Red Hat JBoss Enterprise Application Platform 8.0

迁移指南

升级到 Red Hat JBoss Enterprise Application Platform 主版本的说明

Last Updated: 2024-03-28

Red Hat JBoss Enterprise Application Platform 8.0 迁移指南

升级到 Red Hat JBoss Enterprise Application Platform 主版本的说明

法律通告

摘要

本指南提供有关如何从以前的 Red Hat JBoss Enterprise Application Platform 版本迁移的信息。

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. 在 **Summary** 中输入问题的简短描述。
3. 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。
4. 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

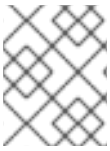
红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 迁移概述

作为系统管理员，您可以从 Red Hat JBoss Enterprise Application Platform 7 升级到 Red Hat JBoss Enterprise Application Platform 8.0。本指南涵盖了发行版本、弃用和不支持的功能以及任何必要的应用程序和服务器配置更新，以保持一致的行为。

它还提供有关有助于迁移的工具信息，如 Migration Toolkit for Runtimes，简化了 Java 应用程序的迁移，以及更新服务器配置的 JBoss 服务器迁移工具。

在成功部署并运行 JBoss EAP 8.0 后，您可以升级各个组件以使用 JBoss EAP 8.0 的新功能和功能。



注意

如果要从较旧版本的 JBoss EAP 迁移，您必须首先迁移到 JBoss EAP 7.4。如需更多信息，请参阅 JBoss EAP 7.4 迁移指南。

1.1. 了解迁移和升级

本节提供了升级和修补 JBoss EAP 的解释和指南，包括主要升级、次要更新和累积补丁。

1.1.1. JBoss EAP 中的主要升级

当应用从一个主版本移到另一个主要版本（如从 JBoss EAP 7 移到 JBoss EAP 8.0）时，需要进行重大升级或迁移。符合 Jakarta EE 8 规范、包含专有代码或已弃用 API 的应用可能需要修改其应用程序代码以便在 JBoss EAP 8.0 上运行。引入 Jakarta EE 10 涉及 Java 软件包名称的变化，以及需要调整 Jakarta EE 应用程序代码的其他方面，以便与 JBoss EAP 8.0 兼容。此外，JBoss EAP 8.0 中更改了服务器配置，需要迁移。本指南解决了此类迁移问题。

1.1.2. JBoss EAP 中的次要更新

JBoss EAP 中的小更新是提供程序错误修复、安全修复和新功能的点版本。每个点版本中所做的更改记录在 Red Hat JBoss Enterprise Application Platform 8.0 的发行注记 中。

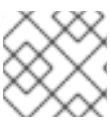
使用 JBoss 服务器迁移工具自动从一个点版本升级到另一个点版本，例如从 JBoss EAP 7.0 升级到 JBoss EAP 7.1。如需更多信息，请参阅使用 JBoss 服务器迁移工具。

1.1.3. JBoss EAP 中的累积补丁

JBoss EAP 定期提供包含漏洞和安全修复的累积补丁。累积修补程序将发行版递增到最后一个数字，例如从 7.1.0 升级到 71.1。

1.2. 使用 &LT ;*EAP_HOME*> 变量

在本文档中，&lt ;***EAP_HOME***& gt; 变量表示 JBoss EAP 安装的路径。将此变量替换为 JBoss EAP 安装的实际路径。



注意

<EAP_HOME> 不是环境变量。在脚本中使用 **JBOSS_HOME** 环境变量。

根据您选择安装 JBoss EAP 的安装选项，您可以找到安装目录或默认路径，如下所示：

- 如果您使用存档安装方法安装 JBoss EAP，安装目录是您提取存档的 **jboss-eap-8.0** 目录。
- 如果您使用 RPM 安装方法安装 JBoss EAP，安装目录是 **/opt/rh/eap8/root/usr/share/wildfly/**。
- 如果您使用安装程序应用程序安装 JBoss EAP，< ***EAP_HOME&*** gt; 的默认路径为 **${user.home}/EAP-8.0.0** ：
  - 对于 Red Hat Enterprise Linux 和 Oracle Solaris： **/home/*USER_NAME*/EAP-8.0.0/**
  - 对于 Microsoft Windows： **C:\Users*USER_NAME*\EAP-8.0.0**
- 如果您使用 Red Hat CodeReady Studio 安装程序应用程序安装和配置 JBoss EAP 服务器，则 < ***EAP_HOME>*** 的默认路径为 **${user.home}/devstudio/runtimes/jboss-eap**:
  - 对于 Red Hat Enterprise Linux：**/home/*USER_NAME*/devstudio/runtimes/jboss-eap/**
  - 对于 Microsoft Windows：**C:\Users*USER_NAME*\devstudio\runtimes\jboss-eap** 或 **C:\Documents and Settings*USER_NAME*\devstudio\runtimes\jboss-eap**

注意

如果您在 Red Hat CodeReady Studio 中将 **Target runtime** 设置为 8.0 或更新的运行时版本，则您的项目与 Jakarta EE 10 规范兼容。

第 2 章 准备迁移到 JBOSS EAP 8.0

作为系统管理员，您需要规划迁移到 JBoss EAP 8.0。此升级对于提高性能、增强安全性以及 Java 应用程序的稳定性至关重要。

JBoss EAP 8.0 为 JBoss EAP 7 应用程序提供向后兼容性。但是，如果您的应用程序使用 JBoss EAP 8.0 已弃用或删除的功能，您可能需要修改应用程序代码。

JBoss EAP 8.0 发行版本引入了一些可能会影响应用程序部署的更改。为确保迁移成功，请在尝试迁移应用程序前进行研究和规划。

在开始迁移过程前，请遵循以下初始步骤：

- 熟悉 Jakarta EE 10 的功能。
- 检查 JBoss EAP 8.0 的功能。
- 查看 JBoss EAP 入门资料。
- 通过 备份数据并查看服务器状态 来确保无缝迁移过程。
- 通过使用 RPM 安装迁移 JBoss EAP，从而简化您的安装过程。
- 通过将 JBoss EAP 作为一个服务迁移，提高管理性和自动化。

熟悉功能更改后，开发材料以及可帮助您迁移工作的工具后，评估您的应用程序和服务器配置，以确定在 JBoss EAP 8.0 上运行它们所需的更改。

2.1. 查看 JAKARTA EE 10 功能

Jakarta EE 10 引入了大量改进，简化了私有和公有云中功能丰富的应用程序的开发和部署。它包含新的功能和最新的标准，如 HTML5、WebSocket、JSON、Batch 和 Concurrency Utilities。更新包括 Jakarta Persistence 3.1、Jakarta RESTful Web Services 3.1、Jakarta Servlet 6.0、Jakarta Expression Language 5.0、Java 消息服务 3.1。Jakarta Server Faces 4.0、Jakarta Enterprise Beans 4.0、上下文和依赖注入 2.0 和 Jakarta Bean Validation 3.0。

其他资源

- Jakarta EE Platform 10

2.2. 查看 JBOSS EAP 8.0 的功能

JBoss EAP 8.0 包括与之前版本相比的升级和改进。有关 JBoss EAP 8.0 中引入的新功能的完整列表，*请参阅红帽客户门户网站上的 Red Hat JBoss Enterprise Application Platform 8.0 发行注记*中的 新功能和增强。

在将应用程序迁移到 JBoss EAP 8.0 之前，请注意，之前版本中的一些功能可能不再被支持，或者因为高维护成本、社区兴趣低或提供更好的替代方案而被弃用。有关 JBoss EAP 8.0 中 已弃用和不支持的功能的完整列表，请参阅红帽客户门户网站上的 *Red Hat JBoss Enterprise Application Platform 8.0 发行注记中的* 不受支持、弃用和删除的功能。

