Red Hat Fuse 7.13

部署到 Apache Karaf

将应用软件包部署到 Apache Karaf 容器中

Last Updated: 2024-06-12

Red Hat Fuse 7.13 部署到 Apache Karaf

将应用软件包部署到 Apache Karaf 容器中

法律通告

摘要

本指南描述了将应用程序部署到 Apache Karaf 容器中的选项。

目录

使开源包含更多 5

部分 I. 开发人员指南 6

第 1 章 OSGI 简介 7
1.1. 概述 7
1.2. APACHE KARAF 架构 7
1.3. JAAS 框架 7
1.4. OSGI 服务 8
1.5. OSGI 捆绑包 10

第 2 章 启动和停止 APACHE KARAF 11
2.1. 启动 APACHE KARAF 11
2.2. 停止 APACHE KARAF 13

第 3 章 基本安全性 16
3.1. 配置基本安全性 16

第 4 章 将 APACHE KARAF 安装为服务 19
4.1. 概述 19
4.2. 将 KARAF 作为服务运行 19
4.3. SYSTEMD 19
4.4. SYSV 19
4.5. SOLARIS SMF 20
4.6. WINDOWS 20
4.7. KARAF-SERVICE.SH 选项 20

第 5 章 构建 OSGI 捆绑包 22
5.1. 生成捆绑包项目 22
5.2. 修改现有 MAVEN 项目 22
5.3. 在捆绑包中打包 WEB 服务 25

第 6 章 热部署与手动部署 34
6.1. 热部署 34
6.2. 热取消部署捆绑包 34
6.3. 手动部署 34
6.4. 使用 BUNDLE:WATCH 自动重新部署捆绑包 36

第 7 章 生命周期管理 38
7.1. 捆绑包生命周期状态 38
7.2. 安装并解决捆绑包 39
7.3. 启动和停止捆绑包 39
7.4. 捆绑包开始级别 40
7.5. 指定捆绑包的启动级别 40
7.6. 系统启动级别 40

第 8 章 依赖项故障排除 41
8.1. 缺少依赖项 41
8.2. 未安装所需的功能或捆绑包 41
8.3. IMPORT-PACKAGE 标头不完整 41
8.4. 如何跟踪缺少的依赖项 42

第 9 章 部署功能 44
9.1. 创建功能 44

9.2. 创建自定义功能存储库 44
9.3. 在自定义功能存储库中添加功能 45
9.4. 将本地存储库 URL 添加到 FEATURES 服务 46
9.5. 在该功能中添加依赖功能 46
9.6. 向该功能添加 OSGI 配置 47
9.7. 自动部署 OSGI 配置 48

第 10 章 部署功能 49
10.1. 概述 49
10.2. 在控制台中安装 49
10.3. 在控制台中卸载 49
10.4. 热部署 50
10.5. 热取消部署功能文件 50
10.6. 在引导配置中添加功能 50

第 11 章 部署 PLAIN JAR 53
11.1. 使用嵌套方案转换 JAR 53
嵌套并安装 54

第 12 章 OSGI 服务 55
12.1. BLUEPRINT 容器 55
12.2. 导出服务 59
12.3. 导入服务 63
12.4. 发布 OSGI 服务 70
12.5. 访问 OSGI 服务 74
12.6. 与 APACHE CAMEL 集成 78

第 13 章 使用 JMS 代理进行部署 81
13.1. AMQ 7 快速入门 81
13.2. 使用 ARTEMIS 核心客户端 84

第 14 章 故障切换部署 85
14.1. 使用简单锁定文件系统 85
14.2. 使用 JDBC 锁定系统 86
14.3. 容器级别的锁定 89

第 15 章 URL 处理程序 91
15.1. 文件 URL 处理程序 91
15.2. HTTP URL 处理程序 91
15.3. MVN URL HANDLER 91
15.4. 嵌套 URL 处理程序 95
15.5. WAR URL 处理程序 96

部分 II. 用户指南 99

第 16 章 在 APACHE KARAF 中部署指南简介 100
16.1. FUSE 配置简介 100
16.2. OSGI 配置 100
16.3. 配置文件 100
16.4. 高级 UNDERTOW 配置 101
16.5. 配置文件命名规则 103
16.6. 设置 JAVA 选项 104
16.7. 配置控制台命令 104
16.8. JMX CONFIGMBEAN 105
16.9. 使用控制台 105

16.10. 置备 118

第 17 章 使用远程连接管理容器 136
17.1. 为远程访问配置容器 136
17.2. 远程连接和断开连接 136
17.3. 停止远程容器 144

第 18 章 使用 MAVEN 构建 145
18.1. MAVEN 目录结构 145
18.2. APACHE KARAF 的 BOM 文件 147

第 19 章 MAVEN INDEXER 插件 149

第 20 章 LOG 150
20.1. 配置文件 150
20.2. 命令 154
20.3. JMX LOGMBEAN 160
20.4. 高级配置 161

第 21 章 安全性 169
21.1. REALMS 169

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。

部分 I. 开发人员指南

本节包含开发人员的信息。

第 1 章 OSGI 简介

摘要

OSGi 规范通过定义可简化构建、部署和管理复杂应用程序的运行时框架来支持模块化应用程序开发。

1.1. 概述

Apache Karaf 是基于 OSGi 的运行时容器，用于部署和管理捆绑包。Apache Karaf 还提供原生操作系统集成，并可作为服务集成到操作系统中，以便生命周期绑定到操作系统。

Apache Karaf 具有以下结构：

- Apache Karaf - 围绕 OSGi 容器实施的打包程序层，为将 OSGi 容器部署为运行时服务器提供支持。Fuse 提供的运行时功能包括热部署、管理和管理功能。
- OSGi 框架 - 实施 OSGi 功能，包括管理依赖项和捆绑包生命周期

1.2. APACHE KARAF 架构

Apache Karaf 使用以下功能扩展了 OSGi 层：

- **控制台 - 控制台管理** 服务、安装和管理应用程序和库，并与 Fuse 运行时交互。它提供了控制台命令来管理 Fuse 实例。请参阅 Apache Karaf 控制台参考。
- **logging** - logging 子系统提供控制台命令来显示、查看和更改日志级别。
- **部署** - 支持使用捆绑包和 **bundle: start** 命令和应用程序热部署，手动部署 OSGi 捆绑包。请参阅 第 6.1 节 “热部署”。
- **置备** - 提供多个安装应用程序和库的机制。请参阅 第 9 章 *部署功能*。
- configuration - 存储在 *InstallDir*/**etc** 文件夹中的属性文件会被持续监控，它们更改会以可配置的间隔自动传播到相关服务。
- **蓝图 - 是一种依赖项注入框架，简化了与 OSGi 容器的交互。例如，提供用于导入和导出 OSGi 服务的标准 XML 元素。当蓝图配置文件复制到热部署文件夹时，红帽 Fuse 会根据情况生成 OSGi 捆绑包并实例化蓝图上下文。**

1.3. JAAS 框架

1.3.1. 概述

OSGi 联盟是一个独立组织，负责定义 OSGi 服务版本 4 的功能。OSGi 服务平台是一组开放规格，简化了复杂软件应用程序的构建、部署和管理。

OSGi 技术通常被称为 Java 的动态模块系统。OSGi 是 Java 的框架，它使用捆绑程序以模块化方式部署 Java 组件，并处理依赖项、版本控制、类路径控制和类加载。OSGi 的生命周期管理允许您在不关闭 JVM 的情况下加载、启动和停止捆绑包。

OSGi 为 Java、卓越的类加载架构和 registry 为服务提供最佳运行时平台。捆绑包可以导出服务、运行进程，并且其依赖项管理。每个捆绑包都可以满足由 OSGi 容器管理的要求。

Fuse 使用 Apache Felix 作为其默认的 OSGi 实施。框架层组成了安装捆绑包的容器。框架以动态、可扩展的方式管理捆绑包的安装和更新，并管理捆绑包和服务之间的依赖项。

1.3.2. OSGi 架构

OSGi 框架包含以下内容：

- 捆绑包 - 组成应用程序的逻辑模块。请参阅 第 1.5 节 “OSGi 捆绑包”。
- 服务层 - 提供模块及其包含组件之间的通信。这个层与生命周期层紧密集成。请参阅 第 1.4 节 “OSGi 服务”。
- 生命周期层 - 提供对底层 OSGi 框架的访问。此层处理单个捆绑包的生命周期，以便您可以动态管理应用程序，包括启动和停止捆绑包。
- 模块层 - 提供 API，以管理捆绑包打包、依赖项解析和类加载。
- 执行环境 - JVM 的配置。此环境使用配置文件来定义捆绑包可以在其中工作的环境。
- 安全层 - 基于 Java 2 安全性的可选层，具有额外的限制和增强。

框架中的每个层都取决于它下的层。例如，生命周期层需要模块层。模块层可以在没有生命周期和服务层的情况下使用。

1.4. OSGI 服务

1.4.1. 概述

OSGi 服务是 Java 类或服务接口，其服务属性定义为名称/值对。服务属性区分通过同一服务接口提供服务的服务供应商。

OSGi 服务由服务界面定义，它作为服务对象实施。服务的功能由它实施的接口定义。因此，不同的应用程序可以实施相同的服务。

服务接口允许捆绑包通过绑定接口而不是实现来交互。服务接口应尽可能使用几个实施详情来指定。

1.4.2. OSGi 服务 registry

在 OSGi 框架中，服务层使用发布、查找和绑定服务模型提供 第 1.5 节 “OSGi 捆绑包” 及其包含组件之间的通信。该服务层包含一个服务 registry，其中：

- 服务供应商将服务注册到框架，供其他捆绑包使用
- 服务请求者查找服务并绑定到服务提供商

服务归捆绑包所有，并在其内运行。捆绑包在一个或多个 Java 接口下将服务的实施注册到框架服务 registry。因此，服务的功能可供框架控制下的其他捆绑包使用，其他捆绑包可以查找和使用该服务。lookup 使用 Java 接口和服务属性来执行。

每个捆绑包都可以使用其接口及其属性的完全限定名称在服务 registry 中注册多个服务。捆绑包使用带有 LDAP 语法的名称和属性来查询服务 registry。

捆绑包负责运行时服务依赖项管理活动，包括发布、发现和绑定。捆绑包也可以适应绑定捆绑包的服务的动态可用性(arrival 或 departure)生成的更改。

事件通知

服务接口由捆绑包创建的对象实施。捆绑包可以：

- 注册服务
- 搜索服务
- 当注册状态发生变化时接收通知

OSGi 框架提供了一个事件通知机制，因此服务请求者可以在发生服务 registry 中的更改时接收通知事件。这些更改包括发布或检索特定服务以及注册、修改或取消注册服务的时间。

服务调用模型

当捆绑包希望使用服务时，它会查找服务，并将 Java 对象作为普通的 Java 调用调用。因此，服务的调用是同步的，并在同一线程中发生。您可以使用回调进行更多异步处理。参数作为 Java 对象引用传递。XML 不需要 marshalling 或 Intermediary 规范格式。OSGi 为服务不可用问题提供了解决方案。

JAAS 框架服务

除了您自己的服务外，OSGi 框架还提供以下可选服务来管理框架的操作：

- Package Admin service- 允许管理代理通过检查共享软件包的状态来定义管理 Java 软件包共享的策略。它还允许管理代理刷新软件包，并根据需要停止和重启捆绑包。此服务可让管理代理在卸载或更新导出捆绑包时就任何共享软件包做出决策。
  该服务还提供刷新自上次刷新后删除或更新的软件包的方法，以及显式解析特定捆绑包。此服务也可以在运行时跟踪捆绑包之间的依赖关系，允许您查看升级可能会影响哪些捆绑包。

- 启动级别服务- 启用管理代理来控制捆绑包的启动和停止顺序。该服务为每个捆绑包分配一个起始级别。管理代理可以修改捆绑包的起始级别，并设置框架的活动启动级别，以启动和停止适当的捆绑包。只有启动级别小于或等于的捆绑包，这个活跃启动级别可以处于活跃状态。

- URL Handlers 服务- 动态将 Java 运行时扩展为 URL 方案和内容处理程序，使任何组件能够提供额外的 URL 处理程序。

- 权限管理服务- 启用 OSGi 框架管理代理，以管理特定捆绑包的权限，并为所有捆绑包提供默认值。捆绑包可以有一组权限，用于验证它是否被授权执行特权代码。您可以通过实时更改策略并为新安装的组件添加新策略来动态操作权限。策略文件用于控制捆绑包可以执行的操作。

- 条件 Permission Admin service- 在检查权限时，扩展具有可在特定条件为 true 或 false 的 Permission Admin 服务的权限。这些条件决定了权限应用到的捆绑包的选择。权限在设置后立即激活。

OSGi 框架服务在 OSGi 联盟网站上 发行 4 发布后的 4版本 4 规范的 单独章节中详细介绍。

OSGi Compendium 服务

除了 OSGi 框架服务外，OSGi 联盟还定义了一组可选的、标准化的编译型服务。OSGi compendium 服务为日志记录和首选项等任务提供 API。这些服务在 OSGi 联盟 Web 站点上的版本 4 下载页面 提供的 OSGi Service Platform （服务编译）中进行了描述。

Configuration Admin compendium 服务类似于一个中央 hub，它会保留配置信息并将其分发到感兴趣的方。Configuration Admin 服务指定部署的捆绑包的配置信息，并确保捆绑包在激活时接收这些数据。捆绑包的配置数据是一个名称值对列表。请参阅 第 1.2 节 “Apache Karaf 架构”。

1.5. OSGI 捆绑包

概述

借助 OSGi，您可以将应用程序模块化为捆绑包。每个捆绑包都是紧密耦合的、可动态加载的类、JAR 和配置文件的集合，明确声明任何外部依赖项。在 OSGi 中，捆绑包是主要的部署格式。捆绑包是打包在 JAR 中的应用程序，可以安装、启动、停止、更新和删除。

OSGi 为开发捆绑包提供动态、简洁且一致的编程模型。从实施中分离了服务的规格(Java 接口)可以简化开发和部署。

OSGi 捆绑包抽象允许模块共享 Java 类。这是静态的重复使用形式。当依赖捆绑包启动时，共享类必须可用。

捆绑包是一个 JAR 文件，其中包含其 OSGi 清单文件中的元数据。捆绑包包含类文件和（可选）其他资源和原生库。您可以明确声明捆绑包中的哪些软件包在外部可见（导出软件包），以及捆绑包需要哪些外部软件包（导入的软件包）。

模块层处理捆绑包和从其他捆绑包隐藏软件包之间的 Java 软件包打包和共享。OSGi 框架在捆绑包之间动态解决依赖项。框架执行捆绑包解析，以匹配导入和导出的软件包。它还可以管理已部署捆绑包的多个版本。

OSGi 中的类加载

OSGi 使用图形模型进行类加载，而不是树状模型（如 JVM 使用）。捆绑包可以以标准化的方式共享和重新使用类，无运行时类加载冲突。

每个捆绑包都有自己的内部类路径，以便在需要时可以充当独立的单元。

OSGi 中类加载的好处包括：

- 在捆绑包之间直接共享类。不需要将 JAR 提升到父类加载器。
- 您可以同时部署同一类的不同版本，且没有冲突。

第 2 章 启动和停止 APACHE KARAF

摘要

Apache Karaf 为启动和停止服务器提供了简单的命令行工具。

2.1. 启动 APACHE KARAF

部署 Apache Karaf 运行时的默认方法是使用活动控制台将它部署为单机服务器。您还可以将运行时部署为没有控制台的后台进程。

2.1.1. 设置您的环境

您可以直接从安装的 bin 子目录启动 Karaf 运行时，而无需修改您的环境。但是，如果您要在其他文件夹中启动它，您需要将 Karaf 安装的 bin 目录添加到 PATH 环境变量中，如下所示：

Windows

```
set PATH=%PATH%;InstallDir\bin
```

Linux/UNIX

```
export PATH=$PATH,InstallDir/bin`
```

2.1.2. 在控制台模式中启动运行时

如果您要从安装目录启动 Karaf 运行时，请使用以下命令：

Windows

```
bin\fuse.bat
```

Linux/UNIX

```
./bin/fuse
```

如果 Karaf 正确启动，您应该在控制台中看到以下内容：

```
Red Hat Fuse starting up. Press Enter to open the shell now...
100%
[========================================================================]

Karaf started in 8s. Bundle stats: 220 active, 220 total


 _____         _  _  _      _   ______
|  _ \ ___  __| | | | | | __ _| |_  |  ____|   _ ___  ___
| |_) / _ \/ _` | | |_| |/ _` | __| | |_ | | | / __|/ _ \
|  _ <  __/ (_| | |  _  | (_| | |_  |  _|| |_| \__ \  __/
|_| \_\___|\__,_| |_| |_|\__,_|\__| |_|   \__,_|___/___|

  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/
```

```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```

注意

自版本 Fuse 6.2.1 起，在控制台模式中启动会创建两个进程：父进程 ./bin/karaf，该进程正在执行 Karaf 控制台；而子进程（在 java JVM 中执行 Karaf 服务器）。但是，关闭过程与之前相同。也就是说，您可以使用 Ctrl-D 或 osgi:shutdown 从控制台关闭服务器，这会终止这两个进程。

2.1.3. 在服务器模式中启动运行时

在服务器模式中启动 会在后台运行 Apache Karaf，无需本地控制台。然后，您将使用远程控制台连接到正在运行的实例。详情请查看 第 17.2 节 “远程连接和断开连接”。

若要在服务器模式中启动 Karaf，请运行以下命令

Windows

```
bin\start.bat
```

Linux/UNIX

```
./bin/start
```

2.1.4. 在客户端模式中启动运行时

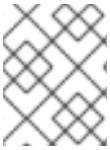
在生产环境中，您可能想让运行时实例只能通过本地控制台访问。换句话说，您无法通过 SSH 控制台端口远程连接到运行时。您可以通过在客户端模式中启动运行时来完成此操作：

Windows

```
bin\fuse.bat client
```

Linux/UNIX

```
./bin/fuse client
```



注意

在客户端模式中启动只阻止 SSH 控制台端口（通常是端口 8101）。其他 Karaf 服务器端口（例如，JMX 管理 RMI 端口）正常打开。

2.1.5. 在调试模式下运行 Fuse

在调试模式下运行 Fuse 有助于更有效地识别和解决错误。默认禁用这个选项。启用后，Fuse 在端口 5005 上启动 JDWP 套接字。

您有三种 *在调试模式下运行* Fuse 的方法。

- 第 2.1.5.1 节 “使用 Karaf 环境变量”
- 第 2.1.5.2 节 “运行 Fuse debug”
- 第 2.1.5.3 节 “运行 Fuse debug”

2.1.5.1. 使用 Karaf 环境变量

这个方法启用了 KARAF_DEBUG 环境变量(=1)，然后启动容器。

```
$ export KARAF_DEBUG=1
$ bin/start
```

2.1.5.2. 运行 Fuse debug

这个方法运行 debug，其中 suspend 选项被设置为 n (no)。

```
$ bin/fuse debug
```

2.1.5.3. 运行 Fuse debug

这个方法运行 debug，其中 suspend 选项被设置为 y （是）。

注意

将 suspend 设为 yes 会导致 JVM 仅在运行main （） 之前暂停，直到调试器附加为止，然后它恢复执行。

```
$ bin/fuse debugs
```

2.2. 停止 APACHE KARAF

您可以从控制台内或使用 stop 脚本停止 Apache Karaf 实例。

2.2.1. 从本地控制台停止实例

如果您通过运行 fuse 或 fuse 客户端 启动 Karaf 实例，您可以在karaf > 提示符下执行以下操作之一来停止它：

- 类型 关闭
- 按 Ctrl+D

2.2.2. 停止在服务器模式下运行的实例

您可以通过调用 InstallDir/bin 目录中的 stop (.bat) 来停止本地运行 Karaf 实例(root 容器)，如下所示：

Windows

```
bin\stop.bat
```

Linux/UNIX

```
./bin/stop
```

Karaf **stop** 脚本调用的关闭机制与 Apache Tomcat 中实施的关闭机制类似。Karaf 服务器打开专用关闭端口（与 SSH 端口相同），以接收关闭通知。默认情况下，会随机选择关闭端口，但如果您愿意，您可以将它配置为使用特定端口。

您可以通过在 **InstallDir/etc/config.properties** 文件中设置以下属性来自定义关闭端口：

karaf.shutdown.port

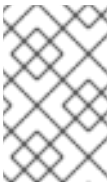
指定用作关闭端口的 TCP 端口。将此属性设置为 **-1** 可禁用端口。默认为 **0 (用于随机端口)**。

注意

如果要使用 **bin/stop** 脚本关闭远程主机上运行的 Karaf 服务器，则需要将此属性设置为与远程主机的关闭端口相等。但请注意，此设置也会影响位于与 **etc/config.properties** 文件相同的主机上的 Karaf 服务器。

karaf.shutdown.host

指定关闭端口绑定到的主机名。此设置对于多主目录主机非常有用。默认为 **localhost**。



注意

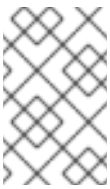
如果要使用 **bin/stop** 脚本关闭远程主机上运行的 Karaf 服务器，则需要将此属性设置为远程主机的主机名（或 IP 地址）。但请注意，此设置也会影响位于与 **etc/config.properties** 文件相同的主机上的 Karaf 服务器。

karaf.shutdown.port.file

在 Karaf 实例启动后，它会将当前关闭端口写入此属性指定的文件中。**stop** 脚本读取此属性指定的文件，以发现当前关闭端口的值。默认为 **${karaf.data}/port**。

karaf.shutdown.command

指定必须发送到关闭端口的 UUID 值，才能触发关闭。这提供了元素级别的安全性，只要 UUID 值保留了一个 secret。例如，**etc/config.properties** 文件可以被读取保护，以防止常规用户读取这个值。当 Apache Karaf 首次启动时，会自动生成随机 UUID 值，并将此设置写入 **etc/config.properties** 文件的末尾。或者，如果已经设置了 **karaf.shutdown.command**，则 Karaf 服务器将使用预先存在的 UUID 值（如果需要，您可以自定义 UUID 设置）。



注意

如果要使用 **bin/stop** 脚本关闭在远程主机上运行的 Karaf 服务器，则需要将此属性设置为等于远程主机 **karaf.shutdown.command** 的值。但请注意，此设置也会影响位于与 **etc/config.properties** 文件相同的主机上的 Karaf 服务器。

2.2.3. 停止远程实例

您可以停止在远程主机上运行的容器实例，如 第 17.3 节 “停止远程容器” 所述。

第 3 章 基本安全性

本章介绍了首次启动 Karaf 之前配置安全性的基本步骤。默认情况下，Ramen 是安全的，但其任何服务都无法远程访问。本章介绍了如何启用对 Karaf 公开的端口的安全访问。

3.1. 配置基本安全性

3.1.1. 概述

默认情况下，Apache Karaf 运行时会防止网络攻击，因为其所有公开端口都需要用户身份验证，并且最初没有定义任何用户。换句话说，默认远程无法访问 Apache Karaf 运行时。

如果要远程访问运行时，您必须首先自定义安全配置，如下所述。

3.1.2. 在启动容器前

如果要启用对 Karaf 容器的远程访问，您必须在启动容器前创建一个安全 JAAS 用户：

3.1.3. 创建安全 JAAS 用户

默认情况下，没有为容器定义 JAAS 用户，这样可有效地禁用远程访问（无法登录）。

要创建安全 JAAS 用户，请编辑 **InstallDir/etc/users.properties** 文件并添加一个新的 user 字段，如下所示：

```
Username=Password,admin
```

其中 **Username** 和 **Password** 是新用户凭证。**admin** 角色授予此用户访问容器的所有管理和管理功能的特权。

不要定义带有前导 0 的数字用户名。此类用户名总是会导致登录尝试失败。这是因为，当输入显示为数字时，控制台使用的 Karaf shell 会丢弃前导零。例如：

```
karaf@root> echo 0123
123
karaf@root> echo 00.123
0.123
karaf@root>
```

警告

强烈建议您使用强密码定义自定义用户凭证。

3.1.4. 基于角色的访问控制

Karaf 容器支持基于角色的访问控制，规范通过 JMX 协议、Karaf 命令控制台和 Fuse 管理控制台的访问。当为用户分配角色时，您可以从标准角色集合中选择，这提供了 表 3.1 “访问控制的标准角色” 中描述的访问级别。

表 3.1. 访问控制的标准角色

角色	描述
viewer	授予容器的只读访问权限。
Manager	在适当的级别，为希望部署和运行应用程序的普通用户授予读写访问权限。但会阻止对敏感容器配置设置的访问。
admin	授予容器不受限制的访问权限。
ssh	通过 SSH 端口授予远程控制台访问的权限。

有关基于角色的访问控制的详情，请参阅基于角色的访问控制。

3.1.5. Apache Karaf 容器公开的端口

容器公开以下端口：

- 通过 Apache Karaf shell 命令，控制台端口 criu-wagon 启用容器实例的远程控制。此端口默认为启用，并由 JAAS 身份验证和 SSH 保护。
- JMX 端口 criu-criu 通过 JMX 协议启用容器管理。此端口默认为启用，并由 JAAS 身份验证进行保护。
- Web 控制台端口 criu- iwlprovides 对可托管 Web 控制台 servlet 的嵌入式 Undertow 容器提供访问权限。默认情况下，Fuse Console 安装在 Undertow 容器中。

3.1.6. 启用远程控制台端口

当满足以下任一条件时，您可以访问远程控制台端口：

- JAAS 至少配置有一组登录凭据。
- Karaf 运行时 没有在 客户端模式中启动（客户端模式会完全禁用远程控制台端口）。

例如，要从运行容器的同一机器登录到远程控制台端口，请输入以下命令：

```
./client -u Username -p Password
```

其中 Username 和 Password 是带有 ssh 角色的 JAAS 用户的凭据。通过远程端口访问 Karaf 控制台时，您的特权取决于分配给用户在 etc/users.properties 文件中的角色。如果要访问完整的控制台命令集合，用户帐户必须具有 admin 角色。

3.1.7. 在远程控制台端口上增强安全性

您可以使用以下措施在远程控制台端口上增强安全性：

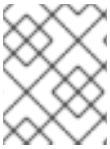
- 确保 JAAS 用户凭据具有强密码。
- 自定义 X.509 证书（用自定义密钥对替换 Java 密钥存储文件 InstallDir/etc/host.key ）。

3.1.8. 启用 JMX 端口

JMX 端口默认是启用的，并由 JAAS 身份验证进行保护。要访问 JMX 端口，您必须已经使用至少一组登录凭据配置了 JAAS。要连接到 JMX 端口，请打开 JMX 客户端（如 jconsole）并连接到以下 JMX URI：

```
service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-root
```

您还必须为 JMX 客户端提供有效的 JAAS 凭据才能连接。



注意

通常，JMX URI 的尾部格式为 /karaf-*ContainerName*。如果将容器名称从 root 更改为其他名称，您必须相应地修改 JMX URI。

3.1.9. 在 Fuse 控制台端口上增强安全性

Fuse 控制台已经由 JAAS 身份验证保护。要添加 SSL 安全性，请参阅 保护 Undertow HTTP 服务器。

第 4 章 将 APACHE KARAF 安装为服务

本章介绍了如何使用提供的模板启动 Apache Karaf 实例作为系统服务的信息。

4.1. 概述

使用服务脚本模板，您可以使用特定于操作系统的初始化脚本运行 Karaf 实例。您可以在 bin/contrib 目录下找到这些模板。

4.2. 将 KARAF 作为服务运行

karaf-service.sh 工具可帮助您自定义模板。这个工具会自动识别操作系统和默认 init 系统，并生成可随时使用的 init 脚本。您还可以通过设置 JAVA_HOME 和一些其他环境变量来自定义脚本，使其适应环境。

生成的脚本由两个文件组成：

- init 脚本
- init 配置文件

4.3. SYSTEMD

当 karaf-service.sh 程序标识 systemd 时，它会生成三个文件：

- 管理 Apache Karaf 容器的 systemd 单元文件。
- 包含 root Apache Karaf 容器使用的变量的 systemd 环境文件。
- *（不支持）* systemd 模板单元文件来管理 Apache Karaf 子容器。

例如，要为在 /opt/karaf-4 上安装的 Karaf 实例设置服务，请为服务提供名称 karaf-4 ：

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.service"
Writing service configuration file ""/opt/karaf-4/etc/karaf-4.conf"
Writing service file "/opt/karaf-4/bin/contrib/karaf-4@.service"
$ sudo cp /opt/karaf-4/bin/contrib/karaf-4.service /etc/systemd/system
$ sudo systemctl enable karaf-4.service
```

4.4. SYSV

当 karaf-service.sh 程序标识 SysV 系统时，它会生成两个文件：

- 管理 Apache Karaf 容器的初始化脚本。
- 包含 root Apache Karaf 容器使用的变量的环境文件。

例如，要为在 /opt/karaf-4 上安装的 Karaf 实例设置服务，请为服务提供名称 karaf-4 ：

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4"
Writing service configuration file "/opt/karaf-4/etc/karaf-4.conf"
```

```
$ sudo ln -s /opt/karaf-4/bin/contrib/karaf-4 /etc/init.d/
$ sudo chkconfig karaf-4 on
```

注意

要在引导时启用服务启动，请参考您的操作系统 init 指南。

4.5. SOLARIS SMF

当 karaf-service.sh 实用程序识别 Solaris 操作系统时，它会生成一个文件。

例如，要为在 /opt/karaf-4 上安装的 Karaf 实例设置服务，请为服务提供名称karaf-4 ：

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.xml"
$ sudo svccfg validate /opt/karaf-4/bin/contrib/karaf-4.xml
$ sudo svccfg import /opt/karaf-4/bin/contrib/karaf-4.xml
```

注意

生成的 SMF 描述符被定义为临时的，因此您只能执行 start 方法一次。

4.6. WINDOWS

通过 winsw 支持将 Apache Karaf 安装为 Windows 服务。

要将 Apache Karaf 作为 Windows 服务安装，请执行以下步骤：

1. **将 karaf-service-win.exe 文件重命名为 karaf-4.exe。**
2. **将 karaf-service-win.xml 文件重命名为 karaf-4.xml。**
3. **根据需要自定义服务描述符。**
4. **使用服务可执行文件安装、启动和停止该服务。**

例如：

```
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe install
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe start
```

4.7. KARAF-SERVICE.SH 选项

您可以将 karaf-service.sh 工具的选项指定为命令行选项或设置环境变量，如下所示：

命令行选项	环境变量	描述
-k	**KARAF_SERVICE_PATH**	Karaf 安装路径

命令行选项	环境变量	描述
-d	**KARAF_SERVICE_DATA**	Karaf 数据路径（默认为 **${KARAF_SERVICE_PATH}/data**）
-c	**KARAF_SERVICE_CONF**	Karaf 配置文件（默认为 **${KARAF_SERVICE_PATH}/etc/${KARAF_SERVICE_NAME}.conf**）
-t	**KARAF_SERVICE_ETC**	Karaf **等** 路径（默认为 **${KARAF_SERVICE_PATH}/etc}**）
-p	**KARAF_SERVICE_PIDFILE**	Karaf PID 路径（默认为 **${KARAF_SERVICE_DATA}/${KARAF_SERVICE_NAME}.pid**）
-n	**KARAF_SERVICE_NAME**	Karaf 服务名称（默认为 **karaf**）
-e	**KARAF_ENV**	为服务指定一个环境变量设置 **NAME=VALUE** （可以多次指定）
-u	**KARAF_SERVICE_USER**	Karaf 用户
-g	**KARAF_SERVICE_GROUP**	Karaf 组（默认为 **${KARAF_SERVICE_USER}** ）
-l	**KARAF_SERVICE_LOG**	Karaf 控制台日志（默认为 **${KARAF_SERVICE_DATA}/log/${KARAF_SERVICE_NAME}-console.log**）
-f	**KARAF_SERVICE_TEMPLATE**	要使用的模板文件
-x	**KARAF_SERVICE_EXECUTABLE**	Karaf 可执行名称（默认为 **karaf** mvapichmust 支持 **守护进程 和停止** 命令）
-h		帮助信息

第 5 章 构建 OSGI 捆绑包

摘要

本章论述了如何使用 Maven 构建 OSGi 捆绑包。为了构建捆绑包，Maven 捆绑包插件会扮演一个关键角色，因为它可让您自动执行 OSGi 捆绑包标头的生成（否则会是一个繁琐的任务）。Maven archetypes（生成完整的示例项目）也可以为您的捆绑包项目提供起点。

5.1. 生成捆绑包项目

5.1.1. 使用 Maven archetypes 生成捆绑包项目

为了帮助您快速开始，您可以调用 Maven archetype 来生成 Maven 项目的初始概述(Maven archetype 与项目向导类似)。以下 Maven archetype 生成用于构建 OSGi 捆绑包的项目。

5.1.2. Apache Camel archetype

Apache Camelosgi archetype 创建一个项目，用于构建可部署到 OSGi 容器中的路由。

以下示例演示了如何使用带有 coordinates, *GroupId*:*ArtifactId*:*Version* , .

```
mvn archetype:generate \
 -DarchetypeGroupId=org.apache.camel.archetypes \
 -DarchetypeArtifactId=camel-archetype-blueprint \
 -DarchetypeVersion=2.23.2.fuse-7_13_0-00013-redhat-00001
```

运行此命令后，Maven 会提示您指定 GroupId、ArtifactId 和 Version。

5.1.3. 构建捆绑包

默认情况下，前面的 archetypes 在新目录中创建一个项目，其名称与指定的工件 ID *ArtifactId* 相同。要构建由新项目定义的捆绑包，请打开命令提示符，进入项目目录（即包含 pom.xml 文件的目录），并输入以下 Maven 命令：

```
mvn install
```

此命令的效果是编译所有 Java 源文件，以在 *ArtifactId*/target 目录下生成捆绑包 JAR，然后在本地 Maven 存储库中安装生成的 JAR。

5.2. 修改现有 MAVEN 项目

5.2.1. 概述

如果您已经有一个 Maven 项目，且您想要修改它以便生成 OSGi 捆绑包，请执行以下步骤：

1. 第 5.2.2 节 “将软件包类型改为 bundle”.
2. 第 5.2.3 节 “在 POM 中添加捆绑包插件”.
3. 第 5.2.4 节 “自定义捆绑包插件”.
4. 第 5.2.5 节 “自定义 JDK 编译器版本”.

