google.*|*.apache.org** - hostname patterns **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__NON_PROXY_HOSTS** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".proxy-server-type	HTTP、socks	http
プロキシーサーバーの種類を指定します。HTTP または SOCKS のいずれかを指定できます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER_TYPE** **Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".proxy-username	string	
プロキシー認証用のユーザー名 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__PROXY_USERNAME** **Quarkus CXF 以降**: 2.2.3		
quarkus.cxf.client."clients".proxy-password	string	
プロキシー認証のパスワード **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__PROXY_PASSWORD** **Quarkus CXF 以降**: 2.2.3		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".http-conduit-factory	**QuarkusCXFDefault、CXFDefault、HttpClientHTTPConduitFactory、URLConnectionHTTPConduitFactory**	
このクライアントの **HTTPConduitFactory** 実装を選択します。 • **QuarkusCXFDefault** （デフォルト）:**io.quarkiverse.cxf:quarkus-cxf-rt-transports-http-hc5** がクラスパスに存在する場合は、その **HTTPConduitFactory** 実装が使用されます。それ以外の場合は、この値は **URLConnectionHTTPConduitFactory** と同等になります（この問題 #992 が CXF で解決されると変更する可能性があります） • **CXFDefault**: **HTTPConduitFactory** 実装の選択が CXF のままになる • **HttpClientHTTPConduitFactory**: このクライアントの **HTTPConduitFactory** は、常に **org.apache.cxf.transport.http.HttpClientHTTPConduit** を返す実装に設定されます。これにより、**java.net.http.HttpClient** が基礎となる HTTP クライアントとして使用されます。 • **URLConnectionHTTPConduitFactory**: このクライアントの **HTTPConduitFactory** は、常に **org.apache.cxf.transport.http.URLConnectionHTTPConduit** を返す実装に設定されます。これにより、**java.net.HttpURLConnection** を基礎となる HTTP クライアントとして使用します。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__HTTP_CONDUIT_FACTORY**		
quarkus.cxf.client."clients".trust-store	**string**	
このクライアントのトラストストアの場所。リソースは最初にクラスパスで検索され、次にファイルシステム内で検索されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE**		
quarkus.cxf.client."clients".trust-store-password	**string**	
トラストストアのパスワード **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE_PASSWORD**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".trust-store-type	string	JKS
トラストストアのタイプ。 環境変数:QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE_TYPE		
quarkus.cxf.client."clients".hostname-verifier	string	
次のいずれかになります。 - よく知られている値のいずれか：AllowAllHostnameVerifier,HttpsURLConnectionDefaultHostnameVerifier - CDI コンテナーで検索する javax.net.ssl.HostnameVerifier を実装する完全修飾クラス名。 - CDI コンテナーで検索される # で始まる Bean 名。例：#myHostnameVerifier が指定されていない場合、HostnameVerifier の作成が CXF に委任され、デフォルトの org.apache.cxf.transport.https.httpclient.DefaultHostnameVerifier に返される org.apache.cxf.transport.https.httpclient. PublicSuffixMatcherLoader に 正式化されます。 環境変数:QUARKUS_CXF_CLIENT__CLIENTS__HOSTNAME_VERIFIER		
quarkus.cxf.client."clients".schema-validation.enabled-for	では、request、out、response、both、none	
XML スキーマ検証を有効にするメッセージを選択します。指定しない場合、@org.apache.cxf.annotations.SchemaValidation や @org.apache.cxf.annotations.EndpointProperty (key = "schema-validation-enabled", value = "true") アノテーションなどの他の方法で有効でない限り、XML スキーマ検証は実行されません。 環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SCHEMA_VALIDATION_ENABLED_FOR 以降、Quarkus CXF: 2.7.0		

6.2. メトリクス機能

Micrometer を使用してメトリクスを収集します。

重要

CXF メトリクス機能 とは異なり、この Quarkus CXF エクステンションは Dropwizard Metrics をサポートしません。Micrometer のみがサポートされています。

6.2.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-rt-features-metrics を使用して新しいプロジェクトを作成するか、これらのコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-features-metrics</artifactId>
</dependency>
```

6.2.2. 使用方法

CXF の Quarkus Micrometer エコシステムへの統合は、io.quarkiverse.cxf.metrics.QuarkusCxfMetricsFeature を使用して実装されます。アプリケーションが quarkus-cxf-rt-features-metrics に依存している限り、QuarkusCxfMetricsFeature のインスタンスは、Quarkus CXF によって作成されたすべてのクライアントおよびサービスエンドポイントに対して内部的に作成され、デフォルトで有効になります。これは、以下に記載されている quarkus.cxf.metrics.enabled-for、quarkus.cxf.client."clients".metrics.enabled および quarkus.cxf.endpoint."endpoints".metrics.enabled プロパティーを介して無効にできます。

6.2.2.1. 実行可能な例

Quarkus CXF ソースツリーには、Micrometer メトリクスに対応する 統合テスト があります。

当然、これは quarkus-cxf-rt-features-metricsに依存します。

pom.xml

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-features-metrics</artifactId>
</dependency>
```

quarkus-micrometer-registry-prometheus エクステンションを使用して、メトリクスを JSON 形式で、Prometheus にはエクスポートします。

pom.xml

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-micrometer-registry-prometheus</artifactId>
</dependency>
```

REST エンドポイントで収集したメトリクスを検査するには、以下の設定が必要です。

application.properties

```
quarkus.micrometer.export.json.enabled = true
quarkus.micrometer.export.json.path = metrics/json
quarkus.micrometer.export.prometheus.path = metrics/prometheus
```

上記をすべて配置したら、アプリケーションを Dev モードで起動できます。

```
$ mvn quarkus:dev
```

HelloService にリクエストを送信します。

```
$ curl \
  -d '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns2:helloResponse xmlns:ns2="http://it.server.metrics.cxf.quarkiverse.io/"><return>Hello
Joe!</return></ns2:helloResponse></soap:Body></soap:Envelope>' \
  -H 'Content-Type: text/xml' \
  -X POST \
  http://localhost:8080/metrics/client/hello
```

その後、上記のエンドポイントの出力の cxf.server.requests にメトリクスが表示されます。

```
$ curl http://localhost:8080/q/metrics/json
metrics: {
    ...
    "cxf.server.requests": {

"count;exception=None;faultCode=None;method=POST;operation=hello;outcome=SUCCESS;
status=200;uri=/soap/hello": 2,

"elapsedTime;exception=None;faultCode=None;method=POST;operation=hello;outcome=SUC
CESS;status=200;uri=/soap/hello": 64.4
    },
    ...
}
```

6.2.3. 設定

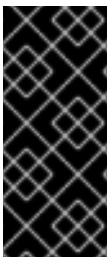
ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

設定プロパティー	タイプ	デフォルト
quarkus.cxf.metrics.enabled-for	クライアント、サービス、両方のサービス	both
メトリック収集をクライアント、サービス、両方、またはなしのどちらに対して有効にするかを指定します。このグローバル設定は、**quarkus.cxf.client."clients".metrics.enabled** または **quarkus.cxf.endpoint."endpoints".metrics.enabled** オプションを使用して、クライアントまたはサービスエンドポイントごとに上書きできます。 環境変数:**QUARKUS_CXF_METRICS_ENABLED_FOR** 以降、Quarkus CXF: 2.7.0		
quarkus.cxf.client."clients".metrics.enabled	boolean	true

設定プロパティー	タイプ	デフォルト
true で、**quarkus.cxf.metrics.enabled-for** が **both** または client に設定されている場合には、**MetricsFeature** がこのクライアントに追加されます。そうしないと、この機能はこのクライアントに追加されません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__METRICS_ENABLED** **Quarkus CXF 以降: 2.7.0**		
quarkus.cxf.endpoint."endpoints".metrics.enabled	**boolean**	**true**
true で、**quarkus.cxf.metrics.enabled-for** が **both** または **services** に設定されている場合には、**MetricsFeature** はこのサービスエンドポイントに追加されます。そうしないと、この機能はこのサービスエンドポイントに追加されません。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__METRICS_ENABLED** Quarkus CXF: 2.7.0		

6.3. OPENTELEMETRY

OpenTelemetry トレースを生成します。



重要

OpenTelemetry メトリクスとロギングはまだ Quarkus 側でも CXF 側でもサポートされていないため、Quarkus CXF はそれらをサポートできません。したがって、トレースは、この拡張機能でサポートされている唯一の OpenTelemetry 機能です。

6.3.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-integration-tracing-opentelemetry を使用して新しいプロジェクトを作成するか、これらのコーディネートを既存のプロジェクトに追加します。

```
<dependency>
  <groupId>io.quarkiverse.cxf</groupId>
  <artifactId>quarkus-cxf-integration-tracing-opentelemetry</artifactId>
```

```
</dependency>
```

6.3.2. 使用方法

このエクステンションは、org.apache.cxf.tracing.opentelemetry.OpenTelemetryFeature （サービスエンドポイントの場合）および org.apache.cxf.tracing.opentelemetry.OpenTelemetryClientFeature （クライアント用）の上に構築されます。これらのインスタンスは、Quarkus OpenTelemetry によって提供される io.opentelemetry.api.OpenTelemetry のインスタンスを使用して内部的に作成され、設定されます。

このトレースは、quarkus.cxf.otel.enabled-for、quarkus.cxf.client."clients".otel.enabled または quarkus.cxf.endpoint."endpoints".otel.enabled で明示的に無効にしない限り、Quarkus CXF によって作成されたすべてのクライアントおよびサービスエンドポイントに対してデフォルトで有効になります。

6.3.2.1. 実行可能な例

Quarkus CXF ソースツリーには、OpenTelemetry に対応する 統合テスト があります。io.opentelemetry:opentelemetry-sdk-testing の InMemorySpanExporter を使用しており、スパンをテストから簡単に検査できるようにします。その他のサポートされているスパンエクスポーターとコレクターの詳細は、Quarkus OpenTelemetry ガイド を参照してください。

6.3.3. 設定

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

設定プロパティー	タイプ	デフォルト
quarkus.cxf.otel.enabled-for	クライアント、サービス、両方のサービス	both

設定プロパティー	タイプ	デフォルト
クライアント、サービス、両方、またはなしに対して OpenTelemetry トレースを有効にするかどうかを指定します。このグローバル設定は、**quarkus.cxf.client."clients".otel.enabled** または **quarkus.cxf.endpoint."endpoints".otel.enabled** オプションをそれぞれ使用して、クライアントまたはサービスエンドポイントごとに上書きできます。 **環境変数:QUARKUS_CXF_OTEL_ENABLED_FOR** **以降、Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".otel.enabled	**boolean**	**true**
true で、**quarkus.cxf.otel.enabled-for** が **both** または client に設定されている場合には、**OpenTelemetryClientFeature** がこのクライアントに追加されます。そうしないと、機能はこのクライアントに追加されません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__OTEL_ENABLED** **Quarkus CXF 以降: 2.7.0**		
quarkus.cxf.endpoint."endpoints".otel.enabled	**boolean**	**true**
true で、**quarkus.cxf.otel.enabled-for** が **both** または **services** に設定されている場合には、**OpenTelemetryFeature** がこのサービスエンドポイントに追加されます。そうしないと、この機能はこのサービスエンドポイントに追加されません。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__OTEL_ENABLED** **Quarkus CXF**: 2.7.0		

6.4. WS-SECURITY

CXF フレームワークの WS-Security 実装を提供し、以下を可能にします。

- **サービス間で認証トークンを渡す**
- **メッセージまたはメッセージの一部の暗号化**

- メッセージの署名
- メッセージにタイムスタンプ

6.4.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-rt-ws-security を使用して新しいプロジェクトを作成するか、以下のコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-ws-security</artifactId>
</dependency>
```

