Red Hat JBoss Enterprise Application Platform 7.4

开发 Jakarta Enterprise Beans 应用程序

面向希望为红帽 JBoss 企业应用平台开发和部署 Jakarta 企业 Bean 应用程序的开发人员和管理员提供了相关说明和信息。

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 7.4 开发 Jakarta Enterprise Beans 应用程序

面向希望为红帽 JBoss 企业应用平台开发和部署 Jakarta 企业 Bean 应用程序的开发人员和管理员提供了相关说明和信息。

法律通告

摘要

本文档为希望使用红帽 JBoss 企业应用平台开发和部署 Jakarta 企业 Bean 应用程序的开发人员和管理员提供了信息。

目录

提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 请包含 **文档 URL、章节编号 并描述问题**。
3. 在 **Summary** 中输入问题的简短描述。
4. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
5. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

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 简介

1.1. JAKARTA ENTERPRISE BEANS 概述

Jakarta Enterprise Beans 是一款 API，通过使用名为 Enterprise Beans 的服务器端组件开发分布式、事务型、安全和可移植的 Jakarta EE 应用程序。企业 Bean 以分离的方式实施应用程序的业务逻辑，从而鼓励重复使用。Jakarta Enterprise Beans 记录在 Jakarta Enterprise Beans 3.2 规范中。

1.2. JAKARTA ENTERPRISE BEANS 3.2 功能集

JBoss EAP 7.3 及更新的版本支持以下 Jakarta Enterprise Beans 3.2 功能：

- 会话组件
- 消息驱动的 Bean
- Jakarta Enterprise Beans API 组
- no-interface 视图
- 本地接口
- 远程接口
- AutoClosable 接口
- 定时器服务
- 异步调用
- Jakarta Interceptors
- RMI/IIOP 互操作性
- 事务支持
- 安全性
- 可嵌入 API

JBoss EAP 7 不再支持以下功能：

- EJB 2.1 实体 bean 客户端视图
- 具有 bean 管理持久性的实体 Bean
- 具有容器管理的持久性的实体 Bean
- EJB 查询语言(EJB QL)
- 基于 JAX-RPC 的 Web 服务：端点和客户端视图

1.3. ENTERPRISE BEANS

企业 Bean 采用 Java 类编写，并附带相应的 Jakarta Enterprise Beans 注释。它们可以在自己的存档（JAR 文件）中部署到应用服务器，或者作为 Jakarta EE 应用的一部分进行部署。应用程序服务器管理每个企业 bean 的生命周期，并提供服务，如安全性、事务和并发管理。

企业 Bean 还可以定义任意数量的业务接口。业务接口可更好地控制 Bean 哪些方法可供客户端使用，也允许访问在远程 JVM 中运行的客户端。

企业 Bean 有三种类型： 会话 Bean、消息驱动型 Bean 和实体 Bean。

注意

JBoss EAP 不支持实体 Bean。

1.3.1. 编写企业 Bean

企业 Bean 打包和部署在 Java 存档(JAR)文件中。您可以将企业 Bean JAR 文件部署到应用服务器，或者将它包含在企业存档(EAR)文件中，并使用该应用进行部署。您还可以在 Web 存档(WAR)文件中以及 Web 应用中部署企业 bean。

1.4. 企业 BEAN 业务接口

Jakarta Enterprise Beans 业务接口是由 bean 开发人员编写的 Java 界面，提供会话 Bean 公共方法的声明以供客户使用。会话 bean 可以实施任意数量的接口，包括 none（*无接口* Bean）。

业务接口可以声明为本地或远程接口，但不能同时声明两者。

Jakarta Enterprise Bean 本地业务接口
Jakarta Enterprise Beans 本地业务接口声明 Bean 和客户端在同一 JVM 时可用的方法。当会话 Bean 实施本地业务接口时，该接口中声明的方法仅供客户端使用。

Jakarta Enterprise Bean 远程业务接口
Jakarta Enterprise Beans 远程业务接口声明了可供远程客户端使用的方法。对实施远程接口的会话 Bean 的远程访问由 Jakarta Enterprise Beans 容器自动提供。

远程客户端是在其他 JVM 中运行的任何客户端，可以包含桌面应用，以及 Web 应用、服务和部署到不同应用服务器的企业 bean。

本地客户端可以访问远程业务接口公开的方法。

Jakarta Enterprise Beans No-interface Beans
不实施任何业务接口的会话 bean 称为 no-interface bean。本地客户端可以访问无接口 Bean 的所有公共方法。

实施业务接口的会话 bean 也可以编写来公开 *no-interface* 视图。

1.5. 旧 EJB 客户端兼容性

JBoss EAP 提供 Jakarta 企业 Beans 客户端库，作为调用远程 Jakarta 企业 Beans 组件的主要 API。

从 JBoss EAP 7.1 开始，提供两个企业 Bean 客户端：

- 企业 Bean 客户端：常规企业 Bean 客户端不完全向后兼容。

- 旧版 EJB 客户端：传统 EJB 客户端提供二进制向后兼容性。这一传统的 EJB 客户端可以使用最初使用 JBoss EAP 7.0 中的 EJB 客户端编译的客户端应用运行。JBoss EAP 7.0 的 EJB 客户端中存在的所有 API 都位于用于 JBoss EAP 7.4 的传统 EJB 客户端中。

您可以通过在配置中包含以下 Maven 依赖项来使用旧的 EJB 客户端兼容性：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.eap</groupId>
      <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
      <version>EAP_BOM_VERSION</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.jboss</groupId>
    <artifactId>jboss-ejb-client-legacy</artifactId>
  </dependency>
</dependencies>
```

您必须使用 JBoss *EAP Maven 存储库中提供的 EAP_BOM_VERSION*。

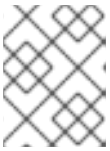
第 2 章 创建企业 BEAN 项目

2.1. 使用红帽 CODEREADY STUDIO 创建 JAKARTA ENTERPRISE BEANS ARCHIVE PROJECT

该任务描述了如何在 Red Hat CodeReady Studio 中创建 Jakarta Enterprise Beans 项目。

先决条件

- 红帽 CodeReady Studio 中已配置了 JBoss EAP 的服务器和服务器运行时。



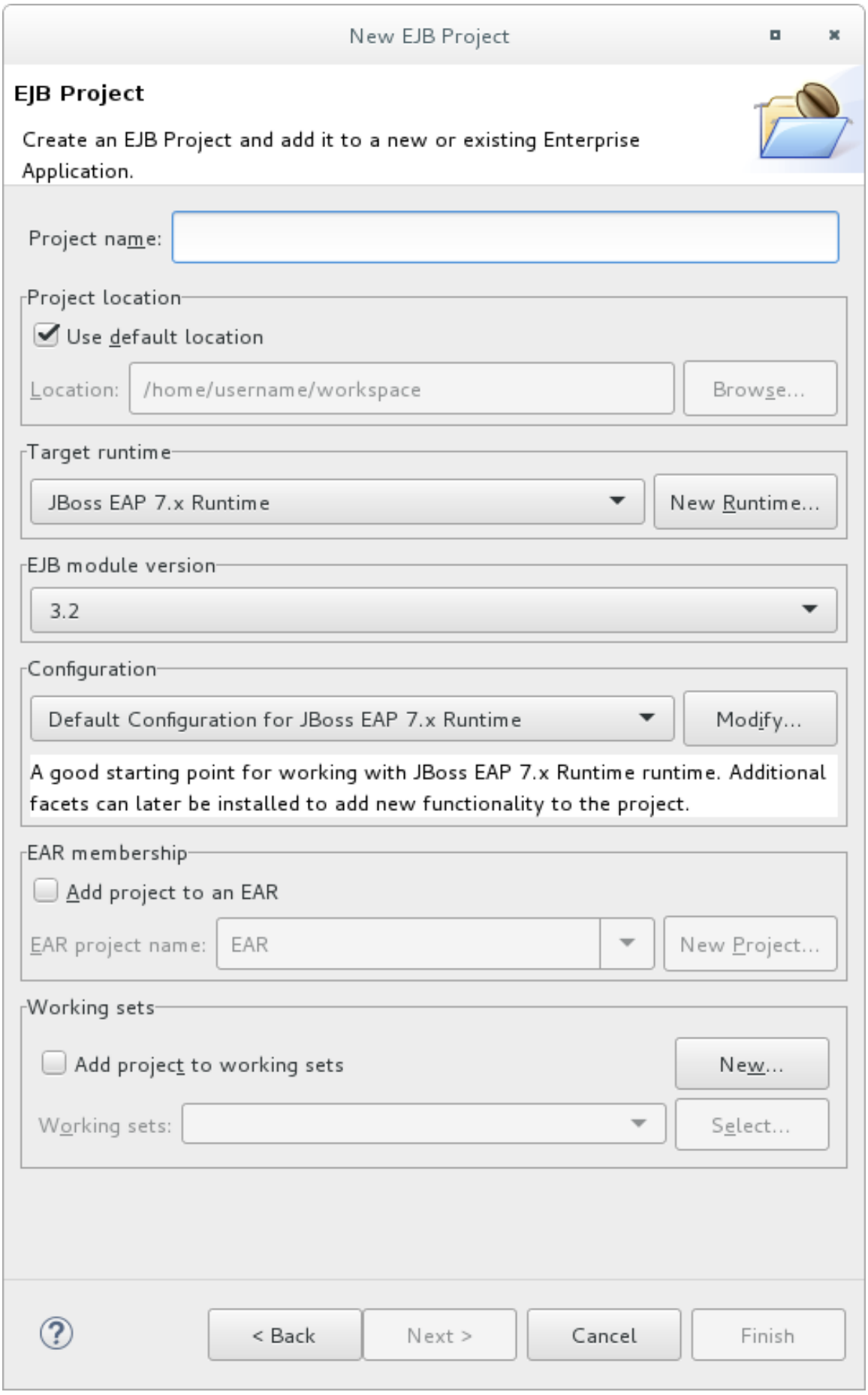
注意

如果您在 Red Hat CodeReady Studio 中将 **Target runtime** 设为 **7.4** 或更新的运行时版本，则您的项目与 Jakarta EE 8 规范兼容。

在 Red Hat CodeReady Studio 中创建 Jakarta Enterprise Beans Project

1. 打开 **New EJB 项目** 向导。
   a. 导航到 **File** 菜单，选择 **New**，然后选择 **Project**。
   b. 出现 **New Project** 向导时，选择 **EJB/EJB Project**，再单击 **Next**。

图 2.1. 新的 EJB 项目向导



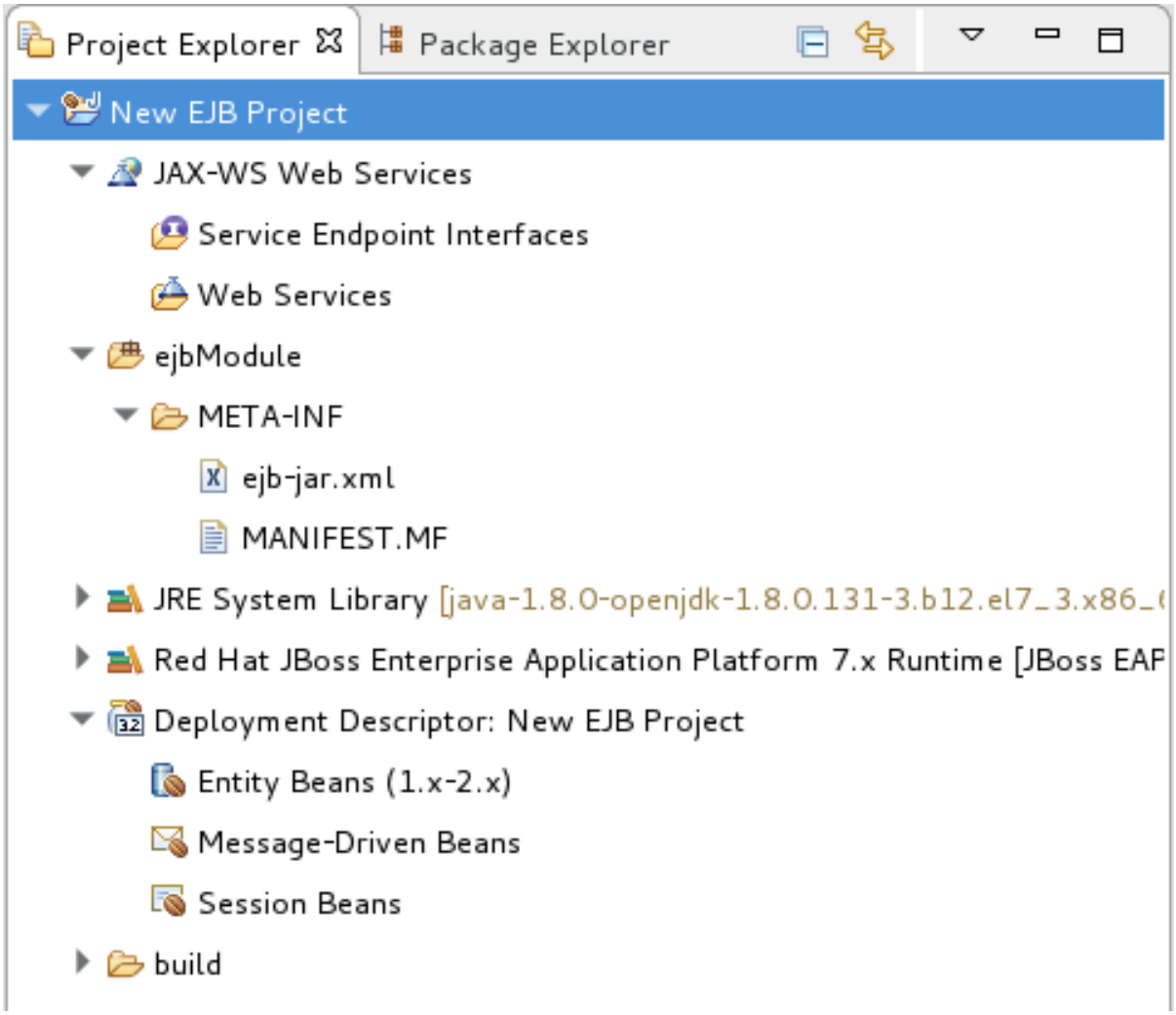
2. 输入以下详情：

- **Project name：** 出现在 Red Hat CodeReady Studio 中的项目名称，以及所部署的 JAR 文件的默认文件名。
- **项目位置：** 保存项目文件的目录。默认为当前工作区中的一个目录。
- **目标运行时：** 这是用于项目的服务器运行时。这将需要设置为您要部署到的服务器的相同 **JBoss EAP** 运行时。
- **EJB 模块版本：** 这是您的企业 Bean 将遵守的 Jakarta 企业 Bean 规范的版本。红帽建议使用 3.2。
- **配置：** 这可让您调整项目中支持的功能。为所选运行时使用默认配置。
  单击 **Next** 以继续。

3. **Java** 项目配置屏幕允许您添加包含 Java 源文件的目录，并指定构建输出的目录。
   保留此配置不变，然后单击" **下一步"。**

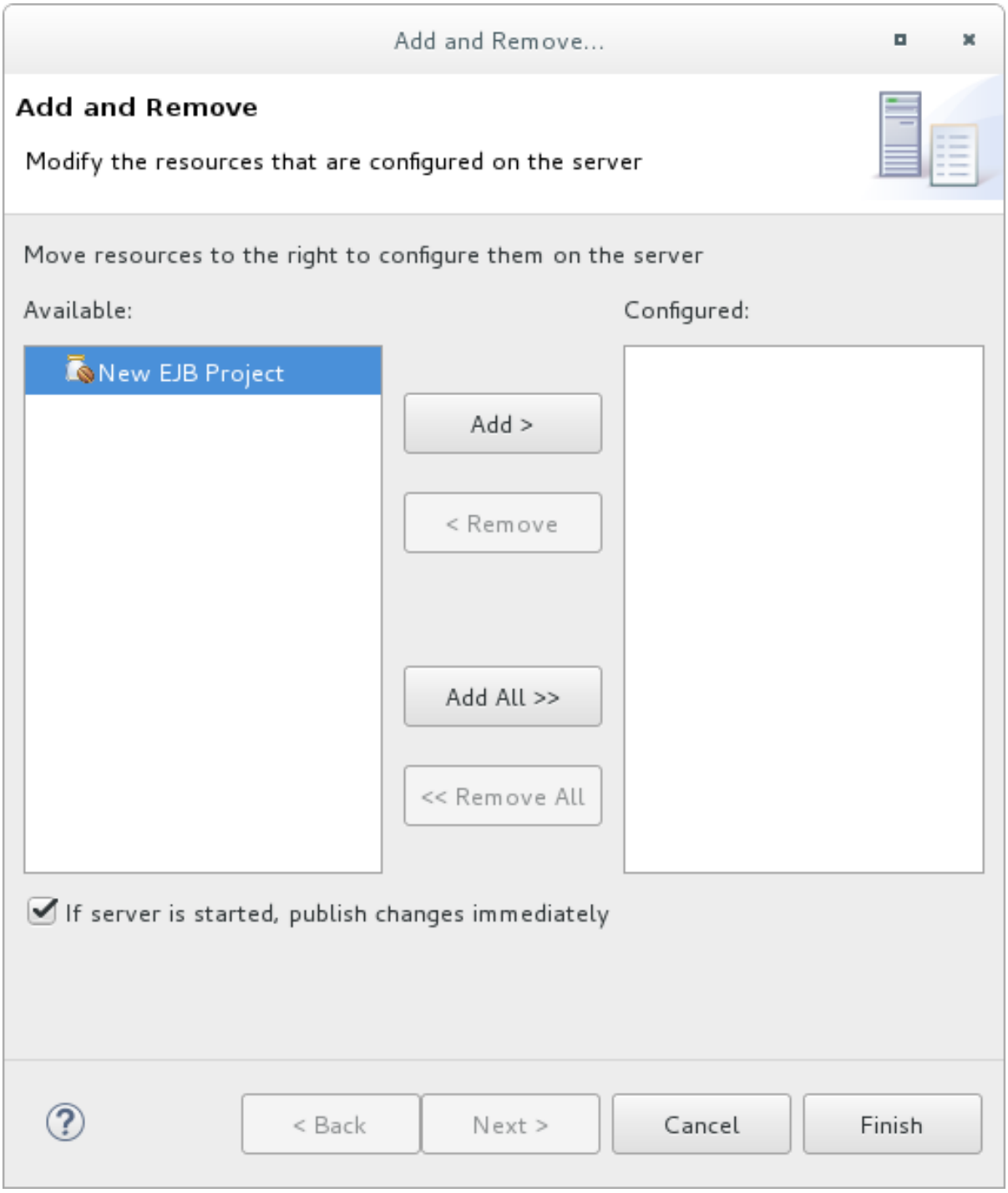
4. 在 **EJB 模块** 设置屏幕中，如果需要 **部署描述符，请检查 Generate ejb-jar.xml** 部署描述符。部署描述符在 Jakarta Enterprise Beans 3.2 中是可选的，必要时可在以后添加。
   单击 **Finish**，项目已创建好，并将显示在 Project Explorer 中。

   图 2.2. Project Explorer 中新创建的 Jakarta Enterprise Beans 项目



5. 要将项目添加到服务器以进行部署，请在 **Servers** 选项卡中右键单击目标服务器，然后选择" **添加和删除** "。
   在 **添加和删除对话框中**，从 **Available** 列中选择要部署的资源，然后单击 **添加** 按钮。资源将移到 **Configured** 列。单击 **Finish** 以关闭该对话框。

图 2.3. 添加和删除对话框



您现在在红帽 CodeReady Studio 中有一个 Jakarta Enterprise Beans 项目，可以构建并部署到指定的服务器。

警告

如果没有企业 Bean 添加到项目中，则红帽代码Ready Studio 将显示警告，表示 ***An EJB 模块必须包含一个或多个企业 Bean。*** 当一个或多个企业 Bean 添加到项目中后，这一警告将消失。

2.2. 在 MAVEN 中创建 JAKARTA ENTERPRISE BEANS ARCHIVE PROJECT

此任务演示了如何使用 Maven 创建包含打包在 JAR 文件中的一个或多个企业 beans 的项目。

先决条件

- 已安装 Maven。
- 您已了解 Maven 的基本用途。

在 Maven 中创建 Jakarta Enterprise Beans Archive Project

1. **创建 Maven 项目：** 可以利用 Maven 的 archetype 系统和 **ejb-javaee7** archetype 创建 Jakarta Enterprise Beans 项目。要执行此操作，请使用参数运行 **mvn** 命令，如下所示：

```
$ mvn archetype:generate -DarchetypeGroupId=org.codehaus.mojo.archetypes -
DarchetypeArtifactId=ejb-javaee7
```

Maven 将提示您输入项目的 **groupId**、**artifactId**、**版本** 和 **软件包**。