5.2.2. 将软件包类型改为 bundle

通过将软件包类型改为项目的 pom.xml 文件中的 捆绑包，将 Maven 配置为生成 OSGi 捆绑包。将 packaging 元素的内容改为 捆绑包，如下例所示：

```
<project ... >
  ...
  <packaging>bundle</packaging>
  ...
</project>
```

此设置的影响是选择 Maven 捆绑包插件 maven-bundle-plugin，以执行此项目的打包。但是，除非将捆绑包插件明确添加到 POM 中，此设置本身不会起作用。

5.2.3. 在 POM 中添加捆绑包插件

要添加 Maven 捆绑包插件，请将以下示例 插件 元素复制并粘贴到项目的 pom.xml 文件的 project/build/plugins 部分：

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
    <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>3.3.0</version>
```

```
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${project.groupId}.${project.artifactId}
            </Bundle-SymbolicName>
            <Import-Package>*</Import-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

其中，捆绑包插件由 instructions 元素中的设置进行配置。

5.2.4. 自定义捆绑包插件

有关为 Apache CXF 配置捆绑包插件的一些特定建议，请参考 第 5.3 节 “在捆绑包中打包 Web 服务”。

5.2.5. 自定义 JDK 编译器版本

几乎需要在 POM 文件中指定 JDK 版本。如果您的代码使用 Java 语言的任何现代功能，如通用、静态导入等，且您已在 POM 中自定义 JDK 版本，则 Maven 将无法编译源代码。将 JAVA_HOME 和 PATH 环境变量设置为 JDK 的正确值不足，还必须修改 POM 文件。

要配置 POM 文件，以便它接受 JDK 1.8 中引入的 Java 语言功能，请在 POM 中添加以下 maven-compiler-plugin 插件设置（如果它们尚不存在）：

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
```

```
  </build>
  ...
</project>
```

5.3. 在捆绑包中打包 WEB 服务

5.3.1. 概述

本节介绍如何为 Apache CXF 应用程序修改现有 Maven 项目，以便项目生成适合在 Red Hat Fuse OSGi 容器中部署的 OSGi 捆绑包。要转换 Maven 项目，您需要修改项目的 POM 文件和项目的 Blueprint 文件（位于 META-INF/spring中）。

5.3.2. 修改 POM 文件来生成捆绑包

要配置 Maven POM 文件来生成捆绑包，基本上有两个更改，您需要进行两个更改：将 POM 的软件包类型改为 bundle; 并将 Maven 捆绑包插件添加到 POM 中。详情请查看 第 5.1 节 “生成捆绑包项目”。

5.3.3. 必需的导入软件包

为了让应用程序使用 Apache CXF 组件，您需要将其软件包导入到应用程序的捆绑包中。由于 Apache CXF 中依赖项的复杂性质，您无法依赖 Maven 捆绑包插件或 bnd 工具来自动确定所需的导入。您需要显式声明它们。

您需要将以下软件包导入到捆绑包中：

```
javax.jws
javax.wsdl
javax.xml.bind
javax.xml.bind.annotation
javax.xml.namespace
javax.xml.ws
org.apache.cxf.bus
org.apache.cxf.bus.spring
org.apache.cxf.bus.resource
org.apache.cxf.configuration.spring
org.apache.cxf.resource
org.apache.cxf.jaxws
org.springframework.beans.factory.config
```

5.3.4. Maven 捆绑插件指令示例

例 5.1 “配置强制导入软件包” 演示了如何在 POM 中配置 Maven 捆绑包插件来导入强制软件包。必需的导入软件包在 Import-Package 元素中以逗号分隔的列表的形式出现。请注意通配符的外观，即 *，作

为列表的最后一个元素。通配符可确保扫描当前捆绑包中的 Java 源文件，以发现需要导入哪些额外软件包。

例 5.1. 配置强制导入软件包

```
<project ... >
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            ...
            <Import-Package>
              javax.jws,
              javax.wsdl,
              javax.xml.bind,
              javax.xml.bind.annotation,
              javax.xml.namespace,
              javax.xml.ws,
              org.apache.cxf.bus,
              org.apache.cxf.bus.spring,
              org.apache.cxf.bus.resource,
              org.apache.cxf.configuration.spring,
              org.apache.cxf.resource,
              org.apache.cxf.jaxws,
              org.springframework.beans.factory.config,
              *
            </Import-Package>
            ...
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

5.3.5. 添加代码生成插件

Web 服务项目通常需要生成代码。Apache CXF 为 JAX-WS 前端提供了两个 Maven 插件，它可让您将代码生成步骤集成到您的构建中。插件选择取决于您是否使用 Java 优先方法或 WSDL 优先方法开发您的服务，如下所示：

- java -first 方法- 使用 cxf-java2ws-plugin 插件。

- WSDL-first 方法- 使用 cxf-codegen-plugin 插件。

5.3.6. OSGi 配置属性

OSGi 配置管理服务定义了将配置设置传递给 OSGi 捆绑包的机制。您不必将此服务用于配置，但通常是配置捆绑包应用程序的最便捷方式。蓝图支持 OSGi 配置，可让您使用 OSGi 配置管理服务获得的值替换蓝图文件中的变量。

有关如何使用 OSGi 配置属性的详情，请参考 第 5.3.7 节 “配置捆绑包插件” 和 第 9.6 节 “向该功能添加 OSGi 配置”。

5.3.7. 配置捆绑包插件

概述

捆绑包插件需要很少的信息才能正常工作。所有必要属性都使用默认设置来生成有效的 OSGi 捆绑包。

虽然您只能使用默认值创建有效的捆绑包，但您可能需要修改一些值。您可以在插件的 instructions 元素中指定大多数属性。

配置属性

一些常用的配置属性是：

- Bundle-SymbolicName
- bundle-Name
- bundle-Version
- export-Package

- private-Package
- import-Package

设置捆绑包的符号名称

默认情况下，捆绑包插件将 Bundle-SymbolicName 属性的值设置为 *groupId* + "." + *artifactId*，但有以下例外：

- 如果 *groupId* 只有一个部分（没有点），则返回第一个带有类的软件包名称。

  例如，如果组 Id 是 commons-logging:commons-logging，则捆绑包的符号名称为 org.apache.commons.logging。

- 如果 *artifactId* 等于 *groupId* 的最后部分，则使用 *groupId*。

  例如，如果 POM 将组 ID 和工件 ID 指定为 org.apache.maven:maven，则捆绑包的符号名称为 org.apache.maven。

- 如果 *artifactId* 以 *groupId* 的最后部分开头，则会删除该部分。

  例如，如果 POM 将组 ID 和工件 ID 指定为 org.apache.maven:maven-core，则捆绑包的符号名称为 org.apache.maven.core。

要为捆绑包的符号名称指定您自己的值，请在插件的 instructions 元素中添加一个 Bundle-SymbolicName 子级，如 例 5.2 “设置捆绑包的符号名称” 所示。

例 5.2. 设置捆绑包的符号名称

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
     ...
    </instructions>
  </configuration>
</plugin>
```

–

设置捆绑包名称

默认情况下，捆绑包的名称设置为 ${project.name}。

要为捆绑包的名称指定您自己的值，请在插件的 instructions 元素中添加一个 Bundle-Name 子对象，如 例 5.3 “设置捆绑包名称” 所示。

例 5.3. 设置捆绑包名称

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-Name>JoeFred</Bundle-Name>
     ...
    </instructions>
  </configuration>
</plugin>
```

设置捆绑包的版本

默认情况下，捆绑包的版本设置为 ${project.version}。任何短划线(-)替换为点(.)，数字会加到四位。例如，4.2-SNAPSHOT 变为 4.2.0.SNAPSHOT。

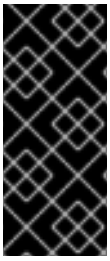
要为捆绑包的版本指定您自己的值，请在插件的 instructions 元素中添加一个 Bundle-Version 子对象，如 例 5.4 “设置捆绑包的版本” 所示。

例 5.4. 设置捆绑包的版本

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-Version>1.0.3.1</Bundle-Version>
     ...
    </instructions>
  </configuration>
</plugin>
```

指定导出的软件包

默认情况下，OSGi 清单的 Export-Package 列表由您的本地 Java 源代码中的所有软件包填充（在 src/main/java下），除了默认软件包.. 以及包含 .impl 或 .internal 的任何软件包。



重要

如果您在插件配置中使用 Private-Package 元素，且您没有指定要导出的软件包列表，则默认行为仅包含捆绑包中 Private-Package 元素中列出的软件包。没有导出软件包。

默认行为可能会导致软件包非常大，并导出应保持私有的软件包。要更改导出的软件包列表，您可以在插件的 instructions 元素中添加 Export-Package 子子。

Export-Package 元素指定要包含在捆绑包中的软件包列表，以及要导出的软件包列表。可以使用 * 通配符符号指定软件包名称。例如，条目 com.fuse.demo 需要包括以 com.fuse.demo 开头的项目的 classpath 中的所有软件包。

您可以使用 ! 指定要排除的软件包作为前缀。例如，条目 !com.fuse.demo.private 排除软件包 com.fuse.demo.private。

在排除软件包时，列表中条目的顺序非常重要。从开始顺序处理列表，并忽略后续字典条目。

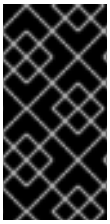
例如，要包含以 com.fuse.demo 开头的所有软件包，软件包 com.fuse.demo.private 除外，请使用以下方法列出软件包：

```
!com.fuse.demo.private,com.fuse.demo.*
```

但是，如果您使用 com.fuse.demofuse,!com.fuse.demo.private 列出软件包，那么 com.fuse.demo.private 将包含在捆绑包中，因为它与第一个模式匹配。

指定私有软件包

如果要在 不 导出捆绑包中指定要包含的软件包列表，您可以在捆绑插件配置中添加 Private-Package 指令。默认情况下，如果您没有指定 Private-Package 指令，则捆绑包中包含本地 Java 源中的所有软件包。



重要

如果软件包与 Private-Package 元素和 Export-Package 元素中的条目匹配，则 Export-Package 元素将具有优先权。软件包被添加到捆绑包中并导出。

Private -Package 元素的工作方式与您要包含在捆绑包中的软件包列表类似。bundle 插件使用列表来查找要包含在捆绑包中的项目的 classpath 中的所有类。这些软件包打包在捆绑包中，但不会导出（除非它们也由 Export-Package 指令选择）。

例 5.5 “在捆绑包中包含私有软件包” 显示在捆绑包中包含私有软件包的配置

例 5.5. 在捆绑包中包含私有软件包

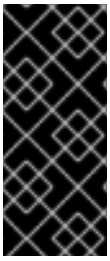
```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Private-Package>org.apache.cxf.wsdlFirst.impl</Private-Package>
     ...
    </instructions>
  </configuration>
</plugin>
```

指定导入的软件包

默认情况下，捆绑包插件使用由捆绑包内容引用的所有软件包列表填充 OSGi 清单的 Import-Package 属性。

虽然默认行为通常足以满足大多数项目，但您可能会找到您想导入没有自动添加到列表中的软件包的实例。默认行为还会导致导入不需要的软件包。

要指定捆绑包要导入的软件包列表，请在插件的 instructions 元素中添加 Import-Package 子级。软件包列表的语法与 Export-Package 元素和 Private-Package 元素的语法相同。



重要

使用 Import-Package 元素时，插件不会自动扫描捆绑包的内容，以确定是否有所需的导入。要确保扫描捆绑包的内容，您必须在软件包列表中放置一个 * 作为最后一个条目。

例 5.6 “指定捆绑包导入的软件包” 显示指定捆绑包导入的软件包的配置

例 5.6. 指定捆绑包导入的软件包

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Import-Package>javax.jws, javax.wsdl, org.apache.cxf.bus, org.apache.cxf.bus.spring,
org.apache.cxf.bus.resource, org.apache.cxf.configuration.spring, org.apache.cxf.resource,
org.springframework.beans.factory.config, * </Import-Package>
      ...
   </instructions>
  </configuration>
</plugin>
```

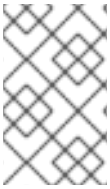
更多信息

有关配置捆绑包插件的更多信息，请参阅：

- **olink:OsgiDependencies/OsgiDependencies**
- **Apache Felix 文档**
- **LL Kriens 的 AQute Software Consultancy 网站**

5.3.8. OSGI configAdmin 文件命名规则

PID 字符串(symbolic-name 语法)允许 OSGI 规格中的连字符。但是，连字符由 Apache Felix.fileinstall 和 config:edit shell 命令解释，以区分"受管服务"和"受管服务工厂"。因此，建议您不要在 PID 字符串中使用连字符。



注意

配置文件名称与 PID 和工厂 PID 相关。

第 6 章 热部署与手动部署

摘要

Fuse 为部署文件提供了两种不同的方法：热部署或手动部署。如果您需要部署一组相关捆绑包，建议您将它们作为 *功能* 部署，而不是完全（请参阅 第 9 章 *部署功能*）。

6.1. 热部署

6.1.1. 热部署目录

Fuse 监控 FUSE_HOME/deploy 目录中的文件，并在此目录中热部署所有内容。每次将文件复制到此目录时，该文件都会在运行时安装并启动。然后，您可以更新或删除 FUSE_HOME/deploy 目录中的文件，并且更改会自动处理。

例如，如果您只构建了捆绑包，*ProjectDir*/target/foo-1.0-SNAPSHOT.jar，您可以通过将其复制到 FUSE_HOME/deploy 目录来进行部署（假设您正在处理 UNIX 平台）：

```
% cp ProjectDir/target/foo-1.0-SNAPSHOT.jar FUSE_HOME/deploy
```

6.2. 热取消部署捆绑包

要从热部署目录取消部署捆绑包，只需在 Apache Karaf 容器运行时 从 FUSE_HOME/deploy 目录中删除捆绑包文件。

重要

在容器关闭时，热取消部署机制 不起作用。如果您关闭 Karaf 容器，请从 FUSE_HOME/deploy 目录中删除捆绑包文件，然后重启 Karaf 容器，则在重启容器后，不会 取消部署捆绑包。

您还可以使用 bundle:uninstall console 命令取消部署捆绑包。

6.3. 手动部署

6.3.1. 概述

您可以通过在 Fuse 控制台中发出命令来手动部署和取消部署捆绑包。

6.3.2. 安装捆绑包

使用 bundle:install 命令，在 OSGi 容器中安装一个或多个捆绑包。这个命令使用以下语法：

```
bundle:install [-s] [--start] [--help] UrlList
```

其中 *UrlList* 是空格分开的 URL 列表，用于指定要部署的每个捆绑包的位置。支持以下命令参数：

-s

安装后启动捆绑包。

--start

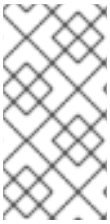
与 **-s** 相同。

--help

显示并说明命令语法。

例如，要安装并启动捆绑包，*ProjectDir*/target/foo-1.0-SNAPSHOT.jar，请在 Karaf 控制台提示符处输入以下命令：

```
bundle:install -s file:ProjectDir/target/foo-1.0-SNAPSHOT.jar
```



注意

在 Windows 平台上，必须小心谨慎，为此命令 中的文件 URL 使用正确的语法。详情请查看 第 15.1 节 “文件 URL 处理程序”。

6.3.3. 卸载捆绑包

要卸载捆绑包，您必须首先使用 bundle:list 命令获取其捆绑包 ID。然后，您可以使用 bundle:uninstall 命令卸载捆绑包（使用捆绑包 ID 作为其参数）。

例如，如果您已安装名为 A Camel OSGi 服务单元 的捆绑包，在控制台提示符下输入 bundle:list 可

能会生成类似如下的输出：

```
...
[ 181] [Resolved   ] [            ] [       ] [   60] A Camel OSGi Service Unit (1.0.0.SNAPSHOT)
```

现在，您可以输入以下 console 命令，使用 ID 181 卸载捆绑包：

```
bundle:uninstall 181
```

6.3.4. 用于查找捆绑包的 URL 方案

当指定 bundle:install 命令的位置 URL 时，您可以使用 Fuse 支持的任何 URL 方案，其中包括以下方案类型：

- 第 15.1 节 “文件 URL 处理程序”.
- 第 15.2 节 “HTTP URL 处理程序”.
- 第 15.3 节 “mvn URL Handler”.

6.4. 使用 BUNDLE:WATCH 自动重新部署捆绑包

在开发环境中，开发人员不断更改并重新构建捆绑包，通常需要多次重新安装捆绑包。使用 bundle:watch 命令，您可以指示 Karaf 监控本地 Maven 存储库，并在其更改本地 Maven 存储库时自动重新安装特定捆绑包。

例如，如果一个特定的捆绑包 ID 带有捆绑包 ID，751 - 您可以通过输入以下命令启用自动重新部署：

```
bundle:watch 751
```

现在，每当您重新构建并安装 Maven 工件并将其安装到本地 Maven 存储库（例如，在 Maven 项目中执行 mvn install ）时，Samium 容器会自动重新安装更改的 Maven 工件。如需了解更多详细信息，请参阅 Apache Karaf 控制台参考。

重要

使用 bundle:watch 命令仅用于开发环境。不建议在生产环境中使用。

第 7 章 生命周期管理

7.1. 捆绑包生命周期状态

OSGi 环境中的应用程序受到其捆绑包的生命周期约束。捆绑包有六个生命周期状态：

1. Installed - 所有捆绑包都以安装的状态启动。安装状态中的捆绑包会等待所有依赖项解决，一旦被解决，捆绑包将移到解析的状态。

2. resolved - 在满足以下条件时，捆绑包将移到已解析的状态：

   - 运行时环境满足或超过捆绑包指定的环境。
   - 捆绑包导入的所有软件包都由捆绑包公开，这些捆绑包处于解析状态，或者可以移到与当前捆绑包相同的解析状态。
   - 所有所需捆绑包都处于解析状态，也可以与当前捆绑包同时解析。

   重要

   所有应用程序的捆绑包都必须处于解析的状态，然后才能启动应用程序。

   如果满足上述任何条件，则捆绑包将移回到安装状态。例如，当容器中删除包含导入软件包的捆绑包时，可能会发生这种情况。

3. starting - 启动 状态是解析的状态和活跃状态之间的传输状态。启动捆绑包时，容器必须为捆绑包创建资源。容器也会在提供捆绑包时调用捆绑包的 start （） 方法。

4. Active - 处于 active 状态的捆绑包可用于工作。捆绑包在活跃状态下执行的操作取决于捆绑包的内容。例如，包含 JAX-WS 服务提供商的捆绑包表示该服务可用于接受请求。

5. stop - 停止 状态是 active 状态和解析状态之间的传输状态。当捆绑包停止时，容器必须清理

捆绑包的资源。容器也会在提供捆绑包时调用捆绑包的 stop （） 方法。

6. uninstall - 卸载捆绑包时，它会从解析的状态移到卸载的状态。处于此状态的捆绑包无法转换为解析的状态或任何其他状态。它必须明确重新安装。

应用程序开发人员最重要的生命周期状态为 start 状态和 stop 状态。应用公开的端点会在启动状态期间发布。发布的端点在停止状态期间停止。

7.2. 安装并解决捆绑包

当您使用 bundle:install 命令（不带 -s 标志）安装捆绑包时，内核会安装指定的捆绑包并尝试将其置于解析状态。如果因为某种原因，捆绑包的解析失败（例如，如果其其中一个依赖项不满意），则内核会将捆绑包处于安装的状态。

稍后（例如，在安装缺少的依赖项后），您可以通过调用 bundle:resolve 命令尝试将捆绑包移到解析的状态，如下所示：

```
bundle:resolve 181
```

其中，参数（本例中为181） 是您要解析的捆绑包的 ID。

7.3. 启动和停止捆绑包

您可以使用 bundle:start 命令启动一个或多个捆绑包（来自安装或解析状态）。例如，要使用 ID、181、185 和 186 启动捆绑包，请输入以下命令：

```
bundle:start 181 185 186
```

您可以使用 bundle:stop 命令停止一个或多个捆绑包。例如，要停止 ID 为 181、1、185 和 186 的捆绑包，请输入以下命令：

```
bundle:stop 181 185 186
```

您可以使用 bundle:restart 重启一个或多个捆绑包（即，从启动状态移到已解析状态），然后再次返回到启动状态。例如，要重启 ID 为 181、1、185 和 186 的捆绑包，请输入以下命令：

```
bundle:restart 181 185 186
```

7.4. 捆绑包开始级别

启动级别 与每个捆绑包相关联。起始级别是一个正整数值，用于控制捆绑包被激活/启动的顺序。在具有高级别高级别的捆绑包之前，启动具有低启动级别的捆绑包。因此，首先启动带有起始级别 1 的捆绑包，并属于内核的捆绑包会较低启动级别，因为它们提供了运行大多数其他捆绑包的先决条件。

通常，用户捆绑包的起始级别为 60 或更高版本。

7.5. 指定捆绑包的启动级别

使用 bundle:start-level 命令设置特定捆绑包的启动级别。例如，要将 ID 为 181 的捆绑包配置为具有 70 的起始级别，请输入以下 console 命令：

```
bundle:start-level 181 70
```

7.6. 系统启动级别

OSGi 容器本身具有与之关联的起始级别，*此系统启动级别* 决定了哪些捆绑包可以处于活跃状态，而不能：只有启动级别 低于或等于 系统启动级别的捆绑包可以处于活跃状态。

要发现当前的系统启动级别，请在控制台中输入 system:start-level，如下所示：

```
karaf@root()> system:start-level
Level 100
```

如果要更改系统启动级别，请提供新的启动级别作为 system:start-level 命令的参数，如下所示：

```
system:start-level 200
```

第 8 章 依赖项故障排除

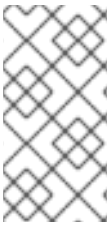
8.1. 缺少依赖项

将 OSGi 捆绑包部署到红帽 Fuse 容器时可能会出现的最常见问题是缺少一个或多个依赖项。当您尝试在 OSGi 容器中解析捆绑包时，这个问题显示自己，通常是启动捆绑包的副作用。捆绑包无法解析（或启动）和 ClassNotFound 错误（若要查看日志，请使用 log:display console 命令，或查看 FUSE_HOME/data/log 目录中的日志文件）。

缺少依赖项有两个基本原因：容器中没有安装所需的功能或捆绑包；或者您的捆绑包的 Import-Package 标头不完整。

8.2. 未安装所需的功能或捆绑包

在尝试解析捆绑包前，您的捆绑包所需的所有功能和捆绑包都必须安装在 OSGi 容器中。特别是，因为 Apache Camel 有一个模块化架构，其中每个组件都作为单独的功能安装，因此易于忘记安装其中一个所需组件。



注意

考虑将捆绑包打包为功能。通过使用功能，您可以将捆绑包与其所有依赖项打包在一起，从而确保它们同时安装。详情请查看 第 9 章 *部署功能*。

8.3. IMPORT-PACKAGE 标头不完整

如果已安装所有必需的功能和捆绑包，并且您仍然收到 ClassNotFound 错误，这意味着捆绑包的 MANIFEST.MF 文件中的 Import-Package 标头不完整。在生成捆绑包的 Import-Package 标头时，maven-bundle-plugin （请参阅 第 5.2 节 “修改现有 Maven 项目”）是一个很好的帮助，但您应该注意以下几点：

- 请确定您在 Maven 捆绑插件配置的 Import-Package 元素中包含通配符。通配符指示插件扫描 Java 源代码，并自动生成软件包依赖项列表。
- Maven 捆绑包插件 无法 找出动态依赖项。例如，如果您的 Java 代码明确调用类加载程序来动态加载类加载程序，则捆绑包插件不会考虑此帐户，且所需的 Java 软件包不会在生成的 Import-Package 标头中列出。
- 如果您定义了蓝图 XML 文件（例如，在 OSGI-INF/blueprint 目录中），所有来自

Blueprint XML 文件的依赖项都会 在运行时自动解决。

8.4. 如何跟踪缺少的依赖项

要跟踪缺少的依赖项，请执行以下步骤：

1. 使用 bundle:diag console 命令。这将提供有关为什么您的捆绑包不活跃的信息。有关使用信息，请参阅 Apache Karaf 控制台参考。

2. 执行快速检查，以确保在 OSGi 容器中实际安装了所有必需的捆绑包和功能。您可以使用 bundle:list 检查已安装哪些捆绑包，并且 features:list 来检查安装了哪些功能。

3. 使用 bundle:install console 命令安装（但不启动）您的捆绑包。例如：

```
karaf@root()> bundle:install MyBundleURL
```

4. 使用 bundle:dynamic-import console 命令在您刚才安装的捆绑包中启用动态导入。例如，如果捆绑包的捆绑包 ID 是 218，您可以通过输入以下命令在这个捆绑包上启用动态导入：

```
karaf@root()> bundle:dynamic-import 218
```

此设置允许使用容器中已安装的 任何 捆绑包解析依赖项，从而有效地绕过常见的依赖关系解析机制（基于 Import-Package 标头）。对于正常部署，这 不会被 重新处理，因为它绕过了版本检查：您可以轻松地获取错误的软件包版本，从而导致应用程序出现故障。

5. 现在，您应该可以解析您的捆绑包。例如，如果您的捆绑包 ID 是 218，请输入 followng console 命令：

```
karaf@root()> bundle:resolve 218
```

6. 假设您的捆绑包现已解决（使用 bundle:list检查捆绑包状态），您可以使用 package:imports 命令获取所有软件包的完整列表。例如，如果您的捆绑包 ID 是 218，请输入以下命令：

```
karaf@root()> package:imports -b 218
```

您应该在控制台窗口中看到依赖软件包列表：

```
Package                          | Version       | Optional | ID | Bundle Name
---------------------------------+---------------+----------+----+-----------------------------------------
org.apache.jasper.servlet        | [2.2.0,3.0.0) | resolved | 217 | org.ops4j.pax.web.pax-web-runtime
org.jasypt.encryption.pbe        |               | resolved | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.jsp            | [7.0.0,)      | resolved | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.model | [7.0.0,)   |          | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.util | [7.0.0,)    |          | 217 | org.ops4j.pax.web.pax-web-runtime
...
```

7. 解包您的捆绑包 JAR 文件，并查看 META-INF/MANIFEST.MF 文件中的 Import-Package 标头下列出的软件包。将此列表与上一步中找到的软件包列表进行比较。现在，编译清单 Import-Package 标头中缺少的软件包列表，并将这些软件包名称添加到项目的 POM 文件中 Maven 捆绑插件配置的 Import-Package 元素中。

8. 要取消动态导入选项，您必须从 OSGi 容器卸载旧的捆绑包。例如，如果您的捆绑包 ID 是 218，请输入以下命令：

```
karaf@root()> bundle:uninstall 218
```

9. 现在，您可以使用导入的软件包更新列表重建捆绑包，并在 OSGi 容器中进行测试。

addurl :experimental: :toc: :toclevels: 4 :numbered:

第 9 章 部署功能

摘要

由于应用程序和其他工具通常由多个 OSGi 捆绑包组成，因此将独立或相关捆绑包聚合到更大的部署单元中通常很方便。因此，Red Hat Fuse 提供了一个可扩展的部署单元，*该功能* 可让您在单个步骤中部署多个捆绑包（以及可选的依赖其他功能）。

9.1. 创建功能

9.1.1. 概述

本质上，通过向特殊的 XML 文件（称为 功能 存储库）添加新的功能元素来创建功能。要创建功能，请执行以下步骤：

1. **第 9.2 节 “创建自定义功能存储库”.**
2. **第 9.3 节 “在自定义功能存储库中添加功能”.**
3. **第 9.4 节 “将本地存储库 URL 添加到 features 服务”.**
4. **第 9.5 节 “在该功能中添加依赖功能”.**
5. **第 9.6 节 “向该功能添加 OSGi 配置”.**

9.2. 创建自定义功能存储库

如果您还没有定义自定义功能存储库，您可以按如下所示创建一个存储库。在文件系统中为功能存储库选择一个方便的位置，例如 C:\Projects\features.xml，并使用您喜欢的文本编辑器将其添加到其中：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomRepository">
</features>
```

通过设置 name 属性，您必须为存储库指定名称 *CustomRepository*。

注意

与 Maven 存储库或 OBR 不同，功能存储库 不为 捆绑包提供存储位置。功能存储库只是存储捆绑包的引用聚合。捆绑包本身存储在其他位置（例如，文件系统或 Maven 存储库中）。

9.3. 在自定义功能存储库中添加功能

要在自定义功能存储库中添加功能，请插入一个新的 feature 元素作为根 features 元素的子部分。您必须命名该功能，并通过插入 bundle 子元素来列出属于该功能的任意数量的捆绑包。例如，要添加名为 example-camel-bundle 的功能，其中包含单个捆绑包 C:\Projects\camel-bundle\target\camel-bundle-1.0-SNAPSHOT.jar，请添加 feature 元素：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
  </feature>
</features>
```

bundle 元素的内容可以是任何有效的 URL，提供捆绑包的位置（请参阅 第 15 章 *URL 处理程序*）。您可以选择在 feature 元素上指定一个 version 属性，为该功能分配非零版本（您可以将版本指定为 features:install 命令的可选参数）。

要检查功能服务是否成功解析新的功能条目，请输入以下命令：

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:list
...
[uninstalled] [0.0.0              ] example-camel-bundle              MyFeaturesRepo
...
```

features:list 命令通常会生成长的功能列表，但您应能够通过滚动浏览列表来查找新功能（本例中为 example-camel-bundle）的条目。features:refreshurl 命令强制内核重新读取所有功能存储库：如果您没有发出此命令，内核将不知道您对任何存储库所做的任何最新更改（特别是，新功能不会出现在列表中）。

要避免滚动到长的功能列表，您可以 grep example-camel-bundle 功能，如下所示：