6.4.2. サポートされる標準

- WS-Security
- WS-SecurityPolicy

6.4.3. 使用方法

CXF フレームワークの WS-Security (WSS)実装は、WSS4J に基づいています。起動する方法は 2 つあります。

- WS-SecurityPolicy の使用
- クライアントおよびサービスエンドポイントに WSS4J インターセプターを追加することにより。

そのため、セキュリティー要件は WSDL コントラクトの一部になるため、WS-SecurityPolicy が推奨されます。これにより、クライアントおよびサービスエンドポイントの実装だけでなく、ベンダー間の相互運用性も大幅に簡素化されます。

ただし、WS-SecurityPolicy を利用すると、CXF は内部で WSS4J インターセプターをセットアッ

プします。

ここでは、WSS4J インターセプターに関する手動のアプローチについては説明しませんが、WS-Security 統合テスト は例として参照できます。

6.4.3.1. WS-Security via WS-SecurityPolicy

ヒント

このセクションで使用されるサンプルコードスニペットは、Quarkus CXF のソースツリーの WS-SecurityPolicy 統合テスト のものです。

たとえば、クライアントとサービス間の通信が 機密（暗号化経由）で、そのメッセージが（デジタル署名を介して）改ざんされていないことを確認することを目的としています。また、X.509 証明書で自身を認証することで、クライアントが本人であることを保証する必要があります。

これらの要件はすべて、1 つの WS-SecurityPolicy ドキュメント で表現できます。

encrypt-sign-policy.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsp:Policy wsu:Id="SecurityServiceEncryptThenSignPolicy"
   xmlns:wsp="http://www.w3.org/ns/ws-policy"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
   <wsp:ExactlyOne>
      <wsp:All>
         (1)
         <sp:AsymmetricBinding xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
            <wsp:Policy>
               (2)
               <sp:InitiatorToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                        <wsp:Policy>
                           <sp:WssX509V3Token11/>
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:InitiatorToken>
               <sp:RecipientToken>
```

```
                <wsp:Policy>
                  <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/Never">
                    <wsp:Policy>
                      <sp:WssX509V3Token11/>
                    </wsp:Policy>
                  </sp:X509Token>
                </wsp:Policy>
              </sp:RecipientToken>
              <sp:AlgorithmSuite>
                <wsp:Policy>
                  <sp:Basic256/>
                </wsp:Policy>
              </sp:AlgorithmSuite>
              <sp:Layout>
                <wsp:Policy>
                  <sp:Strict/>
                </wsp:Policy>
              </sp:Layout>
              <sp:IncludeTimestamp/>
              <sp:ProtectTokens/>
              <sp:OnlySignEntireHeadersAndBody/>
              <sp:EncryptBeforeSigning/>
            </wsp:Policy>
          </sp:AsymmetricBinding>
          3
          <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body/>
          </sp:SignedParts>
          4
          <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body/>
          </sp:EncryptedParts>
          <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <wsp:Policy>
              <sp:MustSupportRefIssuerSerial/>
            </wsp:Policy>
          </sp:Wss10>
        </wsp:All>
      </wsp:ExactlyOne>
    </wsp:Policy>
```

1

AsymmetricBinding は、2 つの当事者間の通信を保護するための非対称（公開/秘密鍵）暗号の使用を指定します。

2

InitiatorToken は、メッセージのイニシエーター（送信側）が受信者に常に提供する必要のある X.509 証明書トークンを使用することを示します。

3

SignedParts は、整合性を確保するために SOAP メッセージのどの部分に署名するかを指定します。

4

EncryptedParts は、機密性を確保するために暗号化する必要がある SOAP メッセージの一部を指定します。

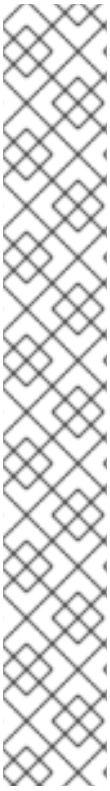
このポリシーは、@org.apache.cxf.annotations.Policy アノテーションを使用して、Service Endpoint Interface (SEI) EncryptSignPolicyHelloService に設定します。

EncryptSignPolicyHelloService.java

```
@WebService(serviceName = "EncryptSignPolicyHelloService")
@Policy(placement = Policy.Placement.BINDING, uri = "encrypt-sign-policy.xml")
public interface EncryptSignPolicyHelloService extends AbstractHelloService {
...
}
```

最初の点では、SEI にポリシーを設定すると、そのポリシーを SEI から生成されたサービスと、サービスによって提供される WSDL から生成されたすべてのクライアントの両方に適用するだけで十分です。しかし、これはすべてではありません。セキュリティーキー、ユーザー名、パスワード、およびその他の信頼できる情報は、パブリックポリシーでは公開できません。

これらは設定で設定する必要があります。



注記

参照が表すものには違いがあるため、サーバーとクライアントは helloEncryptSign を参照します。

サーバー参照は、HTTP 経由でアクセス可能なサービスエンドポイントへのパスです。つまり、で開始する必要があります。

クライアント参照はラベルです。これは、クライアントオプションをグループ化し、@CXFClient ("helloEncryptSign") で注入するためにクライアントを指定するために使用されます。

クライアントのラベルは、完全に異なるか(fooなど)、/ で開始できます。サービスに関連付けるには、正しい client-endpoint-url 値を指定することが重要です。

最初にサービスに対して行います。

application.properties

```
# A service with encrypt-sign-policy.xml set
quarkus.cxf.endpoint."/helloEncryptSign".implementor =
io.quarkiverse.cxf.it.security.policy.EncryptSignPolicyHelloServiceImpl
# can be jks or pkcs12 - set from Maven profiles in this test
keystore.type = ${keystore.type}
# Signature settings
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.username = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.password = password
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.securi
ty.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.securi
ty.crypto.merlin.keystore.type" = ${keystore.type}
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.securi
ty.crypto.merlin.keystore.password" = password
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.securi
ty.crypto.merlin.keystore.alias" = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.securi
ty.crypto.merlin.file" = bob.${keystore.type}
# Encryption settings
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.username = alice
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.secu
rity.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.secu
rity.crypto.merlin.keystore.type" = ${keystore.type}
```

```
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.secu
rity.crypto.merlin.keystore.password" = password
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.secu
rity.crypto.merlin.keystore.alias" = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.secu
rity.crypto.merlin.file" = bob.${keystore.type}
```

クライアント側で同様の設定が必要です。

application.properties

```
# A client with encrypt-sign-policy.xml set
quarkus.cxf.client.helloEncryptSign.client-endpoint-url = https://localhost:${quarkus.http.test-
ssl-port}/services/helloEncryptSign
quarkus.cxf.client.helloEncryptSign.service-interface =
io.quarkiverse.cxf.it.security.policy.EncryptSignPolicyHelloService
quarkus.cxf.client.helloEncryptSign.features = #messageCollector
# The client-endpoint-url above is HTTPS, so we have to setup the server's SSL certificates
quarkus.cxf.client.helloEncryptSign.trust-store = client-truststore.${keystore.type}
quarkus.cxf.client.helloEncryptSign.trust-store-password = password
# Signature settings
quarkus.cxf.client.helloEncryptSign.security.signature.username = alice
quarkus.cxf.client.helloEncryptSign.security.signature.password = password
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.cry
pto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.cry
pto.merlin.keystore.type" = pkcs12
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.cry
pto.merlin.keystore.password" = password
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.cry
pto.merlin.keystore.alias" = alice
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.cry
pto.merlin.file" = alice.${keystore.type}
# Encryption settings
quarkus.cxf.client.helloEncryptSign.security.encryption.username = bob
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.cr
ypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.cr
ypto.merlin.keystore.type" = pkcs12
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.cr
ypto.merlin.keystore.password" = password
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.cr
ypto.merlin.keystore.alias" = alice
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.cr
ypto.merlin.file" = alice.${keystore.type}
```

メッセージのフローを検証するには、以下のように EncryptSignPolicyTest を実行します。

