Red Hat Fuse 7.5

Spring Boot へのデプロイ

スタンドアロンモードでの Spring Boot アプリケーションのビルドおよび実行

Last Updated: 2023-07-25

Red Hat Fuse 7.5 Spring Boot へのデプロイ

スタンドアロンモードでの Spring Boot アプリケーションのビルドおよび実行

法律上の通知

概要

本ガイドでは、Jar ファイルとしてパッケージ化され、JVM で直接実行 (スタンドアロンモード) される Spring Boot アプリケーションをビルドする方法について説明します。

目次

第1章 SPRING BOOT スタンドアロンの使用

1.1. SPRING BOOT スタンドアロンデプロイメントモード

スタンドアロンデプロイメントモードでは、Spring Boot アプリケーションは Jar ファイルとしてパッケージ化され、Java 仮想マシン(JVM)内で直接実行されます。つまり、Spring Boot アプリケーションは、**java** コマンドと **-jar** オプションを提供すれば直接実行できます。以下に例を示します。

```
java -jar SpringBootApplication.jar
```

Spring Boot は、実行可能な Jar のメインクラスを提供します。

アプリケーションをパッケージ化および実行するこの方法は、サービスが最小限の要件でパッケージ化されるマイクロサービスの概念と一致しています。コンテナーも最小で、JVM 自体になります。

Fuse で Spring Boot スタンドアロンアプリケーションをビルドするには、以下が必要です。

- **Fuse BOM (Bill of Materials)** Red Hat Maven リポジトリーから、厳選された依存関係のセットを定義します。BOM は、Maven の **依存関係管理** メカニズムを利用して、適切なバージョンの Maven 依存関係を定義します。
  Fuse BOM で定義された依存関係のみが Red Hat によってサポートされることに注意してください。

- **Spring Boot Maven プラグインは、Maven で**スタンドアロン Spring Boot アプリケーションのビルドプロセスを実装します。このプラグインは、Spring Boot アプリケーションを実行可能な Jar ファイルとしてパッケージ化します。

1.1.1. Spring Boot 2 へのデプロイ

Spring Boot 1 に加えて、Spring Boot 2 にデプロイするオプションもあります。

注記

OpenShift のデプロイメントのモードに関する詳細は、Fuse on OpenShift ガイド を参照してください。

注記

Spring Boot 1 と Spring Boot 2 の相違点はすべて、Spring Boot の Migration Guide と Spring Boot 2 リリースノート を参照してください。

1.1.1.1. Spring Boot 2 の新しい Camel コンポーネント

Spring Boot 2 は Camel バージョン 2.**23** をサポートしているため、Spring Boot 1 では利用できない新しい camel コンポーネントの一部をサポートします。

Spring Boot 2 の新しい Camel コンポーネント

- as2-component
- aws-iam-component
- fhir-component

- google-calendar-stream-component
- google-mail-stream-component
- google-sheets-component
- google-sheets-stream-component
- ipfs コンポーネント
- kubernetes-hpa-component
- kubernetes-job-component
- micrometer-component
- mybatis-bean-component
- nsq-component
- rxjava2
- service-component
- spring-cloud-consul
- spring-cloud-zookeeper
- testcontainers-spring
- testcontainers
- web3j-component

第2章 FUSE ブースターの使用

Red Hat Fuse では、Fuse アプリケーションや便利なコンポーネントを使用するために、以下のブースターが提供されます。

- 「Circuit Breaker ブースター」 : 分散アプリケーションがネットワーク接続の中断やバックエンドサービスの一時的な利用停止に対処できるようにする例。
- 「外部化設定ブースター」 - Apache Camel ルートの設定を外部化する方法の例。
- 「REST API ブースター」 - HTTP プロトコルを使用して、Apache Camel によって公開されるリモートサービスと対話するメカニズムを導入した例。

2.1. 前提条件

ブースターデモをビルドして実行するには、以下の前提条件をインストールします。

- サポートされるバージョンの Java Developer Kit (JDK)。詳細は Red Hat Fuse でサポートされる設定 を参照してください。
- Apache Maven 3.3.x 以上。Maven の Download ページを参照してください。

2.2. ブースタープロジェクトの生成

Fuse ブースタープロジェクトは、スタンドアロンアプリケーションの実行を手助けする開発者向けのプロジェクトです。ここでは、ブースタープロジェクトの 1 つである Circuit Breaker ブースターの生成手順を説明します。この演習では、Fuse on Spring Boot の便利なコンポーネントを使用します。

Netflix/Hystrix サーキットブレーカーを使用すると、ネットワーク接続の中断やバックエンドサービスの一時的な利用停止に分散アプリケーションが対処できるようになります。サーキットブレーカーパターンの基本概念は、バックエンドサービスが一時的に利用できなくなった場合に備えて、依存するサービスの損失を自動的に検出し、代替動作をプログラムで作成できるようにすることです。

Fuse サーキットブレーカーブースターは、次の 2 つの関連サービスで構成されます。

- 呼び名を返すバックエンドサービスである **name** サービス。
- 名前を取得する **name** サービスを呼び出し、文字列 **Hello, NAME** を返すフロントエンドサービスである **greetings** サービス。

このブースターデモンストレーションでは、Hystrix サーキットブレーカーは **greetings** サービスと **name** サービスとの間に挿入されます。バックエンドの **name** サービスが利用できなくなると、**name** サービスが再起動するまでの間に **greetings** サービスはブロックされず、代替動作にフォールバックして即座にクライアントに応答します。

前提条件

- Red Hat Developer Platform にアクセスできる。
- サポートされるバージョンの Java Developer Kit (JDK) を持っている。詳細は Red Hat Fuse でサポートされる設定 を参照してください。
- Apache Maven 3.3.x 以上が必要です。

手順

1. https://developers.redhat.com/launch に移動します。
2. **START** をクリックします。
   ランチャーウィザードによって、Red Hat アカウントにログインするよう要求されます。
3. **Log in or register**ボタンをクリックし、ログインします。
4. **Launcher** ページで **Deploy an Example Application**ボタンをクリックします。
5. **Create Example Application**ページで **Create Example Application as**フィールドに名前 **fuse-circuit-breaker** を入力します。
6. **Select an Example** をクリックします。
7. **Example** ダイアログで、**Circuit Breaker** オプションを選択します。**Select a Runtime** ドロップダウンメニューが表示されます。
   a. **Select a Runtime** ドロップダウンメニューで **Fuse** を選択します。
   b. バージョンのドロップダウンメニューで **7. 5 (Red Hat Fuse) を選択します。2.21.2 (Community)** バージョンは選択しないでください。
   c. **Save** をクリックします。
8. **Create Example Application**ページで **Download** をクリックします。
9. **Your Application is Ready** ダイアログが表示されたら、**Download.zip** をクリックします。ブラウザーが生成されたブースタープロジェクト (ZIP ファイルとしてパッケージ) をダウンロードします。
10. アーカイブユーティリティーを使用して、生成されたプロジェクトをローカルファイルシステムの任意の場所に展開します。