```
JBossFuse:karaf@root> features:list | grep example-camel-bundle
[uninstalled] [0.0.0              ] example-camel-bundle              MyFeaturesRepo
```

其中 grep 命令（标准 UNIX 模式匹配实用程序）内置在 shell 中，因此此命令也适用于 Windows 平台。

9.4. 将本地存储库 URL 添加到 FEATURES 服务

要使新功能存储库可供 Apache Karaf 使用，您必须使用 features:addurl console 命令添加功能存储库。例如，要使存储库的内容 C:\Projects\features.xml 可用于内核，您需要输入以下 console 命令：

```
features:addurl file:C:/Projects/features.xml
```

可以使用任何支持的 URL 格式来指定 features:addurl 的参数（请参阅 第 15 章 *URL 处理程序*）。

您可以通过输入 features:listUrl console 命令来检查存储库的 URL 是否已正确注册，以获取所有注册的功能存储库 URL 的完整列表，如下所示：

```
JBossFuse:karaf@root> features:listUrl
file:C:/Projects/features.xml
mvn:org.apache.ode/ode-jbi-karaf/1.3.3-fuse-01-00/xml/features
mvn:org.apache.felix.karaf/apache-felix-karaf/1.2.0-fuse-01-00/xml/features
```

9.5. 在该功能中添加依赖功能

如果您的功能依赖于其他功能，可以通过添加 feature 元素作为原始 feature 元素的子项来指定这些依赖项。每个子 功能 元素都包含当前功能依赖的功能名称。当您使用依赖功能部署功能时，依赖项机制会检查容器中是否安装了依赖功能。如果没有，依赖项机制会自动安装缺少的依赖项（以及任何递归依赖项）。

例如，对于自定义 Apache Camel 功能，example-camel-bundle，您可以明确指定它所依赖的标准 Apache Camel 功能。这样做的好处是，应用程序现在可以成功部署并运行，即使 OSGi 容器没有预先部署所需的功能。例如，您可以使用 Apache Camel 依赖项定义 example-camel-bundle 功能，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
    <feature version="7.13.0.fuse-7_13_0-00012-redhat-00001">camel-core</feature>
    <feature version="7.13.0.fuse-7_13_0-00012-redhat-00001">camel-spring-osgi</feature>
  </feature>
</features>
```

指定 version 属性是可选的。存在时，它可让您选择指定的功能版本。

9.6. 向该功能添加 OSGI 配置

如果您的应用程序使用 *OSGi 配置管理服务*，您可以使用功能定义的 config 子元素指定该服务的配置设置。例如，要指定 prefix 属性具有值 MyTransform，请将以下 config 子元素添加到功能配置中：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <config name="org.fusesource.fuseesb.example">
      prefix=MyTransform
    </config>
  </feature>
</features>
```

其中 config 元素的 name 属性指定属性设置 *的持久性 ID*（持久性 ID 充当属性名称的名称范围）。config 元素的内容与 Java 属性文件 相同。

config 元素中的设置可以被 *InstallDir*/etc 目录中的 Java 属性文件中的设置覆盖，该文件以持久 ID 命名，如下所示：

```
InstallDir/etc/org.fusesource.fuseesb.example.cfg
```

作为在实践中如何使用上述配置属性的示例，请考虑以下蓝图 XML 文件来访问 OSGi 配置属性：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0">

    <!-- osgi blueprint property placeholder -->
    <cm:property-placeholder id="placeholder"
                             persistent-id="org.fusesource.fuseesb.example">
        <cm:default-properties>
            <cm:property name="prefix" value="DefaultValue"/>
        </cm:default-properties>
    </cm:property-placeholder>

    <bean id="myTransform" class="org.fusesource.fuseesb.example.MyTransform">
      <property name="prefix" value="${prefix}"/>
    </bean>

</blueprint>
```

当此 Blueprint XML 文件部署到 example-camel-bundle 捆绑包中时，属性引用 ${prefix} 替换为值 MyTransform，它由 feature 存储库中的 config 元素指定。

9.7. 自动部署 OSGI 配置

通过向功能添加 configfile 元素，您可以确保在安装该功能时同时将 OSGi 配置文件添加到 *InstallDir*/etc 目录中。这意味着您可以方便地安装功能及其关联的配置。

例如，如果 org.fusesource.fuseesb.example.cfg 配置文件归档在 mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg 的 Maven 存储库中，您可以通过向该功能中添加以下元素来部署配置文件：

```
<configfile finalname="etc/org.fusesource.fuseesb.example.cfg">
  mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg
</configfile>
```

第 10 章 部署功能

10.1. 概述

您可以使用以下方法之一部署功能：

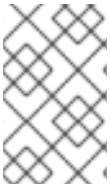
- 使用 features:install 在控制台中安装。
- 使用热部署。
- 修改启动配置（仅第一次引导！）。

10.2. 在控制台中安装

创建功能后（通过在功能存储库中添加条目并注册功能存储库）后，使用 features:install console 命令部署该功能相对容易。例如，要部署 example-camel-bundle 功能，请输入以下 console 命令对：

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:install example-camel-bundle
```

建议您在调用 features:install 之前调用 features:refreshurl 命令，以防对内核尚未获取的功能进行了任何最新的更改。features:install 命令将功能名称用作其参数（以及可选，功能版本作为其第二个参数）。



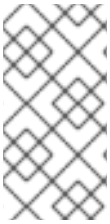
注意

功能使用扁平命名空间。因此，在命名功能时，请小心以避免与现有功能冲突。

10.3. 在控制台中卸载

要卸载某个功能，请按如下所示调用 features:uninstall 命令：

```
JBossFuse:karaf@root> features:uninstall example-camel-bundle
```



注意

卸载后，这个功能仍会在调用 features:list 时看到，但其状态现在将被标记为 [uninstalled]。

10.4. 热部署

只需将功能存储库文件复制到 *InstallDir*/deploy 目录，即可热部署功能存储库中 的所有功能。

由于您不太可能一次热部署整个功能存储库，因此通常更方便地定义缩减的功能存储库或功能 *描述符*（仅引用您想要部署的功能）。功能描述符与功能存储库的语法完全相同，但以不同的样式编写。区别在于，功能描述符仅包含从功能存储库中对现有功能的引用。

例如，您可以定义一个功能描述符来加载 example-camel-bundle 功能，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomDescriptor">
  <repository>RepositoryURL</repository>
  <feature name="hot-example-camel-bundle">
    <feature>example-camel-bundle</feature>
  </feature>
</features>
```

repository 元素指定自定义功能存储库 *RepositoryURL* 的位置（您可以使用 *第 15 章 URL 处理程序*中描述的任何 URL 格式）。feature hot-example-camel-bundle 只是对现有功能 example-camel-bundle 的引用。

10.5. 热取消部署功能文件

要从 hot deploy 目录取消部署功能文件，只需在 Apache Karaf 容器运行时 从 *InstallDir*/deploy 目录中删除 features 文件。

重要

在容器关闭时，热取消部署机制 不起作用。如果您关闭 Karaf 容器，请从 deploy/ 中删除功能文件，然后重新启动 Karaf 容器，则在重新启动容器后，这些功能 将不会被 取消部署（但您可以使用 features:uninstall console 命令手动取消部署功能）。

10.6. 在引导配置中添加功能

如果要在多个主机上调配 Apache Karaf 副本以进行部署，您可能有兴趣添加功能到引导配置，这决定了 Apache Karaf 首次启动时安装的功能集合。

安装目录中的配置文件 /etc/org.apache.karaf.features.cfg 包含以下设置：

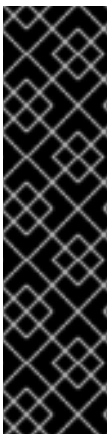
```
...
#
# Comma separated list of features repositories to register by default
#
featuresRepositories = \
    mvn:org.apache-extras.camel-extra.karaf/camel-extra/2.21.0.fuse-000032-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/spring-legacy/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.activemq/artemis-features/2.4.0.amq-710008-redhat-1/xml/features, \
    mvn:org.jboss.fuse.modules.patch/patch-features/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/framework/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.jboss.fuse/fuse-karaf-framework/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/standard/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.karaf.features/enterprise/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.camel.karaf/apache-camel/2.21.0.fuse-000055-redhat-2/xml/features, \
    mvn:org.apache.cxf.karaf/apache-cxf/3.1.11.fuse-000199-redhat-1/xml/features, \
    mvn:io.hawt/hawtio-karaf/2.0.0.fuse-000145-redhat-1/xml/features

#
# Comma separated list of features to install at startup
#
featuresBoot = \
    instance/4.2.0.fuse-000191-redhat-1, \
    cxf-commands/3.1.11.fuse-000199-redhat-1, \
    log/4.2.0.fuse-000191-redhat-1, \
    pax-cdi-weld/1.0.0, \
    camel-jms/2.21.0.fuse-000055-redhat-2, \
    ssh/4.2.0.fuse-000191-redhat-1, \
    camel-cxf/2.21.0.fuse-000055-redhat-2, \
    aries-blueprint/4.2.0.fuse-000191-redhat-1, \
    cxf/3.1.11.fuse-000199-redhat-1, \
    cxf-http-undertow/3.1.11.fuse-000199-redhat-1, \
    pax-jdbc-pool-narayana/1.2.0, \
    patch/7.0.0.fuse-000163-redhat-2, \
    cxf-rs-description-swagger2/3.1.11.fuse-000199-redhat-1, \
    feature/4.2.0.fuse-000191-redhat-1, \
    camel/2.21.0.fuse-000055-redhat-2, \
    jaas/4.2.0.fuse-000191-redhat-1, \
    camel-jaxb/2.21.0.fuse-000055-redhat-2, \
    camel-paxlogging/2.21.0.fuse-000055-redhat-2, \
    deployer/4.2.0.fuse-000191-redhat-1, \
    diagnostic/4.2.0.fuse-000191-redhat-1, \
    patch-management/7.0.0.fuse-000163-redhat-2, \
    bundle/4.2.0.fuse-000191-redhat-1, \
    kar/4.2.0.fuse-000191-redhat-1, \
    camel-csv/2.21.0.fuse-000055-redhat-2, \
    package/4.2.0.fuse-000191-redhat-1, \
    scr/4.2.0.fuse-000191-redhat-1, \
```

```
maven/4.2.0.fuse-000191-redhat-1, \
war/4.2.0.fuse-000191-redhat-1, \
camel-mail/2.21.0.fuse-000055-redhat-2, \
fuse-credential-store/7.0.0.fuse-000163-redhat-2, \
framework/4.2.0.fuse-000191-redhat-1, \
system/4.2.0.fuse-000191-redhat-1, \
pax-http-undertow/6.1.2, \
camel-jdbc/2.21.0.fuse-000055-redhat-2, \
shell/4.2.0.fuse-000191-redhat-1, \
management/4.2.0.fuse-000191-redhat-1, \
service/4.2.0.fuse-000191-redhat-1, \
camel-undertow/2.21.0.fuse-000055-redhat-2, \
camel-blueprint/2.21.0.fuse-000055-redhat-2, \
camel-spring/2.21.0.fuse-000055-redhat-2, \
hawtio/2.0.0.fuse-000145-redhat-1, \
camel-ftp/2.21.0.fuse-000055-redhat-2, \
wrap/2.5.4, \
config/4.2.0.fuse-000191-redhat-1, \
transaction-manager-narayana/5.7.2.Final
```

这个配置文件有两个属性：

- **featuresRepositories-comma 分隔要在启动时加载的功能存储库列表。**
- **featuresBoot-comma 分隔要在启动时安装的功能列表。**

您可以修改配置来自定义在 Fuse 启动时安装的功能。如果您计划使用预安装的功能分发 Fuse，您也可以修改此配置文件。



重要

这种添加功能的方法仅在特定 Apache Karaf 实例引导时有效。之后对 **featuresRepositories** 设置所做的所有更改，并且 **featuresBoot** 设置将被忽略，即使您重启了容器。

您可以强制容器恢复到其初始状态，但通过删除 **InstallDir/data/cache** 的所有内容（同时丢失所有容器的自定义设置）。

第 11 章 部署 PLAIN JAR

摘要

将应用部署到 Apache Karaf 的替代方法是使用普通 JAR 文件。这些通常是 不包含部署元数据 的库。普通 JAR 不是 WAR，也不是 OSGi 捆绑包。

如果作为捆绑包的依赖项进行普通 JAR，您必须将捆绑包标头添加到 JAR。如果 JAR 公开了公共 API，通常的最佳解决方案是将现有的 JAR 转换为捆绑包，使 JAR 能够与其他捆绑包共享。使用本章中的说明，使用开源 Bnd 工具自动执行转换过程。

有关 Bnd 工具的更多信息，请参阅 Bnd 工具网站。

11.1. 使用嵌套方案转换 JAR

概述

您可以选择使用 wrap: 协议将 JAR 转换为捆绑包，该协议可用于任何现有 URL 格式。嵌套： 协议基于 Bnd 工具。

语法

嵌套： 协议有以下基本语法：

```
wrap:LocationURL
```

The wrap: 协议可以加上任何查找 JAR 的 URL 前缀。URL 的定位部分 *LocationURL* 用于获取普通 JAR 和嵌套的 URL 处理程序 ： 协议，然后自动将 JAR 转换为捆绑包。

注意

嵌套： 协议还支持更详细的语法，它允许您通过指定第二属性文件或在 URL 中指定单独的 Bnd 属性来自定义转换。但是，一般情况下，换行： 协议仅与默认设置一起使用。

默认属性

wrap: 协议基于 **Bnd** 工具，因此它使用完全相同的默认属性生成捆绑包，因为 **Bnd** 的作用。

嵌套并安装

以下示例演示了如何使用单个控制台命令从远程 **Maven** 存储库下载普通 **commons-logging JAR**，动态将其转换为 **OSGi** 捆绑包，然后安装它并在 **OSGi** 容器中启动：

```
karaf@root> bundle:install -s wrap:mvn:commons-logging/commons-logging/1.1.1
```

参考

结束： 协议由 **Pax** 项目提供，该项目 是各种开源 **OSGi** 工具的 **umbrella** 项目。有关 嵌套的完整文档： 协议，请参阅 **Wrap Protocol** 参考页面。

第 12 章 OSGI 服务

摘要

OSGi 核心框架定义了 *OSGi 服务层*，它通过将 Java 对象注册为 *OSGi 服务注册表中的* 服务来交互，为捆绑提供简单机制。OSGi 服务模型的一大优势 是任何 Java 对象都可以作为服务提供：没有必须应用到服务类的特定约束、继承规则或注解。本章论述了如何使用 OSGi *Blueprint 容器部署* OSGi 服务。

12.1. BLUEPRINT 容器

摘要

Blueprint 容器是一种依赖项注入框架，简化了与 OSGi 容器的交互。Blueprint 容器支持使用 OSGi 服务 registry 进行基于配置的方法，例如，提供用于导入和导出 OSGi 服务的标准 XML 元素。

12.1.1. 蓝图配置

JAR 文件中蓝图文件的位置

与捆绑包 JAR 文件的根目录相比，Blueprint 配置文件的标准位置是以下目录：

```
OSGI-INF/blueprint
```

在此目录中带有后缀 .xml 的任何文件都解释为 Blueprint 配置文件；换句话说，任何与模式匹配的文件，OSGI -INF/blueprintAttr.xml。

Maven 项目中的蓝图文件的位置

在 Maven 项目(*ProjectDir*)的上下文中，Blueprint 配置文件的标准位置是以下目录：

```
ProjectDir/src/main/resources/OSGI-INF/blueprint
```

蓝图命名空间和 root 元素

蓝图配置元素与以下 XML 命名空间关联：

```
http://www.osgi.org/xmlns/blueprint/v1.0.0
```

Blueprint 配置的根元素是 蓝图 XML 配置文件，因此蓝图 XML 配置文件通常具有以下概述形式：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  ...
</blueprint>
```

注意

在蓝图 root 元素中，不需要使用 xsi:schemaLocation 属性指定 Blueprint 模式的位置，因为 schema 位置已经对 Blueprint 框架知道。

蓝图清单配置

Blueprint 配置的一些方面由 JAR 清单文件中的标头控制，META-INF/MANIFEST.MF，如下所示：

- 自定义蓝图文件位置.
- 必需的依赖项.

自定义蓝图文件位置

如果您需要将蓝图配置文件放在非标准位置（即 OSGI-INF/blueprintAttr.xml以外的其他位置），您可以在清单文件的 Bundle-Blueprint 标头中指定以逗号分隔的替代位置列表，例如：

```
Bundle-Blueprint: lib/account.xml, security.bp, cnf/*.xml
```

必需的依赖项

默认情况下，对 OSGi 服务的依赖项是强制的（虽然可以通过在 reference 项或 reference -list 元素上将 availability 属性设置为 optional 来更改此设置）。要声明依赖项是强制的，意味着捆绑包在没有依赖项的情况下无法正常工作，并且依赖项必须始终可用。

通常，在蓝图容器初始化时，它会通过 *宽限期*，这时尝试解析所有强制依赖项。如果此时无法解决强制依赖项（默认超时为 5 分钟），则容器初始化将中止，且捆绑包没有启动。以下设置可以附加到 Bundle-SymbolicName 清单标头中，以配置宽限期：

blueprint.graceperiod

如果为 true （默认），则宽限期是否已启用，并且 Blueprint 容器会在初始化过程中等待强制依赖项在初始化过程中解决；如果为 false，则跳过宽限期，容器不会检查强制依赖项是否已解决。

blueprint.timeout

以毫秒为单位指定宽限期超时。默认值为 300000 (5 分钟)。

例如，要启用 10 秒的宽限期，您可以在清单文件中定义以下 Bundle-SymbolicName 标头：

```
Bundle-SymbolicName: org.fusesource.example.osgi-client;
 blueprint.graceperiod:=true;
 blueprint.timeout:= 10000
```

Bundle-SymbolicName 标头的值是分号分隔的列表，其中第一个项是实际捆绑包符号名称，第二个项是 blueprint.graceperiod:=true，启用宽限期和第三个项目 blueprint.timeout:= 10000，指定 10 秒超时。

12.1.2. 定义服务 Bean

概述

Blueprint 容器允许您使用 bean 元素实例化 Java 类。您可以以这种方式创建所有主应用程序对象。特别是，您可以使用 bean 元素来创建代表 OSGi 服务实例的 Java 对象。

蓝图 bean 元素

Blueprint bean 元素在 Blueprint schema 命名空间 http://www.osgi.org/xmlns/blueprint/v1.0.0 中定义。

Bean 示例

以下示例演示了如何使用 Blueprint 的 bean 元素创建几个不同类型的 bean ：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="label" class="java.lang.String">
    <argument value="LABEL_VALUE"/>
  </bean>

  <bean id="myList" class="java.util.ArrayList">
    <argument type="int" value="10"/>
  </bean>

  <bean id="account" class="org.fusesource.example.Account">
    <property name="accountName" value="john.doe"/>
    <property name="balance" value="10000"/>
  </bean>

</blueprint>
```

其中，最后一个 bean 示例引用的 Account 类可以定义如下：

```
package org.fusesource.example;

public class Account
{
    private String accountName;
    private int balance;

    public Account () { }

    public void setAccountName(String name) {
        this.accountName = name;
    }

    public void setBalance(int bal) {
        this.balance = bal;
    }
    ...
}
```

参考

有关定义 Blueprint Bean 的详情，请查看以下引用：

- Spring Dynamic Modules Reference Guide v2.0, Blueprint 章节.
- 第 121 节 蓝图容器规范，来自 OSGi Compendium 服务 R4.2 规范。

12.1.3. 使用属性配置蓝图

概述

这部分描述了如何使用在 Camel 上下文之外的文件中保留的属性配置 Blueprint。

配置蓝图 Bean

蓝图 Bean 可使用可以从外部文件属性从属的变量进行配置。您需要声明 ext 命名空间，并在 Blueprint xml 中添加 属性占位符 bean。使用 Property-Placeholder bean 将属性文件的位置声明给 Blueprint。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.2.0">
```

```
    <ext:property-placeholder>
      <ext:location>file:etc/ldap.properties</ext:location>
    </ext:property-placeholder>
    ...
    <bean ...>
       <property name="myProperty" value="${myProperty}" />
    </bean>
</blueprint>
```

属性拥有者配置选项 的规格可在 http://aries.apache.org/schemas/blueprint-ext/blueprint-ext.xsd 中找到。

12.2. 导出服务

概述

这部分描述了如何将 Java 对象导出到 OSGi 服务注册表，从而使它可作为 service 到 OSGi 容器中其他捆绑包的服务。

使用单个接口导出

要在单一接口名称下将服务导出到 OSGi 服务注册表，请定义一个 service 元素来引用相关服务 bean，使用 ref 属性指定发布的接口，并使用 interface 属性指定公布的 接口。

例如，您可以使用 例 12.1 “使用单一接口导出服务示例” 中显示的 Blueprint 配置代码，在 org.fusesource.example.Account 接口名称下导出 SavingsAccountImpl 类的实例。

例 12.1. 使用单一接口导出服务示例

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>

  <service ref="savings" interface="org.fusesource.example.Account"/>

</blueprint>
```

其中 ref 属性指定对应 bean 实例的 ID，而 interface 属性指定服务在 OSGi 服务注册表中注册的公共 Java 接口的名称。本例中使用的类和接口显示在中 例 12.2 “帐户类和接口示例”

例 12.2. 帐户类和接口示例

```
package org.fusesource.example
```

```
public interface Account { ... }

public interface SavingsAccount { ... }

public interface CheckingAccount { ... }

public class SavingsAccountImpl implements SavingsAccount
{
    ...
}

public class CheckingAccountImpl implements CheckingAccount
{
    ...
}
```

使用多个接口导出

要将服务导出到多个接口名称下的 OSGi 服务注册表，请定义一个 service 元素来引用相关服务 bean，使用 ref 属性指定公布的接口，并使用 interfaces 子元素指定发布的接口。

例如，您可以使用以下 Blueprint 配置代码在公共 Java 接口列表下导出 Savings Account Impl 类的实例：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings">
    <interfaces>
      <value>org.fusesource.example.Account</value>
      <value>org.fusesource.example.SavingsAccount</value>
    </interfaces>
  </service>
  ...
</blueprint>
```

注意

interface 属性和 interfaces 元素不能在同一 service 元素中同时使用。您必须使用一个或多个.

使用自动导出导出

如果要在 其所有 实施的公共 Java 接口下将服务导出到 OSGi 服务注册表，则可以使用 auto-export 属性完成此任务的简单方法。

例如，要在其所有实施的公共接口下导出 SavingsAccountImpl 类的实例，请使用以下 Blueprint 配置代码：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings" auto-export="interfaces"/>
  ...
</blueprint>
```

其中，auto-export 属性 的接口 值表示 Blueprint 应注册 SavingsAccountImpl 实施的所有公共接口。auto-export 属性可以具有以下有效值：

disabled

禁用自动导出。这是默认值。

interfaces

将服务注册到其所有实施的公共 Java 接口下。

class-hierarchy

将服务注册到其自己的类型(class)，并在所有 super-types (super-classes)下注册，但对象类除外。

all-classes

与 class-hierarchy 选项类似，但还包括所有实施的公共 Java 接口。

设置服务属性

OSGi 服务注册表还允许您将服务属性与注册 *的服务* 相关联。然后，服务的客户端可以使用服务属性来搜索或过滤服务。要将服务属性与导出的服务关联，请添加一个 service-properties 子元素，其中包含一个或多个 beans:entry 元素（每个服务属性有一个 beans:entry 元素）。

例如，要将 bank.name 字符串属性与节省帐户服务相关联，您可以使用以下 Blueprint 配置：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:beans="http://www.springframework.org/schema/beans"
           ...>
  ...
  <service ref="savings" auto-export="interfaces">
    <service-properties>
      <beans:entry key="bank.name" value="HighStreetBank"/>
```

```
    </service-properties>
  </service>
  ...
</blueprint>
```

其中 bank.name 字符串属性具有值 HighStreetBank。可以定义字符串以外的类型的服务属性：即，也支持原语类型、数组和集合。有关如何定义这些类型的详情，请参阅 Spring 参考指南 中的 控制公告属性 集。

注意

条目 元素 ought 属于 Blueprint 命名空间。Spring 的蓝图实现中的 beans:entry 元素的使用是非标准。

默认服务属性

使用 service 元素导出服务时可能会自动设置两个服务属性，如下所示：

- osgi.service.blueprint.compname-is 始终设置为服务的 bean 元素的 id，除非 bean 被内联（即，bean 被定义为 service 元素的子元素）。内联 Bean 始终是匿名的。
- 如果 ranking 属性为零，则会自动设置 service.ranking-is。

指定等级属性

如果捆绑包在服务 registry 中查找服务并找到多个匹配服务，您可以使用等级来确定返回哪些服务。规则是，每当查找与多个服务匹配时，会返回具有最高等级的服务。服务等级可以是任何非负整数，0 是默认值。您可以通过在 service 元素上设置 ranking 属性来指定服务等级，例如：

```
<service ref="savings" interface="org.fusesource.example.Account" ranking="10"/>
```

指定注册监听程序

如果要跟踪服务注册和未注册事件，您可以定义接收 *注册和未注册事件通知的注册监听程序* 回调 Bean。要定义注册监听程序，请在 service 元素中添加 registration-listener 子元素。

例如，以下 Blueprint 配置定义了一个监听器 bean bean, listenerBean，它由 registration-listener 元素引用，以便在 帐户 服务注册或取消注册时监听程序 bean 接收回调：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" ...>
  ...
  <bean id="listenerBean" class="org.fusesource.example.Listener"/>

  <service ref="savings" auto-export="interfaces">
    <registration-listener
        ref="listenerBean"
        registration-method="register"
        unregistration-method="unregister"/>
  </service>
  ...
</blueprint>
```

其中 registration-listener 元素的 ref 属性引用监听器 bean 的 id，registration-method 属性指定接收注册回调的监听程序方法的名称，unregistration-method 属性指定接收未注册回调的监听程序方法的名称。

以下 Java 代码显示了一个 Listener 类的示例定义，用于接收注册和未注册事件通知：

```
package org.fusesource.example;

public class Listener
{
    public void register(Account service, java.util.Map serviceProperties) {
        ...
    }

    public void unregister(Account service, java.util.Map serviceProperties) {
        ...
    }
}
```

方法名称 register 和 unregister 分别由 registration-method 和 unregistration-method 属性指定。这些方法的签名必须符合以下语法：

- **第一方法参数- 从服务对象类型分配的任何类型 T。换句话说，服务类或由服务类实施的任何接口的超级类型类。此参数包含服务实例，除非服务 bean 声明了要 原型化 的范围，在这种情况下，此参数为 null （当范围是 原型 的时，在注册时没有服务实例可用）。**

- **第二种方法参数- 必须是 java.util.Map 类型或 java.util.Dictionary 类型。此映射包含与此服务注册关联的服务属性。**

12.3. 导入服务

概述

这部分描述了如何获取和使用已导出到 OSGi 服务 registry 的服务的引用。您可以使用参考元素或 reference -list 元素来导入 OSGi 服务。reference 元素适合访问 无状态 服务，而 reference-list 元素则适合访问 有状态 服务。

管理服务引用

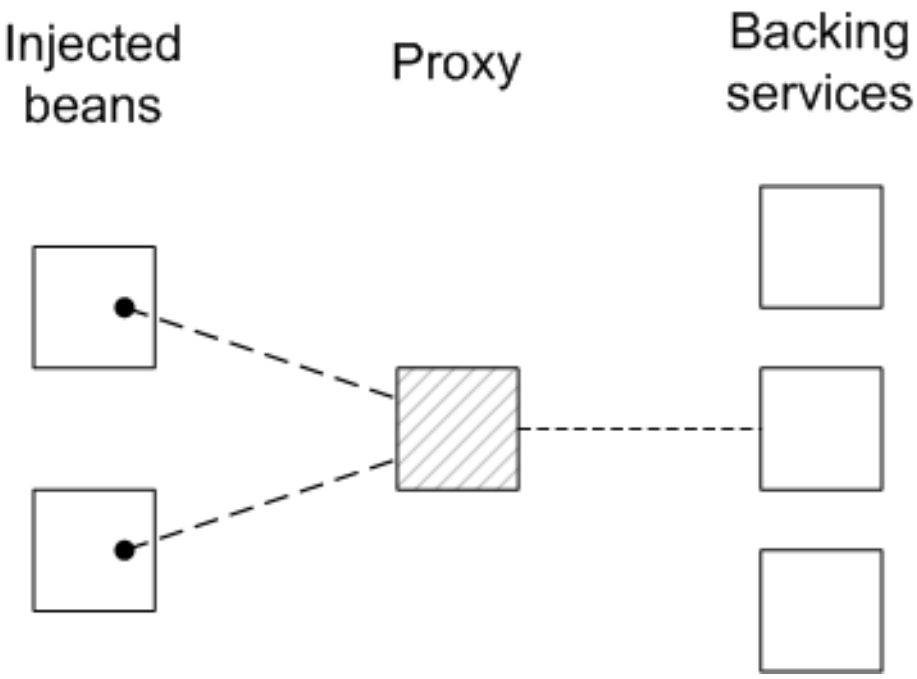
支持以下用于获取 OSGi 服务引用的模型：

- 参考资料管理器.
- 参考列表管理器.