```
# Clone the repository
$ git clone https://github.com/quarkiverse/quarkus-cxf.git -o upstream
$ cd quarkus-cxf
# Build the whole source tree
$ mvn clean install -DskipTests -Dquarkus.build.skip
# Run the test
$ cd integration-tests/ws-security-policy
$ mvn clean test -Dtest=EncryptSignPolicyTest
```

Signature 要素と暗号化されたボディーを含む一部のメッセージがコンソール出力に表示されます。

6.4.4. 設定

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.username	string	
ユーザーの名前。以下のように使用されます。 • WS-Security の UsernameToken の名前として • **signature.username** が設定されていない場合は、署名のためにユーザーの証明書と秘密鍵を取得するためのキーストアのエイリアス名 • **encryption.username** が設定されていない場合は、暗号化用にユーザーの公開鍵を取得するためのキーストアのエイリアス名 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.password	string	

設定プロパティー	タイプ	デフォルト
callback-handler が定義されていない場合のユーザーのパスワード。これは WS-Security UsernameToken のパスワードにのみ使用されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_PASSWORD** **Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.username	**string**	
署名用のユーザーの名前。これは、キーストアのエイリアス名として使用され、署名用にユーザーの証明書および秘密鍵を取得します。これが定義されていない場合は、代わりに **username** が使用されます。これも指定されない場合は、**signature.properties** によって参照されるプロパティーファイルに設定されたデフォルトのエイリアスが使用されます。これが設定されておらず、キーストアに単一のキーのみが含まれる場合は、そのキーが使用されます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.password	**string**	
callback-handler が定義されていない場合の署名用のユーザーのパスワード。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_PASSWORD**		
quarkus.cxf.client."clients".security.encryption.username	**string**	
暗号化用のユーザーの名前。これは、キーストアのエイリアス名として使用され、暗号化用にユーザーの公開鍵を取得します。これが定義されていない場合は、代わりに **username** が使用されます。これも指定されていない場合、**encrypt.properties** によって参照されるプロパティーファイルに設定されたデフォルトのエイリアスが使用されます。これが設定されておらず、キーストアに単一のキーのみが含まれる場合は、そのキーが使用されます。 WS-Security Web サービスプロバイダーの場合、**useReqSigCert** 値を使用して、公開鍵がサービスのトラストストアにあるクライアント(**encrypt.properties**で定義)を許可（暗号化）できます。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.callback-handler	**string**	
アウトバウンドリクエストと受信リクエストの両方のパスワードを取得するために使用される **javax.security.auth.callback.CallbackHandler** Bean への 参照。 **環境変数**:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CALLBACK_HANDLER** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.client."clients".security.saml-callback-handler	**string**	

設定プロパティー	タイプ	デフォルト
SAML Assertions を構築するために使用される **javax.security.auth.callback.CallbackHandler** 実装への参照。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_CALLBACK_HANDLER** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.client."clients".security.signature.properties	**Map<String,String>**	
signature.crypto が設定されていない場合、署名に使用する Crypto プロパティー設定。 例 [prefix].signature.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].signature.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_PROPERTIES**		
quarkus.cxf.client."clients".security.encryption.properties	**Map<String,String>**	
encryption.crypto が設定されていない場合、暗号化に使用する Crypto プロパティー設定。 例 [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_PROPERTIES** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.crypto	**string**	
署名に使用される **org.apache.wss4j.common.crypto.Crypto** Bean への参照。設定されていない場合、**signature.properties** を使用して **Crypto** インスタンスを設定します。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_SIGNATURE_CRYPTO**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.encryption.crypto	string	

暗号化に使用される **org.apache.wss4j.common.crypto.Crypto** への参照。設定されていない場合、**encryption.properties** を使用して **Crypto** インスタンスを設定します。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_CRYPTO
Quarkus CXF: 2.5.0

quarkus.cxf.client."clients".security.encryption.certificate	string	

暗号化に使用される準備済み X509 証明書のメッセージプロパティー。これが定義されていない場合、証明書はキーストア **encryption.properties** から読み込むか、要求から抽出されます(WS-Security が使用され、**encryption.username** の値が **useReqSigCert** である場合)。

このオプションはまだテストで対応されていない ため、実験的なものです。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_CERTIFICATE
Quarkus CXF: 2.5.0 以降

quarkus.cxf.client."clients".security.enable-revocation	boolean	false

true の場合、証明書の信頼を検証するときに証明書失効リスト(CRL)チェックが有効になります。それ以外の場合は有効ではありません。

このオプションはまだテストで対応されていない ため、実験的なものです。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_REVOCATION
以降、 Quarkus CXF: 2.5.0

quarkus.cxf.client."clients".security.enable-unsigned-saml-assertion-principal	boolean	false

true の場合、署名されていない SAML アサーションは SecurityContext プリンシパルとして許可されます。それ以外の場合は、SecurityContext Principals としては許可されません。

署名

ラベルが署名されていない場合は、内部署名を参照します。(sender-vouches 要件に従って)トークンが外部署名によって署名されている場合でも、トークンを使用してセキュリティーコンテキストを設定する場合は、このブール値を設定する必要があります。

このオプションはまだテストで対応されていない ため、実験的なものです。

環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_ENABLE_UNSIGNED_SAML_ASSERTION_PRINCIPAL
Quarkus CXF : 2.5.0 以降

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.validate-saml-subject-confirmation	boolean	true
true の場合、受信した SAML トークンの**SubjectConfirmation** 要件(sender-vouches または holder-of-key)は検証されます。それ以外の場合は検証されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_VALIDATE_SAML_SUBJECT_CONFIRMATION**		
quarkus.cxf.client."clients".security.sc-from-jaas-subject	boolean	true
true の場合、セキュリティーコンテキストは JAAS Subject から作成できます。それ以外の場合は、JAAS Subject から作成しないでください。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_SC_FROM_JAAS_SUBJECT** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.audience-restriction-validation	boolean	true
true の場合、SAML トークンに Audience Restriction URI が含まれている場合、その 1 つが **audience.restrictions** の値のいずれかと一致する必要があります。一致しない場合は、SAML AudienceRestriction 検証が無効になります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_AUDIENCE_RESTRICTION_VALIDATION** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.saml-role-attributename	string	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role

設定プロパティー	タイプ	デフォルト
ロール情報が保存される SAML **AttributeStatement** の属性 URI。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_ROLE_ATTRIBUTENAME** **(Quarkus CXF: 2.5.0 以降)**		
quarkus.cxf.client."clients".security.subject-cert-constraints	**string**	
証明書に関連付けられた証明書チェーンの信頼検証後に、署名の検証に使用される証明書のサブジェクト DN に適用される正規表現の文字列(**security.cert.constraints.separator**)。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SUBJECT_CERT_CONSTRAINTS** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.cert-constraints-separator	**string**	**,**
security.subject.cert.constraintsに設定された証明書制約を解析するために使用される区切り文字 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CERT_CONSTRAINTS_SEPARATOR**		
quarkus.cxf.client."clients".security.actor	**string**	
wsse:Security ヘッダーのアクターまたはロール名。このパラメーターを省略すると、アクター名は設定されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ACTOR** **Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.validate.token	**boolean**	**true**
true の場合、受信した **UsernameToken** のパスワードが検証されます。そうでない場合は検証されません。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_VALIDATE_TOKEN** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.username-token.always.encrypted	**boolean**	**true**

設定プロパティー	タイプ	デフォルト
SupportingToken として定義された **UsernameToken** を常に暗号化するかどうか。これは、ネットワーク上でパスワード（またはパスワードのダイジェスト）を公開するため、実稼働環境で **false** に設定しないでください。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAME_TOKEN_ALWAYS_ENCRYPTED** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.is-bsp-compliant	boolean	true
true の場合、Basic Security Profile (BSP) 1.1 への準拠が保証されます。そうでない場合は保証されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_IS_BSP_COMPLIANT**		
quarkus.cxf.client."clients".security.enable.nonce.cache	boolean	
true の場合、**UsernameToken** nonces はメッセージイニシエーターと受信者の両方にキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。 **キャッシュ** キャッシングは、**UsernameToken** WS-SecurityPolicy が有効な場合、または **UsernameToken** アクションが非セキュリティーポリシーケースに対して設定されている場合にのみ適用されます。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITYS__SECURITY_ENABLE_NONCE_CACHE** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable.timestamp.cache	boolean	

設定プロパティー	タイプ	デフォルト

true の場合、**タイムスタンプ Created** Strings （メッセージ署名と併せてキャッシュされる）は、メッセージイニシエーターと受信者の両方に対してキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。

キャッシュ

キャッシングは、**IncludeTimestamp** ポリシーが有効な場合、またはセキュリティーポリシー以外のケースに対して **Timestamp** アクションが設定されている場合にのみ適用されます。

このオプションはまだ テストで対応されていない ため、実験的なものです。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_TIMESTAMP_CACHE
以降、 Quarkus CXF: 2.5.0

quarkus.cxf.client."clients".security.enable.streaming	**boolean**	**false**

true の場合、WS-Security の新しいストリーミング(StAX)実装が使用されます。それ以外の場合は、古い DOM 実装が使用されます。

環境変数:QUARKUS_CXF_CLIENT_CLIENTS__SECURITY_ENABLE_STREAMING
以降、 Quarkus CXF: 2.5.0

quarkus.cxf.client."clients".security.return.security.error	**boolean**	**false**

true の場合、詳細なセキュリティーエラーメッセージがクライアントに送信されます。それ以外の場合は、詳細が省略され、一般的なエラーメッセージのみが送信されます。

実際のセキュリティーエラーは、デプロイメントに関する情報が漏洩する場合があるか、攻撃のために Oracle を提供する可能性があるため、実稼働環境ではクライアントに返すことはできません。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_RETURN_SECURITY_ERROR
。Quarkus CXF: 2.5.0

quarkus.cxf.client."clients".security.must-understand	**boolean**	**true**

true の場合、SOAP **mustUnderstand** ヘッダーは WS-SecurityPolicy に基づいてセキュリティーヘッダーに含まれます。それ以外の場合、ヘッダーは常に省略されます。

enable.streaming = true でのみ機能します - CXF-8940を参照してください。

環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_MUST_UNDERSTAND
。Quarkus CXF: 2.5.0

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.enable.saml.cache	boolean	
true で、トークンに **OneTimeUse** Condition が含まれる場合、SAML2 トークン識別子はメッセージイニシエーターと受信者の両方にキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。 キャッシングは、**SamlToken** ポリシーが有効な場合や、非セキュリティーポリシーケースに対して SAML アクションが設定されている場合にのみ適用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_SAML_CACHE** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.client."clients".security.store.bytes.in.attachment	boolean	
バイト(CipherData または BinarySecurityToken)を添付ファイルに保存するかどうか。MTOM が有効な場合は、デフォルトは true になります。これを BASE-64 でエンコードし、それらをメッセージでインライン化する代わりに、false に設定します。これを true に設定すると、BASE-64 エンコーディングステップをスキップできるため、より効率的です。これは、DOM WS-Security スタックにのみ適用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_STORE_BYTES_IN_ATTACHMENT**		
quarkus.cxf.client."clients".security.swa.encryption.attachment.transform.content	boolean	false
true の場合、Attachment が WS-SecurityPolicy 式を介して暗号化されている場合に**Attachment-Content-Only** 変換が使用されます。それ以外の場合は、Attachment が WS-SecurityPolicy 式を介して暗号化されるときに **Attachment-Complete** 変換が使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SWA_ENCRYPTION_ATTACHMENT_TRANSFORM_CONTENT** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.client."clients".security.use.str.transform	boolean	true

設定プロパティー	タイプ	デフォルト
true の場合、STR (Security Token Reference) Transform は、（外部）SAML トークンの署名時に使用されます。それ以外の場合は、STR （セキュリティートークン参照）変換は使用されません。 一部のフレームワークは、**SecurityTokenReference** を処理できません。このような場合には、これを **false** に設定できます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USE_STR_TRANSFORM** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.add.inclusive.prefixes	**boolean**	**true**
true の場合、**WSConstants.C14N_EXCL_OMIT_COMMENTS** を使用して署名を生成するときに、**InclusiveNamespaces PrefixList** が **CanonicalizationMethod** の子として追加されます。それ以外の場合は、**PrefixList** は追加されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_ADD_INCLUSIVE_PREFIXES** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.disable.require.client.cert.check	**boolean**	**false**