```
$ mvn archetype:generate -DarchetypeGroupId=org.codehaus.mojo.archetypes -
DarchetypeArtifactId=ejb-javaee7
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] Archetype [org.codehaus.mojo.archetypes:ejb-javaee7:1.5] found in catalog remote
Define value for property 'groupId': : com.shinysparkly
Define value for property 'artifactId': : payment-arrangements
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  com.shinysparkly: :
Confirm properties configuration:
groupId: com.company
artifactId: payment-arrangements
version: 1.0-SNAPSHOT
package: com.company.collections
Y: :
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 32.440s
[INFO] Finished at: Mon Oct 31 10:11:12 EST 2011
[INFO] Final Memory: 7M/81M
[INFO] ------------------------------------------------------------------------
[localhost]$
```

2. **添加您的企业 Bean：** 编写企业 Bean，并将它们添加到 bean 软件包相应子目录下的 **src/main/java** 目录下。

3. **构建项目：** 要构建项目，请在与 **pom.xml** 文件相同的目录中运行 **mvn package** 命令。这将编译 Java 类并打包 JAR 文件。内置的 JAR 文件名为 **-.jar**，并放置在 **target/** 目录中。

您现在有一个 Maven 项目，用于构建和打包 JAR 文件。此项目可以包含企业 Bean，并且 JAR 文件可以部署到应用服务器。

2.3. 创建包含 JAKARTA ENTERPRISE BEANS 项目的 EAR 项目

此任务描述了如何在包含 Jakarta Enterprise Beans 项目的红帽 CodeReady Studio 中创建一个新的企业存档(EAR)项目。

先决条件

- JBoss EAP 的服务器和服务器运行时已经设置。

注意

如果您在 Red Hat CodeReady Studio 中将 **Target runtime** 设为 **7.4** 或更新的运行时版本，则您的项目与 Jakarta EE 8 规范兼容。

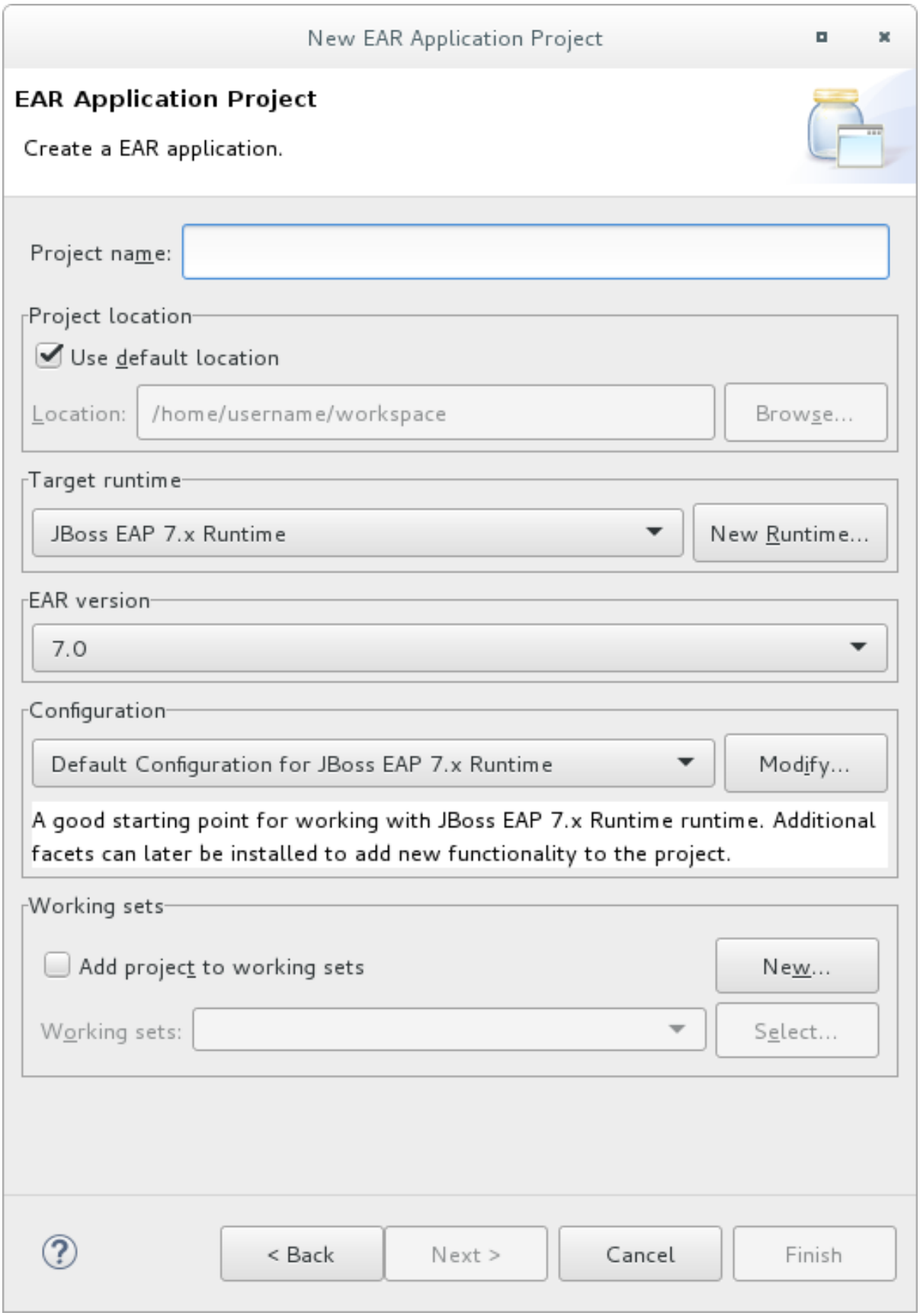
创建包含 Jakarta Enterprise Beans 项目的 EAR 项目

1. 打开 **新的 Java EE EAR 项目**向导.

   a. 导航到 **File** 菜单，选择 **New**，然后选择 **Project**。

   b. 出现 **New Project** 向导时，选择 **Java EE/Enterprise Application Project**，再单击 **Next**。

图 2.4. 新的 EAR 应用项目向导



2. 输入以下详情：

- **项目名称：** 显示在 Red Hat CodeReady Studio 中的项目名称，以及所部署 EAR 文件的默认文件名。
- **项目位置：** 保存项目文件的目录。默认为当前工作区中的一个目录。

- **目标运行时：** 这是用于项目的服务器运行时。这将需要设置为您要部署到的服务器的相同 JBoss EAP 运行时。

- **EAR 版本：** 这是项目将遵守的 Jakarta EE 8 规范的版本。
红帽建议使用 Jakarta EE 8。

- **配置：** 这可让您调整项目中支持的功能。为所选运行时使用默认配置。
单击 **Next** 以继续。

3. 添加新的 Jakarta Enterprise Beans 模块。
可在向导的 **Enterprise Application** 页面中添加新模块。要添加新的 Jakarta Enterprise Beans 项目，请按照以下步骤操作：

   a. 单击 **New Module**，取消选中 **Create Default Modules** 复选框，选中 **Enterprise Java Bean** 并单击 **Next**。这时 **将显示 New EJB 项目向导**。

   b. **New EJB 项目向导** 与用于创建新的独立 Jakarta Enterprise Beans Projects 的向导相同，在使用红帽 CodeReady Studio 的 Create Jakarta Enterprise Beans Archive Project 中进行介绍。
   创建项目所需的最小详情为：

      - 项目名称
      - 目标运行时
      - Jakarta Enterprise Beans 模块版本
      - Configuration
      向导的所有其他步骤都是可选的。单击 **Finish** 以完成创建 Jakarta Enterprise Beans 项目。

      新创建的 Jakarta Enterprise Beans 项目列在 Java EE 模块依赖项中，并选中复选框。

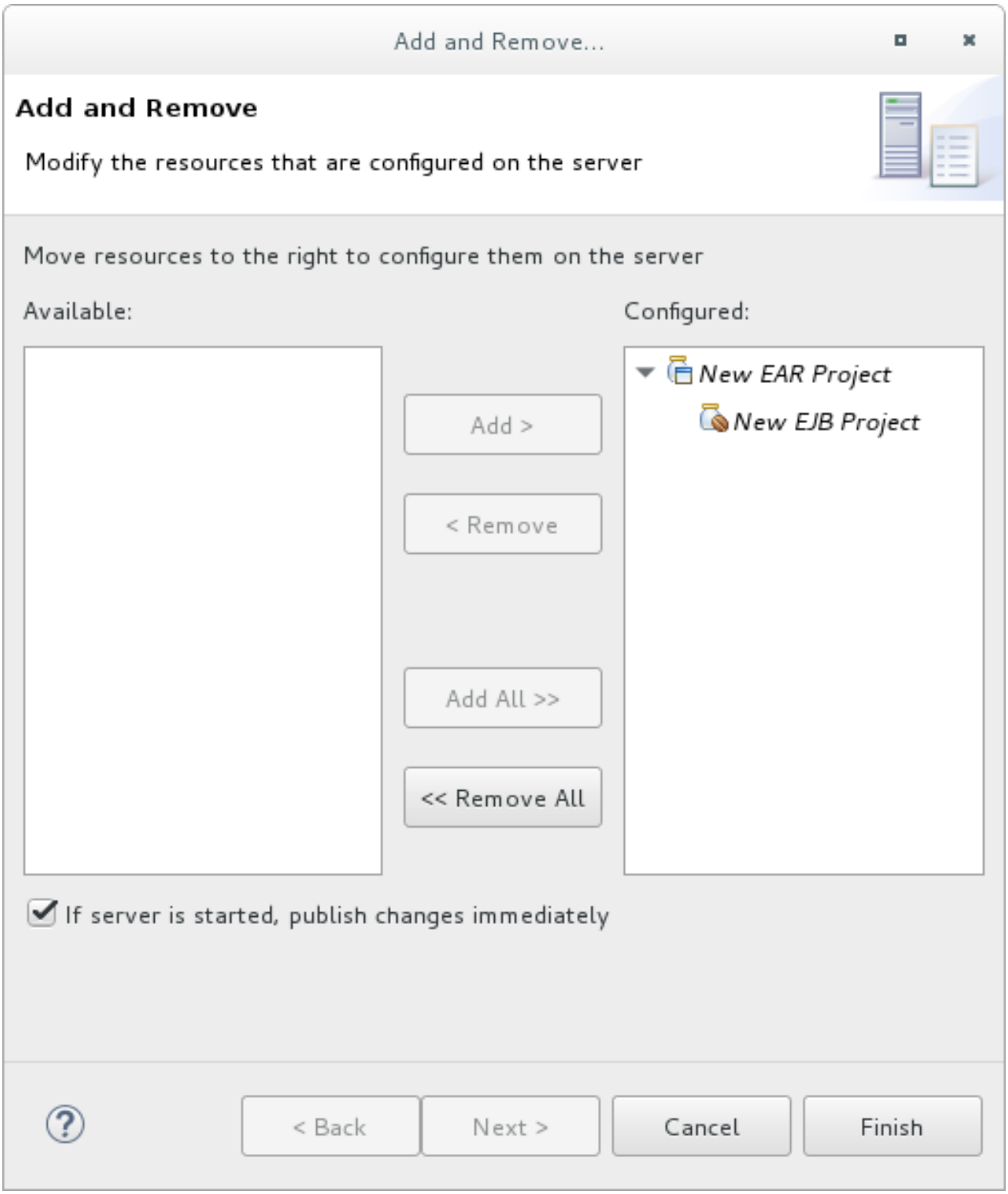
4. （可选）添加 **application.xml 部署** 描述符。
如果需要，选中 **Generate application.xml 部署描述符** 复选框。

5. 点 **Finish**。
这时将显示两个新项目：Jkarta Enterprise Beans 项目和 EAR 项目。

6. 将构建构件添加到要部署的服务器。
在 server 选项卡中，**右键单击要将构建构件部署到 的服务器上的 Servers 选项卡，然后选择"添加和删除"，以打开" 添加和删除"对话框。**

   从 **Available** 列选择要部署的 EAR 资源，然后单击 **Add** 按钮。资源将移到 Configured 列。单击 **Finish** 以关闭该对话框。

图 2.5. 添加和删除对话框



您现在有一个企业应用项目，其中包含了 Jakarta Enterprise Beans 项目的成员。这将作为包含 Jakarta Enterprise Beans 子部署的单个 EAR 部署来构建和部署到指定的服务器。

2.4. 将 DEPLOYMENT 描述符添加到 JAKARTA ENTERPRISE BEANS 项目中

Jakarta Enterprise Beans 部署描述符可以添加到 Jakarta Enterprise Beans 项目中，该项目创建时没有一个。要做到这一点，请按照以下步骤操作。

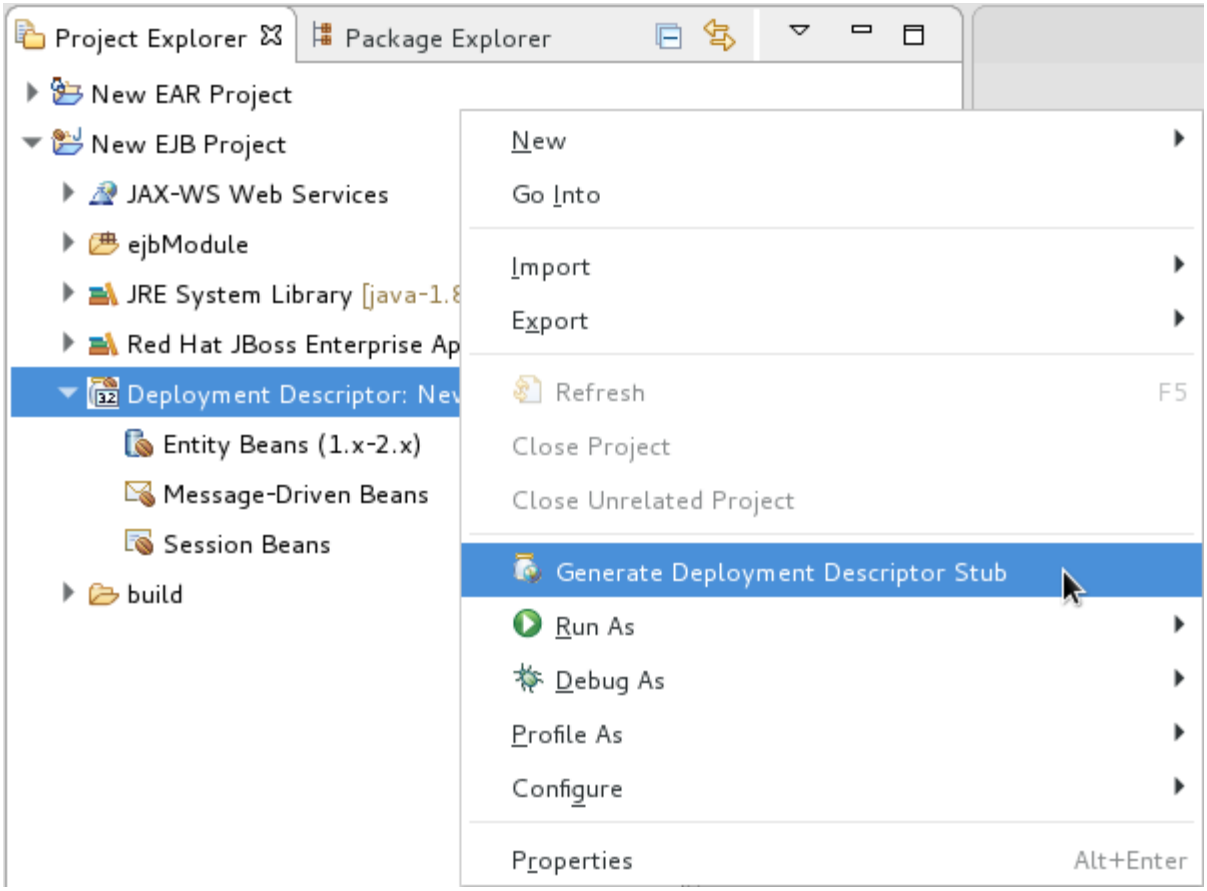
先决条件

- 您在红帽 CodeReady Studio 中有一个 Jakarta Enterprise Beans 项目，您要添加 Jakarta Enterprise Beans 部署描述符。

将 Deployment 描述符添加到 Jakarta Enterprise Beans 项目中

1. 打开 Red Hat CodeReady Studio 中的项目。
2. 添加部署描述符。
   右键单击项目视图中的 **Deployment Descriptor** 文件夹，然后选择 **Generate Deployment Descriptor** 选项卡。

   图 2.6. 添加部署描述符



   新文件 **ejb-jar.xml** 在 **ejbModule/META-INF/** 中创建。双击项目视图中 **的 Deployment Descriptor** 文件夹，以打开此文件。

2.5. BEAN 的运行时部署信息

您可以将运行时部署信息添加到 bean 中以进行性能监控。

有关可用运行时数据的详情，请查看 JBoss EAP 管理模型中的 **ejb3** 子系统。应用程序可以将运行时数据包含为 bean 代码或部署描述符中的注解。应用可以同时使用这两个选项。

其他资源

- 有关可用运行时数据的更多信息，请参见 JBoss EAP 管理模型 中的 **ejb3** 子系统。
- 有关检索用于评估性能的运行时数据的更多信息，请参阅 JBoss EAP 性能调优指南。

第 3 章 会话组件

3.1. 会话 BEAN

会话 Bean 是企业 Bean，它封装了一组相关的业务流程或任务，并注入到请求它们的类中。会话 Bean 有三种类型：无状态、有状态和单例。

3.2. 无状态会话 BEAN

无状态会话 Bean 是最为简单但使用最广泛的会话 Bean 类型。它们向客户端应用提供业务方法，但不在方法调用之间维护任何状态。每种方法都是一个完整的任务，不依赖于该会话 Bean 内的任何共享状态。由于没有状态，应用服务器不需要确保在同一实例上执行每个方法调用。这使得无状态会话变得非常高效且可扩展。

3.3. 有状态会话 BEAN

有状态会话 Bean 是企业 Bean，为客户端应用提供业务方法，并与客户端保持对话状态。它们应当用于必须在几个步骤或方法调用中执行的任务，每个任务都依赖于要维护的上一步骤的状态。应用服务器确保每个客户端针对每个方法调用接收同一有状态会话 Bean 实例。

3.4. 单例会话 BEAN

单例会话 Bean 是每个应用实例化一次会话 Bean，对单例 Bean 的每个客户端请求发送到同一实例。Singleton Bean 是 Singleton *Design Patterns 的一个实施过程，如《设计模式》一书所述：Erich Gamma*、*Richard Helm*、*Ralph Johnson 和 John Vlissides 发布，由* Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 发布。

单例 Bean 提供所有会话 Bean 类型的最小内存占用量，但必须设计为线程安全。Jakarta Enterprise Beans 3.2 提供容器管理的并发(CMC)，使开发人员能够轻松地实施线程安全单例 bean。但是，如果 CMC 不提供足够的灵活性，则可以使用传统的多线程代码（bean 管理的并发代码或 BMC）编写单例 bean。

3.5. 将 SESSION BEANS 添加到 RED HAT CODEREADY STUDIO 中的项目

Red Hat CodeReady Studio 具有多个向导，可用于快速创建企业 Bean 类。以下流程演示了如何使用 Red Hat CodeReady Studio 向导将会话 Bean 添加到项目中。

先决条件

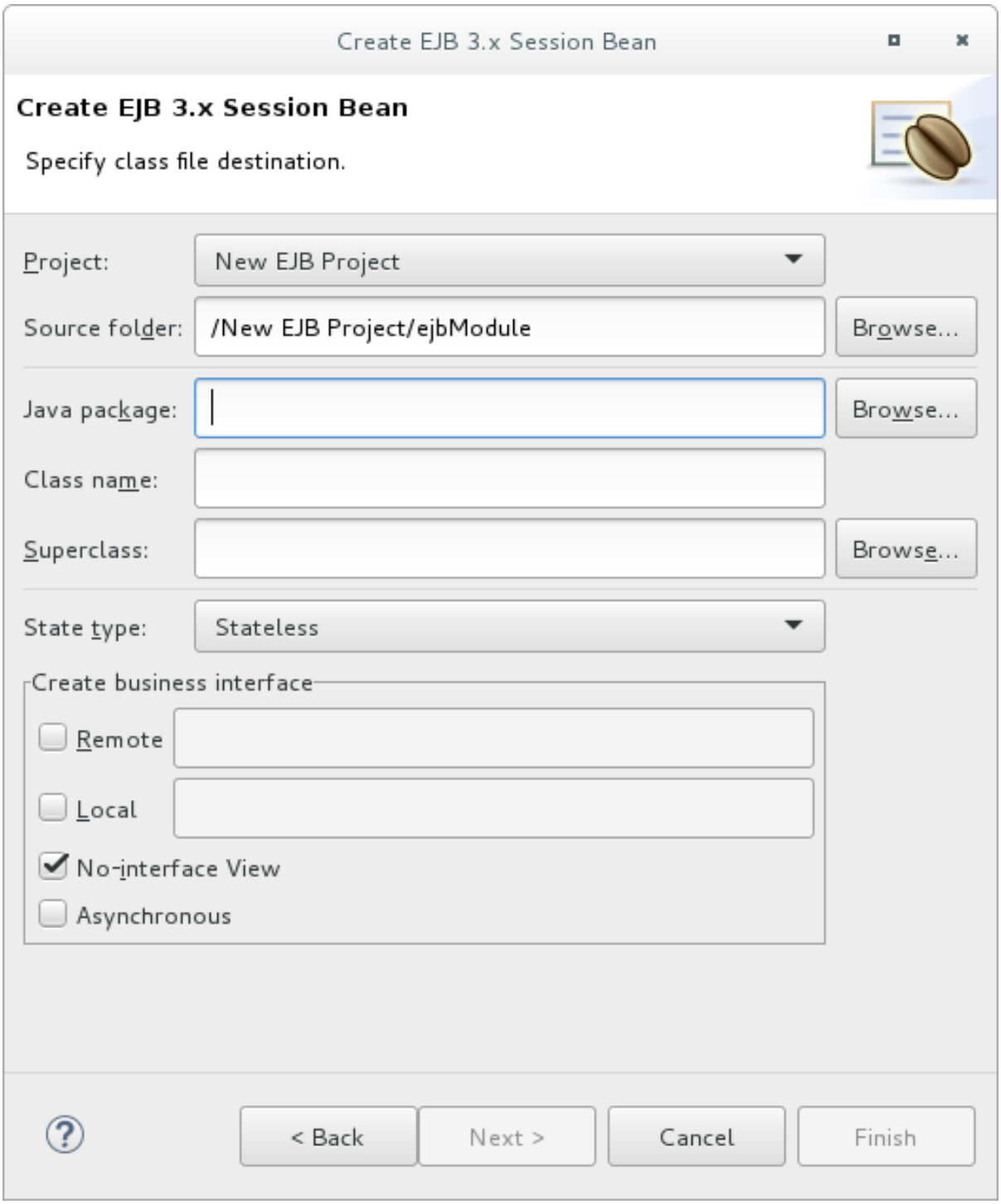
- 在红帽 CodeReady Studio 中，您有一个 Jakarta Enterprise Beans 或 Dynamic Web Project，您要添加一个或多个会话 Bean。

将 Session Beans 添加到 Red Hat CodeReady Studio 中的项目

1. 打开 Red Hat CodeReady Studio 中的项目。

2. 打开 Create EJB 3.x Session Bean 向导。
   要打开 **Create EJB 3.x Session Bean 向导**，导航到 **File** 菜单，选择 **New**，然后选择 **Session Bean(EJB 3.x)**。

图 3.1. 创建 EJB 3.x Session Bean 向导



3. 指定以下详情：

- **项目：** 验证是否已选中正确的项目。
- **源文件夹：** 这是在其中创建 Java 源文件的文件夹。这通常不需要更改。
- package **：指定该类所属的软件包。**
- **类名称：** 指定将成为会话 Ban 的类的名称。
- **超级类：** 会话 Bean 类可以从超级类继承。请在此处指定会话具有超级类。
- **State type：** 指定会话 Bean 的状态类型：无状态、有状态或单例。

- **业务接口：** 默认情况下，选中"不 **接口** "框，因此不会创建接口。选中您想要定义和调整名称的接口框。
  请记住，Web 存档(WAR)中的企业 Bean 仅支持 Jakarta Enterprise Beans 3.2 Lite，这不包括远程业务接口。

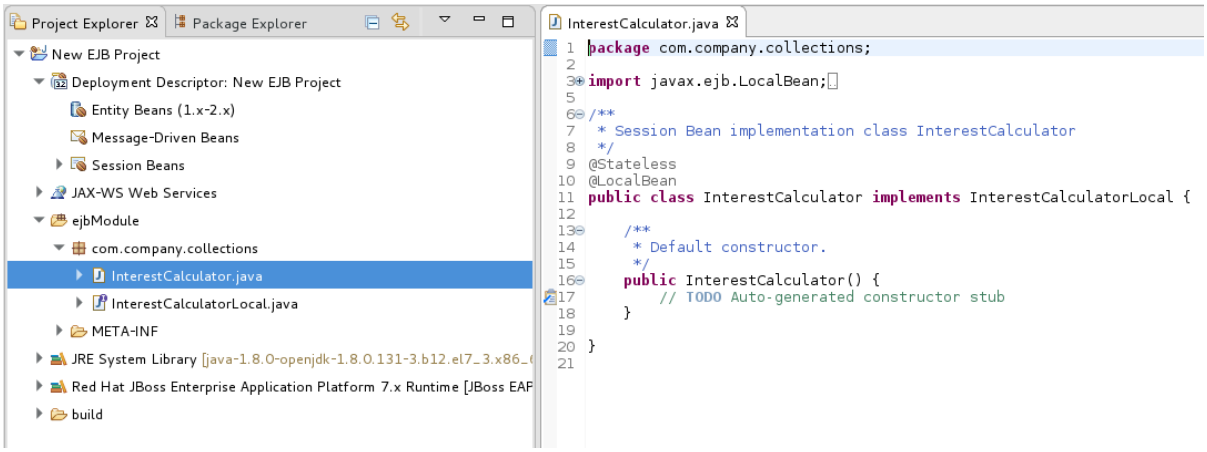
  点 **Next**。

4. 您可以在此处输入其他信息，以进一步自定义会话 Bean。此处不需要更改任何信息。
   您可以更改的项目有：

   - bean 名称
   - 映射名称
   - 事务类型（托管或 bean 管理）
   - 可以提供 bean 必须实施的额外接口
   - 如果需要，您还可以指定 EJB 2.x Home 和 组件接口

5. 单击 **Finish**，新会话 bean 将创建并添加到项目中。如果已指定任何新业务接口的文件，也将创建它们。

图 3.2. Red Hat CodeReady Studio 中的新 Session Bean



第 4 章 消息驱动 BEANS

4.1. 消息驱动 BEANS

消息驱动型 Bean(MDB)为应用程序开发提供事件驱动的模型。MDB 的方法不注入或从客户端代码调用，而是由接收来自 Jakarta 消息传递服务器等消息传递服务的消息触发。Jakarta EE 规范要求支持 Jakarta 消息传递，但也支持其他消息传递系统。

MDB 是一种特殊的无状态会话 Bean。它们实施名为 **onMessage（Message 消息）** 的方法。当 MDB 正在侦听的 Jakarta 消息目的地收到消息时，会触发此方法。也就是说，MDB 是由从 Jakarta 消息传递提供商接收消息触发的，这与无状态会话 Bean 不同，其中方法通常由 Jakarta 企业 Bean 客户端调用。

MDB 异步处理消息.默认情况下，每个 MDB 最多可以有 16 个会话，每个会话处理一个消息。没有消息顺序保证。若要实现消息排序，需要将 MDB 的会话池限制为 **1**。

示例：管理CLI命令将会话池 设置为 1:

