Red Hat AMQ 2021.Q3

使用 AMQ OpenWire JMS 客户端

用于 AMQ 客户端 2.10

Last Updated: 2023-02-20

Red Hat AMQ 2021.Q3 使用 AMQ OpenWire JMS 客户端

用于 AMQ 客户端 2.10

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本指南描述了如何安装和配置客户端，运行实践示例，并将您的客户端与其他 AMQ 组件搭配使用。

目录

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。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 CTO Chris Wright 信息。

第 1 章 概述

AMQ OpenWire JMS 是 Java 消息服务(JMS)1.1 客户端，用于发送和接收 OpenWire 消息的消息传递应用。

重要

AMQ OpenWire JMS 客户端现已在 AMQ 7 中弃用。建议此客户端的用户迁移到 AMQ JMS 或 AMQ 核心协议 JMS。

AMQ OpenWire JMS 是 AMQ 客户端的一部分，这是支持多种语言和平台的消息传递库套件。有关客户端的概述，请参阅 AMQ 客户端概述。有关此发行版本的详情，请参考 AMQ Clients 2.10 发行注记。

AMQ OpenWire JMS 基于 Apache ActiveMQ 的 JMS 实施。如需有关 JMS API 的更多信息，请参阅 JMS API 参考 和 JMS 教程。

1.1. 主要特性

- JMS 1.1 compatible
- 用于安全通信的 SSL/TLS
- 自动重新连接和故障转移
- 分布式事务(XA)
- 纯 Java 实施

1.2. 支持的标准和协议

AMQ OpenWire JMS 支持以下业界认可的标准和网络协议：

- Java 消息服务 API 版本 1.1.
- 使用 IPv6的现代 TCP

1.3. 支持的配置

有关 AMQ OpenWire JMS 支持的配置，请参阅红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

1.4. 术语和概念

本节介绍核心 API 实体，并描述它们如何协同运作。

表 1.1. API 术语

实体	描述
ConnectionFactory	用于创建连接的入口点。
Connection	个网络上两个同级之间的通信通道。它包含会话。

实体	描述
Session	用于生成和使用消息的环境。它包含消息制作者和消费者。
MessageProducer	用于将消息发送到目的地的频道。它具有目标目的地。
MessageConsumer	从目的地接收消息的频道。它具有源目的地。
Destination	消息的指定位置，可以是队列，也可以是主题。
Queue	存储的消息序列.
Topic	存储的用于多播分发的消息序列。
Message	特定于应用的信息。

AMQ OpenWire JMS 发送并接收 *消息*。*消息使用消息生产者和消费者* 在连接的对等点之间传输 。生产者和消费者通过 *会话* 建立。通过 *连接* 建立会话。*连接由连接工厂* 创建。

发送对等点会创建一个制作者来发送消息。制作者具有在远程同级上标识目标队列或主题 *的目的地*。接收方创建接收消息的消费者。与制作者一样，消费者也有在远程同级上标识源队列或主题的目的地。

目标是 *队列* 或 *主题*。在 JMS 中，队列和主题是包含消息的指定代理实体的客户端显示。

队列实施点对点语义。每条消息仅能被一个使用者看到，消息会在读取后从队列中删除。主题实施发布与订阅语义。每条消息都由多个使用者看到，该消息在读取后可供其他消费者使用。

如需更多信息，请参阅 JMS 指南。

1.5. 文档惯例

sudo 命令
在本文档中，**sudo** 用于任何需要 root 权限的命令。使用 **sudo** 时要小心，因为任何更改都可能会影响整个系统。有关 **sudo** 的详情请参考 使用 sudo 命令。

文件路径
在这个文档中，所有文件路径都对 Linux、UNIX 和类似操作系统有效（例如 **/home/andrea**）。在 Microsoft Windows 中，您必须使用等效的 Windows 路径（例如 **C:\Users\andrea**）。

变量文本
本文档包含代码块，它们需要使用特定于环境的值替换。变量文本括在箭头大括号内，样式为圆形单空间。例如，在以下命令中，将 ***<project-dir>*** 替换为您的环境的值：

```
$ cd <project-dir>
```

第 2 章 安装

本章将指导您完成在您的环境中安装 AMQ OpenWire JMS 的步骤。

2.1. 先决条件

- 您必须有 订阅 才能访问 AMQ 发行文件和存储库。
- 要使用 AMQ OpenWire JMS 构建程序，您必须安装 Apache Maven。
- 要使用 AMQ OpenWire JMS，您必须安装 Java。

