Red Hat Fuse 7.12

Fuse on Spring Boot のスタートガイド

Spring Boot で Red Hat Fuse をすぐに使い始める

Last Updated: 2023-11-25

Red Hat Fuse 7.12 Fuse on Spring Boot のスタートガイド

Spring Boot で Red Hat Fuse をすぐに使い始める

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

Fuse on Spring Boot の使用を開始します。

目次

はじめに

Fuse を使い始めるには、Spring Boot コンテナーのファイルをダウンロードしてインストールする必要があります。ここでは、初めて Fuse アプリケーションをインストール、開発、および構築するための情報および手順を提供します。

- 1章*Fuse on Spring Boot の使用*
- 2章*Maven のローカルでの設定*

多様性を受け入れるオープンソースの強化

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

第1章 FUSE ON SPRING BOOT の使用

Spring Boot で Fuse アプリケーションを開発するには、最初に Spring Boot で実行する Fuse のサンプルブースタープロジェクトを生成およびビルドします。詳細は以下のセクションを参照してください。

- 「Fuse on Spring Boot」
- 「ブースタープロジェクトの生成」
- 「ブースタープロジェクトのビルド」

1.1. FUSE ON SPRING BOOT

Spring Boot は、よく知られている Spring コンテナーがさらに進化したものです。Spring Boot コンテナーの特徴は、個別にデプロイできる小さなチャンクにコンテナー機能が分割されていることです。このような特性を持つため、特定のサービスに特化して、小さなフットプリントでコンテナーをデプロイできます。この機能は、**マイクロサービスアーキテクチャー** のパラダイムに対応するために必要なものです。

このコンテナー技術の特徴は次のとおりです。

- 特に、スケーラブルなクラウドプラットフォーム (Kubernetes および OpenShift) での実行に適しています。
- マイクロサービスアーキテクチャーに適した小さなフットプリント。
- **設定より規約** (Convention over Configuration) 向けに最適化されています。
- アプリケーションサーバーは必要ありません。Spring Boot アプリケーション Jar を直接 JVM で実行できます。

1.2. 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 でサポートされる構成 を参照してください。
- Maven のローカルでの設定 の手順に従って、Apache Maven 3.3.x 以降をインストールおよび設定している。

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 **7.12 (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。

1.3. ブースタープロジェクトのビルド

以下の手順では、Fuse on Spring Boot で Circuit Breaker ブースターをビルドする方法を説明します。

前提条件

- Red Hat Developer Portal からブースタープロジェクトの生成およびダウンロードが完了している。
- 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 でサポートされる構成 を参照してください。

- Maven のローカルでの設定 の手順に従って、Apache Maven 3.3.x 以降をインストールおよび設定している。

手順

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 fuse-circuit-breaker
```

```
mvn clean package
```

Maven によってプロジェクトがビルドされると、**Build Success** というメッセージが表示されます。

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 -Dspring-boot.run.arguments="--server.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。

```
...
2019-05-06 20:19:59.401  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Route: route1 started and consuming from: servlet:/name?httpMethodRestrict=GET
2019-05-06 20:19:59.402  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Total 1 routes, of which 1 are started
2019-05-06 20:19:59.403  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Apache Camel 2.21.0.fuse-730078-redhat-00001 (CamelContext: camel-1) started in 0.287
seconds
2019-05-06 20:19:59.406  INFO 9553 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:19:59.473  INFO 9553 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8081 (http)
2019-05-06 20:19:59.479  INFO 9553 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 5.485 seconds (JVM
running for 9.841)
```

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。

```
...
2019-05-06 20:22:19.051  INFO 9729 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:22:19.115  INFO 9729 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8080 (http)
```

```
2019-05-06 20:22:19.123  INFO 9729 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 7.68 seconds (JVM running
for 12.66)
```

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

4. web ブラウザーで URL を開くか、別のシェルプロンプトで以下の **curl** コマンドを入力して、REST エンドポイントを呼び出します。