```
/subsystem=ejb3/strict-max-bean-instance-pool=mdb-strict-max-pool:write-attribute(name=derive-size,value=undefined)

/subsystem=ejb3/strict-max-bean-instance-pool=mdb-strict-max-pool:write-attribute(name=max-pool-size,value=1)

reload
```

4.2. 消息驱动 BEANS CONTROLLED DELIVERY

JBoss EAP 提供三个属性来控制在特定 MDB 上主动接收消息：

- delivery Active
- 交付组
- 集群的单例 MDB

4.2.1. delivery Active

消息驱动型 Bean(MDB)的发送活动配置指示 MDB 是否在接收消息。如果 MDB 未接收消息，则消息将根据主题或队列规则保存在队列或主题中。

您可以使用 XML 或注释配置 **delivery-group** 的 **active** 属性，您可以在部署后使用管理 CLI 更改其值。默认情况下激活 **active** 属性，并在部署 MDB 后立即发送消息。

在 jboss-ejb3.xml 文件中配置交付活动
在 **jboss-ejb3.xml** 文件中，将 **active** 的 值设置为 **false**，以表示 MDB 在部署后不会立即收到消息：

```
<?xml version="1.1" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:d="urn:delivery-active:1.1"
    xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
    version="3.1"
```

```
    impl-version="2.0">
    <assembly-descriptor>
        <d:delivery>
            <ejb-name>HelloWorldQueueMDB</ejb-name>
            <d:active>false</d:active>
        </d:delivery>
    </assembly-descriptor>
</jboss:ejb-jar>
```

如果要将活跃值应用到应用程序中的所有 MDB，您可以使用通配符 * 来代替 **ejb-name**。

使用注解配置交付活跃
您还可以使用 **org.jboss.ejb3.annotation.DeliveryActive** 注解。例如：

```
@MessageDriven(name = "HelloWorldMDB", activationConfig = {
 @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
 @ActivationConfigProperty(propertyName = "destination", propertyValue =
"queue/HELLOWORLDMDBQueue"),
 @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge") })
@DeliveryActive(false)

public class HelloWorldMDB implements MessageListener {
    public void onMessage(Message rcvMessage) {
      // ...
    }
}
```

如果使用 Maven 构建项目，请确保在项目的 **pom.xml** 文件中添加以下依赖项：

```
<dependency>
    <groupId>org.jboss.ejb3</groupId>
    <artifactId>jboss-ejb3-ext-api</artifactId>
    <version>2.2.0.Final</version>
</dependency>
```

使用管理 CLI 配置交付活跃
在部署后，您可以使用管理 CLI 配置 **delivery-group** 的 **active** 属性。这些管理操作动态更改 **active** 属性的值，为 MDB 启用或禁用交付。如果重新启动服务器，这种更改交付活跃值的方法不会保留。在运行时，连接到您要管理的实例，然后输入您要管理的 MDB 的路径。例如：

- 进入您要管理的实例：

  ```
  cd deployment=helloworld-mdb.war/subsystem=ejb3/message-driven-
  bean=HelloWorldQueueMDB
  ```

- 停止发送到 MDB:

  ```
  :stop-delivery
  ```

- 开始交付到 MDB:

  ```
  :start-delivery
  ```

查看 MDB Delivery Active 状态
您可以使用管理控制台查看任何 MDB 的当前交付活跃状态：

1. 选择 **Runtime** 选项卡，然后选择相应的服务器。
2. 单击 **EJB**，再选择子资源，如 **HelloWorldQueueMDB**。

结果

您看到的状态为 **Delivery Active: true 或 Delivery Active: false**。

4.2.2. 交付组

交付组提供了一种管理一组 MDB **的交付-主动** 状态的方法。MDB 可以属于一个或多个交付组。只有 MDB 所属的所有交付组都处于活动状态时，才会启用消息发送。对于集群单例 MDB，消息发送仅在集群的单例节点中活跃，只有在与 MDB 关联的所有交付组都处于活动状态时。

您可以使用 XML 配置或管理 CLI 将交付组添加到 **ejb3** 子系统。

在 jboss-ejb3.xml 文件中配置 Delivery Group

```
<delivery>
  <ejb-name>MdbName<ejb-name>
  <delivery-group>passive</delivery-group>
</delivery>
```

在服务器端，可以通过将其 **active** 属性设置为 **true 来** 启用 **delivery-groups**，或者将其 **active** 属性设置为 **false** 来禁用，如下例所示：

```
<delivery-groups>
  <delivery-group name="group" active="true"/>
</delivery-groups>
```

使用管理 CLI 配置交付组
可以使用管理 CLI 更新 **交付组** 状态。例如：

```
/subsystem=ejb3/mdb-delivery-group=group:add
/subsystem=ejb3/mdb-delivery-group=group:remove
/subsystem=ejb3/mdb-delivery-group=group:write-attribute(name=active,value=true)
```

当您在 **jboss-ejb3.xml** 文件中设置了活动的交付或使用注释时，它会在服务器重新启动时持久保留。但是，当使用管理 CLI 停止或启动交付时，它不会在服务器重启时持久保留。

使用注解配置多个交付组
您可以在属于某个组的每个 MDB 类上使用 **org.jboss.ejb3.annotation.DeliveryGroup** 注解：

```
@MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
      @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
      @ActivationConfigProperty(propertyName = "destination", propertyValue =
"queue/HELLOWORLDMDBQueue"),
      @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge") })
@DeliveryGroup("delivery-group-1")
@DeliveryGroup("delivery-group-2")
```

```
public class HelloWorldQueueMDB implements MessageListener {
...
}
```

4.2.3. 集群的单例 MDB

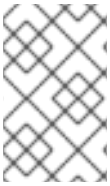
当 MDB 被识别为集群单例并部署到集群中时，只有一个节点处于活跃状态。此节点可以按顺序使用消息。当服务器节点出现故障时，来自集群单例 MDB 的活动节点会开始使用消息。

将 MDB 识别为集群的单例
您可以使用以下步骤之一将 MDB 识别为集群单例。

- 使用 clustered-singleton XML 元素，如下例所示：

```
<?xml version="1.1" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
               xmlns="http://java.sun.com/xml/ns/javaee"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:c="urn:clustering:1.1"
               xmlns:d="urn:delivery-active:1.2"
               xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
               version="3.1"
               impl-version="2.0">
   <assembly-descriptor>
      <c:clustering>
         <ejb-name>HelloWorldQueueMDB</ejb-name>
         <c:clustered-singleton>true</c:clustered-singleton>
      </c:clustering>
       <d:delivery>
            <ejb-name>*</ejb-name>
            <d:group>delivery-group-1</d:group>
            <d:group>delivery-group-2</d:group>
      </d:delivery>
   </assembly-descriptor>
</jboss:ejb-jar>
```

- 在 MDB 类中，使用 **@org.jboss.ejb3.annotation.ClusteredSingleton**。此过程不需要额外的服务器配置。您需要在集群环境中运行该服务。



注意

您必须激活整个集群中的 **delivery-group**，特别是集群中的所有节点，因为您不知道哪个集群的节点被选为 **单例 master**。如果服务器选择要成为 **单例 master** 的节点，并且该节点没有激活所需的 **delivery-group**，则集群中没有节点接收消息。

JBoss EAP 随附 **的 messaging-clustering-singleton** 快速入门演示了如何将集群与 Apache ActiveMQ Artemis 集成使用。它使用与 **helloworld-mdb** 快速启动相同的源代码，只有配置有差别才能将它作为集群单例运行。此快速入门中包含两种 Jakarta 消息传递资源：

- 在 Java 命名和目录接口中绑定名为 **HELLOWORLDMDBQueue 的** 队列，作为 **java:/queue/HELLOWORLDMDBQueue**

- Java 命名和目录界面中名为 **HELLOWORLDMDBTopic** 的主题，作为 **java:/topic/HELLOWORLDMDBTopic** 绑定

这两个配置都包含 **jboss-ejb3.xml** 文件中指定的单例配置：

```
<c:clustering>
   <ejb-name>*</ejb-name>
   <c:clustered-singleton>true</c:clustered-singleton>
</c:clustering>
```

<ejb-name> 元素中的通配符 **星号 *** 表示应用程序中包含的所有 MDB 都将被集群化。因此，集群中只有一个节点会在特定时间激活这些 MDB。如果此活跃节点关闭，集群中的另一个节点将成为 MDB 的活动节点，然后变为单例供应商。

您还可以在 **jboss-ejb3.xml** 文件中找到交付组的配置：

```
<d:delivery>
   <ejb-name>HelloWorldTopicMDB</ejb-name>
   <d:group>my-mdb-delivery-group</d:group>
</d:delivery>
```

在这种情况下，只有一个 MDB **HelloWorldTopicMDB** 与交付组关联。必须在 **ejb3** 子系统配置中配置 MDB 使用的所有交付组。可以启用或禁用交付组。如果在集群节点中禁用了交付组，属于该交付组的所有 MDB 分别在相应的群集节点中不可用。在非集群环境中使用交付组时，每当启用交付组时 MDB 就会活跃。

如果交付组与单例提供程序结合使用，只有在该节点启用了交付组时，MDB 才能在单例供应商节点中处于活动状态。否则，MDB 将在那个节点中不活跃，以及集群中的所有其他节点。

有关如何为消息传递集群配置服务器并查看代码示例，请参阅此快速入门中包含的 **README.html** 文件。

有关如何下载和使用 JBoss EAP 快速入门的信息，请参见 JBoss EAP 入门 *指南中的使用快速入门示例* 章节。

4.3. 在红帽代码READY STUDIO 中创建一个基于消息的 JAKARTA MESSAGING-DRIVEN BEAN

此流程演示了如何将基于 Jakarta 消息驱动的 Bean 添加到红帽 CodeReady Studio 中的项目。此流程创建一个使用注解的 Jakarta Enterprise Beans 3.x 消息驱动 Bean。

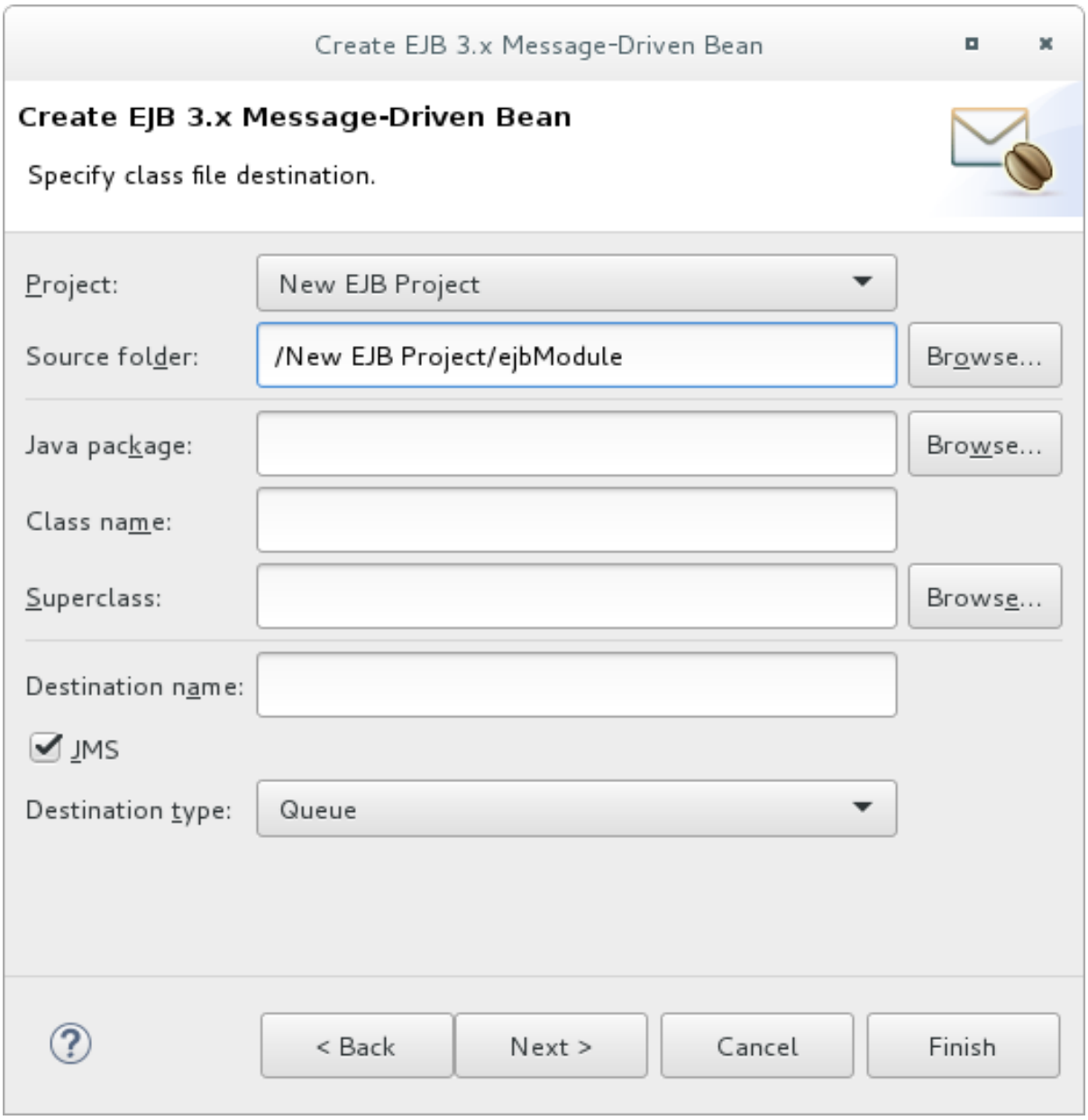
先决条件

- 您必须在 Red Hat CodeReady Studio 中打开一个现有项目。
- 您必须知道 bean 将侦听的 Jakarta 消息目的地的名称和类型。
- 必须在将 Bean 部署到的 JBoss EAP 配置中启用对 Jakarta 消息传递的支持。

在红帽代码Ready Studio 中添加基于消息驱动的 Bean

1. 打开 **Create EJB 3.x Message-Driven Bean**向导。
   前往 **File → New → Other**。选择 **EJB/Message-Driven Bean(EJB 3.x)**，然后单击 **下一步** 按钮。

图 4.1. 创建 EJB 3.x 消息驱动 Bean 向导



2. 指定类文件目标详细信息.
这里可为 bean 类指定三组详细信息：项目、Java 类和消息目的地。

- **项目：**
  - 如果工作区中存在多个项目，请确保在 **Project** 菜单中选择了正确的项目。
  - 将在其中为新 bean 创建源文件的文件夹位于 **所选** 项目的 目录下。只有在您有特定要求时才更改此设置。
- **Java 类：**
  - 必填字段为： **Java 软件包** 和 **类名称**。
  - 不需要提供超级类，除非应用程序的业务逻辑需要它。
- **Message Destination:**
  - 以下是您在基于 Jakarta 消息驱动型 Bean 时必须提供的详细信息：

- **目标名称**，即包含 bean 将响应的邮件的队列或主题名称。
- 默认情况下，选中 **JMS** 复选框。不要更改此设置。
- 根据需要将 **Destination type** 设置为 **Queue** 或 **Topic**。
  单击 **Next** 按钮。

3. 输入消息驱动的 Bean 特定信息。
   此处的默认值适合使用容器管理的事务基于 Jakarta 消息驱动的 Bean。
   - 如果 Bean 将使用 Bean 管理的事务，请将交易 **类型** 更改为 Bean。
   - 如果需要与类名称不同的 **Bean 名称**，则更改 Bean 名称。
   - 将已列出 **JMS Message Listener** 接口。您不需要添加或删除任何接口，除非它们特定于您的应用的业务逻辑。
   - 保留用于创建方法存根的复选框。
     单击 **Finish** 按钮。

结**果**

使用 stub 方法为默认构造器和 **onMessage（）** 方法创建消息驱动型 Bean。将打开 Red Hat CodeReady Studio 编辑器窗口，其中包含对应的 文件。

4.4. 在 JBOSS-EJB3.XML 中为 MDB 指定资源适配器

在 **jboss-ejb3.xml** 部署描述符中，您可以指定要使用的 MDB 资源适配器。

若要在 **jboss-ejb3.xml** 中为 MDB 指定资源适配器，请使用以下示例：

示例：MDB 资源适配器 jboss-ejb3.xml 配置

```
<jboss xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:jee="http://java.sun.com/xml/ns/javaee"
  xmlns:mdb="urn:resource-adapter-binding">
  <jee:assembly-descriptor>
    <mdb:resource-adapter-binding>
      <jee:ejb-name>MyMDB</jee:ejb-name>
      <mdb:resource-adapter-name>MyResourceAdapter.rar</mdb:resource-adapter-name>
    </mdb:resource-adapter-binding>
  </jee:assembly-descriptor>
</jboss>
```

对于位于 EAR 中的资源适配器，您必须对 **<mdb:resource-adapter-name>** 使用以下语法：

- 对于位于另一个 EAR 中的资源适配器：

  ```
  <mdb:resource-adapter-
  name>OtherDeployment.ear#MyResourceAdapter.rar</mdb:resource-adapter-name>
  ```

- 对于与 MDB 位于相同 EAR 的资源适配器，您可以省略 EAR 名称：

  ```
  <mdb:resource-adapter-name>#MyResourceAdapter.rar</mdb:resource-adapter-name>
  ```

4.5. 在 MDB 中使用部署到集群中的资源定义注解

如果您使用 **@JMSConnectionFactoryDefinition** 和 **@JMSDestinationDefinition** 注释来为消息驱动的 Bean 创建连接工厂和目标，请注意对象仅在部署 MDB 的服务器上创建。它们不会在集群中的所有节点上创建，除非 MDB 也部署到集群中的所有节点上。由于这些注解配置的对象仅在部署 MDB 的服务器上创建，这会影响 MDB 从远程服务器读取消息的远程 Jakarta Connectors 拓扑，然后将消息发送到远程服务器。

4.6. 在应用程序中启用 JAKARTA ENTERPRISE BEANS 和 MDB 属性替换

红帽 JBoss 企业应用平台允许您使用 **@ActivationConfigProperty 和 @ Resource** 注释在 Jakarta 企业 Bean 和 MDB 中启用属性替换。属性替换需要以下配置和代码更改：

- 您必须在 JBoss EAP 服务器配置文件中 启用属性替换。
- 您必须在 服务器配置文件中定义系统属性，并在启动 JBoss EAP 服务器时将它们作为参数传递。
- 您必须 修改应用程序代码 以使用替换变量。

以下示例演示了如何修改 JBoss EAP **附带的 helloworld-mdb** 快速入门以使用属性替换。有关已完成的工作示例，请参见 **helloworld-mdb-propertysubative 快速** 入门。

4.6.1. 配置服务器以启用属性替换

要在 JBoss EAP 服务器中启用属性替换，您必须在服务器配置 **的 ee 子系统中将 annotations-property-replacement** 属性设置为 **true**。

1. 备份服务器配置文件。
   helloworld-mdb-propertysubative quick start 示例需要单机服务器的完整配置集，因此这是 ***EAP_HOME*/standalone/configuration/standalone-full.xml** 文件。如果您在受管域中运行服务器，这是 ***EAP_HOME*/domain/configuration/domain.xml** 文件。
2. 导航到 JBoss EAP 安装目录，再使用 full 配置文件启动服务器。

```
$ EAP_HOME/bin/standalone.sh -c standalone-full.xml
```

注意

对于 Windows Server，请使用 ***EAP_HOME*\bin\standalone.bat** 脚本。

3. 启动管理 CLI。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

注意

对于 Windows Server，请使用 ***EAP_HOME*\bin\jboss-cli.bat** 脚本。

4. 键入以下命令以启用注解属性替换：

```
/subsystem=ee:write-attribute(name=annotation-property-replacement,value=true)
```

您应看到以下结果：

```
{"outcome" => "success"}
```

5. 检查对 JBoss EAP 服务器配置文件的更改。The **ee** 子系统现在应包含以下 XML：

示例 ee 子系统配置

```
<subsystem xmlns="urn:jboss:domain:ee:4.0">
  ...
  <annotation-property-replacement>true</annotation-property-replacement>
  ...
</subsystem>
```

4.6.2. 定义系统属性

您可以在服务器配置文件中指定系统属性，也可以在启动 JBoss EAP 服务器时将它们作为命令行参数传递。服务器配置文件中定义的系统属性优先于启动服务器时在命令行中传递的属性。

4.6.2.1. 在服务器配置中定义系统属性

1. 启动管理 CLI。
2. 使用以下命令语法在 JBoss EAP 服务器中配置系统属性：

添加系统属性的语法