参考资料管理器

参考管理器 实例由 Blueprint 参考 元素创建。此元素返回单个服务引用，是访问 无状态 服务的首选方法。图 12.1 “对无状态服务的引用” 显示了使用参考管理器访问无状态服务的模型概述。

图 12.1. 对无状态服务的引用



客户端蓝图容器中的 Bean 使用代理对象(*提供的对象*)注入，后者由 OSGi 服务 registry 中的 *服务对象（后备服务*）提供支持。此模型以以下方式明确利用无状态服务可交换的事实：

- 如果发现多个服务实例与 引用 元素中的条件匹配，则参考管理器可以任意选择其中一个作为后备实例（因为它们是相互交换的）。

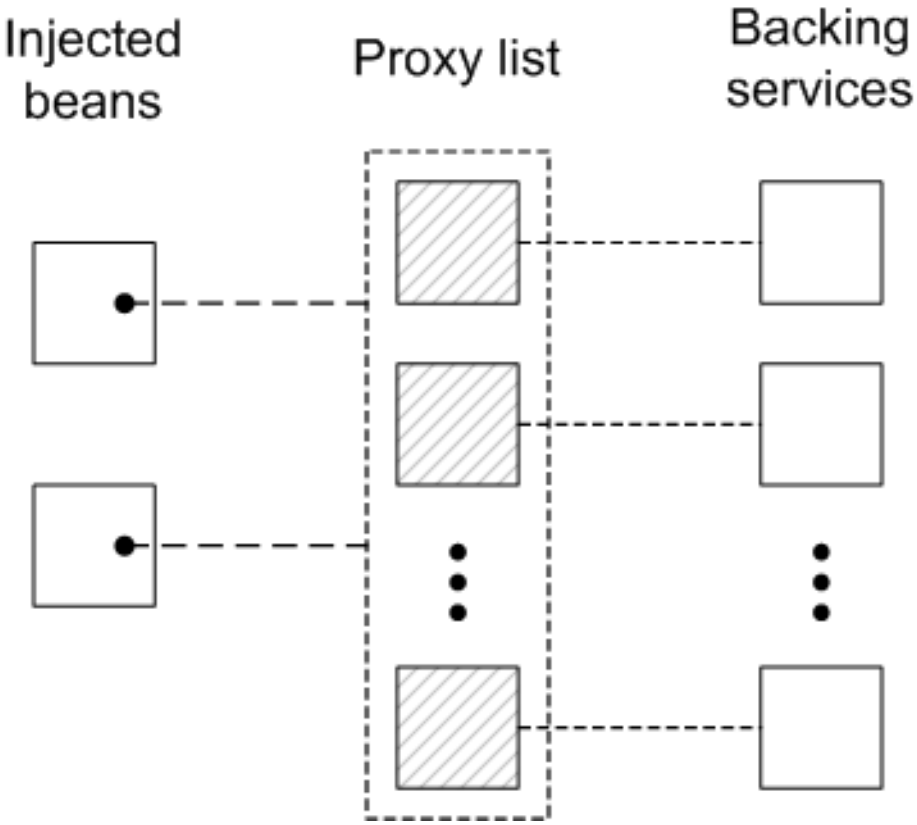
- 如果后备服务消失，则参考管理器可以立即切换为使用同一类型的其它可用服务之一。因此，不能保证，从一种方法调用下一个方法，代理仍连接到同一后端服务。

因此，客户端和后备服务之间的合同 是无状态的，客户端 不得 认为它始终与同一个服务实例通信。如果没有匹配的服务实例可用，代理会在抛出 ServiceUnavailable 异常前等待一定时间。通过在 reference 元素上设置 timeout 属性，可以配置超时的长度。

参考列表管理器

参考资料 *列表管理器* 实例由 Blueprint reference-list 元素创建。此元素返回服务引用列表，是访问有状态 服务的首选方法。图 12.2 “Stateful Services 的引用列表” 显示了使用参考列表管理器访问有状态服务的模型概述。

图 12.2. Stateful Services 的引用列表



客户端蓝图容器中的 Bean 使用 java.util.List 对象(*提供的对象*)注入，其中包含代理对象列表。每个代理都由 OSGi 服务注册表中的唯一服务实例提供支持。与无状态模型不同，后备服务 不被视为 在此处相互交换。实际上，列表中的每个代理的生命周期都与相应后备服务的生命周期紧密链接：当服务在 OSGi 注册表中注册时，对应的代理会被同步创建并添加到代理列表中；当服务从 OSGi 注册表中取消注册时，对应的代理将从代理列表中同步删除。

因此，代理及其后备服务之间的合同是 有状态的，客户端可能会在特定代理上调用方法时假设，它始终 与同一 后备服务通信。但是，可能会发生后备服务不可用，在这种情况下，代理会变得过时。在过时

的代理上调用方法的任何尝试都会生成 ServiceUnavailable 异常。

按接口匹配（无状态）

获取 状态 服务引用的最简单方法是通过使用 引用 元素上的 interface 属性来指定要匹配的接口。如果接口属性值是服务的超级类型，或者属性值是服务实施的 Java 接口(interface 属性可以指定 Java 类或 Java 接口)，则服务被认为是匹配的。

例如，要引用无状态 SavingsAccount 服务（请参阅 例 12.1 “使用单一接口导出服务示例”），请定义 参考 元素，如下所示：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
             interface="org.fusesource.example.SavingsAccount"/>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccount" ref="savingsRef"/>
  </bean>

</blueprint>
```

其中 reference 元素创建 ID 为 savingsRef 的参考管理器 bean。要使用引用的服务，请将 savingsRef bean 注入您的客户端类之一，如下所示。

注入客户端类的 bean 属性可以是从 SavingsAccount 分配的任何类型。例如，您可以定义 Client 类，如下所示：

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    SavingsAccount savingsAccount;

    // Bean properties
    public SavingsAccount getSavingsAccount() {
        return savingsAccount;
    }

    public void setSavingsAccount(SavingsAccount savingsAccount) {
        this.savingsAccount = savingsAccount;
    }
    ...
}
```

按接口匹配(stateful)

获取 有状态 服务引用的最简单方法是通过使用 reference-list 元素上的 interface 属性来指定匹配的接口。然后，参考列表管理器获取所有服务的列表，其接口属性值是服务的超级类型，或者由服务实施的 Java 接口(interface 属性可以指定 Java 类或 Java 接口)。

例如，要引用有状态的 SavingsAccount 服务（请参阅 例 12.1 “使用单一接口导出服务示例”），请定义 reference-list 元素，如下所示：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference-list id="savingsListRef"
                  interface="org.fusesource.example.SavingsAccount"/>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccountList" ref="savingsListRef"/>
  </bean>

</blueprint>
```

其中 reference-list 元素创建 ID 为 savingsListRef 的参考列表 manager bean。要使用引用的服务列表，请将 savingsListRef bean 引用注入其中一个客户端类，如下所示。

默认情况下，savingsAccountList bean 属性是一个服务对象列表（例如 java.util.List<SavingsAccount>）。您可以按照以下方式定义客户端类：

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    java.util.List<SavingsAccount> accountList;

    // Bean properties
    public java.util.List<SavingsAccount> getSavingsAccountList() {
        return accountList;
    }

    public void setSavingsAccountList(
                    java.util.List<SavingsAccount> accountList
    ) {
        this.accountList = accountList;
    }
    ...
}
```

根据接口和组件名称匹配

要匹配 无状态 服务的接口和组件名称(bean ID)，请在 reference 元素中指定 interface 属性和

component-name 属性，如下所示：

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        component-name="savings"/>
```

要匹配 有状态 服务的接口名称和组件名称(bean ID)，请在 reference-list 元素中指定 interface 属性和 component-name 属性，如下所示：

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        component-name="savings"/>
```

使用过滤器匹配服务属性

您可以通过针对过滤器匹配服务属性来选择服务。该过滤器使用 reference 元素上的 filter 属性或 reference -list 元素来指定。filter 属性的值必须是 *LDAP 过滤器表达式*。例如，要定义在 bank.name 服务属性等于 HighStreetBank 时匹配的过滤器，您可以使用以下 LDAP 过滤器表达式：

```
(bank.name=HighStreetBank)
```

要匹配两个服务属性值，您可以使用 & amp; 组合将表达式与逻辑 和.For 例如，要求 foo 属性等于 FooValue，bar 属性等于 BarValue，您可以使用以下 LDAP 过滤器表达式：

```
(&(foo=FooValue)(bar=BarValue))
```

有关 LDAP 过滤器表达式的完整语法，请参阅 OSGi 内核规格的 3.2.7 部分。

过滤器也可以与 interface 和 component-name 设置结合使用，在这种情况下，所有指定条件都需要匹配。

例如，要匹配 SavingsAccount 类型的 无状态 服务，其 bank.name 服务属性等于 HighStreetBank，您可以定义 参考 元素，如下所示：

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

要匹配 SavingsAccount 类型 的有状态 服务，其 bank.name 服务属性等于 HighStreetBank，您可以定义一个 reference-list 元素，如下所示：

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

指定是否强制或可选

默认情况下，假定对 OSGi 服务的引用是强制的（请参阅 必需的依赖项）。通过在元素上设置 availability 属性来自定义 reference 元素或 reference-list 元素的依赖项行为。

availability 属性有两个可能的值：

- **必需 （默认），这意味着在正常的 Blueprint 容器初始化过程中 必须 解决依赖项**
- **可选，表示在初始化过程中 不需要 解析依赖项。**

以下 reference 元素的示例演示了如何明确声明引用是强制依赖项：

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        availability="mandatory"/>
```

指定参考监听程序

为了应对 OSGi 环境的动态性质，例如，如果您声明了某些服务引用具有 可选 可用性，则通常在后备服务绑定到 registry 时以及从 registry 中绑定绑定时，这通常很有用。要接收服务绑定和 unbinding 事件的通知，您可以定义一个 reference-listener 元素作为 reference 元素或 reference-list 元素的子级。

例如，以下 Blueprint 配置演示了如何将参考监听程序定义为 ID 为 savingsRef 的参考管理器的子级：

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
          interface="org.fusesource.example.SavingsAccount"
          >
    <reference-listener bind-method="onBind" unbind-method="onUnbind">
      <bean class="org.fusesource.example.client.Listener"/>
    </reference-listener>
  </reference>
```

```
  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAcc" ref="savingsRef"/>
  </bean>

</blueprint>
```

上述配置将 org.fusesource.example.client.Listener 类型的实例注册为侦听 bind 和 unbind 事件的回调。每当 savingsRef 参考管理器的后备服务绑定或取消绑定时，都会生成事件。

以下示例显示了 Listener 类的示例实现：

```
package org.fusesource.example.client;

import org.osgi.framework.ServiceReference;

public class Listener {

    public void onBind(ServiceReference ref) {
        System.out.println("Bound service: " + ref);
    }

    public void onUnbind(ServiceReference ref) {
        System.out.println("Unbound service: " + ref);
    }

}
```

方法名称 onBind 和 onUnbind 分别由 bind-method 和 unbind-method 属性指定。这两个回调方法都采用 org.osgi.framework.ServiceReference 参数。

12.4. 发布 OSGI 服务

12.4.1. 概述

本节介绍如何在 OSGi 容器中生成、构建和部署简单的 OSGi 服务。该服务是一个简单的 Hello World Java 类，而 OSGi 配置是使用 Blueprint 配置文件定义的。

12.4.2. 先决条件

要使用 Maven Quickstart archetype 生成项目，您必须满足以下先决条件：

- Maven 安装- Maven 是 Apache 的免费开源构建工具。您可以从 http://maven.apache.org/download.html 下载最新版本（最小为 2.0.9）。

- 互联网连接- 执行构建，Maven 会动态搜索外部存储库，并即时下载所需的工件。为了正常工作，您的构建机器 必须 连接到互联网。

12.4.3. 生成 Maven 项目

maven-archetype-quickstart archetype 创建一个通用 Maven 项目，然后您可以针对您想要的任何目的进行自定义。要使用协调(org.fusesource.example:osgi-service)生成 Maven 项目，请输入以下命令：

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-service
```

此命令的结果是一个目录 *ProjectDir*/osgi-service，其中包含生成的项目的文件。

注意

请注意，不要为您的工件选择一个组 ID，这些工件与现有产品的组 ID 冲突！这可能会导致项目的软件包和现有产品中的软件包之间冲突（因为组 ID 通常用作项目 Java 软件包名称的根目录）。

12.4.4. 自定义 POM 文件

您必须自定义 POM 文件来生成 OSGi 捆绑包，如下所示：

1. 按照 第 5.1 节 “生成捆绑包项目” 中描述的 POM 自定义步骤操作。

2. 在 Maven 捆绑包插件配置中，修改捆绑包指令以导出 org.fusesource.example.service 软件包，如下所示：

```
<project ... >
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
```

```
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${pom.groupId}.${pom.artifactId}</Bundle-SymbolicName>
        <Export-Package>org.fusesource.example.service</Export-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

12.4.5. 编写服务接口

创建 ***ProjectDir*/osgi-service/src/main/java/org/fusesource/example/service** 子目录。在这个目录中，使用您首选的文本编辑器创建文件 **HelloWorldSvc.java**，并将代码从 **例 12.3 “HelloWorldSvc 接口”** 添加到其中。

例 12.3. HelloWorldSvc 接口

```
package org.fusesource.example.service;

public interface HelloWorldSvc
{
    public void sayHello();
}
```

12.4.6. 编写服务类

创建 ***ProjectDir*/osgi-service/src/main/java/org/fusesource/example/service/impl** 子目录。在这个目录中，使用您首选的文本编辑器创建文件 **HelloWorldSvcImpl.java**，并将代码从 **例 12.4 “HelloWorldSvcImpl 类”** 添加到该文件中。

例 12.4. HelloWorldSvcImpl 类

```
package org.fusesource.example.service.impl;

import org.fusesource.example.service.HelloWorldSvc;

public class HelloWorldSvcImpl implements HelloWorldSvc {

    public void sayHello()
    {
        System.out.println( "Hello World!" );
```

```
    }

}
```

12.4.7. 编写蓝图文件

Blueprint 配置文件是在类路径上的 OSGI-INF/blueprint 目录下存储的 XML 文件。要在项目中添加 Blueprint 文件，首先创建以下子目录：

```
ProjectDir/osgi-service/src/main/resources
ProjectDir/osgi-service/src/main/resources/OSGI-INF
ProjectDir/osgi-service/src/main/resources/OSGI-INF/blueprint
```

其中 src/main/resources 是所有 JAR 资源的标准 Maven 位置。此目录下的资源文件将自动打包到生成的捆绑包 JAR 的根目录范围内。

例 12.5 “用于导出服务的蓝图文件” 显示一个示例 Blueprint 文件，它会创建一个 HelloWorldSvc bean，使用 bean 元素，然后使用 service 元素将 bean 导出为 OSGi 服务。

在 *ProjectDir*/osgi-service/src/main/resources/OSGI-INF/blueprint 目录下，使用您首选的文本编辑器创建文件 config.xml，并添加来自 例 12.5 “用于导出服务的蓝图文件” 的 XML 代码。

例 12.5. 用于导出服务的蓝图文件

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.service.impl.HelloWorldSvcImpl"/>

  <service ref="hello" interface="org.fusesource.example.service.HelloWorldSvc"/>

</blueprint>
```

12.4.8. 运行服务捆绑包

要安装并运行 osgi-service 项目，请执行以下步骤：

1. 构建项目-open a command prompt，并将目录改为 *ProjectDir*/osgi-service。输入以下命令来使用 Maven 构建演示：

```
mvn install
```

如果这个命令成功运行，*ProjectDir*/osgi-service/target 目录应包含捆绑包文件 osgi-service-1.0-SNAPSHOT.jar。

2. 在 Red Hat Fuse 控制台中安装并启动 osgi-service 捆绑包-at，输入以下命令：

```
Jkaraf@root()> bundle:install -s file:ProjectDir/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
```

其中 *ProjectDir* 是包含 Maven 项目和 -s 标志的目录，则 -s 标志可指示容器立即启动捆绑包。例如，如果您的项目目录在 Windows 机器上是 C:\Projects，您可以输入以下命令：

```
karaf@root()> bundle:install -s file:C:/Projects/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
```

注意

在 Windows 机器中，注意如何格式化 文件 URL - 有关 文件 URL 处理程序理解的语法的详情，请参阅 第 15.1 节 “文件 URL 处理程序”。

3. 检查该服务是否已创建- 检查捆绑包是否已成功启动，输入以下 Red Hat Fuse console 命令：

```
karaf@root()> bundle:list
```

在此列表中，您应该会看到 osgi-service 捆绑包的行，例如：

```
[ 236] [Active     ] [Created     ] [       ] [   60] osgi-service (1.0.0.SNAPSHOT)
```

12.5. 访问 OSGI 服务

12.5.1. 概述

本节介绍如何在 OSGi 容器中生成、构建和部署简单的 OSGi 客户端。客户端在 OSGi 注册表中找到简单的 Hello World 服务，并在其上调用 sayHello （） 方法。

12.5.2. 先决条件

要使用 Maven Quickstart archetype 生成项目，您必须满足以下先决条件：

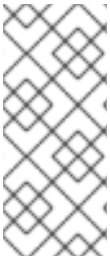
- Maven 安装- Maven 是 Apache 的免费开源构建工具。您可以从 http://maven.apache.org/download.html 下载最新版本（最小为 2.0.9）。
- 互联网连接- 执行构建，Maven 会动态搜索外部存储库，并即时下载所需的工件。为了正常工作，您的构建机器 必须 连接到互联网。

12.5.3. 生成 Maven 项目

maven-archetype-quickstart archetype 创建一个通用 Maven 项目，然后您可以针对您想要的任何目的进行自定义。要使用协调(org.fusesource.example:osgi-client)生成 Maven 项目，请输入以下命令：

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-client
```

此命令的结果是一个目录 *ProjectDir*/osgi-client，包含所生成的项目的文件。



注意

请注意，不要为您的工件选择一个组 ID，这些工件与现有产品的组 ID 冲突！这可能会导致项目的软件包和现有产品中的软件包之间冲突（因为组 ID 通常用作项目 Java 软件包名称的根目录）。

12.5.4. 自定义 POM 文件

您必须自定义 POM 文件来生成 OSGi 捆绑包，如下所示：

1. 按照 第 5.1 节 “生成捆绑包项目” 中描述的 POM 自定义步骤操作。
2. 由于客户端使用 HelloWorldSvc Java 接口，它在 osgi-service 捆绑包中定义，因此必须对 osgi-service 捆绑包添加 Maven 依赖项。假设 Maven 协调 osgi-service 捆绑包是

org.fusesource.example:osgi-service:1.0-SNAPSHOT，您应该将以下依赖项添加到客户端的 POM 文件中：

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

12.5.5. 编写蓝图文件

要在客户端项目中添加 Blueprint 文件，首先创建以下子目录：

```
ProjectDir/osgi-client/src/main/resources
ProjectDir/osgi-client/src/main/resources/OSGI-INF
ProjectDir/osgi-client/src/main/resources/OSGI-INF/blueprint
```

在 *ProjectDir*/osgi-client/src/main/resources/OSGI-INF/blueprint 目录下，使用您首选的文本编辑器创建文件 config.xml，并添加来自 例 12.6 “用于导入服务的蓝图文件” 的 XML 代码。

例 12.6. 用于导入服务的蓝图文件

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="helloWorld"
      interface="org.fusesource.example.service.HelloWorldSvc"/>

  <bean id="client"
      class="org.fusesource.example.client.Client"
      init-method="init">
    <property name="helloWorldSvc" ref="helloWorld"/>
  </bean>

</blueprint>
```

其中 reference 元素会创建一个参考管理器，它在 OSGi 注册表中找到 HelloWorldSvc 类型的服务。bean 元素创建 Client 类的实例，注入作为 bean 属性 helloWorldSvc 的服务引用。此外，init-

method 属性指定在 bean 初始化阶段调用 Client.init （） 方法（即，服务引用注入客户端 bean 后）。

12.5.6. 编写客户端类

在 *ProjectDir*/osgi-client/src/main/java/org/fusesource/example/client 目录下，使用您首选的文本编辑器创建文件 Client.java，并添加来自 例 12.7 “客户端类” 的 Java 代码。

例 12.7. 客户端类

```
package org.fusesource.example.client;

import org.fusesource.example.service.HelloWorldSvc;

public class Client {
    HelloWorldSvc helloWorldSvc;

    // Bean properties
    public HelloWorldSvc getHelloWorldSvc() {
        return helloWorldSvc;
    }

    public void setHelloWorldSvc(HelloWorldSvc helloWorldSvc) {
        this.helloWorldSvc = helloWorldSvc;
    }

    public void init() {
        System.out.println("OSGi client started.");
        if (helloWorldSvc != null) {
            System.out.println("Calling sayHello()");
            helloWorldSvc.sayHello();  // Invoke the OSGi service!
        }
    }

}
```

Client 类定义 getter 和 setter 方法，用于 helloWorldSvc bean 属性，它允许它通过注入来接收对 Hello World 服务的引用。init （） 方法在 bean 初始化阶段调用，在属性注入后调用，这意味着通常在此方法范围内调用 Hello World 服务。

12.5.7. 运行客户端捆绑包

要安装并运行 osgi-client 项目，请执行以下步骤：

1. 构建项目-open a command prompt，并将目录改为 *ProjectDir*/osgi-client。输入以下命

令来使用 Maven 构建演示：

```
mvn install
```

如果这个命令成功运行，*ProjectDir*/osgi-client/target 目录应包含捆绑包文件 osgi-client-1.0-SNAPSHOT.jar。

2. 在 Red Hat Fuse 控制台中安装并启动 osgi-service 捆绑包-at，输入以下命令：

```
karaf@root()> bundle:install -s file:ProjectDir/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```

其中 *ProjectDir* 是包含 Maven 项目和 -s 标志的目录，则 -s 标志可指示容器立即启动捆绑包。例如，如果您的项目目录在 Windows 机器上是 C:\Projects，您可以输入以下命令：

```
karaf@root()> bundle:install -s file:C:/Projects/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```

注意

在 Windows 机器中，注意如何格式化 文件 URL - 有关 文件 URL 处理程序理解的语法的详情，请参阅 第 15.1 节 “文件 URL 处理程序”。

3. 客户端输出-f 客户端捆绑包成功启动，您应该在控制台中立即看到类似如下的输出：

```
Bundle ID: 239
OSGi client started.
Calling sayHello()
Hello World!
```

12.6. 与 APACHE CAMEL 集成

12.6.1. 概述

Apache Camel 提供了使用 Bean 语言调用 OSGi 服务的简单方法。当 Apache Camel 应用程序部署到 OSGi 容器中且不需要特殊配置时，此功能会自动可用。

12.6.2. registry 链

当 Apache Camel 路由部署到 OSGi 容器中时，CamelContext 会自动设置用于解析 bean 实例的 registry 链：registry 链由 OSGi 注册表组成，后跟 Blueprint registry。现在，如果您尝试引用特定的 bean 类或 bean 实例，registry 会解析 bean，如下所示：

1. 首先在 OSGi 注册表中查找 bean。如果指定了类名称，请尝试与 OSGi 服务的接口或类匹配。

2. 如果在 OSGi 注册中心中没有找到匹配项，请回退到 Blueprint registry。

12.6.3. OSGi 服务接口示例

考虑由以下 Java 接口定义的 OSGi 服务，它定义了单一方法 getGreeting （） ：

```
package org.fusesource.example.hello.boston;

public interface HelloBoston {
    public String getGreeting();
}
```

12.6.4. 服务导出示例

在定义实现 HelloBoston OSGi 服务的捆绑包时，您可以使用以下 Blueprint 配置导出服务：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.hello.boston.HelloBostonImpl"/>

  <service ref="hello" interface="org.fusesource.example.hello.boston.HelloBoston"/>

</blueprint>
```

假设 HelloBoston 接口由 HelloBostonImpl 类（未显示）实现。

12.6.5. 从 Java DSL 调用 OSGi 服务

部署包含 HelloBoston OSGi 服务的捆绑包后，您可以使用 Java DSL 从 Apache Camel 应用程序调用该服务。在 Java DSL 中，您可以通过 Bean 语言调用 OSGi 服务，如下所示：

```
from("timer:foo?period=5000")
  .bean(org.fusesource.example.hello.boston.HelloBoston.class, "getGreeting")
  .log("The message contains: ${body}")
```

在 bean 命令中，第一个参数是 OSGi 接口或类，它必须与从 OSGi 服务捆绑包导出的接口匹配。第二个参数是您要调用的 bean 方法的名称。有关 bean 命令语法的详情，请参阅 Apache Camel 开发指南 Bean 集成。

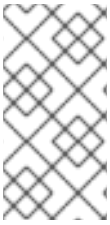
注意

使用这种方法时，将隐式导入 OSGi 服务。在这种情况下，不需要显式 导入 OSGi 服务。

12.6.6. 从 XML DSL 调用 OSGi 服务

在 XML DSL 中，您还可以使用 Bean 语言调用 HelloBoston OSGi 服务，但语法略有不同。在 XML DSL 中，您可以使用 method 元素通过 Bean 语言调用 OSGi 服务，如下所示：

```
<beans ...>
  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="timer:foo?period=5000"/>
      <setBody>
         <method ref="org.fusesource.example.hello.boston.HelloBoston" method="getGreeting"/>
      </setBody>
      <log message="The message contains: ${body}"/>
    </route>
  </camelContext>
</beans>
```



注意

使用这种方法时，将隐式导入 OSGi 服务。在这种情况下，不需要显式 导入 OSGi 服务。

第 13 章 使用 JMS 代理进行部署

摘要

Fuse 7.13 不附带默认的内部代理，但设计为与四个外部 JMS 代理的接口。

Fuse 7.13 容器包含支持的外部代理的代理客户端库。

如需有关可用于 Fuse 7.13 上消息传递的外部代理、客户端和 Camel 组件组合的更多信息，请参阅 支持的配置。

13.1. AMQ 7 快速入门

提供了快速入门来演示使用 AMQ 7 代理的应用程序设置和部署。

下载快速入门

您可以从 Fuse Software Downloads 页面安装所有快速入门。

将下载的 zip 文件的内容提取到本地文件夹中，例如，名为 quickstarts 的文件夹。

设置快速入门

1. 导航到 quickstarts/camel/camel-jms 文件夹。
2. 输入 mvn clean install 以构建 Quickstart。
3. 将文件 org.ops4j.connectionfactory-amq7.cfg 从 /camel/camel-jms/src/main 目录复制到 Fuse 安装中的 FUSE_HOME/etc 目录。验证它的内容以获取正确的代理 URL 和凭证。默认情况下，代理 URL 在 AMQ 7 的 CORE 协议后被设置为 tcp://localhost:61616。凭据设置为 admin/admin。更改这些详情以适合您的外部代理。
4. 通过在 Windows 上运行 ./bin/fuse on Linux 或 bin\fuse.bat 来启动 Fuse。

5. 在 Fuse 控制台中输入以下命令：

```
feature:install pax-jms-pool artemis-jms-client camel-blueprint camel-jms

install -s mvn:org.jboss.fuse.quickstarts/camel-jms/${project.version}
```

在部署捆绑包时，Fuse 将为您提供捆绑包 ID。

6. 输入 log:display 以查看启动日志信息。检查以确保捆绑包已被成功部署。

```
12:13:50.445 INFO [Blueprint Event Dispatcher: 1] Attempting to start Camel Context jms-example-context
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] Apache Camel 2.21.0.fuse-000030 (CamelContext: jms-example-context) is starting
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] JMX is enabled
12:13:50.528 INFO [Blueprint Event Dispatcher: 1] StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at http://camel.apache.org/stream-caching.html
12:13:50.553 INFO [Blueprint Event Dispatcher: 1] Route: file-to-jms-route started and consuming from: file://work/jms/input
12:13:50.555 INFO [Blueprint Event Dispatcher: 1] Route: jms-cbr-route started and consuming from: jms://queue:incomingOrders?transacted=true
12:13:50.556 INFO [Blueprint Event Dispatcher: 1] Total 2 routes, of which 2 are started
```

运行快速启动

1. 当 Camel 路由运行时，将创建 /camel/camel-jms/work/jms/input 目录。将 /camel/camel-jms/src/main/data 目录中的文件复制到 /camel/camel-jms/work/jms/input 目录。

2. 复制到 …/src/main/data 文件中的文件是订购文件。等待一分钟，然后检查 /camel/camel-jms/work/jms/output 目录。文件将根据其目标国家/地区排序为单独的目录：

- order1.xml,order2.xml 和 order4.xml in /camel/camel-jms/work/jms/output/others/
- order3.xml 和 order5.xml 位于 /camel/camel-jms/work/jms/output/us
- order6.xml in /camel/camel-jms/work/jms/output/fr

3. **使用 log:display 查看日志消息：**

```
Receiving order order1.xml
Sending order order1.xml to another country
Done processing order1.xml
```

1. **Camel 命令将显示上下文详情：**

使用 camel:context-list 显示上下文详情：

```
Context             Status            Total #       Failed #     Inflight #   Uptime
-------             ------            -------       --------     ----------   ------
jms-example-context   Started               12              0              0   3 minutes
```

使用 camel:route-list 来显示上下文中的 Camel 路由：

```
Context             Route              Status            Total #       Failed #     Inflight #   Uptime
-------             -----              ------            -------       --------     ----------   ------
jms-example-context   file-to-jms-route   Started               6              0              0   3 minutes
jms-example-context   jms-cbr-route       Started               6              0              0   3 minutes
```

使用 camel:route-info 显示交换统计信息：

```
karaf@root()> camel:route-info jms-cbr-route jms-example-context
Camel Route jms-cbr-route
    Camel Context: jms-example-context
    State: Started
    State: Started

Statistics
    Exchanges Total: 6
    Exchanges Completed: 6
    Exchanges Failed: 0
    Exchanges Inflight: 0
    Min Processing Time: 2 ms
    Max Processing Time: 12 ms
    Mean Processing Time: 4 ms
    Total Processing Time: 29 ms
    Last Processing Time: 4 ms
    Delta Processing Time: 1 ms
    Start Statistics Date: 2018-01-30 12:13:50
    Reset Statistics Date: 2018-01-30 12:13:50
    First Exchange Date: 2018-01-30 12:19:47
    Last Exchange Date: 2018-01-30 12:19:47
```

13.2. 使用 ARTEMIS 核心客户端

Artemis 核心客户端可用于连接到外部代理，而不是 qpid-jms-client。

使用 Artemis 核心客户端进行连接

1. 要启用 Artemis 核心客户端，启动 Fuse。导航到 FUSE_HOME 目录，然后在 Linux 或 Windows 上输入 ./ bin/fuse.bat。

2. 使用以下命令将 Artemis 客户端添加为功能： feature:install artemis-core-client

3. 当您编写代码时，您需要将 Camel 组件与连接工厂连接。

导入连接工厂：

```
import org.apache.qpid.jms.JmsConnectionFactory;
```

设置连接：

```
ConnectionFactory connectionFactory = new
JmsConnectionFactory("amqp://localhost:5672");
    try (Connection connection = connectionFactory.createConnection()) {
```

第 14 章 故障切换部署

摘要

红帽 Fuse 使用简单的锁定文件系统或 JDBC 锁定机制提供故障转移功能。在这两种情况下，容器级锁定系统都允许将捆绑包预加载到二级内核实例中，从而提供更快的故障切换性能。

14.1. 使用简单锁定文件系统

概述

当您首次启动 Red Hat Fuse 时，会在安装目录的根目录下创建一个锁定文件。您可以设置主实例故障时的主/次要系统，该锁定将传递到驻留在同一主机上的辅助实例。

配置锁定文件系统

要配置锁定文件故障转移部署，请编辑主和辅助安装中的 etc/system.properties 文件，以在 例 14.1 “锁定文件故障切换配置” 中包含属性。

例 14.1. 锁定文件故障切换配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.SimpleFileLock
karaf.lock.dir=PathToLockFileDirectory
karaf.lock.delay=10000
```

- Karaf.lock- 指定是否写入锁定文件。
- Karaf.lock.class-specifies Java 类实施锁定。对于简单的文件锁定，它应始终是 org.apache.karaf.main.SimpleFileLock。
- Karaf.lock.dir- 指定将锁定文件写入的目录。对于主安装和次安装，这 必须相同。

- Karaf.lock.delay-specifies，以毫秒为单位，尝试查询锁定之间的延迟。

14.2. 使用 JDBC 锁定系统

概述

JDBC 锁定机制主要用于在独立的机器上存在红帽 Fuse 实例的故障转移部署。

在这种情况下，主实例在托管在数据库上的锁定表中保存锁定。如果主实例丢失了锁定，则等待次要进程可以访问锁定表，并完全启动其容器。

将 JDBC 驱动程序添加到 classpath

在 JDBC 锁定系统中，JDBC 驱动程序需要位于主/次要设置中各个实例的类路径上。在 classpath 中添加 JDBC 驱动程序，如下所示：

1. 将 JDBC 驱动程序 JAR 文件复制到每个红帽 Fuse 实例的 *ESBInstallDir*/lib/ext 目录中。

2. 修改 bin/karaf 启动脚本，使其在其 CLASSPATH 变量中包含 JDBC 驱动程序 JAR。

   例如，如果 JDBC JAR 文件 *JDBCJarFile*.jar，您可以按如下方式修改启动脚本（在 *NIX 操作系统上）：