2.2.1. Circuit Breaker ブースター

Netflix/Hystrix サーキットブレーカーコンポーネントは、ネットワーク接続の中断や、バックエンドサービスの一時的な利用停止に分散アプリケーションが対応できるようにします。サーキットブレーカーパターンの基本概念は、バックエンドサービスが一時的に利用できなくなった場合に、依存するサービスの損失が自動的に検出され、代替動作をプログラムで作成できることです。

Fuse サーキットブローカーブースターは 2 つの関連サービスで設定されます。

- 対象の名前を返す **name サービス**。
- 名前を取得するために name サービスを呼び出し、文字列 **Hello, NAME** を返す **greetings サービス**。

このデモンストレーションでは、Hystrix サーキットブレーカーは greetings サービスと name サービスとの間に挿入されます。name サービスが利用できなくなると、greetings サービスは name サービスが再起動するまでの間にブロックまたはタイムアウトせずに、代替動作にフォールバックして即座にクライアントに応答することができます。

2.2.1.1. サーキットブレーカーブースターのビルドと実行

Circuit Breaker ミッションの 「ブースタープロジェクトの生成」 の手順に従って、Circuit Breaker ブースタープロジェクトをビルドして実行します。

1. シェルプロンプトを開き、Maven を使用してコマンドラインからプロジェクトをビルドします。

```
cd PROJECT_DIR
mvn clean package
```

2. 新しいシェルプロンプトを開き、以下のように name サービスを起動します。

```
cd name-service
mvn spring-boot:run -DskipTests -Dserver.port=8081
```

Spring Boot が起動すると、以下のような出力が表示されます。

```
...
2017-12-08 15:44:24.223  INFO 22758 --- [           main]
o.a.camel.spring.SpringCamelContext      : Total 1 routes, of which 1 are started
2017-12-08 15:44:24.227  INFO 22758 --- [           main]
o.a.camel.spring.SpringCamelContext      : Apache Camel 2.20.0 (CamelContext: camel-1)
started in 0.776 seconds
2017-12-08 15:44:24.234  INFO 22758 --- [           main]
org.jboss.fuse.boosters.cb.Application   : Started Application in 4.137 seconds (JVM running
for 4.744)
```

3. 新しいシェルプロンプトを開き、以下のように greetings サービスを起動します。

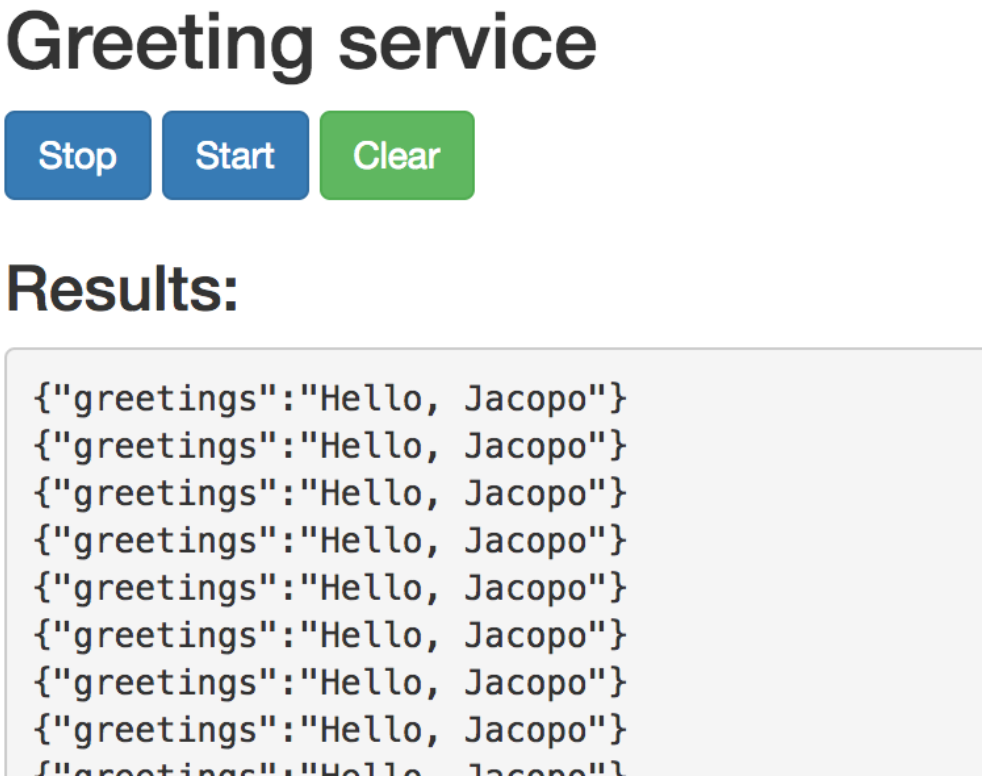
```
cd greetings-service
mvn spring-boot:run -DskipTests
```

Spring Boot が起動すると、以下のような出力が表示されます。

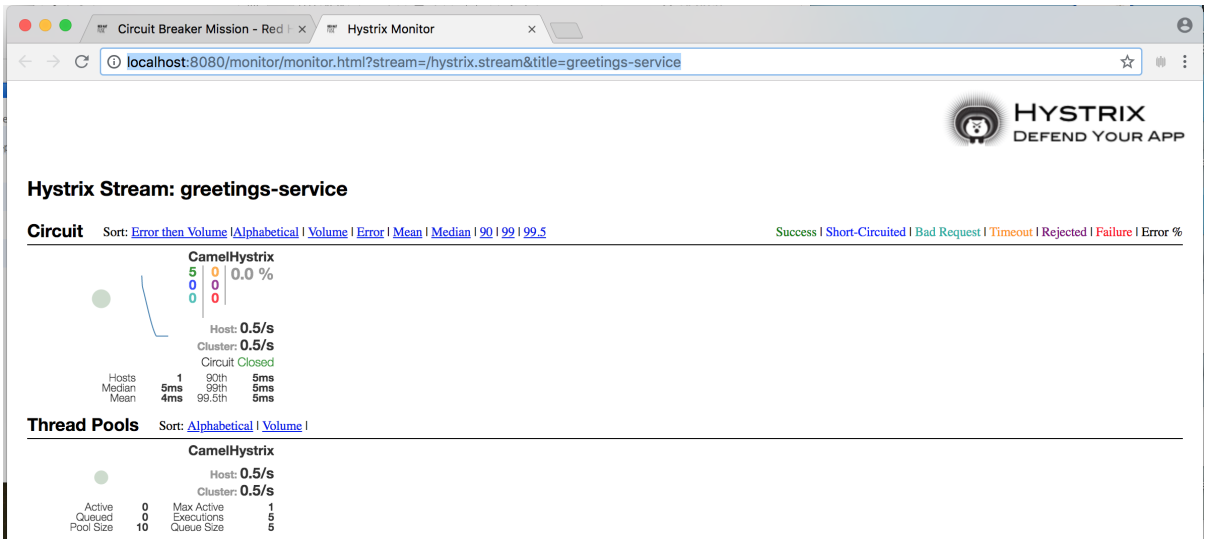
```
...
2017-12-08 15:46:58.521  INFO 22887 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2017-12-08 15:46:58.524  INFO 22887 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-12-08 15:46:58.536  INFO 22887 --- [           main]
org.jboss.fuse.boosters.cb.Application   : Started Application in 6.263 seconds (JVM running
for 6.819)
```

greetings サービスは、URL **http://localhost:8080/camel/greetings** で REST エンドポイントを公開します。