2.3. 查看 JBOSS EAP 入门资料

本节介绍 JBoss EAP 入门资料的关键组件，提供基本信息的简单概述，以帮助您从 JBoss EAP 开始。

有关以下重要信息，请参阅 JBoss EAP 入门指南 ：

- 下载并安装 JBoss EAP 8.0，以有效地设置您的环境。
- 下载并安装 JBoss 工具以提高您的开发环境。

重要

JBoss 工具是一个社区项目，不受红帽支持。有关建立和运行 JBoss 工具实例的帮助，请参考 社区网站。要下载 JBoss 工具，请参阅 JBoss 工具下载。

- 为您的开发环境配置 Maven 并管理项目依赖项。
- 下载并运行随产品附带的快速启动示例应用程序。

其他资源

- 使用 JBoss EAP 开发应用程序

2.4. 备份您的数据并查看服务器状态

本节强调需要备份数据、检查服务器状态并在迁移应用程序前处理潜在的问题。通过保护部署、管理开放事务和评估计时器数据，您可以确保平稳迁移。

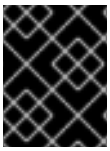
在开始迁移前，请考虑以下潜在问题：

- 迁移过程可能会删除临时文件夹。在迁移前，请确保备份 **data/content/** 目录中的任何部署。之后，在完成后恢复数据以避免缺少内容而导致服务器失败。
- 在迁移前，处理打开的事务并删除 **data/tx-object-store/** transaction 目录。
- 在继续迁移前，查看 **data/timer-service-data** 中的持久计时器数据，以确定其在升级后的适用性。在升级前，检查该目录中的 **deployment** bang 文件，以识别仍在使用哪些计时器。

确保在开始迁移前备份当前的服务器配置和应用程序。

2.5. 使用 RPM 安装迁移 JBOSS EAP

本指南中的迁移建议也适用于迁移 JBoss EAP 的 RPM 安装，但您可能需要更改一些步骤，如如何启动 JBoss EAP 以适合与存档或 **jboss-eap-installation-manager** 安装相比的 RPM 安装。



重要

不支持在单个 Red Hat Enterprise Linux 服务器上拥有多个 RPM 安装的 JBoss EAP 实例。因此，建议在迁移到 JBoss EAP 8.0 时将 JBoss EAP 安装迁移到新机器。

其他资源

- 使用 RPM 安装方法安装 JBoss EAP

2.6. 将 JBOSS EAP 迁移为服务

如果您将 JBoss EAP 7 作为服务运行，请在 Red Hat JBoss Enterprise Application Platform 安装方法中查看 JBoss EAP 8.0 的更新配置说明。

2.7. 迁移集群

如果您运行 JBoss EAP 集群，请按照 JBoss EAP 7.4 *补丁和升级指南* 中的升级集群一节中的说明进行操作。https://access.redhat.com/documentation/zh-cn/red_hat_jboss_enterprise_application_platform/7.4/html-single/patching_and_upgrading_guide/#proc_upgrading-a-cluster_default

第 3 章 使用有效工具简化 JBOSS EAP 8.0 迁移

作为系统管理员，您可以使用两个基本工具简化您的迁移过程到 JBoss EAP 8.0。Migration Toolkit for Runtimes (MTR)分析您的应用程序并提供详细的迁移报告，而 JBoss 服务器迁移工具会更新您的服务器配置，使其包含新功能和设置。

3.1. 迁移前分析应用程序

在将 JBoss EAP 6.4 和 7 应用程序迁移到 JBoss EAP 8.0 之前，您可以使用 Migration Toolkit for Runtimes (MTR)来分析 JBoss EAP 6.4 和 7 应用程序的代码和架构。用于迁移到 JBoss EAP 8.0 的 MTR 规则集提供有关在迁移到 JBoss EAP 8.0 时需要被替代配置替换的 XML 描述符、特定应用程序代码和参数。

MTR 是可扩展且可自定义的基于规则的工具集合，可帮助简化 Java 应用程序的迁移。MTR 分析您计划迁移的应用程序所使用的 API、技术和架构，为每个应用程序提供详细的迁移报告。这些报告提供以下信息：

- 所需迁移更改的详细说明
- 报告的更改是强制的还是可选的
- 报告的变化是复杂还是简单
- 需要更改迁移代码的链接
- 有关如何进行所需更改的信息提示和链接
- 估算发现的每个迁移问题的工作量程度以及迁移应用的总预计工作量

其他资源

- *Migration Toolkit for Runtimes*

3.2. 简化服务器配置迁移

JBoss 服务器迁移工具是更新您的服务器配置的首选方法，在 JBoss EAP 8.0 中包括新功能和设置，同时保持现有的配置。JBoss 服务器迁移工具读取您现有的 JBoss EAP 服务器配置文件，并添加任何新子系统的配置，使用新功能更新现有的子系统配置，并删除任何过时的子系统配置。

您可以使用 JBoss 服务器迁移工具迁移独立服务器并管理域。

3.2.1. 迁移到 JBoss EAP 8.0

JBoss 服务器迁移工具支持从所有版本的 JBoss EAP 7 迁移到 JBoss EAP 8.0。

注意

如果要从 JBoss EAP 6.4 迁移，您必须首先迁移到 JBoss EAP 7.4 的最新 Cumulative Patch (CP)版本。如需更多信息，请参阅 JBoss EAP 7.4 迁移指南。因此，您可以从 JBoss EAP 7.4 CP 版本迁移到 JBoss EAP 8.0。

先决条件

- JBoss EAP 未在运行。

流程

1. 从 JBoss EAP 下载工具，下载页面。
2. 提取下载的存档。

```
$ unzip <NAME_OF_THE_FILE>
```

3. 导航到 **MIGRATION_TOOL_HOME/bin** 目录。
4. 执行 **jboss-server-migration** 脚本。
   - Red Hat Enterprise Linux：

```
$ ./jboss-server-migration.sh --source EAP_PREVIOUS_HOME --target
EAP_NEW_HOME
```

   - 对于 Microsoft Windows：

```
jboss-server-migration.bat --source EAP_PREVIOUS_HOME --target EAP_NEW_HOME
```

注意

将 **EAP_PREVIOUS_HOME** 和 **EAP_NEW_HOME** 替换为 JBoss EAP 之前和新安装的实际路径。

其他资源

- 使用 JBoss 服务器迁移工具

第 4 章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 应用程序从 JAKARTA EE 8 迁移到 10

JBoss EAP 8.0 支持 Jakarta EE 10。与 JBoss EAP 7 支持的 Jakarta EE 8 规格相比，Jakarta EE 10 对 Jakarta EE 带来了很大的变化。在本章中，在准备将其应用从 JBoss EAP 7 迁移到 JBoss EAP 8.0 时，必须了解 Jakarta EE API 中的兼容性差异。

注意

本章的重点是，应用程序开发人员将其应用迁移到 JBoss EAP 8.0 的 Jakarta EE 8 和 Jakarta EE 10 之间的区别可能需要处理，而不是如何迁移。有关 JBoss EAP 7 到 JBoss EAP 8.0 应用程序迁移以及红帽提供的工具的更多信息，请参阅通过 有效工具和 了解应用程序迁移来简化您的 JBoss EAP 8.0 迁移 。

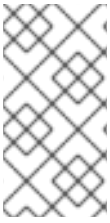
4.1. JAVAX 到 JAKARTA PACKAGE NAMESPACE 的变化

在 Jakarta EE 8 和 EE 10 之间最大的兼容性差异是把 EE API Java 软件包从 **javax** 重命名为 **jakarta prerequisites**。

随着 Java EE 迁移至 Eclipse Foundation 并建立 Jakarta EE，Eclipse 和 Oracle 同意 Jakarta EE 社区无法发展 **javax.** 软件包命名空间。因此，为了继续发展 EE API，从 Jakarta EE 9 开始，用于所有 EE API 的软件包已从 **javax** prerequisites 改为 **jakarta prerequisites**。这个更改不会影响作为 Java SE 一部分的 javax 软件包。