```
...
# Add the jars in the lib dir
for file in "$KARAF_HOME"/lib/karaf*.jar
do
    if [ -z "$CLASSPATH" ]; then
        CLASSPATH="$file"
    else
        CLASSPATH="$CLASSPATH:$file"
    fi
done
CLASSPATH="$CLASSPATH:$KARAF_HOME/lib/JDBCJarFile.jar"
```

注意

如果要添加 MySQL 驱动程序 JAR 或 PostgreSQL 驱动程序 JAR，您必须用 karaf- 前缀作为前缀来重命名驱动程序 JAR。否则，Apache Karaf 将挂起，日志将告诉您 Apache Karaf 无法找到驱动程序。

配置 JDBC 锁定系统

要配置 JDBC 锁定系统，请更新主/次要部署中每个实例的 etc/system.properties 文件，如下所示

例 14.2. JDBC 锁定文件配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DefaultJDBCLock
karaf.lock.level=50
karaf.lock.delay=10000
karaf.lock.jdbc.url=jdbc:derby://dbserver:1527/sample
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

在示例中，如果不存在，则会创建一个名为 sample 的数据库。第一个用于获取锁定表的 Red Hat Fuse 实例是主实例。如果与数据库的连接丢失，主实例将尝试正常关闭，允许次要实例在恢复数据库服务时成为主实例。前一个主实例需要手动重启。

在 Oracle 上配置 JDBC 锁定

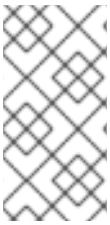
如果您在 JDBC 锁定场景中使用 Oracle 作为数据库，则 etc/system.properties 文件中的 karaf.lock.class 属性必须指向 org.apache.karaf.main.lock.oracle 2.0.0Lock。

否则，为您的设置配置 system.properties 文件，如下所示：

例 14.3. Oracle 的 JDBC 锁定文件配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.OracleJDBCLock
karaf.lock.jdbc.url=jdbc:oracle:thin:@hostname:1521:XE
karaf.lock.jdbc.driver=oracle.jdbc.OracleDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
```

```
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```



注意

karaf.lock.jdbc.url 需要活跃的 Oracle 系统 ID (SID)。这意味着，您必须在使用此特定锁定前手动创建数据库实例。

在 Derby 上配置 JDBC 锁定

如果您在 JDBC 锁定场景中将 Derby 用作数据库，则 etc/system.properties 文件中的 karaf.lock.class 属性应指向 org.apache.karaf.main.lock.DerbyDatabaseLock。例如，您可以配置 system.properties 文件，如下所示：

例 14.4. 用于 Derby 的 JDBC 锁定文件配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DerbyJDBCLock
karaf.lock.jdbc.url=jdbc:derby://127.0.0.1:1527/dbname
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

在 MySQL 上配置 JDBC 锁定

如果您在 JDBC 锁定场景中使用 MySQL 作为数据库，则 etc/system.properties 文件中的 karaf.lock.class 属性必须指向 org.apache.karaf.main.lock.MySQL Database。例如，您可以配置 system.properties 文件，如下所示：

例 14.5. MySQL 的 JDBC 锁定文件配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.MySQLJDBCLock
karaf.lock.jdbc.url=jdbc:mysql://127.0.0.1:3306/dbname
karaf.lock.jdbc.driver=com.mysql.jdbc.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

在 PostgreSQL 上配置 JDBC 锁定

如果您在 JDBC 锁定场景中将 PostgreSQL 用作数据库，则 etc/system.properties 文件中的 karaf.lock.class 属性必须指向 org.apache.karaf.main.lock.PostgreSQL JDBCLock。例如，您可以配置 system.properties 文件，如下所示：

例 14.6. PostgreSQL 的 JDBC 锁定文件配置

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.PostgreSQLJDBCLock
karaf.lock.jdbc.url=jdbc:postgresql://127.0.0.1:5432/dbname
karaf.lock.jdbc.driver=org.postgresql.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=0
```

JDBC 锁定类

Apache Karaf 当前提供以下 JDBC 锁定类：

```
org.apache.karaf.main.lock.DefaultJDBCLock
org.apache.karaf.main.lock.DerbyJDBCLock
org.apache.karaf.main.lock.MySQLJDBCLock
org.apache.karaf.main.lock.OracleJDBCLock
org.apache.karaf.main.lock.PostgreSQLJDBCLock
```

14.3. 容器级别的锁定

概述

容器级锁定允许将捆绑包预加载到二级内核实例中，从而提供更快的故障切换性能。简单文件和 JDBC 锁定机制都支持容器级锁定。

配置容器级别锁定

要实现容器级别的锁定，请将以下内容添加到主/次要设置中的每个系统上的 etc/system.properties 文件中：

例 14.7. 容器级别的锁定配置

```
karaf.lock=true
karaf.lock.level=50
karaf.lock.delay=10000
```

karaf.lock.level 属性告知 Red Hat Fuse 实例启动 OSGi 容器的引导过程多多。然后，分配到同一起始级别或较低的捆绑包也会在该 Fuse 实例中启动。

捆绑包启动级别在 **etc/startup.properties 中指定，格式为 *BundleName*.jar=level。核心系统捆绑包的级别低于 50，其中用户捆绑包的级别大于 50。**

表 14.1. 捆绑包开始级别

开始级别	行为
1	'cold' 备用实例。核心捆绑包不会加载到容器中。辅助实例将等到锁定启动服务器前等待。
<50	'hot' 备用实例。核心捆绑包被加载到容器中。辅助实例将等到锁定到启动用户级捆绑包为止。此级别的每个二级实例都可以访问控制台。
>50	不建议此设置，因为将启动用户捆绑包。

避免端口冲突

当在同一主机上使用"热"备用时，您需要将 JMX 远程端口设置为唯一值，以避免绑定冲突。您可以编辑 fuse 启动脚本（或子实例上的 karaf 脚本），使其包含以下内容：

```
DEFAULT_JAVA_OPTS="-server $DEFAULT_JAVA_OPTS -
Dcom.sun.management.jmxremote.port=1100 -
Dcom.sun.management.jmxremote.authenticate=false"
```

第 15 章 URL 处理程序

红帽 Fuse 中有很多上下文，您需要提供 URL 来指定资源的位置（例如，作为 console 命令的参数）。通常，在指定 URL 时，您可以使用 Fuse 的内置 URL 处理程序支持的任何方案。本附录描述了所有可用 URL 处理程序的语法。

15.1. 文件 URL 处理程序

15.1.1. 语法

文件 URL 的语法为*PathName*，其中 *PathName* 是类路径上可用的文件的相对或绝对路径名。提供的 *PathName* 由 Java 的内置文件 *URL 处理程序* 解析。因此，*PathName* 语法受到 Java 路径名称的常用约定：特别是，在 Windows 上，每个反斜杠必须被另一个反斜杠转义或被正斜杠替代。

15.1.2. 例子

例如，在 Windows 上考虑 pathname, C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar。以下示例显示了 Windows 上文件 URL 的正确 替代方案：

```
file:C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar
file:C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar
```

以下示例显示了 Windows 上文件 URL 的一些 不正确的 替代方案：

```
file:C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar       // WRONG!
file://C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar     // WRONG!
file://C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar  // WRONG!
```

15.2. HTTP URL 处理程序

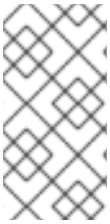
15.2.1. 语法

HTTP URL 有标准语法 http:*Host*[:*Port*]/[*Path*][*#AnchorName*][?*Query*]。您还可以使用 https 方案指定安全 HTTP URL。提供的 HTTP URL 由 Java 的内置 HTTP URL 处理程序解析，因此 HTTP URL 的行为是 Java 应用的一般方式。

15.3. MVN URL HANDLER

15.3.1. 概述

如果使用 Maven 构建捆绑包，或者您知道特定捆绑包可从 Maven 存储库获得，您可以使用 Mvn 处理程序方案来定位捆绑包。



注意

为确保 Mvn URL 处理程序可以找到本地和远程 Maven 工件，您可能会发现自定义 Mvn URL 处理程序配置的必要。详情请查看 第 15.3.5 节 “配置 Mvn URL 处理程序”。

15.3.2. 语法

Mvn URL 的语法如下：

```
mvn:[repositoryUrl!]groupId/artifactId[/[version][/[packaging][/[classifier]]]]
```

其中 ***repositoryUrl*** （可选）指定 Maven 存储库的 URL。***groupId***,***artifactId***,***version***,***packaging*** , 和 **classifier** 是标准的 Maven 协调，用于定位 Maven 工件。

15.3.3. 省略协调

当指定 Mvn URL 时，只需要 ***groupId*** 和 ***artifactId*** 协调。以下示例引用了带有 ***groupId***、**org.fusesource.example** 的 Maven 捆绑包，以及 ***artifactId***,**bundle-demo**:

```
mvn:org.fusesource.example/bundle-demo
mvn:org.fusesource.example/bundle-demo/1.1
```

当省略了 *版本时*，如第一个示例所示，默认为 **LATEST**，它根据可用的 Maven 元数据解析为最新版本。

要在没有指定 *打包* 或 ***版本*** 值的情况下指定 *分类器* 值，最好在 Mvn URL 中留下差距。同样，如果您要指定没有 ***version*** 值的 *打包* 值。例如：

```
mvn:groupId/artifactId///classifier
mvn:groupId/artifactId/version//classifier
mvn:groupId/artifactId//packaging/classifier
mvn:groupId/artifactId//packaging
```

15.3.4. 指定版本范围

在 Mvn URL 中指定 *版本* 值时，您可以指定一个版本范围（使用标准 Maven 版本范围语法）来代替简单的版本号。您可以使用方括号[和]- 表示包含范围和括号(和)- 表示专用范围。例如，范围 [1.0.4,2.0） 匹配任何版本 v，它满足 1.0.4 categories v < 2.0。您可以在 Mvn URL 中使用这个版本范围，如下所示：

```
mvn:org.fusesource.example/bundle-demo/[1.0.4,2.0)
```

15.3.5. 配置 Mvn URL 处理程序

首次使用 Mvn URL 之前，您可能需要自定义 Mvn URL 处理程序设置，如下所示：

1. 第 15.3.6 节 “检查 Mvn URL 设置”.
2. 第 15.3.7 节 “编辑配置文件”.
3. 第 15.3.8 节 “自定义本地存储库的位置”.

15.3.6. 检查 Mvn URL 设置

Mvn URL 处理程序解析对本地 Maven 存储库的引用，并维护远程 Maven 存储库列表。解析 Mvn URL 时，处理程序会首先搜索本地存储库，然后搜索远程存储库以查找指定的 Maven 模版。如果解析 Mvn URL 存在问题，您应该首先检查处理器设置，以查看处理器设置以查看它用于解析 URL 的本地存储库和远程存储库。

要检查 Mvn URL 设置，请在控制台中输入以下命令：

```
JBossFuse:karaf@root> config:edit org.ops4j.pax.url.mvn
JBossFuse:karaf@root> config:proplist
```

config:edit 命令将 config:edit 命令的重点切换到属于 org.ops4j.pax.url.mvn persistent ID 的属性。config:proplist 命令输出当前持久性 ID 的所有属性设置。对于 org.ops4j.pax.url.mvn，您应该会看到类似如下的列表：

```
   org.ops4j.pax.url.mvn.defaultRepositories = file:/path/to/JBossFuse/jboss-fuse-7.13.0.fuse-7_13_0-00012-redhat-
00001/system@snapshots@id=karaf.system,file:/home/userid/.m2/repository@snapshots@id=local,file:
path/to/JBossFuse/jboss-fuse-7.13.0.fuse-7_13_0-00012-redhat-00001/local-
repo@snapshots@id=karaf.local-repo,file:/path/to/JBossFuse/jboss-fuse-7.13.0.fuse-7_13_0-00012-
redhat-00001/system@snapshots@id=child.karaf.system
```

```
  org.ops4j.pax.url.mvn.globalChecksumPolicy = warn
  org.ops4j.pax.url.mvn.globalUpdatePolicy = daily
  org.ops4j.pax.url.mvn.localRepository = /path/to/JBossFuse/jboss-fuse-7.13.0.fuse-7_13_0-00012-redhat-00001/data/repository
  org.ops4j.pax.url.mvn.repositories = http://repo1.maven.org/maven2@id=maven.central.repo, https://maven.repository.redhat.com/ga@id=redhat.ga.repo, https://maven.repository.redhat.com/earlyaccess/all@id=redhat.ea.repo, https://repository.jboss.org/nexus/content/groups/ea@id=fuseearlyaccess
  org.ops4j.pax.url.mvn.settings = /path/to/jboss-fuse-7.13.0.fuse-7_13_0-00012-redhat-00001/etc/maven-settings.xml
  org.ops4j.pax.url.mvn.useFallbackRepositories = false
  service.pid = org.ops4j.pax.url.mvn
```

其中 localRepository 设置显示处理器当前使用的本地存储库位置，而 存储库 设置则显示处理程序当前使用的远程存储库列表。

15.3.7. 编辑配置文件

要自定义 Mvn URL 处理程序的属性设置，请编辑以下配置文件：

```
InstallDir/etc/org.ops4j.pax.url.mvn.cfg
```

此文件中的设置允许您指定本地 Maven 存储库的位置、删除 Maven 存储库、Maven 代理服务器设置等。有关这些设置的详情，请查看配置文件中的注释。

15.3.8. 自定义本地存储库的位置

特别是，如果您的本地 Maven 存储库位于非默认位置，您可以明确配置它，以便访问您本地构建的 Maven 工件。在 org.ops4j.pax.url.mvn.cfg 配置文件中，取消注释 org.ops4j.pax.url.mvn.localRepository 属性，并将其设置为本地 Maven 存储库的位置。例如：

```
# Path to the local maven repository which is used to avoid downloading
# artifacts when they already exist locally.
# The value of this property will be extracted from the settings.xml file
# above, or defaulted to:
#     System.getProperty( "user.home" ) + "/.m2/repository"
#
org.ops4j.pax.url.mvn.localRepository=file:E:/Data/.m2/repository
```

15.3.9. 参考

有关 mvn URL 语法的详情，请查看原始 Pax URL Mvn 协议 文档。

15.4. 嵌套 URL 处理程序

15.4.1. 概述

如果您需要引用尚未打包为捆绑包的 JAR 文件，您可以使用 Wrap URL 处理程序来动态转换它。Wrap URL 处理程序的实施基于 Peter Krien 的开源 Bnd 实用程序。

15.4.2. 语法

Wrap URL 的语法如下：

```
wrap:locationURL[,instructionsURL][$instructions]
```

locationURL 可以是找到 JAR 的任何 URL （引用 JAR 的位置 不是 捆绑包的格式化）。可选 *的说明URL* 引用一个第二属性文件，该文件指定如何执行捆绑包转换。可选 *的说明* 是一个 符号，以及 分隔的 Bnd 属性列表，用于指定如何执行捆绑包转换。

15.4.3. 默认说明

在大多数情况下，默认的 Bnd 指令适合嵌套 API JAR 文件。默认情况下，Wrap 将清单标头添加到 JAR 的 META-INF/Manifest.mf 文件中，如 表 15.1 “用于编写 JAR 的默认指令” 所示。

表 15.1. 用于编写 JAR 的默认指令

清单标头	默认值
import-Package	***;resolution:=optional**
export-Package	来自嵌套 JAR 的所有软件包。
Bundle-SymbolicName	JAR 文件的名称，其中不属于 set **[a-zA-Z0-9_-]** 的任何字符都被下划线 _ 替换。

15.4.4. 例子

以下 Wrap URL 在 Maven 存储库中找到 commons-logging JAR 版本 1.1，并使用默认的 Bnd 属性将其转换为 OSGi 捆绑包：

```
wrap:mvn:commons-logging/commons-logging/1.1
```

以下 Wrap URL 使用来自文件 E:\Data\Examples\commons-logging-1.1.bnd 的 Bnd 属性：

```
wrap:mvn:commons-logging/commons-logging/1.1,file:E:/Data/Examples/commons-logging-1.1.bnd
```

以下 Wrap URL 明确指定 Bundle-SymbolicName 属性和 Bundle-Version 属性：

```
wrap:mvn:commons-logging/commons-logging/1.1$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

如果前面的 URL 用作命令行参数，则可能需要转义 $ 符号 \$ 以防止命令行处理它，如下所示：

```
wrap:mvn:commons-logging/commons-logging/1.1\$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

15.4.5. 参考

有关 嵌套 URL 处理程序的详情，请查看以下引用：

- Bnd 工具文档，以了解有关第 Bnd 属性和第二条指令文件的详细信息。
- 原始 Pax URL 封装协议 文档。

15.5. WAR URL 处理程序

15.5.1. 概述

如果您需要在 OSGi 容器中部署 WAR 文件，您可以通过为 war 前缀的 WAR URL 自动将必要的清单标头添加到 WAR 文件中，如下所述。

15.5.2. 语法

使用以下语法之一指定 War URL：

```
war:warURL
warref:instructionsURL
```

第一个语法（使用 war 方案）指定一个 WAR 文件，该文件使用默认指令转换为捆绑包中。*warURL* 可以是找到 WAR 文件的任何 URL。

第二个语法使用 warref 方案，指定第二属性文件 *instructionsURL*，其中包含转换指令（包括特定于此处理程序的一些指令）。在这个语法中，引用 WAR 文件的位置 不会在 URL 中明确显示。WAR 文件由属性文件中的（必需） WAR-URL 属性指定。

15.5.3. 特定于 WAR 的属性/结构

.bnd 指令文件中的一些属性特定于 War URL 处理程序，如下所示：

WAR-URL

（必需）指定要转换为捆绑包的 War 文件的位置。

Web-ContextPath

指定在 Web 容器中部署后用于访问此 Web 应用的 URL 路径的部分。

注意

PAX Web 的早期版本使用属性 Webapp-Context，它 现已弃用。

15.5.4. 默认说明

默认情况下，War URL 处理程序将清单标头添加到 WAR 的 META-INF/Manifest.mf 文件中，如 表 15.2 “用于编写 WAR 文件的默认说明” 所示。

表 15.2. 用于编写 WAR 文件的默认说明

清单标头	默认值
import-Package	javax.,org.xml.,org.w3c.*
export-Package	没有导出软件包。
Bundle-SymbolicName	WAR 文件的名称，其中任何不在集合中的字符 [a-zA-Z0-9_-\ .] 替换为句点。

清单标头	默认值
Web-ContextPath	没有默认值。但是 *WAR* 扩展器默认使用 **Bundle-SymbolicName** 的值。
bundle-ClassPath	除了明确指定的任何类路径条目外，会自动添加以下条目： • **.** • **WEB-INF/classes** • 来自 **WEB-INF/lib** 目录的所有 JAR。

15.5.5. 例子

以下 War URL 在 Maven 存储库中找到 wicket-examples WAR 的 1.4.7 版本，并使用默认说明将其转换为 OSGi 捆绑包：

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war
```

以下 Wrap URL 明确指定 Web-ContextPath ：

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war?Web-ContextPath=wicket
```

以下 War URL 会将 WAR-URL 属性引用的 WAR 文件转换为 wicket-examples-1.4.7.bnd 文件中的 WAR 文件，然后使用 .bnd 文件中的其他指令将 WAR 转换为 OSGi 捆绑包：

```
warref:file:E:/Data/Examples/wicket-examples-1.4.7.bnd
```

15.5.6. 参考

有关 war URL 语法的详情，请查看原始 Pax URL 封装 协议 文档。

部分 II. 用户指南

本节包含在红帽 Fuse 上 Apache Karaf 的配置和准备信息。

第 16 章 在 APACHE KARAF 中部署指南简介

摘要

在 Apache Karaf 指南中的使用此用户指南一节前，您必须已安装了最新版本的 Red Hat Fuse，按照在 Apache Karaf 上安装 中的说明进行操作。

16.1. FUSE 配置简介

OSGi 配置管理服务指定已部署服务的配置信息，并确保服务在活动时收到这些数据。

16.2. OSGI 配置

配置是从 FUSE_HOME/etc 目录中的 .cfg 文件中读取的名称值对列表。该文件使用 Java 属性文件格式进行解释。文件名映射到要配置的服务的持久标识符(PID)。在 OSGi 中，PID 用于在容器重启后识别服务。

16.3. 配置文件

您可以使用以下文件配置 Red Hat Fuse 运行时：

表 16.1. Fuse 配置文件

fileName	描述
config.properties	容器的主配置文件。
custom.properties	容器自定义属性的主要配置文件。
keys.properties	列出可以使用基于 SSH 密钥的协议访问 Fuse 运行时的用户。文件的内容采用 ***username=publicKey*,*role*** 格式
org.apache.karaf.features.repos.cfg	features 存储库 URL。
org.apache.karaf.features.cfg	配置要注册的功能存储库列表，以及 Fuse 首次启动时要安装的功能列表。
org.apache.karaf.jaas.cfg	配置 Karaf JAAS 登录模块的选项。主要用于配置加密的密码（默认为禁用）。
org.apache.karaf.log.cfg	**配置日志** 控制台命令的输出。

fileName	描述
org.apache.karaf.management.cfg	配置 JMX 系统。
org.apache.karaf.shell.cfg	配置远程控制台的属性。
org.ops4j.pax.logging.cfg	配置日志记录系统。
org.ops4j.pax.transx.tm.narayana.cfg	Narayana 事务管理器配置
org.ops4j.pax.url.mvn.cfg	配置额外的 URL 解析器。
org.ops4j.pax.web.cfg	配置默认的 Undertow 容器(Web 服务器)。请参阅 Red Hat Fuse Apache CXF 安全指南中的 保护 Undertow HTTP 服务器。
startup.properties	指定容器中启动哪些捆绑包及其启动级别。条目采用格式 ***bundle=起始级别***。
system.properties	指定 Java 系统属性。此文件中设置的任何属性均可使用 **System.getProperties （）** 在运行时可用。
users.properties	列出可以远程访问 Fuse 运行时的用户，或者通过 Web 控制台访问。文件的内容采用 ***username=password,role***格式
SetEnv 或 **setenv.bat**	此文件位于 **/bin** 目录中。它用于设置 JVM 选项。文件的内容采用 **JAVA_MIN_MEM=*512M***格式，其中 *512M* 是 Java 内存的最小大小。请参阅 第 16.6 节 “设置 Java 选项” 了解更多信息。

16.4. 高级 UNDERTOW 配置

16.4.1. IO 配置

从 PAXWEB-1255 开始，可以更改监听器使用的 XNIO worker 和缓冲池的配置。在 undertow.xml 模板中，有一个部分指定某些 IO 相关参数的默认值：

```
<!-- Only "default" worker and buffer-pool are supported and can be used to override the default
values used by all listeners
    buffer-pool:
     - buffer-size defaults to:
        - when < 64MB of Xmx: 512
        - when < 128MB of Xmx: 1024
        - when >= 128MB of Xmx: 16K - 20
     - direct-buffers defaults to:
        - when < 64MB of Xmx: false
```

```
        - when >= 64MB of Xmx: true
    worker:
     - io-threads defaults to Math.max(Runtime.getRuntime().availableProcessors(), 2);
     - task-core-threads and task-max-threads default to io-threads * 8
-->

<!--
<subsystem xmlns="urn:jboss:domain:io:3.0">
    <buffer-pool name="default" buffer-size="16364" direct-buffers="true" />
    <worker name="default" io-threads="8" task-core-threads="64" task-max-threads="64" task-
keepalive="60000" />
</subsystem>
-->
```

可以指定以下 buffer-pool 参数：

buffer-size

指定用于 IO 操作的缓冲区的大小。如果没有指定，则根据可用内存计算大小。

direct-buffers

确定是否应使用 java.nio.ByteBuffer#allocateDirect 或 java.nio.ByteBuffer#allocate。

可以指定以下 worker 参数：

io-threads

为 worker 创建的 I/O 线程数量。如果没有指定，则线程数量被设置为 CPU hieradata 2 的数量。

task-core-threads

核心任务线程池的线程数量。

task-max-threads

worker 任务线程池的最大线程数量。如果没有指定，则最大线程数设置为 CPU Thycotic 16 的数量。

16.4.2. worker IO 配置

Undertow 线程池及其名称可以在每个服务或捆绑包上配置，这有助于提高从 Hawtio 控制台和调试的监控效率。

在捆绑包蓝图配置文件中（通常存储在 Maven 项目中的 src/main/resources/OSGI-INF/blueprint 目录下），您可以配置 workerIOName 和 ThreadPool，如下例所示。

例 16.1. httpu:engine-factory 元素带有 workerIOName 和 ThreadPool 配置

```
<httpu:engine-factory>
    <httpu:engine port="9001">
        <httpu:threadingParameters minThreads="99" maxThreads="777" workerIOThreads="8"
workerIOName="WorkerIOTest"/>
    </httpu:engine>
</httpu:engine-factory>
```

可以指定以下 threadingParameters ：

minThreads

指定 worker 任务线程池的 "core" 线程数量。通常，这应该具有合理性，每个 CPU 内核至少为 10 个。

maxThreads

指定 worker 任务线程池的最大线程数量。

可以指定以下 worker 参数：

workerIOThreads

指定为 worker 创建的 I/O 线程数量。如果未指定，则会选择默认值。每个 CPU 内核有一个 IO 线程是合理的默认值。

workerIOName

指定 worker 的名称。如果没有指定，则会选择默认的 "XNIO-1"。

16.5. 配置文件命名规则

配置文件的文件命名约定取决于配置是否用于 OSGi 托管服务还是 OSGi 受管服务工厂。

OSGi 管理的服务的配置文件遵循以下命名约定：

-

```
<PID>.cfg
```

其中 <PID > 是 OSGi 管理服务 *的持久 ID*（如 OSGi 配置管理规格中定义的）。持久 ID 通常是以点分隔的，例如 org.ops4j.pax.web。

Gi Managed Service Factory 的配置文件遵循以下命名约定：

```
<PID>-<InstanceID>.cfg
```

其中 <PID > 是 OSGi 管理的服务工厂 *的持久 ID*。对于受管服务工厂的 < PID>， 您可以附加一个连字符，后跟一个任意实例 ID < InstanceID>。然后，受管服务工厂会为找到的每个 < InstanceID> 创建一个唯一的服务实例。

16.6. 设置 JAVA 选项

Java 选项可以使用 Linux 中的 /bin/setenv 文件设置，或者 Windows 中的 bin/setenv.bat 文件。使用此文件直接设置一组 Java 选项： JAVA_MIN_MEM, JAVA_MAX_MEM, JAVA_PERM_MEM, JAVA_MAX_PERM_MEM。其他 Java 选项可使用 EXTRA_JAVA_OPTS 变量设置。

例如，要为 JVM 分配最小内存

```
JAVA_MIN_MEM=512M # Minimum memory for the JVM
```

```
To set a Java option other than the direct options, use
```

```
EXTRA_JAVA_OPTS="Java option"
```

```
For example,
```

```
EXTRA_JAVA_OPTS="-XX:+UseG1GC"
```

16.7. 配置控制台命令

有许多控制台命令可用于更改或划分 Fuse 7.13 的配置。

有关 config: 命令的更多详细信息，请参阅 Apache Karaf 控制台参考中的 Config 部分。

16.8. JMX CONFIGMBEAN

在 JMX 层上，MBean 专用于配置管理。

ConfigMBean 对象名称是： org.apache.karaf:type=config,namePROFILE'。

14.1.2.1.属性

config MBean 包含所有配置 PID 的列表。

14.1.2.2.操作

表 16.2. JMX MBean 操作

操作名称	描述
listProperties(pid)	返回配置 pid 的属性列表(property=value formatted)。
deleteProperty (pid, property)	从配置 pid 中删除属性。
appendProperty(pid, property, value)	在配置 pid 的属性值的末尾附加值值。
setProperty(pid, property, value)	为配置 pid 的属性值设置值。
delete (pid)	删除由 pid 标识的配置。
create (pid)	使用 pid 创建一个空（不带任何属性）配置。
update (pid, properties)	使用提供的属性映射更新使用 pid 标识的配置。

16.9. 使用控制台

16.9.1. 可用命令

要在控制台中查看可用命令列表，您可以使用 帮助 ：

```
karaf@root()> help
bundle                    Enter the subshell
```

```
bundle:capabilities             Displays OSGi capabilities of a given bundles.
bundle:classes                  Displays a list of classes/resources contained in the bundle
bundle:diag                     Displays diagnostic information why a bundle is not Active
bundle:dynamic-import           Enables/disables dynamic-import for a given bundle.
bundle:find-class               Locates a specified class in any deployed bundle
bundle:headers                  Displays OSGi headers of a given bundles.
bundle:id                       Gets the bundle ID.
...
```

您有带有简短描述的所有命令列表。

您可以使用 tab 键获取所有命令的快速列表：

```
karaf@root()> Display all 294 possibilities? (y or n)
...
```

16.9.2. subshell 和 completion 模式

命令具有范围和名称。例如，命令 feature:list 具有 scope，并列出 为 name。

2008 年按范围"组"命令。每个范围组成一个子 shell。

您可以使用全限定名称(scope:name)直接执行命令：

```
karaf@root()> feature:list
...
```

或者，在子 shell 中输入命令上下文到子 shell：

```
karaf@root()> feature
karaf@root(feature)> list
```

请注意，您可以通过输入子 shell 名称（此处为 功能）直接输入子 shell。您可以直接从子 shell "switch" 改为另一个：

```
karaf@root()> feature
karaf@root(feature)> bundle
karaf@root(bundle)>
```

提示符显示（）之间的当前子 shell。

exit 命令进入父子 shell：

```
karaf@root()> feature
karaf@root(feature)> exit
karaf@root()>
```

完成模式定义了 tab 键和 help 命令的行为。

您有三种不同的模式可用：

- **全局**
- **第一个**
- **SUBSHELL**

您可以使用 etc/org.apache.karaf.shell.cfg 文件中的 completionMode 属性来定义默认的完成模式。默认情况下，您有：

```
completionMode = GLOBAL
```

您还可以使用 shell:completion 命令更改完成模式"实时"（使用 Karaf shell 控制台）：

```
karaf@root()> shell:completion
GLOBAL
karaf@root()> shell:completion FIRST
karaf@root()> shell:completion
FIRST
```

shell: 补全 可以告知您当前使用的完成模式。您还可以提供您想要的新完成模式。

GLOBAL 完成模式是 Karaf 4.0.0 中的默认模式（主要用于转换目的）。

GLOBAL 模式并不实际使用 subshell：它与之前 Karaf 版本的行为相同。

当您输入 tab 键时，无论您在哪个子 shell 中，都会显示所有命令和所有别名：

```
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)> Display all 273 possibilities? (y or n)
```

FIRST 完成模式是 GLOBAL 完成模式的替代选择。

如果您在 root 级别 subshell 上输入 tab 键，则完成将显示所有子 shell 中的命令和别名（如 GLOBAL 模式）。但是，如果您在子 shell 中键入了 tab 键，则完成将仅显示当前子 shell 的命令：

```
karaf@root()> shell:completion FIRST
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)>
info install list repo-add repo-list repo-remove uninstall version-list
karaf@root(feature)> exit
karaf@root()> log
karaf@root(log)> <TAB>
karaf@root(log)>
clear display exception-display get log set tail
```

SUBSHELL completion 模式是实际的子 shell 模式。

如果您在根级别上输入 tab 键，则完成会显示子 shell 命令（转至子 shell）和全局别名。在子 shell 中后，如果您键入 TAB 密钥，则完成会显示当前子 shell 的命令：

```
karaf@root()> shell:completion SUBSHELL
karaf@root()> <TAB>
karaf@root()>
* bundle cl config dev feature help instance jaas kar la ld lde log log:list man package region service
shell ssh system
```

```
karaf@root()> bundle
karaf@root(bundle)> <TAB>
karaf@root(bundle)>
capabilities classes diag dynamic-import find-class headers info install list refresh requirements
resolve restart services start start-level stop
uninstall update watch
karaf@root(bundle)> exit
karaf@root()> camel
karaf@root(camel)> <TAB>
karaf@root(camel)>
backlog-tracer-dump backlog-tracer-info backlog-tracer-start backlog-tracer-stop context-info
context-list context-start context-stop endpoint-list route-info route-list route-profile route-reset-stats
route-resume route-show route-start route-stop route-suspend
```

16.9.3. Unix 比如环境

Karaf 控制台提供完整的 Unix 环境，如环境。

16.9.3.1. help 或 man

我们已经看到了使用 help 命令来显示所有可用的命令。

但是，您也可以使用 help 命令获取有关命令或 help 命令的 man 命令的详细信息，该命令是 help 命令的别名。您还可以通过在命令中使用 --help 选项使用另一个表单来获取命令帮助。

因此这些命令

```
karaf@root()> help feature:list
karaf@root()> man feature:list
karaf@root()> feature:list --help
```

所有都生成相同的帮助输出：

```
DESCRIPTION
        feature:list

        Lists all existing features available from the defined repositories.

SYNTAX
        feature:list [options]

OPTIONS
        --help
                Display this help message
```

```
    -o, --ordered
            Display a list using alphabetical order
    -i, --installed
            Display a list of all installed features only
    --no-format
            Disable table rendered output
```

16.9.3.2. completion

当您输入 tab 键时，wam 会尝试完成：

- **subshell**
- **commands**
- **别名**
- **命令参数**
- **命令选项**

16.9.3.3. Alias

别名是与给定命令关联的另一个名称。

shell:alias 命令创建一个新的别名。例如，要将 list-installed-features 别名创建到实际 feature:list -i 命令，您可以：

```
karaf@root()> alias "list-features-installed = { feature:list -i }"
karaf@root()> list-features-installed
Name       | Version | Required | State   | Repository     | Description
--------------------------------------------------------------------------------------------------------------------------------
---
feature    | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell      | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
deployer   | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle     | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config     | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
```

```
instance   | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas       | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package    | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service    | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system     | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar        | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh        | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
```