```
/system-property=PROPERTY_NAME:add(value=PROPERTY_VALUE)
```

为 **helloworld-mdb-propertysubative 快速入门配置了以下系统属性：**

添加系统属性的命令示例

```
/system-
property=property.helloworldmdb.queue:add(value=java:/queue/HELLOWORLDMDBPropQue
ue)
/system-
property=property.helloworldmdb.topic:add(value=java:/topic/HELLOWORLDMDBPropTopic)
/system-property=property.connection.factory:add(value=java:/ConnectionFactory)
```

3. 检查对 JBoss EAP 服务器配置文件的更改。以下系统属性现在应出现在 **<extensions> 后面**。

系统属性配置示例

```
<system-properties>
    <property name="property.helloworldmdb.queue"
value="java:/queue/HELLOWORLDMDBPropQueue"/>
    <property name="property.helloworldmdb.topic"
value="java:/topic/HELLOWORLDMDBPropTopic"/>
    <property name="property.connection.factory" value="java:/ConnectionFactory"/>
</system-properties>
```

4.6.2.2. 在服务器启动时将系统属性作为参数传递

如果您愿意，可以在您以 **-D*PROPERTY_NAME =PROPERTY_ VALUE*** 的形式启动 JBoss EAP 服务器时在命令行中传递参数。以下是如何为上一节中定义的系统属性传递参数的示例。

服务器启动命令传递系统属性示例

```
$ EAP_HOME/bin/standalone.sh -c standalone-full.xml  -
Dproperty.helloworldmdb.queue=java:/queue/HELLOWORLDMDBPropQueue  -
Dproperty.helloworldmdb.topic=java:/topic/HELLOWORLDMDBPropTopic  -
Dproperty.connection.factory=java:/ConnectionFactory
```

4.6.3. 修改应用程序代码以使用系统属性替代

将硬编码的 **@ActivationConfigProperty** 和 **@Resource** 注释值替换为新定义的系统属性。以下是如何更改 **helloworld-mdb** quickstart 以使用新定义的系统属性替换的示例：

1. 更改 **HelloWorldQueueMDB** 类中的 **@ActivationConfigProperty destination** 属性值，以使用 system 属性替换。**@MessageDriven** 注释现在应如下所示：

   HelloWorldQueueMDB 代码示例

   ```
   @MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
       @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
   "${property.helloworldmdb.queue}"),
       @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
   "javax.jms.Queue"),
       @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
   acknowledge") })
   ```

2. 更改 **HelloWorldTopicMDB** 类中的 **@ActivationConfigProperty 目标** 属性值，以使用 system 属性替换。**@MessageDriven** 注释现在应如下所示：

   HelloWorldTopicMDB Code Example

   ```
   @MessageDriven(name = "HelloWorldQTopicMDB", activationConfig = {
       @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
   "${property.helloworldmdb.topic}"),
       @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
   "javax.jms.Topic"),
       @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
   acknowledge") })
   ```

3. 更改 **HelloWorldMDBServletClient** 类中的 **@Resource** 注释，以使用系统属性替换。现在，代码应该类似如下：

   HelloWorldMDBServletClient Code 示例

   ```
   /**
    * Definition of the two Jakarta Messaging Service destinations used by the quickstart
    * (one queue and one topic).
    */
   @JMSDestinationDefinitions(
       value = {
   ```

```
        @JMSDestinationDefinition(
            name = "java:/${property.helloworldmdb.queue}",
            interfaceName = "javax.jms.Queue",
            destinationName = "HelloWorldMDBQueue"
        ),
        @JMSDestinationDefinition(
            name = "java:/${property.helloworldmdb.topic}",
            interfaceName = "javax.jms.Topic",
            destinationName = "HelloWorldMDBTopic"
        )
    })
/**
 * <p>
 * A simple servlet 3 as client that sends several messages to a queue or a topic.
 * </p>
 *
 * <p>
 * The servlet is registered and mapped to /HelloWorldMDBServletClient using the
{@linkplain WebServlet
 * @HttpServlet}.
 * </p>
 *
 * @author Serge Pagop (spagop@redhat.com)
 *
 */
@WebServlet("/HelloWorldMDBServletClient")
public class HelloWorldMDBServletClient extends HttpServlet {

    private static final long serialVersionUID = -8314035702649252239L;

    private static final int MSG_COUNT = 5;

    @Inject
    private JMSContext context;

    @Resource(lookup = "${property.helloworldmdb.queue}")
    private Queue queue;

    @Resource(lookup = "${property.helloworldmdb.topic}")
    private Topic topic;

  <!-- Remainder of code can be found in the `helloworld-mdb-propertysubstitution` quickstart.
-->
```

4. 修改 **activemq-jms.xml** 文件，以使用系统属性替换值。

.activemq-jms.xml 文件示例

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-activemq-deployment:1.0">
    <server>
        <jms-destinations>
            <jms-queue name="HELLOWORLDMDBQueue">
                <entry name="${property.helloworldmdb.queue}"/>
            </jms-queue>
            <jms-topic name="HELLOWORLDMDBTopic">
```

```
            <entry name="${property.helloworldmdb.topic}"/>
        </jms-topic>
    </jms-destinations>
  </server>
</messaging-deployment>
```

5. 部署应用。应用现在将系统属性指定的值用于 **@Resource 和 @ ActivationConfigProperty** 属性值。

4.7. 激活配置属性

4.7.1. 使用注解配置 MDB

您可以使用 **@MessageDriven 元素和**对应于 **@ ActivationConfigProperty 注释的子元素来配置激活** 属性。**@ActivationConfigProperty** 是 MDB 的一组激活配置属性。 **@ActivationConfigProperty** 注释规范如下：

```
@Target(value={})
@Retention(value=RUNTIME)
public @interface ActivationConfigProperty
{
 String propertyName();
 String propertyValue();
}
```

显示 @ActivationConfigProperty 的示例

```
@MessageDriven(name="MyMDBName",
activationConfig =
{
   @ActivationConfigProperty(propertyName="destinationLookup",propertyValue="queueA"),
   @ActivationConfigProperty(propertyName = "destinationType",propertyValue =
"javax.jms.Queue"),
   @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge"),
})
```

4.7.2. 使用部署描述符配置 MDB

ejb -jar.xml 中的 <message-driven > 元素 将 bean 定义为 MDB。 **<activation-config>** 和元素包含通过 **activation-config-property 元素的** MDB 配置。

ejb-jar.xml示例

```
<?xml version="1.1" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
               xmlns="http://java.sun.com/xml/ns/javaee"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
               version="3.1">
   <enterprise-beans>
```

```
        <message-driven>
          <ejb-name>MyMDBName</ejb-name>
          <ejb-class>org.jboss.tutorial.mdb_deployment_descriptor.bean.MyMDBName</ejb-class>
          <activation-config>
            <activation-config-property>
              <activation-config-property-name>destinationLookup</activation-config-property-name>
              <activation-config-property-value>queueA</activation-config-property-value>
            </activation-config-property>
            <activation-config-property>
              <activation-config-property-name>destinationType</activation-config-property-name>
              <activation-config-property-value>javax.jms.Queue</activation-config-property-value>
            </activation-config-property>
            <activation-config-property>
              <activation-config-property-name>acknowledgeMode</activation-config-property-name>
              <activation-config-property-value>Auto-acknowledge</activation-config-property-value>
            </activation-config-property>
          </activation-config>
        </message-driven>
      <enterprise-beans>
    </jboss:ejb-jar>
```

表 4.1. 根据 Jakarta 消息传递服务规范定义的激活配置属性

Name	描述
destinationLookup	队列或主题的 Java 命名和目录接口名称。这是一个强制值。
connectionFactoryLookup	管理员定义的 **javax.jms.ConnectionFactory**、**javax.jms.QueueConnectionFactory** 或 **javax.jms.TopicConnectionFactory** 对象，用于连接到端点将从其接收消息的 Jakarta 消息提供商。 如果未明确定义，则使用名称为 **activemq-ra** 的池连接工厂。
destinationType	目标有效值的类型为 **javax.jms.Queue 或 javax.jms.Topic**。这是一个强制值。
messageSelector	**messageSelector** 属性的值是一个字符串，用于选择可用消息的子集。其语法基于 SQL 92 条件表达式语法的一个子集，在 Jakarta Messaging 规范中进行了详细介绍。在 **ActivationSpec** JavaBean 上为 **messageSelector** 属性指定一个值是可选的。
acknowledgeMode	不使用转换的 Jakarta Messaging 时进行确认的类型。有效值为 **Auto-acknowledge 或 Dups-ok-acknowledge**。这不是强制值。 默认值为 **Auto-acknowledge**。
clientID	连接的客户端 ID。这不是强制值。
subscriptionDurability	主题订阅是否持久.有效值为 **Durable** 或 **NonDurable**。这不是强制值。 默认值为 **NonDurable**。

Name	描述
subscriptionName	主题订阅的订阅名称。这不是强制值。

表 4.2. JBoss EAP 定义的激活配置属性

Name	描述
destination	将此属性与 **useJNDI=true** 一起使用与 **destinationLookup** 的含义相同。将其与 **useJNDI=false** 一起使用时，不会查找目的地，而是实例化。您可以使用此属性而不是 **destinationLookup**。这不是强制值。
shareSubscriptions	连接是否已配置为共享订阅。 默认值为 **False**。
user	Jakarta 消息传递连接的用户。这不是强制值。
password	Jakarta Messaging 连接的密码。这不是强制值。
maxSession	要使用的最大并发会话数。这不是强制值。 默认值为 **15**。
transactionTimeout	会话的事务超时，以毫秒为单位。这不是强制值。 如果没有指定或 0，则忽略该属性，并且不会覆盖 **transactionTimeout**，并使用 Transaction Manager 中定义的默认 transaction **Timeout**。
useJNDI	是否使用 Java 命名和目录接口来查找目的地。 默认值为 **True**。
jndiParams	要在连接中使用的 Java 命名和目录接口参数。参数定义为 **name=value** 对，用; 分隔 **;**
localTx	使用本地事务而不是 XA。 默认值为 **False**。
setupAttempts	设置雅加达消息传递连接的尝试次数。有可能在 Jakarta Messaging 资源可用之前部署 MDB。在这种情况下，资源适配器将尝试设置几次，直到资源可用为止。这只适用于入站连接。 默认值为 **-1**。
setupInterval	设置 Jakarta 消息传递连接的连续尝试间隔，以毫秒为单位。这只适用于入站连接。 默认值为 **2000**。

Name	描述
rebalanceConnections	是否启用入站连接重新平衡。此参数允许在底层集群拓扑更改时重新平衡所有入站连接。 出站连接没有重新平衡。 默认值为 **False**。
deserializationWhiteList	白名单以逗号分隔的条目列表，这是受信任的类和软件包列表。此属性供 Jakarta 消息传递资源适配器用于允许取消序列列表中的对象。 如需更多信息，请参阅为 JBoss EAP *配置消息传递中的控制 Jakarta 消息传递对象Message Deserialization*。
deserializationBlackList	黑名单以逗号分隔的条目列表，这是不受信任的类和软件包列表。此属性供 Jakarta 消息传递资源适配器使用，以防止列表中的对象被非序列化。 如需更多信息，请参阅为 JBoss EAP *配置消息传递中的控制 Jakarta 消息传递对象Message Deserialization*。

4.7.3. 配置 MDB 的一些用例示例

- MDB 接收消息的用例
  有关 MDB 收到消息的基本场景，请参阅 JBoss EAP 附带的 **helloworld-mdb** 快速启动。

- MDB 发送消息的用例
  在处理完消息后，您可能需要通知其他业务系统或回复该消息。在这种情况下，您可以从 MDB 发送消息，如下所示：

```
package org.jboss.as.quickstarts.mdb;

import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.inject.Inject;
import javax.jms.JMSContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.Queue;

@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"queue/MyMDBRequest"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge") })
public class MyMDB implements MessageListener {

    @Inject
    private JMSContext jmsContext;
```

```
    @Resource(lookup = "java:/queue/ResponseDefault")
    private Queue defaultDestination;

    /**
     * @see MessageListener#onMessage(Message)
     */
    public void onMessage(Message rcvMessage) {
        try {
            Message response = jmsContext.createTextMessage("Response for message " +
rcvMessage.getJMSMessageID());
            if (rcvMessage.getJMSReplyTo() != null) {
                jmsContext.createProducer().send(rcvMessage.getJMSReplyTo(), response);
            } else {
                jmsContext.createProducer().send(defaultDestination, response);
            }
        } catch (JMSException e) {
            throw new RuntimeException(e);
        }
    }
}
```

在上例中，在 MDB 收到消息后，它将回复 **JMSReplyTo** 中指定的目的地或绑定到 Java 命名和目录接口名称 **java:/queue/ResponseDefault** 的目的地。

- MDB 配置入站连接重新平衡的用例

```
@MessageDriven(name="MyMDBName",
    activationConfig =
    {
        @ActivationConfigProperty(propertyName = "destinationType",propertyValue =
"javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"queueA"),
        @ActivationConfigProperty(propertyName = "rebalanceConnections", propertyValue =
"true")
    }
)
```

第 5 章 调用会话 BEAN

5.1. 关于 JAKARTA ENTERPRISE BEANS 客户端上下文

JBoss EAP 引入了 Jakarta Enterprise Beans 客户端 API，用于管理远程 Jakarta Enterprise Beans 调用。JBoss Jakarta Enterprise Beans 客户端 API 使用 EJBClientContext，可以与一个或多个线程同时关联并使用。这意味着，EJBClientContext 可能包含任意数量的 Jakarta Enterprise Beans 接收器。Jakarta Enterprise Beans 接收器是一个组件，它知道如何与能够处理 Jakarta Enterprise Beans 调用的服务器进行通信。通常，Jakarta Enterprise Beans 远程应用程序可分为以下几类：

- 远程客户端，作为独立 Java 应用运行。
- 远程客户端，在另一个 JBoss EAP 实例中运行。

根据远程客户端的类型，从 Jakarta Enterprise Beans 客户端 API 角度来看，JVM 中可能存在多个 EJBClientContext。

虽然独立应用通常具有单个 EJBClientContext，可由任意数量的 Jakarta Enterprise Beans 接收器来支持，但这不是强制要求。如果独立应用有多个 EJBClientContext，则 Jakarta Enterprise Beans 客户端上下文选择器负责返回适当的上下文。

如果远程客户端在另一个 JBoss EAP 实例中运行，每个部署的应用都将具有对应的 Jakarta 企业 Beans 客户端上下文。每当该应用调用另一个 Jakarta Enterprise Beans 时，对应的 Jakarta Enterprise Beans 客户端上下文用于查找正确的 Jakarta Enterprise Beans 接收器，然后处理调用。

5.2. 使用远程 JAKARTA ENTERPRISE BEANS 客户端

5.2.1. 初始上下文查找

在创建初始上下文时，您可以使用 **PROVIDER_URL** 属性传递远程服务器的地址：

```
public class Client {
    public static void main(String[] args)
            throws NamingException, PrivilegedActionException, InterruptedException {
        InitialContext ctx = new InitialContext(getCtxProperties());
        String lookupName = "ejb:/server/HelloBean!ejb.HelloBeanRemote";
        HelloBeanRemote bean = (HelloBeanRemote)ctx.lookup(lookupName);
        System.out.println(bean.hello());
        ctx.close();
    }
    public static Properties getCtxProperties() {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
WildFlyInitialContextFactory.class.getName());
        props.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
        props.put(Context.SECURITY_PRINCIPAL, "joe");
        props.put(Context.SECURITY_CREDENTIALS, "joeIsAwesome2013!");
        return props;
    }
}
```

注意

用于查找的初始上下文工厂为 **org.wildfly.naming.client.WildFlyInitialContextFactory**。

5.2.2. 远程 Jakarta Enterprise Beans 配置文件

JBoss EAP 具有 Elytron 安全框架。**wildfly-config.xml** 文件存在于客户端应用的类路径的 **META-INF/** 目录中，允许针对 Elytron 安全框架和 Jakarta Enterprise Beans 客户端配置提供广泛的身份验证和授权选项。

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="default" />
    </authentication-rules>
    <authentication-configurations>
      <configuration name="default">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <set-user-name name="admin" />
        <credentials>
          <clear-password password="password123!" />
        </credentials>
      </configuration>
    </authentication-configurations>
  </authentication-client>
  <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.0">
    <connections>
      <connection uri="remote+http://127.0.0.1:8080" />
    </connections>
  </jboss-ejb-client>
</configuration>
```

作为在初始上下文中嵌入 **PROVIDER_URL**、**SECURITY_PRINCIPAL** 和 **SECURITY_CREDENTIALS** 参数的替代方案，您可以在 **wildfly -config.xml 文件中分别使用 <connection-uri> 和 <authentication-client** > 元素来配置连接 URI 和安全设置。

5.2.3. ClientTransaction 注解

@org.jboss.ejb.client.annotation.ClientTransaction 注释处理来自 Jakarta Enterprise Beans 客户端的交易传播。如果客户没有交易，您可以强制传播失败，或者防止交易传播，即使客户有一个主动交易。您可以使用 **org.jboss.ejb.client.annotation.ClientTransactionPolicy** 接口的常量来控制 **ClientTransaction** 注解的策略。以下是 **org.jboss.ejb.client.annotation.ClientTransactionPolicy** 接口的常量：

- MANDATORY: Fail（无客户端交易上下文时除外）；如果存在客户端侧事务上下文，则传播客户端交易上下文。
- NEVER：不传播任何事务上下文；如果存在客户端交易上下文，则会引发异常。
- 注意_SUPPORTED：不传播任何事务上下文，无论是否存在客户端交易上下文。
- SUPPORTS：如果没有客户端端交易上下文，则中断交易；传播客户端交易上下文（如果存在）。

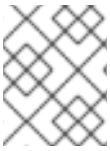
如果没有注解，则默认策略为 **org.jboss.ejb.client.annotation.ClientTransactionPolicy#SUPPORTS**，这意味着事务会被传播（如果存在），但传播不会失败，无论事务是否存在。

```
@ClientTransaction(ClientTransactionPolicy.MANDATORY)
@Remote
public interface RemoteCalculator  {
  public void callRemoteEjb() { }
}
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

  @Override
  public void callRemoteEjb()  {   }
}
```

该注释允许远程接口提供商告知远程接口使用者是否需要方法进行事务。

5.3. 远程 JAKARTA ENTERPRISE BEANS 数据压缩

您可以压缩包含 Enterprise Beans 协议消息的消息流



注意

目前，压缩只能通过在 Jakarta Enterprise Beans 接口上的注释指定，该界面应该位于客户端和服务器端。当前没有与指定压缩提示相同的 XML。

数据压缩提示可以通过 JBoss 注释 **org.jboss.ejb.client.annotation.CompressionHint** 指定。hint 值指定是压缩请求、响应还是请求和响应。添加 **@CompressionHint** 默认为 **compressResponse=true** 和 **compressRequest=true**.

该注解可以在接口级别指定，以应用到 Jakarta Enterprise Beans 接口中的所有方法，例如：

```
import org.jboss.ejb.client.annotation.CompressionHint;

@CompressionHint(compressResponse = false)
public interface ClassLevelRequestCompressionRemoteView {
    String echo(String msg);
}
```

或者注解可应用于 Jakarta Enterprise Beans 接口中的特定方法，例如：

```
import org.jboss.ejb.client.annotation.CompressionHint;

public interface CompressableDataRemoteView {

    @CompressionHint(compressResponse = false, compressionLevel =
Deflater.BEST_COMPRESSION)
    String echoWithRequestCompress(String msg);

    @CompressionHint(compressRequest = false)
    String echoWithResponseCompress(String msg);
```

```
    @CompressionHint
    String echoWithRequestAndResponseCompress(String msg);

    String echoWithNoCompress(String msg);
}
```

上面显示的 **compressionLevel** 设置可具有以下值：

- BEST_COMPRESSION
- BEST_SPEED
- DEFAULT_COMPRESSION
- NO_COMPRESSION

compressionLevel 设置默认为 **Deflater.DEFAULT_COMPRESSION**。

带有方法级别覆盖的类级注解：

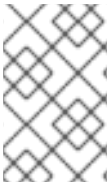
```
@CompressionHint
public interface MethodOverrideDataCompressionRemoteView {

    @CompressionHint(compressRequest = false)
    String echoWithResponseCompress(final String msg);

    @CompressionHint(compressResponse = false)
    String echoWithRequestCompress(final String msg);

    String echoWithNoExplicitDataCompressionHintOnMethod(String msg);
}
```

在客户端上，确保 **org.jboss.ejb.client.view.annotation.scan.enabled** 系统属性设置为 **true**。此属性告知 JBoss Jakarta Enterprise Beans 客户端扫描注释。

5.4. JAKARTA ENTERPRISE BEANS 客户端远程互操作性

远程 Jakarta Enterprise Beans 客户端应用程序使用 **remoting** 子系统中定义的连接器来连接到服务器。您可以根据您的需要使用以下连接器之一：

- **http-connector** ：通过 **undertow** 的 HTTP 升级功能支持到服务器的客户端连接，默认端口 8080。配置此连接器后，客户端将 **remote+http** URI 方案用于未加密的连接，或使用 **remote+https** URI 方案进行加密连接。
- **连接器** ：通过旧的 **远程** URI 方案支持到服务器的客户端连接。此连接器适合与旧的 Jakarta Enterprise Beans 客户端应用程序一起使用。



注意

除了使用以前的基于 remoting 的连接器外，Jakarta Enterprise Beans 客户端还可以使用 **http** URI 方案通过 **undertow** 和 HTTP 协议连接到服务器。如需了解更多详细信息，请参阅 Jakarta Enterprise Beans Invocation Over HTTP 。

默认 HTTP 连接器

默认连接器是 **http-connector**，这需要客户端使用 URI 方案 **remote+http** 或 **remote+https**。默认远程连接端口为 **8080**，它是 **undertow** 的默认端口相同。以下示例显示了 **jboss-ejb-client** 属性文件：

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

支持来自不同 JBoss EAP 版本的客户端

如果客户端应用程序使用来自 JBoss EAP 6 的 Jakarta Enterprise Beans 客户端库，并且需要连接到 JBoss EAP 7 服务器，则必须将服务器配置为在 **8080** 以外的端口上公开重新移动连接器。然后，客户端必须使用新配置的连接器进行连接。

使用 JBoss EAP 7 中的 Jakarta Enterprise Beans 客户端库并需要连接到 JBoss EAP 6 服务器的客户端应用程序必须注意服务器实例不使用远程 **http-remoting** 连接器，而是使用补救 **连接器**。这可以通过定义以下新客户端的连接属性来实现：

