Red Hat Fuse 7.9

JBoss EAP へのデプロイ

JBoss Enterprise Application Platform (EAP) コンテナーへのアプリケーションパッケージのデプロイ

Last Updated: 2021-08-16

Red Hat Fuse 7.9 JBoss EAP へのデプロイ

JBoss Enterprise Application Platform (EAP) コンテナーへのアプリケーションパッケージのデプロイ

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本ガイドでは、アプリケーションを JBoss EAP コンテナーにデプロイするためのオプションを説明します。

目次

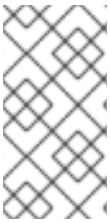
多様性を受け入れるオープンソースの強化

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、CTO である Chris Wright のメッセージ をご覧ください。

第1章 JBOSS FUSE ON JBOSS EAP デプロイメントの概要

Fuse on EAP パッケージを JBoss EAP コンテナーへインストールした後に、Red Hat JBoss Enterprise Application Platform (JBoss EAP) に Fuse アプリケーションをデプロイできます。

ここでは、Camel on EAP サブシステムを使用するデプロイメントモデルを説明します。Fuse の Apache Camel では、コンテナーを選択して統合されたアプリケーションを実行できます。



注記

Red Hat JBoss EAP は、管理者と開発者の両方に対応するためのさまざまなアプリケーションデプロイメントおよび設定オプションを備えています。JBOSS EAP 設定とデプロイメントプロセスに関する詳細は『Red Red Hat JBoss EAP Configuration』 ガイドを参照してください。

1.1. サポート対象の製品バージョン

Fuse 7.9 をサポートする最新バージョンの JBoss EAP を確認するには、「Red Hat Fuse でサポートされる構成」を参照してください。

1.2. CAMEL ON EAP サブシステム

Camel on EAP サブシステムは、Apache Camel を直接 JBoss EAP コンテナーに統合します。このサブシステムは、Fuse on EAP パッケージを JBoss EAP コンテナーにインストールした後に利用できます。Camel デプロイメントには、Camel コンポーネントの簡素化されたデプロイメントや、基盤の JBoss EAP コンテナーによるより密接なインテグレーションなど、多くの利点があります。

Red Hat は、JBoss EAP での Apache Camel アプリケーションのデプロイメントには、Camel on EAP サブシステムデプロイメントモデルを使用することを推奨します。

第2章 JBOSS EAP でのアプリケーションのビルド

2.1. 概要

以下の例は、CDI Bean を Camel ルートと統合するために Red Hat Fuse on EAP で **camel-cdi** コンポーネントを使用する方法を実証します。

この例では、Camel ルートはサーブレット **HTTP GET** リクエストからメッセージペイロードを取得し、それをダイレクトエンドポイントに渡します。その後、ペイロードを Camel CDI Bean 呼び出しに渡してメッセージの応答を生成し、Web ブラウザーページで出力を表示します。

2.2. プロジェクトの実行

プロジェクトを実行する前に、セットアップに Maven と Red Hat Fuse とのアプリケーションサーバーが含まれるようにしてください。プロジェクトを実行するには、以下の手順を実行します。

1. スタンドアロンモードでアプリケーションサーバーを起動します。
   - Linux の場合: **${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml**
   - Windows の場合: **%JBOSS_HOME%\bin\standalone.bat -c standalone-full.xml**
2. プロジェクトをビルドしてデプロイします: **mvn install -Pdeploy**
3. ここで、**http://localhost:8080/example-camel-cdi/?name=World** にアクセスします。以下のメッセージ **Hello World from 127.0.0.1** は、Web ページの出力として表示されます。また、以下のように **MyRouteBuilder.java** クラスの Camel Route を確認することもできます。

```
from("direct:start").bean("helloBean");
```

bean DSL により、Camel は Bean レジストリーで **helloBean** という名前の Bean を検索します。また、**SomeBean** クラスによって、Bean は Camel で利用できます。**@Named** アノテーションを使用して、**camel-cdi** は Bean を Camel Bean レジストリーに追加します。

```
@Named("helloBean")

public class SomeBean {

    public String someMethod(String name) throws Exception {

        return String.format("Hello %s from %s", name, InetAddress.getLocalHost().getHostAddress());

    }

}
```

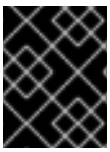
詳細は、**$ EAP_HOME/quickstarts/camel/camel-cdi** ディレクトリーを参照してください。

2.3. JBOSS EAP の BOM ファイル

Maven BOM (Bill of Materials) ファイルの目的は、正常に動作する Maven 依存関係バージョンのセットを提供し、各 Maven アーティファクトに対して個別にバージョンを定義する必要をなくすことです。

JBoss EAP の Fuse BOM には以下の利点があります。

- Maven 依存関係のバージョンを定義するため、依存関係を POM に追加するときにバージョンを指定する必要がありません。
- 特定バージョンの Fuse に対して完全にテストされ、完全にサポートする依存関係のセットを定義します。
- Fuse のアップグレードを簡素化します。



重要

Fuse BOM によって定義される依存関係のセットのみが Red Hat によってサポートされます。

Maven プロジェクトに BOM ファイルを組み込むには、以下の例のように、プロジェクトの **pom.xml** ファイル (または親 POM ファイル内の) **dependencyManagement** 要素を指定します。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.9.0.fuse-sb2-790065-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-eap-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

依存関係管理のメカニズムを使用して BOM を指定した後、アーティファクトのバージョンを指定**しなくても**、Maven 依存関係を POM に追加できるようになります。たとえば、 **camel-velocity** コンポーネントの依存関係を追加するには、以下の XML フラグメントを POM の **dependencies** 要素に追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <scope>provided</scope>
</dependency>
```

この依存関係の定義では、**version** 要素が省略されることに注意してください。

第3章 FEATURES

本章では、Camel on EAP の機能に必要な情報を提供します。

Camel コンテキスト定義

Camel コンテキストは、以下のようなサブシステム定義の一部として standalone-camel.xml および domain.xml で設定できます。

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel コンテキストデプロイメント

-camel-context.xml サフィックスを使用すると、以下として Camel コンテキストを JBossEAP にデプロイできます。

- スタンドアロンの XML ファイル
- その他のサポートされるデプロイメントの一部

デプロイメントには、複数の **-camel-context.xml** ファイルが含まれる場合があります。

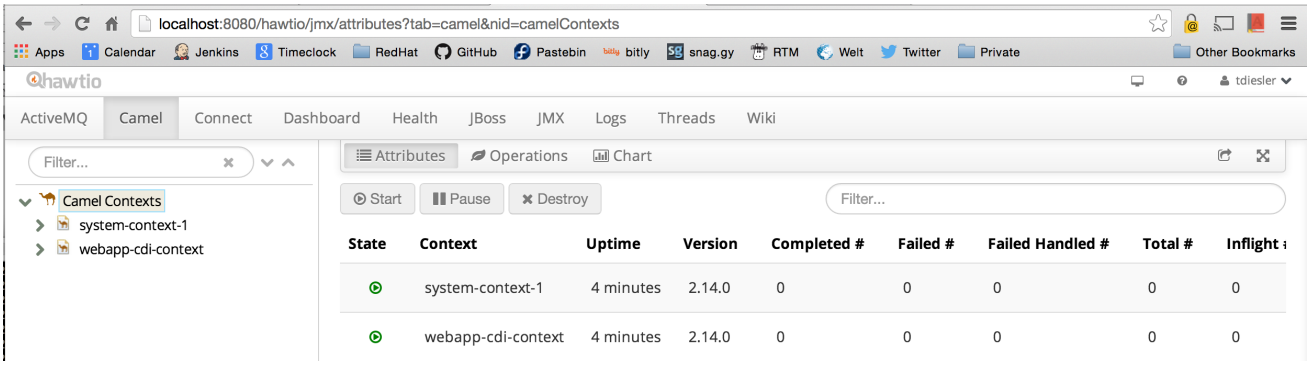
デプロイされた Camel コンテキストは、以下のように CDI のインジェクトが可能です。

```
@Resource(lookup = "java:jboss/camel/context/mycontext")
CamelContext camelContext;
[discrete]
### Management Console
```

デフォルトでは、管理コンソールへのアクセスは保護されています。したがって、最初に管理ユーザーを設定する必要があります。

```
$ bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
```

Hawt.io コンソールには、サブシステム設定の camel コンテキストが表示されるはずです。



Arquillian テストサポート

Camel on EAP テストスイートは WildFly Arquillian 管理のコンテナーを使用します。これにより、すでに実行中の JBoss EAP インスタンスに接続したり、必要に応じてスタンドアロンサーバーインスタンスを起動したりできます。

Arquillianテストケースにこれらの Camel on EAP 固有のタイプを注入できるようにする、多数のテスト Enricher が実装されています。

```
@ArquillianResource
CamelContextFactory contextFactory;

