Red Hat Fuse 7.13

在 Apache Karaf 中使用 Fuse

在 Karaf 上使用红帽 Fuse 快速入门

Last Updated: 2024-06-06

Red Hat Fuse 7.13 在 Apache Karaf 中使用 Fuse

在 Karaf 上使用红帽 Fuse 快速入门

法律通告

摘要

开始在 Apache Karaf 上使用 Fuse 构建应用程序。

目录

前言

要开始使用 Fuse，您需要为 Apache Karaf 容器下载并安装文件。此处的信息和说明指导您安装、开发和构建第一个 Fuse 应用程序。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看我们的 CTO Chris Wright 信息。

第 1 章 在 KARAF 上使用 FUSE

要了解 Fuse on Karaf 上的 Fuse 以及在 Karaf 容器上安装、开发和构建第一个 Fuse 应用程序的信息，此处的信息和说明可帮助您实现这一目的。详情请查看以下主题：

- 第 1.1 节 “关于 Karaf 上的 Fuse”
- 第 1.2 节 “在 Karaf 上安装 Fuse”
- 第 1.3 节 “在 Karaf 上构建第一个 Fuse 应用程序”

1.1. 关于 KARAF 上的 FUSE

Apache Karaf 基于 OSGi 联盟中的 OSGi 标准。OSGi 源自电信行业，它用于开发可即时升级的网关服务器，而无需关闭服务器（称为 *热代码交换*）。因此，SGiOS 容器技术发现了各种其他用途，并被模块化应用程序（例如，Eclipse IDE）。

此容器技术的主要特性是：

- 特别适用于在独立模式运行。
- 对模块化(OSGi 捆绑包)的强支持，具有复杂的类加载支持。
- 容器中的多个依赖项版本可以并行部署（但在实践中需要注意）。
- 热代码交换功能，允许您在不关闭容器的情况下升级或替换模块。这是一个独特的功能，但为了使其正常工作，需要大量的努力。

注意： 不支持 Spring Dynamic Modules (Spring-DM)（将 Spring XML 与 Apache Karaf 中的 OSGi 服务层集成）。反之，您应该使用 Blueprint 框架。使用 Blueprint XML 不会阻止您使用 Spring 框架中的 Java 库：最新版本的 Spring 与 Blueprint 兼容。

1.2. 在 KARAF 上安装 FUSE

红帽客户门户网站上 Fuse 7.13 的标准安装软件包可从红帽客户门户下载。它安装 datacenter 容器的标准装配，并提供完整的 Fuse 技术堆栈。

先决条件

- 在红帽客户门户网站中 您需要一个全订阅帐户。
- 您必须登录到客户门户网站。
- 您必须已下载 CodeReady Studio 安装程序。
- 您必须在 Karaf 安装程序中下载 Fuse。

流程

1. 将 Apache Karaf 上 Fuse 下载的 **.zip** 归档文件解压缩到您的文件系统中 **FUSE_INSTALL** 的便捷位置。
2. 将管理员用户添加到 Fuse 运行时。
   a. 在文本编辑器中打开 **FUSE_INSTALL/etc/users.properties** 文件。

b. 删除以 **# admin = admin** 开头的行开头的 # 字符。

c. 删除以 **# _g_\:admingroup** 开头的行开头的 # 字符。

d. 自定义用户条目的用户名、**USERNAME** 和密码 **PASSWORD**，以便您有一个用户条目和 admin 组条目，如下所示（连续行）：

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

e. 保存 **etc/users.properties** 文件。

3. 运行 CodeReady Studio 安装程序，如下所示：

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

4. 在安装过程中：

a. 接受条款和条件。

b. 选择您首选的安装路径。

c. 选择 Java 8 JVM。

d. 在 **Select Platforms and Servers** 步骤中，通过单击 **Add** 并浏览到 **FUSE_INSTALL** 目录的位置，配置 FUSE 运行时上的 Fuse。