2.2. 使用 RED HAT MAVEN 存储库

配置您的 Maven 环境，以从红帽 Maven 存储库下载客户端库。

流程

1. 将红帽存储库添加到您的 Maven 设置或 POM 文件。如需示例配置文件，请参阅 第 B.1 节 “使用在线存储库”。

```
<repository>
  <id>red-hat-ga</id>
  <url>https://maven.repository.redhat.com/ga</url>
</repository>
```

2. 将库依赖关系添加到您的 POM 文件。

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-client</artifactId>
  <version>5.11.0.redhat-630416</version>
</dependency>
```

该客户端现在在 Maven 项目中可用。

2.3. 安装本地 MAVEN 存储库

作为在线存储库的替代选择，可以将 AMQ OpenWire JMS 作为基于文件的 Maven 存储库安装到本地文件系统中。

流程

1. 使用您的订阅 下载 **AMQ Broker 7.8.2 Maven 存储库**.zip 文件。
2. 将文件内容提取到您选择的目录中。
   在 Linux 或 UNIX 中，使用 **unzip** 命令提取文件内容。

```
$ unzip amq-broker-7.8.2-maven-repository.zip
```

   在 Windows 上，右键单击 .zip 文件并选择" **提取所有"。**

3. 配置 Maven，以使用提取的安装目录中 **maven-repository** 目录中的存储库。如需更多信息，请参阅 第 B.2 节 “使用本地存储库”。

2.4. 安装示例

流程

1. 使用您的订阅 下载 **AMQ Broker 7.8.2**.zip 文件。

2. 将文件内容提取到您选择的目录中。
在 Linux 或 UNIX 中，使用 **unzip** 命令提取文件内容。

```
$ unzip amq-broker-7.8.2.zip
```

在 Windows 上，右键单击 .zip 文件并选择" **提取所有"。**

当您提取 .zip 文件的内容时，将创建一个名为 **amq-broker-7.8.2** 的目录。这是安装的顶级目录，在整个文档中被称为 ***<install-dir>***。

第 3 章 入门

本章将引导您完成设置环境并运行简单消息传递程序的步骤。

3.1. 先决条件

- 若要构建示例，必须将 Maven 配置为 使用红帽存储库 或 本地存储库。
- 您必须 安装示例。
- 您必须有一个在 **localhost** 中侦听连接的消息代理。它必须启用匿名访问。如需更多信息，请参阅 启动代理。
- 您必须有一个名为 **exampleQueue** 的队列。如需更多信息，请参阅 创建队列。

3.2. 运行第一个示例

这个示例为名为 **exampleQueue** 的队列创建使用者和制作者。它发送文本消息，然后接收回消息，将收到的消息打印到控制台。

流程

1. 通过在 ***<install-dir>*/examples/protocols/openwire/queue** 目录中运行以下命令来使用 Maven 构建示例。

```
$ mvn clean package dependency:copy-dependencies -DincludeScope=runtime -DskipTests
```

添加 **dependency:copy-dependencies** 会导致依赖项复制到 **target/dependency** 目录中。

2. 使用 **java** 命令来运行示例。
在 Linux 或 UNIX 中：

```
$ java -cp "target/classes:target/dependency/*"
org.apache.activemq.artemis.jms.example.QueueExample
```

在 Windows 中：

```
> java -cp "target\classes;target\dependency\*"
org.apache.activemq.artemis.jms.example.QueueExample
```

在 Linux 上运行它会产生以下输出：

```
$ java -cp "target/classes:target/dependency/*"
org.apache.activemq.artemis.jms.example.QueueExample
Sent message: This is a text message
Received message: This is a text message
```

这个示例的源代码位于 ***<install-dir>*/examples/protocols/openwire/queue/src** 目录中。***<install-dir>*/examples/protocols/openwire** 目录中提供了其他示例。

第 4 章 CONFIGURATION

本章介绍将 AMQ OpenWire JMS 实施绑定到您的 JMS 应用并设置配置选项的流程。

JMS 使用 Java 命名目录接口(JNDI)来注册和查找 API 实施和其他资源。这可让您编写代码到 JMS API，而不将它与特定的实施关联。

配置选项作为连接 URI 上的查询参数公开。

有关配置 AMQ OpenWire JMS 的更多信息，请参阅 ActiveMQ 用户指南。

4.1. 配置 JNDI 初始上下文

JMS 应用使用从 **InitialContextFactory** 获取的 JNDI **InitialContext** 对象来查找 JMS 对象，如连接工厂。AMQ OpenWire JMS 在 **org.apache.activemq.jndi.ActiveMQInitialContextFactory** 类中提供 **InitialContextFactory** 实施。

当 **InitialContext** 对象被实例化时，**InitialContextFactory** 实现会被发现：

