Red Hat Fuse 7.12

Fuse on Apache Karaf のスタートガイド

Red Hat Fuse on Karaf をすぐに使い始める

Last Updated: 2023-12-26

Red Hat Fuse 7.12 Fuse on Apache Karaf のスタートガイド

Red Hat Fuse on Karaf をすぐに使い始める

法律上の通知

概要

Fuse on Apache Karaf を使用して、アプリケーションの構築を開始します。

目次

はじめに

Fuse を使い始めるには、Apache Karaf 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 Karaf 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、CTO である Chris Wright のメッセージ をご覧ください。

第1章 FUSE ON KARAF の使用

ここで説明する情報や手順は、Fuse On Karaf について学び、Karaf コンテナーで初めて Fuse アプリケーションをインストール、開発、および構築するのに役立ちます。詳細は以下のトピックを参照してください。

- 「Fuse on Karaf」
- 「Fuse on Karaf のインストール」
- 「Karaf で初めて Fuse アプリケーションを構築する」

1.1. FUSE ON KARAF

Apache Karaf は、OSGi Alliance の OSGi 標準 をベースとしています。OSGi は電気通信業界で開発され、サーバーをシャットダウンする必要なく、オンザフライでアップグレードできる (**ホットコードスワッピング** として知られる機能) ゲートウェイサーバーを開発するために使用されました。その後、OSGi コンテナー技術はその他多くの用途に使用されるようになり、一般的に Eclipse IDE などのモジュール化されたアプリケーションで使用されています。

このコンテナー技術の特徴は次のとおりです。

- 特にスタンドアロンモードでの実行に適しています。
- モジュール化 (OSGi バンドル) の強力なサポート、および高度なクラスローディングのサポート。
- 1 つのコンテナーに複数のバージョンの依存関係を一緒にデプロイできます (ただし、これは注意して行う必要があります)。
- ホットコードスワッピングにより、コンテナーをシャットダウンせずにモジュールをアップグレードまたは置き換えできます。これは独自の機能ですが、適切に行うには多大な労力が必要になります。

注記: Spring Dynamic Modules (Spring-DM) (Spring XML と Apache Karaf の OSGi サービス層を統合) はサポート対象ではありません。代わりに Blueprint フレームワークを使用する必要があります。Blueprint XML を使用しても、Spring フレームワークから Java ライブラリーを使用することはできます。最新バージョンの Spring は Blueprint と互換性があります。

1.2. FUSE ON KARAF のインストール

Red Hat カスタマーポータルから Fuse 7.12 on Karaf の標準インストールパッケージをダウンロードできます。このパッケージは Karaf コンテナーの標準アセンブリーをインストールし、完全な Fuse テクノロジースタックを提供します。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインしている。
- CodeReady Studio インストーラー がダウンロードされている。
- Fuse on Karaf インストーラー がダウンロードされている。

手順

手順

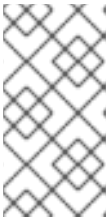
1. ダウンロードした Fuse on Apache Karaf の **.zip** アーカイブファイルを、ファイルシステム **FUSE_INSTALL** の任意の場所にデプロイメントします。
2. Fuse ランタイムに管理ユーザーを追加します。
   a. テキストエディターで **FUSE_INSTALL/etc/users.properties** ファイルを開きます。
   b. **#admin = admin** で始まる行の最初の **#** 文字を削除します。
   c. **#_g_\:admingroup** で始まる行の最初の **#** 文字を削除します。
   d. ユーザーエントリーのユーザー名 **USERNAME** とパスワード **PASSWORD** をカスタマイズして、次のようなユーザーエントリーと管理グループエントリーを (連続した行に) 作成します。

      ```
      USERNAME = PASSWORD,_g_:admingroup
      _g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
      ```

   e. **etc/users.properties** ファイルを保存します。
3. 以下のように、CodeReady Studio インストーラー を実行します。

   ```
   java -jar DOWNLOAD_LOCATION/codereadystudio-12.21.3.GA-installer-standalone.jar
   ```

4. インストール中、以下を行います。
   a. 契約条件に同意します。
   b. インストールパスを選択します。
   c. Java 8 JVM を選択します。
   d. **Select Platforms and Servers** で、**Add** をクリックして **FUSE_INSTALL** ディレクトリーの場所を確認し、Fuse on Karaf ランタイムを設定します。
   e. **Select Additional Features to Install** で **Red Hat Fuse Tooling** を選択します。
5. CodeReady Studio が起動します。**Searching for runtimes** ダイアログが表示されたら **OK** をクリックして Fuse on Karaf ランタイムを作成します。
6. **(任意手順)**: コマンドラインから Apache Maven を使用するには、Maven のローカルでの設定 の説明どおりに Maven をインストールおよび設定する必要があります。



注記

CodeReady Studio のみを使用する場合、CodeReady Studio には Maven が事前インストールおよび設定されているため、厳密には Maven をインストールする必要はありません。しかし、コマンドラインから Maven を呼び出す場合は、インストールを行う必要があります。