@ArquillianResource
CamelContextRegistry contextRegistry;
```

第4章 設定

本章では、Camel サブシステムおよびデプロイメント設定に必要な情報を提供します。

Camel サブシステムの設定

Camel サブシステム設定には、静的ルートが含まれる場合があります。ただし、システムはルートを自動的に開始します。

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel デプロイメント設定

Camel デプロイメントのデフォルト設定を変更する場合は、デプロイメントで **WEB-INF/jboss-all.xml** または **META-INF/jboss-all.xml** 設定ファイルのいずれかを編集できます。

jboss-all.xml ファイル内の **<jboss-camel>** XML 要素を使用して、Camel 設定を制御します。

Camel サブシステムの無効化

camel サブシステムをデプロイメントに追加しない場合は、**jboss-camel** XML 要素で **enabled="false"** 属性を設定します。

jboss-all.xml ファイルの例:

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0" enabled="false"/>
</jboss>
```

コンポーネントの選択

ネストされた **<component>** または **<component-module>** XML 要素を追加する場合、Camel コンポーネントのデフォルトリストをデプロイメントに追加する代わりに、指定されたコンポーネントのみがデプロイメントに追加されます。

jboss-all.xml ファイルの例:

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0">
    <component name="camel-ftp"/>
    <component-module name="org.apache.camel.component.rss"/>
  </jboss-camel>
</jboss>
```

第5章 JARKARTA EE インテグレーション

本章では、Jarkarta EE とのインテグレーションポイントに必要な情報を提供します。

5.1. CDI

Camel CDI コンポーネントは、CDI を依存性注入フレームワークとして使用して、Apache Camel の自動設定を提供します。ただし、設定よりも規約に基づいています。CDI Bean で Camel アノテーションを簡単に使用できるように、標準の Camel Bean インテグレーションを実装します。

CDI の詳細は、cdi ドキュメントを参照してください。

以下の例は、ルートで Camel Context を使用および関連付けする方法を表しています。

```
@Startup
@ApplicationScoped
@ContextName("cdi-context")
public class MyRouteBuilder extends RouteBuilder {

    @Override
    public void configure() throws Exception {
     from("direct:start").transform(body().prepend("Hi"));
    }
}
```

```
@Inject
@ContextName("cdi-context")
private CamelContext camelctx;
```

5.1.1. XML DSL 設定のインポート

Camel CDI インテグレーションを使用すると、**@ImportResource** アノテーションで既存の XML DSL ファイルをインポートできます。

```
@ImportResource("camel-context.xml")
class MyBean {
}
```

注記

インポートされたファイルの場所がデプロイメントクラスパスに存在する必要があります。ファイルを **WEB-INF** などの場所に配置しても機能しません。ただし、**WEB-INF/classes** は問題なく機能します。

5.2. EJB

管理サポートは、EJB3 サブシステムと統合する ejb コンポーネントを介して提供されます。

```
CamelContext camelctx = new DefaultCamelContext();
    camelctx.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
```

```
            from("direct:start").to("ejb:java:module/HelloBean");
        }
    });
```

5.3. JAXB

JAXB のサポートは Camel JAXB データフォーマット を通じて提供されます。

Camelは、XML データの JAXB アノテーション付きクラスへのマーシャリングと、クラスから XML へのマーシャリングをサポートしています。以下は、Camel JAXB データフォーマットクラスでマーシャリングおよびアンマーシャリングするための簡単な Camel ルートを示しています。

5.3.1. JAXB アノテーション付きクラス

```
@XmlRootElement(name = "customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer implements Serializable {

    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

5.3.2. JAXB クラス XML 表現

```
<customer xmlns="http://org/wildfly/test/jaxb/model/Customer">
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</customer>
```

5.3.3. Camel JAXB アンマーシャリング

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext(getClass().getClassLoader());

final JaxbDataFormat jaxb = new JaxbDataFormat();
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");

camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .unmarshal(jaxb);
    }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();

// Send an XML representation of the customer to the direct:start endpoint
Customer customer = producer.requestBody("direct:start", readCustomerXml(), Customer.class);
Assert.assertEquals("John", customer.getFirstName());
Assert.assertEquals("Doe", customer.getLastName());
```

5.3.4. Camel JAXB マーシャリング

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext();

final JaxbDataFormat jaxb = new JaxbDataFormat();
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");

camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .marshal(jaxb);
    }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();
Customer customer = new Customer("John", "Doe");
String customerXML = producer.requestBody("direct:start", customer, String.class);
Assert.assertEquals(readCustomerXml(), customerXML);
```

5.4. JAX-RS

JAX-RS のサポートは Camel CXF-RS によって提供されます。

5.4.1. CXF-RS プロデューサー

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://camel.apache.org/schema/cxf"
```

```
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

    <cxf:rsClient id="cxfProducer"
                  address="http://localhost:8080/rest"
                  serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

    <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:start" />
            <setHeader headerName="operationName">
                <simple>greet</simple>
            </setHeader>
            <setHeader headerName="CamelCxfRsUsingHttpAPI">
                <constant>false</constant>
            </setHeader>
            <to uri="cxfrs:bean:cxfProducer" />
        </route>
    </camelContext>
</beans>
```

5.4.2. CXF-RS コンシューマー

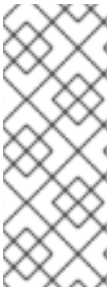
```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://camel.apache.org/schema/cxf"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd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

    <cxf:rsServer id="cxfConsumer"
                  address="http://localhost:8080/rest"
                  serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

    <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="cxfrs:bean:cxfConsumer" />
            <setBody>
                <constant>Hello world</constant>
            </setBody>
        </route>
    </camelContext>
</beans>
```

5.4.3. Camel REST DSL での JAX-RS コンシューマー

Camel REST DSL は、JAX-RS コンシューマーとして動作する Camel ルートを書き込む機能を提供します。以下の RouteBuilder クラスはこれを示しています。

```
@Startup
@ApplicationScoped
@ContextName("rest-camel-context")
public class RestConsumerRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {

    // Use the camel-undertow component to provide REST integration
    restConfiguration().component("undertow")
      .contextPath("/rest").port(8080).bindingMode(RestBindingMode.json);

    rest("/customer")
      // GET /rest/customer
      .get()
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomers")
      // GET /rest/customer/1
      .get("/{id}")
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomer")
      // POST /rest/customer
      .post()
        .type(Customer.class)
        .to("direct:createCustomer");
      // PUT /rest/customer
      .put()
        .type(Customer.class)
        .to("direct:updateCustomer");
      // DELETE /rest/customer/1
      .delete("/{id}")
        .to("direct:deleteCustomer");
  }
}
```

バインディングモードを設定することで、Camel は 'produces()' または 'type()' 設定ステップを指定して JSON データをマーシャリングおよびアンマーシャリングできます。



注記

- REST DSL 設定は **restConfiguration().component("undertow")** で始まります。
- Camel on EAP サブシステムは、REST DSL と使用する camel-servlet コンポーネントおよび camel-undertow コンポーネントのみをサポートします。ただし、他のコンポーネントを設定すると動作しません。

5.4.4. セキュリティー

JAX-RS セキュリティーのセクション を参照してください。

5.4.5. Fuse on EAP のクイックスタートサンプル

クイックスタートの例は、**quickstarts/camel/camel-cxf-jaxrs** ディレクトリーの Fuse on EAP インストールで利用できます。

5.5. JAX-WS

WebService サポートは、Apache CXF も使用する JBoss EAP WebServices サブシステムと統合される CXF コンポーネントを介して提供されます。

5.5.1. JAX-WS CXF プロデューサー

以下のコード例では CXF を使用して、WildFly web services サブシステム によってデプロイされた web サービスを使用します。

5.5.1.1. JAX-WS Web サービス

以下の簡単な Web サービスには、2 つの文字列引数を連結して返す、単純な「greet」メソッドがあります。

JBoss EAP の web サービスサブシステムが JAX-WS アノテーションが含まれるクラスを検出すると、CXF エンドポイントをブートストラップします。以下の例では、サービスのエンドポイントは http://hostname:port/context-root/greeting にあります。

```
// Service interface
@WebService(name = "greeting")
public interface GreetingService {
    @WebMethod(operationName = "greet", action = "urn:greet")
    String greet(@WebParam(name = "message") String message, @WebParam(name = "name")
String name);
}

// Service implementation
public class GreetingServiceImpl implements GreetingService{
    public String greet(String message, String name) {
        return message + " " + name ;
    }
}
```

5.5.1.2. Camel ルート設定

この RouteBuilder は、上記で定義された 'greeting' Web サービスを消費する CXF プロデューサーエンドポイントを設定します。CDI を camel-cdi コンポーネントと組み合わせて使用して、RouteBuilder と CamelContext をブートストラップします。

```
@Startup
@ApplicationScoped
@ContextName("cxf-camel-context")
public class CxfRouteBuilder extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start")
        .to("cxf://http://localhost:8080/example-camel-cxf/greeting?serviceClass=" +
```

```
GreetingService.class.getName());
    }
}
```

greeting web サービスの「挨拶」には 2 つのパラメーターが必要です。これらは、**ProducerTemplate** の方法で上記のルートに提供できます。Web サービスメソッドの引数の値は、エクスチェンジボディーとして渡されるオブジェクト配列を構築することで設定されます。