登录时，Apache Karaf 控制台会读取 etc/shell.init.script 文件，您可以在其中创建别名。它与 Unix 中的 bashrc 或 profile 文件类似。

```
ld = { log:display $args } ;
lde = { log:exception-display $args } ;
la = { bundle:list -t 0 $args } ;
ls = { service:list $args } ;
cl = { config:list "(service.pid=$args)" } ;
halt = { system:shutdown -h -f $args } ;
help = { *:help $args | more } ;
man = { help $args } ;
log:list = { log:get ALL } ;
```

您可以在默认情况下看到可用的别名：

- **ld 是显示日志的简短表单（用于 log:display 命令别名）**
- **lde 是显示例外的简短表单（别名为 log:exception-display 命令）**
- **la 是列出所有捆绑包的简短形式（别名到 bundle:list -t 0 命令）**
- **ls 是列出所有服务的简短形式（别名到 service:list 命令）**
- **CL 是一个简短的表单，用于列出所有配置（别名到 config:list 命令）**
- **halt 是关闭 Apache Karaf 的简短形式（针对 system:shutdown -h -f 命令）**

- help 是显示帮助的简短形式（别名为 *:help 命令）
- man 与 help (help 命令别名)相同。
- log:list 显示所有日志记录器和级别（用于 log:get ALL 命令）

您可以在 etc/shell.init.script 文件中创建自己的别名。

16.9.3.4. 密钥绑定

与大多数 Unix 环境一样，Ramon 控制台支持一些关键绑定：

- 在命令历史记录中导航的方向键
- CTRL-D 以注销/关闭 Karaf
- CTRL-R 搜索之前执行的命令
- CTRL-U 删除当前行

16.9.3.5. 管道

您可以将一个命令的输出作为输入传送到另一个命令的输出。它是使用 | 字符的管道：

```
karaf@root()> feature:list |grep -i war
pax-war                   | 4.1.4                    |          | Uninstalled | org.ops4j.pax.web-4.1.4  | Provide
support of a full WebContainer
pax-war-tomcat            | 4.1.4                    |          | Uninstalled | org.ops4j.pax.web-4.1.4  |
war                       | 4.0.0                    |          | Uninstalled | standard-4.0.0           | Turn Karaf as
a full WebContainer
blueprint-web             | 4.0.0                    |          | Uninstalled | standard-4.0.0           | Provides
an OSGI-aware Servlet ContextListener fo
```

16.9.3.6. grep, more, find, …

Karaf 控制台提供一些类似于 Unix 环境的核心命令：

- shell:alias 为现有命令创建一个别名
- shell:cat 显示文件或 URL 的内容
- shell:clear 清除当前控制台显示
- shell:completion 显示或更改当前完成模式
- shell:date 显示当前日期（可选使用格式）
- shell:each 对参数列表执行冲突
- shell:echo echoes 并将参数输出到 stdout
- shell:edit 在当前文件或 URL 中调用文本编辑器
- shell:env 显示或设置 shell 会话变量的值
- shell:exec 执行系统命令
- shell：grep 打印与给定模式匹配的行
- shell:head 显示输入的第一行
- shell:history 打印命令历史记录

- shell：如果 允许您在脚本中使用条件（如果，则是其他块）
- shell:info 打印有关当前 Karaf 实例的各种信息
- shell:java 执行 Java 应用程序
- shell:less file pager
- shell:logout 断开 shell 与当前会话的连接
- shell:more 是一个文件页器
- shell:new 创建一个新的 Java 对象
- shell:printf 格式和打印参数
- shell:sleep sleep 用于一个位，然后唤醒
- shell:sort 将所有文件的排序串联写入 stdout
- shell:source 执行命令
- 当执行命令抛出异常时，shell:stack-traces-print 会在控制台中打印完整的堆栈追踪
- shell:tac 捕获 STDIN 并将其返回为字符串
- shell:tail 显示输入的最后几行

- shell:threads 打印当前线程
- shell:watch 定期执行命令并刷新输出
- shell:wc 会为每个文件打印换行符、字数和字节数
- shell: while loop while the condition 为 true

您不必使用 命令的完全限定名称，只要它是唯一的，就可以直接使用该命令名称。因此您可以使用 'head' 而不是 'shell:head'

同样，您可以使用 help 命令或 --help 选项查找这些命令的详细信息和所有选项。

16.9.3.7. 脚本脚本

Apache Karaf 控制台支持完整的脚本语言，类似于 Unix 上的 bash 或 csh。

每个 (shell:each)命令可以迭代列表：

```
karaf@root()> list = [1 2 3]; each ($list) { echo $it }
1
2
3
```

注意

同一循环可以使用 shell 编写：同时 命令：

```
karaf@root()> a = 0 ; while { %((a+=1) <= 3) } { echo $a }
1
2
3
```

您可以自己创建列表（如上例中所示），或者某些命令也可以返回列表。

我们可以注意到，控制台会创建一个"会话"变量，其中包含您可以使用 $ list 访问的名称列表。

$it 变量是与当前对象对应的隐式值（列表中当前迭代的值）。

使用 [] 创建列表时，Apache Karaf 控制台会创建一个 Java ArrayList。这意味着您可以使用 ArrayList 对象（如实例 get 或 size）中提供的方法：

```
karaf@root()> list = ["Hello" world]; echo ($list get 0) ($list get 1)
Hello world
```

我们可以在此处注意，在对象上调用方法直接使用 （对象方法参数）。这里 ($list get 0) 表示 $list.get (0)，其中 $list 是 ArrayList。

类 表示法将显示有关对象的详情：

```
karaf@root()> $list class
...
ProtectionDomain     ProtectionDomain  null
 null
 <no principals>
 java.security.Permissions@6521c24e (
 ("java.security.AllPermission" "<all permissions>" "<all actions>")
)


Signers              null
SimpleName           ArrayList
TypeParameters       [E]
```

您可以将变量"广播"到指定类型。

```
karaf@root()> ("hello world" toCharArray)
[h, e, l, l, o,  , w, o, r, l, d]
```

如果失败，您将看到 casting 异常：

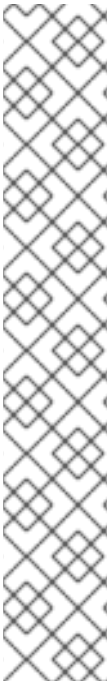
```
karaf@root()> ("hello world" toCharArray)[0]
Error executing command: [C cannot be cast to [Ljava.lang.Object;
```

您可以使用 shell:source 命令"调用"脚本：

```
karaf@root> shell:source script.txt
True!
```

其中 script.txt 包含：

```
foo = "foo"
if { $foo equals "foo" } {
  echo "True!"
}
```



注意

在编写脚本时，空格非常重要。例如，以下脚本不正确：

```
if{ $foo equals "foo" } ...
```

并使用以下内容失败：

```
karaf@root> shell:source script.txt
Error executing command: Cannot coerce echo "true!"() to any of []
```

由于 if 语句后缺少空格。

对于别名，您可以在 etc/shell.init.script 文件中创建初始化脚本。您还可以使用别名命名脚本。实际上，别名只是脚本。

详情请参阅开发人员指南中的脚本部分。

16.9.4. 安全性

Apache Karaf 控制台支持基于角色的访问控制(RBAC)安全机制。这意味着，根据连接到控制台的用户，您可以定义取决于用户的组和角色、执行某些命令的权限或限制参数允许的值。

此用户指南的 Security 部分中详述了 控制台安全性。

16.10. 置备

Apache Karaf 支持使用 Karaf Features 的概念来调配应用程序和模块。

16.10.1. Application

通过配置应用程序，它意味着安装所有模块、配置和传输的应用程序。

16.10.2. OSGi

它原生支持部署 OSGi 应用程序。

OSGi 应用程序是一组 OSGi 捆绑包。OSGi 捆绑包是一个常规 jar 文件，它带有 jar MANIFEST 中的附加元数据。

在 OSGi 中，捆绑包可以依赖于其他捆绑包。因此，这意味着，在大多数时候部署 OSGi 应用程序，您首先需要部署应用程序所需的大量其他捆绑包。

因此，您必须首先找到这些捆绑包，安装捆绑包。同样，这些"独立"捆绑包可能需要其他捆绑包来满足自己的依赖项。

更深入，应用程序还需要配置（请参阅用户指南的 [Configuration section|configuration]）。因此，在启动应用程序之前，除了依赖项捆绑包外，您必须创建或部署配置。

正如我们所见，应用程序的调配可能非常长且快速。

16.10.3. 功能和解析器

Apache Karaf 为调配应用提供了简单而灵活的方式。

在 Apache Karaf 中，应用调配是 Apache Karaf "功能"。

功能将应用程序描述为：

- 名称
- 版本
- 可选的描述（通常带有长描述）
- 一组捆绑包
- （可选）设置配置或配置文件
- （可选）依赖项功能

安装功能时，Apache Karaf 将安装功能中描述的所有资源。这意味着，它将自动解析和安装功能中描述的所有捆绑包、配置和依赖项功能。

功能解析器检查服务要求，并安装提供符合要求的服务的捆绑包。默认模式只对"新风格"功能启用此行为（基本来说，带有等于或大于 1.3.0 的架构仓库 XML）。它不适用于"旧的"功能存储库（从 Karaf 2 或 3 开始）。

您可以使用 serviceRequirements 属性更改 etc/org.apache.karaf.features.cfg 文件中的服务要求强制模式。

```
serviceRequirements=default
```

可能的值有：

- Disable：完全忽略服务要求，对于"旧样式"和"新样式"功能存储库

- 默认：对于"旧风格"功能存储库，会忽略服务要求，并为"新风格"功能存储库启用。

- enforce：始终验证服务要求，了解"旧样式"和"新样式"功能存储库。

此外，功能也可以定义要求。在这种情况下，2019 年可以自动额外的捆绑包或功能，提供满足要求的功能。

功能具有完整的生命周期：install、start、stop、update、uninstall。

16.10.4. 功能软件仓库

该功能在 XML 描述符中描述。此 XML 文件包含一组功能的描述。

功能 XML 描述符被命名为 "features repository"。在能够安装功能前，您必须注册提供该功能的功能存储库（使用 feature:repo-add 命令或 FeatureMBean ），如稍后所述。

例如，以下 XML 文件（或"features 存储库"）描述了 feature1 和 feature2 功能：

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.3.0">
  <feature name="feature1" version="1.0.0">
    <bundle>...</bundle>
    <bundle>...</bundle>
  </feature>
  <feature name="feature2" version="1.1.0">
    <feature>feature1</feature>
    <bundle>...</bundle>
  </feature>
</features>
```

我们可以注意到，功能 XML 具有模式。详情请参阅 用户指南的 [Features XML Schema 部分|provisioning-schema]。feature1 功能在 1.0.0 版本中可用，包含两个捆绑包。& lt;bundle/& gt; 元素包含捆绑包工件的 URL （请参阅 [Artifacts repositories 和 URL 部分|urls] 部分）。如果您安装 feature1 功能（使用 feature:install 或 FeatureMBean），则 Apache Karaf 将自动安装上述两个捆绑包。feature2 功能在 1.1.0 版本中提供，包含对 feature1 功能和捆绑包的引用。& lt;feature/& gt; 元素包含功能的名称。可以使用 version 属性到 <feature/&gt ; 元素(& lt;feature version="1.0.0">feature1</feature>)来定义特定的功能版本。如果没有指定 version 属性，则 Apache Karaf 将安装最新的可用版本。如果您安装 feature2 功能（使用 feature:install 或 FeatureMBean），则 Apache Karaf 将自动安装 feature1 （如果尚未安装）和捆绑包。

功能存储库使用到功能 XML 文件的 URL 注册。

features 状态存储在 Apache Karaf 缓存中（在 KARAF_DATA 文件夹中）。您可以重新启动 Apache Karaf，之前安装的功能仍然保持安装并在重新启动后可用。如果您进行干净的重启，或者删除 Apache Karaf 缓存（删除 KARAF_DATA 文件夹），则之前注册的软件仓库和功能都将丢失：您必须注册功能并再次安装功能。要防止这种情况，您可以将功能指定为引导功能。

16.10.5. 引导特性

您可以将一些功能描述为引导功能。Apache Karaf 将自动安装引导功能，即使之前还没有使用 feature:install 或 FeatureMBean 安装。

Apache Karaf 功能配置位于 etc/org.apache.karaf.features.cfg 配置文件中。

此配置文件包含用于定义引导功能的两个属性：

- featuresRepositories 包含功能存储库（功能 XML）URL 的列表（任意列表）。
- featuresBoot 包含要在启动时安装的功能列表（用逗号分开）。

16.10.6. 升级功能

您可以通过安装相同的功能（具有相同的 SNAPSHOT 版本或不同的版本）来更新发行版本。

由于功能生命周期，您可以控制功能的状态（启动、停止等）。

您还可以使用模拟来查看更新的作用。

16.10.7. overrides

功能中定义的捆绑包可以使用文件 etc/overrides.properties 覆盖。文件中的每一行都定义一个覆盖。语法为：<bundle-uri>[;range="[min,max）"]，如果覆盖的版本大于覆盖捆绑包的版本，则给定的捆绑包将使用相同的符号名称覆盖所有捆绑包定义中。如果未指定范围，则假定为微版本级别的兼容性。

例如，覆盖 mvn:org.ops4j.pax.logging/pax-logging-service/1.8.5 会过度使用 pax-logging-service 1.8.3，而不是 1.8.6 或 1.7.0。

16.10.8. 功能捆绑包

16.10.8.1. 开始级别

默认情况下，功能部署的捆绑包将具有与 karaf.startlevel.bundle 属性中 etc/config.properties 配置文件中定义的值的开始级别。

这个值可以被 < bundle /> 元素的 start-level 属性 "overrided"，在 features XML 中被 "overrided"。

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

start-level 属性确保在使用该捆绑包的捆绑包之前启动 myproject-dao 捆绑包。

更好的解决方法是，使用 start-level 只是通过定义您需要的软件包或服务来了解您的依赖项。它比设置启动级别更强大。

16.10.8.2. 模拟、启动和停止

您可以使用 -t 选项在 feature:install 命令模拟安装功能。

您可以在不启动捆绑包的情况下安装捆绑包。默认情况下，功能中的捆绑包会自动启动。

功能可以指定捆绑包不应自动启动（捆绑包处于解析状态）。要做到这一点，功能可以在 < bundle /> 项中将 start 属性指定为 false ：

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

16.10.8.3. 依赖项

捆绑包可以在 <bundle/> 元素上使用 dependency 属性设置为 true。

此信息可供解析器用于计算要安装的捆绑包的完整列表。

16.10.9. 依赖功能

功能可以依赖于一组其他功能：

```
<feature name="my-project" version="1.0.0">
  <feature>other</feature>
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

将安装 my-project 功能时，也会自动安装 其他功能。

可以为依赖功能定义版本范围：

```
<feature name="spring-dm">
  <feature version="[2.5.6,4)">spring</feature>
  ...
</feature>
```

将安装范围内最高版本的功能。

如果指定了单个版本，则范围将被视为开放。

如果未指定，将安装可用最高的可用。

要指定准确的版本，请使用关闭的范围，如 [3.1,3.1]。

16.10.9.1. 功能先决条件

先决条件功能是特殊的依赖项。如果您要向依赖功能标签添加 prerequisites 属性，那么它将在安装

实际功能前强制安装，并在依赖功能中激活捆绑包。如果给定功能中的捆绑包没有使用预安装的 URL（如 wrap 或 war ）时，这可能很方便。

16.10.10. 功能配置

功能 XML 中的 <config /> 元素允许功能创建和/或填充配置（由配置 PID 识别）。

```
<config name="com.foo.bar">
  myProperty = myValue
</config>
```

< config/> 元素的 name 属性对应于配置 PID （详情请参阅 [Configuration section|configuration] ）。

功能的安装效果与在 etc 文件夹中丢弃名为 com.foo.bar.cfg 的文件具有相同的效果。

< config/> 元素的内容是一组属性，遵循 key=value 标准。

16.10.11. 功能配置文件

功能可以指定 &lt ;config file/& gt; 元素，而不使用 <config/> 元素。

```
<configfile finalname="/etc/myfile.cfg" override="false">URL</configfile>
```

< config/> 元素不直接操作 Apache Karaf 配置层（如使用 < config/> 元素时），<configfile/> 元素直接取一个由 URL 指定的文件，并将该文件复制到 finalname 属性指定的位置。

如果没有指定，则位置来自 KARAF_BASE 变量。也可以使用诸如 ${karaf.home}、${karaf.base}、${karaf.etc} 或系统属性等变量。

例如：

```
<configfile finalname="${karaf.etc}/myfile.cfg" override="false">URL</configfile>
```

如果该文件已存在于所需的位置，并且跳过了配置文件部署，因为已存在的文件可能包含自定义。此行为可以通过覆盖设置为 true 来覆盖。

文件 URL 是 Apache Karaf 支持的 URL （有关详细信息，请参阅用户指南的 [Artifacts 存储库和 URL|urls]）。

16.10.11.1. 要求

功能也可以指定预期的要求。功能解析器将尝试满足要求。为此，它会检查功能和捆绑包功能，并自动安装捆绑包来满足要求。

例如，功能可以包含：

```
<requirement>osgi.ee;filter:=&amp;quot;(&amp;amp;(osgi.ee=JavaSE)(!(version&amp;gt;=1.8)))&amp;quot;
</requirement>
```

要求指定只有在 JDK 版本不是 1.8 时，该功能才能正常工作（因此基本上为 1.7）。

功能解析器也可以在满足可选依赖项时刷新捆绑包，从而重写可选的导入。

16.10.12. 命令

16.10.12.1. feature:repo-list

feature:repo-list 命令列出所有注册的功能存储库：

```
karaf@root()> feature:repo-list
Repository              | URL
-------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

每个存储库都有一个名称和功能 XML 的 URL。

当您注册功能存储库 URL （使用 feature:repo-add 命令或 FeatureMBean ）时，Apache Karaf 会解析功能 XML。如果要强制 Apache Karaf 重新加载功能存储库 URL （以及更新功能定义），您可以使

用 **-r** 选项：

```
karaf@root()> feature:repo-list -r
Reloading all repositories from their urls

Repository              | URL
-------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

16.10.12.2. feature:repo-add

要注册功能存储库（因此有 Apache Karaf 中的新功能），您必须使用 feature:repo-add 命令。

feature:repo-add 命令需要 name/url 参数。这个参数接受：

- 功能存储库 URL。它是功能 XML 文件的 URL。支持用户指南的 [Artifacts 仓库和 URL 部分|urls] 中描述的 URL。
- etc/org.apache.karaf.features.repos.cfg 配置文件中定义的功能存储库名称。

etc/org.apache.karaf.features.repos.cfg 定义"pre-installed/available"功能存储库的列表：

```
################################################################################
#
#   Licensed to the Apache Software Foundation (ASF) under one or more
#   contributor license agreements.  See the NOTICE file distributed with
#   this work for additional information regarding copyright ownership.
#   The ASF licenses this file to You under the Apache License, Version 2.0
#   (the "License"); you may not use this file except in compliance with
#   the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
#
################################################################################
```

```
#
# This file describes the features repository URL
# It could be directly installed using feature:repo-add command
#
enterprise=mvn:org.apache.karaf.features/enterprise/LATEST/xml/features
spring=mvn:org.apache.karaf.features/spring/LATEST/xml/features
cellar=mvn:org.apache.karaf.cellar/apache-karaf-cellar/LATEST/xml/features
cave=mvn:org.apache.karaf.cave/apache-karaf-cave/LATEST/xml/features
camel=mvn:org.apache.camel.karaf/apache-camel/LATEST/xml/features
camel-extras=mvn:org.apache-extras.camel-extra.karaf/camel-extra/LATEST/xml/features
cxf=mvn:org.apache.cxf.karaf/apache-cxf/LATEST/xml/features
cxf-dosgi=mvn:org.apache.cxf.dosgi/cxf-dosgi/LATEST/xml/features
cxf-xkms=mvn:org.apache.cxf.services.xkms/cxf-services-xkms-features/LATEST/xml
activemq=mvn:org.apache.activemq/activemq-karaf/LATEST/xml/features
jclouds=mvn:org.apache.jclouds.karaf/jclouds-karaf/LATEST/xml/features
openejb=mvn:org.apache.openejb/openejb-feature/LATEST/xml/features
wicket=mvn:org.ops4j.pax.wicket/features/LATEST/xml/features
hawtio=mvn:io.hawt/hawtio-karaf/LATEST/xml/features
pax-cdi=mvn:org.ops4j.pax.cdi/pax-cdi-features/LATEST/xml/features
pax-jdbc=mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
pax-jpa=mvn:org.ops4j.pax.jpa/pax-jpa-features/LATEST/xml/features
pax-web=mvn:org.ops4j.pax.web/pax-web-features/LATEST/xml/features
pax-wicket=mvn:org.ops4j.pax.wicket/pax-wicket-features/LATEST/xml/features
ecf=http://download.eclipse.org/rt/ecf/latest/site.p2/karaf-features.xml
decanter=mvn:org.apache.karaf.decanter/apache-karaf-decanter/LATEST/xml/features
```

您可以直接向 feature:repo-add 命令提供一个 features 存储库名称。要安装 PAX JDBC，您可以：

```
karaf@root()> feature:repo-add pax-jdbc
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

当您不提供可选 版本 参数时，Apache Karaf 将安装最新版本的功能存储库。您可以使用 version 参数指定 目标版本 ：

```
karaf@root()> feature:repo-add pax-jdbc 1.3.0
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

除了提供 etc/org.apache.karaf.features.repos.cfg 配置文件中定义的功能存储库名称外，您可以直接向 feature:repo-add 命令提供功能存储库 URL：

```
karaf@root()> feature:repo-add mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

默认情况下，feature:repo-add 命令只注册 features 存储库，它不会安装任何功能。如果您指定了 -i 选项，feature:repo-add 命令会注册 features 存储库并安装此功能存储库中描述的所有功能：

```
karaf@root()> feature:repo-add -i pax-jdbc
```

16.10.12.3. feature:repo-refresh

当您注册时，Apache Karaf 会解析功能存储库 XML （使用 feature:repo-add 命令或 FeatureMBean）。如果 features 存储库 XML 发生变化，您必须指示 Apache Karaf 刷新 features 存储库以加载更改。

feature:repo-refresh 命令刷新 features 存储库。

如果没有参数，命令会刷新所有功能存储库：

```
karaf@root()> feature:repo-refresh
Refreshing feature url mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
Refreshing feature url mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

您可以通过提供 URL 或 features 存储库名称（以及可选版本）来指定要刷新的所有功能存储库，而不是刷新所有功能存储库：

```
karaf@root()> feature:repo-refresh mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
```

```
karaf@root()> feature:repo-refresh pax-jdbc
Refreshing feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

16.10.12.4. feature:repo-remove

feature:repo-remove 命令从注册的功能存储库中删除。

feature:repo-remove 命令需要一个参数：

- features 存储库名称（如 feature:repo-list 命令的输出的仓库列中显示）
- features 存储库 URL （如 feature:repo-list 命令的输出的 URL 列中显示）

```
karaf@root()> feature:repo-remove org.ops4j.pax.jdbc-1.3.0
```

```
karaf@root()> feature:repo-remove mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

默认情况下，feature:repo-remove 命令只从注册的功能仓库中删除 features 存储库：它不会卸载 features 存储库提供的功能。

如果使用 -u 选项，feature:repo-remove 命令会卸载 features 存储库描述的所有功能：

```
karaf@root()> feature:repo-remove -u org.ops4j.pax.jdbc-1.3.0
```

16.10.12.5. feature:list

feature:list 命令列出所有可用的功能（由不同的注册的功能存储库提供）：

```
Name                        | Version                     | Required | State       | Repository              |
Description
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
pax-cdi                     | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1                 | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.2                 | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
pax-cdi-weld                | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Weld
CDI support
pax-cdi-1.1-weld            | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2-weld            | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.2 support
pax-cdi-openwebbeans        | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
OpenWebBeans CDI support
pax-cdi-web                 | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI support
pax-cdi-1.1-web             | 0.12.0                      |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
...
```

如果您希望按字母顺序对功能进行排序，您可以使用 -o 选项：

```
karaf@root()> feature:list -o
Name                        | Version                     | Required | State       | Repository              |
Description
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
```

```
deltaspike-core              | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike core support
deltaspike-data              | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike data support
deltaspike-jpa               | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike jpa support
deltaspike-partial-bean      | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike partial bean support
pax-cdi                      | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1                  | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.1-web              | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
pax-cdi-1.1-web-weld         | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld Web CDI 1.1 support
pax-cdi-1.1-weld             | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2                  | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
...
```

默认情况下，feature:list 命令显示所有功能，无论其当前状态（已安装或未安装）。

使用 -i 选项仅显示安装的功能：

```
karaf@root()> feature:list -i
Name            | Version | Required | State   | Repository     | Description
--------------------------------------------------------------------------------------------------------------------
aries-proxy     | 4.0.0   |          | Started | standard-4.0.0 | Aries Proxy
aries-blueprint | 4.0.0   | x        | Started | standard-4.0.0 | Aries Blueprint
feature         | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell           | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
shell-compat    | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell Compatibility
deployer        | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle          | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config          | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic      | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
instance        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas            | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log             | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package         | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service         | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system          | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar             | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh             | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management      | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
wrap            | 0.0.0   | x        | Started | standard-4.0.0 | Wrap URL handler
```

16.10.12.6. feature:install

feature:install 命令安装一项功能。

它需要 feature 参数。feature 参数是功能的名称，或功能的名称/版本。如果只提供该功能的名称（不是版本），则会安装最新的可用版本。

```
karaf@root()> feature:install eventadmin
```

我们可以使用 -t 或 --simulate 选项模拟安装：它只是显示它的作用，但它不做：

```
karaf@root()> feature:install -t -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
  Managing bundle:
    org.apache.felix.metatype / 1.0.12
```

您可以指定要安装的功能版本：

```
karaf@root()> feature:install eventadmin/4.0.0
```

默认情况下，feature:install 命令没有详细。如果要对 feature:install 命令执行的操作有一些详情，您可以使用 -v 选项：

```
karaf@root()> feature:install -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

如果功能包含已安装的捆绑包，默认情况下，Apache Karaf 将刷新此捆绑包。有时，这个刷新可能会导致其他正在运行的应用程序出现问题。如果要禁用已安装的捆绑包的自动刷新，您可以使用 -r 选项：

```
karaf@root()> feature:install -v -r eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

您可以使用 -s 或 --no-auto-start 选项决定不启动由功能安装的捆绑包：

```
karaf@root()> feature:install -s eventadmin
```

16.10.12.7. feature:start

默认情况下，当您安装某个功能时，它会自动安装。但是，您可以在 feature:install 命令中指定 -s 选项。

安装功能（启动或不启动）后，该功能中定义的所有软件包都将可用，并可用于其他捆绑包中的 wiring。

启动功能时，会启动所有捆绑包，因此该功能也会公开相关的服务。

16.10.12.8. feature:stop

您还可以停止某个功能：这意味着该功能提供的所有服务都将停止并从服务 registry 中删除。但是，软件包仍可用于 wiring （捆绑包处于解析状态）。

16.10.12.9. feature:uninstall

feature:uninstall 命令卸载功能。作为 feature:install 命令，feature:uninstall 命令需要 feature 参数。feature 参数是功能的名称，或功能的名称/版本。如果只提供该功能的名称（不是版本），则会安装最新的可用版本。

```
karaf@root()> feature:uninstall eventadmin
```

解析器在功能卸载过程中涉及：如果未由其他功能使用，可以卸载由卸载的功能安装的功能。

16.10.13. deployer

您可以通过在 deploy 文件夹中直接丢弃该文件来"热部署"功能 XML。

Apache Karaf 提供功能部署器。

当您丢弃 deploy 文件夹中的功能 XML 时，功能部署器将执行：* 将功能 XML 注册为功能存储库，并将 install 属性设置为 "auto" 的功能由部署器自动安装。

例如，丢弃部署文件夹中的以下 XML 将自动安装 feature1 和 feature2，而 feature3 不会被安装：

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="my-features" xmlns="http://karaf.apache.org/xmlns/features/v1.3.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.3.0
http://karaf.apache.org/xmlns/features/v1.3.0">

   <feature name="feature1" version="1.0" install="auto">
      ...
   </feature>

   <feature name="feature2" version="1.0" install="auto">
      ...
   </feature>

   <feature name="feature3" version="1.0">
      ...
   </feature>

</features>
```

16.10.14. JMX FeatureMBean

在 JMX 层上，您有一个 MBean 专用于管理功能和特性存储库： FeatureMBean。

FeatureMBean 对象名称是： org.apache.karaf:type=feature,name iwl.

16.10.14.1. 属性

FeatureMBean 提供两个属性：

- 功能是 所有可用功能的表格式数据集。
- 存储库 是所有注册的功能存储库的表格式数据集。

Repositories 属性提供以下信息：

- name 是 features 存储库的名称。
- URI 是此存储库功能 XML 的 URI。

- 功能是 此功能存储库提供的所有功能（名称和版本）的表格式数据集。
- 存储库 是此功能存储库中"导入"的功能集合。

Features 属性提供以下信息：

- name 是功能的名称。
- version 是功能的版本。
- installed 是一个布尔值。如果为 true，这表示当前已安装该功能。
- 捆绑包 是功能中描述的所有捆绑包（捆绑包 URL）的表格式数据集。
- 配置 是功能中描述的所有配置的表格式数据集。
- 配置文件是 功能中描述的所有配置文件的表格数据集。
- 依赖项 是功能中描述的所有依赖功能的表格式数据集。

16.10.14.2. 操作

- addRepository (url) 使用 url 添加 features 存储库。url 可以是 feature:repo-add 命令中的 名称。
- addRepository (url, install) 使用 url 添加 features 存储库，并在 install 为 true 时自动安装所有捆绑包。url 可以是 feature:repo-add 命令中的 名称。
- removeRepository (url) 使用 url 删除 features 存储库。url 可以是 feature:repo-remove 命令中的 名称。

- installFeature (name) 安装名为 的功能。
- installFeature (name, version) 使用名称和版本 安装该功能。
- installFeature (name, noClean, noRefresh) 使用名称安装功能，而不在失败时清理捆绑包，而不刷新已安装的捆绑包。
- installFeature (name, version, noClean, noRefresh) " install the feature with the 'name and version ）' 在失败时没有清理捆绑包，且不会刷新已安装的捆绑包。
- uninstallFeature (name) 使用名称 卸载功能。
- uninstallFeature (name, version) 使用名称和版本 卸载该功能。

16.10.14.3. 通知

FeatureMBean 发送两种类型的通知（您可以订阅并响应这些通知）：

- 当功能存储库更改时（添加或删除）。
- 当功能更改时（安装或卸载）。

第 17 章 使用远程连接管理容器

使用本地控制台管理容器并非始终有意义。Red Hat Fuse 有多种远程管理容器的方法。您可以使用远程容器的命令控制台或启动远程客户端。

17.1. 为远程访问配置容器

17.1.1. 概述

当您以默认模式或 第 2.1.3 节 “在服务器模式中启动运行时” 中启动 Red Hat Fuse 运行时，它允许通过 SSH 从任何其他 Fuse 控制台访问的远程控制台。远程控制台提供本地控制台的所有功能，并允许远程用户对容器及其内部运行的服务进行完全控制。

注意

当在 第 2.1.4 节 “在客户端模式中启动运行时” 中运行时，Fuse 运行时会禁用远程控制台。

17.1.2. 配置独立容器以进行远程访问

SSH 主机名和端口号在 INSTALL_DIR/etc/org.apache.karaf.shell.cfg 配置文件中配置。更改远程访问的端口 显示更改用于 8102 端口的示例配置。

更改远程访问的端口