适应此命名空间更改是将应用程序从 JBoss EAP 7 迁移到 JBoss EAP 8 的最大变化。迁移到 Jakarta EE 10 的应用程序需要：

- 将 **javax** 软件包中的 EE API 类的任何导入语句或其他源代码使用到 **jakarta**
- 更新任何 EE 指定的系统属性的名称或其他以 **javax** 开头的配置属性，以 **jakarta 开始。**
- 将标识实现类的资源名称从 **META-INF/services/javax.[rest_of_name]** 改为 **META-INF/services/jakarta.[rest_of_name]**，用于任何由 EE 接口提供的实现，或使用 **java.util.ServiceLoader** 机制引导的抽象类。



注意

Red Hat Migration Toolkit 可帮助更新应用程序源代码中的命名空间。如需更多信息，请参阅 如何使用 Red Hat Migration Toolkit for Auto-Migration of a Application to the Jakarta EE 10 Namespace。如果源代码迁移不是一个选项，开源 Eclipse Transformer 项目会提供字节代码转换工具，用于将现有的 Java 存档从 javax 命名空间转换为 jakarta。

4.2. 其他更改

除了软件包命名空间更改外，为较早 EE 版本编写的应用程序可能需要适应 Jakarta EE 10 中包含的多个规格中所做的更改。以下小节描述了这些更改，主要是删除较长的 API 元素。

在以下部分中，对于已删除使用 **javax** 命名空间的 API 元素的任何实例，在 Jakarta EE 9 中使用的 **jakarta** 命名空间中进行了等同的删除。因此，如果您更新了应用程序，将 **javax** 命名空间替换为 **jakarta**，假设提到 **javax** 的项目适用于您的应用程序。

4.2.1. Jakarta 上下文和依赖注入 Bean 发现

根据 CDI 4.0 spec 更改备注，在带有空 **beans.xml** 文件的部署中发现上下文和依赖注入或 CDI Bean 的默认行为已从 **all** 改为 **annotated**。这意味着，CDI 只发现一个带有 bean 定义注解的 部署类。如果使用 Bean 的所有应用程序类都有这样的注解，则此 CDI 更改不会影响。否则，当 CDI 找不到提供特定 bean 的类型时，应用程序部署可能会失败。

如果您的应用程序受这个更改的影响，则有几个选项：

- 将 **beans.xml** 文件留空，但向需要它的所有类添加一个 bean 定义注解。
- 将类保持不变，但将 **beans.xml** 文件从空更改为包含以下内容：< **beans bean-discovery-mode="all"></beans>**
- 将应用保持不变，但更改服务器的 weld 子系统配置，以将空 **beans.xml** 文件的处理恢复回 JBoss EAP 7 行为。此设置会影响服务器上的所有部署。例如，使用 CLI: **/subsystem=weld:write-attribute (name=legacy-empty-beans-xml-treatment,value=true)**

4.2.2. CDI API 更改

Jakarta 上下文和依赖注入 4.0 删除了以下已弃用的 API 元素：

- **javax.enterprise.inject.spi.Bean.isNullable （）** 方法已被删除。现在，这个方法在很多年内一直返回 **false**，因此调用它的应用程序可以将调用替换为 **false** 或移除任何分支逻辑，并只保留 **false** 分支的内容。
- **javax.enterprise.inject.spi.BeanManager.createInjectionTarget (AnnotatedType)** 方法已被删除。将此方法调用替换为 **BeanManager.getInjectionTargetFactory (AnnotatedType)**，并使用返回的工厂来创建注入目标。如需更多信息，请参阅 Jakarta 上下文和依赖注入规格中的类包含注入目标。
- **javax.enterprise.inject.spi.BeanManager.fireEvent (Object,Annotation)** 方法已被删除。使用 **BeanManager.getEvent （）** 作为指向类似 API 的入口点。如需更多信息， 请参阅 Jakarta 上下文和依赖注入规格中的查找事件。
- **javax.enterprise.inject.spi.BeforeBeanDiscovery.addAnnotatedType (AnnotatedType)** 方法已被删除。如果您的应用程序调用此方法，您可以将它替换为 **BeforeBeanDiscovery.addAnnotatedType (AnnotatedType,(String) null)**。

4.2.3. Jakarta Enterprise Beans

Java SE 14 删除了 **java.security.Identity** 类，因此其使用量已从 Jakarta Enterprise Beans 4.0 API 中删除。

- 弃用的 **javax.ejb.ejbContext.getCallerIdentity** （） 方法已被删除。您可以使用 **EJBContext.getCallerPrincipal （）**，这将返回 **java.security.Principal**。
- 弃用的 **javax.ejb.ejbContext.isCallerInRole (Identity role)** 方法已被删除。您可以使用 **EJBContext.isCallerInRole (String roleName)** 替代。
- Jakarta XML RPC 规范已从 Jakarta EE 10 Full Platform 中删除，因此删除了 returned **javax.xml.rpc.handler.MessageContext.getMessageContext 的 javax.ejb.Session** Context.getMessageContext 方法。
- Jakarta XML RPC 规范在 Jakarta EE 8 中是可选的，Red Hat JBoss EAP 7 不支持它。此规格的任何用法都会抛出 **IllegalStateException**，因此此 EJB API 更改不会影响 JBoss EAP 7 上运行的任何现有应用。

- 弃用的 **javax.ejb.ejbContext.getEnvironment （）** 方法已被删除。使用 JNDI 命名上下文 **java:comp/env** 来访问企业 bean 的环境。

4.2.4. Jakarta Expression Language

删除了错误拼写 **javax.el.MethodExpression.isParmetersProvided （）** 方法。您可以使用 **MethodExpression.isParametersProvided （）** 替代。

4.2.5. Jakarta JSON 绑定

默认情况下，使用 **jakarta.json.bind.annotation.JsonbCreator** 注解标注的类型不需要 JSON 内容中的所有参数都可用。如果被解析的 JSON 缺少其中一个参数，则将使用默认值。可以通过调用 **jakarta.json.bind.JsonbConfig （）.withCreatorParametersRequired (true)** 来开启需要 JSON 中的所有参数的 EE 8 行为。

4.2.6. jakarta Faces

Jakarta Faces 4.0 中删除了以下已弃用的功能。

4.2.6.1. jakarta Faces 和 Java Server 页

Jakarta Server Pages (JSP)支持在 Jakarta Faces 2.0 及更新的版本中已弃用。Jakarta Faces 4.0 中删除了对 JSP 的支持。Facelets 取代了 JSP 作为首选视图定义语言(VDL)。使用 JSP 用于 Faces 视图的应用程序可以使用 Facelet 修改。您可以通过将 **FacesServlet** 映射到 **web.xml** 中的 **lfjsp** 后缀来识别应用程序。

4.2.6.2. Faces Managed-Beans

在 Faces 4.0 中删除了特定于 Jakarta Faces 的 managed-bean 概念，用于 Jakarta 上下文和依赖注入 (CDI) Bean。使用 Faces managed-beans （即，使用 **javax.faces.bean.ManagedBean** 注解标注的类）或在 **faces-config.xml**的 managed-bean 元素中引用的类可能需要进行以下更改：

- 使用 **javax.faces.bean.ManagedBean** 标注的类在 **faces-config.xml** 中的 managed-bean 元素中引用的类应该被注解为 **jakarta.inject.Named**，以及 **faces-config.xml** 中的 managed-bean 元素应该被删除。

- 使用 **javax.faces.bean.ManagedProperty** 注解标注的成员应该使用 **jakarta.faces.annotation.ManagedProperty**，以及 **jakarta.inject.Inject** 注解。要获得类似于旧的 **javax.faces.bean.ManagedBean (name="foo", eager=true)** 的启动语义，请添加 **public void xxx (@Observes jakarta.enterprise.event.Startup event)** 方法或 **公共 void xxx （@Observes @Initialized (ApplicationScoped.class)对象上下文** 方法。**jakarta.enterprise.event.Startup** 选项是 CDI 4.0 中的新选项。

- 使用 **javax.faces.bean.ApplicationScoped** 注解应该被 **jakarta.enterprise.context.ApplicationScoped** 替换。

- 使用 **javax.faces.bean.CustomScoped** 注解应该替换为 CDI 自定义范围和 **jakarta.enterprise.context.spi.Context**。如需了解更多详细信息，请参阅 CDI 4.0 规格中定义新的范围类型 和 上下文接口。