4. http://localhost:8080 にアクセスします。
このページを開くと、Greeting Service が呼び出されます。



このページには、サーキットブレーカーの状態を監視する Hystrix ダッシュボードへのリンクも提供されます。



5. Camel Hystrix によって提供されるサーキットブレーカー機能を実証するには、name サービスが実行されているシェルプロンプトウインドウで **Ctrl+C** を押して、バックエンド name サービスを中止します。
これで name サービスが利用できなくなるため、呼び出されたときに greetings サービスがハングしないよう、サーキットブレーカーが作動します。

6. Hystrix Monitor ダッシュボードおよび Greeting Service の出力で変更を確認します。

2.2.2. 外部化設定ブースター

Externalized Configuration (外部化設定) ブースターは、Apache Camel ルートの設定を外部化する方法の例を提供します。Spring Boot スタンドアロンデプロイメントでは、設定データは **application.properties** ファイルに保存されます。

注記

Fuse on OpenShift デプロイメントでは、設定データは ConfigMap オブジェクトに保存されます。

2.2.2.1. Externalized Configuration ブースターのビルドおよび実行

Externalized Configuration ミッションの 「ブースタープロジェクトの生成」 の手順に従った後、以下のステップに従って Externalized Configuration ブースターをローカルマシンのスタンドアロンプロジェクトとしてビルドおよび実行します。

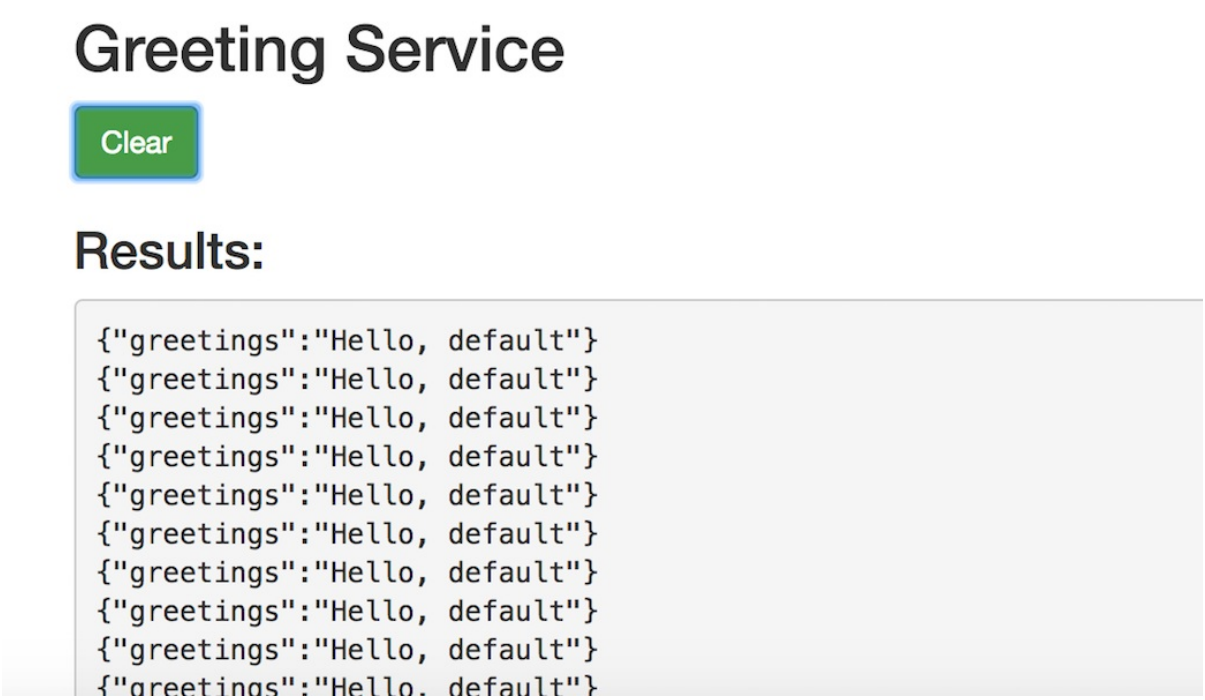
1. プロジェクトをダウンロードし、ローカルファイルシステムでアーカイブを展開します。
2. プロジェクトをビルドします。

```
cd PROJECT_DIR
mvn clean package
```

3. サービスを実行します。

```
mvn spring-boot:run
```

4. Web ブラウザーで http://localhost:8080 を開きます。このページは、5 秒ごとに Greeting Service を呼び出します。Greetings サービスは、**target/classes/application.properties** ファイルから **booster.nameToGreetvalue** を使用して応答します。
Greeting Service は default へのグリーティングを 5 秒ごとに出力します。



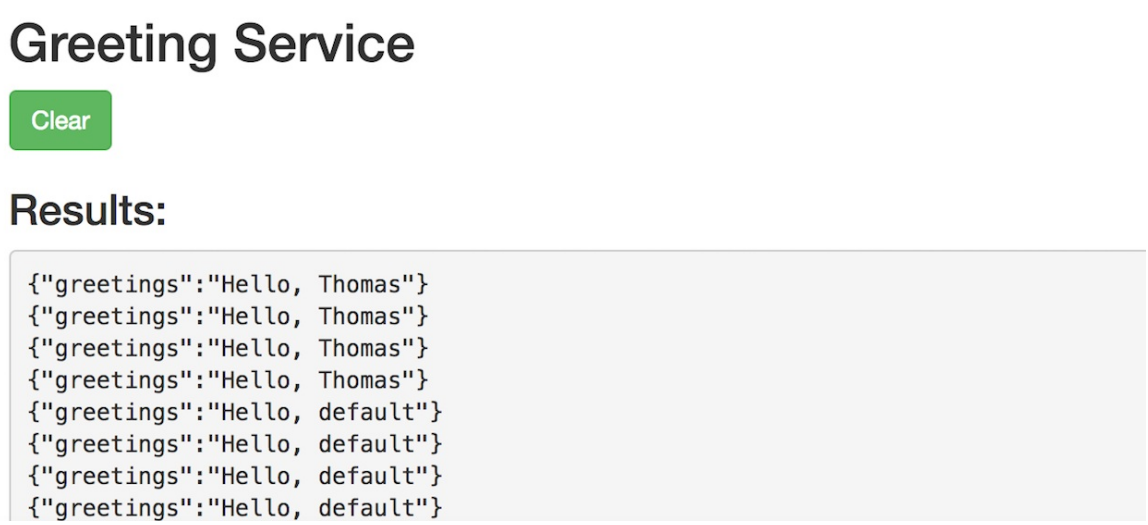
5. **booster.nameToGreet** の値を変更します。
   a. エディターで **target/classes/application.properties** ファイルを開きます。
   b. **booster.nameToGreetvalue** の値を、**default** から他の値 (例: **Thomas**) に変更します。

```
booster.nameToGreetvalue=Thomas
```