true の場合、WS-SecurityPolicy **RequireClientCertificate** ポリシーの適用が無効になります。それ以外の場合は、WS-SecurityPolicy **RequireClientCertificate** ポリシーの適用が有効になります。 一部のサーバーでは、SSL ハンドシェイクの開始時にクライアント証明書の検証が行われない場合があるため、ポリシー検証のために WS-Security レイヤーではクライアント証明書を使用できない場合があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_DISABLE_REQUIRE_CLIENT_CERT_CHECK**		
quarkus.cxf.client."clients".security.expand.xop.include	**boolean**	
true の場合、**xop:Include** 要素で暗号化および署名が（アウトバウンド側で）、または（インバウンド側での）署名の検証が検索されます。そうでない場合は検索は行われません。これにより、参照だけでなく、実際のバイトが確実に署名されるようになります。MTOM が有効な場合はデフォルトで **true** になり、それ以外の場合のデフォルトは **false** になります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_EXPAND_XOP_INCLUDE**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.timestamp.timeToLive	string	300
受信 **タイムスタンプ** の作成値に追加して、有効とみなされるかどうかを決定する時間（秒単位）。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TIMESTAMP_TIMETOLIVE** : 2.5.0		
quarkus.cxf.client."clients".security.timestamp.futureTimeToLive	string	60
入力 **タイムスタンプ** の **Created** 時間が有効である将来の時間（秒単位）。クロックがわずかにアスケートされる場合の問題を回避するために、デフォルトはゼロより大きいです。将来のすべてのタイムスタンプを拒否するには、これを **0** に設定します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITYS__SECURITY_TIMESTAMP_FUTURETIMETOLIVE**		
quarkus.cxf.client."clients".security.usernametoken.timeToLive	string	300
有効かどうかを決定するために、受信 **UsernameToken** の作成値に追加する時間（秒単位）。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAMETOKEN_TIMETOLIVE**		
quarkus.cxf.client."clients".security.usernametoken.futureTimeToLive	string	60
入力 **UsernameToken** の **Created** 時間が有効な将来の時間（秒単位）。クロックがわずかにアスケートされる場合の問題を回避するために、デフォルトはゼロより大きいです。将来のすべての UsernameToken を拒否するには、これを **0** に設定します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_USERNAMETOKEN_FUTURETIMETOLIVE**		
quarkus.cxf.client."clients".security.spnego.client.action	string	

設定プロパティー	タイプ	デフォルト
SPNEGO に使用する **org.apache.wss4j.common.spnego.SpnegoClientAction** Bean への参照。これにより、ユーザーは別の実装をプラグインして、サービスチケットを取得できます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SPNEGO_CLIENT_ACTION** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.nonce.cache.instance	**string**	
UsernameToken nonces のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への参照。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_NONCE_CACHE_INSTANCE** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.timestamp.cache.instance	**string**	
Timestamp Created Strings のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への参照。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TIMESTAMP_CACHE_INSTANCE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.saml.cache.instance	**string**	
SAML2 トークン識別子文字列のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への参照 （トークンに **OneTimeUse** 条件が含まれる場合）。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_CACHE_INSTANCE** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.client."clients".security.cache.config.file	**string**	

設定プロパティー	タイプ	デフォルト
このプロパティーを、**TokenStore** の基礎となるキャッシュ実装の設定ファイルを参照するように設定します。使用されるデフォルトの設定ファイルは、**org.apache.cxf:cxf-rt-security** JAR の **cxf-ehcache.xml** です。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CACHE_CONFIG_FILE** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.client."clients".security.token-store-cache-instance	**string**	
セキュリティートークンのキャッシュに使用する **org.apache.cxf.ws.security.tokenstore.TokenStore** Bean への 参照。デフォルトでは、これはインスタンスを使用します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TOKEN_STORE_CACHE_INSTANCE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.cache.identifier	**string**	
TokenStore で使用するキャッシュ識別子。CXF は次のキーを使用してトークンストアを取得します：**org.apache.cxf.ws.security.tokenstore.TokenStore-<identifier>**;このキーを使用して、サービス固有のキャッシュ設定を設定できます。識別子が一致しない場合は、キー **org.apache.cxf.ws.security.tokenstore.TokenStore** を持つキャッシュ設定にフォールバックします。 デフォルトの &**lt;identifier** > は、問題のサービスの QName です。ただし、カスタムキャッシュ設定（たとえば、クライアントごとに TokenStore を指定する場合は）を選択するには、代わりにこの識別子を使用して設定できます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CACHE_IDENTIFIER**		
quarkus.cxf.client."clients".security.role.classifier	**string**	
使用するサブジェクトのロールの分類子。WSS4J Validators の 1 つが Validation から JAAS Subject を返す場合、**WSS4JInInterceptor** はこのサブジェクトに基づいて **SecurityContext** の作成を試みます。この値が指定されていない場合、**org.apache.cxf:cxf-core** で **DefaultSecurityContext** を使用してロールを取得しようとします。それ以外の場合は、この値を **role.classifier.type** と組み合わせて使用し、**Subject** からロールを取得します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ROLE_CLASSIFIER** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.role.classifier.type	**string**	**prefix**

設定プロパティー	タイプ	デフォルト
使用するサブジェクトのロールの分類子タイプ。WSS4J Validators の 1 つが Validation から JAAS Subject を返す場合、**WSS4JInInterceptor** はこのサブジェクトに基づいて **SecurityContext** の作成を試みます。現在許可される値は **prefix** または **classname** です。**role.classifier** と組み合わせて使用する必要があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_ROLE_CLASSIFIER_TYPE**		
quarkus.cxf.client."clients".security.asymmetric.signature.algorithm	**string**	
この設定タグにより、WS-SecurityPolicy 仕様では他のアルゴリズムの使用が許可されないため、ユーザーは WS-SecurityPolicy で使用するためにデフォルトの非対称署名アルゴリズム(RSA-SHA1)を上書きできます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ASYMMETRIC_SIGNATURE_ALGORITHM** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.client."clients".security.symmetric.signature.algorithm	**string**	
この設定タグにより、WS-SecurityPolicy 仕様では他のアルゴリズムの使用が許可されないため、ユーザーは WS-SecurityPolicy で使用するためにデフォルトの対称署名アルゴリズム(HMAC-SHA1)を上書きできます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_SYMMETRIC_SIGNATURE_ALGORITHM**		
quarkus.cxf.client."clients".security.password.encryptor.instance	**string**	
Merlin Crypto 実装（またはカスタム Crypto 実装）でパスワードを暗号化または復号化するために使用される **org.apache.wss4j.common.crypto.PasswordEncryptor** Bean への 参照。 デフォルトでは、WSS4J は **org.apache.wss4j.common.crypto.JasyptPasswordEncryptor** を使用します。これは、Merlin Crypto 定義でキーストアパスワードを復号化するために使用するパスワードでインスタンス化する必要があります。このパスワードは、**callback-handler**を介して定義された CallbackHandler を介して取得されます。 暗号化されたパスワードは、ENC (encoded encrypted password)の形式で保存する必要があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_PASSWORD_ENCRYPTOR_INSTANCE** **以降、 Quarkus CXF**: 2.5.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.delegated.credential	string	
WS-Security に使用する Kerberos **org.ietf.jgss.GSSCredential** Bean への参照。これは、クライアント認証情報を使用する代わりにサービスチケットを取得するために使用されます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_DELEGATED_CREDENTIAL** **。Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.security.context.creator	string	
WSS4J 処理結果のセットから CXF SecurityContext を作成するために使用される **org.apache.cxf.ws.security.wss4j.WSS4JSecurityContextCreator** Bean への参照。デフォルトの実装は **org.apache.cxf.ws.security.wss4j.DefaultWSS4JSecurityContextCreator** です。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT_CLIENT__SECURITY_SECURITY_CONTEXT_CREATOR**		
quarkus.cxf.client."clients".security.security.token.lifetime	long	300000
セキュリティートークンの有効期間値（ミリ秒単位）。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SECURITY_TOKEN_LIFETIME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.request.credential.delegation	boolean	false
true の場合、認証情報の委譲は KerberosClient で要求されます。それ以外の場合は、認証情報の委譲は KerberosClient にされません。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_REQUEST_CREDENTIAL_DELEGATION** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.use.credential.delegation	boolean	false

設定プロパティー	タイプ	デフォルト
true の場合、GSSCredential Bean は**delegated.credential** プロパティーを使用してメッセージコンテキストから取得され、サービスチケットの取得に使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_USE_CREDENTIAL_DELEGATION** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.is.username.in.servicename.form	**boolean**	**false**
true の場合、Kerberos ユーザー名は servicename 形式になります。それ以外の場合、Kerberos ユーザー名は servicename 形式ではありません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_IS_USERNAME_IN_SERVICENAME_FORM**		
quarkus.cxf.client."clients".security.kerberos.jaas.context	**string**	
Kerberos に使用する JAAS コンテキスト名。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_JAAS_CONTEXT**		
quarkus.cxf.client."clients".security.kerberos.spn	**string**	
使用する Kerberos サービスプロバイダー名(spn)。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_SPN** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.client."clients".security.kerberos.client	**string**	
サービスチケットの取得に使用される **org.apache.cxf.ws.security.kerberos.KerberosClient** Bean への参照。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_CLIENT** **Quarkus CXF : 2.5.0 以降**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.client."clients".security.sts.client	string	

STS と通信する、完全に設定された **org.apache.cxf.ws.security.trust.STSClient** Bean への参照。設定されていない場合、STS クライアントは他の **[prefix].security.sts.client.*** プロパティーに基づいて作成され、設定されます。

org.apache.cxf.ws.security.trust.STSClient には no-args コンストラクターがないため、CDI Bean タイプとして使用できないという事実を回避するには、ラッパークラス **io.quarkiverse.cxf.ws.security.sts.client.STSClientBean** を代わりに使用できます。

ヒント

WS-Trust の詳細は、Security Token Service (STS) 拡張ページを確認してください。

環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT**
Quarkus CXF 以降: 3.8.0

quarkus.cxf.client."clients".security.sts.client.wsdl	string	

STS クライアントのサービスプロキシーの生成時に使用する WSDL ドキュメントを参照する URL、リソースパス、またはローカルファイルシステムパス。

環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_WSDL**
Quarkus CXF: 3.8.0 以降

quarkus.cxf.client."clients".security.sts.client.service-name	string	

STS サービスの完全修飾名。一般的な値は以下のとおりです。

- WS-Trust 1.0: **{http://schemas.xmlsoap.org/ws/2005/02/trust/}SecurityTokenService**
- WS-Trust 1.3: **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}SecurityTokenService**
- WS-Trust 1.4: **{http://docs.oasis-open.org/ws-sx/ws-trust/200802/}SecurityTokenService**

環境変数:**QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_SERVICE_NAME**
Quarkus CXF: 3.8.0

quarkus.cxf.client."clients".security.sts.client.endpoint-name	string	

設定プロパティー	タイプ	デフォルト
STS エンドポイント名の完全修飾名。一般的な値は以下のとおりです。 • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}X509_Port** • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}Transport_Port** • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}UT_Port** **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENDPOINT_NAME** **Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.username	**string**	
STS に対して認証する際に使用するユーザー名。以下のように使用されます。 • WS-Security の UsernameToken の名前として • **signature.username** が設定されていない場合は、署名のためにユーザーの証明書と秘密鍵を取得するためのキーストアのエイリアス名 • **encryption.username** が設定されていない場合は、暗号化用にユーザーの公開鍵を取得するためのキーストアのエイリアス名 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_USERNAME** **以降、 Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.password	**string**	
ユーザー名に関連付けられた **パスワード**。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_PASSWORD** **Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.username	**string**	