- 使用 **javax.faces.bean.NoneScoped** 注解应该替换为 **jakarta.enterprise.context.Dependent**，它是 CDI 内置范围，具有大约类似语义。

- 使用 **javax.faces.bean.RequestScoped** 注解应该被 **jakarta.enterprise.context.RequestScoped** 替换。

- 使用 **javax.faces.bean.SessionScoped** 注解应该被 **jakarta.enterprise.context.SessionScoped** 替换。

4.2.6.3. 其他 Faces API 更改

javax.faces.bean.ViewScoped 注解已被删除。您可以使用 **jakarta.faces.view.ViewScoped** 替代。

javax.faces.view.facelets.ResourceResolver 和 **javax.faces.view.facelets.FaceletsResourceResolver** 注解已被删除。对于应用程序中的任何 ResourceResolver，实施 **jakarta.faces.application.ResourceHandler** 接口，并在 **faces-config.xml** 中的 **application/resource-handler** 元素中注册实现的完全限定类名称。

4.2.7. Jakarta Servlet

Jakarta Servlet 6.0 删除了在 Servlet 5.0 及更早版本中被弃用的数字 API 类和方法，主要在 Servlet 2.x 版本中。

javax.servlet.SingleThreadModel marker 接口已被删除，且实施此接口的 servlets 必须删除 interface 声明，并确保 servlet 代码正确保护状态，以及针对并发访问的其他资源访问。例如，通过避免使用实例变量或同步代码访问资源的块。但是，建议开发人员不同步 **服务** 方法（或 **doGet** 和 **doPost** 等方法），因为此类同步性能会降低此类同步的影响。

javax.servlet.http.HttpSessionContext 接口已被删除，以及 **javax.servlet.http.HttpSession.getSessionContext （）** 方法。由于 Servlet 2.1 需要它的实现，因此此界面没有用例，因为规格不需要提供任何可用的数据。

javax.servlet.http.HttpUtils 实用程序类已被删除。应用程序应使用 **ServletRequest** 和 **HttpServletRequest** 接口，而不是以下方法：

- **parseQueryString (String s)** and **parsePostData (int len, ServletInputStream in)** - Use **ServletRequest.getParameterMap （）**.如果应用程序需要区分查询字符串参数和请求正文参数，应用程序必须通过解析查询字符串本身来实现该代码。
- **getRequestURL (HttpServletRequest req)**- Use **HttpServletRequest.getRequestURL （）**.

另外，以下各种方法和构造器已被删除：

class/Interface	删除	使用 Instead
javax.servlet.ServletContext	getServlet (String name)	没有替换方案
	getServlets ()	没有替换方案
	getServletNames()	没有替换方案
	log (Exception exception, String msg)	log (String message, Throwableable)

class/Interface	删除	使用 Instead
javax.servlet.ServletRequest	getRealPath (String path)	ServletContext.getRealPath(String path)
javax.servlet.ServletRequestWrapper	getRealPath (String path)	ServletContext.getRealPath(String path)
javax.servlet.UnavailableException	getServlet()	没有替换方案
	UnavailableException (Servlet servlet, String msg)	UnavailableException (String)
	UnavailableException (int seconds, Servlet servlet, String msg)	UnavailableException (String, int)
javax.servlet.http.HttpServletRequest	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()
javax.servlet.http.HttpServletRequestWrapper	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()
javax.servlet.http.HttpServletResponse	encodeUrl (String url)	encodeURL (String url)
	encodeRedirectUrl(String url)	encodeRedirectURL (String url)
	setStatus (int sc, String sm)	sendError (int, String)
javax.servlet.http.HttpServletResponseWrapper	encodeUrl (String url)	encodeURL (String url)

class/Interface	删除	使用 Instead
	encodeRedirectUrl(String url)	encodeRedirectURL (String url)
	setStatus (int sc, String sm)	sendError (int, String)
javax.servlet.http.HttpSession	getValue (String name)	getAttribute (String name)
	getValueNames()	getAttributeNames()
	putValue (String name, Object value)	setAttribute (String name, Object value)
	removeValue (String name)	removeAttribute (String name)

4.2.8. jakarta Soap with Attachments

删除了对通过 **jaxm.properties** 文件进行供应商查找的支持。

弃用的 **javax.xml.soap.SOAPElementFactory** 类已被删除。使用 **jakarta.xml.soap.SOAPFactory** 创建 SOAPElements。

SOAPElementFactory 方法	SOAPFactory 等效
newInstance()	newInstance()
create (Name)	createElement(Name)
create (String)	createElement (String)
create (String, String, String)	createElement (String, String, String)

4.2.9. Jakarta XML 绑定

xml 绑定文件中使用的 XML 命名空间已更改。**http://java.sun.com/xml/ns/jaxb** 命名空间应该替换为 **https://jakarta.ee/xml/ns/jaxb**。

弃用的 **javax.xml.bind.Validator** 接口已被删除，因为 **关联的 javax.xml.bind.JAXBContext.createValidator （）** 方法。要验证 marshalling 和 unmarshalling 操作，请为 **jakarta.xml.validation.Schema 提供 javax.xml.validation.Schema.bind.Marshaller.setSchema (Schema)**。

删除了对与 JAXB 1.0 兼容的支持。

JAXBContext 实施查找算法中的一些已弃用步骤已被删除。通过 **jaxb.properties** 文件、**javax.xml.bind.context.factory** 或 **jakarta.xml.bind.JAXBContext** 属性和 **/META-INF/services/javax.xml.bind.JAXBContext** 资源文件搜索实施类名称已被丢弃。有关当前实现发现算法的更多信息，请参阅 Jakarta XML Binding 4.0 规范。

javax.xml.bind.Marshaller 接口中的多个方法的通用要求已更改，如下所示：

Jakarta XML Binding 2.3 / 3.0	Jakarta XML Binding 4.0
<a extends XmlAdapter> void setAdapter (A adapter)	<a extends XmlAdapter<?, ?>> void setAdapter (A adapter)
<a extends XmlAdapter> void setAdapter (Class<A> type, A adapter)	<a extends XmlAdapter<?, ?>> void setAdapter (Class<A> type, A adapter)
<a extends XmlAdapter> A getAdapter (Class<A> type)	<a extends XmlAdapter<?, ?>> A getAdapter (Class<A> type)

除了 Jakarta XML Binding API 的更改外，实施库 EAP 8 中还有显著的软件包名称更改，这可能会影响访问实施库的一些应用：

- **com.sun.xml.bind** 软件包中的任何使用都应替换为 **org.glassfish.jaxb.runtime** 软件包中的类。**com.sun.xml.bind** 的子软件包中的类应替换为对应的 **org.glassfish.jaxb.runtime** 子软件包中的类。
- 对于 **jakarta.xml.bind.Marshaller** 属性设置，将属性常量名称从 **com.sun.xml.bindAPPEND** 改为 **org.glassfish.jaxbBYO**。例如，**marshaller.setProperty ("com.sun.xml.bind.namespacePrefixMapper", mapper)** 变为 **marshaller.setProperty ("org.glassfish.jaxb.namespacePrefixMapper", mapper)**。

第 5 章 将 JBOSS EAP 应用的 MAVEN 项目迁移到 JBOSS EAP 8.0

将应用的 Maven 项目迁移到 JBoss EAP 8.0 时，它使用 JBoss EAP BOM 管理依赖项，您必须更新 pom.xml 文件，因为 JBoss EAP 8.0 BOMs 引入了以下显著变化。

注意

如果应用程序在不进行任何更改的情况下迁移到 JBoss EAP 8.0，应用将使用不正确的依赖项构建，且可能无法在 JBoss EAP 8.0 上部署。

5.1. 重命名 JBOSS EAP JAKARTA EE 8

JBoss EAP **Jakarta EE 8** BOM 已重命名为 **JBoss EAP EE**，其 Maven Coordinates 从 **org.jboss.bom:jboss-eap-jakartaee8** 改为 **org.jboss.bom:jboss-eap-ee**。在 Maven 项目(**pom.xml**)中使用此 BOM 可使用以下依赖项管理导入来识别：