```
sshPort=8102
sshHost=0.0.0.0
```

17.2. 远程连接和断开连接

有两种方法可以连接到远程容器。如果您已经运行 Red Hat Fuse 命令 shell，您必须调用 console 命令来连接到远程容器。或者，您可以在命令行中直接运行实用程序来连接到远程容器。

17.2.1. 从远程容器连接到独立容器

17.2.1.1. 概述

任何容器的命令控制台都可用于访问远程容器。使用 SSH 时，本地容器的控制台作为远程容器的命令控制台连接到远程容器，并作为命令控制台连接。

17.2.1.2. 使用 ssh:ssh console 命令

您可以使用 ssh:ssh console 命令连接到远程容器的控制台。

ssh:ssh 命令语法

```
ssh:ssh -l username -P password -p port hostname
```

-l

用于连接到远程容器的用户名。使用具有 admin 特权的有效 JAAS 登录凭证。

-P

用于连接到远程容器的密码。

-p

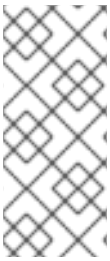
用于访问所需容器远程控制台的 SSH 端口。默认值为 8101。有关更改端口号的详情，请查看 第 17.1.2 节 “配置独立容器以进行远程访问”。

hostname

运行远程容器的机器的主机名。有关更改主机名的详情，请查看 第 17.1.2 节 “配置独立容器以进行远程访问”。

警告

我们建议您在 etc/users.properties 文件中自定义用户名和密码。



注意

如果您的远程容器部署在 用于 SPARC 实例的 Oracle VM Server 上，则默认的 SSH 端口值 8101 已被逻辑卷管理器守护进程占用。在这种情况下，您需要重新配置容器的 SSH 端口，如 第 17.1.2 节 “配置独立容器以进行远程访问” 所述。

要确认您已连接到正确的容器，请在 Karaf 控制台提示符处键入 shell:info，这将返回有关当前连接实例的信息。

17.2.1.3. 断开与远程控制台的连接

要断开与远程控制台的连接，请在提示符后输入 logout 或按 Ctrl+D。

您将与远程容器断开连接，控制台将再次管理本地容器。

17.2.2. 使用客户端命令行实用程序连接到容器

17.2.2.1. 使用远程客户端

远程客户端允许您安全地连接到远程 Red Hat Fuse 容器，而无需在本地启动完整的 Fuse 容器。

例如，要快速连接到在同一台机器上以服务器模式运行的 Fuse 实例，请打开命令提示符并运行 client[.bat] 脚本（位于 InstallDir/bin 目录中），如下所示：

```
client
```

通常，您可以提供一个主机名、端口、用户名和密码来连接远程实例。如果您在更大的脚本中使用客户端，例如在测试套件中，您可以附加控制台命令，如下所示：

```
client -a 8101 -h hostname -u username -p password shell:info
```

或者，如果您省略 -p 选项，会提示您输入密码。

对于独立容器，请使用具有 admin 特权的任何有效的 JAAS 用户凭据。

要显示客户端的可用选项，请输入：

```
client --help
```

Karaf 客户端帮助

```
Apache Felix Karaf client
  -a [port]     specify the port to connect to
  -h [host]     specify the host to connect to
  -u [user]     specify the user name
  -p [password] specify the password
  --help        shows this help message
  -v            raise verbosity
  -r [attempts] retry connection establishment (up to attempts times)
  -d [delay]    intra-retry delay (defaults to 2 seconds)
  [commands]    commands to run
If no commands are specified, the client will be put in an interactive mode
```

17.2.2.2. 远程客户端默认凭证

您可能会意外地发现您可以使用 bin/client 登录 Karaf 容器，而无需提供任何凭据。这是因为远程客户端程序已预先配置为使用默认凭证。如果没有指定凭证，远程客户端会自动尝试使用以下默认凭证（按顺序）：

- 默认 SSH 密钥 criu-criutries，以使用默认的 Apache Karaf SSH 密钥进行登录。在 etc/keys.properties 文件中默认注释掉允许此登录的对应配置条目。
- 使用admin / admin组合使用用户名和密码登录的默认用户名/密码凭证 。在 etc/users.properties 文件中默认注释掉允许此登录的对应配置条目。

因此，如果您在 Karaf 容器中创建新用户，只需在 users.properties 中取消注释默认的 admin / admin 凭据，您会发现 bin/client 实用程序可以在不提供凭证的情况下登录。

重要

为安全起见，Fuse 在首次安装 Karaf 容器时禁用了默认凭据（通过注释）。但是，如果您简单地取消注释这些默认凭证，而不更改 默认密码或 SSH 公钥，您将在 Karaf 容器中打开一个安全漏洞。不得 在生产环境中执行此操作。如果发现，您可以在不提供凭证的情况下使用 bin/client 登录 容器，这表明您的容器不安全，您必须执行相应的步骤来在生产环境中修复此问题。

17.2.2.3. 断开与远程客户端控制台的连接

如果您使用远程客户端打开远程控制台，而不是使用它传递命令，则需要与其断开连接。要断开与远程客户端的控制台的连接，请在提示符后输入 logout 或按 Ctrl-D。

客户端将断开连接并退出。

17.2.3. 使用 SSH 命令行工具连接到容器

17.2.3.1. 概述

您还可以使用 ssh 命令行工具（类似 UNIX 的操作系统的标准实用程序）登录到红帽 Fuse 容器，其中身份验证机制基于公钥加密（必须首先在容器中安装公钥）。例如，如果容器被配置为侦听 TCP 端口 8101，您可以按如下方式登录：

```
ssh -p 8101 jdoe@localhost
```

重要

目前仅在独立容器上支持基于密钥的登录，不支持 Fabric 容器。

17.2.3.2. 先决条件

要使用基于密钥的 SSH 登录，必须满足以下先决条件：

- 容器必须独立（不支持 Fabric），并且安装了 PublickeyLoginModule。
- 您必须已创建了 SSH 密钥对（请参阅 第 17.2.3.4 节 “创建新的 SSH 密钥对”）。
-

您必须从 SSH 密钥对安装公钥到容器中（请参阅 第 17.2.3.5 节 “在容器中安装 SSH 公钥”）。

17.2.3.3. 默认密钥位置

ssh 命令会在默认密钥位置自动查找私钥。建议您在默认位置安装密钥，因为它会显式指定位置的问题。

在 *NIX 操作系统中，RSA 密钥对的默认位置为：

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

在 Windows 操作系统中，RSA 密钥对的默认位置是：

```
C:\Documents and Settings\Username\.ssh\id_rsa
C:\Documents and Settings\Username\.ssh\id_rsa.pub
```

注意

Red Hat Fuse 仅支持 RSA 密钥。DSA 密钥 不起作用。

17.2.3.4. 创建新的 SSH 密钥对

使用 ssh-keygen 工具生成 RSA 密钥对。打开新命令提示符并输入以下命令：

```
ssh-keygen -t rsa -b 2048
```

前面的命令生成 RSA 密钥，密钥长度为 2048 位。然后，系统会提示您为密钥对指定文件名：

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/Username/.ssh/id_rsa):
```

键入 return 将密钥对保存到默认位置。然后会提示您输入密码短语：

```
Enter passphrase (empty for no passphrase):
```

您可以选择在此处输入 pass phrase，或者键入 return 两次以选择 no pass phrase。

注意

如果您要使用相同的密钥对来运行 Fabric 控制台命令，建议您不要选择 不通过短语，因为 Fabric 不支持使用加密的私钥。

17.2.3.5. 在容器中安装 SSH 公钥

要使用 SSH 密钥对登录到红帽 JBoss Fuse 容器，您必须在 INSTALL_DIR/etc/keys.properties 文件中创建新用户条目，在容器中安装 SSH 公钥。此文件中的每个用户条目都出现在一行中，格式为：

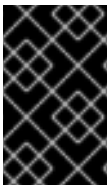
```
Username=PublicKey,Role1,Role2,...
```

例如，如果您的公钥文件 ~/.ssh/id_rsa.pub 具有以下内容：

```
ssh-rsa
AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2
NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskI8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx
jdoe@doemachine.local
```

您可以通过在 InstallDir/etc/keys.properties 文件中添加以下条目，创建具有 admin 角色的 jdoe 用户（在一行中）：

```
jdoe=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskI8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,g:admingroup
```



重要

不要在此处插入 id_rsa.pub 文件的整个内容。插入代表公钥本身的符号块。

17.2.3.6. 支持检查公钥身份验证

启动容器后，您可以通过运行 jaas:realms 控制台命令来检查是否支持公钥身份验证，如下所示：

```
karaf@root()> jaas:realms
Index │ Realm Name │ Login Module Class Name
──────┼────────────┼──────────────────────────────────────────────────────────────
──────────────-
1 │ karaf │ org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2 │ karaf │ org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
3 │ karaf │ org.apache.karaf.jaas.modules.audit.FileAuditLoginModule
4 │ karaf │ org.apache.karaf.jaas.modules.audit.LogAuditLoginModule
5 │ karaf │ org.apache.karaf.jaas.modules.audit.EventAdminAuditLoginModule
karaf@root()>
```

您应该会看到已安装了 PublickeyLoginModule。使用这个配置，您可以使用用户名/密码凭证或公钥凭证登录到容器。

17.2.3.7. 将 ssh 角色添加到 etc/keys.properties

etc/keys.properties 中定义的 admingroup 必须包含 ssh 角色，如下例所示：

```
#
# For security reason, the default auto-signed key is disabled.
# The user guide describes how to generate/update the key.
#
#karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIl
H7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E
+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9
QAAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6
EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuW
fBpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPlcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

如果 admingroup 的定义中没有 ssh 角色，您必须编辑 etc/keys.properties 并添加 ssh 角色。

17.2.3.8. 使用基于密钥的 SSH 登录

现在，您可以使用基于密钥的 SSH 工具登录到容器。例如：

```
$ ssh -p 8101 jdoe@localhost
_____          _   _   _       _     _______
|  _ \ ___  __| | | | | | ___ _| |_  |  ___|   _ ___  ___
| |_) / _ \/ _` | | |_| |/ _ | __| | |_ | | | / __|/ _ \
|  _ <  __/ (_| | |  _  | (_| | |_  |  _|| |_| \__ \  __/
|_| \_\___|\__,_| |_| |_|\__,_|\__| |_|   \__,_|___/___|

  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```

注意

如果您使用加密的私钥，ssh 实用程序将提示您输入密语。

17.3. 停止远程容器

如果您使用 ssh:ssh 命令或远程客户端连接到远程控制台，您可以使用 osgi:shutdown 命令停止远程实例。

注意

在远程控制台中按 Ctrl+D 只需关闭远程连接并返回到本地 shell 即可。

第 18 章 使用 MAVEN 构建

摘要

Maven 是一个开源构建系统，可从 Apache Maven 项目获取。本章介绍了一些基本的 Maven 概念，并描述了如何设置 Maven 以使用 Red Hat Fuse。在原则上，您可以使用任何构建系统来构建 OSGi 捆绑包。但强烈建议 Maven，因为 Red Hat Fuse 得到良好支持。

18.1. MAVEN 目录结构

18.1.1. 概述

Maven 构建系统最重要的原则之一是 Maven 项目中所有文件的标准位置。这个原则有几个优点。一个优点是 Maven 项目通常具有相同的目录布局，因此可以轻松地在项目中找到文件。另一个优点是，与 Maven 集成的各种工具需要 几乎不需要 初始配置。例如，Java 编译器知道它应当编译 src/main/java 下的所有源文件，并将结果置于 target/classes 中。

18.1.2. 标准目录布局

例 18.1 “标准 Maven 目录布局” 显示与构建 OSGi 捆绑包项目相关的标准 Maven 目录布局的元素。此外，还显示了蓝图配置文件(不是由 Maven 定义)的标准位置。

例 18.1. 标准 Maven 目录布局

```
ProjectDir/
  pom.xml
  src/
    main/
      java/
        ...
      resources/
        META-INF/

        OSGI-INF/
          blueprint/
            *.xml
    test/
      java/
      resources/
  target/
    ...
```

注意

可以覆盖标准目录布局，但不建议 在 Maven 中这样做。

18.1.3. pom.xml 文件

pom.xml 文件是当前项目的项目对象模型(POM)，其中包含如何构建当前项目的完整描述。pom.xml 文件可以完全自包含，但通常（对于更复杂的 Maven 项目而言），它可以 *从父 POM* 文件导入设置。

构建项目后，在生成的 JAR 文件中的以下位置自动嵌入 pom.xml 文件的副本：

```
META-INF/maven/groupId/artifactId/pom.xml
```

18.1.4. src 和目标目录

src/ 目录包含您在开发项目时工作的所有代码和资源文件。

target/ 目录包含构建的结果（通常是 JAR 文件），以及构建期间生成的所有中间文件。例如，在执行构建后，target/classes/ 目录将包含资源文件的副本以及已编译的 Java 类。

18.1.5. 主和测试目录

src/main/ 目录包含构建工件所需的所有代码和资源。

src/test/ 目录包含针对编译的工件运行单元测试的所有代码和资源。

18.1.6. java directory

每个 java/ 子目录都包含带有标准 Java 目录布局（即目录路径名称的 Java 源代码）的 Java 源代码 (.java 文件)，其中目录名会镜像 Java 软件包名，其 / 代替 . 字符。src/main/java/ 目录包含捆绑包源代码，而 src/test/java/ 目录包含单元测试源代码。

18.1.7. 资源目录

如果您有捆绑包中包含的任何配置文件、数据文件或 Java 属性，则这些配置文件应放在 src/main/resources/ 目录下。src/main/resources/ 下的文件和目录将复制到 Maven 构建过程生成的

JAR 文件的根目录中。

src/test/resources/ 下的文件仅在测试阶段使用，且不会 复制到生成的 JAR 文件中。

18.1.8. 蓝图容器

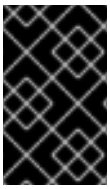
OSGi R4.2 定义蓝图容器。Red Hat Fuse 对 Blueprint 容器内置了支持，只需在项目中包含 Blueprint 配置文件 OSGI-INF/blueprintAttr.xml 即可启用这些容器。有关 Blueprint 容器的详情，请参考 第 12 章 *OSGi 服务*。

18.2. APACHE KARAF 的 BOM 文件

Maven Bill of Materials (BOM) 文件的目的是提供一组策展的 Maven 依赖项版本，这些版本可以很好地协同工作，从而为您为每个 Maven 工件单独定义版本。

Apache Karaf 的 Fuse BOM 提供以下优点：

- 定义 Maven 依赖项的版本，以便在将依赖项添加到 POM 时不需要指定版本。
- 定义一组对特定版本的 Fuse 经过全面测试并支持的策展依赖关系。
- 简化 Fuse 的升级。



重要

红帽仅支持由 Fuse BOM 定义的一组依赖项。

要将 Maven BOM 文件合并到 Maven 项目中，请在项目的 pom.xml 文件中指定 dependencies Management 元素（或者，在父 POM 文件中），如下例所示：

```
<?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>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>

  </properties>

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

注意

org.jboss.redhat-fuse BOM 是 Fuse 7 中的新功能，旨在简化 BOM 版本控制。Fuse Quickstart 和 Maven archetypes 仍然使用旧的 BOM 样式，因为它们尚未重构为使用新的 BOM。这两个 BOM 都正确，您可以在 Maven 项目中使用一个。在即将发布的 Fuse 版本中，快速入门和 Maven 构架类型将重构为使用新的 BOM。

使用依赖项管理机制指定 BOM 后，可以在不指定工件版本 *的情况下* 向 POM 添加 Maven 依赖项。例如，要为 camel-velocity 组件添加依赖项，您可以在 POM 中的 dependencies 元素中添加以下 XML 片段：

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
</dependency>
```

注意如何在这个依赖项定义中省略 version 元素。

第 19 章 MAVEN INDEXER 插件

Maven 插件需要 Maven Indexer 插件，以便它快速搜索 Maven Central 以查找工件。

要部署 Maven Indexer 插件，请使用以下命令：

先决条件

在部署 Maven Indexer 插件前，请确保已遵循在 Apache Karaf 上准备 使用 Maven 部分的"安装"中的说明。

部署 Maven Indexer 插件

1. 进入 Karaf 控制台，输入以下命令来安装 Maven Indexer 插件：

```
features:install hawtio-maven-indexer
```

2. 输入以下命令配置 Maven Indexer 插件：

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories 'https://maven.oracle.com'
config:proplist
config:update
```

3. 等待 Maven Indexer 插件部署。这可能需要几分钟时间。请查看下面显示的消息，以便在日志标签页中显示。

INFO	org.apache.felix.fileinstall	Creating configuration from io.hawt.maven.indexer.cfg
INFO	io.fabric8.internal.ProfileServiceImpl	updateProfile: Profile[ver=1.0,id=fabric,atts={parents=karaf hawtio}]
INFO	io.fabric8.internal.ProfileServiceImpl	updateProfile: Profile[ver=1.0,id=fabric,atts={parents=karaf hawtio}]

部署 Maven Indexer 插件后，使用以下命令将其他外部 Maven 存储库添加到 Maven Indexer 插件配置中：

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories external repository
config:proplist
config:update
```

第 20 章 LOG

Apache Karaf 提供动态、强大的日志记录系统。

它支持：

- OSGi 日志服务
- Apache Log4j v1 和 v2 框架
- Apache Commons Logging 框架
- Logback 框架
- SLF4J 框架
- 原生 Java Util Logging 框架

这意味着应用可以使用任何日志记录框架，Apache Karaf 将使用中央日志系统来管理日志记录器、附加程序等。

20.1. 配置文件

初始日志配置从 etc/org.ops4j.pax.logging.cfg 加载。

此文件是 标准 Log4j2 配置文件。

找到不同的 Log4j2 元素：

- loggers

- **appenders**
- **布局**

您可以在 文件中直接添加自己的初始配置。

默认配置如下：

```
#
#  Copyright 2005-2018 Red Hat, Inc.
#
#  Red Hat licenses this file to you under the Apache License, version
#  2.0 (the "License"); you may not use this file except in compliance
#  with the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
#  implied.  See the License for the specific language governing
#  permissions and limitations under the License.
#

#
# Internal Log4j2 configuration
#
log4j2.status = WARN
log4j2.verbose = false
log4j2.dest = out

#
# Common pattern layouts for appenders defined as reusable properties
# See https://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout
# references will be replaced by felix.fileinstall
#
log4j2.pattern = %d{DEFAULT} | %-5.5p | %-20.20t | %-32.32c{1.} | %X{bundle.id} -
%X{bundle.name} - %X{bundle.version} | %m%n
#log4j2.pattern = %d{DEFAULT} %-5.5p {%t} [%C.%M()] (%F:%L) : %m%n

#
# Appenders configuration
#

# JDBC Appender
log4j2.appender.jdbc.type = JDBC
log4j2.appender.jdbc.name = JdbcAppender
log4j2.appender.jdbc.tableName = EVENTS
```

```
log4j2.appender.jdbc.cs.type = DataSource
log4j2.appender.jdbc.cs.lazy = true
log4j2.appender.jdbc.cs.jndiName = osgi:service/jdbc/logdb
log4j2.appender.jdbc.c1.type = Column
log4j2.appender.jdbc.c1.name = DATE
log4j2.appender.jdbc.c1.isEventTimestamp = true
log4j2.appender.jdbc.c2.type = Column
log4j2.appender.jdbc.c2.name = LEVEL
log4j2.appender.jdbc.c2.pattern = %level
log4j2.appender.jdbc.c2.isUnicode = false
log4j2.appender.jdbc.c3.type = Column
log4j2.appender.jdbc.c3.name = SOURCE
log4j2.appender.jdbc.c3.pattern = %logger
log4j2.appender.jdbc.c3.isUnicode = false
log4j2.appender.jdbc.c4.type = Column
log4j2.appender.jdbc.c4.name = THREAD_ID
log4j2.appender.jdbc.c4.pattern = %thread
log4j2.appender.jdbc.c4.isUnicode = false
log4j2.appender.jdbc.c5.type = Column
log4j2.appender.jdbc.c5.name = MESSAGE
log4j2.appender.jdbc.c5.pattern = %message
log4j2.appender.jdbc.c5.isUnicode = false


# Console appender not used by default (see log4j2.rootLogger.appenderRefs)
log4j2.appender.console.type = Console
log4j2.appender.console.name = Console
log4j2.appender.console.layout.type = PatternLayout
log4j2.appender.console.layout.pattern = ${log4j2.pattern}

# Rolling file appender
log4j2.appender.rolling.type = RollingRandomAccessFile
log4j2.appender.rolling.name = RollingFile
log4j2.appender.rolling.fileName = ${karaf.data}/log/fuse.log
log4j2.appender.rolling.filePattern = ${karaf.data}/log/fuse-%i.log.gz
# uncomment to not force a disk flush
#log4j2.appender.rolling.immediateFlush = false
log4j2.appender.rolling.append = true
log4j2.appender.rolling.layout.type = PatternLayout
log4j2.appender.rolling.layout.pattern = ${log4j2.pattern}
log4j2.appender.rolling.policies.type = Policies
log4j2.appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.rolling.policies.size.size = 16MB
log4j2.appender.rolling.strategy.type = DefaultRolloverStrategy
log4j2.appender.rolling.strategy.max = 20

# Audit file appender
log4j2.appender.audit.type = RollingRandomAccessFile
log4j2.appender.audit.name = AuditRollingFile
log4j2.appender.audit.fileName = ${karaf.data}/security/audit.log
log4j2.appender.audit.filePattern = ${karaf.data}/security/audit.log.%i
log4j2.appender.audit.append = true
log4j2.appender.audit.layout.type = PatternLayout
log4j2.appender.audit.layout.pattern = ${log4j2.pattern}
log4j2.appender.audit.policies.type = Policies
log4j2.appender.audit.policies.size.type = SizeBasedTriggeringPolicy
```

```
log4j2.appender.audit.policies.size.size = 8MB

# OSGi appender
log4j2.appender.osgi.type = PaxOsgi
log4j2.appender.osgi.name = PaxOsgi
log4j2.appender.osgi.filter = *

#
# Loggers configuration
#

# Root logger
log4j2.rootLogger.level = INFO
log4j2.rootLogger.appenderRef.RollingFile.ref = RollingFile
log4j2.rootLogger.appenderRef.PaxOsgi.ref = PaxOsgi
log4j2.rootLogger.appenderRef.Console.ref = Console
log4j2.rootLogger.appenderRef.Console.filter.threshold.type = ThresholdFilter
log4j2.rootLogger.appenderRef.Console.filter.threshold.level = ${karaf.log.console:-OFF}
#log4j2.rootLogger.appenderRef.Sift.ref = Routing

# Spifly logger
log4j2.logger.spifly.name = org.apache.aries.spifly
log4j2.logger.spifly.level = WARN

# Security audit logger
log4j2.logger.audit.name = org.apache.karaf.jaas.modules.audit
log4j2.logger.audit.level = INFO
log4j2.logger.audit.additivity = false
log4j2.logger.audit.appenderRef.AuditRollingFile.ref = AuditRollingFile

# help with identification of maven-related problems with pax-url-aether
#log4j2.logger.aether.name = shaded.org.eclipse.aether
#log4j2.logger.aether.level = TRACE
#log4j2.logger.http-headers.name = shaded.org.apache.http.headers
#log4j2.logger.http-headers.level = DEBUG
#log4j2.logger.maven.name = org.ops4j.pax.url.mvn
#log4j2.logger.maven.level = TRACE
```

默认配置使用 out 文件 appender 定义 ROOT 日志记录器，其 INFO 日志级别。您可以将日志级别更改为任何 Log4j2 有效值。从最详细到最详细的信息，您可以指定 TRACE, DEBUG, INFO, ERROR, 或 FATAL。

OSGi appender

osgi:* 附加程序是一个特殊的附加程序，用于将日志消息发送到 OSGi 日志服务。

stdout appender

stdout 控制台附加程序是预先配置的，但没有默认启用。此附加程序允许您在标准输出中直接显示日志消息。如果您计划在服务器模式下运行 Apache Karaf （不带控制台），这是值得注意的。

要启用它，您必须在 rootLogger 中添加 stdout 附加程序：

```
log4j2.rootLogger=INFO, out, stdout, osgi:*
```

out appender

out appender 是默认值。它是维护并轮转 10 1MB 日志文件的滚动文件。日志文件默认位于 data/log/fuse.log 中。

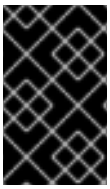
sift appender

默认情况下，sift appender 不会被启用。此附加器允许您为每个部署的捆绑包有一个日志文件。默认情况下，日志文件名称格式使用捆绑包符号名称（在 data/log 文件夹中）。您可以在运行时编辑此文件。Apache Karaf 重新加载文件，并且更改生效。您不需要重新启动 Apache Karaf。Apache Karaf 使用另一个配置文件，即 etc/org.apache.karaf.log.cfg。此文件配置由日志命令使用的日志服务（请参阅后文）。

JDBC appender

jdbc appender 有一个 lazy 标志，当 true (*启用*)时，如果数据源不可用，则不会将日志记录添加到数据库中。但是，当 jndi 时，数据源或连接会返回，日志会重启。

```
log4j2.appender.jdbc.cs.lazy = true
```



重要

如果要避免丢失日志记录消息，我们还会重新配置紧急附加程序。

20.2. 命令

Apache Karaf 不是更改 etc/org.ops4j.pax.logging.cfg 文件，而是提供一组命令来动态更改日志配置并查看日志内容：

20.2.1. log:clear

log:clear 命令清除日志条目。

20.2.2. log:display

log:display 命令显示日志条目。

默认情况下，它显示 rootLogger 的日志条目：

```
karaf@root()> log:display
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
2015-07-01 19:12:47,368 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
```

您还可以使用 logger 参数显示来自特定日志记录器 的日志条目 ：

```
karaf@root()> log:display ssh
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
```

默认情况下，所有日志条目都将显示。如果 Apache Karaf 容器自很长时间以来正在运行，则可能比较长。您可以使用 -n 选项限制要显示的条目数：

```
karaf@root()> log:display -n 5
2015-07-01 06:53:24,143 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.BundleStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=bundleState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.PackageStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=packageState,version=1.5,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.ServiceStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=serviceState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,152 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering
org.osgi.jmx.framework.wiring.BundleWiringStateMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=wiringState,version=1.1,framework=org.apache.felix.framework,uuid=5335370f-9dee-
449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,501 | INFO  | FelixStartLevel  | RegionsPersistenceImpl           | 78 -
org.apache.karaf.region.persist - 4.0.0 | Loading region digraph persistence
```

您还可以使用 etc/org.apache.karaf.log.cfg 文件中的 size 属性限制存储和保留的条目数量：

```
#
# The number of log statements to be displayed using log:display. It also defines the number
# of lines searched for exceptions using log:display exception. You can override this value
```

```
# at runtime using -n in log:display.
#
size = 500
```

默认情况下，每个日志级别都显示不同的颜色： ERROR/FATAL are in red, DEBUG in purple, INFO in cyan 等。您可以使用 --no-color 选项禁用颜色。

日志条目格式模式不使用 etc/org.ops4j.pax.logging.cfg 文件中定义的转换模式。默认情况下，它使用 etc/org.apache.karaf.log.cfg 中定义的 pattern 属性。

```
#
# The pattern used to format the log statement when using log:display. This pattern is according
# to the log4j2 layout. You can override this parameter at runtime using log:display with -p.
#
pattern = %d{ISO8601} | %-5.5p | %-16.16t | %-32.32c{1} | %X{bundle.id} - %X{bundle.name} -
%X{bundle.version} | %m%n
```

您还可以使用 -p 选项动态更改模式（对于一个执行）：

```
karaf@root()> log:display -p "\%d - \%c - \%m\%n"
2015-07-01 07:01:58,007 - org.apache.sshd.common.util.SecurityUtils - BouncyCastle not registered,
using the default JCE provider
2015-07-01 07:01:58,725 - org.apache.aries.jmx.core - Starting JMX OSGi agent
2015-07-01 07:01:58,744 - org.apache.aries.jmx.core - Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83] for service with service.id [13]
2015-07-01 07:01:58,747 - org.apache.aries.jmx.core - Registering
org.osgi.jmx.service.cm.ConfigurationAdminMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83
```

模式是一个常规的 Log4j2 模式，您可以使用关键字，如 %d 作为日期，%c 用于类，%m 用于日志消息等。

20.2.3. log:exception-display

log:exception-display 命令显示最后发生的异常。

对于 log:display 命令，log:exception-display 命令默认使用 rootLogger，但您可以使用 logger 参数指定 日志记录器。

20.2.4. log:get

log:get 命令显示日志记录器的当前日志级别。

默认情况下，显示的日志级别是根日志记录器中的日志级别：

```
karaf@root()> log:get
Logger                          | Level
--------------------------------------------------------------------------------+-------
ROOT                            | INFO
org.apache.aries.spifly         | WARN
org.apache.karaf.jaas.modules.audit | INFO
org.apache.sshd                 | INFO
```

您可以使用 logger 参数指定特定的 日志记录器 ：

```
karaf@root()> log:get ssh
INFO
```

logger 参数接受 ALL 关键字，以显示所有日志记录器的日志级别（作为列表）。

例如，如果您在 etc/org.ops4j.pax.logging.cfg 文件中定义了自己的日志记录器，如下所示：

```
log4j2.logger.my.name = MyLogger
log4j2.logger.my.level = DEBUG
```

您可以查看带有对应日志级别的日志记录器列表：

```
karaf@root()> log:get ALL
Logger                          | Level
--------------------------------------------------------------------------------+-------
MyLogger                        | DEBUG
ROOT                            | INFO
org.apache.aries.spifly         | WARN
org.apache.karaf.jaas.modules.audit | INFO
org.apache.sshd                 | INFO
```

log:list 命令是 log:get ALL 的别名。

20.2.5. log:log

log:log 命令允许您在日志中手动添加消息。创建 Apache Karaf 脚本时很有趣：

```
karaf@root()> log:log "Hello World"
karaf@root()> log:display
12:55:21.706 INFO [pipe-log:log "Hello World"] Hello World
```

默认情况下，日志级别为 INFO，但您可以使用 -l 选项指定不同的日志级别：

```
karaf@root()> log:clear
karaf@root()> log:log -l ERROR "Hello World"
karaf@root()> log:display
12:55:41.460 ERROR [pipe-log:log "Hello World"] Hello World
```

20.2.6. log:set

log:set 命令设置日志记录器的日志级别。

默认情况下，它会更改 rootLogger 的日志级别：

```
karaf@root()> log:set DEBUG
karaf@root()> log:get
Logger                    | Level
--------------------------+-------
ROOT                      | DEBUG
...
```

您可以在 级别 后面使用 logger 参数指定特定的日志记录器：

```
karaf@root()> log:set INFO my.logger
karaf@root()> log:get my.logger
Logger    | Level
----------------
my.logger | INFO
```

level 参数接受任何 Log4j2 日志级别：TRACE, DEBUG, INFO, WARN, ERROR, FATAL。

通过它，它还接受 DEFAULT 特殊关键字。

DEFAULT 关键字的目的是删除日志记录器的当前级别（并且仅级别，其他属性（如 appender ）不

会被删除，以使用日志记录器父级级别（日志记录器是分级）。

例如，您定义了以下日志记录器（在 etc/org.ops4j.pax.logging.cfg 文件中）：

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=DEBUG,appender2
```

您可以更改 my.logger.custom 日志记录器的级别：

```
karaf@root()> log:set INFO my.logger.custom
```

现在，我们有：

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=INFO,appender2
```

您可以使用 my.logger.custom 日志记录器上的 DEFAULT 关键字来删除级别：

```
karaf@root()> log:set DEFAULT my.logger.custom
```

现在，我们有：

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=appender2
```

这意味着，在运行时，my.logger.custom 日志记录器使用其父 my.logger 的级别，因此 INFO。

现在，如果我们将 DEFAULT 关键字与 my.logger 日志记录器一起使用：