```
remote.connection.default.protocol=remote
```

为 Jakarta Enterprise Beans 客户端应用程序支持多个连接器

在 JBoss EAP 7.4 之前，Jakarta Enterprise Beans 客户端应用程序仅限于只使用一个远程连接器（在 **remoting** 子系统中定义）以连接到服务器。这个连接器在 **ejb3** 子系统的 **remote** 元素的 **connector-ref** 属性中指定。您可以使用默认的 **http-connector** 通过 **undertow** 的 HTTP 升级功能通过 **remote+http** 协议提供连接，或使用传统的 **连接器** 通过传统远程协议 **提供连接**。

借助 JBoss EAP 7.4，您可以指定 Jakarta Enterprise Beans 客户端可用于连接目的的连接器列表。要指定此列表，请使用 **远程** 元素的新 **连接器** 属性。**connector** 属性接受 **remoting** 子系统中定义的连接器列表。这可让单个服务器为 Jakarta Enterprise Beans 客户端应用程序提供多个连接。例如，与 EAP 7.2 或更高版本兼容的客户端可以使用 **remote +http** 协议与之前的 EAP 7.2 版本兼容的旧客户端连接到服务器，其使用传统的远程协议与 **连接器** 进行连接。

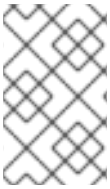
Example

考虑 legacy-remoting- connector 是 remoting 子系统中定义的连接器，以下示例演示了使用 write 属性更新 remoting 连接器值的 ejb3 子系统配置：

```
/subsystem=ejb3/service=remote:write-attribute(name=connectors, value=[http-remoting-connector,
legacy-remoting-connector])
```

您可以查看 domain.xml 或 standalone.xml 文件，以检查 ejb3 子系统中配置的 remoting 连接器：

```
<remote cluster="ejb" connectors="http-remoting-connector legacy-remoting-connector" thread-pool-
name="default">
  <channel-creation-options>
   <option name="MAX_OUTBOUND_MESSAGES" value="1234" type="remoting"/>
  </channel-creation-options>
</remote>
```



注意

Jakarta Enterprise Beans 远程调用仅支持 JBoss EAP 6 的 JBoss EAP 7。

除了 Jakarta Enterprise Beans 客户端远程互操作外，您还可以使用以下选项连接到旧的客户端：

- 在 JBoss EAP *配置指南中* 配置用于 JTS 事务的 ORB。

5.5. 为远程 JAKARTA ENTERPRISE BEANS 调用配置 IIOP

JBoss EAP 支持对 JBoss EAP 上部署的 Jakarta 企业 Bean 进行基于 CORBA/IIOP 的访问。

<iiop> 元素用于启用 IIOP、CORBA（调用 Jakarta Enterprise Beans）。此元素的存在意味着安装了 iiop-openjdk 子系统。<iiop> 元素包括以下两个属性：

- 默认启用： 如果 情况如此，则所有带有企业 Bean 2.x 主接口的 Jakarta Enterprise Beans 都通过 IIOP 公开。否则，必须通过 jboss-ejb3.xml 显式启用它们。

- Use-qualified-name ：如果为 true，则 Jakarta Enterprise Beans 与其绑定至 CORBA 命名上下文，其中包含部署的应用程序和模块名称，如 myear/myejbjar/MyBean。如果这是 false，则默认绑定名称就是 bean 名称。

注意

尽管 Jakarta Enterprise Beans 3 远程调用通常不需要 RemoteHome 接口，但任何使用 IIOP 公开的 Jakarta Enterprise Beans 3 bean 都需要此接口。然后，您必须使用 jboss-ejb3.xml 文件启用 IIOP，或在 standalone-full.xml 配置文件中为所有 Jakarta Enterprise Beanss 启用 IIOP。

启用 IIOP

要启用 IIOP，您必须安装 IIOP OpenJDK ORB 子系统，ejb 3 子系统配置中有 <iiop/> 元素。发行版附带的 standalone-full.xml 配置启用了这两个配置。

IIOP 在服务器配置文件的 iiop-openjdk 子系统中配置。

```
<subsystem xmlns="urn:jboss:domain:iiop-openjdk:2.1">
```

使用以下管理 CLI 命令访问和更新 iiop-openjdk 子系统：

```
/subsystem=iiop-openjdk
```

IIOP 元素采用控制服务器默认行为的两个属性。

```
<subsystem xmlns="urn:jboss:domain:ejb3:5.0">
  ...
  <iiop enable-by-default="false" use-qualified-name="false"/>
  ...
</subsystem>
```

以下管理 CLI 命令在 ejb3 子系统中添加 <iiop> 元素 ：

```
/subsystem=ejb3/service=iiop:add(enable-by-default=false, use-qualified-name=false)
```

使用 IIOP 创建 Communicates 的 Jakarta Enterprise Beans

以下示例演示了如何从客户端进行远程 IIOP 调用：

1. 在服务器上创建一个企业 Bean 2 Bean：

```
@Remote(IIOPRemote.class)
@RemoteHome(IIOPBeanHome.class)
@Stateless
public class IIOPBean {
    public String sayHello() throws RemoteException {
        return "hello";
    }
}
```

2. 创建具有强制方法 create（） 的主页实施。客户端调用这个方法来获取远程接口的代理来调用业务方法：

```
public interface IIOPBeanHome extends EJBHome {
    public IIOPRemote create() throws RemoteException;
}
```

3. 为远程连接企业 Bean 创建一个远程接口：

```
public interface IIOPRemote extends EJBObject {
   String sayHello() throws RemoteException;
}
```

4. 通过在 META-INF 中创建描述符文件 jboss-ejb3.xml 来 引入用于远程调用的 Bean ：

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
               xmlns="http://java.sun.com/xml/ns/javaee"
               xmlns:iiop="urn:iiop"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd
                 http://java.sun.com/xml/ns/javaee http://www.jboss.org/j2ee/schema/jboss-ejb3-
spec-2_0.xsd
                 urn:iiop jboss-ejb-iiop_1_0.xsd"
               version="3.1"
               impl-version="2.0">
   <assembly-descriptor>
      <iiop:iiop>
         <ejb-name>*</ejb-name>
      </iiop:iiop>
   </assembly-descriptor>
</jboss:ejb-jar>
```

注意

现在，打包的 Bean 以及 JAR 文件中的描述符已准备好部署到 JBoss EAP 容器。

5. 在客户端创建上下文：

```
System.setProperty("com.sun.CORBA.ORBUseDynamicStub", "true");
final Properties props = new Properties();
props.put(Context.PROVIDER_URL, "corbaloc::localhost:3528/JBoss/Naming/root");
props.setProperty(Context.URL_PKG_PREFIXES,
"org.jboss.iiop.naming:org.jboss.naming.client");
props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.cosnaming.CNCtxFactory");
props.put(Context.OBJECT_FACTORIES,
"org.jboss.tm.iiop.client.IIOPClientUserTransactionObjectFactory");
```

注意

客户端需要将 wildfly iiop openjdk 库添加到其类路径中。客户端可能还需要将 org.wildfly:wildfly-iiop-openjdk 构件添加为 Maven 依赖项。

6. 使用 context lookup 来缩小对 IIOPBeanHome 主目录的引用范围。然后，调用 home 接口 create（） 方法来访问远程接口，允许您调用其方法：

```
try {
   Context context = new InitialContext(props);

   final Object iiopObj = context.lookup(IIOPBean.class.getSimpleName());
   final IIOPBeanHome beanHome = (IIOPBeanHome)
PortableRemoteObject.narrow(iiopObj, IIOPBeanHome.class);
   final IIOPRemote bean = beanHome.create();

   System.out.println("Bean saying: " + bean.sayHello());
} catch (Exception e) {
   e.printStackTrace();
}
```

5.6. 配置 JAKARTA ENTERPRISE BEANS 客户端地址

您可以使用 SessionContext 界面确定 Jakarta Enterprise Beans 客户端地址，如下例所示。

```
public class HelloBean implements HelloBeanRemote {
   @Resource
   SessionContext ctx;
   private Long counter;
   public HelloBean() {
   }
   @PostConstruct
   public void init() {
      counter = 0L;
   }
   @Override
   @RolesAllowed("users")
   public String hello() {
      final String message = "method hello() invoked by user " +
ctx.getCallerPrincipal().getName()
            + ", source addr = " +  ctx.getContextData().get("jboss.source-address").toString();
      System.out.println(message);
      return message;
   }
}
```

独立客户端配置

您可以在 wildfly -config.xml 文件中带有 namespace urn:xnio:3.5 的 worker 元素中配置 outbound-bind- addresses 元素。bind-address 子元素取 匹配、bind-address 和bind- port 的属性，如下定义。

以下是使用 wildfly-config.xml 文件的独立客户端配置示例：

```
<configuration>
    <worker xmlns="urn:xnio:3.5">
        <worker-name value="default"/>
        <outbound-bind-addresses>
            <bind-address bind-address=IP_ADDRESS_TO_BIND_TO bind-
port=OPTIONAL_SOURCE_PORT_NUMBER match=CIDR_BLOCK />
        </outbound-bind-addresses>
    </worker>
</configuration>
```

outbound -bind-address 需要以下属性：

- match 是一个无类别域间路由(CIDR)块，如 10.0.0.0/8,ff00::\8,0.0.0.0/0,::/0。
- bind-address 指定目标地址与 match 参数中指定的 CIDR 块匹配时要绑定的 IP 地址。它应当与 CIDR 块相同的地址系列。
- bind-port 是一个可选的源端口号，默认为 0。

  如果没有匹配的表达式，则不会显式绑定出站套接字。

基于容器的配置

基于容器的 Jakarta Enterprise Beans 客户端地址的配置与 wildfly-config.xml 文件中定义的独立客户端配置类似。

以下示例在 io 子系统的默认 worker 元素上配置 outbound -bind-address，ej b3 子系统默认使用它。

```
/subsystem=io/worker=default/outbound-bind-
address=SPECIFY_OUTBOUND_BIND_ADDRESS:add(bind-
address=IP_ADDRESS_TO_BIND_TO, bind-port=OPTIONAL_SOURCE_PORT_NUMBER,
match=CIDR_BLOCK)
```

5.7. JAKARTA ENTERPRISE BEANS INVOCATION OVER HTTP

通过 HTTP 调用 Jakarta Enterprise Beans 包含两个不同的部分：客户端和服务器端实施。

5.7.1. 客户端实施

客户端实施由使用 Undertow HTTP 客户端调用服务器的 EJBReceiver 组成。连接管理使用连接池自动处理。

要将 Jakarta Enterprise Beans 客户端应用程序配置为使用 HTTP 传输，您必须根据 HTTP 传输实施添加以下依赖关系：

```
<dependency>
   <groupId>org.wildfly.wildfly-http-client</groupId>
   <artifactId>wildfly-http-ejb-client</artifactId>
</dependency>
```

要执行 HTTP 调用，您必须使用 http URL 方案，并包含 HTTP 调用器的上下文名称 wildfly-services。例如：如果您使用 remote+http://localhost:8080 作为目标 URL，为了使用 HTTP 传输，您必须将其更新为 http://localhost:8080/wildfly-services。

5.7.2. 服务器端实施

服务器端实施由处理传入 HTTP 请求的服务组成，取消发送请求并将结果传递到内部 Jakarta Enterprise Beans 调用代码。

若要配置服务器，必须在您要在 undertow 子系统中使用的每个虚拟主机上启用 http-invoker。这在标准配置中默认启用。如果禁用，可使用以下管理 CLI 命令重新启用它：

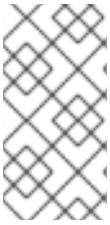
```
/subsystem=undertow/server=default-server/host=default-host/setting=http-invoker:add(http-
authentication-factory=myfactory, path="wildfly-services")
```

HTTP-invoker 有两个属性：一个默认为 wildfly-services 的路径，以及以下之一：

- http-authentication-factory 必须是 Elytron http-authentication-factory 的 引用，如上述命令所示。

- 传统 **安全领域.**

请注意，上述两个属性是互斥的：您不能同时指定 **http-authentication-factory** 和 **security -realm**。



注意

任何旨在使用 **http-authentication-factory** 的部署都必须使用 **Elytron** 安全性和与指定 **HTTP** 身份验证工厂对应的相同安全域。

第 6 章 JAKARTA ENTERPRISE BEANS 应用程序安全性

6.1. 安全身份

6.1.1. 关于 Jakarta Enterprise Beans 安全身份

Jakarta Enterprise Beans 可以指定在调用其他组件时要使用的身份。这是 Jakarta Enterprise Beans 安全身份，也称为调用身份。

默认情况下，Jakarta Enterprise Beans 使用自己的调用者身份。身份也可以设置为特定的安全角色。当您想构建分段安全模型时，使用特定的安全角色很有用处，例如，限制对一组组件的访问，以仅访问内部 Jakarta Enterprise Beans。

6.1.2. 设置 Jakarta Enterprise Beans 的安全身份

Jakarta Enterprise Beans 的安全身份通过安全配置中的 <security-identity> 标签指定。如果没有 <security-identity> 标签，则默认使用 Jakarta Enterprise Beans 的调用者身份。

示例：将 Jakarta Enterprise Beans 的安全身份设置为与 Its Caller 相同

本例将 Jakarta Enterprise Beans 进行的方法调用的安全身份设置为与当前调用者的身份相同。如果没有指定 <security-identity> 元素声明，则此行为是默认设置。

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>ASessionBean</ejb-name>
      ...
      <security-identity>
        <use-caller-identity/>
      </security-identity>
    </session>
    ...
  </enterprise-beans>
</ejb-jar>
```

示例：将 Jakarta Enterprise Beans 的安全身份设置为特定角色

要将安全身份设置为特定角色，请在 < security-identity> 标签中使用 <run-as > 和 <role-name > 标签。

```
<ejb-jar>
  <enterprise-beans>
```

```
    <session>
      <ejb-name>RunAsBean</ejb-name>
      ...
      <security-identity>
        <run-as>
          <description>A private internal role</description>
          <role-name>InternalRole</role-name>
        </run-as>
      </security-identity>
    </session>
  </enterprise-beans>
  ...
</ejb-jar>
```

默认情况下，在使用 <run-as> 时，会为传出调用分配一个名为 anonymous 的主体。要分配不同的主体，请使用 <run-as-principal>。

```
<session>
   <ejb-name>RunAsBean</ejb-name>
   <security-identity>
      <run-as-principal>internal</run-as-principal>
   </security-identity>
</session>
```

注意

您还可以在 servlet 元素中使用 <run-as> 和 <run-as-principal> 元素。

6.2. JAKARTA ENTERPRISE BEANS 方法权限

6.2.1. 关于 Jakarta Enterprise Beans 方法权限

Jakarta Enterprise Beans 可以将方法的访问权限限制为特定的安全角色。

Jakarta Enterprise Beans <method-permission> 元素声明指定可调用 Jakarta Enterprise Beans 接口方法的角色。您可以为以下组合指定权限：

- 名为 Jakarta Enterprise Beans 的所有家庭和组件接口方法
- 名为 Jakarta Enterprise Beans 的家或组件接口的指定方法

- 带有过载名称的一组方法中的指定方法

6.2.2. 使用 Jakarta Enterprise Beans 方法权限

<method-permission> 元素定义允许访问 <method> 元素定义的 Jakarta Enterprise Beans 方法的逻辑角色。几个示例演示了 xml 的语法。可能存在多个方法权限语句，它们具有累积效果。<method-permission> 元素是 < ejb-jar> 描述符 <assembly-descriptor > 元素的子项。

XML 语法是使用 Jakarta Enterprise Beans 方法权限标注的替代方案。

示例：允许角色访问 Jakarta Enterprise Beans 的所有方法

```
<method-permission>
  <description>The employee and temp-employee roles may access any method
  of the EmployeeService bean </description>
  <role-name>employee</role-name>
  <role-name>temp-employee</role-name>
  <method>
    <ejb-name>EmployeeService</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
```

示例：允许角色访问 Jakarta Enterprise Beans 和 Limit Method 参数的 Certain 方法

```
<method-permission>
  <description>The employee role may access the findByPrimaryKey,
  getEmployeeInfo, and the updateEmployeeInfo(String) method of
  the AcmePayroll bean </description>
  <role-name>employee</role-name>
  <method>
    <ejb-name>AcmePayroll</ejb-name>
    <method-name>findByPrimaryKey</method-name>
  </method>
  <method>
    <ejb-name>AcmePayroll</ejb-name>
    <method-name>getEmployeeInfo</method-name>
  </method>
  <method>
    <ejb-name>AcmePayroll</ejb-name>
```

```
    <method-name>updateEmployeeInfo</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
</method-permission>
```

示例：允许任何经过身份验证的用户访问 Jakarta Enterprise Beans 的方法

使用 <unchecked/> 元素时，任何经过身份验证的用户都可以使用指定的方法。

```
<method-permission>
  <description>Any authenticated user may access any method of the
  EmployeeServiceHelp bean</description>
  <unchecked/>
  <method>
    <ejb-name>EmployeeServiceHelp</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
```

示例：完全排除特定的 Jakarta Enterprise Beans 方法

```
<exclude-list>
  <description>No fireTheCTO methods of the EmployeeFiring bean may be
  used in this deployment</description>
  <method>
    <ejb-name>EmployeeFiring</ejb-name>
    <method-name>fireTheCTO</method-name>
  </method>
</exclude-list>
```

示例：一个 Complete <assembly-descriptor> 包含 Severinal <method-permission> Blocks

```
<ejb-jar>
   <assembly-descriptor>
      <method-permission>
         <description>The employee and temp-employee roles may access any method of the
EmployeeService bean </description>
         <role-name>employee</role-name>
```

```
        <role-name>temp-employee</role-name>
        <method>
          <ejb-name>EmployeeService</ejb-name>
          <method-name>*</method-name>
        </method>
      </method-permission>
      <method-permission>
        <description>The employee role may access the findByPrimaryKey, getEmployeeInfo,
and the updateEmployeeInfo(String) method of the AcmePayroll bean </description>
        <role-name>employee</role-name>
        <method>
          <ejb-name>AcmePayroll</ejb-name>
          <method-name>findByPrimaryKey</method-name>
        </method>
        <method>
          <ejb-name>AcmePayroll</ejb-name>
          <method-name>getEmployeeInfo</method-name>
        </method>
        <method>
          <ejb-name>AcmePayroll</ejb-name>
          <method-name>updateEmployeeInfo</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </method>
      </method-permission>
      <method-permission>
        <description>The admin role may access any method of the EmployeeServiceAdmin
bean </description>
        <role-name>admin</role-name>
        <method>
          <ejb-name>EmployeeServiceAdmin</ejb-name>
          <method-name>*</method-name>
        </method>
      </method-permission>
      <method-permission>
        <description>Any authenticated user may access any method of the
EmployeeServiceHelp bean</description>
        <unchecked/>
        <method>
          <ejb-name>EmployeeServiceHelp</ejb-name>
          <method-name>*</method-name>
        </method>
      </method-permission>
      <exclude-list>
        <description>No fireTheCTO methods of the EmployeeFiring bean may be used in this
deployment</description>
        <method>
          <ejb-name>EmployeeFiring</ejb-name>
          <method-name>fireTheCTO</method-name>
        </method>
      </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

6.3. JAKARTA ENTERPRISE BEANS 安全注释

6.3.1. 关于 Jakarta Enterprise Beans 安全注释

Jakarta Enterprise Beans javax.annotation.security 注释在 Jakarta Annotations 1.3 规范中定义。

Jakarta Enterprise Beans 使用安全注释将安全信息传递给部署者。包括：

@DeclareRoles

声明哪些角色可用。

@RunAs

配置组件的传播安全身份。

6.3.2. 使用 Jakarta Enterprise Beans 安全注释

您可以使用 XML 描述符或注释来控制哪些安全角色可以在 Jakarta Enterprise Beans 中调用方法。有关使用 XML 描述符的详情，请参考使用 Jakarta Enterprise Beans 方法权限。

部署描述符中指定的任何方法值都会覆盖注释值。如果在部署描述符中未指定方法值，则使用使用注解设置的值。覆盖粒度以每个方法为基础。

控制 Jakarta Enterprise Beans 的安全权限注释

@DeclareRoles

使用 @DeclareRoles 定义要针对哪个安全角色检查权限。如果没有 @DeclareRoles，则从 @RolesAllowed 注释自动构建列表。

@RolesAllowed, @PermitAll, @DenyAll

使用 @RolesAllowed 列出哪些角色可以访问某一方法或方法。使用 @PermitAll 或 @DenyAll 允许或拒绝所有角色使用方法或方法。

@RunAs

使用 @RunAs 指定方法在从注释方法发出调用时使用的角色。

示例：安全注解示例

```
@Stateless
@RolesAllowed({"admin"})
@SecurityDomain("other")
public class WelcomeEJB implements Welcome {
   @PermitAll
   public String WelcomeEveryone(String msg) {
      return "Welcome to " + msg;
   }
   @RunAs("tempemployee")
   public String GoodBye(String msg) {
      return "Goodbye, " + msg;
   }
   public String GoodbyeAdmin(String msg) {
      return "See you later, " + msg;
   }
}
```

在此代码中，所有角色都可以访问方法 WelcomeEveryone。GoodBye 方法在发出调用 时使用 temp employee 角色。只有 admin 角色可以访问方法 GoodbyeAdmin，以及任何其他没有安全注释的方法。

6.4. 远程访问 JAKARTA ENTERPRISE BEANS

6.4.1. 将 Security Realms 与 Remote Jakarta Enterprise Beans 客户端搭配使用

向客户端添加安全以远程调用 Jakarta 企业 Bean 的一种方式是使用安全域。安全域是用户名/密码对和用户名/角色对的简单数据库。术语也用于 Web 容器的上下文，其含义略有不同。

要针对 Jakarta Enterprise Beans 验证安全域中存在的特定用户名/密码对，请按照以下步骤执行：

- 将新的安全域添加到域控制器或单机服务器。
- 配置 wildfly-config.xml 文件，该文件位于应用程序的类路径中，如下例所示：

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="default" />
    </authentication-rules>
    <authentication-configurations>
      <configuration name="default">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <set-user-name name="admin" />
        <credentials>
          <clear-password password="password123!" />
        </credentials>
      </configuration>
    </authentication-configurations>
  </authentication-client>
  <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.0">
    <connections>
      <connection uri="remote+http://127.0.0.1:8080" />
    </connections>
  </jboss-ejb-client>
</configuration>
```

- 在使用新安全域的域或单机服务器上创建自定义远程连接器。

- 将 Jakarta Enterprise Beans 部署到服务器组，该组配置为将配置集与自定义远程连接器搭配使用，如果您不使用受管域，则部署到您的单机服务器。

6.4.2. 添加新安全域

1. 运行管理 CLI：

   执行 jboss-cli.sh 或 jboss-cli.bat 脚本并连接到服务器。

2. 创建新的安全域本身：

   运行以下命令，在域控制器或单机服务器上创建一个名为 MyDomainRealm 的新安全域：

   对于域实例，使用以下命令：

```
/host=master/core-service=management/security-realm=MyDomainRealm:add()
```

对于独立实例，使用以下命令：

```
/core-service=management/security-realm=MyDomainRealm:add()
```

3. 创建名为 **myfile.properties** 的属性文件：

对于单机实例，创建一个 ***EAP_HOME*/standalone/configuration/myfile.properties** 文件，并为域实例创建 ***EAP_HOME*/domain/configuration/myfile.properties** 文件。这些文件需要具有文件所有者的读取和写入权限。