```
curl http://localhost:8080/camel/greetings
```

応答は次のとおりです。

```
{"greetings":"Hello, Jacopo"}
```

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ために、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** を入力し、バックエンド name サービスを強制終了します。
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. web ブラウザーで **http://localhost:8080/camel/greetings** を開くか、別のシェルプロンプトウインドウに以下の **curl** コマンドを入力して、greetings REST エンドポイントを呼び出します。

```
curl http://localhost:8080/camel/greetings
```

応答は次のとおりです。

```
{"greetings":"Hello, default fallback"}
```

greetings サービスが実行されているウインドウで、ログに以下のメッセージシーケンスが表示されます。

```
2019-05-06 20:24:16.952  INFO 9729 --- [-CamelHystrix-2] route2                                  : Try
to call name Service
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.964  INFO 9729 --- [-CamelHystrix-2] route2                                  : We
are falling back!!!!
```

7. この例に関する詳細は、**greetings-service** の実行中に http://localhost:8080/ で **Circuit Breaker - Red Hat Fuse** ページを開いてください。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が含まれます。

第2章 MAVEN のローカルでの設定

一般的な Fuse アプリケーションの開発では、Maven を使用してプロジェクトをビルドおよび管理します。

以下のトピックでは、Maven をローカルで設定する方法を説明します。

- 「Maven 設定の準備」
- 「Maven への Red Hat リポジトリーの追加」
- 「ローカル Maven リポジトリーの使用」
- 「環境変数またはシステムプロパティーを使用した Maven ミラーの設定」
- 「Maven アーティファクトおよびコーディネート」

2.1. MAVEN 設定の準備

Maven は、Apache の無料のオープンソースビルドツールです。通常は、Maven を使用して Fuse アプリケーションを構築します。

手順

1. Maven ダウンロードページ から最新バージョンの Maven をダウンロードします。
2. システムがインターネットに接続していることを確認します。
   デフォルトの動作では、プロジェクトのビルド中、Maven は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。Maven はインターネット上でアクセス可能なリポジトリーを探します。

   このデフォルト動作を変更し、Maven によってローカルネットワーク上のリポジトリーのみが検索されるようにすることができます。これは Maven をオフラインモードで実行できることを意味します。オフラインモードでは、Maven によってローカルリポジトリーのアーティファクトが検索されます。「ローカル Maven リポジトリーの使用」 を参照してください。

2.2. MAVEN への RED HAT リポジトリーの追加

Red Hat Maven リポジトリーあるアーティファクトにアクセスするには、Red Hat Maven リポジトリーを 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** ファイルを探します。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。

前提条件

Red Hat リポジトリーを追加する **settings.xml** ファイルがある場所を知っている。

手順

以下の例のように、**settings.xml** ファイルに Red Hat リポジトリーの **repository** 要素を追加します。