```
javax.naming.Context context = new javax.naming.InitialContext();
```

要查找实施，必须在您的环境中配置 JNDI。实现这一目标的方法有三种：使用 **jndi.properties** 文件、使用系统属性或使用初始上下文 API。

使用 jndi.properties 文件

创建名为 **jndi.properties** 的文件，并将其放置在 Java 类路径中。使用键 **java.naming.factory.initial** 添加属性。

示例：使用 jndi.properties 文件设置 JNDI 初始上下文工厂

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
```

在基于 Maven 的项目中，**jndi.properties** 文件放置在 ***<project-dir>*/src/main/resources** 目录中。

使用系统属性

设置 **java.naming.factory.initial** 系统属性。

示例：使用系统属性设置 JNDI 初始上下文工厂

```
$ java -Djava.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory ...
```

使用初始上下文 API

使用 JNDI 初始上下文 API 以编程方式设置属性。

示例：以编程方式设置 JNDI 属性

```
Hashtable<Object, Object> env = new Hashtable<>();

env.put("java.naming.factory.initial", "org.apache.activemq.jndi.ActiveMQInitialContextFactory");

InitialContext context = new InitialContext(env);
```

请注意，您可以使用同一 API 为连接工厂、队列和主题设置 JNDI 属性。

4.2. 配置连接工厂

JMS 连接工厂是创建连接的入口点。它使用对应用特定配置设置进行编码的连接 URI。

要设置工厂名称和连接 URI，请按照以下格式创建一个属性：您可以将该配置存储在 **jndi.properties** 文件中，或者设置对应的系统属性。

连接工厂的 **JNDI** 属性格式

```
connectionFactory.<lookup-name> = <connection-uri>
```

例如，这是您可以如何配置一个名为 **app1** 的工厂：

示例：在 **jndi.properties** 文件中设置连接工厂

```
connectionFactory.app1 = tcp://example.net:61616?jms.clientID=backend
```

然后，您可以使用 JNDI 上下文使用名称 **app1** 来查找配置的连接工厂：

```
ConnectionFactory factory = (ConnectionFactory) context.lookup("app1");
```

4.3. 连接 URI

连接配置使用连接 URI。连接 URI 指定远程主机、端口和一组配置选项，这些选项设置为查询参数。有关可用选项的详情请参考 第 5 章 *配置选项*。

连接 **URI** 格式

```
<scheme>://<host>:<port>[?<option>=<value>[&<option>=<value>...]]
```

这个方案是 **tcp** 用于未加密的连接，**ssl** 用于 SSL/TLS 连接。

例如，以下连接 URI 通过端口 **61616** 连接到主机 **example.net**，并将客户端 ID 设置为 **backend** ：

示例：连接 **URI**

```
tcp://example.net:61616?jms.clientID=backend
```

故障切换 **URI**
用于重新连接和故障转移的 URI 可以包含多个连接 URI。它们采用以下形式：

故障转移 **URI** 格式

```
failover:(<connection-uri>[,<connection-uri>])[?<option>=<value>[&<option>=<value>...]]
```

前缀为 **nested.** 的传输选项应用到列表中的每个连接 URI。

4.4. 配置队列和主题名称

JMS 提供使用 JNDI 来查找特定于部署的队列和主题资源的选项。

要在 JNDI 中设置队列和主题名称，请使用以下格式创建属性：将此配置放在 **jndi.properties** 文件中，或者设置对应的系统属性。

队列和主题的 JNDI 属性格式

```
queue.<lookup-name> = <queue-name>
topic.<lookup-name> = <topic-name>
```

例如，以下属性为两个特定于部署的资源定义名称 **jobs** 和 **notifications** ：

示例：在 jndi.properties 文件中设置队列和主题名称

```
queue.jobs = app1/work-items
topic.notifications = app1/updates
```