```
$ chmod 600 myfile.properties
```

4. 创建对存储有关新角色信息的属性文件的引用：

运行以下命令，以创建指向 **myfile.properties** 文件的指针，该文件将包含与新角色相关的属性：

注意

属性文件不会由附带的 **add-user.sh** 和 **add-user.bat** 脚本创建。它必须在外部创建。

对于域实例，使用以下命令：

```
/host=master/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
```

对于独立实例，使用以下命令：

```
/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
```

您的新安全域已创建好。将用户和角色添加到这一新域时，信息将存储在独立于默认安全域的文件中。您可以使用自己的应用程序或程序来管理此新文件。

注意

在使用 add-user.sh 脚本将用户添加到 application-users.properties 以外的非默认文件时，您必须向其传递参数 --user-properties myfile.properties，否则它将尝试使用 application-users.properties。

6.4.3. 将用户添加到安全域

1. 运行 add-user 脚本。打开一个终端，并将目录更改到 *EAP_HOME*/bin/ 目录。如果您在红帽企业 Linux 或任何其他类 UNIX 操作系统上，请运行 add-user.sh。如果您在 Windows Server 上，请运行 add-user.bat。

2. 选择是否添加管理用户或应用程序用户。对于这个过程，键入 b 来添加应用程序用户。

3. 选择要添加用户的域。默认情况下，唯一可用的域是 ApplicationRealm。如果您添加了自定义域，您可以改为将用户添加到该域。

4. 出现提示时，键入用户名、密码和角色。出现提示时，键入所需的用户名、密码和可选角色。通过键入 yes 或键入 no 以取消更改 来 验证您的选择。更改写入到安全域的每个属性文件中。

6.4.4. 安全域和安全域之间的关系

重要

若要通过安全域保护 Jakarta Enterprise Beans，他们必须使用配置为从安全域检索用户凭据的安全域。这意味着域需要包含 Remoting 和 RealmDirect 登录模块。通过 @SecurityDomain 注释来分配安全域，该注释可应用于 Jakarta Enterprise Beans。

其他 安全域从底层安全域检索用户和密码数据。如果 Jakarta Enterprise Beans 没有 @SecurityDomain 注释，但 Jakarta Enterprise Beans 包含任何其他安全相关注释，则此安全域是默认域。

底层 http-remoting 连接器 （供客户端用于建立连接）决定了使用哪个安全域。有关 http-remoting 连接器 的更多信息，请参阅 JBoss EAP *配置指南中* 的关于删除 子系统 的更多信息。

可以通过这种方式更改默认连接器的安全域：

```
/subsystem=remoting/http-connector=http-remoting-connector:write-attribute(name=security-realm,value=MyDomainRealm)
```

6.4.5. 关于使用 SSL 加密的远程 Jakarta Enterprise Beans 访问

默认情况下，EJB2 和 Jakarta Enterprise Beans3 Beans3 Beans3 Bean 的远程方法调用(RMI)的网络流量不会被加密。在需要加密的实例中，可以使用安全套接字层(SSL)，以便加密客户端和服务器之间的连接。使用 SSL 还增加了允许网络流量遍历某些防火墙的好处，具体取决于防火墙配置。

警告

红帽建议显式禁用 SSLv2、SSLv3 和 TLSv1.0，以便在所有受影响的软件包中明确禁用 TLSv1.1 或 TLSv1.2。

6.5. ELYTRON 与 EJB 子系统集成

从 JBoss EAP 7.1 开始，可以映射部署，使其安全性由 elytron 子系统来处理。如果部署引用映射的安全域，其安全性将由 Elytron 处理，否则其安全性将由旧版安全子系统来处理。此映射在 ejb 子系统中定义。

在 ejb 子系统内，从部署中引用的安全域名创建映射至引用的 Elytron security-domain。当为部署中的 Bean 配置映射的安全域名时，这表示 Elytron 应处理安全性。设置新的 Jakarta Enterprise Beans 安全拦截器而不是现有的拦截器。

新的 Jakarta Enterprise Beans 安全拦截器利用与调用关联的 Elytron SecurityDomain 获取当前的 SecurityIdentity 并执行以下任务：

- 建立运行主体。
- 为 run-as 主体创建任何额外角色。

- 创建 run-as 角色。
- 做出授权决策。

JBoss EAP 7.1 在 ejb 子系统 application-security-domains 中引入了新的管理资源。application-security-domains 元素包含应用安全域，应映射到 Elytron 安全域。

表 6.1. application-security-domain 的属性

属性	描述
name	此属性引用部署中指定的安全域名称。
security-domain	此属性是对应当使用的 Elytron 安全域的引用。
enable-jacc	此属性启用使用 Jakarta 授权的授权。
referencing-deployments	这是一个运行时属性，它会列出所有当前引用 ASD 的部署。
legacy-compliant-principal-propagation	当未定义传入的以 运行时 身份时，此属性决定了本地未安全的 Jakarta Enterprise Beans 的主体。 如果设置为 **true**，则本地未保护的 Jakarta Enterprise Beans 的主体是当前验证的身份。 如果设置为 **false**，则本地未保护的 Jakarta Enterprise Beans 的主体是匿名的。 此属性为可选，默认值为 'true'。

您可以使用管理控制台或管理 CLI，在 ejb 子系统中配置 application-security-domain。如需更多信息，请参阅以下主题：

- 使用管理控制台配置应用程序安全域
- 使用管理 CLI 配置应用安全域

6.5.1. 使用管理控制台配置应用程序安全域

1. 访问管理控制台。如需更多信息，请参阅 JBoss EAP *配置指南* 中的 管理控制台。

2. 导航到 **Configuration → Subsystems → EJB，然后点 View。**

3. 选择 **Security Domain** 选项卡，并根据需要配置应用程序安全域。

6.5.2. 使用管理 CLI 配置应用安全域

在以下示例中，**MyAppSecurity** 是部署中引用的安全域，**App Domain** 是 **elytron** 子系统中配置的 **Elytron** 安全域。

```
/subsystem=ejb3/application-security-domain=MyAppSecurity:add(security-
domain=ApplicationDomain)
```

由于此命令，以下 **XML** 添加到服务器配置文件的 **ejb** 子系统中：

```
<application-security-domains>
    <application-security-domain name="MyAppSecurity" security-domain="ApplicationDomain"/>
</application-security-domains>
```

有关使用 **Elytron** 处理安全性的 **Jakarta** 企业 **Beans** 的简单工作示例，请参见 **JBoss EAP** 附带的 **ejb-security** 快速入门。

第 7 章 JAKARTA ENTERPRISE BEANS INTERCEPTORS

7.1. 自定义 INTERCEPTORS

JBoss EAP 允许您开发和管理自定义 Jakarta 企业 Beans 拦截器。

您可以创建以下拦截器类型：

- 客户端拦截器

  当 JBoss EAP 充当客户端时，运行客户端拦截器。

- 服务器拦截器

  当 JBoss EAP 充当服务器时，运行服务器拦截器。这些拦截器为服务器全局配置。

- 容器拦截器

  当 JBoss EAP 充当服务器时，容器拦截器会运行。这些拦截器在 Jakarta Enterprise Beans 容器中配置。

自定义拦截器类应添加到模块中，并存储在 $JBOSS_HOME/modules 目录中。

7.1.1. Interceptor Chain

自定义拦截器在拦截器链中的特定点执行。

为 Jakarta Enterprise Beans 配置的容器拦截器在 Wildfly 提供的拦截器（如安全拦截器或事务管理拦截器）之前执行。因此，容器拦截器可以在调用 Wildfly 拦截器或全局拦截器前处理或配置上下文数据。

服务器和客户端拦截器会在 Wildfly 特定拦截器后执行。

7.1.2. 自定义客户端 Interceptors

自定义客户端拦截器实施 org.jboss.ejb.client.EJBClientInterceptor 接口。

此外，还应包含 org.jboss.ejb.client.EJBClientInvocationContext 界面。

以下代码演示了客户端拦截器示例。

客户端拦截器代码示例

```
package org.foo;
import org.jboss.ejb.client.EJBClientInterceptor;
import org.jboss.ejb.client.EJBClientInvocationContext;
public class FooInterceptor implements EJBClientInterceptor {
   @Override
   public void handleInvocation(EJBClientInvocationContext context) throws Exception {
      context.sendRequest();
   }
   @Override
   public Object handleInvocationResult(EJBClientInvocationContext context) throws
Exception {
      return context.getResult();
   }
}
```

7.1.3. 自定义服务器 Interceptors

服务器拦截器使用 @javax.annotation.AroundInvoke 注释或 javax.interceptor.AroundTimeout 注释来标记 bean 上调用期间调用的方法。

以下代码演示了一个服务器拦截器示例。

服务器拦截器代码示例

```
package org.testsuite.ejb.serverinterceptor;
import javax.annotation.PostConstruct;
```

```
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;
public class TestServerInterceptor {
   @AroundInvoke
   public Object aroundInvoke(final InvocationContext invocationContext) throws Exception {

      return invocationContext.proceed();
   }
}
```

7.1.4. 自定义容器拦截器

容器拦截器使用 @javax.annotation.AroundInvoke 注释或 javax.interceptor.AroundTimeout 注释来标记 bean 调用期间调用的方法。

根据 Jakarta Enterprise Beans 3.2 规范的定义，标准 Jakarta EE 拦截器应该在容器完成了安全性上下文传播、交易管理和其他容器提供的调用处理后运行。

以下代码演示了一个拦截器类，用于标记调用的 iAmAround 方法。

容器拦截器代码示例

```
public class ClassLevelContainerInterceptor {
   @AroundInvoke
   private Object iAmAround(final InvocationContext invocationContext) throws Exception {
      return this.getClass().getName() + " " + invocationContext.proceed();
   }
}
```

容器拦截器和 Jakarta EE Interceptor API 之间的差异

虽然容器拦截器被建模为类似于 Jakarta EE 拦截器，但 API 的语义存在一些差别。例如，容器拦截器调用 javax.interceptor.InvocationContext.getTarget（） 方法是非法的，因为这些拦截器会在 Jakarta Enterprise Beans 组件设置或实例化前被调用。

7.1.5. 配置容器拦截器

容器拦截器使用标准的 Jakarta EE 拦截器库。

因此，它们将 ejb-jar.xml 文件中允许的相同的 XSD 元素用于 ejb-jar 部署描述符的 3.2 版本。

由于它们基于标准的 Jakarta EE 拦截器库，因此只能使用部署描述符来配置容器拦截器。按照设计，应用不需要任何特定于 JBoss EAP 的注释或其他库依赖关系。

配置容器拦截器：

1. 在 Jakarta Enterprise Beans 部署的 META-INF/' 目录中创建一个 jboss-ejb3.xml 文件。
2. 在描述符文件中配置容器拦截器元素。
   a. 使用 urn:container-interceptors:1.0 命名空间来指定容器拦截器元素的配置。
   b. 使用 <container-interceptors> 元素来指定容器拦截器。
   c. 使用 <interceptor-binding> 元素将容器拦截器绑定到 Jakarta Enterprise Beans。可以使用以下任一方式绑定拦截器：
      - 使用通配符(*)将拦截器绑定到部署中的所有 Jakarta Enterprise Beans。
      - 使用特定的 Jakarta Enterprise Beans 名称，在个人 Bean 级别绑定拦截器。
      - 在 Jakarta 企业 Bean 的具体方法级别上绑定拦截器。

注意

这些元素使用 Jakarta Enterprise Beans 3.2 XSD 配置，其方式与 Jakarta EE 拦截器相同。

以下示例描述符文件演示了配置选项。

容器 Interceptor jboss-ejb3.xml 文件示例

```
<jboss xmlns="http://www.jboss.com/xml/ns/javaee"
      xmlns:jee="http://java.sun.com/xml/ns/javaee"
      xmlns:ci ="urn:container-interceptors:1.0">
   <jee:assembly-descriptor>
      <ci:container-interceptors>
         <!-- Default interceptor -->
         <jee:interceptor-binding>
            <ejb-name>*</ejb-name>
            <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.ContainerInterceptorOne</intercept
or-class>
         </jee:interceptor-binding>
         <!-- Class level container-interceptor -->
         <jee:interceptor-binding>
            <ejb-name>AnotherFlowTrackingBean</ejb-name>
            <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.ClassLevelContainerInterceptor</i
nterceptor-class>
         </jee:interceptor-binding>
         <!-- Method specific container-interceptor -->
         <jee:interceptor-binding>
            <ejb-name>AnotherFlowTrackingBean</ejb-name>
            <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.MethodSpecificContainerIntercept
or</interceptor-class>
            <method>
               <method-name>echoWithMethodSpecificContainerInterceptor</method-name>
            </method>
         </jee:interceptor-binding>
         <!-- container interceptors in a specific order -->
         <jee:interceptor-binding>
            <ejb-name>AnotherFlowTrackingBean</ejb-name>
            <interceptor-order>
               <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.ClassLevelContainerInterceptor</i
nterceptor-class>
               <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.MethodSpecificContainerIntercept
or</interceptor-class>
               <interceptor-
class>org.jboss.as.test.integration.ejb.container.interceptor.ContainerInterceptorOne</intercept
or-class>
            </interceptor-order>
            <method>
               <method-name>echoInSpecificOrderOfContainerInterceptors</method-name>
            </method>
         </jee:interceptor-binding>
```

```
        </ci:container-interceptors>
    </jee:assembly-descriptor>
</jboss>
```

allow-ejb-name-regex 属性允许您在拦截器绑定中使用正则表达式，并将拦截器映射到与指定正则表达式匹配的所有 Bean。使用以下命令，将 ejb3 子系统 的 allow-ejb-name-regex 属性 启用为 true ：

```
/subsystem=ejb3:write-attribute(name=allow-ejb-name-regex,value=true)
```

urn:container-interceptors:1.0 命名空间的架构位于 http://www.jboss.org/schema/jbossas/jboss-ejb-container-interceptors_1_0.xsd。

7.1.6. 服务器和客户端拦截器配置

服务器和客户端拦截器将全局添加到所用配置文件中的 JBoss EAP 配置中。

服务器拦截器添加到 ejb3 子系统配置中的 <server-interceptors> 元素 中。客户端拦截器添加到 ejb3 子系统配置中的 <client-interceptors> 元素 中。

以下示例演示了如何添加服务器拦截器。

```
/subsystem=ejb3:list-add(name=server-interceptors,value={module=org.abccorp:tracing-
interceptors:1.0,class=org.abccorp.TracingInterceptor})
```

以下示例演示了如何添加客户端拦截器。

```
/subsystem=ejb3:list-add(name=client-interceptors,value=
{module=org.abccorp:clientInterceptor:1.0,class=org.abccorp.clientInterceptor})
```

每当添加服务器拦截器或客户端拦截器或更改拦截器配置时，必须重新加载服务器。

7.1.7. 更改安全性上下文身份

您可以向经过身份验证的用户授予权限，以不同用户身份切换身份和对现有连接执行请求，而不是打开多个客户端连接。

默认情况下，当您远程调用部署到应用服务器的 Jakarta Enterprise Beans 时，与服务器的连接将进行身份验证，并且使用连接的任何后续请求都使用原始身份验证的身份执行。对于客户端到服务器调用和服务器对服务器调用也是如此。如果您需要使用来自同一客户端的不同身份，通常您必须打开与服务器的多个连接，以便每个连接都作为不同的身份进行身份验证。相反，您可以允许经过身份验证的用户更改身份。

更改经过身份验证的用户的身份：

1. 在拦截器代码中实施身份更改。

- 客户端拦截器

  拦截器必须通过上下文数据映射传递请求的身份，该映射可通过对 EJBClientInvocationContext.getContext.getContext（）的调用来获取。以下示例代码演示了可切换身份的客户端拦截器。

  客户端拦截器代码示例

```
public class ClientSecurityInterceptor implements EJBClientInterceptor {

   public void handleInvocation(EJBClientInvocationContext context) throws
Exception {
      Principal currentPrincipal = SecurityActions.securityContextGetPrincipal();

      if (currentPrincipal != null) {
         Map<String, Object> contextData = context.getContextData();
         contextData.put(ServerSecurityInterceptor.DELEGATED_USER_KEY,
currentPrincipal.getName());
      }
      context.sendRequest();
   }

   public Object handleInvocationResult(EJBClientInvocationContext context)
throws Exception {
      return context.getResult();
   }
}
```

- 容器和服务器拦截器

这些拦截器接收包含身份的 InvocationContext，并请求切换到新身份。以下代码演示了容器拦截器的隔离示例：

容器拦截器代码示例

```
public class ServerSecurityInterceptor {

   private static final Logger logger =
Logger.getLogger(ServerSecurityInterceptor.class);

   static final String DELEGATED_USER_KEY =
ServerSecurityInterceptor.class.getName() + ".DelegationUser";

   @AroundInvoke
   public Object aroundInvoke(final InvocationContext invocationContext) throws
Exception {
      Principal desiredUser = null;
      UserPrincipal connectionUser = null;

      Map<String, Object> contextData = invocationContext.getContextData();
      if (contextData.containsKey(DELEGATED_USER_KEY)) {
         desiredUser = new SimplePrincipal((String)
contextData.get(DELEGATED_USER_KEY));

         Collection<Principal> connectionPrincipals =
SecurityActions.getConnectionPrincipals();

         if (connectionPrincipals != null) {
            for (Principal current : connectionPrincipals) {
               if (current instanceof UserPrincipal) {
                  connectionUser = (UserPrincipal) current;
                  break;
               }
            }

         } else {
            throw new IllegalStateException("Delegation user requested but no user
on connection found.");
         }
      }

      ContextStateCache stateCache = null;
      try {
         if (desiredUser != null && connectionUser != null
            && (desiredUser.getName().equals(connectionUser.getName()) == false))
{
            // The final part of this check is to verify that the change does actually
indicate a change in user.
```

```
            try {
                // We have been requested to use an authentication token
                // so now we attempt the switch.
                stateCache = SecurityActions.pushIdentity(desiredUser, new
OuterUserCredential(connectionUser));
            } catch (Exception e) {
                logger.error("Failed to switch security context for user", e);
                // Don't propagate the exception stacktrace back to the client for
security reasons
                throw new EJBAccessException("Unable to attempt switching of
user.");
            }
        }

        return invocationContext.proceed();
    } finally {
        // switch back to original context
        if (stateCache != null) {
            SecurityActions.popIdentity(stateCache);;
        }
    }
  }
```

2.

应用可以通过编程方式或使用服务加载器机制将客户端拦截器插入到 EJBClientContext 拦截器链中。有关配置客户端拦截器的步骤，请参阅 在应用程序中使用客户端拦截器。

3.

创建 Jakarta 身份验证登录模块。

Jakarta Authentication LoginModule 组件负责验证允许用户是否按请求的身份执行请求。以下 abridged 代码示例显示了执行登录和验证的方法：

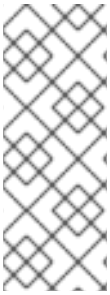
LoginModule 代码示例

```
@SuppressWarnings("unchecked")
@Override
public boolean login() throws LoginException {
    if (super.login() == true) {
        log.debug("super.login()==true");
        return true;
    }

    // Time to see if this is a delegation request.
    NameCallback ncb = new NameCallback("Username:");
    ObjectCallback ocb = new ObjectCallback("Password:");
```

```
        try {
            callbackHandler.handle(new Callback[] { ncb, ocb });
        } catch (Exception e) {
            if (e instanceof RuntimeException) {
                throw (RuntimeException) e;
            }
            // If the CallbackHandler can not handle the required callbacks then no chance.
            return false;
        }

        String name = ncb.getName();
        Object credential = ocb.getCredential();

        if (credential instanceof OuterUserCredential) {
            // This credential type will only be seen for a delegation request, if not seen
then the request is not for us.

            if (delegationAcceptable(name, (OuterUserCredential) credential)) {
                identity = new SimplePrincipal(name);
                if (getUseFirstPass()) {
                    String userName = identity.getName();
                    if (log.isDebugEnabled())
                        log.debug("Storing username '" + userName + "' and empty password");
                    // Add the username and an empty password to the shared state map
                    sharedState.put("javax.security.auth.login.name", identity);
                    sharedState.put("javax.security.auth.login.password", "");
                }
                loginOk = true;
                return true;
            }
        }
        return false; // Attempted login but not successful.
    }

    // Make a trust user to decide if the user switch is acceptable.
    protected boolean delegationAcceptable(String requestedUser, OuterUserCredential
connectionUser) {
    if (delegationMappings == null) {
        return false;
    }

    String[] allowedMappings = loadPropertyValue(connectionUser.getName(),
connectionUser.getRealm());
    if (allowedMappings.length == 1 && "*".equals(allowedMappings[0])) {
        // A wild card mapping was found.
        return true;
    }
    for (String current : allowedMappings) {
        if (requestedUser.equals(current)) {
            return true;
        }
    }
    return false;
}
```

7.1.8. 在应用程序中使用客户端拦截器

应用可以使用服务加载器，或使用 ClientInterceptors 注释，以编程方式将客户端拦截器插入到 EJBClientContext 侦听器链中。



注意

EJBClientInterceptor 可通过调用 org.jboss.ejb.client.EJBClientInvocationContext#addReturnedContext(*String key*) 从服务器端调用上下文请求特定数据。如果上下文数据映射中提供的键下存在请求的数据，则会将其发送到客户端。

7.1.8.1. 动态插入客户端拦截器程序

创建带有拦截器注册的 EJBClientContext 后，插入拦截器。

以下代码演示了如何使用拦截器注册创建 EJBClientContext ：

```
EJBClientContext ctxWithInterceptors =
EJBClientContext.getCurrent().withAddedInterceptors(clientInterceptor);
```

创建 EJBClientContext 后，有两个选项可用于插入拦截器：

- 您可以使用 Callable 操作，使用应用的 EJBClientContext 运行以下代码：在 Callable 操作中执行的 Jakarta Enterprise Beans 调用将应用客户端侧拦截器：

```
ctxWithInterceptors.runCallable(() -> {
    // perform the calls which should use the interceptor
})
```

- 或者，您可以将新创建的 EJBClientContext 标记为新默认值：

```
EJBClientContext.getContextManager().setThreadDefault(ctxWithInterceptors);
```

7.1.8.2. 使用服务加载器机制插入客户端拦截器

创建 META-INF/services/org.jboss.ejb.client.EJBClientInterceptor 文件，并将它放到客户端应用的类路径中。

文件的规则由 Java ServiceLoader 机制 规定。

- 此文件应当包含单独的行，用于 Jakarta Enterprise Beans 客户端拦截器实施的每个完全限定类名称。
- 在课程路径中必须有 Jakarta Enterprise Beans 客户端拦截器类。

使用服务加载器机制添加的 Jakarta Enterprise Beans 客户端拦截器将按照在类路径中找到并添加到客户端拦截器链中的顺序添加。

7.1.8.3. 使用 ClientInterceptor 注解插入客户端拦截器

通过 @org.jboss.ejb.client.annotation.ClientInterceptors 注释，您可以将 Jakarta Enterprise Beans 拦截器放在远程调用的客户端中。