```
String message = "Hello"
String name = "Kermit"

ProducerTemplate producer = camelContext.createProducerTemplate();
Object[] serviceParams = new Object[] {message, name};
String result = producer.requestBody("direct:start", serviceParams, String.class);
```

5.5.2. Camel CXF JAX-WS コンシューマー

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

    <cxf:cxfEndpoint id="cxfConsumer"
                     address="http://localhost:8080/webservices/greeting"
                     serviceClass="org.wildfly.camel.examples.cxf.jaxws.GreetingService" />

    <camelContext id="cxfws-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="cxf:bean:cxfConsumer" />
            <to uri="log:ws" />
        </route>
    </camelContext>

</beans>
```

5.5.3. セキュリティー

JAX-WS セキュリティーのセクション を参照してください。

5.5.4. Fuse on EAP のクイックスタートサンプル

クイックスタートの例は、**quickstarts/camel/camel-cxf-jaxws** ディレクトリーの Fuse on EAP インストールで利用できます。

5.6. JMS

メッセージングサポートは、JBoss EAP Messaging (ActiveMQ Artemis) サブシステムと統合される JMS コンポーネントを介して提供されます。

他の JMS 実装とのインテグレーションは、ベンダー固有のリソースアダプターを設定するか、使用できない場合は、JBoss Generic JMS リソースアダプターを使用して行うことができます。

5.6.1. JBoss EAP JMS 設定

標準の JBoss EAP XML 設定ファイルを使用して JBoss EAP messaging サブシステム を設定できます。たとえば、standalone.xml または domain.xml です。

この例では、メモリーインスタンスで組み込み ActiveMQ Artemis を使用します。最初に、以下の XML 設定を jms-destinations セクションに追加して、messaging サブシステムの新しい JMS キューを設定します。

```
<jms-queue name="WildFlyCamelQueue">
  <entry name="java:/jms/queue/WildFlyCamelQueue"/>
</jms-queue>
```

または、CLI スクリプトを使用してキューを追加することもできます。

```
$ jms-queue add --queue-address=WildFlyCamelQueue --
entries=queue/WildFlyCamelQueue,java:/jms/queue/WildFlyCamelQueue
```

また、カスタム jms.xml デプロイメント記述子内に **messaging-deployment** 設定を作成することもできます。詳細は、JBoss EAP messaging サブシステムサブシステムのドキュメント内の「Deployment of -jms.xml files」セクションを参照してください。

5.6.2. Camel ルート設定

以下の JMS プロデューサーおよびコンシューマーの例では、JBoss EAP の組み込み ActiveMQ Artemis サーバーを利用し、メッセージを宛先との間でパブリッシュおよび使用します。

この例では、camel-cdi コンポーネントとともに CDI も使用します。JMS ConnectionFactory インスタンスは、JNDI ルックアップを介して Camel RouteBuilder に注入されます。

5.6.2.1. JMS プロデューサー

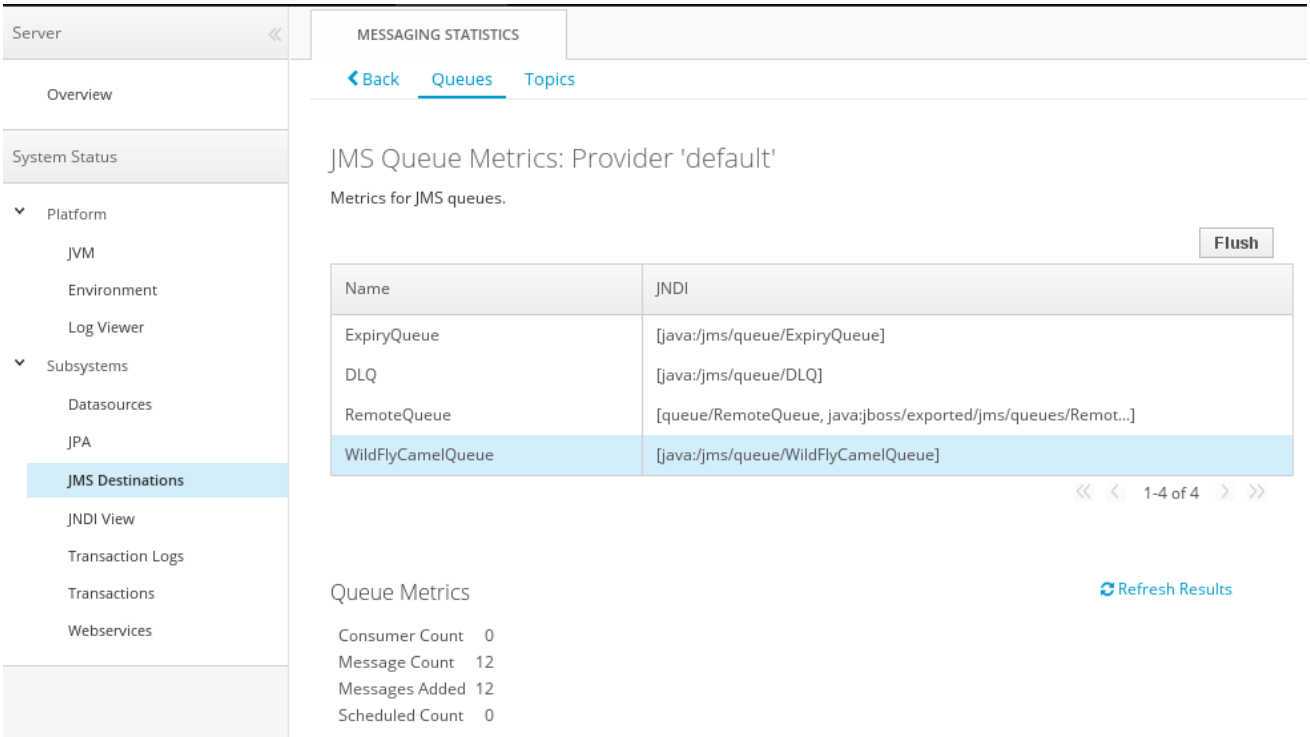
DefaultJMSConnectionFactory 接続ファクトリーは、JNDI から RouteBuilder に注入されます。JBoss EAP XML 設定では、messaging サブシステム内で接続ファクトリーを確認できます。

次に、タイマーエンドポイントが 10 秒ごとに実行され、以前に設定された WildFlyCamelQueue 宛先に XML ペイロードが送信されます。

```
@Startup
@ApplicationScoped
@ContextName("jms-camel-context")
public class JmsRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:jboss/DefaultJMSConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Override
  public void configure() throws Exception {
    JmsComponent component = new JmsComponent();
    component.setConnectionFactory(connectionFactory);
```

```
    getContext().addComponent("jms", component);

    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("jms:queue:WildFlyCamelQueue")
    .log("JMS Message sent");
  }
}
```

JMS メッセージが WildFlyCamelQueue 宛先に追加されるたびに、ログメッセージがコンソールに出力されます。メッセージが実際にキューに配置されていることを確認するには、JBossEAP 管理コンソールを使用できます。



5.6.2.2. JMS コンシューマー

JMS メッセージを使用するために、Camel RouteBuilder 実装はプロデューサーの例と似ています。

前述と同様に、接続ファクトリーは JNDI から検出され、JMSComponent インスタンスに注入および設定されます。

JMS エンドポイントが WildFlyCamelQueue 宛先からメッセージを消費すると、コンテンツはコンソールに記録されます。

```
@Override
public void configure() throws Exception {
  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);

  getContext().addComponent("jms", component);

  from("jms:queue:WildFlyCamelQueue")
  .to("log:jms?showAll=true");
}
```

5.6.2.3. JTA トランザクション

Camel JMS ルートが JMS トランザクションに参加できるようにするには、追加の設定が必要です。camel-jms は spring-jms を中心に構築されているため、JBossEAP のトランザクションマネージャーおよび接続ファクトリーと連携できるようにいくつかの Spring クラスを設定する必要があります。以下のコード例は、CDI を使用してトランザクション JMS Camel ルートを設定する方法を示しています。

camel-jms コンポーネントには、**org.springframework.transaction.PlatformTransactionManager** タイプのトランザクションマネージャーが必要です。そのため、最初に **JtaTransactionManager** を拡張する Bean を作成します。Bean には **@Named** アノテーションが付けられており、Camel Bean レジストリー内に Bean を登録できることに注意してください。また、JBoss EAP トランザクションマネージャーとユーザートランザクションインスタンスは CDI を使用して注入されることに注意してください。

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {

  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

次に、使用するトランザクションポリシーを宣言する必要があります。ここでも、**@Named** アノテーションを使用して Bean を Camel で利用可能にします。また、トランザクションマネージャーも、必要なトランザクションポリシーで TransactionTemplate を作成できるように注入されます。このインスタンスの PROPAGATION_REQUIRED

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

これで、Camel RouteBuilder クラスを設定し、Camel JMS コンポーネントに必要な依存関係を注入できるようになります。JBoss EAP XA 接続ファクトリーは、以前に設定されたトランザクションマネージャーと注入されます。