暗号化用のユーザーの名前。これは、キーストアのエイリアス名として使用され、暗号化用にユーザーの公開鍵を取得します。これが定義されていない場合は、代わりに **username** が使用されます。これも指定されていない場合、**encrypt.properties** によって参照されるプロパティーファイルに設定されたデフォルトのエイリアスが使用されます。これが設定されておらず、キーストアに単一のキーのみが含まれる場合は、そのキーが使用されます。 WS-Security Web サービスプロバイダーの場合、**useReqSigCert** 値を使用して、公開鍵がサービスのトラストストアにあるクライアント(**encrypt.properties**で定義)を許可（暗号化）できます。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_USERNAME** **以降、 Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.properties	**Map<String,String>**	

設定プロパティー	タイプ	デフォルト
encryption.crypto が設定されていない場合、暗号化に使用する Crypto プロパティー設定。 例 [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_PROPERTIES** **以降、 Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.crypto	**string**	
暗号化に使用される **org.apache.wss4j.common.crypto.Crypto** への 参照。設定されていない場合、**encryption.properties** を使用して **Crypto** インスタンスを設定します。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_CRYPTO** **Quarkus CXF 以降**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.crypto	**string**	
STS に使用される **org.apache.wss4j.common.crypto.Crypto** への 参照。設定されていない場合、**token.properties** を使用して **Crypto** インスタンスを設定します。 WCF の信頼サーバーは、IN ADDITION 応答でトークンを暗号化することがあり、メッセージの完全なセキュリティーが必要です。これらのプロパティーは、STS クライアントが応答で EncryptedData 要素を復号化する方法を制御します。 これらは、KeyType が **PublicKey**の場合に使用される RSA/DSAKeyValue トークンを送信/処理するために **token.properties** によっても使用されます。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_CRYPTO** **Quarkus CXF 以降**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.properties	**Map<String,String>**	

設定プロパティー	タイプ	デフォルト
encryption.crypto が設定されていない場合、暗号化に使用する Crypto プロパティー設定。 例 [prefix].token.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].token.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].token.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_PROPERTIES** **以降、 Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.username	**string**	
PublicKey KeyType ケースの STS に送信するユーザーの公開鍵を取得するためのキーストアのエイリアス名。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_USERNAME** **以降、 Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.usecert	**boolean**	**false**
UseKey/KeyInfo で X509Certificate 構造を書き込むか、KeyValue 構造を書き込むかどうか。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_USECERT** **。Quarkus CXF 以降**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.soap12-binding	**boolean**	**false**
true の場合、STS クライアントは Soap 1.2 メッセージを送信するように設定されます。それ以外の場合は、SOAP 1.1 メッセージを送信します。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_SOAP12_BINDING** **。Quarkus CXF 以降**: 3.8.0		
quarkus.cxf.endpoint."endpoints".security.username	**string**	

設定プロパティー	タイプ	デフォルト
ユーザーの名前。以下のように使用されます。 • WS-Security の UsernameToken の名前として • **signature.username** が設定されていない場合は、署名のためにユーザーの証明書と秘密鍵を取得するためのキーストアのエイリアス名 • **encryption.username** が設定されていない場合は、暗号化用にユーザーの公開鍵を取得するためのキーストアのエイリアス名 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.password	**string**	
callback-handler が定義されていない場合のユーザーのパスワード。これは WS-Security UsernameToken のパスワードにのみ使用されます。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_PASSWORD** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.username	**string**	
署名用のユーザーの名前。これは、キーストアのエイリアス名として使用され、署名用にユーザーの証明書および秘密鍵を取得します。これが定義されていない場合は、代わりに **username** が使用されます。これも指定されない場合は、**signature.properties** によって参照されるプロパティーファイルに設定されたデフォルトのエイリアスが使用されます。これが設定されておらず、キーストアに単一のキーのみが含まれる場合は、そのキーが使用されます。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.password	**string**	
callback-handler が定義されていない場合の署名用のユーザーのパスワード。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_PASSWORD**		
quarkus.cxf.endpoint."endpoints".security.encryption.username	**string**	

設定プロパティー	タイプ	デフォルト
暗号化用のユーザーの名前。これは、キーストアのエイリアス名として使用され、暗号化用にユーザーの公開鍵を取得します。これが定義されていない場合は、代わりに **username** が使用されます。これも指定されていない場合、**encrypt.properties** によって参照されるプロパティーファイルに設定されたデフォルトのエイリアスが使用されます。これが設定されておらず、キーストアに単一のキーのみが含まれる場合は、そのキーが使用されます。 WS-Security Web サービスプロバイダーの場合、**useReqSigCert** 値を使用して、公開鍵がサービスのトラストストアにあるクライアント(**encrypt.properties**で定義)を許可（暗号化）できます。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_USERNAME** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.callback-handler	**string**	
アウトバウンドリクエストと受信リクエストの両方のパスワードを取得するために使用される **javax.security.auth.callback.CallbackHandler** Bean への参照。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CALLBACK_HANDLER** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.endpoint."endpoints".security.saml-callback-handler	**string**	
SAML Assertions を構築するために使用される **javax.security.auth.callback.CallbackHandler** 実装への参照。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_CALLBACK_HANDLER** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.endpoint."endpoints".security.signature.properties	**Map<String,String>**	
signature.crypto が設定されていない場合、署名に使用する Crypto プロパティー設定。 例 [prefix].signature.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].signature.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_PROPERTIES**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.encryption.properties	Map<String,String>	
encryption.crypto が設定されていない場合、暗号化に使用する Crypto プロパティー設定。 例 [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_PROPERTIES** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.crypto	string	
署名に使用される **org.apache.wss4j.common.crypto.Crypto** Bean への参照。設定されていない場合、**signature.properties** を使用して **Crypto** インスタンスを設定します。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_CRYPTO**		
quarkus.cxf.endpoint."endpoints".security.encryption.crypto	string	
暗号化に使用される **org.apache.wss4j.common.crypto.Crypto** への参照。設定されていない場合、**encryption.properties** を使用して **Crypto** インスタンスを設定します。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_CRYPTO** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.encryption.certificate	string	
暗号化に使用される準備済み X509 証明書のメッセージプロパティー。これが定義されていない場合、証明書はキーストア **encryption.properties** から読み込むか、要求から抽出されます(WS-Security が使用され、**encryption.username** の値が **useReqSigCert** である場合)。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_CERTIFICATE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable-revocation	boolean	false

設定プロパティー	タイプ	デフォルト
true の場合、証明書の信頼を検証するときに証明書失効リスト(CRL)チェックが有効になります。それ以外の場合は有効ではありません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_REVOCATION**		
quarkus.cxf.endpoint."endpoints".security.enable-unsigned-saml-assertion-principal	**boolean**	**false**
true の場合、署名されていない SAML アサーションは SecurityContext プリンシパルとして許可されます。それ以外の場合は、SecurityContext Principals としては許可されません。 **署名** ラベルが署名されていない場合は、内部署名を参照します。(sender-vouches 要件に従って)トークンが外部署名によって署名されている場合でも、トークンを使用してセキュリティーコンテキストを設定する場合は、このブール値を設定する必要があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_UNSIGNED_SAML_ASSERTION_PRINCIPAL** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.endpoint."endpoints".security.validate-saml-subject-confirmation	**boolean**	**true**
true の場合、受信した SAML トークンの**SubjectConfirmation** 要件(sender-vouches または holder-of-key)は検証されます。それ以外の場合は検証されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_VALIDATE_SAML_SUBJECT_CONFIRMATION**		
quarkus.cxf.endpoint."endpoints".security.sc-from-jaas-subject	**boolean**	**true**
true の場合、セキュリティーコンテキストは JAAS Subject から作成できます。それ以外の場合は、JAAS Subject から作成しないでください。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SC_FROM_JAAS_SUBJECT** **以降、 Quarkus CXF**: 2.5.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.audience-restriction-validation	boolean	true
true の場合、SAML トークンに Audience Restriction URI が含まれている場合、その 1 つが **audience.restrictions** の値のいずれかと一致する必要があります。一致しない場合は、SAML AudienceRestriction 検証が無効になります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_AUDIENCE_RESTRICTION_VALIDATION** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.saml-role-attributename	string	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role
ロール情報が保存される SAML **AttributeStatement** の属性 URI。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_ROLE_ATTRIBUTENAME** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.endpoint."endpoints".security.subject-cert-constraints	string	
証明書に関連付けられた証明書チェーンの信頼検証後に、署名の検証に使用される証明書のサブジェクト DN に適用される正規表現の文字列(**security.cert.constraints.separator**)。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SUBJECT_CERT_CONSTRAINTS** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.cert-constraints-separator	string	,

設定プロパティー	タイプ	デフォルト
security.subject.cert.constraintsに設定された証明書制約を解析するために使用される区切り文字 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CERT_CONSTRAINTS_SEPARATOR**		
quarkus.cxf.endpoint."endpoints".security.actor	string	
wsse:Security ヘッダーのアクターまたはロール名。このパラメーターを省略すると、アクター名は設定されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ACTOR** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.validate.token	boolean	true
true の場合、受信した **UsernameToken** のパスワードが検証されます。そうでない場合は検証されません。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_VALIDATE_TOKEN** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.username-token.always.encrypted	boolean	true
SupportingToken として定義された **UsernameToken** を常に暗号化するかどうか。これは、ネットワーク上でパスワード（またはパスワードのダイジェスト）を公開するため、実稼働環境で **false** に設定しないでください。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAME_TOKEN_ALWAYS_ENCRYPTED** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.is-bsp-compliant	boolean	true

設定プロパティー	タイプ	デフォルト
true の場合、Basic Security Profile (BSP) 1.1 への準拠が保証されます。そうでない場合は保証されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_IS_BSP_COMPLIANT**		
quarkus.cxf.endpoint."endpoints".security.enable.nonce.cache	**boolean**	
true の場合、**UsernameToken** nonces はメッセージイニシエーターと受信者の両方にキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。 **キャッシュ** キャッシングは、**UsernameToken** WS-SecurityPolicy が有効な場合、または **UsernameToken** アクションが非セキュリティーポリシーケースに対して設定されている場合にのみ適用されます。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_NONCE_CACHE** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.endpoint."endpoints".security.enable.timestamp.cache	**boolean**	
true の場合、**タイムスタンプ Created** Strings （メッセージ署名と併せてキャッシュされる）は、メッセージイニシエーターと受信者の両方に対してキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。 **キャッシュ** キャッシングは、**IncludeTimestamp** ポリシーが有効な場合、またはセキュリティーポリシー以外のケースに対して **Timestamp** アクションが設定されている場合にのみ適用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_TIMESTAMP_CACHE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable.streaming	**boolean**	**false**

設定プロパティー	タイプ	デフォルト
true の場合、WS-Security の新しいストリーミング(StAX)実装が使用されます。それ以外の場合は、古い DOM 実装が使用されます。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_STREAMING** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.return.security.error	**boolean**	**false**
true の場合、詳細なセキュリティーエラーメッセージがクライアントに送信されます。それ以外の場合は、詳細が省略され、一般的なエラーメッセージのみが送信されます。 実際のセキュリティーエラーは、デプロイメントに関する情報が漏洩する場合があるか、攻撃のために Oracle を提供する可能性があるため、実稼働環境ではクライアントに返すことはできません。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_RETURN_SECURITY_ERROR** 。**Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.must-understand	**boolean**	**true**
true の場合、SOAP **mustUnderstand** ヘッダーは WS-SecurityPolicy に基づいてセキュリティーヘッダーに含まれます。それ以外の場合、ヘッダーは常に省略されます。 **enable.streaming = true** でのみ機能します - CXF-8940を参照してください。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_MUST_UNDERSTAND** 。**Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable.saml.cache	**boolean**	