6. ターミナルウィンドウで **CTRL+C** を押してサービスを停止します。
7. サービスを再実行します。

```
mvn spring-boot:run
```

8. Web ブラウザーで http://localhost:8080 ページに戻り、Greeting Service の Results ウィンドウで変更された値を確認します。



2.2.3. REST API ブースター

REST API Level 0 のミッションでは、REST フレームワークを使用して、HTTP 経由でリモートプロシージャー呼び出しエンドポイントにビジネスオペレーションをマッピングする方法を示します。このミッションは、Richardson Maturity Model の Level 0 に該当します。

このブースターは、HTTP プロトコルを使用して Apache Camel によって公開されるリモートサービスと対話するメカニズムを導入します。この Fuse ブースターを使用すると、迅速に REST API のプロトタイプを作成し、柔軟に REST API を設定することができます。

このブースターを使用して、以下を行います。

- **camel/greetings/{name}** エンドポイントで HTTP GET 要求を実行します。このリクエストは、ペイロード **Hello, $name!** を使用して JSON 形式の応答を生成します (**$name** は HTTP GET リクエストからの URL パラメーターの値に置き換えられます)。
- URL **{name}** パラメーターの値を変更すると、変更後の値が応答に反映されます。
- REST API の Swagger ページを表示します。

2.2.3.1. REST API ブースターのビルドおよび実行

REST API ミッションの 「ブースタープロジェクトの生成」 の手順に従って、REST API ブースターをローカルマシンでスタンドアロンプロジェクトとしてビルドおよび実行します。

1. プロジェクトをダウンロードし、ローカルファイルシステムでアーカイブを展開します。
2. プロジェクトをビルドします。

```
cd PROJECT_DIR
mvn clean package
```

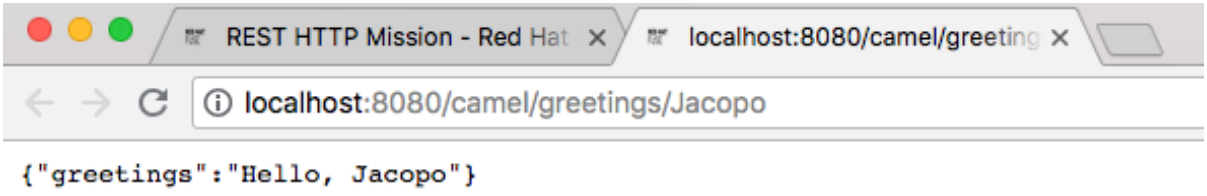
3. サービスを実行します。

```
mvn spring-boot:run
```

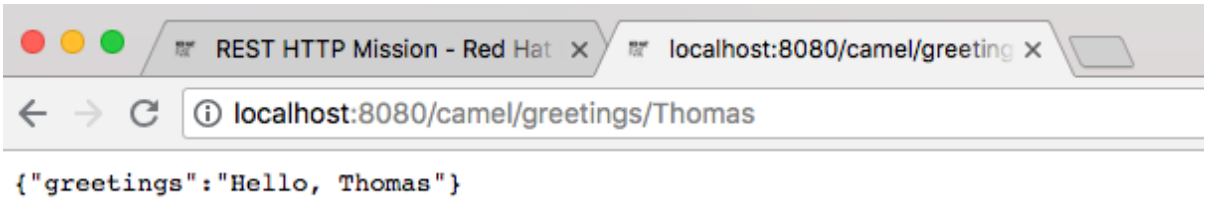
4. Web ブラウザーで http://localhost:8080 を開きます。

5. HTTP GET リクエストの例を実行するには、**camel/greetings/{name}** ボタンをクリックします。
localhost:8080/camel/greetings/Jacopo URL で新しい Web ブラウザーウインドウが開きます。URL **{name}** パラメーターのデフォルト値は **Jacopo** です。

