



## Red Hat Fuse 7.9

# Fuse on Spring Boot のスタートガイド

Red Hat Fuse on Spring Boot を今すぐ使用する



## Red Hat Fuse 7.9 Fuse on Spring Boot のスタートガイド

---

Red Hat Fuse on Spring Boot を今すぐ使用する

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

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Getting\_Started\_with\_Fuse\_on\_Spring\_Boot.ent file |.

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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

前書き .....	3
多様性を受け入れるオープンソースの強化 .....	4
第1章 FUSE ON SPRING BOOT の使用 .....	5
1.1. FUSE ON SPRING BOOT .....	5
1.2. ブースタープロジェクトの生成 .....	5
1.3. ブースタープロジェクトのビルド .....	6
第2章 MAVEN のローカルでの設定 .....	10
2.1. MAVEN 設定の準備 .....	10
2.2. RED HAT リポジトリを MAVEN へ追加 .....	10
2.3. ローカル MAVEN リポジトリの追加 .....	12
2.4. 環境変数またはシステムプロパティを使用した MAVEN ミラーの設定 .....	12
2.4.1. Maven ミラー .....	13
2.4.2. Maven ミラーの settings.xml への追加 .....	13
2.4.3. 環境変数またはシステムプロパティを使用した Maven ミラーの設定 .....	13
2.4.4. Maven オプションを使用した Maven ミラー URL の指定 .....	13
2.5. MAVEN アーティファクトおよびコーディネート .....	13



## 前書き

Fuse を使用するには、Spring Boot コンテナのファイルをダウンロードおよびインストールする必要があります。本書では、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 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 コンテナがさらに進化したものです。Spring Boot コンテナの特徴は、コンテナ機能が独立してデプロイできる小さなチャンクに分割されていることです。この品質により、特定のサービスに特化し、小さなフットプリントでコンテナをデプロイできます。これは、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のに必要な機能です。

このコンテナ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。
- 名前を取得するよう **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** サービスは再起動するまでの間にブロックされるのではなく、**greetings** サービスが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。

### 前提条件

- [Red Hat Developer Platform](#) にアクセスできる必要があります。
- サポートされるバージョンの Java Developer Kit (JDK) が必要です。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。
- 「[Setting up Maven locally](#)」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必要があります。

## 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 **7.9 (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

これらの手順は、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必要があります。
- サポートされるバージョンの Java Developer Kit (JDK) が必要です。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。

- 「[Setting up Maven locally](#)」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する必要があります。

## 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た後、**Build Success** 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 -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。

```
...
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。

- 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"}
```

- 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ィンドウで **Ctrl-C** を入力し、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。
- 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ィンドウに以下の **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ィンドウで、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 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」](#)
- [「Red Hat リポジトリを Maven へ追加」](#)
- [「ローカル Maven リポジトリの追加」](#)
- [「環境変数またはシステムプロパティを使用した 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ビルドします。

#### 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。  
プロジェクトのビルド中、Maven が外部リポジトリを探し、必要なアーティファクトをダウンロードするのがデフォルトの動作になります。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 リポジトリの追加」](#) を参照してください。

### 2.2. RED HAT リポジトリを MAVEN へ追加

Red Hat Maven リポジトリあるアーティファクトにアクセスするには、Red Hat Maven リポジトリを Maven の **settings.xml** ファイルに追加する必要があります。Maven は、**.m2** ディレクトリで **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** のシステムレベルの **settings.xml** ファイルを使用します。

#### 前提条件

Red Hat リポジトリを追加する **settings.xml** ファイルがある場所を知っている必要があります。

#### 手順

以下の例のように、**settings.xml** ファイルに Red Hat リポジトリの **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 リポジトリの追加

インターネットへ接続せずにコンテナを実行し、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、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にダウンロードすることができます。ダウンロード後、このカスタマイズされた Maven オフラインリポジトリ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. **etc/org.ops4j.pax.url.mvn.cfg** ファイルを編集し、**org.ops4j.pax.url.mvn.offline** を true に設定します。これによりオフラインモードが有効になります。

```

##
# If set to true, no remote repository will be accessed when resolving artifacts
#
org.ops4j.pax.url.mvn.offline = true

```

3. このカスタマイズされた Maven オフラインリポジトリを、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を使用した MAVEN ミラーの設定

アプリケーションの稼働中に、Red Hat Maven リポジトリ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。これらのリポジトリは、Maven の **settings.xml** ファイルに追加されます。Maven は **settings.xml** で以下の場所をチェックします。

- 指定の URL を検索します。
- 見つからなかった場合は **\${user.home}/.m2/settings.xml** を検索します。
- 見つからなかった場合は **\${maven.home}/conf/settings.xml** を検索します。



- 見つからなかった場合は `${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を使用して、現在ローカルリポジトリ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、リポジトリのリストには Maven Central リポジトリが含まれますが、Red Hat Fuse では Maven Red Hat リポジトリも含まれます。リモートリポジトリへのアクセスが不可能な場合や許可されない場合に、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 URL を異なるリポジトリ 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 に転送することができます。

### 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、ミラーは使用されません。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 `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`  
例: `-DmavenMirrorUrl=my-mirror::http://mirror.net/repository`
- `-DmavenMirrorUrl=mirrorUrl`  
例: `-DmavenMirrorUrl=http://mirror.net/repository` 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のビルドブロックです。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

基本的なコーディネートに **packaging** の値、または **packaging** と **classifier** の値の両方を追加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。

#### 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 として使用します。例: **org.fusesource.example**

#### 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、最後の部分は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。

アーティファクトの POM ファイルの要素は、以下のように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

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、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。ただし、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。