然后，您可以根据 JNDI 名称查找资源：

```
Queue queue = (Queue) context.lookup("jobs");
Topic topic = (Topic) context.lookup("notifications");
```

第 5 章 配置选项

本章列出了 AMQ OpenWire JMS 的可用配置选项。

JMS 配置选项设置为连接 URI 上的查询参数。如需更多信息，请参阅 第 4.3 节 “连接 URI”。

5.1. JMS 选项

jms.username

客户端用于验证连接的用户名。

jms.password

客户端用于验证连接的密码。

jms.clientID

客户端应用到连接的客户端 ID。

jms.closeTimeout

JMS 关闭操作的超时时间（毫秒）。默认值为 15000（15 秒）。

jms.connectResponseTimeout

JMS 连接操作的超时时间（以毫秒为单位）。默认值为 0，即没有超时。

jms.sendTimeout

JMS 发送操作的超时时间（毫秒）。默认值为 0，即没有超时。

jms.checkForDuplicates

如果启用，忽略重复的信息。它会被默认启用。

jms.disableTimeStampsByDefault

如果启用，则不要时间戳信息。它默认是禁用的。

jms.useAsyncSend

如果启用，则在不等待确认的情况下发送消息。它默认是禁用的。

jms.alwaysSyncSend

如果启用，发送会在所有交付模式中等待确认。它默认是禁用的。

jms.useCompression

如果启用，压缩消息正文。它默认是禁用的。

jms.useRetroactiveConsumer

如果启用，不可更改的订阅者可接收在订阅启动前发布的消息。它默认是禁用的。

预定义策略选项

prefetch 策略决定每个 **MessageConsumer** 从远程 peer 获取的信息数，并保存在本地"prefetch"缓冲区中。

jms.prefetchPolicy.queuePrefetch

用于队列的 prefetch 消息数。默认值为 1000。

jms.prefetchPolicy.queueBrowserPrefetch

用于队列浏览器的 prefetch 消息数。默认值为 500。

jms.prefetchPolicy.topicPrefetch

用于非持久性主题的 prefetch 消息数。默认值为 32766。

jms.prefetchPolicy.durableTopicPrefetch

要预先填充持久主题的消息数量。默认值为 100。

jms.prefetchPolicy.all

这可以用于一次性设置所有 prefetch 值。

prefetch 的值可能会影响消息到队列上的多个消费者的分发。更高的值可能会导致向每位消费者同时发送的批处理。当消费者以不同速度运作时，要达到更加均匀的循环分布，请使用较低的值。

重新传送策略选项
重新传送策略控制如何在客户端上处理重新传送消息。

jms.redeliveryPolicy.maximumRedeliveries

在将消息发送到死信队列之前，尝试重新传送的次数。默认值为 6. -1 表示没有限制。

jms.redeliveryPolicy.redeliveryDelay

重新传送尝试之间的时间（毫秒）。如果 **initialRedeliveryDelay** 是 0，则会使用它。默认值为 1000（1 秒）。

jms.redeliveryPolicy.initialRedeliveryDelay

第一次重新传送尝试前的时间（毫秒）。默认值为 1000（1 秒）。

jms.redeliveryPolicy.maximumRedeliveryDelay

重新传送尝试之间的最长时间（毫秒）。如果启用了 **useExponentialBackOff**，则会使用它。默认值为 1000（1 秒）。-1 表示没有限制。

jms.redeliveryPolicy.useExponentialBackOff

如果启用，在每次后续尝试中增加重新传送延迟。它默认是禁用的。

jms.redeliveryPolicy.backOffMultiplier

增加重新传送延迟的倍数。默认值为 5。

jms.redeliveryPolicy.useCollisionAvoidance

如果启用，请调整重新传送延迟略微上或下移以避免冲突。它默认是禁用的。

jms.redeliveryPolicy.collisionAvoidanceFactor

用于调整重新传送延迟的倍数。默认值为 0.15。

nonBlockingRedelivery

如果启用，则允许不按顺序重新传送，以避免行头阻止。它默认是禁用的。

5.2. TCP 选项

closeAsync

如果启用，请在单独的线程中关闭套接字。它会被默认启用。

connectionTimeout

TCP 连接操作的超时时间（毫秒）。默认值为 30000（30 秒）。0 表示没有超时。

dynamicManagement

如果启用，允许对传输日志记录器进行 JMX 管理。它默认是禁用的。

ioBufferSize

I/O 缓冲区大小（以字节为单位）。默认值为 8192(8 KiB)。

jmxPort

JMX 管理的端口。默认值为 1099。

keepAlive

如果启用，请使用 TCP keepalive。这与基于 **KeepAliveInfo** 信息的 keepalive 机制不同。它默认是禁用的。

logWriterName

org.apache.activemq.transport.LogWriter 实现的名称。名称解析映射保存在 **resources/META-INF/services/org/apache/activemq/transport/logwriters** 目录中。默认值为 **default**。