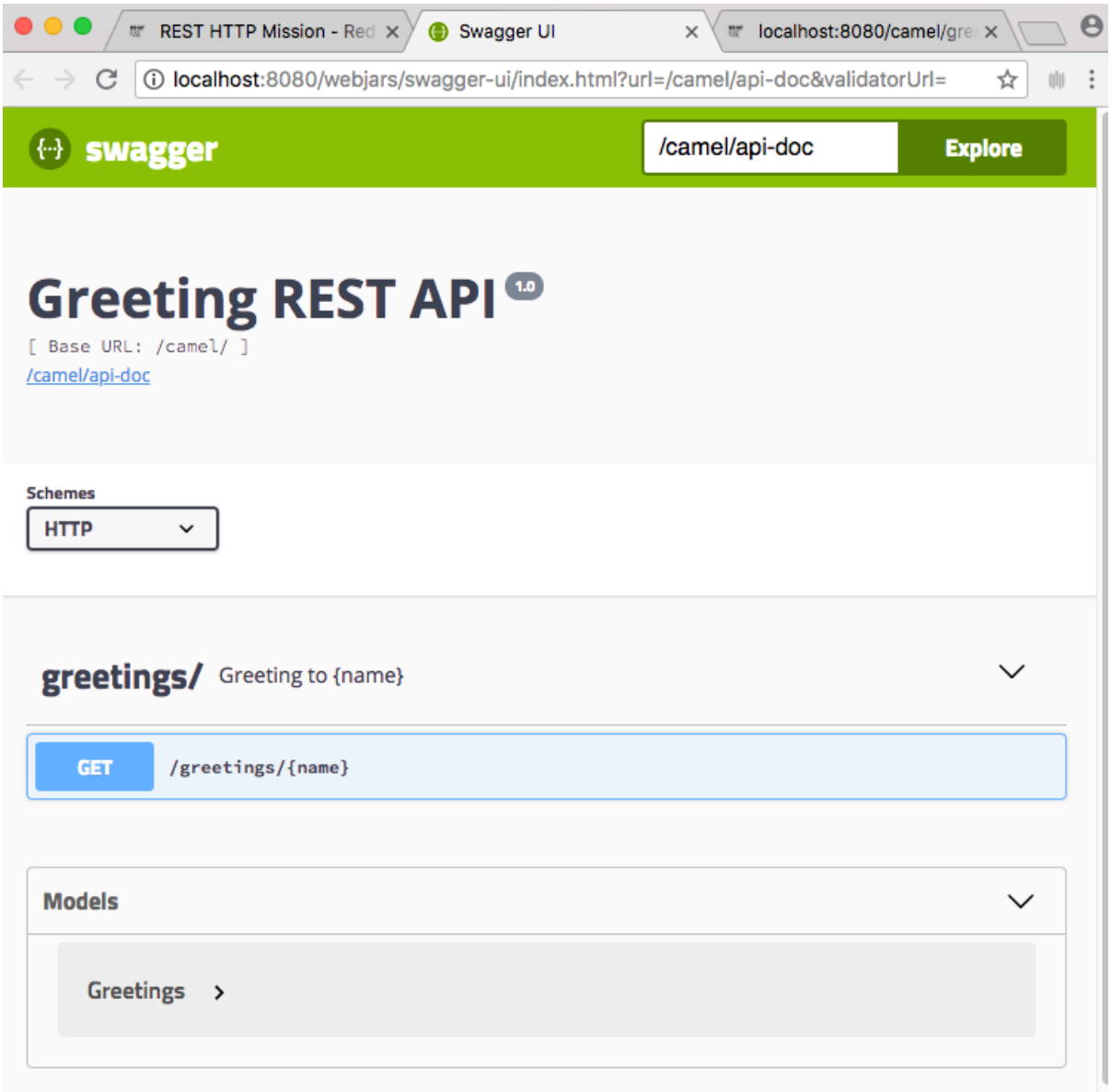
   ブラウザーウインドウに JSON 応答が表示されます。



6. **{name} ' パラメーターの値を変更するには、URL を変更します。たとえば、名前を Thomas に変更するには、URL 'localhost:8080/camel/greetings/Thomas** を使用します。
ブラウザーウインドウに更新された JSON 応答が表示されます。



7. REST API の Swagger ページを表示するには、API Swagger ページボタンをクリックします。
ブラウザーウィンドウに API swagger ページが表示されます。


REST HTTP Mission - Red
Swagger UI
localhost:8080/camel/gre
localhost:8080/webjars/swagger-ui/index.html?url=/camel/api-doc&validatorUrl=
swagger
/camel/api-doc
Explore
Greeting REST API
1.0
[Base URL: /camel/]
/camel/api-doc
Schemes
HTTP
greetings/ Greeting to {name}
GET
/greetings/{name}
Models
Greetings

第3章 MAVEN でのビルド

Fuse で Spring Boot のアプリケーションを開発する場合、Apache Maven ビルドツールを使用して、ソースコードを Maven プロジェクトとして構築することが標準的な方法です。すぐに開発できるようにするため、Fuse には Maven クイックスタートが提供されています。また、多くの Fuse ビルドツールは Maven プラグインとして提供されています。このため、Fuse の Spring Boot プロジェクトのビルドツールとして Maven を採用することが強く推奨されます。

3.1. MAVEN プロジェクトの生成

Fuse には、Maven アーキタイプを基にした複数の Spring Boot アプリケーションが提供され、これらを使用して Spring Boot アプリケーションの最初の Maven プロジェクトを生成できます。さまざまな Maven アーキタイプの場所情報とバージョンを把握する必要をなくすため、Fuse はスタンドアロン Spring Boot プロジェクトの Maven プロジェクトを生成するためのツールを提供します。

3.1.1. developers.redhat.com/launch のプロジェクトジェネレーター

Fuse で Spring Boot スタンドアロンを使い始める最も簡単な方法は、developers.redhat.com/launch にアクセスし、Spring Boot スタンドアロンランタイムの手順に従って、新しい Maven プロジェクトを生成することです。画面の指示に従うと、ローカルにビルドおよび実行できる完全な Maven プロジェクトが含まれるアーカイブファイルをダウンロードするように指示されます。

3.1.2. Developer Studio の Fuse ツールウィザード

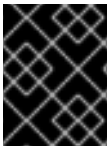
また、Fuse Tooling が含まれる Red Hat JBoss Developer Studio をダウンロードおよびインストールすることもできます。**Fuse New Integration Project** ウィザードを使用すると、新しい Spring Boot スタンドアロンプロジェクトを生成し、Eclipse ベースの IDE 内で開発を継続できます。

3.2. SPRING BOOT の BOM に依存します。

最初の Spring Boot プロジェクトを作成およびビルドした後、コンポーネントをすぐ追加したくなるでしょう。しかし、プロジェクトに追加する Maven 依存関係のバージョンはどのように判断したらよいのでしょうか。最も簡単な方法は、すべてのバージョンの依存関係を自動的に定義する、BOM (Bill of Materials) ファイルを使用することです。これは推奨される方法でもあります。

3.2.1. Spring Boot 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。



重要

使用している Spring Boot のバージョン (Spring Boot 1 または Spring Boot 2) に適した Fuse BOM が使用されているようにしてください。

Spring Boot 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。

3.2.1.1. BOM ファイルの組み込み

Maven プロジェクトに BOM ファイルを組み込むには、以下の Spring Boot 2 および Spring Boot 1 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。

- Spring Boot 2 の BOM
- Spring Boot 1 の BOM

```
<?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>7.5.0.fuse-sb2-750029-redhat-00003</fuse.version>
  </properties>

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

```
<?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>7.5.0.fuse-750029-redhat-00002</fuse.version>
  </properties>

  <dependencyManagement>
    <dependencies>
```

```
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-springboot-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-hystrix** 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-hystrix-starter</artifactId>
</dependency>
```

Camel アーティファクト ID が **-starter** 接尾辞とともに追加されていることに注意してください。つまり、Camel Hystrix コンポーネントを **camel-hystrix** ではなく **camel-hystrix-starter** として指定します。Camel スターターコンポーネントは、Spring Boot 環境に対して最適化されるようにパッケージ化されています。

3.2.2. Spring Boot Maven プラグイン

Spring Boot Maven プラグインは Spring Boot によって提供されます。これは、Spring Boot プロジェクトをビルドおよび実行するための開発者ユーティリティーです。

- **ビルド**: プロジェクトディレクトリーでコマンド **mvn package** を入力し、Spring Boot アプリケーションの実行可能な Jar パッケージを作成します。ビルドの出力は、Maven プロジェクトの **target/** サブディレクトリーに格納されます。
- **実行**: 新規ビルドされたアプリケーションは **mvn spring-boot:start** コマンドで実行することができます。

Spring Boot Maven プラグインをプロジェクトの POM ファイルに組み込むには、以下の例のように、プラグイン設定を **pom.xml** ファイルの **project/build/plugins** セクションに追加します。

```
<?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>7.5.0.fuse-750029-redhat-00002</fuse.version>

  </properties>
  ...
  <build>
    <plugins>
```

```
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${fuse.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>repackage</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

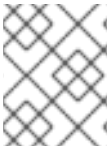
第4章 SPRING BOOT の APACHE CAMEL

4.1. CAMEL SPRING BOOT について

Camel Spring Boot コンポーネントは、Apache Camel の自動設定を提供します。Camel コンテキストの自動設定は、Spring コンテキストで利用可能な Camel ルートを自動検出し、プロデューサーテンプレート、コンシューマーテンプレート、タイプコンバーターなどの主要な Camel ユーティリティーを Bean として登録します。

Camel Spring Boot アプリケーションはすべて、製品化バージョンで **dependencyManagement** を使用する必要があります。クイックスタート pom を参照してください。BOM のバージョンをオーバーライドしないように、後でタグが付けられたバージョンを省略することができます。

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



注記

camel-spring-boot jar には **spring.factories** ファイルが同梱されており、クラスパスにその依存関係を追加できるため、Spring Boot は自動的に Camel を自動設定します。