e. 在 **Select Additional Features to Install** 步骤中，选择 **Red Hat Fuse Tooling**。

5. CodeReady Studio 启动。出现 Searching **for** runtime 对话框时，单击 **OK** 以在 Karaf 运行时上创建 Fuse。

6. *（可选）*为了从命令行使用 Apache Maven，您需要安装并配置 Maven，如 在 本地设置 Maven所述。

注意

如果您只使用 CodeReady Studio，则不需要安装 Maven，因为 CodeReady Studio 已为您预安装并配置了 Maven。但是，如果您计划从命令行调用 Maven，则需要执行此步骤。

1.3. 在 KARAF 上构建第一个 FUSE 应用程序

这组说明可帮助您在 Karaf 上构建第一个 Fuse 应用程序。

先决条件

- 在红帽客户门户网站中 您需要一个全订阅帐户。
- 您必须登录到客户门户网站。
- 您必须已下载 CodeReady Studio 安装程序。
- 您必须已在 Karaf 上下载并成功安装了 Fuse。

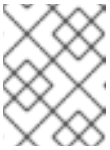
流程

1. 在 CodeReady Studio 中，创建一个新项目，如下所示：
   a. 选择 **File→New→Fuse Integration Project**。
   b. 在 **Project Name** 字段中输入 **fuse-camel-cbr**。
   c. 点击 **Next**。
   d. 在 **Select a Target Environment** 窗格中选择以下设置：
      - 选择 **Standalone** 作为部署平台。
      - 选择 **Karaf 上的 Karaf/Fuse** 作为运行时环境，并使用 **Runtime （可选）** 下拉菜单选择 **fuse-karaf-7.13.0.fuse-7_13_0-00012-redhat-00001 Runtime** 服务器作为目标运行时。
   e. 选择目标运行时后，会自动为您选择 **Camel Version**，字段将灰掉。
   f. 点击 **Next**。
   g. 在 **Advanced Project Setup** 窗格中，选择 **Beginner→Content Based Router - Blueprint DSL** 模板。
   h. 点 **Finish**。
   i. 如果系统提示您打开关联的 Fuse 集成透视图，请单击 **Yes**。
   j. 等待 CodeReady Studio 下载所需的工件，并在后台构建项目。

重要

如果您在 CodeReady Studio 中首次构建 Fuse 项目时，*向导将* 需要几分钟才能完成生成项目，因为它会从远程 Maven 存储库下载依赖项。在项目在后台构建时，请勿尝试中断向导或关闭 CodeReady Studio。

2. 将项目部署到服务器，如下所示：
   a. 在 **Servers** 视图(Fuse Integration 视角的左下角)中，如果服务器尚未启动，请选择 **fuse-karaf-7.13.0.fuse-7_13_0-00012-redhat-00001 Runtime Server** 服务器，再单击绿色箭头来启动它。



注意

如果您看到对话框，**警告：无法建立主机"localhost"的真实性。** 单击 **Yes** 以连接到服务器并访问 Karaf 控制台。

   b. 等待 **Console** 视图中看到类似如下的消息：

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

   c. 服务器启动后，切回到 **Servers** 视图，右键单击服务器，然后从上下文菜单中选择 **Add and Remove**。
   d. 在 **Add and Remove** 对话框中，选择 **fuse-camel-cbr** 项目，然后点击 **Add >** 按钮。

e. 点 **Finish**。

f. 您可以通过进入 **Terminal** 视图并输入 **bundle:list | tail** 来检查项目的 OSGi 捆绑包是否已启动。您应该看到类似如下的输出：

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

注意

Camel 路由启动后，它将在您的 Fuse 安装(*不在* **fuse-camel-cbr** 项目中)创建一个目录 **work/cbr/input**。

3. 将项目的 **src/main/data** 目录中找到的文件复制到 **FUSE_INSTALL/work/cbr/input** 目录中。您可以在系统文件浏览器中（在 Eclipse 之外）执行此操作。