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

Maven 项目(**pom.xml**)应将新的"JBoss EAP EE" BOM 导入到其依赖项管理中：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.2. 使用工具重命名 JBOSS EAP JAKARTA EE 8

JBoss EAP **Jakarta EE 8 With Tools** BOM 已重命名为 **JBoss EAP EE**，其 Maven Coordinates 已从 **org.jboss.bom:jboss-eap-jakartaee8-with-tools** 改为 **org.jboss.bom:jboss-eap-ee-with-tools**。在 Maven 项目(**pom.xml**)中使用此 BOM 可使用以下依赖项管理导入来识别：

```
<dependencyManagement>
  <dependencies>
    <dependency>
```

```
            <groupId>org.jboss.bom</groupId>
            <artifactId>jboss-eap-jakartaee8-with-tools</artifactId>
            <version>${version.bom}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Maven 项目(**pom.xml**)必须把新的 "JBoss EAP EE with Tools" BOM 导入到其依赖项管理中：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.jboss.bom</groupId>
            <artifactId>jboss-eap-ee-with-tools</artifactId>
            <version>${version.bom}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

5.3. 删除 JBOSS EAP JAKARTA EE 8 API

JBoss EAP **Jakarta EE 8 API** BOM 已在 JBoss EAP 8.0 中移除，且必须使用新的 **JBoss EAP EE** BOM。在 Maven 项目(**pom.xml**)中使用此 BOM 可使用以下依赖项管理导入来识别：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.jboss.spec</groupId>
            <artifactId>jboss-jakartaee-8.0</artifactId>
            <version>${version.bom}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>org.jboss.spec</groupId>
            <artifactId>jboss-jakartaee-web-8.0</artifactId>
            <version>${version.bom}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Maven 项目(**pom.xml**)必须导入新的 **JBoss EAP EE** BOM 到其依赖项管理：

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.jboss.bom</groupId>
            <artifactId>jboss-eap-ee</artifactId>
```

```
        <version>${version.bom}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
</dependencyManagement>
```

5.4. 删除 JBOSS EAP RUNTIME BOM

JBoss EAP **Runtime** BOM 不再与 JBoss EAP 8.0 一起分发，您必须使用新的 **JBoss EAP EE** BOM。在 Maven 项目(**pom.xml**)中使用此 BOM 可使用以下依赖项管理导入来识别：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>eap-runtime-artifacts</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven 项目(**pom.xml**)必须把新的 **JBoss EAP EE** BOM 导入到其依赖项管理中：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.5. JAKARTA EE 和 JBOSS API MAVEN COORDINATES 的变化

以下 Jakarta EE 和 JBoss API 工件（由 JBoss EAP BOM 提供）已更改 Maven Coordinate，或者被其他工件替代：

- **com.sun.activation:jakarta.activation**
- **org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec**
- **org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec**
- **org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec**
- **org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec**
- **org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec**

- **org.jboss.spec.javax.el:jboss-el-api_3.0_spec**
- **org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec**
- **org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec**
- **org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec**
- **org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec**
- **com.sun.mail:jakarta.mail**
- **org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec**
- **org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec**
- **org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec**
- **org.apache.taglibs:taglibs-standard-spec**
- **org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec**
- **org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec**
- **org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec**
- **javax.jws:jsr181-api**
- **org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec**
- **org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec**
- **org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec**
- **org.hibernate:hibernate-core**
- **org.hibernate:hibernate-jpamodelgen**
- **org.jboss.narayana.xts:jbossxts**

Maven Project (**pom.xml**)应该更新依赖项的 Maven Coordinate （如果工件已更改或替换），或删除依赖项（如果工件不再受支持）。

```
<dependencies>
  <!-- replaces com.sun.activation:jakarta.activation -->
  <dependency>
     <groupId>jakarta.activation</groupId>
     <artifactId>jakarta.activation-api</artifactId>
  </dependency>
  <!-- replaces org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec -->
  <dependency>
     <groupId>jakarta.annotation</groupId>
     <artifactId>jakarta.annotation-api</artifactId>
  </dependency>
  <!-- replaces org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec -->
  <dependency>
     <groupId>jakarta.authentication</groupId>
```

```
    <artifactId>jakarta.authentication-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec -->
<dependency>
    <groupId>jakarta.authorization</groupId>
    <artifactId>jakarta.authorization-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec -->
<dependency>
    <groupId>jakarta.batch</groupId>
    <artifactId>jakarta.batch-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec -->
<dependency>
    <groupId>jakarta.ejb</groupId>
    <artifactId>jakarta.ejb-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.el:jboss-el-api_3.0_spec -->
<dependency>
    <groupId>org.jboss.spec.jakarta.el</groupId>
    <artifactId>jboss-el-api_5.0_spec</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec -->
<dependency>
    <groupId>jakarta.enterprise.concurrent</groupId>
    <artifactId>jakarta.enterprise.concurrent-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec -->
<dependency>
    <groupId>jakarta.faces</groupId>
    <artifactId>jakarta.faces-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec -->
<dependency>
    <groupId>jakarta.interceptor</groupId>
    <artifactId>jakarta.interceptor-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec  -->
<dependency>
    <groupId>jakarta.jms</groupId>
    <artifactId>jakarta.jms-api</artifactId>
</dependency>
<!-- replaces com.sun.mail:jakarta.mail -->
<dependency>
    <groupId>jakarta.mail</groupId>
    <artifactId>jakarta.mail-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec -->
<dependency>
    <groupId>jakarta.resource</groupId>
    <artifactId>jakarta.resource-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec -->
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
```

```
</dependency>
<!-- replaces org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec -->
<dependency>
    <groupId>jakarta.servlet.jsp</groupId>
    <artifactId>jakarta.servlet.jsp-api</artifactId>
</dependency>
<!-- replaces org.apache.taglibs:taglibs-standard-spec -->
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec  -->
<dependency>
    <groupId>jakarta.transaction</groupId>
    <artifactId>jakarta.transaction-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec  -->
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec and javax.jws:jsr181-api -->
<dependency>
    <groupId>org.jboss.spec.jakarta.xml.ws</groupId>
    <artifactId>jboss-jakarta-xml-ws-api_4.0_spec</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec -->
<dependency>
    <groupId>jakarta.websocket</groupId>
    <artifactId>jakarta.websocket-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec -->
<dependency>
    <groupId>jakarta.ws.rs</groupId>
    <artifactId>jakarta.ws.rs-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec -->
<dependency>
    <groupId>org.jboss.spec.jakarta.xml.soap</groupId>
    <artifactId>jboss-saaj-api_3.0_spec</artifactId>
</dependency>
<!-- replaces org.hibernate:hibernate-core -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
</dependency>
<!-- replaces org.hibernate:hibernate-jpamodelgen -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-jpamodelgen</artifactId>
</dependency>
<!-- replaces org.jboss.narayana.xts:jbossxts -->
<dependency>
    <groupId>org.jboss.narayana.xts</groupId>
    <artifactId>jbossxts-jakarta</artifactId>
```

```
      <classifier>api</classifier>
    </dependency>
  </dependencies>
```

5.6. 删除 JBOSS EJB 客户端旧的 BOM

带有 Maven **groupId:artifactId** 的 JBoss EJB Client Legacy BOM 协调 **org.jboss.eap:wildfly-ejb-client-legacy-bom** 不再由 JBoss EAP 提供。

您可以识别将此 BOM 用作依赖项管理导入，或作为 Maven 项目 **pom.xml** 配置文件中的依赖项引用，如下例所示：

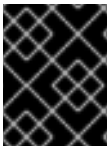
依赖项管理导入示例

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

依赖项参考示例

```
<dependencies>
...
  <dependency>
    <groupId>org.jboss.bom</groupId>
    <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
    <version>${version.bom}</version>
    <type>pom</type>
  </dependency>
...
</dependencies>
```