4.2. CAMEL SPRING BOOT スターターについて

Apache Camel には、スターターを使用して Spring Boot アプリケーションを開発できる、Spring Boot スターターモジュールが含まれています。

注記

詳細は、Apache Camel Spring-Boot examples を参照してください。

スターターを使用するには、以下のスニペットを Spring Boot の **pom.xml** ファイルに追加します。

```
<dependency>
   <groupId>org.apache.camel</groupId>
   <artifactId>camel-spring-boot-starter</artifactId>
</dependency>
```

スターターを使用すると、以下のスニペットのように、Camel ルートでクラスを追加できます。これらのルートがクラスパスに追加されると、ルートは自動的に開始されます。

```
package com.example;

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

```
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("timer:foo").to("log:bar");
    }
}
```

application.properties または **application.yml** ファイルで Camel アプリケーションをカスタマイズできます。

Camel Spring Boot は、Camel スターターコンポーネントのいずれかを設定するときに、設定ファイル (application.properties または yaml ファイル) の ID 名による Bean の参照をサポートするようになりました。**src/main/resources/application.properties** （または yaml）ファイルで、Bean ID 名を参照して他の Bean を参照する Camel のオプションを簡単に設定できるようになりました。たとえば、以下のように Bean ID を使用すると、xslt コンポーネントはカスタム Bean を参照できます。

以下のように、IDmyExtensionFactory でカスタム Bean を参照します。

```
camel.component.xslt.saxon-extension-functions=myExtensionFactory
```

以下のように、Spring Boot @Bean アノテーションを使用して作成します。

```
@Bean(name = "myExtensionFactory")
public ExtensionFunctionDefinition myExtensionFactory() {
    }
```

または、**camel-jackson** データ形式の Jackson ObjectMapper の場合：

```
camel.dataformat.json-jackson.object-mapper=myJacksonMapper
```

4.3. 自動設定された CAMEL コンテキスト

Camel auto configuration は **CamelContext** インスタンスを提供し、**SpringCamelContext** を作成します。また、コンテキストの初期化およびシャットダウンを実行します。この Camel コンテキストは、**camelContext** Bean 名で Spring アプリケーションコンテキストに登録され、他の Spring Bean と同様にアクセスできます。

たとえば、以下に示すように **camelContext** にアクセスできます。

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

  @Bean
  MyService myService() {
    return new DefaultMyService(camelContext);
```

```
  }

}
```

4.4. CAMEL ルートの自動検出

Camel auto-configuration は、Spring コンテキストからすべての **RouteBuilder** インスタンスを収集し、自動的に **CamelContext** に注入します。Spring Boot スターターで新しい Camel ルートを作成する処理が簡単になります。**@Component** アノテーションが付けられたクラスをクラスパスに追加することで、ルートを作成できます。

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("jms:invoices").to("file:/invoices");
  }

}
```

@Configuration クラスに新しいルート **RouteBuilder** Bean を作成するには、以下を参照してください。

```
@Configuration
public class MyRouterConfiguration {

  @Bean
  RoutesBuilder myRouter() {
    return new RouteBuilder() {

      @Override
      public void configure() throws Exception {
        from("jms:invoices").to("file:/invoices");
      }

    };
  }

}
```

4.5. CAMEL プロパティー

Spring Boot の自動設定は、プロパティープレースホルダー、OS 環境変数、Camel プロパティーがサポートされるシステムプロパティーなどの Spring Boot 外部設定に自動的に接続します。

これらのプロパティーは **application.properties** ファイルで定義されます。

```
route.from = jms:invoices
```

システムプロパティーとしての使用

```
java -Droute.to=jms:processed.invoices -jar mySpringApp.jar
```

Camel ルートのプレースホルダーとしての使用

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("{{route.from}}").to("{{route.to}}");
  }

}
```

4.6. カスタムの CAMEL コンテキスト設定

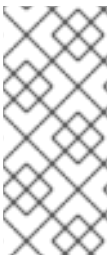
Camel auto-configuration によって作成された **CamelContext** Bean で操作を実行するには、次のように **CamelContextConfiguration** インスタンスを Spring コンテキストに登録する必要があります。

```
@Configuration
public class MyAppConfig {

  ...

  @Bean
  CamelContextConfiguration contextConfiguration() {
    return new CamelContextConfiguration() {
      @Override
      void beforeApplicationStart(CamelContext context) {
        // your custom configuration goes here
      }
    };
  }

}
```



注記

Spring コンテキストの開始前に **CamelContextConfiguration** メソッドおよび **beforeApplicationStart (CamelContext)** メソッドが呼び出されるため、このコールバックに渡される **CamelContext** インスタンスは完全に自動設定されます。複数のインスタンスの **CamelContextConfiguration** を Spring コンテキストに追加でき、すべてが実行されます。

4.7. JMX の無効化

自動設定された **CamelContext** の JMX を無効にするには、JMX がデフォルトで有効になっているため、**camel.springboot.jmxEnabled** プロパティーを使用します。

たとえば、以下のプロパティーを **application.properties** ファイルに追加します。

```
camel.springboot.jmxEnabled = false
```

4.8. 自動設定されたコンシューマーおよびプロデューサーのテンプレート

Camel auto configuration によって、事前設定された **ConsumerTemplate** および **ProducerTemplate** インスタンスが提供されます。これらを Spring 管理の Bean にインジェクトすることができます。

```
@Component
public class InvoiceProcessor {

  @Autowired
  private ProducerTemplate producerTemplate;

  @Autowired
  private ConsumerTemplate consumerTemplate;
  public void processNextInvoice() {
    Invoice invoice = consumerTemplate.receiveBody("jms:invoices", Invoice.class);
    ...
    producerTemplate.sendBody("netty-http:http://invoicing.com/received/" + invoice.id());
  }

}
```

デフォルトでは、コンシューマーテンプレートとプロデューサーテンプレートのエンドポイントキャッシュサイズは 1000 に設定されています。以下の Spring プロパティーを使用すると、これらの値を変更できます。

```
camel.springboot.consumerTemplateCacheSize = 100
camel.springboot.producerTemplateCacheSize = 200
```

4.9. 自動設定された TYPECONVERTER

Camel auto configuration では、Spring コンテキストの **typeConverter** という名前の **TypeConverter** インスタンスが登録されます。

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public long parseInvoiceValue(Invoice invoice) {
    String invoiceValue = invoice.grossValue();
    return typeConverter.convertTo(Long.class, invoiceValue);
  }

}
```

4.10. SPRING タイプコンバージョン API ブリッジ

Spring は タイプ変換 API で設定されます。Spring API は Camel の 型コンバーター API と似ています。これらの API は似ているため、Camel Spring Boot は Spring コンバージョン API に委譲するブリッジコンバーター (**SpringTypeConverter**) を自動的に登録します。つまり、追加設定のない Camel は Spring コンバーターを Camel と同様に扱います。

これにより、以下のように Camel **TypeConverter** API を使用して、Camel および Spring コンバーターの両方にアクセスできます。

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public UUID parseInvoiceId(Invoice invoice) {
    // Using Spring's StringToUUIDConverter
    UUID id = invoice.typeConverter.convertTo(UUID.class, invoice.getId());
  }

}
```