soLinger

套接字闲置器选项。默认值为 0。

soTimeout

套接字读取操作的超时时间（毫秒）。默认值为 0，即没有超时。

soWriteTimeout

套接字写入操作的超时时间（毫秒）。默认值为 0，即没有超时。

startLogging

如果启用，并启用了 **trace** 选项，日志传输启动事件。它会被默认启用。

tcpNoDelay

如果启用，则不要延迟，也不会缓冲 TCP。它默认是禁用的。

threadName

如果设置，则分配给传输线程的名称。远程地址附加到名称中。默认情况下取消设置。

Trace

如果启用，将传输事件记录到 **log4j.logger.org.apache.activemq.transport.TransportLogger**。它默认是禁用的。

useInactivityMonitor

如果启用，则超时连接无法发送 **KeepAliveInfo** 信息。它会被默认启用。

useKeepAlive

如果启用，定期发送 **KeepAliveInfo** 信息以防止连接超时。它会被默认启用。

useLocalHost

如果启用，使用名称 **localhost** 而不是当前主机名进行本地连接。它默认是禁用的。

5.3. SSL/TLS 选项

socket.keyStore

到 SSL/TLS 密钥存储的路径。mutual SSL/TLS 身份验证需要密钥存储。如果未设置，则使用 **javax.net.ssl.keyStore** 系统属性的值。