独立客户端 Java 应用程序可以继续使用已删除 BOM 的同一版本，如 **7.4.0.GA**，以用于 JBoss EAP 8。但是，建议您替换 EJB Client Legacy API。有关配置 EJB 客户端的详情，请参考 如何在 EAP 7.1+ / 8.0+ 中配置 EJB 客户端。有关在 JBoss EAP 7.4 中弃用 EJB Client Legacy API 的详情，请参考 Red Hat JBoss Enterprise Application Platform (EAP) 7 中弃用。



重要

JBoss EAP 7.4 客户端和 JBoss EAP 8.x 服务器遵循不同的产品生命周期。如需更多信息，请参阅 JBoss EAP 的产品生命周期。

第 6 章 服务器迁移更改

在迁移前，请确保您了解在服务器上部署应用程序所需的迁移更改，并在 Red Hat JBoss Enterprise Application Platform 8.0 中升级它们。

6.1. WEB 服务器配置更改

了解 Red Hat JBoss Enterprise Application Platform 中 **mod_cluster** 和 Undertow 的更改，它们会影响根上下文行为并增强服务器信息的安全性。

6.1.1. 默认 Web 模块行为更改

在 JBoss EAP 7.0 中，**mod_cluster** 中默认禁用 Web 应用的根上下文。

自 JBoss EAP 7.1 起，情况已不再如此。如果您希望禁用根上下文，这可能会造成意外的后果。例如，可以将请求错误路由到不合要求的节点，或者不应公开的私有应用可以通过公共代理意外访问。Undertow 位置现在会自动注册到 **mod_cluster** 负载平衡器，除非它们被明确排除。

使用以下管理 CLI 命令，将 ROOT 从 **modcluster** 子系统配置中排除：

```
/subsystem=modcluster/mod-cluster-config=configuration:write-attribute(name=excluded-contexts,value=ROOT)
```

使用以下管理 CLI 命令，禁用默认的欢迎 Web 应用：

```
/subsystem=undertow/server=default-server/host=default-host/location=\/:remove
/subsystem=undertow/configuration=handler/file=welcome-content:remove
reload
```

其他资源

- 配置默认欢迎 Web 应用

6.1.2. Undertow 子系统默认配置更改

在 Red Hat JBoss Enterprise Application Platform 7.2 之前，默认的 **undertow** 子系统配置包括两个响应标头过滤器，这些标头被 **default-host** 附加到每个 HTTP 响应中：

- **server** 之前已设置为 **JBoss EAP/7**。
- **X-Powered-By** 之前设置为 **Undertow/1**。

这些响应标头过滤器已从默认的 JBoss EAP 7.2 配置中删除，以防止意外泄漏有关正在使用的服务器的信息。

以下是 JBoss EAP 7.1 中默认 **undertow** 子系统配置的示例。

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
```

```
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <response-header name="server-header" header-name="Server" header-value="JBoss-EAP/7"/>
        <response-header name="x-powered-by-header" header-name="X-Powered-By" header-
value="Undertow/1"/>
    </filters>
</subsystem>
```

以下是 JBoss EAP 7.4 中默认 **undertow** 子系统配置的示例：

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0" default-server="default-server" default-virtual-
host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

以下是 JBoss EAP 8.0 中默认 **undertow** 子系统配置的示例。

```
<subsystem xmlns="urn:jboss:domain:undertow:14.0" default-virtual-host="default-host" default-
servlet-container="default" default-server="default-server" statistics-
enabled="${wildfly.undertow.statistics-enabled:${wildfly.statistics-enabled:false}}" default-security-
domain="other">
    <byte-buffer-pool name="default"/>
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" ssl-context="applicationSSC" enable-
```

```
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker http-authentication-factory="application-http-authentication"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <application-security-domains>
        <application-security-domain name="other" security-domain="ApplicationDomain"/>
    </application-security-domains>
</subsystem>
```

6.2. INFINISPAN 服务器配置更改

在考虑以下方面时，配置自定义有状态会话 Bean (SFSB)缓存以便在 Red Hat JBoss Enterprise Application Platform 7.1 及更高版本中进行传递：

- 弃用 idle-timeout 属性
- 实现 lazy 传递
- 集群名称的决定
- 适当的驱除和过期配置
- 缓存容器传输协议中的修改以提高性能。

通过遵循这些注意事项，您可以优化 SFSB 缓存配置，以提高 JBoss EAP 7.1 及更高版本中的传递。

6.2.1. 配置自定义有状态会话 Bean 缓存以进行传递

在 JBoss EAP 7.1 及更高版本中，启用了 passivation 的自定义有状态会话 Bean (SFSB)缓存已更改。当使用 passivation 配置 SFSB 缓存时，请考虑以下密钥更改：

- 弃用 idle-timeout 属性
- 从 eager 变为 lazy passivation
- 确定集群名称
- 在 Jakarta Enterprise Beans 缓存中配置驱除和过期

在配置自定义 SFSB 缓存以便在 JBoss EAP 7.1 及更高版本中传递时，请考虑以下限制：

- **idle-timeout** 属性在 **ejb3** 子系统的 **infinispan passivation-store** 中配置，在 JBoss EAP 7.1 及更高版本中被弃用。JBoss EAP 7.1 及更高版本只支持 lazy passivation，这在达到 **max-size** 阈值时发生。

注意

这些版本不再支持通过 idle-timeout 的 eager passivation。

- 在 JBoss EAP 7.1 及更高版本中，Jakarta Enterprise Beans 客户端使用的集群名称由通道的实际集群名称决定，如 **jgroups** 子系统中所配置。
- JBoss EAP 7.1 及更高版本仍允许您设置 **max-size** 属性来控制传递阈值。

6.2.2. Infinispan 缓存容器传输更改

JBoss EAP 7.0 及更新版本之间的行为更改需要在批处理模式下执行对缓存容器传输协议的更新，或使用特殊标头。此更改还影响用于管理 JBoss EAP 服务器的工具。

以下是用于在 JBoss EAP 7.0 中配置缓存容器传输协议的管理 CLI 命令示例。

```
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
```

以下是在 JBoss EAP 7.1 中执行相同配置所需的管理 CLI 命令示例。请注意，命令是以批处理模式执行的。

```
batch
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
run-batch
```

如果您不希望使用批处理模式，您可以在定义传输时指定操作标头 **allow-resource-service-restart=true**。

如果您使用脚本更新缓存容器传输协议，请务必检查它们并添加批处理模式。

6.2.3. 从版本 8.0 及之后的版本中的 EJB 子系统配置更改

JBoss EAP 8.0 引入了对可分布式有状态会话 Bean (SFSB)的 Enterprise 3.0.0 (EJB)子系统配置的更改，包括对多个资源的新子系统和更新。JBoss EAP 6 和 7 中使用的几个资源也被弃用。这些更改启用了服务器配置迁移，以确保您的应用程序与未来的主要版本兼容。

JBoss EAP 8.0 将 JBoss EAP 6 和 7 中使用的资源替换为两个新资源，以及一个 **distributable-ejb** 子系统来配置 SFSB 缓存。下表概述了已弃用的资源和替换它们的新资源。

表 6.1. SFSB 缓存配置更改

弃用的资源	新的不可分布式 SFSB 缓存	新的可分布式 SFSB 缓存
/subsystem=ejb3/cache	**/subsystem=ejb3/simple-cache**	**/subsystem=ejb3/distributable-cache**
/subsystem=ejb3/passivation-store	不适用	**/subsystem=ejb3/distributable-cache=”name”/bean-management"=..**

非不同 SFSB 缓存 **/subsystem=ejb3/simple-cache** 相当于 SFSB 缓存 **/subsystem=ejb3/cache**，在 JBoss EAP 7 中使用，没有定义 passivation 存储。

distributable SFSB 缓存 **/subsystem=ejb3/distributable-cache** 包含一个可选的 **bean-management** 属性，引用 **distributable-ejb** 子系统的对应资源。如果您没有定义资源，则默认为 **distributable-ejb** 子系统中的 **bean-management** 资源。

考虑将服务器配置迁移到 JBoss EAP 8.0 中更新的方法。虽然当前发行版本继续使用已弃用的资源，但当它们被删除时，可能不会出现在将来的版本中。

JBoss EAP 7 和首选 JBoss EAP 8.0 配置之间的比较示例如下：

JBoss EAP 7 配置：