ここでは、Spring Boot はアプリケーションコンテキストで使用できる Spring の **ConversionService** インスタンスに変換を委譲します。**ConversionService** インスタンスが利用できない場合、Camel Spring Boot auto-configuration は **ConversionService** のインスタンスを作成します。

4.11. タイプ変換機能の無効化

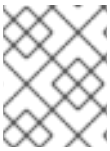
TypeConverter インスタンスや Spring ブリッジなどの Camel Spring Boot のタイプ変換機能の登録を無効にするには、以下のように **camel.springboot.typeConversion** プロパティーを **false** に設定します。

```
camel.springboot.typeConversion = false
```

4.12. XML ルートの追加

デフォルトでは、Camel XML ルートを **camel-spring-boot** が自動検出して含めるディレクトリー camel の下のクラスパスに配置できます。**Camel バージョン 2.17** 以降では、以下に示すように、設定オプションを使用してディレクトリー名を設定したり、この機能を無効にすることができます。

```
// turn off
camel.springboot.xmlRoutes = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRoutes = classpath:com/foo/routes/*.xml
```



注記

XML ファイルは Camel XML ルートであり、次のような **CamelContext** ではありません。

```
<routes xmlns="http://camel.apache.org/schema/spring">
    <route id="test">
        <from uri="timer://trigger"/>
        <transform>
            <simple>ref:myBean</simple>
        </transform>
        <to uri="log:out"/>
    </route>
</routes>
```

<camelContext> と Spring XML ファイルを使用する場合、Camel を Spring XML ファイルおよび application.properties ファイルで設定できます。たとえば、Camel に名前を設定し、ストリームキャッシュを有効にするには、以下を追加します。

```
camel.springboot.name = MyCamel
camel.springboot.stream-caching-enabled=true
```

4.13. XML REST-DSL の追加

デフォルトでは、Camel Rest-DSL XML ルートを **camel-rest** ディレクトリー下のクラスパスに配置できます。これは、**camel-spring-boot** が自動検出して組み込みます。以下のように設定オプションを使用して、ディレクトリー名を設定したり、この機能を無効にすることができます。

```
// turn off
camel.springboot.xmlRests = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRests = classpath:com/foo/rests/*.xml
```

注記

Rest-DSL XML ファイルは、以下のような **CamelContext** ではなく Camel XML rest である必要があります。

```
<rests xmlns="http://camel.apache.org/schema/spring">
  <rest>
    <post uri="/persons">
      <to uri="direct:postPersons"/>
    </post>
    <get uri="/persons">
      <to uri="direct:getPersons"/>
    </get>
    <get uri="/persons/{personId}">
       <to uri="direct:getPersionId"/>
    </get>
    <put uri="/persons/{personId}">
       <to uri="direct:putPersionId"/>
    </put>
    <delete uri="/persons/{personId}">
       <to uri="direct:deletePersionId"/>
    </delete>
  </rest>
</rests>
```

4.14. CAMEL SPRING BOOT でのテスト

Spring Boot で実行されている Camel の場合、Spring Boot は自動的に Camel と **@Component** アノテーションが付けられたそのルートを組み込みます。Spring Boot でテストする場合、**@ContextConfiguration** の代わりに **@SpringBootTest** を使用して、使用する設定クラスを指定します。

異なる RouteBuilder クラスに複数の Camel ルートがある場合、Camel Spring Boot にはこれらのルートがすべて含まれます。1 つの RouteBuilder クラスのみからルートをテストする場合は、以下のパター

ンを使用して、有効にする RouteBuilder を include (含める) または exclude (除外) することができます。

- java-routes-include-pattern: パターンに一致する RouteBuilder クラスを include (含める) ために使用されます。
- java-routes-exclude-pattern: パターンに一致する RouteBuilder クラスを exclude (除外) するために使用されます。exclude は include よりも優先されます。

これらのパターンは、以下に示すように、ユニットテストクラスで **@SpringBootTest** アノテーションのプロパティーとして指定できます。

```
@RunWith(CamelSpringBootRunner.class)
@SpringBootTest(classes = {MyApplication.class);
  properties = {"camel.springboot.java-routes-include-pattern=**/Foo*"})
public class FooTest {
```

FooTest クラスの include パターンは Ant スタイルパターンを表す **/Foo* です。このパターンは、すべてのパッケージ名と一致する 2 つのアスタリスクで始まります。/**Foo*** は、FooRoute のようにクラス名が Foo で始まる必要があることを意味します。以下の Maven コマンドを使用してテストを実行できます。

```
mvn test -Dtest=FooTest
```

4.15. 関連項目

- コンポーネント
- Endpoint (エンドポイント)
- Getting Started (スタートガイド)

4.16. SPRING BOOT、APACHE CAMEL、および外部メッセージングブローカーの使用

4.16.1. 外部メッセージングブローカーの使用

Fuse は外部メッセージングブローカーを使用します。サポートされるブローカー、クライアント、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 でサポートされる設定 を参照してください。

Camel コンポーネントは JMS 接続ファクトリーに接続されている必要があります。以下の例は、**camel-amqp** コンポーネントを JMS 接続ファクトリーに接続する方法を示しています。

```
import org.apache.activemq.jms.pool.PooledConnectionFactory;
import org.apache.camel.component.amqp.AMQPComponent;
import org.apache.qpid.jms.JmsConnectionFactory;
...

AMQPComponent amqpComponent(AMQPConfiguration config) {
      JmsConnectionFactory qpid = new JmsConnectionFactory(config.getUsername(),
config.getPassword(), "amqp://"+ config.getHost() + ":" + config.getPort());
      qpid.setTopicPrefix("topic://");
```

```
        PooledConnectionFactory factory = new PooledConnectionFactory();
        factory.setConnectionFactory(qpid);

AMQPComponent amqpcomp = new AMQPComponent(factory);
```

付録A MAVEN を使用する準備

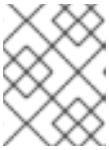
A.1. 概要

ここでは、Red Hat Fuse プロジェクトをビルドするために Maven を準備する方法の概要を説明し、Maven アーティファクトの検索に使用される Maven コーディネートの概念を紹介します。

A.2. 前提条件

Maven を使用してプロジェクトをビルドするには、以下が必要になります。

- **Maven インストール**: Maven は Apache の無料のオープンソースビルドツールです。最新バージョンは Maven のダウンロードページ からダウンロードできます。
- **ネットワーク接続**: ビルドの実行中、Maven は追加設定を必要とせずに動的に外部リポジトリーを検索し、必要なアーティファクトをダウンロードします。デフォルトでは、Maven はインターネット経由でアクセスされるリポジトリーを検索します。Maven がローカルネットワーク上のリポジトリーを優先して検索するように、この挙動を変更することができます。