socket.keyStorePassword

SSL/TLS 密钥存储的密码。如果未设置，则使用 **javax.net.ssl.keyStorePassword** 系统属性的值。

socket.keyStoreType

信任存储类型的字符串名称。默认值为 **java.security.KeyStore.getDefaultType()** 的值。

socket.trustStore

SSL/TLS 信任存储的路径。如果未设置，则使用 **javax.net.ssl.trustStore** 系统属性的值。

socket.trustStorePassword

SSL/TLS 信任存储的密码。如果未设置，则使用 **javax.net.ssl.trustStorePassword** 系统属性的值。

socket.trustStoreType

信任存储类型的字符串名称。默认值为 **java.security.KeyStore.getDefaultType()** 的值。

socket.enabledCipherSuites

要启用的密码套件的逗号分隔列表。如果未设置，则使用 JVM 默认密码。

socket.enabledProtocols

要启用的 SSL/TLS 协议的逗号分隔列表。如果未设置，则使用 JVM 默认协议。

5.4. OPENWIRE 选项

wireFormat.cacheEnabled

如果启用，请通过缓存常用值以避免过多的清理和带宽消耗。它会被默认启用。

wireFormat.cacheSize

缓存条目的数量。缓存是每个连接的。默认值为 1024。

wireFormat.maxInactivityDuration

连接前的最长时间（以毫秒为单位）被视为无效。默认值为 30000（30 秒）。

wireFormat.maxInactivityDurationInitalDelay

不活跃检查开始前的初始延迟（毫秒）。请注意，**Inital** 拼写错误。默认值为 10000（10 秒）。

wireFormat.maxFrameSize

最大帧大小，以字节为单位。默认值为 **java.lang.Long.MAX_VALUE** 的值。

wireFormat.sizePrefixDisabled

如果设置为 true，则不要使用它们的大小为数据包添加前缀。默认为 false。

wireFormat.stackTraceEnabled

如果启用，将堆栈跟踪从服务器上的异常发送到客户端。它会被默认启用。

wireFormat.tcpNoDelayEnabled

如果启用，告诉服务器激活 **TCP_NODELAY**。它会被默认启用。

wireFormat.tightEncodingEnabled

如果启用，则对线路上的较小的编码进行优化。这会增加 CPU 用量。它会被默认启用。

5.5. 故障切换选项

maxReconnectAttempts

在将连接报告失败前允许的重新连接次数。默认值为 -1，即没有限制。0 可禁用重新连接。

maxReconnectDelay

第二次和后续重新连接尝试之间的最长时间，以毫秒为单位。默认值为 30000（30 秒）。

randomize

如果启用，随机选择其中一个故障切换端点。它会被默认启用。

reconnectDelayExponent

增加重新连接的倍数会延迟 backoff。默认值为 2.0。

useExponentialBackOff

如果启用，在每次后续尝试增加重新连接延迟。它会被默认启用。

timeout

发送等待重新连接的操作的超时时间（毫秒）。默认值为 -1，即没有超时。

第 6 章 消息发送

6.1. 写入到流传输的大信息

要写入到大信息，使用 **BytesMessage.writeBytes()** 方法。以下示例从文件中读取字节并将其写入信息：

示例：编写到流传输的大信息

```
BytesMessage message = session.createBytesMessage();
File inputFile = new File(inputFilePath);
InputStream inputStream = new FileInputStream(inputFile);

int numRead;
byte[] buffer = new byte[1024];

while ((numRead = inputStream.read(buffer, 0, buffer.length)) != -1) {
    message.writeBytes(buffer, 0, numRead);
}
```

6.2. 从流传输的大消息中读取

要从大消息中读取，请使用 **BytesMessage.readBytes()** 方法。以下示例从信息中读取字节并将其写入文件：

示例：从流传输的大消息中读取

```
BytesMessage message = (BytesMessage) consumer.receive();
File outputFile = new File(outputFilePath);
OutputStream outputStream = new FileOutputStream(outputFile);

int numRead;
byte buffer[] = new byte[1024];

for (int pos = 0; pos < message.getBodyLength(); pos += buffer.length) {
    numRead = message.readBytes(buffer);
    outputStream.write(buffer, 0, numRead);
}
```

附录 A. 使用您的订阅

AMQ 通过软件订阅提供。要管理您的订阅，请访问红帽客户门户中的帐户。

A.1. 访问您的帐户

流程

1. 转至 access.redhat.com。
2. 如果您还没有帐户，请创建一个帐户。
3. 登录到您的帐户。

A.2. 激活订阅

流程

1. 转至 access.redhat.com。
2. 导航到 **My Subscriptions**。
3. 导航到 **激活订阅** 并输入您的 16 位激活号。

A.3. 下载发行文件

要访问 .zip、.tar.gz 和其他发布文件，请使用客户门户查找要下载的相关文件。如果您使用 RPM 软件包或 Red Hat Maven 存储库，则不需要这一步。

流程

1. 打开浏览器并登录红帽客户门户网站 **产品下载页面**，网址为 access.redhat.com/downloads。
2. 查找 **INTEGRATION 类别中 的红帽 AMQ** 条目。
3. 选择所需的 AMQ 产品。此时会打开 **Software Downloads** 页面。
4. 单击组件的 **Download** 链接。

A.4. 为系统注册软件包

要在 Red Hat Enterprise Linux 上安装此产品的 RPM 软件包，必须注册您的系统。如果您使用下载的发行文件，则不需要这一步。

流程

1. 转至 access.redhat.com。
2. 进入 **Registration Assistant**。
3. 选择您的操作系统版本，再继续到下一页。
4. 使用您的系统终端中列出的命令完成注册。

有关注册您的系统的更多信息，请参阅以下资源之一：

- Red Hat Enterprise Linux 7 - 注册系统并管理订阅
- Red Hat Enterprise Linux 8 - 注册系统并管理订阅

附录 B. 使用红帽 MAVEN 存储库

本节论述了如何在您的软件中使用红帽提供的 Maven 存储库。

B.1. 使用在线存储库

红帽维护一个中央 Maven 存储库，用于基于 Maven 的项目。如需更多信息，请参阅 存储库欢迎页面。

可以通过两种方式将 Maven 配置为使用 Red Hat 存储库：

- 将存储库添加到您的 Maven 设置中
- 将软件仓库添加到您的 POM 文件

将存储库添加到 Maven 设置中

这种配置方法适用于您的用户拥有的所有 Maven 项目，只要您的 POM 文件不覆盖存储库配置并启用包含的配置集。

流程

1. 找到 Maven **settings.xml** 文件。它通常位于用户主目录的 **.m2** 目录中。如果文件不存在，请使用文本编辑器创建该文件。
   在 Linux 或 UNIX 中：