この例では、queue1 からメッセージが消費されるたびに queue2 という名前の別の JMS キューにルーティングされます。queue2 から消費されるメッセージにより、JMS トランザクションは rollback () DSL メソッドを使用してロールバックされます。これにより、元のメッセージがデッドレターキュー (DLQ) に配置されます。

```
@Startup
@ApplicationScoped
```

```
@ContextName("jms-camel-context")
public class JMSRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/JmsXA")
  private ConnectionFactory connectionFactory;

  @Inject
  CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    // Creates a JMS component which supports transactions
    JmsComponent jmsComponent = JmsComponent.jmsComponentTransacted(connectionFactory,
transactionManager);
    getContext().addComponent("jms", jmsComponent);

    from("jms:queue:queue1")
      .transacted("PROPAGATION_REQUIRED")
      .to("jms:queue:queue2");

    // Force the transaction to roll back. The message will end up on the Wildfly 'DLQ' message queue
    from("jms:queue:queue2")
      .to("log:end")
      .rollback();
  }
```

5.6.2.4. リモート JMS 宛先

1 つの JBoss EAP インスタンスは、リモート JNDI を介して別の JBoss EAP インスタンスに設定された ActiveMQ Artemis 宛先にメッセージを送信できます。

そのためには、追加の JBoss EAP 設定が必要になります。最初に、エクスポートされた JMS キューが設定されます。

java:jboss/exported namespace でバインドされた JNDI 名のみがリモートクライアントの候補として考慮されるため、キューには適切な名前が付けられます。

注記

JBoss EAP クライアントアプリケーションサーバー とJBoss EAP リモートサーバーにキューを設定する必要があります。

```
<jms-queue name="RemoteQueue">
  <entry name="java:jboss/exported/jms/queues/RemoteQueue"/>
</jms-queue>
```

クライアントがリモートサーバーに接続できるようにするには、ユーザーのアクセスクレデンシャルを設定する必要があります。リモートサーバーで add user ユーティリティー を実行し、guest グループ内に新しいアプリケーションユーザーを作成します。この例には、「admin」という名前のユーザーと「secret」というパスワードがあります。

RouteBuilder 実装は前述の例とは異なります。接続ファクトリーを注入する代わりに、InitialContext を設定し、JNDI から取得する必要があります。

この **configureInitialContext** メソッドにより、この InitialContext が作成されます。リモート JBoss EAP インスタンスのホスト名とポート番号を参照するプロバイダー URL を設定する必要があることに注意してください。この例では、JBoss EAP JMS http-connector が使用されていますが、代替方法は ここ に記載されています。

最後に、ルートは 10 秒ごとに XML ペイロードを以前設定されたリモート宛先 (RemoteQueue) に送信するよう設定されます。

```
@Override
public void configure() throws Exception {
  Context initialContext = configureInitialContext();
  ConnectionFactory connectionFactory = (ConnectionFactory)
initialContext.lookup("java:jms/RemoteConnectionFactory");

  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);

  getContext().addComponent("jms", component);

  from("timer://foo?fixedRate=true&period=10000")
  .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
  .to("jms:queue:RemoteQueue?username=admin&password=secret")
  .to("log:jms?showAll=true");
}

private Context configureInitialContext() throws NamingException {
  final Properties env = new Properties();
  env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
  env.put(Context.PROVIDER_URL, System.getProperty(Context.PROVIDER_URL, "http-
remoting://my-remote-host:8080"));
  env.put(Context.SECURITY_PRINCIPAL, System.getProperty("username", "admin"));
  env.put(Context.SECURITY_CREDENTIALS, System.getProperty("password", "secret"));
  return new InitialContext(env);
}
```

5.6.3. セキュリティー

JMS セキュリティーのセクション を参照してください。

5.6.4. Fuse on EAP のクイックスタートサンプル

クイックスタートの例は、**quickstarts/camel/camel-jms** ディレクトリーの Fuse on EAP インストールで利用できます。

5.7. JMX

JBoss EAP JMX サブシステムと統合される JMX コンポーネントを介して、管理サポートを提供できます。

```
CamelContext camelctx = contextFactory.createWildflyCamelContext(getClass().getClassLoader());
camelctx.addRoutes(new RouteBuilder() {
   @Override
```

```
    public void configure() throws Exception {
        String host = InetAddress.getLocalHost().getHostName();
        from("jmx:platform?format=raw&objectDomain=org.apache.camel&key.context=" + host +
"/system-context-1&key.type=routes&key.name=\"route1\"" +
        "&monitorType=counter&observedAttribute=ExchangesTotal&granularityPeriod=500").
        to("direct:end");
    }
});
camelctx.start();

ConsumerTemplate consumer = camelctx.createConsumerTemplate();
MonitorNotification notifcation = consumer.receiveBody("direct:end", MonitorNotification.class);
Assert.assertEquals("ExchangesTotal", notifcation.getObservedAttribute());
```

5.8. JNDI

JNDI インテグレーションは、以下のように JBoss EAP 固有の CamelContext によって提供されます。

```
InitialContext inictx = new InitialContext();
CamelContextFactory factory = inictx.lookup("java:jboss/camel/CamelContextFactory");
WildFlyCamelContext camelctx = factory.createCamelContext();
```

WildFlyCamelContext から、事前設定されたネーミングコンテキストを取得できます。

```
Context context = camelctx.getNamingContext();
context.bind("helloBean", new HelloBean());
```

その後、Camel ルートから参照できます。

```
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").beanRef("helloBean");
    }
});
camelctx.start();
```

5.9. JPA

JPA インテグレーションは Camel JPA コンポーネント によって提供されます。persistence.xml 設定ファイルを一部の JPA アノテーションが付けられたクラスとともに提供することにより、Camel JPA アプリケーションを開発できます。

5.9.1. persistence.xml の例

以下の例では、JBoss EAP standalone.xml 設定ファイル内で設定された JBoss EAP インメモリー ExampleDS データソースを使用できます。

```
<persistence version="2.0"
    xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
```

```
    <persistence-unit name="camel">
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
        <class>org.wildfly.camel.test.jpa.model.Customer</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>

</persistence>
```

5.9.2. JPA エンティティーの例

```
@Entity
@Table(name = "customer")
public class Customer implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Long getId() {
        return id;
    }

    public void setId(final Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

5.9.3. Camel JPA エンドポイント / ルート設定

JPA を設定したら、CDI を使用して EntityManager および UserTransaction インスタンスを RouteBuilder クラスまたはテストケースに注入できます。

```
@PersistenceContext
EntityManager em;

@Inject
UserTransaction userTransaction;
```

ここで、Camel ルートと JPA エンドポイントを設定します。

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext(getClass().getClassLoader());

EntityManagerFactory entityManagerFactory = em.getEntityManagerFactory();

// Configure a transaction manager
JtaTransactionManager transactionManager = new JtaTransactionManager();
transactionManager.setUserTransaction(userTransaction);
transactionManager.afterPropertiesSet();

// Configure the JPA endpoint to use the correct EntityManagerFactory and JtaTransactionManager
final JpaEndpoint jpaEndpoint = new JpaEndpoint();
jpaEndpoint.setCamelContext(camelctx);
jpaEndpoint.setEntityType(Customer.class);
jpaEndpoint.setEntityManagerFactory(entityManagerFactory);
jpaEndpoint.setTransactionManager(transactionManager);

camelctx.addRoutes(new RouteBuilder() {
@Override
public void configure() throws Exception {
    from("direct:start")
    .to(jpaEndpoint);
}
});

camelctx.start();
```

最後に、Customer エンティティーを direct:start エンドポイントに送信してから ExampleDS データソースをクエリーし、レコードが保存されていることを確認できます。

```
Customer customer = new Customer("John", "Doe");
ProducerTemplate producer = camelctx.createProducerTemplate();
producer.sendBody("direct:start", customer);

// Query the in memory database customer table to verify that a record was saved
CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
CriteriaQuery<Long> query = criteriaBuilder.createQuery(Long.class);
query.select(criteriaBuilder.count(query.from(Customer.class)));

long recordCount = em.createQuery(query).getSingleResult();

Assert.assertEquals(1L, recordCount);
```

第6章 CAMEL コンポーネント

本章では、サポートされる camel コンポーネントについて説明します。

6.1. CAMEL-ACTIVEMQ

Camel ActiveMQ インテグレーションは activemq コンポーネントによって提供されます。

コンポーネントは、組み込みまたは外部ブローカーと連携するように設定できます。Wildfly / EAP コンテナー管理接続プールおよび XA-Transaction サポートでは、ActiveMQ リソースアダプター をコンテナー設定ファイルに設定できます。

6.1.1. JBoss EAP ActiveMQ リソースアダプターの設定

ActiveMQ リソースアダプター rar ファイル をダウンロードします。以下の手順では、ActiveMQ リソースアダプターの設定方法を説明します。

1. JBoss EAP インスタンスを停止します。
2. リソースアダプターをダウンロードし、関連する JBoss EAP デプロイメントディレクトリーにコピーします。スタンドアロンモードの場合:

```
cp activemq-rar-5.11.1.rar ${JBOSS_HOME}/standalone/deployments/activemq-rar.rar
```