注記

Maven はオフラインモードで実行できます。オフラインモードでは、Maven はローカルリポジトリーのアーティファクトのみを検索します。

A.3. RED HAT MAVEN リポジトリーの追加

Red Hat Maven リポジトリーからアーティファクトにアクセスするには、そのアーティファクトを Maven の **settings.xml** ファイルに追加する必要があります。Maven は、ユーザーのホームディレクトリーの **.m2** ディレクトリーで **settings.xml** ファイルを検索します。ユーザー指定の **settings.xml** ファイルがない場合、Maven は **M2_HOME/conf/ settings.xml にあるシステムレベルの settings.xml** ファイルを使用します。

Red Hat リポジトリーを Maven のリポジトリーの一覧に追加するには、新しい . **m2/settings.xml** ファイルを作成するか、システムレベルの設定を変更します。**settings.xml** ファイルに、Red Hat Fuse リポジトリーの Maven への追加 に示すように、Red Hat リポジトリーの **repository** 要素を追加します。

Red Hat Fuse リポジトリーの Maven への追加

```
<?xml version="1.0"?>
<settings>

  <profiles>
   <profile>
     <id>extra-repos</id>
     <activation>
       <activeByDefault>true</activeByDefault>
     </activation>
     <repositories>
      <repository>
          <id>redhat-ga-repository</id>
          <url>https://maven.repository.redhat.com/ga</url>
          <releases>
```

```
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
        <repository>
          <id>redhat-ea-repository</id>
          <url>https://maven.repository.redhat.com/earlyaccess/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-public</id>
          <name>JBoss Public Repository Group</name>
          <url>https://repository.jboss.org/nexus/content/groups/public/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>redhat-ga-repository</id>
          <url>https://maven.repository.redhat.com/ga</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>redhat-ea-repository</id>
          <url>https://maven.repository.redhat.com/earlyaccess/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-public</id>
          <name>JBoss Public Repository Group</name>
          <url>https://repository.jboss.org/nexus/content/groups/public</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <activeProfile>extra-repos</activeProfile>
```

```
  </activeProfiles>

</settings>
```

A.4. アーティファクト

アーティファクトは Maven ビルドシステムの基本的な要素です。Maven ビルドの実行後、アーティファクトの出力は通常、JAR や WAR などのアーカイブになります。

A.5. MAVEN コーディネート

Maven の主な機能には、アーティファクトを見つけ、アーティファクトとの間の依存関係を管理する機能があります。Maven は、特定のアーティファクトの場所を一意に定義する **Maven コーディネート** を使用して、アーティファクトの場所を定義します。基本的なコーディネートタプルの形式は **{groupId,artifactId,version}** です。追加のコーディネートである **packaging** および **classifier** をタプルに使用することもあります。タプルは、基本のコーディネート、追加の **packaging** コーディネート、または追加の **packaging** および **classifier** コーディネートの両方使用して、以下のように作成できます。

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

各コーディネートの説明は次のとおりです。

groupId

アーティファクトの名前の範囲を定義します。通常、パッケージ名のすべてまたは一部をグループ IDアプリケーションに使用します（例： **org.fusesource.example** ）。

artifactId

アーティファクト名 (グループ ID に関連する) を定義します。

version

アーティファクトのバージョンを指定します。バージョン番号には **n .n.n.n の 4 つの部分を含めることができ、バージョン番号の最後の部分には数字以外の文字を含めることができます（たとえば、1.0-SNAPSHOT** の最後の部分は英数字のサブ文字列、**0-SNAPSHOT**）。

packaging

プロジェクトのビルド時に生成されるパッケージ化されたエンティティーを定義します。OSGi プロジェクトでは、パッケージングは **bundle** になります。デフォルト値は **jar** です。

classifier

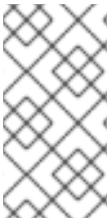
同じ POM からビルドされた内容が異なるアーティファクトを区別できるようにします。

グループ ID、アーティファクト ID、パッケージング、およびバージョンは、アーティファクトの POM ファイルの対応する要素によって定義されます。以下に例を示します。

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

たとえば、前述のアーティファクトの依存関係を定義するには、以下の **dependency** 要素を POM に追加します。

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```



注記

前述の依存関係に **bundle** パッケージを指定する必要はあり **ません**。バンドルは特定タイプの JAR ファイルで、**jar** はデフォルトの Maven パッケージタイプであるためです。依存関係でパッケージタイプを明示的に指定する必要がある場合は、**type** 要素を使用できます。

付録B SPRING BOOT MAVEN プラグイン

B.1. SPRING BOOT MAVEN プラグインの概要

この付録では、Spring Boot Maven プラグインについて説明します。Maven で Spring Boot サポートを提供し、実行可能な jar または war アーカイブをパッケージ化し、アプリケーションを **インプレースで** 実行できます。

B.2. ゴール

Spring Boot プラグインのゴールには以下が含まれます。

1. **spring-boot:run** は Spring Boot アプリケーションを実行します。
2. **spring-boot:repackage** は、**.jar** および **.war** ファイルを再パッケージして実行可能にします。
3. **spring-boot:start** および **spring-boot:stop** の両方は、Spring Boot アプリケーションのライフサイクルを管理するために使用されます。
4. **spring-boot:build-info** は、Actuator が使用できるビルド情報を生成します。

B.3. USAGE

Spring Boot プラグインの使用方法に関する一般的な手順は、https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/#using を参照してください。以下の例は、Spring Boot の **spring-boot-maven-plugin** の使用方法を示しています。

- Spring Boot 2 の例
- Spring Boot 1 の例

注記

Spring Boot Maven プラグインの詳細は、https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/ を参照してください。

B.3.1. Spring Boot 2 の Spring Boot Maven プラグイン

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>

    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

      <!-- configure the Fuse version you want to use here -->
      <fuse.bom.version>7.5.0.fuse-sb2-750029-redhat-00003</fuse.bom.version>

      <!-- maven plugin versions -->
```

```
    <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
    <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
</properties>


<build>
    <defaultGoal>spring-boot:run</defaultGoal>

    <plugins>
        <plugin>
            <groupId>org.jboss.redhat-fuse</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>${fuse.bom.version}</version>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

<repositories>
    <repository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
    <repository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>

<pluginRepositories>
    <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
```

```
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
</project>
```

B.3.2. Spring Boot 1 の Spring Boot Maven プラグイン

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>

    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

      <!-- configure the Fuse version you want to use here -->
      <fuse.bom.version>7.5.0.fuse-750029-redhat-00002</fuse.bom.version>

      <!-- maven plugin versions -->
      <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
      <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
    </properties>


    <build>
      <defaultGoal>spring-boot:run</defaultGoal>

      <plugins>
        <plugin>
          <groupId>org.jboss.redhat-fuse</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
          <version>${fuse.bom.version}</version>
          <executions>
            <execution>
              <goals>
                <goal>repackage</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
```

```
  <repositories>
    <repository>
      <id>redhat-ga-repository</id>
      <url>https://maven.repository.redhat.com/ga</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>redhat-ga-repository</id>
      <url>https://maven.repository.redhat.com/ga</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</project>
```