```
import org.jboss.ejb.client.annotation.ClientInterceptors;
@ClientInterceptors({HelloClientInterceptor.class})

public interface HelloBeanRemote {
  public String hello();
}
```

第 8 章 集群的 JAKARTA ENTERPRISE BEANS

8.1. 关于集群 JAKARTA ENTERPRISE BEANS

Jakarta Enterprise Beans 组件可以针对高可用性情景进行群集化。它们使用与 HTTP 组件不同的协议，因此它们以不同的方式集群。企业 Bean 2 和 3 个有状态和无状态 Bean 可以集群。

有关单例的信息，请参阅 JBoss EAP *开发指南*中的 HA 单例服务

8.2. JAKARTA ENTERPRISE BEANS 客户代码简化

在调用 Jakarta Enterprise Beans 服务器集群组件时，您可以简化 Jakarta Enterprise Beans 客户端代码。以下流程概述了简化 Jakarta Enterprise Beans 客户端代码的多种方法：

- 初始上下文查找
- 远程 Jakarta Enterprise Beans 配置文件
- Jakarta Enterprise Beans 的自动交易粘性
- 集群环境中的 Jakarta Enterprise Beans 事务

注意

jboss-ejb-client.properties 文件的使用已弃用，而是使用 wildfly-config.xml 文件。

8.3. 部署集群 JAKARTA ENTERPRISE BEANS

JBoss EAP 7.4 的 HA 配置文件中提供了集群支持。启动启用了 HA 功能的单机服务器涉及从 standalone-ha.xml 或 standalone- full-ha.xml 文件启动它：

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
```

这将启动具有 HA 功能的服务器的单个实例。

要查看群集的优势，您将需要多个服务器实例。因此，让我们启动具有 HA 功能的另一台服务器。该服务器的另一个实例可以在同一台计算机上，也可以位于其他计算机上。如果它位于同一台机器上，则需要处理两个问题：

- 为第二个实例传递端口偏移
- 确保每个服务器实例都具有唯一的 jboss.node.name 系统属性。

您可以通过将以下两个系统属性传递给启动命令来做到这一点：

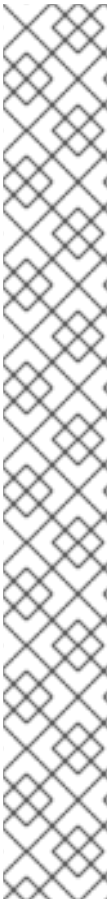
```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -Djboss.socket.binding.port-offset=PORT_OFFSET -Djboss.node.name=UNIQUE_NODE_NAME
```

遵循您也习惯于将 Jakarta 企业 Bean 部署至该实例的方法。

警告

仅将应用部署到群集服务器独立实例的一个节点上，并不意味着它将自动部署到其他集群实例。您还必须明确将它部署到其他独立集群实例上。或者，您可以在域模式中启动服务器，以便将部署部署到服务器组中的所有服务器上。

现在，您已在两个实例上部署了集群 Jakarta Enterprise Beans 的应用程序，Jkarta 企业 Bean 现在能够利用集群功能。



注意

从 JBoss EAP 7 开始，如果使用 HA 配置文件启动 JBoss EAP，则将复制有状态会话 Bean 的状态。您不再需要使用 @Clustered 注释来启用集群行为。

您可以通过在 @Stateful 注释中将 passivationCapable 设置为 false 来为有状态会话 Bean 禁用复制：

```
@Stateful(passivationCapable=false)
```

这会指示服务器使用 passivation-disabled-cache-ref 定义的 ejb 缓存， 而不是 cache-ref。

要全局禁用有状态会话 Bean 的复制，请使用以下管理 CLI 命令：

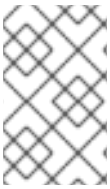
```
/subsystem=ejb3:write-attribute(name=default-sfsb-cache,value=simple)
```

8.4. 集群 JAKARTA ENTERPRISE BEANS 的故障转移

集群 Jakarta Enterprise Beans 具有故障转移功能。@Stateful Jakarta Enterprise Beans 的状态在集群节点之间复制，以便在群集中的一个节点发生故障时，某些其他节点将接管调用。

在某些情况下，比如集群中的服务器崩溃时，Jakarta Enterprise Beans 客户端可能会收到异常而不是响应。Jakarta Enterprise Beans 客户端库将在安全时自动重试调用，具体取决于发生的故障类型。但是，如果请求失败，并且无法最终确定可以安全地重试，您可以根据您的环境处理异常。但是，您可以使用自定义拦截器添加额外的重试行为。

8.5. 远程单机客户端



注意

jboss-ejb-client.properties 文件的使用已弃用，而是使用 wildfly-config.xml 文件。

独立的远程客户端可以使用 Java 命名和目录接口方法或本机 JBoss Jakarta Enterprise Beans 客户端 API 与服务器通信。需要注意的是，当您调用集群 Jakarta Enterprise Beans 部署时，您不必列出集群中的所有服务器。由于群集中群集节点的动态性质，这就不可行。

远程客户端必须仅列出其中一个具有集群功能的服务器。此服务器将充当客户端和群集节点之间群集拓扑通信的起点。

请注意，您必须在 **jboss-ejb-client.properties** 配置文件中配置 **ejb** 集群：

```
remote.clusters=ejb
remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED=false
```

8.6. 集群拓扑通信

注意

jboss-ejb-client.properties 文件的使用已弃用，而是使用 **wildfly-config.xml** 文件。

当客户端连接到服务器时，如果服务器具有集群功能，**JBoss Jakarta Enterprise Beans** 客户端实施会在内部与服务器通信以提供集群拓扑信息。例如，假设服务器 **X** 列为要连接的初始服务器，当客户端连接到服务器 **X** 时，服务器会将异步群集拓扑消息发回给客户端。此拓扑消息由集群名称和属于集群的节点的信息组成。需要时，节点信息包括要连接的节点地址和端口号。因此在此示例中，服务器 **X** 将发回由属于该群集的其他服务器 **Y** 组成的群集拓扑。

如果集群为 **Jakarta Enterprise Beans**，则调用流程分为两个步骤。

1. 为有状态 **Bean** 创建会话，当您为该 **Bean** 进行 **Java** 命名和目录接口查找时会出现这种情况。
2. 调用返回的代理。

查找有状态 **Bean** 在内部会触发从客户端到服务器同步会话创建请求。在这种情况下，会话创建请求将转到 **server X**，因为它已在 **jboss-ejb-client.properties** 文件中配置。由于服务器 **X** 已群集化，因此它将返回会话 **ID** 并发回 *该* 会话的关联。如果是集群服务器，则 *关联性* 等同于有状态 **Bean** 所属群集的名称。对于非集群 **Bean**，关联性是创建会话的节点名称。这种 *关联性* 将帮助 **Jakarta Enterprise Beans** 客户端根据需要将代理上的调用路由到集群中用于集群 **Bean** 的节点，或路由到非集群 **Bean** 的特定节点。在此会话创建请求期间，服务器 **X** 也会发回包含群集拓扑的异步消息。**JBoss Jakarta Enterprise Beans** 客户端实施将记录此拓扑信息，并在稍后将其用于连接到集群内的节点，并在需要时将调用路由到这些节点。

要了解故障转移的工作方式，请考虑同一服务器 X 示例作为起点，以及寻找有状态 Bean 并调用它的客户端应用。在这些调用过程中，客户端从服务器收集集群拓扑信息。假设出于某种原因，服务器 X 停机，客户端应用随后在代理上调用。在此阶段，JBoss Jakarta 企业 Beans 客户端实施必须了解 *相关性*，在这种情况下，这是集群相关性。从客户端具有的集群拓扑信息中，它知道群集有两个节点：服务器 X 和 server Y。当调用到达时，客户端会注意到服务器 X 已停机，因此它使用选择器从群集节点获取合适的节点。当选择器从集群节点返回节点时，如果之前尚未创建连接，JBoss Jakarta Enterprise Beans 客户端会创建一个连接该节点，并为它创建一个 Jakarta Enterprise Beans 接收器。在本示例中，集群中唯一的其他节点是 server Y，选择器将返回服务器 Y 作为节点，JBoss Jakarta Enterprise Beans 客户端将使用它来创建 Jakarta Enterprise Beans 接收器，并使用此接收器在代理上传递调用。实际上，调用现在切换到集群中的不同节点。

8.7. JAKARTA ENTERPRISE BEANS 的自动交易粘性

从与 Jakarta Enterprise Beans 代理相同的上下文查找事务对象，以同一主机为目标。如果上下文是多主机或集群，则具有活跃的事务会将调用上下文固定到同一节点。

这个行为取决于您是否已溢出您的事务或使用的是远程用户事务。

对于流出事务，当应用程序在特定节点上查找时，同一事务中对该应用的所有调用都将尝试将此节点作为目标。已收到溢出事务的节点将优先于尚未接收流出事务的节点。

对于远程用户事务，第一次成功调用将交易锁定到给定节点，此事务下的后续调用必须进入同一节点，否则将引发异常。

8.8. 另一个实例上的远程客户端

本节介绍在 JBoss EAP 实例上部署的客户端应用如何调用部署在另一个 JBoss EAP 实例上的集群状态 Bean。

在以下示例中，涉及三台服务器：服务器 X 和 Y 均属于群集，并且在其上部署了群集 Jakarta Enterprise Bean。还有另一个服务器实例服务器 C，它们不一定具有群集功能。服务器 C 充当客户端，上面有想要调用服务器 X 和 Y 上部署的集群 Bean 并实现故障转移的部署。

配置在 jboss-ejb-client.xml 文件中完成，该文件指向到其他服务器的远程出站连接。jboss-ejb-client.xml 文件中的配置正在服务器 C 部署中，因为 server C 是客户端。客户端配置不需要指向所有群集节点，而仅指向其中一个节点。这将充当通信的起点。

在这种情况下，远程出站连接从服务器 C 创建至服务器 X，然后将服务器 X 用作通信的起点。与远程单机客户端类似，服务器 C 上的应用查找有状态 Bean 时，会话创建请求将发送到服务器 X，该服务器返

回会话 ID 及其群集关联性。服务器 X 也会将异步消息发回到包含集群拓扑的服务器 C。此拓扑信息包括服务器 Y 的节点信息，因为服务器 Y 属于群集和服务器 X。代理上的 Subsequent 调用将相应地路由到群集中的节点。如果服务器 X 停机，如前文中所述，将选择与群集不同的节点，并将调用转发到该节点。

其他 JBoss EAP 实例上的远程单机客户端和远程客户端在故障转移方面表现相似。

8.9. 独立和服务器内客户端配置

注意

jboss-ejb-client.properties 文件的使用已弃用，而是使用 wildfly-config.xml 文件。

要将 Jakarta Enterprise Beans 客户端连接到集群的 Jakarta Enterprise Beans 应用，您需要扩展独立 Jakarta Enterprise Beans 客户端或服务器内 Jakarta Enterprise Beans 客户端中的现有配置，以包含集群连接配置。适用于独立 Jakarta Enterprise Beans 客户端的 jboss-ejb-client.properties，甚至是用于服务器端应用的 jboss-ejb-client.xml 文件，必须扩展为包含群集配置。

注意

Jakarta Enterprise Beans 客户端是在远程服务器上使用 Jakarta 企业 Bean 的任何程序。当 Jakarta Enterprise Beans 客户端调用远程服务器本身在服务器内部 时，客户端就位于服务器内。换句话说，调用另一 JBoss EAP 实例的 JBoss EAP 实例将被视为服务器内客户端。

本例显示了独立 Jakarta Enterprise Beans 客户端所需的额外集群配置。

```
remote.clusters=ejb
remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED=false
remote.cluster.ejb.username=test
remote.cluster.ejb.password=password
```

如果应用程序使用 remote-outbound-connection，您需要配置 jboss-ejb-client.xml 文件并添加集群配置，如下例所示：

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.2" xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
  <client-context>
    <ejb-receivers>
      <!-- this is the connection to access the app-one -->
```

```
    <remoting-ejb-receiver outbound-connection-ref="remote-ejb-connection-1" />
    <!-- this is the connection to access the app-two -->
    <remoting-ejb-receiver outbound-connection-ref="remote-ejb-connection-2" />
  </ejb-receivers>

  <!-- If an outbound connection connects to a cluster,
         a list of members is provided after successful connection.
       To connect to this node this cluster element must be defined. -->

  <clusters>
    <!-- cluster of remote-ejb-connection-1 -->
    <cluster name="ejb" security-realm="ejb-security-realm-1" username="quickuser1">
      <connection-creation-options>
      <property name="org.xnio.Options.SSL_ENABLED" value="false" />
      <property name="org.xnio.Options.SASL_POLICY_NOANONYMOUS" value="false" />
      </connection-creation-options>
    </cluster>
  </clusters>
 </client-context>
</jboss-ejb-client>
```

有关远程出站连接的更多信息，请参阅 JBoss EAP *配置指南中* 的关于删除 子系统 的更多信息。

注意

对于安全连接，您需要将凭证添加到集群配置中，以避免身份验证异常。

8.10. 为 JAKARTA ENTERPRISE BEANS 调用实施自定义负载平衡策略

注意

jboss-ejb-client.properties 文件的使用已弃用，而是使用 **wildfly-config.xml** 文件。

可以实施备用或自定义负载平衡策略，以平衡应用的 Jakarta 企业 Beans 调用服务器。

您可以为 Jakarta Enterprise Beans 调用实施 AllClusterNodeSelector。AllClusterNodeSelector 的节点选择行为类似于默认选择器，但 AllClusterNodeSelector 使用所有可用的集群节点（即使大型集群（node 的数量 > 20）。如果返回一个未连接的集群节点，它会自动打开。以下示例演示了 AllClusterNodeSelector 实现：

```
package org.jboss.as.quickstarts.ejb.clients.selector;

import java.util.Arrays;
import java.util.Random;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;

import org.jboss.ejb.client.ClusterNodeSelector;
public class AllClusterNodeSelector implements ClusterNodeSelector {
  private static final Logger LOGGER =
Logger.getLogger(AllClusterNodeSelector.class.getName());

  @Override
  public String selectNode(final String clusterName, final String[] connectedNodes, final
String[] availableNodes) {
    if(LOGGER.isLoggable(Level.FINER)) {
      LOGGER.finer("INSTANCE "+this+ " : cluster:"+clusterName+"
connected:"+Arrays.deepToString(connectedNodes)+"
available:"+Arrays.deepToString(availableNodes));
    }

    if (availableNodes.length == 1) {
       return availableNodes[0];
    }
    final Random random = new Random();
    final int randomSelection = random.nextInt(availableNodes.length);
    return availableNodes[randomSelection];
  }

}
```

您还可以为 Jakarta Enterprise Beans 调用实施 SimpleLoadFactorNodeSelector。SimpleLoadFactorNodeSelector 中的 负载均衡会根据负载因数进行。负载因数(2/3/4)根据节点的名称(A/B/C)计算，不论每个节点上的负载如何。以下示例显示了 SimpleLoadFactorNodeSelector 实现：

```
package org.jboss.as.quickstarts.ejb.clients.selector;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.jboss.ejb.client.DeploymentNodeSelector;
public class SimpleLoadFactorNodeSelector implements DeploymentNodeSelector {
  private static final Logger LOGGER =
Logger.getLogger(SimpleLoadFactorNodeSelector.class.getName());
  private final Map<String, List<String>[]> nodes = new HashMap<String, List<String>[]>();
  private final Map<String, Integer> cursor = new HashMap<String, Integer>();

  private ArrayList<String> calculateNodes(Collection<String> eligibleNodes) {
    ArrayList<String> nodeList = new ArrayList<String>();

    for (String string : eligibleNodes) {
```

```
    if(string.contains("A") || string.contains("2")) {
      nodeList.add(string);
      nodeList.add(string);
    } else if(string.contains("B") || string.contains("3")) {
      nodeList.add(string);
      nodeList.add(string);
      nodeList.add(string);
    } else if(string.contains("C") || string.contains("4")) {
      nodeList.add(string);
      nodeList.add(string);
      nodeList.add(string);
      nodeList.add(string);
    }
   }
   return nodeList;
  }

  @SuppressWarnings("unchecked")
  private void checkNodeNames(String[] eligibleNodes, String key) {
   if(!nodes.containsKey(key) || nodes.get(key)[0].size() != eligibleNodes.length ||
!nodes.get(key)[0].containsAll(Arrays.asList(eligibleNodes))) {
    // must be synchronized as the client might call it concurrent
    synchronized (nodes) {
     if(!nodes.containsKey(key) || nodes.get(key)[0].size() != eligibleNodes.length ||
!nodes.get(key)[0].containsAll(Arrays.asList(eligibleNodes))) {
       ArrayList<String> nodeList = new ArrayList<String>();
       nodeList.addAll(Arrays.asList(eligibleNodes));

       nodes.put(key, new List[] { nodeList, calculateNodes(nodeList) });
     }
    }
   }
  }
   private synchronized String nextNode(String key) {
   Integer c = cursor.get(key);
   List<String> nodeList = nodes.get(key)[1];

   if(c == null || c >= nodeList.size()) {
    c = Integer.valueOf(0);
   }

   String node = nodeList.get(c);
   cursor.put(key, Integer.valueOf(c + 1));

   return node;
  }

  @Override
  public String selectNode(String[] eligibleNodes, String appName, String moduleName, String
distinctName) {
   if (LOGGER.isLoggable(Level.FINER)) {
    LOGGER.finer("INSTANCE " + this + " : nodes:" + Arrays.deepToString(eligibleNodes) + "
appName:" + appName + " moduleName:" + moduleName
      + " distinctName:" + distinctName);
   }
```

```
    // if there is only one there is no sense to choice
    if (eligibleNodes.length == 1) {
      return eligibleNodes[0];
    }
    final String key = appName + "|" + moduleName + "|" + distinctName;

    checkNodeNames(eligibleNodes, key);
    return nextNode(key);
  }
}
```

配置 jboss-ejb-client.properties 文件

您需要使用实现类（AllClusterNodeSelector 或 SimpleLoadF actorNodeSelector ）的名称添加 remote.cluster.ejb.clusternode.selector属性。选择器将显示调用时可用的所有已配置服务器。以下示例使用 AllClusterNodeSelector 作为集群节点选择器：

```
remote.clusters=ejb
remote.cluster.ejb.clusternode.selector=org.jboss.as.quickstarts.ejb.clients.selector.AllClusterNodeSelec
or
remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED=false
remote.cluster.ejb.username=test
remote.cluster.ejb.password=password

remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=one,two
remote.connection.one.host=localhost
remote.connection.one.port = 8080
remote.connection.one.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.one.username=user
remote.connection.one.password=user123
remote.connection.two.host=localhost
remote.connection.two.port = 8180
remote.connection.two.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

使用 Jakarta Enterprise Beans 客户端 API

您需要将属性 remote.cluster.ejb.clusternode.selector 添加到 PropertiesBasedEJBClientConfiguration 构造器的列表中。以下示例使用 AllClusterNodeSelector 作为集群节点选择器：

```
Properties p = new Properties();
p.put("remote.clusters", "ejb");
p.put("remote.cluster.ejb.clusternode.selector",
"org.jboss.as.quickstarts.ejb.clients.selector.AllClusterNodeSelector");
p.put("remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS",
"false");
p.put("remote.cluster.ejb.connect.options.org.xnio.Options.SSL_ENABLED", "false");
p.put("remote.cluster.ejb.username", "test");
p.put("remote.cluster.ejb.password", "password");

p.put("remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED", "false");
```

```
p.put("remote.connections", "one,two");
p.put("remote.connection.one.port", "8080");
p.put("remote.connection.one.host", "localhost");
p.put("remote.connection.two.port", "8180");
p.put("remote.connection.two.host", "localhost");

EJBClientConfiguration cc = new PropertiesBasedEJBClientConfiguration(p);
ContextSelector<EJBClientContext> selector = new ConfigBasedEJBClientContextSelector(cc);
EJBClientContext.setSelector(selector);

p = new Properties();
p.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
InitialContext context = new InitialContext(p);
```

配置 jboss-ejb-client.xml 文件

要使用服务器负载平衡策略以进行服务器通信，请将类与应用打包，并在位于 META-INF 文件夹下的 jboss-ejb-client.xml 设置中进行配置。以下示例使用 AllClusterNodeSelector 作为集群节点选择器：

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.2" xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
  <client-context deployment-node-selector="org.jboss.ejb.client.DeploymentNodeSelector">
    <ejb-receivers>
      <!-- This is the connection to access the application. -->
      <remoting-ejb-receiver outbound-connection-ref="remote-ejb-connection-1" />
    </ejb-receivers>
    <!-- Specify the cluster configurations applicable for this client context -->
    <clusters>
      <!-- Configure the cluster of remote-ejb-connection-1. -->
      <cluster name="ejb" security-realm="ejb-security-realm-1" username="test" cluster-node-selector="org.jboss.as.quickstarts.ejb.clients.selector.AllClusterNodeSelector">
        <connection-creation-options>
          <property name="org.xnio.Options.SSL_ENABLED" value="false" />
          <property name="org.xnio.Options.SASL_POLICY_NOANONYMOUS" value="false" />
        </connection-creation-options>
      </cluster>
    </clusters>
  </client-context>
</jboss-ejb-client>
```

要将上述配置与安全性一起使用，您需要将 ejb-security-realm-1 添加到客户端-服务器配置。下例演示了添加安全域的 CLI 命令(ejb-security-realm-1)该值是用户 "test" 的 base64 编码密码：

```
/core-service=management/security-realm=ejb-security-realm-1:add()
/core-service=management/security-realm=ejb-security-realm-1/server-identity=secret:add(value=cXVpY2sxMjMr)
```

如果负载平衡策略应用于服务器通信，可以将类与应用一起打包，或打包为模块。此类配置在顶级 EAR 存档的 META -INF 目录中的 jboss- ejb-client 设置文件中。以下示例使用 roundRobinNodeSelector 作为部署节点选择器。

```
<jboss-ejb-client xmlns="urn:jboss:ejb-client:1.2">
  <client-context deployment-node-selector="org.jboss.example.RoundRobinNodeSelector">
    <ejb-receivers>
      <remoting-ejb-receiver outbound-connection-ref="..."/>
    </ejb-receivers>
    ...
  </client-context>
</jboss-ejb-client>
```

注意

如果您正在运行单机服务器，请使用 start 选项 -Djboss.node.name= 或服务器配置文件 standalone.xml 来配置服务器名称。确保服务器名称唯一。如果您正在运行受管域，主机控制器会自动验证名称是否唯一。

8.11. 集群环境中的 JAKARTA ENTERPRISE BEANS 事务

如果客户端代码调用集群 Jakarta Enterprise Beans，则会自动设置集群关联性。如果您在客户端管理事务，您可以选择以 集群中的特定节点为目标， 或者 允许客户端精细选择集群节点来处理事务。本节描述了这两个选项。

Jakarta Enterprise Beans Transactions 以特定节点为目标

您可以针对集群中的特定节点来按照以下步骤处理事务。

1. 在创建 InitialContext 时，使用 PROVIDER_URL 属性指定目标集群节点地址。