```
/subsystem=ejb3/cache=example-simple-cache:add()
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, bean-cache=default, max-size=1024)
/subsystem=ejb3/cache=example-distributed-cache:add(passivation-store=infinispan)
```

首选 JBoss EAP 8.0 配置：

```
/subsystem=ejb3/simple-cache=example-simple-cache:add()
/subsystem=distributable-ejb=example-distributed-cache/infinispan-bean-management=example-bean-cache:add(cache-container=ejb, cache=default, max-active-beans=1024)
/subsystem=ejb3/distributable-cache=example-distributed-cache:add(bean-management=example-bean-cache)
```

采用首选 JBoss EAP 8.0 配置可确保您的服务器与最新版本和将来的主版本兼容。您还可以从改进的资源和子系统中受益，用于可分布式 SFSB。

6.3. JAKARTA ENTERPRISE BEANS 服务器配置更改

在 JBoss EAP 7 中配置 **ejb3** 子系统时，在部署企业 bean 应用期间可能会出现在服务器日志中。

重要

如果您使用 JBoss 服务器迁移工具更新服务器配置，请确保正确配置了 **ejb3** 子系统，且部署 Jakarta Enterprise Beans 应用时没有问题。有关配置和运行工具的详情，请参考使用 JBoss 服务器迁移工具。

6.3.1. 解决由于缓存更改导致的 DuplicateServiceException

以下 **DuplicateServiceException** 错误是由 JBoss EAP 7 中的缓存更改导致的。

服务器日志中的 DuplicateServiceException

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-3) MSC000001: Failed to start service jboss.deployment.unit."mdb-1.0-SNAPSHOT.jar".cache-dependencies-installer: org.jboss.msc.service.StartException in service jboss.deployment.unit."mdb-1.0-SNAPSHOT.jar".cache-dependencies-installer: Failed to start service
...
Caused by: org.jboss.msc.service.DuplicateServiceException: Service jboss.infinispan.ejb."mdb-1.0-SNAPSHOT.jar".config is already registered
```

若要解决由于 JBoss EAP 7 中缓存更改导致的 **DuplicateServiceException**，请运行以下命令在 **ejb3** 子系统中重新配置缓存：

```
/subsystem=ejb3/file-passivation-store=file:remove
/subsystem=ejb3/cluster-passivation-store=infinispan:remove
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, max-size=10000)

/subsystem=ejb3/cache=passivating:remove
/subsystem=ejb3/cache=clustered:remove
/subsystem=ejb3/cache=distributable:add(passivation-store=infinispan, aliases=[passivating, clustered])
```

通过重新配置缓存，您可以解决这个错误并防止 **DuplicateServiceException** 发生。

6.4. 消息传递服务器配置变化

了解如何将配置和关联的消息传递数据迁移到 ActiveMQ Artemis，后者充当 Red Hat JBoss Enterprise Application Platform 8.0 中的 Jakarta 消息传递支持提供程序。

6.4.1. 迁移消息传递数据

回顾在 Red Hat JBoss Enterprise Application Platform 中迁移消息传递数据的方法。

要将消息传递数据从以前的 JBoss EAP 7.x 版本迁移到 JBoss EAP 8.0，您可以使用导出和导入方法迁移消息传递数据。此方法涉及从上一版本导出消息传递数据，并使用管理 CLI **import-journal** 操作将其导入到 JBoss EAP 8.0 中。请注意，这种方法仅适用于基于文件的消息传递系统。

与版本 7 一样，JBoss EAP 8.0 继续使用 ActiveMQ Artemis 作为 Jakarta 消息传递支持提供程序，这有助于使迁移过程平稳进行。

6.4.1.1. 使用导出和导入方法迁移消息传递数据

使用以下方法将上一发行版本中的消息传递数据导出到 XML 文件，然后使用 **import-journal** 操作导入该文件：

1. 从 JBoss EAP 7.x 导出消息传递数据。
2. 导入 XML 格式的消息传递数据

重要

您不能使用导出和导入方法在使用基于 JDBC 的日志进行存储的系统之间移动消息传递数据。

6.4.1.1.1. 从 JBoss EAP 7.x 发行版本导出消息传递数据

要从 Red Hat JBoss Enterprise Application Platform 7.x 发行版本导出消息传递数据，请按照概述的步骤操作。

先决条件

- JBoss EAP 7.x 已安装在您的系统上。
- 您可以访问终端或命令行界面。

- 您有浏览目录和执行命令所需的权限。

流程

1. 打开一个终端，导航到 JBoss EAP 7.x 安装目录，并以 **admin-only** 模式启动服务器。

```
$ EAP_HOME/bin/standalone.sh -c standalone-full.xml --start-mode=admin-only
```

2. 打开一个新终端，前往 JBoss EAP 7.x 安装目录，再连接管理 CLI。

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

3. 使用以下管理 CLI 命令导出消息传递日志数据：

```
/subsystem=messaging-activemq/server=default:export-journal()
```

验证

- 确保完成 命令时，日志中没有任何错误或警告消息。
- 使用与您的操作系统兼容的工具，在生成的输出文件中验证 XML。

6.4.1.1.2. 导入 XML 格式的消息传递数据

从 JBoss EAP 8.0 导出消息传递数据后，您需要使用 **import-journal** 操作将 XML 文件导入到 JBoss EAP 8.0 或更高版本。

先决条件

- 使用管理 CLI 迁移操作或 JBoss 服务器迁移工具完成 JBoss EAP 8.0 的迁移。
- 以正常模式启动 JBoss EAP 8.0 服务器，无需任何连接的 Jakarta 消息传递客户端。

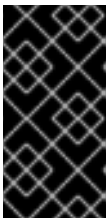
流程

要将 XML 文件导入到 JBoss EAP 8.0 或更高版本，请使用 **import-journal** 操作按照以下步骤操作：

重要

如果您的目标服务器已执行一些消息传递任务，请确保在开始 **import-journal** 操作前备份您的消息传递文件夹，以防止导入失败时出现数据丢失。如需更多信息，请参阅 备份消息传递文件夹数据。

1. 以正常模式启动 JBoss EAP 8.0 服务器，*没有连接* Jakarta 消息传递客户端。



重要

您在没有连接 Jakarta 消息传递客户端的情况下启动服务器非常重要。这是因为，**import-journal** 操作的行为类似于 Jakarta 消息传递制作者。操作过程中，消息会立即可用。如果此操作在导入期间失败，并且连接了 Jakarta 消息传递客户端，则无法恢复，因为 Jakarta 消息传递客户端可能已经消耗了一些消息。

2. 打开一个新终端，前往 JBoss EAP 8.0 安装目录，再连接到管理 CLI。

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

3. 使用以下管理 CLI 命令导入消息传递数据：

```
/subsystem=messaging-activemq/server=default:import-
journal(file=OUTPUT_DIRECTORY/OldMessagingData.xml)
```

重要

不要多次运行此命令，因为这样做会导致重复的消息。

6.4.1.1.3. 从导入消息传递数据失败中恢复

如果 **import-journal** 操作失败，您可以从导入消息传递数据失败中恢复。

先决条件

- 熟悉 JBoss EAP 8.0 服务器及其管理 CLI 命令。
- 了解消息传递日志文件夹的目录位置。
- 目标服务器消息传递数据之前备份（如果可用）。

流程

1. 关闭 JBoss EAP 8.0 服务器。
2. 删除所有消息传递日志文件夹。请参阅为管理 CLI 命令备份消息传递文件夹数据，以确定消息传递日志文件夹的正确目录位置。
3. 如果您在导入前备份了目标服务器消息传递数据，请将消息传递文件夹从备份位置复制到在上一步中确定的消息传递日志目录。
4. 重复步骤 以导入 XML 格式的消息传递数据。