3. ActiveMQ アダプターの JBoss EAP リソースアダプターサブシステムを設定します。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
    <resource-adapters>
        <resource-adapter id="activemq-rar.rar">
            <archive>
                activemq-rar.rar
            </archive>
            <transaction-support>XATransaction</transaction-support>
            <config-property name="UseInboundSession">
                false
            </config-property>
            <config-property name="Password">
                defaultPassword
            </config-property>
            <config-property name="UserName">
                defaultUser
            </config-property>
            <config-property name="ServerUrl">
                tcp://localhost:61616?jms.rmIdFromConnectionId=true
            </config-property>
            <connection-definitions>
                <connection-definition class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory" jndi-
name="java:/ActiveMQConnectionFactory" enabled="true" pool-name="ConnectionFactory">
                    <xa-pool>
                        <min-pool-size>1</min-pool-size>
                        <max-pool-size>20</max-pool-size>
                        <prefill>false</prefill>
                        <is-same-rm-override>false</is-same-rm-override>
```

```
                </xa-pool>
            </connection-definition>
        </connection-definitions>
        <admin-objects>
            <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/HELLOWORLDMDBQueue" use-java-context="true" pool-
name="HELLOWORLDMDBQueue">
                <config-property name="PhysicalName">
                    HELLOWORLDMDBQueue
                </config-property>
            </admin-object>
            <admin-object class-name="org.apache.activemq.command.ActiveMQTopic" jndi-
name="java:/topic/HELLOWORLDMDBTopic" use-java-context="true" pool-
name="HELLOWORLDMDBTopic">
                <config-property name="PhysicalName">
                    HELLOWORLDMDBTopic
                </config-property>
            </admin-object>
        </admin-objects>
    </resource-adapter>
  </resource-adapters>
 </subsystem>
```

リソースアダプタのアーカイブファイル名が activemq-rar.rar とは異なる場合、アーカイブファイルの名前と一致するように、前述の設定の archive 要素の内容を変更する必要があります。

UserName および Password 設定プロパティーの値は、外部ブローカーの有効なユーザーのクレデンシャルと一致するように、選択する必要があります。

外部ブローカーによって公開される実際のホスト名およびポートに一致するように、ServerUrl 設定プロパティーの値を変更する必要がある場合があります。

4) JBoss EAP を起動します。すべてが正しく設定されていれば、JBoss EAP server.log 内に以下のようなメッセージが表示されます。

```
13:16:08,412 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1-5) JBAS010406:
Registered connection factory java:/AMQConnectionFactory`
```

6.1.2. Camel ルート設定

以下の ActiveMQ プロデューサーおよびコンシューマーの例は、ActiveMQ 組み込みブローカーと 'vm' トランスポートを使用します (よって、外部 ActiveMQ ブローカーが必要ないようにします)。

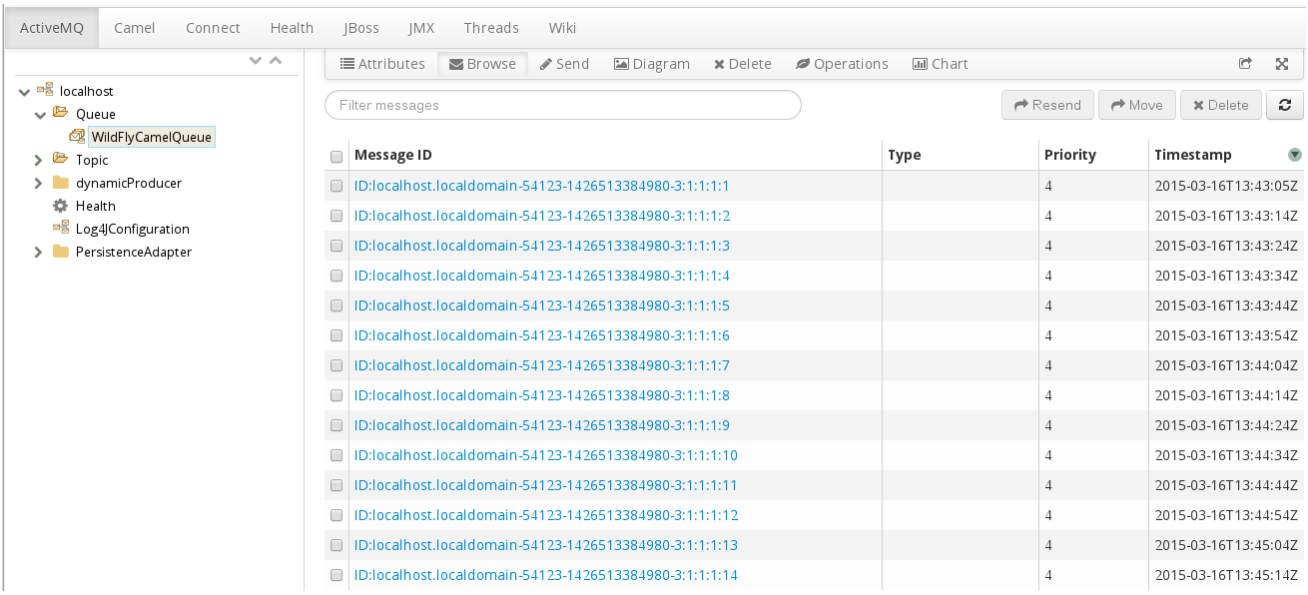
この例では、camel-cdi コンポーネントとともに CDI を使用します。JMS ConnectionFactory インスタンスは、JNDI ルックアップを介して Camel RouteBuilder に注入されます。

6.1.2.1. ActiveMQ プロデューサー

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Override
  public void configure() throws Exception {
```

```
    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
    .log("JMS Message sent");
  }
}
```

メッセージが WildFlyCamelQueue 宛先に追加されるたびに、ログメッセージがコンソールに出力されます。メッセージをキューに配置されていることを確認するには、Camel on EAP サブシステムによって提供される ../features/hawtio.md[Hawtio console,window=_blank] を使用できます。



6.1.2.2. ActiveMQ コンシューマー

ActiveMQ メッセージを使用するために、Camel RouteBuilder 実装はプロデューサーの例と似ています。

ActiveMQ エンドポイントが WildFlyCamelQueue 宛先からメッセージを消費すると、コンテンツはコンソールに記録されます。

```
@Override
public void configure() throws Exception {
  from("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
  .to("log:jms?showAll=true");
}
```

6.1.2.3. ActiveMQ トランザクション

6.1.2.3.1. ActiveMQ リソースアダプターの設定

XA トランザクションサポートや接続プールなどを活用するには、ActiveMQ リソースアダプターが必要です。

以下の XML スニペットは、JBoss EAP サーバーの XML 設定内でリソースアダプターがどのように設定されているかを示しています。**ServerURL** は組み込みブローカーを使用するように設定されていることに注意してください。接続ファクトリーは JNDI 名 **java:/ActiveMQConnectionFactory** にバインドされます。これは、次の RouteBuilder の例で検索されます。

最後に、queue1 と queue2 という名前の 2 つのキューが設定されます。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
  <resource-adapters>
    <resource-adapter id="activemq-rar.rar">
      ...
      <admin-objects>
        <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue1" use-java-context="true" pool-name="queue1pool">
          <config-property name="PhysicalName">queue1</config-property>
        </admin-object>
        <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue2" use-java-context="true" pool-name="queue2pool">
          <config-property name="PhysicalName">queue2</config-property>
        </admin-object>
      </admin-objects>
    </resource-adapter>
  </resource-adapters>
</subsystem>
```

6.1.2.4. トランザクションマネージャー

camel-activemq コンポーネントに
は、**org.springframework.transaction.PlatformTransactionManager** タイプのトランザクションマネージャーが必要です。そのため、最初にこの要件を満たす **JtaTransactionManager** を拡張する Bean を作成します。Bean には **@Named** アノテーションが付けられており、Camel Bean レジストリー内に Bean を登録できることに注意してください。また、JBoss EAP トランザクションマネージャーとユーザートランザクションインスタンスは CDI を使用して注入されることに注意してください。

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {

  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

6.1.2.5. トランザクションポリシー

次に、使用するトランザクションポリシーを宣言する必要があります。ここでも、**@Named** アノテーションを使用して Bean を Camel で利用可能にします。また、トランザクションマネージャーも、必要なトランザクションポリシーで **TransactionTemplate** を作成できるように注入されます。このインスタンスの **PROPAGATION_REQUIRED**。

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
```

```
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

6.1.2.6. ルートビルダー

これで、Camel RouteBuilder クラスを設定し、Camel ActiveMQ コンポーネントに必要な依存関係を注入できるようになります。リソースアダプターに設定した ActiveMQ 接続ファクトリーは、以前設定したトランザクションマネージャーとともに注入されます。

この例では、queue1 からメッセージが消費されるたびに queue2 という名前の別の JMS キューにルーティングされます。queue2 から消費されるメッセージにより、JMS トランザクションは rollback () DSL メソッドを使用してロールバックされます。これにより、元のメッセージがデッドレターキュー (DLQ) に配置されます。

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/ActiveMQConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Inject
  private CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
    activeMQComponent.setTransacted(false);
    activeMQComponent.setConnectionFactory(connectionFactory);
    activeMQComponent.setTransactionManager(transactionManager);

    getContext().addComponent("activemq", activeMQComponent);

      errorHandler(deadLetterChannel("activemq:queue:ActiveMQ.DLQ")
      .useOriginalMessage()
      .maximumRedeliveries(0)
      .redeliveryDelay(1000));

    from("activemq:queue:queue1F")
      .transacted("PROPAGATION_REQUIRED")
      .to("activemq:queue:queue2");

    from("activemq:queue:queue2")
      .to("log:end")
      .rollback();
  }
}
```