```
karaf@root()> log:set DEFAULT my.logger
```

我们有：

```
rootLogger=INFO,out,osgi:*
my.logger=appender1
my.logger.custom=appender2
```

因此，my.logger.custom 和 my.logger 使用父 rootLogger 的日志级别。

无法将 DEFAULT 关键字与 rootLogger 一起使用，且没有父项。

20.2.7. log:tail

log:tail 与 log:display 完全相同，但它持续显示日志条目。

您可以使用与 log:display 命令相同的选项和参数。

默认情况下，它显示 rootLogger 中的条目：

```
karaf@root()> log:tail
2015-07-01 07:40:28,152 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.9.0 | BouncyCastle not registered, using the default JCE provider
2015-07-01 07:40:28,909 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
2015-07-01 07:40:28,928 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b] for service with service.id [13]
2015-07-01 07:40:28,936 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.service.cm.ConfigurationAdminMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b
```

要从 log:tail 命令退出，只需键入 CTRL-C。

20.3. JMX LOGMBEAN

您可以使用 Log MBean 命令执行的所有操作。

LogMBean 对象名称是 org.apache.karaf:type=log,name iwl。

20.3.1. 属性

- level 属性是 ROOT 日志记录器的级别。

20.3.2. 操作

- getLevel (logger) 以获取特定日志记录器的日志级别。由于此操作支持 ALL 关键字，它会返回一个带有各个日志记录器级别的 Map。

- setLevel (level, logger)，以设置特定日志记录器的日志级别。此操作支持 log:set 命令的 DEFAULT 关键字。

20.4. 高级配置

20.4.1. SIFT 日志

fuse-Karaf 提供了示例（默认情况下为 out）Log4j2 sift appender 配置，并使用 $FUSE_HOME/etc/org.ops4j.pax.logging.cfg 文件中的这个附件程序：

```
# Sift appender
log4j2.appender.mdc.type = Routing
log4j2.appender.mdc.name = SiftAppender
log4j2.appender.mdc.routes.type = Routes
# see: http://logging.apache.org/log4j/2.x/manual/appenders.html#Routes
log4j2.appender.mdc.routes.pattern = $\\{ctx:bundle.name}
log4j2.appender.mdc.routes.sift.type = Route
log4j2.appender.mdc.routes.sift.appender.type = RollingRandomAccessFile
log4j2.appender.mdc.routes.sift.appender.name = RollingFile
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:bundle.name}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:bundle.name}-
%i.log.gz
log4j2.appender.mdc.routes.sift.appender.append = true
log4j2.appender.mdc.routes.sift.appender.layout.type = PatternLayout
log4j2.appender.mdc.routes.sift.appender.layout.pattern = ${log4j2.pattern}
log4j2.appender.mdc.routes.sift.appender.policies.type = Policies
log4j2.appender.mdc.routes.sift.appender.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.mdc.routes.sift.appender.policies.size.size = 16MB
log4j2.appender.mdc.routes.sift.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.mdc.routes.sift.appender.strategy.max = 20
...
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

该配置在 http://logging.apache.org/log4j/2.x/manual/appenders.html#RoutingAppender中进行了描述

SIFT/Routing 附加器的模式属性是可用于区分用于记录的目标位置。

可用的查找有多种，如下所述 ：http://logging.apache.org/log4j/2.x/manual/lookups.html

最重要的查找是 ctx，它会在 ThreadContext 映射(a.k.a)中查找值（键）。MDC）。

Fuse Karaf 提供的默认配置使用 ctx:bundle.name 作为模式，这意味着：

```
lookup bundle.name key in MDC
```

bundle. 前缀的密钥由 pax-logging 本身提供，有 3 个不同的值可供选择：

- bundle.name == org.osgi.framework.Bundle.getSymbolicName()
- bundle.id == org.osgi.framework.Bundle.getBundleId()
- bundle.version == org.osgi.framework.Bundle.getVersion().toString()

但是，如果 Camel 上下文是使用 MDC 支持的 Camel 上下文（蓝色打印 XML DSL）：

```
<camelContext id="my-context" xmlns="http://camel.apache.org/schema/blueprint"
useMDCLogging="true">
```

在 MDC/ThreadContext 中，我还有更多可用的密钥，然后在 SIFT appender 配置中可用作模式：

- camel.exchangeId - Exchange id
- camel.messageId - 消息 id

- camel.correlationId - 如果交换的关联 ID （如果其关联）。例如，Splitter EIP 中的子消息
- camel.transactionKey - 转换交换的事务 ID。请注意，id 并不唯一，而是其事务模板的 id，用于标记给定事务的事务边界。因此，我们决定将关键的 transactionKey 而不是 transactionID 命名来指出这一事实。
- camel.routeId - 路由的 id，其中交换当前正在路由
- camel.breadcrumbId - 用于跨传输跟踪消息的唯一 id。
- camel.contextId - 用于从不同 camel 上下文跟踪消息的 camel 上下文 id。

请参阅 https://people.apache.org/~dkulp/camel/mdc-logging.html

例如，要通过 Camel 的路由 ID 区分日志目标文件，请使用：

```
log4j2.appender.mdc.routes.pattern = $\\{ctx:camel.routeId}
...
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}-
%i.log.gz
```

更重要的是 - 单独附加程序配置不够 - 您必须将其附加到一些日志记录器。同样，示例配置包含：

```
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

（请注意，log4j2.logger.example.appenderRef.SiftAppender.ref 属性的值应与 appender 配置中的 log4j2.appender.mdc.name 的值匹配。

在这里，org.apache.camel 是一个日志记录器名称（或类别名称）。这与 Camel log: 端点中使用的值完全相同。因此，如果您在 Camel 路由中：

```
<to uri="log:org.apache.camel" />
```

日志记录将正常工作。

另一个工作配置是：

```
<to uri="log:my-special-logger" />
```

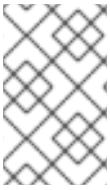
和：

```
log4j2.logger.example.name = my-special-logger
log4j2.logger.example.level = DEBUG
log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

20.4.2. 过滤器

您可以将过滤器应用到附加程序。过滤器评估每个日志事件，并确定是否将其发送到日志。

Log4j2 提供可供使用过滤器使用。



注意

有关这些内容的综合视图，请参阅 Log4J 站点的 过滤器。

20.4.3. 嵌套附加器

嵌套的附加程序是一个特殊的附加程序，您可以使用"inside"另一个附加程序。它允许您在附加器链之间创建某种类型的"routing"。

最常用的"嵌套合规"附加器是：

- AsyncAppender (org.apache.log4j2.AsyncAppender)异步日志事件。此附加程序收集事件并将其分配给附加到它的所有附加者。
- RewriteAppender (org.apache.log4j2.rewrite.RewriteAppender)在可能重写日志事件后

将日志事件转发到另一个附加程序。

这个附加程序接受 appender 定义中的 appenders 属性：

```
log4j2.appender.[appender-name].appenders=[comma-separated-list-of-appender-names]
```

例如，您可以创建一个名为 async 的 AsyncAppender，并将日志事件异步分配给 JMS 附加器：

```
log4j2.appender.async=org.apache.log4j2.AsyncAppender
log4j2.appender.async.appenders=jms

log4j2.appender.jms=org.apache.log4j2.net.JMSAppender
...
```

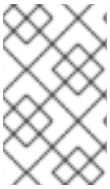
20.4.4. 错误处理程序

有时，附加程序可能会失败。例如，RollingFileAppender 会尝试写入文件系统，但文件系统已满，或者 JMS 附加程序会尝试发送消息，但 JMS 代理不可用。

日志记录可能是关键的，因此务必要知道日志附加程序是否失败。

每个日志附加程序都可以将错误处理委托给错误处理程序，从而有机会对附加错误做出反应。

- FailoverAppender (org.apache.log4j2.varia.FailoverAppender)允许次要附加程序在主附加程序失败时接管。错误消息会在 System.err 上打印，并记录在次要附加器中。



注意

有关 FailoverAppender 的更多信息，请访问 Log4j2 的 Apppender Page。

您可以使用 appender 定义本身上的 errorhandler 属性定义要用于每个 appender 的错误处理程序：

```
log4j2.appender.[appender-name].errorhandler=[error-handler-class]
log4j2.appender.[appender-name].errorhandler.root-ref=[true|false]
log4j2.appender.[appender-name].errorhandler.logger-ref=[logger-ref]
log4j2.appender.[appender-name].errorhandler.appender-ref=[appender-ref]
```

20.4.5. 特定于 OSGi 的 MDC 属性

路由 附加程序是一种面向 OSGi 的附加程序，允许您根据 MDC （映射诊断上下文）属性分割日志事件。

MDC 允许您区分不同的日志事件来源。

路由 附加程序默认提供面向 OSGi 的 MDC 属性：

- bundle.id 是捆绑包 ID
- bundle.name 是捆绑包符号名称
- bundle.version 是捆绑包版本

您可以使用这些 MDC 属性为每个捆绑包创建日志文件：

```
log4j2.appender.routing.type = Routing
log4j2.appender.routing.name = Routing
log4j2.appender.routing.routes.type = Routes
log4j2.appender.routing.routes.pattern = $$\\{ctx:bundle.name\\}
log4j2.appender.routing.routes.bundle.type = Route
log4j2.appender.routing.routes.bundle.appender.type = RollingRandomAccessFile
log4j2.appender.routing.routes.bundle.appender.name = Bundle-$\\{ctx:bundle.name\}
log4j2.appender.routing.routes.bundle.appender.fileName = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log
log4j2.appender.routing.routes.bundle.appender.filePattern = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log.%d{yyyy-MM-dd}
log4j2.appender.routing.routes.bundle.appender.append = true
log4j2.appender.routing.routes.bundle.appender.layout.type = PatternLayout
log4j2.appender.routing.routes.bundle.appender.policies.type = Policies
log4j2.appender.routing.routes.bundle.appender.policies.time.type = TimeBasedTriggeringPolicy
log4j2.appender.routing.routes.bundle.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.routing.routes.bundle.appender.strategy.max = 31

log4j2.rootLogger.appenderRef.Routing.ref = Routing
```

20.4.6. 增强的 OSGi 堆栈追踪器

默认情况下，Apache Karaf 提供特殊的堆栈跟踪呈现器，添加一些 OSGi 特定信息。

在堆栈追踪中，除了引发异常的类外，您可以在每个堆栈追踪行的末尾找到模式 [id:name:version]，其中：

- id 是捆绑包 ID
- name 是捆绑包名称
- version 是捆绑包版本

诊断问题来源非常有帮助。

例如，在以下 IllegalArgumentException 堆栈追踪中，我们可以看到有关例外来源的 OSGi 详情：

```
java.lang.IllegalArgumentException: Command not found:  *:foo
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:225)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.executeStatement(Closure.java:162)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Pipe.run(Pipe.java:101)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:79)
[21:org.apache.karaf.shell.console:4.0.0]
 at
org.apache.felix.gogo.runtime.shell.CommandSessionImpl.execute(CommandSessionImpl.java:71)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.karaf.shell.console.jline.Console.run(Console.java:169)
[21:org.apache.karaf.shell.console:4.0.0]
 at java.lang.Thread.run(Thread.java:637)[:1.7.0_21]
```

20.4.7. 自定义附加器

您可以在 Apache Karaf 中使用您自己的附加程序。

执行此操作的最简单方法是将您的附加程序打包为 OSGi 捆绑包，并将其附加为 org.ops4j.pax.logging.pax-logging-service 捆绑包的片段。

例如，您可以创建 MyAppender ：

```
public class MyAppender extends AppenderSkeleton {
...
}
```

您编译和打包为包含 MANIFEST 的 OSGi 捆绑包，如下所示：

```
Manifest:
Bundle-SymbolicName: org.mydomain.myappender
Fragment-Host: org.ops4j.pax.logging.pax-logging-service
...
```

在 Apache Karaf 系统 文件夹中复制您的捆绑包。系统 文件夹使用标准 Maven 目录布局：groupId/artifactId/version。

在 etc/startup.properties 配置文件中，您可以在 pax-logging-service 捆绑包前在列表中定义捆绑包。

您必须通过干净运行（提升 数据 文件夹）来重新启动 Apache Karaf，才能重新加载系统捆绑包。现在，您可以在 etc/org.ops4j.pax.logging.cfg 配置文件中直接使用您的 appender。

第 21 章 安全性

Apache Karaf 提供高级且灵活的安全系统，由 JAAS (Java 身份验证和授权服务)驱动。

它提供动态安全系统。

Apache Karaf 安全框架在内部用来控制对以下的访问：

- OSGi 服务（开发人员指南中所述）
- 控制台命令
- JMX 层
- WebConsole

您的应用程序也可以使用安全框架（请参阅开发人员指南了解详细信息）。

21.1. REALMS

Apache Karaf 可以管理多个域。realm 包含用于此域中身份验证和/或授权的登录模块的定义。登录模块定义域的身份验证和授权。

jaas:realm-list 命令列出当前定义的域：

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
---------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

您可以看到 Apache Karaf 提供了名为 karaf 的默认域。

这个域有两个登录模块：

- PropertiesLoginModule 使用 etc/users.properties 文件作为用户、组、角色和密码的后端。此登录模块验证用户并返回用户角色。
- PublickeyLoginModule 特别供 SSHd 使用。它使用 etc/keys.properties 文件。此文件包含用户以及与每个用户关联的公钥。

Apache Karaf 提供额外的登录模块（请参阅开发人员指南），了解详细信息：

- JDBCLoginModule 使用数据库作为后端
- LDAPLoginModule 使用 LDAP 服务器作为后端
- SyncopeLoginModule 使用 Apache Syncope 作为后端
- OsgiConfigLoginModule 使用配置作为后端
- Krb5LoginModule 使用 Kerberos 服务器作为后端
- GSSAPILdapLoginModule 使用 LDAP 服务器作为后端，但将 LDAP 服务器身份验证委派给其他后端（通常为 Krb5LoginModule）

您可以使用 jaas:realm-manage 命令管理现有域、登录模块或创建自己的域。

21.1.1. 用户、组、角色和密码

正如我们所看到的那样，Apache Karaf 使用 PropertiesLoginModule。

此登录模块使用 etc/users.properties 文件作为用户、组、角色和密码的存储。

initial etc/users.properties 文件包含：

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# This file contains the users, groups, and roles.
# Each line has to be of the format:
#
# USER=PASSWORD,ROLE1,ROLE2,...
# USER=PASSWORD,_g_:GROUP,...
# _g_\:GROUP=ROLE1,ROLE2,...
#
# All users, grousp, and roles entered in this file are available after Karaf startup
# and modifiable via the JAAS command group. These users reside in a JAAS domain
# with the name "karaf".
#
karaf = karaf,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer
```

我们可以在此文件中看到，默认情况下我们有一个用户：karaf。默认密码是 karaf。

karaf 用户是一个组的成员： admingroup。

组始终以 *g* 前缀：没有此前缀的条目是一个用户。

组定义了一组角色。默认情况下，admin group 定义组、admin、manager 和 viewer 角色。

这意味着 karaf 用户将具有由 admingroup 定义的角色。

21.1.1.1. 命令

jaas:* 命令在控制台中管理域、用户、组、角色。

21.1.1.1.1. jaas:realm-list

本节之前我们已使用过 jaas:realm-list。

jaas:realm-list 命令列出每个域的 realm 和登录模块：

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
-------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

我们这里有一个域(karaf)，其中包含两个登录模块(PropertiesLoginModule 和 PublickeyLoginModule)。

索引 供 jaas:realm-manage 命令用来轻松识别我们要管理的 realm/login 模块。

21.1.1.1.2. jaas:realm-manage

realm/login 模块编辑模式中的 jaas:realm-manage 命令切换，您可以在其中管理登录模块中的用户、组和角色。

要识别要管理的 realm 和 login 模块，您可以使用 --index 选项。索引由 jaas:realm-list 命令显示：

```
karaf@root()> jaas:realm-manage --index 1
```

另一种方法是使用 --realm 和 --module 选项。--realm 选项需要 realm 名称，-- module 选项需要登录模块类名称：

```
karaf@root()> jaas:realm-manage --realm karaf --module
org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
```

21.1.1.1.3. jaas:user-list

当您处于编辑模式时，您可以使用 jaas:user-list 列出登录模块中的用户：

```
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

您可以根据角色查看用户名和组。

21.1.1.1.4. jaas:user-add

jaas:user-add 命令会在当前编辑的登录模块中添加新用户（和密码）：

```
karaf@root()> jaas:user-add foo bar
```

要"提交"您的更改（在用户添加的地方），您必须执行 jaas:update 命令：

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
foo       |            |
```

另一方面，如果要回滚用户添加，您可以使用 jaas:cancel 命令。

21.1.1.1.5. jaas:user-delete

jaas:user-delete 命令从当前编辑的登录模块中删除用户：

```
karaf@root()> jaas:user-delete foo
```

与 jaas:user-add 命令类似，您必须使用 jaas:update 来提交您的更改（或 jaas:cancel 要回滚）：

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

21.1.1.1.6. jaas:group-add

jaas:group-add 命令会在当前编辑的登录模块中为用户分配组（最终创建组）：

```
karaf@root()> jaas:group-add karaf mygroup
```

21.1.1.1.7. jaas:group-delete

jaas:group-delete 命令从当前编辑的登录模块中的组中删除用户：

```
karaf@root()> jaas:group-delete karaf mygroup
```

21.1.1.1.8. jaas:group-role-add

jaas:group-role-add 命令在当前编辑的登录模块的组中添加了一个角色：

```
karaf@root()> jaas:group-role-add mygroup myrole
```

21.1.1.1.9. jaas:group-role-delete

jaas:group-role-delete 命令从当前编辑的登录模块中的组中删除角色：

```
karaf@root()> jaas:group-role-delete mygroup myrole
```

21.1.1.1.10. jaas:update

jaas:update 命令会在登录模块后端提交您的更改。例如，如果是 PropertiesLoginModule，则 etc/users.properties 仅在执行 jaas:update 命令后更新。

21.1.1.1.11. jaas:cancel

jaas:cancel 命令回滚您的更改，且不更新登录模块后端。

21.1.2. 密码加密

默认情况下，密码以明文形式存储在 etc/users.properties 文件中。

可以在 etc/org.apache.karaf.jaas.cfg 配置文件中启用加密：

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# Boolean enabling / disabling encrypted passwords
#
encryption.enabled = false

#
# Encryption Service name
#   the default one is 'basic'
#   a more powerful one named 'jasypt' is available
#       when installing the encryption feature
#
encryption.name =

#
# Encryption prefix
#
encryption.prefix = {CRYPT}

#
# Encryption suffix
#
```

```
encryption.suffix = {CRYPT}

#
# Set the encryption algorithm to use in Karaf JAAS login module
# Supported encryption algorithms follow:
#   MD2
#   MD5
#   SHA-1
#   SHA-256
#   SHA-384
#   SHA-512
#
encryption.algorithm = MD5

#
# Encoding of the encrypted password.
# Can be:
#   hexadecimal
#   base64
#
encryption.encoding = hexadecimal
```

如果 encryption.enabled y 设为 true，则启用密码加密。

启用加密后，密码在第一次登录时加密。加密的密码的前缀为 \{CRYPT\}。要重新加密密码，您可以以明文形式重置密码（在 etc/users.properties 文件中），而无需 \{CRYPT\} 前缀和后缀。Apache Karaf 将检测到此密码的清除状态（因为它没有前缀为 \{CRYPT\}），并再次对其进行加密。

etc/org.apache.karaf.jaas.cfg 配置文件允许您定义高级加密行为：

- **encryption.prefix 属性定义"flag"密码加密的前缀。默认值为 \{CRYPT\}。**
- **encryption.suffix 属性定义作为加密的密码"flag"的后缀。默认值为 \{CRYPT\}。**
- **encryption.algorithm 属性定义用于加密的算法（摘要）。可能的值有 MD2,MD5,SHA-1,SHA-256,SHA-384,SHA-512。默认值为 MD5。**
- **encryption.encoding 属性定义加密密码的编码。可能的值有 十六进制 或 base64。默认值为 十六进制。**

21.1.3. 通过密钥管理身份验证

对于 SSH 层，2019 年通过密钥支持身份验证，允许在不提供密码的情况下登录。

SSH 客户端(so bin/client 由 Karaf 本身提供，或者任何 ssh 客户端（如 OpenSSH)都使用一个公钥/私钥对，该密钥对将识别 Karaf SSHD （服务器端）上的自己。

允许连接的键存储在 etc/keys.properties 文件中，采用以下格式：

```
user=key,role
```

默认情况下，Tamener 允许 karaf 用户的密钥：

```
#
karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+
4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9Q
AAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6E
woFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWf
BpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,admin
```

注意

为安全起见，这个密钥被禁用。我们鼓励每个客户端创建密钥对并更新 etc/keys.properties 文件。

创建密钥对的最简单方法是使用 OpenSSH。

您可以使用以下方法创建密钥对：

```
ssh-keygen -t dsa -f karaf.id_dsa -N karaf
```

您现在有公钥和私钥：

```
-rw-------  1 jbonofre jbonofre    771 Jul 25 22:05 karaf.id_dsa
-rw-r--r--  1 jbonofre jbonofre    607 Jul 25 22:05 karaf.id_dsa.pub
```

您可以在 etc/keys.properties 中的 karaf.id_dsa.pub 文件的内容中复制：

```
karaf=AAAAB3NzaC1kc3MAAACBAJLj9vnEhu3/Q9Cvym2jRDaNWkATgQiHZxmErCmiLRuD5Klfv+HT/
+8WoYdnvj0YaXFP80phYhzZ7fbIO2LRFhYhPmGLa9nSeOsQlFuX5A9kY1120yB2kxSIZI0fU2hy1UCg
mTxdTQPSYtdWBJyvO/vczoX/8I3FziEfss07Hj1NAAAAFQD1dKEzkt4e7rBPDokPOMZigBh4kwAAAIEAi
LnpbGNbKm8SNLUEc/fJFswg4G4VjjngjbPZAjhkYe4+H2uYmynry6V+GOTS2kaFQGZRf9XhSpSwfdxK
tx7vCCaoH9bZ6S5Pe0voWmeBhJXi/Sww8f2stpitW2Oq7V7lDdDG81+N/D7/rKDD5PjUyMsVqc1n9wCT
mfqmi6XPEw8AAACAHAGwPn/Mv7P9Q9+JZRWtGq+i4pL1zs1OluiStCN9e/Ok96t3gRVKPheQ6IwLac
NjC9KkSKrLtsVyepGA+V5j/N+Cmsl6csZilnLvMUTvL/cmHDEEhTlQnPNrDDv+tED2BFqkajQqYLgMWe
GVqXsBU6IT66itZlYtrq4v6uDQG/o=,admin
```

并指定客户端使用 karaf.id_dsa 私钥：

```
bin/client -k ~/karaf.id_dsa
```

或 ssh

```
ssh -p 8101 -i ~/karaf.id_dsa karaf@localhost
```

21.1.4. RBAC

Apache Karaf 使用角色来控制对资源的访问：它是 RBAC （基于角色的访问控制）系统。

角色用于控制：

- 访问 OSGi 服务
- 访问控制台（控制命令的执行）
- 访问 JMX (MBeans 和/或操作)
- 访问 WebConsole

21.1.4.1. OSGi 服务

开发人员指南中介绍了有关 OSGi 服务 RBAC 支持的详细信息。

21.1.4.2. 控制台（Console）

控制台 RBAC 支持是 OSGi 服务 RBAC 的分类。实际上，在 Apache Karaf 中，所有控制台命令都定义为 OSGi 服务。

console 命令名称遵循 scope:name 格式。

ACL (Access Lists)在 etc/org.apache.karaf.command.acl.<scope>.cfg 配置文件中定义，其中 &lt ;scope> 是命令范围。

例如，我们可以通过创建 etc/org.apache.karaf.command.acl.feature.cfg 配置文件来定义到 feature:* 命令的 ACL。在这个 etc/org.apache.karaf.command.acl.feature.cfg 配置文件中，您可以设置：

```
list = viewer
info = viewer
install = admin
uninstall = admin
```

在这里，我们定义了 feature:list 和 feature:info 命令可由具有 viewer 角色的用户执行，而 feature:install 和 feature:uninstall 命令只能由具有 admin 角色的用户执行。请注意，admin 组中的用户也具有 viewer 角色，因此 能够执行所有内容。

Apache Karaf 命令 ACL 可以使用 （在给定命令范围内）来控制访问：

- 命令名称 regex （例如 name = role）
- 命令名称和选项或参数值 regex （例如 name[/.[0-9][0-9][0-9]+./] = role 仅执行名称，其参数值高于 100）

命令名称和选项/参数都支持完全匹配或正则表达式匹配。

默认情况下，Apache Karaf 定义以下命令 ACL：

- etc/org.apache.karaf.command.acl.bundle.cfg 配置文件定义 bundle:* 命令的 ACL。此 ACL 将 system 捆绑包的执行限制为具有 admin 角色的用户，而非系统的 捆绑包的 bundle:* 命令则可由具有 manager 角色的用户执行。

- etc/org.apache.karaf.command.acl.config.cfg 配置文件定义 config:* 命令的 ACL。此 ACL 将 config:* 命令的执行限制为带有 jmx.aclMapping, org.apache.karaf.command.acl, 和 org.apache.karaf.acl ruby 配置 PID 到具有 admin 角色的用户。对于其他配置 PID，具有 manager 角色的用户可以执行 config:* 命令。

- etc/org.apache.karaf.command.acl.feature.cfg 配置文件定义 feature:* 命令的 ACL。只有具有 admin 角色的用户才能执行 feature:install 和 feature:uninstall 命令。其他 feature:* 命令可由任何用户执行。

- etc/org.apache.karaf.command.acl.jaas.cfg 配置文件定义 jaas:* 命令的 ACL。只有具有 admin 角色的用户才能执行 jaas:update 命令。其他 jaas:* 命令可由任何用户执行。

- etc/org.apache.karaf.command.acl.kar.cfg 配置文件定义 kar:* 命令的 ACL。只有具有 admin 角色的用户才能执行 kar:install 和 kar:uninstall 命令。其他 kar:* 命令可由任何用户执行。

- etc/org.apache.karaf.command.acl.shell.cfg 配置文件定义 shell:* 和"direct"命令的 ACL。只有具有 admin 角色的用户才能执行 shell:edit、shell:exec、shell:new、shell:java 命令。其他 shell:* 命令可由任何用户执行。

您可以更改这些默认 ACL，并添加您自己的 ACL 以了解额外的命令范围（例如，对于 Apache Camel、... 中的 etc/org.apache.karaf.command.acl.cluster.cfg，等等/org.apache.karaf.command.acl.camel.cfg ）。

您可以通过编辑 etc/system.properties 中的 karaf.secured.services 属性来微调命令 RBAC 支持：

```
#
# By default, only Karaf shell commands are secured, but additional services can be
# secured by expanding this filter
#
karaf.secured.services = (&(osgi.command.scope=*)(osgi.command.function=*))
```

21.1.4.3. JMX

与 console 命令类似，您可以将 ACL (AccessList)定义为 JMX 层。

JMX ACL 在 etc/jmx.acl<ObjectName>.cfg 配置文件中定义，其中 & lt;ObjectName > 是一个 MBean 对象名称（用于实例 org.apache.karaf.bundle 代表 org.apache.karaf;type=Bundle MBean）。

etc/jmx.acl.cfg 是最通用的配置文件，在没有找到特定配置文件时使用。它包含"global"ACL 定义。

JMX ACL 可以使用 （在 JMX MBean 中）来控制访问：

- 操作名称 regex （如 operation* = role）
- 操作参数值 regex (e.g. operation (java.lang.String, int)[/([1-4])?[0-9]/,/ ruby/] = role)

默认情况下，Apache Karaf 定义以下 JMX ACL：

- etc/jmx.acl.org.apache.karaf.bundle.cfg 配置文件定义 org.apache.karaf:type=bundle MBean 的 ACL。此 ACL 限制了针对具有 admin 角色的用户的系统捆绑包的 setStartLevel （）、start （）、stop （） 和 update （） 操作。其他操作可由具有 manager 角色的用户执行。
- etc/jmx.acl.org.apache.karaf.config.cfg 配置文件定义 org.apache.karaf:type=config MBean 的 ACL。此 ACL 限制了对具有 admin 角色的用户的 jmx.acl*、org.apache.karaf.command.acl* 和 org.apache.karaf.service.acl* 配置 PID 的更改。其他操作可由具有 manager 角色的用户执行。
- etc/jmx.acl.org.apache.karaf.security.jmx.cfg 配置文件定义 org.apache.karaf:type=security,area=jmx MBean 的 ACL。此 ACL 限制具有 viewer 角色的用户的 canInvoke （） 操作的调用。
- etc/jmx.acl.osgi.compendium.cm.cfg 配置文件定义 osgi.compendium:type=cm MBean 的 ACL。此 ACL 限制了对具有 admin 角色的用户的

jmx.acl*、org.apache.karaf.command.acl* 和 org.apache.karaf.service.acl* 配置 PID 的更改。其他操作可由具有 manager 角色的用户执行。

- etc/jmx.acl.java.lang.Memory.cfg 配置文件定义核心 JVM 内存 MBean 的 ACL。此 ACL 限制仅针对具有 manager 角色的用户调用 gc 操作。

- etc/jmx.acl.cfg 配置文件是最通用的文件。当没有其他特定 ACL 匹配时，会使用此处定义的 ACL （通过特定的 ACL，它是其它 MBean 特定的 MBean 特定 etc/jmx.acl125.cfg 配置文件中定义的 ACL）。list*（）、get* （） 是* （） 操作，可由具有 viewer 角色的用户执行。set*（） 和所有其他 *（） 操作可由具有 admin 角色的用户执行。

21.1.4.4. WebConsole

默认情况下，Apache Karaf WebConsole 不可用。要启用它，您必须安装 webconsole 功能：

```
karaf@root()> feature:install webconsole
```

WebConsole 不支持目前的精细 RBAC，如控制台或 JMX。

具有 admin 角色的所有用户都可以记录 WebConsole 并执行任何操作。

21.1.5. SecurityMBean

Apache Karaf 提供了一个 JMX MBean，用于检查当前用户是否可以调用给定的 MBean 和/或操作。

canInvoke （） 操作获取当前用户的角色，并检查一个角色是否可以调用 MBean 和/或操作，最终具有给定参数值。

21.1.5.1. 操作

- 如果当前用户可以使用 objectName,false other 调用 MBean，则 canInvoke (objectName) 返回 true。

- 如果当前用户可以在 MBean 上调用带有 objectName,false 其他的 operation methodName，则 canInvoke (objectName, methodName) 返回 true。

- 如果当前用户可以调用带有 objectName 的 MBean 类型 parameterTypes 数组的 operation methodName，则 canInvoke (objectName, methodName, parameter Types) returns true。

- canInvoke (bulkQuery) 返回包含 bulkQuery 选项卡中每个操作的表格数据（如果 canInvoke 为 true 或 false ）。

21.1.6. 安全供应商

有些应用程序需要特定的安全供应商可用，如 [BouncyCastle|http://www.bouncycastle.org]。

JVM 对此类 jars 的使用施加一些限制：它们必须签名并在引导类路径上可用。

部署这些提供程序的一种方法是将其放在 $JAVA_HOME/jre/lib/ext 的 JRE 文件夹中，并修改安全策略配置($JAVA_HOME/jre/lib/security/java.security)以注册此类提供程序。

虽然这种方法可以正常工作，但它具有全局效果，但要求您相应地配置所有服务器。

Apache Karaf 提供了配置额外安全提供程序的简单方法：* 将您的供应商 jar 置于 lib/ext * 中，修改 etc/config.properties 配置文件以添加下列属性

```
org.apache.karaf.security.providers = xxx,yyy
```

此属性的值是要注册的供应商类名称的逗号分隔列表。

例如，要添加 bouncycastle 安全供应商，您可以定义：

```
org.apache.karaf.security.providers = org.bouncycastle.jce.provider.BouncyCastleProvider
```

此外，您可能还希望提供从系统捆绑包中对这些提供程序的类的访问权限，以便所有捆绑包都可以访问这些捆绑包。

它可以通过修改同一配置文件中的 org.osgi.framework.bootdelegation 属性来实现：

```
org.osgi.framework.bootdelegation = ...,org.bouncycastle*
```