true で、トークンに **OneTimeUse** Condition が含まれる場合、SAML2 トークン識別子はメッセージイニシエーターと受信者の両方にキャッシュされます。それ以外の場合は、メッセージイニシエーターも受信者もキャッシュされません。デフォルトは、メッセージの受信者の場合は **true**、メッセージイニシエーターの場合は **false** です。 キャッシングは、**SamlToken** ポリシーが有効な場合や、非セキュリティーポリシーケースに対して SAML アクションが設定されている場合にのみ適用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_SAML_CACHE** (**Quarkus CXF: 2.5.0 以降**)		
quarkus.cxf.endpoint."endpoints".security.store.bytes.in.attachment	**boolean**	

設定プロパティー	タイプ	デフォルト
バイト(CipherData または BinarySecurityToken)を添付ファイルに保存するかどうか。MTOM が有効な場合は、デフォルトは true になります。これを BASE-64 でエンコードし、それらをメッセージでインライン化する代わりに、false に設定します。これを true に設定すると、BASE-64 エンコーディングステップをスキップできるため、より効率的です。これは、DOM WS-Security スタックにのみ適用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_STORE_BYTES_IN_ATTACHMENT**		
quarkus.cxf.endpoint."endpoints".security.swa.encryption.attachment.transform.content	**boolean**	**false**
true の場合、Attachment が WS-SecurityPolicy 式を介して暗号化されている場合に**Attachment-Content-Only** 変換が使用されます。それ以外の場合は、Attachment が WS-SecurityPolicy 式を介して暗号化されるときに **Attachment-Complete** 変換が使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SWA_ENCRYPTION_ATTACHMENT_TRANSFORM_CONTENT**		
quarkus.cxf.endpoint."endpoints".security.use.str.transform	**boolean**	**true**
true の場合、STR (Security Token Reference) Transform は、（外部）SAML トークンの署名時に使用されます。それ以外の場合は、STR （セキュリティートークン参照）変換は使用されません。 一部のフレームワークは、**SecurityTokenReference** を処理できません。このような場合には、これを **false** に設定できます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USE_STR_TRANSFORM** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.add.inclusive.prefixes	**boolean**	**true**
true の場合、**WSConstants.C14N_EXCL_OMIT_COMMENTS** を使用して署名を生成するときに、**InclusiveNamespaces PrefixList** が **CanonicalizationMethod** の子として追加されます。それ以外の場合は、**PrefixList** は追加されません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ADD_INCLUSIVE_PREFIXES** **以降、Quarkus CXF**: 2.5.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.disable.require.client.cert.check	boolean	false
true の場合、WS-SecurityPolicy **RequireClientCertificate** ポリシーの適用が無効になります。それ以外の場合は、WS-SecurityPolicy **RequireClientCertificate** ポリシーの適用が有効になります。 一部のサーバーでは、SSL ハンドシェイクの開始時にクライアント証明書の検証が行われない場合があるため、ポリシー検証のために WS-Security レイヤーではクライアント証明書を使用できない場合があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_DISABLE_REQUIRE_CLIENT_CERT_CHECK**		
quarkus.cxf.endpoint."endpoints".security.expand.xop.include	boolean	
true の場合、**xop:Include** 要素で暗号化および署名が（アウトバウンド側で）、または（インバウンド側での）署名の検証が検索されます。そうでない場合は検索は行われません。これにより、参照だけでなく、実際のバイトが確実に署名されるようになります。MTOM が有効な場合はデフォルトで **true** になり、それ以外の場合のデフォルトは **false** になります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_EXPAND_XOP_INCLUDE**		
quarkus.cxf.endpoint."endpoints".security.timestamp.timeToLive	string	300
受信 **タイムスタンプ** の作成値に追加して、有効とみなされるかどうかを決定する時間（秒単位）。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_TIMETOLIVE** **:** 2.5.0		
quarkus.cxf.endpoint."endpoints".security.timestamp.futureTimeToLive	string	60
入力 **タイムスタンプ** の **Created** 時間が有効である将来の時間（秒単位）。クロックがわずかにアスケートされる場合の問題を回避するために、デフォルトはゼロより大きいです。将来のすべてのタイムスタンプを拒否するには、これを **0** に設定します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_FUTURETIMETOLIVE** **CXF : 2.5.0 以降**		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.usernametoken.timeToLive	string	300
有効かどうかを決定するために、受信 **UsernameToken** の作成値に追加する時間（秒単位）。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAMETOKEN_TIMETOLIVE**		
quarkus.cxf.endpoint."endpoints".security.usernametoken.futureTimeToLive	string	60
入力 **UsernameToken** の **Created** 時間が有効な将来の時間（秒単位）。クロックがわずかにアスケートされる場合の問題を回避するために、デフォルトはゼロより大きいです。将来のすべての UsernameToken を拒否するには、これを **0** に設定します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAMETOKEN_FUTURETIMETOLIVE**		
quarkus.cxf.endpoint."endpoints".security.spnego.client.action	string	
SPNEGO に使用する **org.apache.wss4j.common.spnego.SpnegoClientAction** Bean への 参照。これにより、ユーザーは別の実装をプラグインして、サービスチケットを取得できます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SPNEGO_CLIENT_ACTION** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.nonce.cache.instance	string	
UsernameToken nonces のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への 参照。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CACHE_CACHE_INSTANCE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.timestamp.cache.instance	string	

設定プロパティー	タイプ	デフォルト
Timestamp Created Strings のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への 参照。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_CACHE_INSTANCE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.saml.cache.instance	**string**	
SAML2 トークン識別子文字列のキャッシュに使用される **org.apache.wss4j.common.cache.ReplayCache** Bean への 参照 （トークンに **OneTimeUse** 条件が含まれる場合）。デフォルトで **org.apache.wss4j.common.cache.EHCacheReplayCache** インスタンスが使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_CACHE_INSTANCE** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.endpoint."endpoints".security.cache.config.file	**string**	
このプロパティーを、**TokenStore** の基礎となるキャッシュ実装の設定ファイルを参照するように設定します。使用されるデフォルトの設定ファイルは、**org.apache.cxf:cxf-rt-security** JAR の **cxf-ehcache.xml** です。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CACHE_CONFIG_FILE** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.token-store-cache-instance	**string**	
セキュリティートークンのキャッシュに使用する **org.apache.cxf.ws.security.tokenstore.TokenStore** Bean への 参照。デフォルトでは、これはインスタンスを使用します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TOKEN_STORE_CACHE_INSTANCE** **Quarkus CXF : 2.5.0 以降**		
quarkus.cxf.endpoint."endpoints".security.cache.identifier	**string**	

設定プロパティー	タイプ	デフォルト
TokenStore で使用するキャッシュ識別子。CXF は次のキーを使用してトークンストアを取得します：**org.apache.cxf.ws.security.tokenstore.TokenStore-<identifier>**;このキーを使用して、サービス固有のキャッシュ設定を設定できます。識別子が一致しない場合は、キー **org.apache.cxf.ws.security.tokenstore.TokenStore** を持つキャッシュ設定にフォールバックします。 デフォルトの &**lt;identifier** > は、問題のサービスの QName です。ただし、カスタムキャッシュ設定（たとえば、クライアントごとに TokenStore を指定する場合は）を選択するには、代わりにこの識別子を使用して設定できます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CACHE_IDENTIFIER**		
quarkus.cxf.endpoint."endpoints".security.role.classifier	**string**	
使用するサブジェクトのロールの分類子。WSS4J Validators の 1 つが Validation から JAAS Subject を返す場合、**WSS4JInInterceptor** はこのサブジェクトに基づいて **SecurityContext** の作成を試みます。この値が指定されていない場合、**org.apache.cxf:cxf-core** で **DefaultSecurityContext** を使用してロールを取得しようとします。それ以外の場合は、この値を **role.classifier.type** と組み合わせて使用し、**Subject** からロールを取得します。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ROLE_CLASSIFIER** **Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.role.classifier.type	**string**	**prefix**
使用するサブジェクトのロールの分類子タイプ。WSS4J Validators の 1 つが Validation から JAAS Subject を返す場合、**WSS4JInInterceptor** はこのサブジェクトに基づいて **SecurityContext** の作成を試みます。現在許可される値は **prefix** または **classname** です。**role.classifier** と組み合わせて使用する必要があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ROLE_CLASSIFIER_TYPE**		
quarkus.cxf.endpoint."endpoints".security.asymmetric.signature.algorithm	**string**	

設定プロパティー	タイプ	デフォルト
この設定タグにより、WS-SecurityPolicy 仕様では他のアルゴリズムの使用が許可されないため、ユーザーは WS-SecurityPolicy で使用するためにデフォルトの非対称署名アルゴリズム(RSA-SHA1)を上書きできます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ASYMMETRIC_SIGNATURE_ALGORITHM**		
quarkus.cxf.endpoint."endpoints".security.symmetric.signature.algorithm	**string**	
この設定タグにより、WS-SecurityPolicy 仕様では他のアルゴリズムの使用が許可されないため、ユーザーは WS-SecurityPolicy で使用するためにデフォルトの対称署名アルゴリズム(HMAC-SHA1)を上書きできます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SYMMETRIC_SIGNATURE_ALGORITHM**		
quarkus.cxf.endpoint."endpoints".security.password.encryptor.instance	**string**	
Merlin Crypto 実装（またはカスタム Crypto 実装）でパスワードを暗号化または復号化するために使用される **org.apache.wss4j.common.crypto.PasswordEncryptor** Bean への 参照。 デフォルトでは、WSS4J は **org.apache.wss4j.common.crypto.JasyptPasswordEncryptor** を使用します。これは、Merlin Crypto 定義でキーストアパスワードを復号化するために使用するパスワードでインスタンス化する必要があります。このパスワードは、**callback-handler**を介して定義された CallbackHandler を介して取得されます。 暗号化されたパスワードは、ENC (encoded encrypted password)の形式で保存する必要があります。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_PASSWORD_ENCRYPTOR_INSTANCE** **以降、 Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.delegated.credential	**string**	
WS-Security に使用する Kerberos **org.ietf.jgss.GSSCredential** Bean への 参照。これは、クライアント認証情報を使用する代わりにサービスチケットを取得するために使用されます。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_DELEGATED_CREDENTIAL** 。**Quarkus CXF**: 2.5.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.security.context.creator	string	
WSS4J 処理結果のセットから CXF SecurityContext を作成するために使用される **org.apache.cxf.ws.security.wss4j.WSS4JSecurityContextCreator** Bean への参照。デフォルトの実装は **org.apache.cxf.ws.security.wss4j.DefaultWSS4JSecurityContextCreator** です。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CONTEXT_CREATOR**		
quarkus.cxf.endpoint."endpoints".security.security.token.lifetime	long	300000
セキュリティートークンの有効期間値（ミリ秒単位）。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TOKEN_LIFETIME** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.request.credential.delegation	boolean	false
true の場合、認証情報の委譲は KerberosClient で要求されます。それ以外の場合は、認証情報の委譲は KerberosClient にされません。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_REQUEST_CREDENTIAL_DELEGATION** **以降、Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.use.credential.delegation	boolean	false
true の場合、GSSCredential Bean は **delegated.credential** プロパティーを使用してメッセージコンテキストから取得され、サービスチケットの取得に使用されます。 このオプションはまだテストで対応されていない ため、実験的なものです。 **環境変数**:**QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_USE_CREDENTIAL_DELEGATION** **以降、Quarkus CXF**: 2.5.0		

設定プロパティー	タイプ	デフォルト
quarkus.cxf.endpoint."endpoints".security.kerberos.is.username.in.servicename.form	boolean	false
true の場合、Kerberos ユーザー名は servicename 形式になります。それ以外の場合、Kerberos ユーザー名は servicename 形式ではありません。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_IS_USERNAME_IN_SERVICENAME_FORM**		
quarkus.cxf.endpoint."endpoints".security.kerberos.jaas.context	string	
Kerberos に使用する JAAS コンテキスト名。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_JAAS_CONTEXT**		
quarkus.cxf.endpoint."endpoints".security.kerberos.spn	string	
使用する Kerberos サービスプロバイダー名(spn)。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_SPN** **(Quarkus CXF: 2.5.0 以降)**		
quarkus.cxf.endpoint."endpoints".security.kerberos.client	string	
サービスチケットの取得に使用される **org.apache.cxf.ws.security.kerberos.KerberosClient** Bean への参照。 このオプションはまだ テストで対応されていない ため、実験的なものです。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_CLIENT** **Quarkus CXF : 2.5.0 以降**		

6.5. WS-RELIABLEMESSAGING

WS-ReliableMessaging (WS-RM)は、ソフトウェア、システム、またはネットワークに障害が発生した場合でも、分散環境でのメッセージの信頼性の高い配信を保証するプロトコルです。

このエクステンションは、CXF フレームワークの WS-ReliableMessaging 実装を提供します。

6.5.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-rt-ws-rm を使用して新しいプロジェクトを作成するか、以下のコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-ws-rm</artifactId>
</dependency>
```