6.1.3. セキュリティー

JMS セキュリティーのセクション を参照してください。

6.1.4. GitHub のコード例

camel-activemq アプリケーション の例は GitHub で利用できます。

6.2. CAMEL-JMS

camel-jms、camel-sjms、および camel-sjms2 エンドポイントをリモート AMQ 7 ブローカーに接続する方法は、2 つサポートされています。

1. JBoss EAP『Configuring Messaging』の「Configuring the Artemis Resource Adapter to Connect to Red Hat JBoss AMQ 7 」の説明どおりに、pooled-connection-factory で remote-connector を設定します。

2. connection-factory での remote-connector の設定 の説明に従って connection-factory で remote-connector を設定します。

最初のオプションは、接続プールおよび XA トランザクションのサポートを提供するため、推奨される方法です。

永続サブスクライバーを使用するメッセージングのシナリオでは、JavaEE 7 仕様による制約があるため、pooled-connection-factory は JBoss EAP の Fuse 7.9 ではサポートされません。このようなシナリオでは、標準のプールされていない接続ファクトリーを設定することが推奨されます。

connection-factory での remote-connector の設定

1. リモートメッセージングサーバーを示す outbound-socket-binding を作成します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=localhost, port=61616)
```

2. ステップ 1 で作成した outbound-socket-binding を参照する remote-connector を作成します。

```
/subsystem=messaging-activemq/server=default/remote-connector=netty-remote-throughput:add(socket-binding=messaging-remote-throughput)
```

3. ステップ 2 で作成した remote-connector を参照する connection-factory を作成します。

```
/subsystem=messaging-activemq/server=default/connection-factory=simple-remote-artemis-connection-factory:add(entries=[java:/jms/RemoteJms],connectors=[netty-remote-throughput])
```

6.2.1. メッセージングブローカーおよびクライアント

概要

Fuse 7.9 にはデフォルトの内部メッセージングブローカーが含まれていませんが、外部 JMS ブローカーとインターフェースするように設計されています。

JBoss EAP の Fuse 7.9 は、「Configuring Messaging on JBoss EAP」で説明されているリソースアダプターを使用して外部メッセージングブローカーにアクセスします。

Fuse 7.9 on JBoss EAP のメッセージングで使用できる外部ブローカー、JCA アダプター、および Camel コンポーネントの組み合わせに関する詳細は、「Red Hat Fuse でサポートされる構成」を参照してください。

JMS を使用する Fuse on JBoss EAP を使用して外部ブローカーに接続する方法は、「camel-jms」 を参照してください。

camel-jms クイックスタート

クイックスタートは、Fuse on JBoss EAP で camel-jms コンポーネントを使用して JMS メッセージを生成および消費する方法を示すために提供されます。

このクイックスタートでは、Camel ルートは **EAP_HOME/standalone/data/orders** からファイルを消費し、コンテンツを **OrdersQueue** という名前のインメモリー ActiveMQ Artemis キューに配置します。別の Camel ルートは **OrdersQueue** の内容を使用し、**EAP_HOME/standalone/data/orders/processed** 内で個別の国ディレクトリーへの注文を並べ替えます。

アプリケーションがデプロイおよびアンデプロイされるときに、JMS **OrdersQueue** の作成および削除を処理する **OrdersQueue** CLI スクリプトが CLI コマンドによって作成および削除されます。これらのスクリプトは **EAP_HOME/quickstarts/camel-jms/src/main/resources/cli** ディレクトリー内にあります。

前提条件

このクイックスタートを実行するには、作業バージョンの Fuse 7.9 が必要です。

「Using JBoss AMQ for remote JMS Communication」の手順に従って、外部 AMQ 7 ブローカーに接続する必要もあります。その後、接続ファクトリーをデフォルトの接続ファクトリーに注入できます。

```
@Resource(mappedName = "java:jboss/RemoteJmsXA")
ConnectionFactory connectionFactory;
```

クイックスタートの設定

1. スタンドアロンモードで JBOSS EAP を起動します。
2. **EAP_HOME/quickstarts/camel/camel-jms** に移動します。
3. **mvn clean install -Pdeploy`** を入力し、クイックスタートをビルドおよびデプロイします。
4. http://localhost:8080/example-camel-jms にアクセスします。

「Orders Received」というタイトルのページが表示されます。サンプルアプリケーションに注文を送信すると、国ごとの注文のリストがこのページに表示されます。

クイックスタートを実行します。

EAP_HOME/quickstarts/camel/camel-jms/src/main/resources ディレクトリー内に、注文 XML ファイルのサンプルがいくつかあります。Camel は 5 秒ごとにランダムにファイルを選択し、処理のために **EAP_HOME/standalone/data/orders** にコピーします。

コンソールは、各注文に何が起こったかを詳細に説明するメッセージを出力します。出力は以下のようになります。

```
JmsConsumer[OrdersQueue]) Sending order to the UK
JmsConsumer[OrdersQueue]) Sending order to another country
JmsConsumer[OrdersQueue]) Sending order to the US
```

ファイルが使用されると、 http://localhost:8080/example-camel-jms/orders に戻ることができます。各国の受注数は 1 つ増えたはずです。

処理された注文はすべて以下の宛先に分割されます。

```
EAP_HOME/standalone/data/orders/processed/uk
EAP_HOME/standalone/data/orders/processed/us
EAP_HOME/standalone/data/orders/processed/other
```

アンデプロイ

例をアンデプロイするには、**EAP_HOME/quickstarts/camel/camel-jms** に移動し、**mvn clean -Pdeploy** を実行します。

6.3. CAMEL-MAIL

メールとの対話は、mail コンポーネントによって提供されます。

デフォルトでは、Camel は独自のメールセッションを作成し、これを使用してメールサーバーと対話します。JBoss EAP はすでにセキュアな接続、ユーザー名、パスワードの暗号化などに関連するすべてのサポートを提供しています。そのため、JBoss EAP 設定内でメールセッションを設定し、JNDI を使用して Camel エンドポイントに接続することが推奨されます。

6.3.1. JBoss EAP の設定

最初に、メールサーバーの JBoss EAP mail サブシステムを設定します。以下の例では、Google Mail IMAP および SMTP の設定を追加します。

追加の mail-session は、デフォルト「default」セッションの後に設定されます。

```
<subsystem xmlns="urn:jboss:domain:mail:2.0">
    <mail-session name="default" jndi-name="java:jboss/mail/Default">
      <smtp-server outbound-socket-binding-ref="mail-smtp"/>
    </mail-session>
```

```
    <mail-session debug="true" name="gmail" jndi-name="java:jboss/mail/gmail">
      <smtp-server outbound-socket-binding-ref="mail-gmail-smtp" ssl="true" username="your-username-here" password="your-password-here"/>
      <imap-server outbound-socket-binding-ref="mail-gmail-imap" ssl="true" username="your-username-here" password="your-password-here"/>
    </mail-session>
</subsystem>
```

mail-gmail-smtp と mail-gmail-imap の **outbound-socket-binding-ref** 値を設定できます。

次のステップで、これらのソケットバインディングを設定します。以下のように、**socket-binding-group** 設定にバインディングを追加できます。

```
<outbound-socket-binding name="mail-gmail-smtp">
  <remote-destination host="smtp.gmail.com" port="465"/>
```

```
</outbound-socket-binding>

<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="imap.gmail.com" port="993"/>
</outbound-socket-binding>
```

これにより、ポート 465 でホスト smtp.gmail.com に接続し、ポート 993 で imap.gmail.com に接続するようにメールセッションが設定されます。別のメールホストを使用している場合は、詳細は異なります。

6.3.2. POP3 設定

POP3 セッションを設定する必要がある場合、原則は上記の例で定義されるものと同じです。

```
<!-- Server configuration -->
<pop3-server outbound-socket-binding-ref="mail-pop3" ssl="true" username="your-username-here"
password="your-password-here"/>