4. 稍等片刻，然后在 **FUSE_INSTALL/work/cbr/output** 目录中查看由国家组织的相同文件：

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

   b. **work/cbr/output/uk**中的 **order2.xml** 和 **order4.xml**

   c. **order3.xml** 和 **order5.xml** in **work/cbr/output/us**

5. 取消部署项目，如下所示：

   a. 在 **Servers** 视图中，选择 **Red Hat Fuse 7+ Runtime Server** 服务器。

   b. 右键单击服务器，然后从上下文菜单中选择 **Add and Remove**。

   c. 在 **Add and Remove** 对话框中，选择 **fuse-camel-cbr** 项目，再单击 &lt ; **Remove** 按钮。

   d. 点 **Finish**。

第 2 章 在本地设置 MAVEN

典型的 Fuse 应用程序开发使用 Maven 来构建和管理项目。

以下主题描述了如何在本地设置 Maven：

- 第 2.1 节 “准备设置 Maven”
- 第 2.2 节 “将红帽软件仓库添加到 Maven”
- 第 2.3 节 “使用本地 Maven 存储库”
- 第 2.4 节 “使用环境变量或系统属性设置 Maven 镜像”
- 第 2.5 节 “关于 Maven 工件和协调”

2.1. 准备设置 MAVEN

Maven 是一个来自 Apache 的免费开源构建工具。通常，您可以使用 Maven 构建 Fuse 应用程序。

流程

1. 从 Maven 下载页面 下载最新版本的 Maven。
2. 确定您的系统已连接到互联网。
   在构建项目时，默认行为是 Maven 搜索外部存储库并下载所需的工件。Maven 查找可通过互联网访问的存储库。

   您可以更改此行为，以便 Maven 仅搜索位于本地网络上的存储库。也就是说，Maven 可以在离线模式下运行。在离线模式中，Maven 会在其本地存储库中查找工件。请参阅 第 2.3 节 “使用本地 Maven 存储库”。

2.2. 将红帽软件仓库添加到 MAVEN

要访问位于 Red Hat Maven 存储库中的工件，您需要将这些存储库添加到 Maven 的 **settings.xml** 文件中。Maven 在用户主目录的 **.m2** 目录中查找 **settings.xml** 文件。如果没有用户指定的 **settings.xml** 文件，Maven 将使用 **M2_HOME/conf/ settings.xml** 中的系统级 settings.xml 文件。

前提条件

您知道要在其中添加红帽软件仓库的 **settings.xml** 文件的位置。

流程

在 **settings.xml** 文件中，为红帽 **软件仓库** 添加软件仓库元素，如下例所示：

```
<?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. 使用本地 MAVEN 存储库

如果您在没有互联网连接的情况下运行容器，并且需要部署一个具有离线依赖项的应用程序，您可以使用 Maven 依赖项插件将应用的依赖项下载到 Maven 离线存储库中。然后，您可以将此自定义 Maven 离线存储库分发到没有互联网连接的机器。

流程

1. 在包含 **pom.xml** 文件的项目目录中，运行以下命令来下载 Maven 项目的存储库，如下所示：

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

   在本例中，构建项目所需的 Maven 依赖项和插件将下载到 **/tmp/my-project** 目录中。

2. 将此自定义 Maven 离线存储库在内部分发到没有互联网连接的任何机器。

2.4. 使用环境变量或系统属性设置 MAVEN 镜像

在运行应用程序时，您需要访问 Red Hat Maven 存储库中的工件。这些存储库添加到 Maven 的 **settings.xml** 文件中。Maven 检查 **settings.xml** 文件的以下位置：

- 查找指定的 url
- if not found looks for **${user.home}/.m2/settings.xml**
- 如果没有找到 **${maven.home}/conf/settings.xml**的查找
- 如果未找到，则查找 **${M2_HOME}/conf/settings.xml**
- 如果没有找到位置，则会创建空的 **org.apache.maven.settings.Settings** 实例。

2.4.1. 关于 Maven 镜像

Maven 使用一组远程存储库来访问工件，这些工件目前在本地存储库中不可用。存储库列表几乎始终包含 Maven Central 存储库，但对于 Red Hat Fuse，它还包含 Maven 红帽存储库。在某些情况下，无法访问不同的远程存储库，您可以使用 Maven 镜像的机制。镜像将特定的存储库 URL 替换为不同的存储库 URL，因此搜索远程工件时的所有 HTTP 流量都可以定向到单个 URL。

2.4.2. 在 settings.xml中添加 Maven 镜像

要设置 Maven 镜像，请在 Maven 的 **settings.xml** 中添加以下部分：