6.5.2. サポートされる標準

- WS-ReliableMessaging

6.5.3. 使用方法

アプリケーションが quarkus-cxf-rt-ws-rm に依存すると、application.properties で定義されているすべてのクライアントおよびサービスエンドポイントで WS-RM が有効になります。これは、quarkus.cxf.client.myClient.rm.enabled および quarkus.cxf.endpoint."/my-endpoint".rm.enabled プロパティーがデフォルトで true であるためです。

クライアントまたはサービスエンドポイントで WS-RM を有効にすると、WS-RM インターセプターが指定のクライアントまたはエンドポイントに追加されます。

さらに、設定オプションまたは以下の WS-Addressing オプションの一部を設定 する場合があります。

- quarkus.cxf.client.myClient.decoupled-endpoint
- quarkus.cxf.decoupled-endpoint-base

6.5.3.1. 実行可能な例

WS-RM に対応する統合テストがあり、Quarkus CXF ソースツリーにエンドポイントが切り離されています。

相互に通信する 2 つのアプリケーションに分割されます。

- Server
- クライアント

これを実行するには、最初にサーバーをローカルの Maven リポジトリーにインストールする必要があります。

```
$ cd test-util-parent/test-ws-rm-server-jvm
$ mvn clean install
```

そして、クライアントモジュールに実装された テストシナリオ を実行できます。

```
$ cd ../../integration-tests/ws-rm-client
$ mvn clean test
```

クライアント、サーバー、および分離されたエンドポイント間の SOAP メッセージの交換がコンソールに示されます。

6.5.4. 設定

ビルド時に修正される設定プロパティー。その他の設定プロパティーはすべて、ランタイム時にオーバーライドが可能です。

設定プロパティー	タイプ	デフォルト
quarkus.cxf.rm.namespace	string	http://schemas.xmlsoap.org/ws/2005/02/rm

設定プロパティー	タイプ	デフォルト
WS-RM バージョン名前空間： **http://schemas.xmlsoap.org/ws/2005/02/rm/** または **http://docs.oasis-open.org/ws-rx/wsrm/200702** **環境変数**:**QUARKUS_CXF_RM_NAMESPACE** **Quarkus CXF 以降**: 2.7.0		
quarkus.cxf.rm.wsa-namespace	**string**	**http://schemas.xmlsoap.org/ws/2004/08/addressing**
WS-Addressing バージョン名前空間： **http://schemas.xmlsoap.org/ws/2004/08/addressing** または **http://www.w3.org/2005/08/addressing**。**http://schemas.xmlsoap.org/ws/2005/02/rm/** RM 名前空間を使用していない限り、このプロパティーは無視されることに注意してください。 **環境変数**:**QUARKUS_CXF_RM_WSA_NAMESPACE** **Quarkus CXF 以降**: 2.7.0		
quarkus.cxf.rm.inactivity-timeout	**long**	
配信期間（ミリ秒単位）は、その期間中に送信者と受信者の間でメッセージ（確認応答やその他の制御メッセージを含む）が交換されない場合に、関連するシーケンスが閉じられます。設定されていない場合、関連するシーケンスは非アクティブのために閉じられることはありません。 **環境変数**:**QUARKUS_CXF_RM_INACTIVITY_TIMEOUT** **Quarkus CXF 以降**: 2.7.0		
quarkus.cxf.rm.retransmission-interval	**long**	**3000**
受信者によって確認されていないメッセージを再送信しようとする間隔（ミリ秒単位）。 **環境変数**:**QUARKUS_CXF_RM_RETRANSMISSION_INTERVAL** **Quarkus CXF 以降**: 2.7.0		
quarkus.cxf.rm.exponential-backoff	**boolean**	**false**
true の場合、再送信間隔は送信試行ごとに 2 倍になります。そうしないと、再送信間隔はすべての再送信試行で **quarkus.cxf.rm.retransmission-interval** と同じです。 **環境変数**:**QUARKUS_CXF_RM_EXPONENTIAL_BACKOFF** **Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.acknowledgement-interval	**long**	

設定プロパティー	タイプ	デフォルト
受信したメッセージの確認応答が RM 宛先によって送信されることが想定される時間（ミリ秒単位）。指定しない場合、確認応答は即座に送信されます。 **環境変数:QUARKUS_CXF_RM_ACKNOWLEDGEMENT_INTERVAL** **Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.store	**string**	
ソースおよび宛先シーケンスおよびメッセージ参照を保存するために使用される **org.apache.cxf.ws.rm.persistence.RMStore** Bean への参照。??? **環境変数:QUARKUS_CXF_RM_STORE** **Quarkus CXF 以降** 2.7.0		
quarkus.cxf.rm.feature-ref	**string**	**#defaultRmFeature**
quarkus.cxf.[client|service]."name".rm.enabled = true を持つクライアントおよびサービスエンドポイントに設定する **org.apache.cxf.ws.rm.feature.RMFeature** Bean への参照。 値が **#defaultRmFeature** の場合、Quarkus CXF は Bean を作成および設定します。 **環境変数:QUARKUS_CXF_RM_FEATURE_REF** **Quarkus CXF 以降**: 2.7.0		
quarkus.cxf.client."clients".rm.enabled	**boolean**	**true**
true の場合、WS-ReliableMessaging インターセプター がこのクライアントまたはサービスエンドポイントに追加されます。 **環境変数:QUARKUS_CXF_CLIENT__CLIENTS__RM_ENABLED**		
quarkus.cxf.endpoint."endpoints".rm.enabled	**boolean**	**true**
true の場合、WS-ReliableMessaging インターセプター がこのクライアントまたはサービスエンドポイントに追加されます。 **環境変数:QUARKUS_CXF_ENDPOINT__ENDPOINTS__RM_ENABLED**		

6.6. セキュリティートークンサービス (STS)

WS-Trust のコンテキストでセキュリティートークンの問題、更新、および検証を行います。

6.6.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-services-sts を使用して新しいプロジェクトを作成するか、これらのコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-services-sts</artifactId>
</dependency>
```

6.6.2. サポートされる標準

- WS-Trust

6.6.3. 使用方法

基本的な WS-Trust シナリオの主な部分は次のとおりです。

- WS-SecurityPolicy - トランスポートプロトコル、暗号化、署名などのセキュリティー要件を定義する場合を除き、< IssuedToken> アサーションも含まれます。サービスにアクセスするときにクライアントが準拠する必要のあるこれらのセキュリティートークンの要件および制約を指定します。
- セキュリティートークンサービス(STS): リクエスト時にセキュリティートークンを発行し、検証し、更新します。これは、クライアントを認証し、クライアントのアイデンティティーおよびパーミッションをアサートするトークンを発行する信頼できる認証局として機能します。
- クライアント - STS から Web サービスにアクセスするためのトークンを要求します。STS に対して認証し、必要なトークンの種類の詳細を提供する必要があります。
- サービス: STS を使用してクライアントを認証し、トークンを検証します。

6.6.3.1. 実行可能な例

Quarkus CXF ソースツリーの WS-Trust に対応する 統合テスト があります。それを確認し、個々のパーツが連携するように設定されている方法を確認します。

6.6.3.1.1. WS-SecurityPolicy

このポリシーは asymmetric-saml2-policy.xml ファイルにあります。その重要な部分は、SAML 2.0 トークンを必要と する <IssuedToken > アサーションです。

asymmetric-saml2-policy.xml

```
                    <sp:IssuedToken
                       sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                       <sp:RequestSecurityTokenTemplate>
                          <t:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-
profile-1.1#SAMLV2.0</t:TokenType>
                          <t:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/PublicKey</t:KeyType>
                       </sp:RequestSecurityTokenTemplate>
                       <wsp:Policy>
                          <sp:RequireInternalReference />
                       </wsp:Policy>
                       <sp:Issuer>
                          <wsaws:Address>http://localhost:8081/services/sts</wsaws:Address>
                          <wsaws:Metadata xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
                                          wsdli:wsdlLocation="http://localhost:8081/services/sts?wsdl">
                             <wsaw:ServiceName
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                                               xmlns:stsns="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"

EndpointName="UT_Port">stsns:SecurityTokenService</wsaw:ServiceName>
                          </wsaws:Metadata>
                       </sp:Issuer>
                    </sp:IssuedToken>
```

6.6.3.1.2. セキュリティートークンサービス (STS)

STS は Sts.java に実装されます。

Sts.java

```
@WebServiceProvider(serviceName = "SecurityTokenService", portName = "UT_Port",
targetNamespace = "http://docs.oasis-open.org/ws-sx/ws-trust/200512/", wsdlLocation = "ws-
trust-1.4-service.wsdl")
public class Sts extends SecurityTokenServiceProvider {

   public Sts() throws Exception {
      super();

      StaticSTSProperties props = new StaticSTSProperties();
      props.setSignatureCryptoProperties("stsKeystore.properties");
      props.setSignatureUsername("sts");
      props.setCallbackHandlerClass(StsCallbackHandler.class.getName());
      props.setIssuer("SampleSTSIssuer");

      List<ServiceMBean> services = new LinkedList<ServiceMBean>();
      StaticService service = new StaticService();
      final Config config = ConfigProvider.getConfig();
      final int port = LaunchMode.current().equals(LaunchMode.TEST) ?
config.getValue("quarkus.http.test-port", Integer.class)
            : config.getValue("quarkus.http.port", Integer.class);
      service.setEndpoints(Arrays.asList(
            "http://localhost:" + port + "/services/hello-ws-trust",
            "http://localhost:" + port + "/services/hello-ws-trust-actas",
            "http://localhost:" + port + "/services/hello-ws-trust-onbehalfof"));
      services.add(service);

      TokenIssueOperation issueOperation = new TokenIssueOperation();
      issueOperation.setServices(services);
      issueOperation.getTokenProviders().add(new SAMLTokenProvider());
      // required for OnBehalfOf
      issueOperation.getTokenValidators().add(new UsernameTokenValidator());
      // added for OnBehalfOf and ActAs
      issueOperation.getDelegationHandlers().add(new UsernameTokenDelegationHandler());
      issueOperation.setStsProperties(props);