<!-- Socket binding configuration -->
<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="pop3.gmail.com" port="993"/>
</outbound-socket-binding>
```

6.3.3. Camel ルート設定

6.3.3.1. メールプロデューサー

この例は、camel-cdi コンポーネントと CDI とともに SMTPS プロトコルを使用します。JBoss EAP 設定内で設定した Java メールセッションは、JNDI を介して Camel RouteBuilder に注入されます。

6.3.3.1.1. ルートビルダー SMTPS の例

GMail のメールセッションは、以前に設定した **jndi-name** 属性への参照とともに **@Resource** アノテーションを使用して Producer クラスに注入されます。これにより、camel-mail エンドポイント設定でメールセッションを参照できます。

```
public class MailSessionProducer {
    @Resource(lookup = "java:jboss/mail/greenmail")
    private Session mailSession;

    @Produces
    @Named
    public Session getMailSession() {
        return mailSession;
    }
}
```

```
public class MailRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:start")
```

```
        .to("smtps://smtp.gmail.com?session=#mailSession");
    }
}
```

メールを送信するには、ProducerTemplate を作成し、必要なメールヘッダーとともに適切なボディーを送信します。

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("To", "destination@test.com");
headers.put("From", "sender@example.com");
headers.put("Subject", "Camel on Wildfly rocks");

String body = "Hi,\n\nCamel on Wildfly rocks!.";

ProducerTemplate template = camelContext.createProducerTemplate();
template.sendBodyAndHeaders("direct:start", body, headers);
```

6.3.3.2. メールコンシューマー

IMAP MailEndpoint を使用してメールを受信できます。Camel ルート設定は以下のようになります。

```
public void configure() throws Exception {
   from("imaps://imap.gmail.com?session=#mailSession")
   .to("log:email");
}
```

6.3.4. セキュリティー

6.3.4.1. SSL 設定

JBoss EAP は、SSL / TLS を使用して Java メールセッションおよび関連するトランスポートを管理するように設定できます。メールセッションを設定する場合は、サーバータイプに SSL または TLS を設定できます。

- smtp-server
- imap-server
- pop-server

属性 **ssl="true"** または **tls="true"** を設定します。

6.3.4.2. パスワードのセキュリティー保護

設定ファイル内のパスワードにはクリアテキストを使用しないことが推奨されます。WildFly Vault を使用して機密データをマスクできます。

6.3.4.3. Camel のセキュリティー

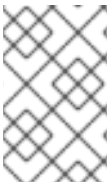
Camel エンドポイントセキュリティーに関するドキュメントは、mail コンポーネントガイドを参照してください。Camel には security summary ページもあります。

6.3.5. GitHub のコード例

メールの送受信を試すための camel-mail アプリケーション のサンプルは GitHub で利用できます。

6.4. CAMEL-REST

rest コンポーネントは、REST DSL やプラグインを REST トランスポートとして他の Camel コンポーネントに使用して、REST エンドポイントを定義できます。



注記

Camel on EAP サブシステムは、REST DSL と使用する camel-servlet コンポーネントおよび camel-undertow コンポーネントのみをサポートします。ただし、その他のコンポーネントの設定を試みると、サブシステムは機能しません。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        restConfiguration().component("servlet").contextPath("camel/rest").port(8080);
        rest("/hello").get("/{name}").to("direct:hello");
        from("direct:hello").transform(simple("Hello ${header.name}"));
    }
});
```

6.5. CAMEL-REST-SWAGGER

rest-swagger コンポーネントは、Swagger ドキュメントから REST プロデューサーを設定し、以下のような **RestProducerFactory** インターフェースを実装するコンポーネントに委譲できます。

- camel-http4
- camel-undertow

6.6. CAMEL-SQL

SQL コンポーネントを使用すると、JDBC クエリーを使用してデータベースと連携できます。このコンポーネントと JDBC コンポーネントの違いは、SQL ではクエリーはエンドポイントのプロパティーであり、クエリーに渡されるパラメーターとしてメッセージペイロードを使用する点です。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("sql:select name from information_schema.users?
dataSource=java:jboss/datasources/ExampleDS")
        .to("direct:end");
    }
});
```

注記

上記の JNDI データソースルックアップは、**DefaultCamelContext** を設定する際にのみ有効です。**CdiCamelContext** および **SpringCamelContext** の例は、以下を参照してください。

camel-cdi コンポーネントとともに使用すると、Jarkarta EE アノテーションはデータソースを Camel で利用できるようにすることができます。この例では、Camel が必要なデータソースを検出できるように、**@Named** アノテーションを使用しています。

```
public class DatasourceProducer {
    @Resource(lookup = "java:jboss/datasources/ExampleDS")
    DataSource dataSource;

    @Produces
    @Named("wildFlyExampleDS")
    public DataSource getDataSource() {
        return dataSource;
    }
}
```

これで、データソースは camel-sql エンドポイント設定の **dataSource** パラメーターから参照できるようになります。

```
@ApplicationScoped
@ContextName("camel-sql-cdi-context")
@Startup
public class CdiRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("sql:select name from information_schema.users?dataSource=wildFlyExampleDS")
        .to("direct:end");
    }
}
```

camel-spring を使用すると、ルート設定は以下のようになります。

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

    <jee:jndi-lookup id="wildFlyExampleDS" jndi-name="java:jboss/datasources/ExampleDS"/>

    <camelContext id="sql-spring-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="sql:select name from information_schema.users?dataSource=#wildFlyExampleDS"
/>
            <to uri="direct:end" />
        </route>
```

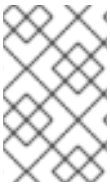
```
    </camelContext>

</beans>
```

6.6.1. Spring JDBC XML namespace のサポート

以下の Spring JDBC XML 設定のサポートはサポートされます。

jdbc:embedded-database

```
<jdbc:embedded-database id="datasource" type="H2">
  <jdbc:script location="db-schema.sql"/>
</jdbc:embedded-database>
```



注記

JBoss EAP ではネイティブサポートがあるため、H2 データベースのみがデフォルトでサポートされます。他の組み込みデータベースプロバイダーを使用する場合は、適切なデータベースドライバーをインストールする必要があります。

jdbc:initialize-database

```
<jdbc:initialize-database data-source="datasource">
  <jdbc:script location="classpath:db-init.sql"/>
</jdbc:initialize-database>
```

6.7. CAMEL-SOAP-REST-BRIDGE

シンプルな Camel ルートは REST 呼び出しをレガシー SOAP サービスにブリッジできます。クイックスタートサンプルは、**camel-soap-rest-bridge** コンポーネントと Camel の REST DSL を使用して、バックエンドの SOAP API サービスを公開する方法を示すために提供されます。

このクイックスタートでは、RH SSO を基盤とする REST エンドポイントと SOAP エンドポイントの両方にセキュリティーが関係します。フロントエンド REST API は OAuth および OpenID Connect で保護され、クライアントは「Resource Owner Password Credentials」OAuth2 モードを使用して RH SSO から JWT (JSON Web トークン) アクセストークンを取得します。クライアントは、このトークンを使用して REST エンドポイントにアクセスします。

ブリッジ Camel ルートでは、クライアントアイデンティティーは SecurityContext から伝播され、**camel-cxf producer** がバックエンド WS-SECURITY protected SOAP サービスと通信する際に、最初にこのクライアントアイデンティティーを使用して CXF STS サービス (Identity Provider として RH SSO を基盤とする) によって発行された SAML2 トークンを取得します 。SAML2 トークンは署名され、WS-SECURITY ヘッダーに追加され、バックエンド WS-SECURITY で保護された SOAP サービスはこの SAML2 トークンを検証します。

SOAP 呼び出しには、XSD スキーマ検証も含まれます。トークンの検証に成功すると、バックエンド SOAP サービスはリクエストを開始した REST クライアントにレスポンスを返します。

前提条件

1. JBoss EAP 7.3 以降のバージョンがインストールされている。
2. Apache Maven 3.3.x 以降のバージョンがインストールされている。

3. RH SSO 7.4 がインストールおよび設定されている。https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.4/html/getting_started_guide/installing-standalone_#installing-server-product のインストール手順に従います。

4. RH SSO EAP アダプターがインストールされている。https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.4/html/getting_started_guide/securing-sample-app_#installing-client-adapter のインストール手順に従います。

クイックスタートの設定

1. スタンドアロンモードで JBOSS EAP を起動します。

2. **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** に移動します。

3. **mvn clean install -Pdeploy** を入力し、クイックスタートをビルドおよびデプロイします。

4. RH-SSO の設定

   a. admin/admin を username/password として使用して、http://localhost:8180/auth から RH SSO 管理コンソールにログインします。

   b. **Add realm** をクリックします。

   c. **Select file** をクリックします。

   d. このサンプルフォルダーの ./src/main/resources/keycloak-config/realm-export-new.json を選択します。これは、事前定義されたこのサンプルに必要な realm/client/user/role をインポートします。

   e. **Create** をクリックします。

Fuse on EAP のクイックスタートサンプル

クイックスタートの実行とテストケースの結果に関する追加情報が含まれるこのクイックスタートサンプルは、Fuse on EAP インストールの **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** ディレクトリーで入手できます。

アンデプロイ

サンプルをアンデプロイするには、**EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** ディレクトリーに移動し、**mvn clean -Pdeploy** を実行します。

6.8. コンポーネントの追加

追加の Camel コンポーネントにサポートを追加するのは簡単です。

modules.xml 定義の追加

modules.xml 記述子は、コンポーネントのクラスローディング動作を定義します。**modules/system/layers/fuse/org/apache/camel/component** でコンポーネントの jar とともに配置する必要があります。モジュール依存関係は、直接コンパイル時の依存関係に対して設定する必要があります。

以下は camel-ftp コンポーネントの例になります。