1.3. KARAF で初めて FUSE アプリケーションを構築する

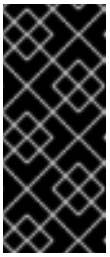
次の手順は、Karaf で初めて Fuse アプリケーションを構築する場合に便利です。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインしている。
- CodeReady Studio インストーラー がダウンロードされている。
- ダウンロードした Fuse on Karaf インストーラー が正常にインストールされている。

手順

1. CodeReady Studio で以下のように新しいプロジェクトを作成します。
   a. **File→New→Fuse Integration Project** と選択します。
   b. **Project Name** フィールドに **fuse-camel-cbr** を入力します。
   c. **Next** をクリックします。
   d. **Select a Target Environment** ペインで以下の設定を選択します。
      - **Standalone** をデプロイメントプラットフォームとして選択します。
      - **Karaf/Fuse on Karaf** をランタイム環境として選択し、**Runtime (optional)** ドロップダウンメニューを使用して **fuse-karaf-7.12.0.fuse-7_12_0-00019-redhat-00001 Runtime** サーバーをターゲットランタイムとして選択します。
   e. ターゲットランタイムの選択後、**Camel Version** が自動的に選択され、フィールドがグレーアウトされます。
   f. **Next** をクリックします。
   g. **Advanced Project Setup** ペインで **Beginner→Content Based Router - Blueprint DSL** テンプレートを選択します。
   h. **Finish** をクリックします。
   i. 関連する Fuse Integration パースペクティブを開くように要求された場合は、**Yes** をクリックします。
   j. CodeReady Studio が必要なアーティファクトをダウンロードし、バックグラウンドでプロジェクトをビルドする間待機します。



重要

CodeReady Studio で初めて Fuse プロジェクトをビルドする場合は、リモート Maven リポジトリーから依存関係をダウンロードするため、ウィザードがプロジェクトの生成を完了するまで **数分**かかることがあります。プロジェクトがバックグラウンドでビルドされている間は、ウィザードを中断したり、CodeReady Studio を閉じたりしないようにしてください。

2. 以下のように、プロジェクトをサーバーにデプロイします。
   a. サーバーが起動していない場合は、**Servers** ビュー (Fuse Integration パースペクティブの左下隅) で **fuse-karaf-7.12.0.fuse-7_12_0-00019-redhat-00001 Runtime Server** サーバーを選択し、緑色の矢印をクリックして起動します。

注記

Warning: The authenticity of host 'localhost' can't be established.というダイアログが表示されたら、**Yes** をクリックしてサーバーに接続し、Karaf コンソールにアクセスします。

b. **Console** ビューに以下のようなメッセージが表示されるまで待機します。

```
Karaf started in 1s. Bundle stats: 12 active, 12 total
```

c. サーバーが起動した後、**Servers** ビューに切り替え、サーバーを右クリックしてコンテキストメニューで **Add and Remove** を選択します。

d. **Add and Remove** ダイアログで **fuse-camel-cbr** プロジェクトを選択し、**Add >** ボタンをクリックします。

e. **Finish** をクリックします。

f. **Terminal** ビューに移動し、**bundle:list | tail** を入力して、プロジェクトの OSGi バンドルが起動したかどうかをチェックします。以下のような出力が表示されるはずです。

```
...
228 | Active |  80 | 1.0.0.201505202023          | org.osgi:org.osgi.service.j
232 | Active |  80 | 1.0.0.SNAPSHOT              | Fuse CBR Quickstart
```

注記

Camel ルートが起動すると、即座に **work/cbr/input** ディレクトリーが Fuse インストールに作成されます (**fuse-camel-cbr** プロジェクトには作成**されません**)。

3. プロジェクトの **src/main/data** ディレクトリーにあるファイルを **FUSE_INSTALL/work/cbr/input** ディレクトリーにコピーします。これは、システムファイルブラウザー (Eclipse の外部) で実行できます。

4. しばらく待ってから、**FUSE_INSTALL/work/cbr/output** ディレクトリーをチェックし、同じファイルが国ごとに整理されていることを確認します。

   a. **work/cbr/output/others** の **order1.xml**

   b. **work/cbr/output/uk** の **order2.xml** および **order4.xml**

   c. **work/cbr/output/us** の **order3.xml** および **order5.xml**

5. 以下のようにプロジェクトをアンデプロイします。

   a. **Servers** ビューで **Red Hat Fuse 7+ Runtime Server** サーバーを選択します。

   b. サーバーを右クリックし、コンテキストメニューで **Add and Remove** を選択します。

   c. **Add and Remove** ダイアログで **fuse-camel-cbr** プロジェクトを選択し、**< Remove** ボタンをクリックします。

   d. **Finish** をクリックします。

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 リポジトリーに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把握している。

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。ダウンロード後、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、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。
- どの場所に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、ローカルリポジトリーで現在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、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 mirror 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

Maven は、**packaging** の値、または **packaging** 値と **classifier** 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 つの部分を使用でき、最後の部分には数字以外の文字を使用できます。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

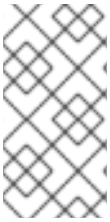
同じ POM からビルドされた内容が異なるアーティファクトを区別できるようにします。

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。