```
<?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>
```

2.3. ローカル MAVEN リポジトリーの使用

インターネットへ接続せずにコンテナーを実行し、オフライン状態では使用できない依存関係を持つアプリケーションをデプロイする場合は、Maven 依存関係プラグインを使用してアプリケーションの依存関係を Maven オフラインリポジトリーにダウンロードできます。ダウンロードしたら、このカスタマイズされた Maven オフラインリポジトリーをインターネットに接続していないマシンに提供することができます。

手順

1. **pom.xml** ファイルが含まれるプロジェクトディレクトリーで、以下のようなコマンドを実行し、Maven プロジェクトのリポジトリーをダウンロードします。

   ```
   mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
   Dmaven.repo.local=/tmp/my-project
   ```

   この例では、プロジェクトのビルドに必要な Maven 依存関係とプラグインは **/tmp/my-project** ディレクトリーにダウンロードされます。

2. このカスタマイズされた Maven オフラインリポジトリーを、インターネットに接続していない内部のマシンに提供します。

2.4. 環境変数またはシステムプロパティーを使用した MAVEN ミラーの設定

アプリケーションの実行時には、Red Hat Maven リポジトリーにあるアーティファクトにアクセスする必要があります。このリポジトリーは、Maven の **settings.xml** ファイルに追加されます。Maven は以下の場所で **settings.xml** を探します。

- 指定の URL を検索します。
- 見つからない場合は **${user.home}/.m2/settings.xml** を検索します。
- 見つからない場合は **${maven.home}/conf/settings.xml** を検索します。
- 見つからない場合は **${M2_HOME}/conf/settings.xml** を検索します。
- どの場所も見つからない場合は、空の **org.apache.maven.settings.Settings** インスタンスが作成されます。

2.4.1. Maven ミラー

Maven では、一連のリモートリポジトリーを使用して、現在ローカルリポジトリーで利用できないアーティファクトにアクセスします。ほとんどの場合で、リポジトリーのリストには Maven Central リポジトリーが含まれますが、Red Hat Fuse では Maven Red Hat リポジトリーも含まれます。リモートリポジトリーへのアクセスが不可能な場合や許可されない場合は、Maven ミラーのメカニズムを使用できます。ミラーは、特定のリポジトリー URL を異なるリポジトリー URL に置き換えるため、リモートアーティファクトの検索時にすべての HTTP トラフィックを単一の URL に転送できます。

2.4.2. Maven ミラーの **settings.xml** への追加

Maven ミラーを設定するには、以下のセクションを Maven の **settings.xml** に追加します。

```
<mirror>
    <id>all</id>
    <mirrorOf>*</mirrorOf>
    <url>http://host:port/path</url>
</mirror>
```

settings.xml ファイルに上記のセクションがない場合は、ミラーが使用されません。XML 設定を提供せずにグローバルミラーを指定するには、システムプロパティーまたは環境変数を使用します。

2.4.3. 環境変数またはシステムプロパティーを使用した Maven ミラーの設定

環境変数またはシステムプロパティーのいずれかを使用して Maven ミラーを設定するには、以下を追加します。

- 環境変数 **MAVEN_MIRROR_URL** を **bin/setenv** ファイルに追加します。
- システムプロパティー **mavenMirrorUrl** を **etc/system.properties** ファイルに追加します。

2.4.4. Maven オプションを使用した Maven ミラー URL の指定

環境変数またはシステムプロパティーによって指定された Maven ミラー URL ではなく、別の Maven ミラー URL を使用するには、アプリケーションの実行時に以下の Maven オプションを使用します。

- **-DmavenMirrorUrl=mirrorId::mirrorUrl**
  たとえば、**-DmavenMirrorUrl=my-mirror::http://mirror.net/repository** となります。
- **-DmavenMirrorUrl=mirrorUrl**
  たとえば、**-DmavenMirrorUrl=http://mirror.net/repository** となります。この例では、<mirror> の <id> はミラーになります。

2.5. MAVEN アーティファクトおよびコーディネート

Maven ビルドシステムでは、**アーティファクト** が基本的なビルディングブロックです。ビルド後のアーティファクトの出力は、通常 JAR や WAR ファイルなどのアーカイブになります。

Maven の主な特徴として、アーティファクトを検索し、検索したアーティファクト間で依存関係を管理できる機能が挙げられます。**Maven コーディネート** は、特定のアーティファクトの場所を特定する値のセットです。基本的なコーディネートには、以下の形式の 3 つの値があります。

groupId:artifactId:version

場合によっては、**packaging** の値、または **packaging** と **classifier** の値の両方を使用して、基本的なコーディネートを拡張することがあります。Maven コーディネートには以下の形式のいずれかを使用できます。

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

値の説明は次のとおりです。

groupdId

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 として使用します。たとえば、**org.fusesource.example** です。

artifactId

グループ名に関連するアーティファクト名を定義します。

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** のように最大 4 つの部分を含めることができ、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** の場合は、最後の部分が英数字のサブ文字列である **0-SNAPSHOT** になります。

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。

次に示すように、アーティファクトの POM ファイル内の要素で、アーティファクトのグループ ID、アーティファクト ID、パッケージング、およびバージョンを定義します。

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

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 ファイルに追加します。

```
<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 ファイルであり、**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。