```
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel.component.ftp">
```

```
  <resources>
    <resource-root path="camel-ftp-2.14.0.jar" />
  </resources>
  <dependencies>
    <module name="com.jcraft.jsch" />
    <module name="javax.xml.bind.api" />
    <module name="org.apache.camel.core" />
    <module name="org.apache.commons.net" />
  </dependencies>
</module>
```

WildFly ですでに利用できるモジュールを複製せず、再利用できるようにしてください。

コンポーネントへの参照の追加

任意の JavaEE デプロイメントがこのモジュールをデフォルトで認識できるようにするには、**modules/system/layers/fuse/org/apache/camel/component/main/module.xml** への参照を追加します。

```
<module xmlns="urn:jboss:module:1.3" name="org.apache.camel.component">
  <dependencies>
    ...
    <module name="org.apache.camel.component.ftp" export="true" services="export"/>
  </dependencies>
</module>
```

第7章 セキュリティー

JBoss EAP のセキュリティーは膨大なトピックです。JBoss EAP と Camel の両方には、設定、エンドポイント、およびペイロードのセキュリティーを保護するための標準的な方法があり、文書化されています。

7.1. HAWTIO のセキュリティー

Hawtio コンソールの保護は、次の手順で実行できます。

1.システムプロパティーを standalone.xml に追加します。

```
<system-properties>
    <property name="hawtio.authenticationEnabled" value="true" />
    <property name="hawtio.realm" value="hawtio-domain" />
</system-properties>
```

2.security サブシステム内に Hawtio のセキュリティーレルムを追加します。

```
<security-domain name="hawtio-domain" cache-type="default">
    <authentication>
        <login-module code="RealmDirect" flag="required">
            <module-option name="realm" value="ManagementRealm"/>
        </login-module>
    </authentication>
</security-domain>
```

3.管理ユーザーを設定します。

```
$JBOSS_HOME/bin/add-user.sh -u someuser -p s3cret
```

http://localhost:8080/hawtio にアクセスし、管理ユーザーに設定されたクレデンシャルを使用して認証します。

7.2. JAX-RS セキュリティー

以下のトピックでは、JAX-RS エンドポイントをセキュアにする方法を説明します。

- WildFly HTTP basic authentication
- Security Realms & SSL
- Securing EJBs

7.3. JAX-WS セキュリティー

以下のトピックでは、JAX-WS エンドポイントをセキュアにする方法を説明します。

- WildFly HTTP basic authentication
- WS-Security
- CXF Security

- Security Realms & SSL
- Securing EJBs

7.4. JMS セキュリティー

以下のトピックでは、JMS エンドポイントのセキュリティーを保護する方法を説明します。

- ActiveMQ Artemis のセキュリティードキュメント
- Security settings for ActiveMQ Artemis addresses and JMS destinations
- ActiveMQ Artemis security domain configuration
- ActiveMQ Security

さらに、Camel のルートポリシーの概念を使用して、JBoss EAP セキュリティーシステムと統合できます。

7.5. ルートポリシー

Camel は、JBoss EAP セキュリティーシステムとの統合に使用できる RoutePolicies の概念をサポートしています。現在、セキュリティーインテグレーションでサポートされる 2 つのシナリオがあります。

7.5.1. Jarkarta EE への Camel 呼び出し

Camel ルートがセキュアな Jarkarta EE コンポーネントへ呼び出されると、クライアントとして機能し、呼び出しに関連する適切なクレデンシャルを提供する必要があります。

以下のように、ルートに **ClientAuthorizationPolicy** を付けます。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .policy(new ClientAuthorizationPolicy())
      .to("ejb:java:module/AnnotatedSLSB?method=doSelected");
   }
});
```

これは、Camel メッセージ処理の一部として認証および承認を実行しません。代わりに、CamelExchange に付属するクレデンシャルを EJB3 レイヤーへの呼び出しに関連付けます。

メッセージコンシューマーを呼び出すクライアントは、以下のように AUTHENTICATION ヘッダーに適切なクレデンシャルを提供する必要があります。

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION, subject,
String.class);
```

認証および承認は Jarkarta EE 層で行われます。

7.5.2. Camel Route のセキュリティー保護

Camel Route をセキュアにするには、**DomainAuthorizationPolicy** をルートに関連付けることができます。このポリシーは、「Role2」の指定のセキュリティードメインおよび承認に対する正常な認証を必要とします。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .policy(new DomainAuthorizationPolicy().roles("Role2"))
        .transform(body().prepend("Hello "));
    }
});
camelctx.start();
```

この場合でも、メッセージコンシューマーを呼び出すクライアントは、以下のように AUTHENTICATION ヘッダーに適切なクレデンシャルを提供する必要があります。

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION, subject,
String.class);
```

7.6. CXF JAX-WS クイックスタートのデプロイ

この例は、Red Hat Fuse on EAP で camel-cxf コンポーネントを使用して、Elytron セキュリティードメインによってセキュア化された JAX-WS Web サービスを作成および消費しています。Elytron は、EAP 7.1 より利用できる新しいセキュリティーフレームワークです。このクイックスタートでは、Camel ルートはダイレクトエンドポイントからメッセージペイロードを取得し、CXF プロデューサーエンドポイントに渡します。プロデューサーはペイロードを使用して、BASIC HTTP 認証によってセキュア化された CXF JAX-WS Web サービスに引数を渡します。

前提条件

- Maven がインストールされ、設定されていることを確認します。
- Red Hat Fuse と共にアプリケーションサーバーがインストールされ、設定されていることを確認します。

手順

1. **JBOSS_HOME** 環境変数を設定して、アプリケーションサーバーインストールのルートディレクトリーを参照します。
   - Linux の場合

     ```
     export JBOSS_HOME=...
     ```

- Windows の場合:

```
set JBOSS_HOME=...
```

2. **add-user** スクリプトを使用して、新しいサーバーアプリケーションのユーザーおよびグループを作成します。

- Linux の場合

```
${JBOSS_HOME}/bin/add-user.sh -a -u testUser -p testPassword1+ -g testRole
```

- Windows の場合:

```
%JBOSS_HOME%\bin\add-user.bat -a -u testUser -p testPassword1+ -g testRole
```

3. スタンドアロンモードでアプリケーションサーバーを起動します。

- Linux の場合

```
${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml
```

- Windows の場合:

```
%JBOSS_HOME%\bin\standalone.bat -c standalone-full.xml
```

このプロジェクトの **webapp/WEB-INF** ディレクトリーの **jboss-web-xml** および **web.xml** ファイルは、アプリケーションセキュリティードメイン、セキュリティーロール、および制約を設定します。

4. プロジェクトをビルドしてデプロイします。

```
mvn install -Pdeploy
```

このコマンドは、セキュリティードメインおよびその他の管理オブジェクトを作成する CLI スクリプト **configure-basic-security.cli** も呼び出します。

5. **http://localhost:8080/example-camel-cxf-jaxws-secure/** を参照します。
Send A Greeting というページが表示されます。この UI を使用すると、すでに起動しているテスト **greeting** Web サービスと対話できます。サービス WSDL は **http://localhost:8080/webservices/greeting-security-basic?wsdl** で利用できます。

greet という名前のサービスオペレーションが 1 つあり、**message** と **name** という 2 つの String パラメータを取ります。Web サービスを呼び出すと、これらの値がリンクされている応答が返されます。

Camel Secure CXF JAX-WS クイックスタートのテスト

1. http://localhost:8080/example-camel-cxf-jaxws-secure/ を参照します。

2. **Send A Greeting** Web フォームで **message** および **name** をテキストフィールドに入力し、**send** ボタンを押します。
入力された情報は、UI の挨拶文として表示されます。**CamelCxfWsServlet** は、Web UI からの POST 要求を処理します。パラメーター値からメッセージおよび名前を取得し、オブジェクトアレイを構築します。このオブジェクト配列は、**direct:start** エンドポイントに送信されるメッ

セージペイロードです。**ProducerTemplate** はメッセージペイロードを Camel に送信します。**The direct:start** エンドポイントはオブジェクトアレイを **cxf:bean** Web サービスプロデューサーに渡します。Web サービスの応答は、**CamelCxfWsServlet** が Web UI に挨拶文を表示するために使用されます。完全な Camel ルートは **src/main/webapp/WEB-INF/cxfws-security-camel-context.xml** ファイルで確認できます。

クイックスタートのアンデプロイ

1. 以下のコマンドを実行してクイックスタートをアンデプロイします。

```
mvn clean -Pdeploy
```