```
<mirror>
    <id>all</id>
    <mirrorOf>*</mirrorOf>
```

```
    <url>http://host:port/path</url>
</mirror>
```

如果在 **settings.xml** 文件中找不到以上部分，则不会使用 mirror。要在不提供 XML 配置的情况下指定全局镜像，您可以使用系统属性或环境变量。

2.4.3. 使用环境变量或系统属性设置 Maven 镜像

要使用环境变量或系统属性设置 Maven 镜像，您可以添加：

- 名为 **MAVEN_MIRROR_URL** 的环境变量到 **bin/setenv** 文件
- 名为 **mavenMirrorUrl** to **etc/system.properties** 文件的系统属性

2.4.4. 使用 Maven 选项指定 Maven 镜像 url

要使用替代的 Maven 镜像 url，在环境变量或系统属性之外，在运行应用程序时使用以下 maven 选项：

- **-DmavenMirrorUrl=mirrorId::mirrorUrl**
  for example, **-DmavenMirrorUrl=my-mirror::http://mirror.net/repository**
- **-DmavenMirrorUrl=mirrorUrl**
  例如，**-DmavenMirrorUrl=http://mirror.net/repository**。在本例中，<mirror> 的 <id> 只是一个镜像(mirror)。

2.5. 关于 MAVEN 工件和协调

在 Maven 构建系统中，基本构建块是一个 *工件*。构建后，工件的输出通常是一个存档，如 JAR 或 WAR 文件。

Maven 的一个关键方面是能够找到工件和管理它们之间的依赖关系。*Maven 协调* 是一组用于标识特定工件位置的值。基本协调有三个值，格式为：

groupId:artifactId:version

有时，Maven 通过 *打包值或使用打包* 值和 *分类器* 值增加一个基本的协调。*Maven 协调可以具有以下形式之一：*

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

以下是值的描述：

groupdId

*定义工件名称的范围。您通常使用所有或部分软件包名称作为组 ID。例如，**org.fusesource.example**。*

artifactId

定义相对于组 ID 的工件名称。

version

*指定工件的版本。版本号最多可有四个部分：**n.n.n.n**，其中版本号的最后一部分可以包含非数字字符。例如，**1.0-SNAPSHOT** 的最后一部分是字母数字字符子字符串 **0-SNAPSHOT**。*

打包

定义构建项目时生成的打包实体。对于 OSGi 项目，打包是 **捆绑包**。默认值为 **jar**。

分类器

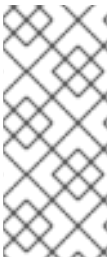
可让您区分从同一 POM 构建但具有不同内容的工件。

工件的 POM 文件中的元素定义工件的组 ID、工件 ID、打包和版本，如下所示：

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

要定义对上述工件的依赖项，您可以在 POM 文件中添加以下 dependencies 元素：

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



注意

不需要在前面的依赖项中指定 bundle 软件包类型，因为捆绑包只是特定类型的 JAR 文件，jar 是默认的 Maven 软件包类型。但是，如果您需要在依赖项中明确指定打包类型，您可以使用 type 元素。