   ```
   /home/<username>/.m2/settings.xml
   ```

   在 Windows 中：

   ```
   C:\Users\<username>\.m2\settings.xml
   ```

2. 在 **settings.xml** 文件的 **profiles** 元素中添加包含红帽存储库的新配置集，如下例所示：

   示例：包含 Red Hat 软件仓库的 Maven **settings.xml** 文件

   ```
   <settings>
     <profiles>
       <profile>
         <id>red-hat</id>
         <repositories>
           <repository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
           </repository>
         </repositories>
         <pluginRepositories>
           <pluginRepository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
             <releases>
               <enabled>true</enabled>
             </releases>
             <snapshots>
               <enabled>false</enabled>
             </snapshots>
   ```

```
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>red-hat</activeProfile>
  </activeProfiles>
</settings>
```

有关 Maven 配置的更多信息，请参阅 Maven 设置参考。

在您的 **POM** 文件中添加软件仓库
要在项目中直接配置存储库，请在 POM 文件的 **repositories** 元素中添加一个新的条目，如下例所示：

示例：包含 **Red Hat** 软件仓库的 **Maven pom.xml** 文件

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>example-app</artifactId>
  <version>1.0.0</version>

  <repositories>
    <repository>
      <id>red-hat-ga</id>
      <url>https://maven.repository.redhat.com/ga</url>
    </repository>
  </repositories>
</project>
```

有关 POM 文件配置的更多信息，请参阅 Maven POM 参考。

B.2. 使用本地存储库

红帽为其部分组件提供基于文件的 Maven 存储库。这些内容作为可下载存档提供，您可以提取到本地文件系统。

要将 Maven 配置为使用本地提取的存储库，请在 Maven 设置或 POM 文件中应用以下 XML：

```
<repository>
  <id>red-hat-local</id>
  <url>${repository-url}</url>
</repository>
```

${repository-url} 必须是包含提取仓库本地文件系统路径的文件 URL。

表 B.1. 本地 Maven 存储库的 URL 示例

操作系统	文件系统路径	URL
Linux 或 UNIX	**/home/alice/maven-repository**	**file:/home/alice/maven-repository**

操作系统	文件系统路径	URL
Windows	**C:\repos\red-hat**	**file:C:\repos\red-hat**

附录 C. 将 AMQ BROKER 与示例搭配使用

AMQ OpenWire JMS 示例需要一个正在运行的消息代理，其中包含名为 **exampleQueue** 的队列。使用以下步骤安装和启动代理并定义队列。

C.1. 安装代理

按照 AMQ Broker 入门 以 安装代理 并创建代理 实例 中的内容进行操作。启用匿名访问。

以下步骤将代理实例的位置称为 ***<broker-instance-dir>***。

C.2. 启动代理

流程

1. 使用 **artemis run** 命令启动代理。

```
$ <broker-instance-dir>/bin/artemis run
```

2. 检查控制台输出中是否有启动过程中记录的严重错误。代理日志记录 **Server is now live** （当它就绪时）。

```
$ example-broker/bin/artemis run

        __  __  ____    ____           _
       /\ |  \/  |/ __ \ |  _ \         | |
      /  \ | \  / | |  | | |_) |_ __ ___| | _____ _ __
     / /\ \| |\/| | |  | | |  _ <| '__/ _ \ |/ / _ \ '__|
    / ____ \ |  | | |__| | | |_) | | |  __/   <  __/ |
   /_/    \_\_|  |_|\____\_\ |____/|_|  \___|_|\_\___|_|

 Red Hat AMQ <version>

2020-06-03 12:12:11,807 INFO  [org.apache.activemq.artemis.integration.bootstrap]
AMQ101000: Starting ActiveMQ Artemis Server
...
2020-06-03 12:12:12,336 INFO  [org.apache.activemq.artemis.core.server] AMQ221007:
Server is now live
...
```

C.3. 创建队列

在新终端中，使用 **artemis queue** 命令创建名为 **exampleQueue** 的队列。

```
$ <broker-instance-dir>/bin/artemis queue create --name exampleQueue --address exampleQueue --auto-create-address --anycast
```

系统将提示您回答一系列"是"或无问题。对所有设备回答 **N**。

队列创建后，代理就可与示例程序配合使用。

C.4. 停止代理

运行完示例后，使用 **artemis stop** 命令停止代理。

```
$ <broker-instance-dir>/bin/artemis stop
```

2021-08-31 15:46:51 +1000 ф┐ ошов