      TokenValidateOperation validateOperation = new TokenValidateOperation();
      validateOperation.getTokenValidators().add(new SAMLTokenValidator());
      validateOperation.setStsProperties(props);

      this.setIssueOperation(issueOperation);
      this.setValidateOperation(validateOperation);
   }
}
```

application.properties で設定および設定されている。

application.properties

```
quarkus.cxf.endpoint."/sts".implementor = io.quarkiverse.cxf.it.ws.trust.sts.Sts
quarkus.cxf.endpoint."/sts".logging.enabled = pretty

quarkus.cxf.endpoint."/sts".security.signature.username = sts
quarkus.cxf.endpoint."/sts".security.signature.password = password
quarkus.cxf.endpoint."/sts".security.validate.token = false

quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.prov
ider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merl
in.keystore.type" = pkcs12
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merl
in.keystore.password" = password
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merl
in.keystore.file" = sts.pkcs12
```

6.6.3.1.3. サービス

サービスは TrustHelloServiceImpl.java に実装されます。

TrustHelloServiceImpl.java

```
@WebService(portName = "TrustHelloServicePort", serviceName = "TrustHelloService",
targetNamespace = "https://quarkiverse.github.io/quarkiverse-docs/quarkus-cxf/test/ws-trust",
endpointInterface = "io.quarkiverse.cxf.it.ws.trust.server.TrustHelloService")
public class TrustHelloServiceImpl implements TrustHelloService {
    @WebMethod
    @Override
    public String hello(String person) {
        return "Hello " + person + "!";
    }
}
```

上記の asymmetric-saml2-policy.xml は、サービスエンドポイントインターフェイス TrustHelloService.java で設定されます。

TrustHelloServiceImpl.java

```
@WebService(targetNamespace = "https://quarkiverse.github.io/quarkiverse-docs/quarkus-cxf/test/ws-trust")
@Policy(placement = Policy.Placement.BINDING, uri = "classpath:/asymmetric-saml2-policy.xml")
public interface TrustHelloService {
    @WebMethod
    @Policies({
            @Policy(placement = Policy.Placement.BINDING_OPERATION_INPUT, uri = "classpath:/io-policy.xml"),
            @Policy(placement = Policy.Placement.BINDING_OPERATION_OUTPUT, uri = "classpath:/io-policy.xml")
    })
    String hello(String person);
}
```

サービスエンドポイントは application.properties で設定されます。

application.properties

```
quarkus.cxf.endpoint."/hello-ws-trust".implementor = io.quarkiverse.cxf.it.ws.trust.server.TrustHelloServiceImpl
quarkus.cxf.endpoint."/hello-ws-trust".logging.enabled = pretty

quarkus.cxf.endpoint."/hello-ws-trust".security.signature.username = service
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.password = password
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.type" = pkcs12
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = service
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.properties."org.apache.ws.security.crypto.merlin.file" = service.pkcs12

quarkus.cxf.endpoint."/hello-ws-trust".security.encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/hello-ws-trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.type" = pkcs12
```

```
quarkus.cxf.endpoint."/hello-ws-trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password
quarkus.cxf.endpoint."/hello-ws-trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = service
quarkus.cxf.endpoint."/hello-ws-trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.file" = service.pkcs12
```

6.6.3.1.4. クライアント

最後に、SOAP クライアントがサービスと通信できるようにするには、STSClient を設定する必要があります。これは application.properties で行うことができます。

application.properties

```
quarkus.cxf.client.hello-ws-trust.security.sts.client.wsdl = http://localhost:${quarkus.http.test-port}/services/sts?wsdl
quarkus.cxf.client.hello-ws-trust.security.sts.client.service-name = {http://docs.oasis-open.org/ws-sx/ws-trust/200512/}SecurityTokenService
quarkus.cxf.client.hello-ws-trust.security.sts.client.endpoint-name = {http://docs.oasis-open.org/ws-sx/ws-trust/200512/}UT_Port
quarkus.cxf.client.hello-ws-trust.security.sts.client.username = client
quarkus.cxf.client.hello-ws-trust.security.sts.client.password = password
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.username = sts
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.type" = pkcs12
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = client
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.file" = client.pkcs12
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.username = client
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.type"
```

```
= pkcs12
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.pass
word" = password
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.alias"
= client
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.file" =
client.pkcs12
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.usecert = true
```

ヒント

STS クライアントを設定するためのプロパティーは、io.quarkiverse.cxf:quarkus-cxf-rt-ws-security 拡張によって提供され、その リファレンスページ に記載されています。

または、クライアントを Bean 参照として設定できます。

application.properties

```
quarkus.cxf.client.hello-ws-trust-bean.security.sts.client = #stsClientBean
```

その場合、@Named Bean はプログラム的に生成する必要があります。たとえば、@jakarta.enterprise.inject.Produces を使用します。

BeanProducers.java

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

import org.apache.cxf.ws.security.SecurityConstants;

import io.quarkiverse.cxf.ws.security.sts.client.STSClientBean;
```

```
public class BeanProducers {

    /**
     * Create and configure an STSClient for use by the TrustHelloService client.
     */
    @Produces
    @ApplicationScoped
    @Named("stsClientBean")
    STSClientBean createSTSClient() {
        /*
         * We cannot use org.apache.cxf.ws.security.trust.STSClient as a return type of this bean
producer method
         * because it does not have a no-args constructor. STSClientBean is a subclass of
STSClient having one.
         */
        STSClientBean stsClient = STSClientBean.create();
        stsClient.setWsdlLocation("http://localhost:8081/services/sts?wsdl");
        stsClient.setServiceQName(new QName("http://docs.oasis-open.org/ws-sx/ws-
trust/200512/", "SecurityTokenService"));
        stsClient.setEndpointQName(new QName("http://docs.oasis-open.org/ws-sx/ws-
trust/200512/", "UT_Port"));
        Map<String, Object> props = stsClient.getProperties();
        props.put(SecurityConstants.USERNAME, "client");
        props.put(SecurityConstants.PASSWORD, "password");
        props.put(SecurityConstants.ENCRYPT_PROPERTIES,

Thread.currentThread().getContextClassLoader().getResource("clientKeystore.properties"));
        props.put(SecurityConstants.ENCRYPT_USERNAME, "sts");
        props.put(SecurityConstants.STS_TOKEN_USERNAME, "client");
        props.put(SecurityConstants.STS_TOKEN_PROPERTIES,

Thread.currentThread().getContextClassLoader().getResource("clientKeystore.properties"));
        props.put(SecurityConstants.STS_TOKEN_USE_CERT_FOR_KEYINFO, "true");
        return stsClient;
    }
}
```

6.7. HTTP 非同期トランスポート

Apache HttpComponents HttpClient 5 を使用して非同期 SOAP クライアントを実装します。

6.7.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-rt-transports-http-hc5 を使用して新しいプロジェクトを作成するか、以下のコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-transports-http-hc5</artifactId>
</dependency>
```

6.7.2. 使用方法

クラスパスで quarkus-cxf-rt-transports-http-hc5 依存関係が利用可能になると、CXF は非同期呼び出しに HttpAsyncClient を使用し、同期呼び出しに HttpURLConnection を使用し続けます。

6.7.2.1. 非同期メソッドの生成

非同期クライアント呼び出しには、サービスエンドポイントインターフェイスで追加のメソッドが必要です。このコードはデフォルトでは生成されません。

これを有効にするには、enableAsyncMapping を true に設定して JAX-WS バインディングファイルを作成する必要があります。

ヒント

このセクションで使用されるサンプルコードスニペットは、Quarkus CXF のソースツリーの HC5 統合テスト のものです。

src/main/resources/wsdl/async-binding.xml

```
<?xml version="1.0"?>
<bindings
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns="https://jakarta.ee/xml/ns/jaxws"
        wsdlLocation="CalculatorService.wsdl">
    <bindings node="wsdl:definitions">
        <enableAsyncMapping>true</enableAsyncMapping>
    </bindings>
</bindings>
```

その後、このファイルは additional-params プロパティーを介して wsdl2java に渡す必要があります。

application.properties

```
quarkus.cxf.codegen.wsdl2java.includes = wsdl/*.wsdl
quarkus.cxf.codegen.wsdl2java.additional-params = -b,src/main/resources/wsdl/async-binding.xml
```

6.7.2.2. 非同期クライアントおよび Mutiny

非同期スタブが利用可能になったら、以下に示すように、クライアント呼び出しを io.smallrye.mutiny.Uni にラップできます。

```
package io.quarkiverse.cxf.hc5.it;

import java.util.concurrent.Future;

import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;

import org.jboss.eap.quickstarts.wscalculator.calculator.AddResponse;
import org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService;

import io.quarkiverse.cxf.annotation.CXFClient;
import io.smallrye.mutiny.Uni;

@Path("/hc5")
public class Hc5Resource {

    @Inject
    @CXFClient("myCalculator") // name used in application.properties
    CalculatorService myCalculator;

    @SuppressWarnings("unchecked")
    @Path("/add-async")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Uni<Integer> addAsync(@QueryParam("a") int a, @QueryParam("b") int b) {
        return Uni.createFrom()
                .future(
                        (Future<AddResponse>) myCalculator
                                .addAsync(a, b, res -> {
```

```
                    }))
            .map(addResponse -> addResponse.getReturn());
    }

}
```

6.7.2.3. スレッドプール

このエクステンションによって提供される非同期クライアントは、Quarkus が提供するスレッドプールとともに ManagedExecutor を利用します。スレッドプールは、オプション の quarkus.thread-pool.* ファミリーを使用して設定できます。そのため、org.apache.cxf.transports.http.configuration.HTTPClientPolicy のエグゼキューターおよびスレッドプール関連の属性は Quarkus の非同期クライアントには適用されません。

ヒント

CXF 非同期クライアントの詳細と、CXF ドキュメント を参照してください。

6.8. XJC PLUGINS

wsdl2java コード生成用の XJC プラグイン。quarkus.cxf.codegen.wsdl2java.additional-params で以下のいずれかを使用する場合は、このエクステンションを追加する必要があります。

- -xjc-Xbg - ブール値フィールドの is Foo （）アクセサーメソッドの代わりに get Foo （）を生成します。
- -xjc-Xdv - フィールドが明示的に設定されていない限り、生成された getter メソッドはスキーマに定義されたデフォルト値を返します。
- -xjc-Xjavadoc - スキーマに存在する xs:documentation をもとに JavaDoc を生成します。
- -xjc-Xproperty-listener - 生成された Bean に PropertyChangeListener サポートを追加します。
- -xjc-Xts: モデルクラスの toString （） メソッドを生成します。
- -xjc-Xwsdlextension - WSDL の extensors として WSDL4J と直接使用できる Bean を生

成します。

ヒント

wsdl2java の詳細は、ユーザーガイドの wsdl2java セクションを確認してください。

6.8.1. Maven コーディネート

code.quarkus.redhat.com で quarkus-cxf-xjc-plugins を使用して新しいプロジェクトを作成するか、以下のコーディネートを既存のプロジェクトに追加します。

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-xjc-plugins</artifactId>
</dependency>
```