```
props.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
...
InitialContext ctx = new InitialContext(props);
```

2. 在客户端中，从 InitialContext 查找 the txn:RemoteUserTransaction。

```
UserTransaction ut = (UserTransaction)ctx.lookup("txn:RemoteUserTransaction");
```

您可以通过将 PROVIDER_URL 属性设置为服务器的 URL，然后查找 up txn:UserTransaction，然后查找 ttxn:UserTransaction，如下代码示例所示：

```
final Hashtable<String, String> jndiProperties = new Hashtable<>();
jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
jndiProperties.put(Context.PROVIDER_URL, "remote+http://localhost:8080");
```

```
final Context context = new InitialContext(jndiProperties);

SecuredEJBRemote reference = (SecuredEJBRemote)
context.lookup("txn:UserTransaction");
```

在实际调用发生之前，UserTransaction 不会绑定到任何特定的目的地。在调用时，此 UserTransaction 绑定到整个事务生命周期中的相应目标。

在开始 UserTransaction 前，您不需要知道节点名称或目的地。org.jboss.ejb.client.EJBClient.getUserTransaction（） 方法为您提供一个远程 UserTransaction，它可根据第一次调用自动选择其目的地。从 Java 命名和目录接口查找远程 UserTransaction 的效果也相同。

注意

org.jboss.ejb.client.EJBClient.getUserTransaction（） 方法已弃用。

3. 当事务开始时，所有 Jakarta Enterprise Beans 调用都会在交易期间绑定到该特定节点，从而建立服务器关联性。

4. 当事务结束时，服务器关联性会被释放，Jakarta 企业 Bean 代理会返回到一般的集群关联性。

Jakarta Enterprise Beans Transactions Lazily 选择节点

您可以允许客户端在与事务相关的第一次调用期间轻松选择集群节点来处理事务。这允许在群集间对事务进行负载平衡。要使用这个选项，请按照以下步骤操作。

1. 不要在 InitialContext 中指定用于调用 Jakarta Enterprise Beans 的 PROVIDER_URL 属性。

2. 在客户端中，从 InitialContext 查找 the txn:RemoteUserTransaction。

```
UserTransaction ut = (UserTransaction)ctx.lookup("txn:RemoteUserTransaction");
```

3. 当事务开始时，将自动选择一个集群节点，建立服务器关联性，并在交易期间将所有 Jakarta Enterprise Beans 调用与该特定节点绑定。

4. 当事务结束时，服务器关联性会被释放，Jakarta 企业 Bean 代理会返回到一般的集群关联性。

8.12. JAKARTA ENTERPRISE BEANS-CLUSTERED 数据库计时器

JBoss EAP 支持群集数据库支持定时器，用于在群集环境中持久保留 Jakarta Enterprise Beans 计时器。因为集群是通过数据库提供的，如果计时器在较短的时间内停机的数量增加，则性能会降低。您可以使用 ejb3/service=timer -service/database -data-store 组件的 refresh-interval 和 allow- execution 属性来优化性能。

您还可以在非集群模式下使用数据库计时器，如下所示：

- 将 refresh-interval 设置为 0。
- 为每个节点提供唯一的分区名称，或者对每个节点使用不同的数据库。

Jakarta Enterprise Beans-clustered 数据库计时器如下：

- 允许执行计时器的每个节点都会为每个计时器调度一个超时时间。
- 当这个超时过期时，每个节点都会尝试通过将其状态更新为 running 来锁定计时器。

  更新其状态的查询类似以下查询：

```
UPDATE JBOSS_EJB_TIMER SET TIMER_STATE=? WHERE ID=? AND
TIMER_STATE<>? AND NEXT_DATE=?;
```

由于使用了事务处理和 READ_COMMITTED 或 SERIALIZABLE 隔离模式，只有一个节点成功更新该行，这是计时器执行的节点。

8.12.1. 设置 Jakarta Enterpise Beans-clustered 计时器

您可以通过添加数据库支持的计时器存储来设置 Jakarta Enterprise Beans-clustered 计时器。

先决条件

- 数据库必须支持 READ_COMMITTED 或 SERIALIZABLE 隔离模式。

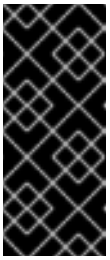
流程

- 创建数据库支持的计时器存储：

```
/subsystem=ejb3/service=timer-service/database-data-store=my-clustered-store:add(allow-execution=true, datasource-jndi-name="java:/MyDatasource", refresh-interval=60000, database="postgresql", partition="mypartition")
```

根据以下内容设置参数：

  - allow-execution：设置为 true，以允许此节点执行计时器。如果将它设置为 false，JBoss EAP 会将此节点上的计时器添加到数据库，供另一个节点执行。当您将定时器执行限制为集群中几个节点时，可以减少总体数据库负载。
  - datasource-jndi-name ：要使用的数据源。
  - refresh-interval ：在此节点检查数据库是否由其他节点添加的新计时器前，设置必须调整的时间间隔。该值以毫秒为单位。



重要

设置较小的值意味着 JBoss EAP 加快使用定时器，但会增加数据库的负载。如果添加计时器的节点因为失败或 allow-execution 为 false，则此计时器可能无法运行，直到节点刷新后为止。

  - Database ：定义正在使用的数据库的类型。些 SQL 语句由数据库自定义。

如果未定义此属性，服务器将尝试自动检测该类型。目前支持的类型有 postgresql、mysql 或a cle、db2、hsql 和 h2。

SQL 存在于以下文件中：module /system/layers/base/org/jboss/as/ejb3/main/timers/timer-sql.properties

您可以通过向此文件中添加新的数据库特定 SQL 语句来修改已执行的 SQL 或添加对新数据库的支持。

- 分区 ：将值设置为您希望此节点所属的分区的名称。只有同一分区的节点的计时器才对此节点可见。使用此属性将大型集群分成几个较小的集群，以提高性能。

注意

要将此数据库数据存储用于非集群计时器，请将 refresh-interval 设置为零，并确保每个节点都有唯一的分区名称，或者对每个节点使用不同的数据库。

8.12.2. 在部署中使用 Jakarta Enterprise Beans-clustered 计时器

您可以使用单个数据存储作为所有应用程序的默认数据存储，或者为每个应用程序使用特定的数据存储。

先决条件

- 您已设置了 Jakarta Enterprise Beans-clustered 数据库支持的定时器存储。

流程

- 要将单个数据存储用作所有应用程序的默认值，请按如下所示更新 ejb3 子系统中的 default-data- 存储 ：

```
<timer-service thread-pool-name="timer" default-data-store="clustered-store">
    <data-stores>
        <database-data-store name="clustered-store" datasource-jndi-
name="java:jboss/datasources/ExampleDS" partition="timer"/>
    </data-stores>
</timer-service>
```

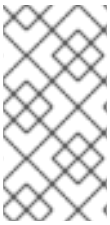
- 要将单独的数据存储用于特定应用程序，请在 jboss-ejb3.xml 文件中设置定时器数据存储名称：

```
<?xml version="1.1" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:timer="urn:timer-service:1.0"
xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
```

```
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd" version="3.1" impl-version="2.0">
   <assembly-descriptor>
      <timer:timer>
         <ejb-name>*</ejb-name>
         <timer:persistence-store-name>my-clustered-store</timer:persistence-store-name>
      </timer:timer>
   </assembly-descriptor>
</jboss:ejb-jar>
```

8.12.3. 使用 Jakarta Interceptors 刷新 Jakarta Enterprise Beans-clustered timers

您可以通过对业务方法配置 Jakarta Interceptors 以编程方式刷新计时器，从而强制计时器在 刷新过期前进行 刷新。



注意

在集群部署中，如果多个节点在较短的时间内更新其数据存储，内存中计时器状态可能会临时变得不同步。

先决条件

- 您已配置了数据库支持的集群 Jakarta Enterprise Beans。

流程

1. 实施 Jakarta Interceptors，使 wildfly.ejb.timer.refresh.enabled 设为 true。

```
import javax.interceptor.AroundInvoke;
import javax.interceptor.Interceptor;
import javax.interceptor.InvocationContext;

/**
 * An interceptor to enable programmatic timer refresh across multiple nodes.
 */
@Interceptor
public class RefreshInterceptor {
   @AroundInvoke
   public Object intercept(InvocationContext context) throws Exception {
      context.getContextData().put("wildfly.ejb.timer.refresh.enabled", Boolean.TRUE);
      return context.proceed();
   }
}
```

2. 配置 Jakarta Interceptors。

- 您可以将 Jakarta Interceptors 配置为目标无状态或单例 bean 商业方法。当 wildfly.ejb.timer.refresh.enabled 设为 true 时，调用 TimerService.getAllTimers（） 会在返回计时器前刷新计时器数据存储。

```
@Singleton
public class RefreshBean1 ... {

   @Interceptors(RefreshInterceptor.class)
   public void businessMethod1() {
      ...
      // since wildfly.ejb.timer.refresh.enabled is set to true in interceptor for this
business method,
      // calling timerService.getAllTimers() will first refresh from timer datastore
before returning timers.
      final Collection<Timer> allTimers = timerService.getAllTimers();
      ...
   }
}
```

- 或者，您也可以实施一种专用的业务方法，以便在需要时以编程方式刷新应用的其他部分调用的计时器。

```
   @Interceptors(RefreshInterceptor.class)
   public List<Timer> getAllTimerInfoWithRefresh() {
      return timerService.getAllTimers();
   }

   public void businessMethod1() {
      final LocalBusinessInterface businessObject =
sessionContext.getBusinessObject(LocalBusinessInterface.class);
      businessObject.getAllTimerInfoWithRefresh();

      // timer has been programmatically refreshed from datastore.
      // continue with other business logic...
   }
```

第 9 章 调整 JAKARTA ENTERPRISE BEANS 3 子系统

有关优化 ejb3 子系统性能的提示，请参阅 *性能调优指南* 中的 Jakarta Enterprise Ban s Subans 调优小节。

附录 A. 参考资料

A.1. JAKARTA ENTERPRISE BEANS JAVA 命名和目录接口参考

会话 Bean 的 Java 命名和目录接口查找名称使用以下语法：

```
ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?stateful
```

- <appName> ：如果会话 Bean 的 JAR 文件已部署在企业存档(EAR)中，则 appName 是相应 EAR 的名称。默认情况下，EAR 的名称是其文件名没有后 缀。可在其 application.xml 文件中覆盖应用名称。如果会话 bean 没有部署到 EAR 中，则将 appName 留 空。
- <moduleName> ： moduleName 是部署会话 bean 的 JAR 文件的名称。JAR 文件的默认名称是其文件名，不含 .jar 后缀。模块名称可以在 JAR 的 ejb-jar.xml 文件中覆盖。
- <distinctName> ：JBoss EAP 允许每个部署指定可选的不同名称。如果部署没有不同名称，则将 unique Name 留 空。
- <beanName> ： beanName 是要调用的会话 Bean 的简单类名称。
- <viewClassName> ：view ClassName 是远程接口的完全限定类名称。这包括接口的软件包名称。
- ?stateful ：当 Java 命名和目录接口名称指代有状态会话 Bean 时，需要？ stateful 后缀。它不包含用于其他 Bean 类型。

例如，如果部署了 hello.jar 带有 一个有状态的 Bean org.jboss.example.HelloBean，它公开了一个远程接口 org.jboss.example.Hello，则 Java 命名和目录接口查找名称将是：

```
ejb:/hello/HelloBean!org.jboss.example.Hello?stateful"
```

A.2. JAKARTA ENTERPRISE BEANS 参考解决方案

本节介绍 JBoss EAP 如何实施 @EJB 和 @Resource。请注意，XML 始终覆盖注释，但应用相同的规则。

@EJB 注释的规则

- @EJB 注释也具有 mapping Name（） 属性。该规范将此保留为特定于供应商的元数据，但 JBoss EAP 将 mapName（） 识别为您要引用的 Jakarta Enterprise Beans 的全局 Java 命名和目录接口名称。如果您指定了map Name（），则忽略所有其他属性，此全局 Java 命名和目录接口名称用于绑定。

- 如果您指定了 @EJB，但不定义任何属性：

```
@EJB
ProcessPayment myEjbref;
```

  然后应用以下规则：

  - 参考 Bean 的 Jakarta Enterprise Beans JAR 将搜索 Jakarta Enterprise Beans 及 @EJB 注入中使用的界面。如果有多个 Jakarta Enterprise Beans 发布相同的业务界面，则会引发异常。如果该接口只有一个 bean，则使用该接口。

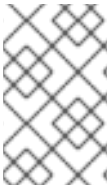
  - 在 EAR 中搜索发布该接口的 Jakarta Enterprise Beans。如果重复，则抛出异常。否则返回匹配的 Bean。

  - 在 JBoss EAP 运行时全局搜索该界面的 Jakarta 企业 Beans。如果发现重复，则会引发异常。

- @EJB.beanName（） 对应于 <ejb-link>。如果定义了 beanName（），则使用与 @EJB 相同的算法，不定义任何属性，除了使用 beanName（） 作为搜索中的键。此规则的一个例外是，如果您使用 ejb-link # 语法：它允许您在 EAR 中放入您引用的 Jakarta Enterprise Beans 的相对路径。更多详情请参阅 Jakarta Enterprise Beans 3.2 规范。

A.3. 远程 JAKARTA ENTERPRISE BEANS 客户端的项目依赖项

包括从远程客户端调用会话 Bean 的 Maven 项目需要来自 JBoss EAP Maven 存储库的下列依赖项：如下面的子章节中所述，有两种方法来声明 Jakarta Enterprise Beans 客户端依赖项。



注意

artifactId 版本可能会有变化。有关最新版本，请参阅 JBoss EAP Maven 存储库。

远程 Jakarta 企业 Bean 客户端的 Maven 依赖项

jboss-eap-jakartaee8 Bill of Materials(BOM)打包了 JBoss EAP 应用通常需要的许多构件的正确版本。BOM 依赖项在 pom.xml 的 <dependencyManagement> 部分中 指定，其范围为 导入。

示例：POM 文件 <dependencyManagement> 部分

```
<dependencyManagement>
  <dependencies>
    <dependency>
    <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-jakartaee8</artifactId>
      <version>7.4.0.GA</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

剩余的依赖项在 pom.xml 文件的 <dependencies> 部分中 指定。

示例：POM 文件 <dependencies> 部分

```
<dependencies>
   <!-- Include the Enterprise Java Bean client JARs -->
   <dependency>
      <groupId>org.jboss.eap</groupId>
      <artifactId>wildfly-ejb-client-bom</artifactId>
      <type>pom</type>
   </dependency>

   <!-- Include any additional dependencies required by the application
      ...
   -->

</dependencies>
```

JBoss EAP 随附的 ejb-remote 快速入门提供了远程 Jakarta 企业 Beans 客户端应用的完整工作示例。有关远程会话 Bean 调用的依赖配置的完整示例，请参见位于快速启动的根目录中的 client/pom.xml 文件。

用于 jboss-ejb-client 依赖项的单个 artifactID

您可以使用 wildfly-ejb-client-bom artifactID 并添加 jboss-ejb-client 库，以包含 Jakarta Enterprise Beans 客户端所需的所有依赖项：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.eap</groupId>
      <artifactId>wildfly-ejb-client-bom</artifactId>
      <version>JAKARTA_ENTERPRISE_BEANS_CLIENT_BOM_VERSION</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.jboss</groupId>
    <artifactId>jboss-ejb-client</artifactId>
  </dependency>
</dependencies>
```

您必须使用 JBoss EAP Maven 存储库中提供的 *JAKARTA_ENTERPRISE_BEANS_CLIENT_BOM_VERSION*。

A.4. JBOSS-EJB3.XML DEPLOYMENT DESCRIPTOR REFERENCE

jboss-ejb3.xml 是一种自定义部署描述符，可用于 Jakarta Enterprise Beans JAR 或 WAR 存档。在 Jakarta Enterprise Beans JAR 存档中，它必须位于 META-INF/ 目录中。在 WAR 存档中，它必须位于 WEB-INF/ 目录中。

格式与 ejb-jar.xml 类似，使用一些相同的命名空间并提供其他一些命名空间。jboss-ejb3.xml 的内容与 ejb-jar.xml 的内容合并，优先 为 jboss-ejb3.xml 项目。

本文档仅涵盖 jboss-ejb3.xml 使用的其他非标准命名空间。有关标准命名空间的文档，请参阅 http://xmlns.jcp.org/xml/ns/javaee/。

root 命名空间是 http://xmlns.jcp.org/xml/ns/javaee。

观察描述符命名空间

以下命名空间都可以在 <assembly-descriptor> 元素中使用。它们可用于将其配置应用到单个 Bean，或使用通配符(*)作为 ejb-name 应用到部署中的所有 bean。

安全命名空间(urn:security)

```
xmlns:s="urn:security"
```

这允许您为 Jakarta Enterprise Beans 设置 security -domain 和 run-as-principal。

```
<s:security>
  <ejb-name>*</ejb-name>
  <s:security-domain>myDomain</s:security-domain>
  <s:run-as-principal>myPrincipal</s:run-as-principal>
</s:security>
```

资源适配器命名空间： urn:resource-adapter-binding

```
xmlns:r="urn:resource-adapter-binding"
```

这可让您为 Message-Driven Bean 设置资源适配器。

```
<r:resource-adapter-binding>
  <ejb-name>*</ejb-name>
  <r:resource-adapter-name>myResourceAdapter</r:resource-adapter-name>
</r:resource-adapter-binding>
```

IIOP namespace: urn:iiop

```
xmlns:u="urn:iiop"
```

IIOP 命名空间是配置 IIOP 设置的位置。

池 namespace: urn:ejb-pool:1.0

```
xmlns:p="urn:ejb-pool:1.0"
```

这允许您选择由包含无状态会话 Bean 或 Message-Driven Beans 使用的池。池在服务器配置中定义。

```
<p:pool>
  <ejb-name>*</ejb-name>
  <p:bean-instance-pool-ref>my-pool</p:bean-instance-pool-ref>
</p:pool>
```

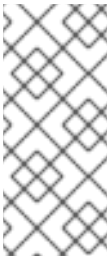
cache namespace: urn:ejb-cache:1.0

```
xmlns:c="urn:ejb-cache:1.0"
```

这可让您选择所含有状态会话 Bean 使用的缓存。缓存在服务器配置中定义。

```
<c:cache>
  <ejb-name>*</ejb-name>
  <c:cache-ref>my-cache</c:cache-ref>
</c:cache>
```

```
<?xml version="1.1" encoding="UTF-8"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd"
    version="3.1"
    impl-version="2.0">
  <enterprise-beans>
    <message-driven>
      <ejb-name>ReplyingMDB</ejb-name>
      <ejb-
class>org.jboss.as.test.integration.ejb.mdb.messagedestination.ReplyingMDB</ejb-class>
      <activation-config>
        <activation-config-property>
          <activation-config-property-name>destination</activation-config-property-name>
          <activation-config-property-value>java:jboss/mdbtest/messageDestinationQueue
          </activation-config-property-value>
        </activation-config-property>
      </activation-config>
    </message-driven>
  </enterprise-beans>
 </jboss:ejb-jar>
```



注意

jboss-ejb3-spec-2_0.xsd 文件存在已知问题，可能会导致架构验证错误。您可以忽略这些错误。如需更多信息，请参阅 https://bugzilla.redhat.com/show_bug.cgi?id=1192591。

A.5. 配置 JAKARTA 企业 BEANS 线程池

您可以使用管理控制台或管理 CLI 创建 Jakarta Enterprise Beans 线程池。

A.5.1. 使用管理控制台配置 Jakarta Enterprise Beans 线程池

流程

1. 登录管理控制台。
2. 导航到 **Configuration → Subsystems → EJB**，然后点 **View**。
3. 选择 **Container → Thread Pool**。
4. 单击 **Add**，再指定 **Name** 和 **Max Threads** 值。
5. 点 **Save**。

A.5.2. 使用管理 CLI 配置 Jakarta Enterprise Beans 线程池

流程

1. 使用 **add** 操作，语法如下：

```
/subsystem=ejb3/thread-pool=THREAD_POOL_NAME:add(max-threads=MAX_SIZE)
```

   a. 将 ***THREAD_POOL_NAME*** 替换为线程池所需的名称。

   b. 使用线程池的最大大小替换 ***MAX_SIZE***。

2. 使用 **read-resource** 操作来确认线程池已创建：

```
/subsystem=ejb3/thread-pool=THREAD_POOL_NAME:read-resource
```

a.

要将 ejb3 子系统中的所有服务重新配置为使用新的线程池，请使用以下命令：

```
/subsystem=ejb3/thread-pool=bigger:add(max-threads=100, core-threads=10)
/subsystem=ejb3/service=async:write-attribute(name=thread-pool-name, value="bigger")
/subsystem=ejb3/service=remote:write-attribute(name=thread-pool-name,
value="bigger")
/subsystem=ejb3/service=timer-service:write-attribute(name=thread-pool-name,
value="bigger")
reload
```

XML 配置示例：

```
<subsystem xmlns="urn:jboss:domain:ejb3:5.0">
    ...
    <async thread-pool-name="bigger"/>
    ...
    <timer-service thread-pool-name="bigger" default-data-store="default-file-store">
    ...
    <remote connectors="http-remoting-connector" thread-pool-name="bigger"/>
    ...
    <thread-pools>
        <thread-pool name="default">
            <max-threads count="10"/>
            <core-threads count="5"/>
            <keepalive-time time="100" unit="milliseconds"/>
        </thread-pool>
        <thread-pool name="bigger">
            <max-threads count="100"/>
            <core-threads count="5"/>
        </thread-pool>
    </thread-pools>
    ...
```

A.5.3. Jakarta Enterprise Beans 线程池属性

可以利用属性配置 Jakarta Enterprise Beans 线程池，以便更高效地运行满足特定配置需求。

- max-threads 属性决定了执行者支持的线程总数或最大数量。

```
/subsystem=ejb3/thread-pool=default:write-attribute(name=max-threads, value=9)
{"outcome" => "success"}
```

- **core-threads 属性决定了 executor 池中保留的线程数。这包括空闲的线程。如果未指定 core-threads 属性，它将默认为 max-threads 的值。**

```
/subsystem=ejb3/thread-pool=default:write-attribute(name=core-threads, value=3)
{"outcome" => "success"}
```

- **keepalive-time 属性决定非核心线程可以保持空闲的时间长度。这一次后，将删除非核心线程。**

```
/subsystem=ejb3/thread-pool=default:write-attribute(name=keepalive-time, value={time=5,
unit=MINUTES})
{"outcome"=> "success"}
```

- **要在不更改 keepalive-time 属性的时间单位的情况下更改时间，请使用以下命令：**

```
/subsystem=ejb3/thread-pool=default:write-attribute(name=keepalive-time.time, value=10)
{"outcome"=> "success"}
```

更新于 2024-02-09