6.4.1.2. 使用消息传递网桥迁移消息传递数据

Jakarta 消息传递网桥使用来自源 Jakarta Messaging 队列或主题的消息，并将它们发送到不同服务器上的目标 Jakarta 消息传递队列或主题。它启用了遵循 Jakarta Messaging 3.1 标准的消息传递服务器之间的消息桥接。使用 Java 命名和目录接口查找源和目标 Jakarta 消息传递资源，确保模块中捆绑了用于 Java 命名和目录接口查找的客户端类，并在 Jakarta 消息传递网桥配置中声明模块名称。

本节介绍了如何配置服务器并部署消息传递网桥，以便将消息传递数据从 JBoss EAP 7 移到 JBoss EAP 8.0。要做到这一点，请执行以下步骤：

1. 配置 JBoss EAP 8.0 服务器
2. 迁移消息传递数据

6.4.1.2.1. 配置 JBoss EAP 8.0 服务器

要在 JBoss EAP 8.0 中配置 Jakarta 消息传递网桥，以便无缝迁移消息传递数据，包括模块依赖项和队列配置，请按照以下步骤操作。

先决条件

- JBoss EAP 8.0 服务器已安装并运行。

流程

1. 在 JBoss EAP 8.0 服务器的 **messaging-activemq** 子系统中，为默认服务器创建以下 **jms-queue** 配置。

```
jms-queue add --queue-address=MigratedMessagesQueue --entries=
[jms/queue/MigratedMessagesQueue
java:jboss/exported/jms/queue/MigratedMessagesQueue]
```

2. 确保 **messaging-activemq** 子系统 **默认服务器** 包含 **InVmConnectionFactory connection-factory** 的配置，如下所示：

```
<connection-factory name="InVmConnectionFactory" factory-type="XA_GENERIC"
entries="java:/ConnectionFactory" connectors="in-vm"/>
```

如果没有包含该条目，请使用以下管理 CLI 命令创建一个：

```
/subsystem=messaging-activemq/server=default/connection-
factory=InVmConnectionFactory:add(factory-type=XA_GENERIC, connectors=[in-vm],
entries=[java:/ConnectionFactory])
```

3. 创建和部署 Jakarta 消息传递网桥，该网桥从 **InQueue** JMS 队列读取消息，并将它们传输到 JBoss EAP 7.x 服务器上配置的 **MigratedMessagesQueue**。

```
/subsystem=messaging-activemq/jms-bridge=myBridge:add(add-messageID-in-
header=true,max-batch-time=100,max-batch-size=10,max-retries=-1,failure-retry-
interval=1000,quality-of-service=AT_MOST_ONCE,module=org.hornetq,source-
destination=jms/queue/InQueue,source-connection-
factory=jms/RemoteConnectionFactory,source-context=
[("java.naming.factory.initial"=>"org.wildfly.naming.client.WildFlyInitialContextFactory"),
("java.naming.provider.url"=>"http-remoting://legacy-host:8080")],target-
destination=jms/queue/MigratedMessagesQueue,target-connection-
factory=java:/ConnectionFactory)
```

这会在 JBoss EAP 8.0 服务器的 **messaging-activemq** 子系统中创建以下 **jms-bridge** 配置。

```
<jms-bridge name="myBridge" add-messageID-in-header="true" max-batch-time="100" max-
batch-size="10" max-retries="-1" failure-retry-interval="1000" quality-of-
service="AT_MOST_ONCE">
    <source destination="jms/queue/InQueue" connection-
factory="jms/RemoteConnectionFactory">
        <source-context>
            <property name="java.naming.factory.initial"
value="org.wildfly.naming.client.WildFlyInitialContextFactory"/>
            <property name="java.naming.provider.url" value="http-remoting://legacy-host:8080"/>
        </source-context>
    </source>
    <target destination="jms/queue/MigratedMessagesQueue" connection-
factory="java:/ConnectionFactory"/>
</jms-bridge>
```

6.4.1.2.2. 迁移消息传递数据

要将消息传递数据从 Red Hat JBoss Enterprise Application Platform 8.0 迁移到 Red Hat JBoss Enterprise Application Platform 8.0，请按照以下步骤执行。

先决条件

- JBoss EAP 8.0 服务器已安装并运行。

流程

1. 验证您为以下配置提供的信息是否正确。
   - 任何队列和主题名称。
   - **java.naming.provider.url** 用于 Java 命名和目录接口查找。
2. 确保您已将目标 Jakarta Messaging 目的地部署到 JBoss EAP 8.0 服务器。
3. 启动 JBoss EAP 8.0 服务器，包括迁移过程中涉及的 JBoss EAP 7 服务器。

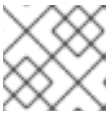
6.4.1.3. 备份消息传递文件夹数据

为确保数据完整性，建议在更改目标消息前备份目标消息文件夹（如果您的服务器已处理消息）。您可以在 ***EAP_HOME*/standalone/data/activemq/** 中找到消息传递文件夹的默认位置，但可能会配置它。如果您不确定消息传递数据的位置，您可以使用以下管理 CLI 命令确定它。

流程

1. 使用以下管理 CLI 命令确定消息传递数据的位置：

```
/subsystem=messaging-activemq/server=default/path=journal-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=paging-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=bindings-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=large-messages-directory:resolve-path
```



注意

在复制数据前，请确保停止服务器。

2. 在确定其各自位置后，将每个消息传递文件夹复制到安全备份位置。

6.4.2. 配置 Jakarta Messaging 资源适配器

在 Red Hat JBoss Enterprise Application Platform 8.0 中更改了配置通用 Jakarta 消息传递资源适配器以用于第三方 Jakarta 消息传递提供程序的方法。如需更多信息，请参阅 JBoss EAP 7.4 配置消息传递指南中的 部署通用 Java 消息服务资源适配器。

6.4.3. 消息传递配置更改

在 Red Hat JBoss Enterprise Application Platform 7.0 中，如果您在没有指定 ***check-for-live-server*** *属性的情况下配置了* ***replication-primary*** *策略，则其默认值被设置为* ***false***。*JBoss EAP 7.1 及更高版本中发生了这一变化。* ***check-for-live-server*** *属性的默认值现在设为* ***true***。

以下是管理 CLI 命令的示例，该命令配置 **replication-primary** 策略，而不指定 **check-for-live-server** 属性。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-primary:add(cluster-name=my-cluster,group-name=group1)
```

使用管理 CLI 读取资源时，请注意 **check-for-live-server** 属性值设为 **true**。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-primary:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "check-for-live-server" => true,
        "cluster-name" => "my-cluster",
        "group-name" => "group1",
        "initial-replication-sync-timeout" => 30000L
    },
    "response-headers" => {"process-state" => "reload-required"}
}
```

6.4.4. 嵌入式代理消息传递的 Galleon 层

在 JBoss EAP 7 中，嵌入式消息传递代理是默认安装的一部分。在 JBoss EAP 8 中，这个功能被添加到一个新的 Galleon 层中，称为 **embedded-activemq**。

这个新层不是默认配置的一部分，因此希望依赖在 JBoss EAP 中嵌入代理的用户必须在其配置中明确包含该代理。

该层提供了一个带有嵌入式代理的 **messaging-activemq** 子系统，即使建议客户在 OpenShift 上使用专用 AMQ 集群。它还置备辅助资源，如 socket-bindings 和支持此用例所需的依赖项。

6.5. JBOSS EAP 8.0 中的安全增强

从 JBoss EAP 8.0 开始，您必须使用 Elytron，因为旧的 security 子系统和旧的安全域不再可用。您只能使用 JBoss 服务器迁移工具配置 Elytron 默认值。因此，必须手动迁移旧的安全配置。

其他资源

- 迁移到 Elytron

6.5.1. Vault 迁移

Vault 已从 JBoss EAP 8.0 中删除。使用 **elytron** 子系统提供的凭据存储来存储敏感字符串。

其他资源

- 迁移安全密码库和属性
- Elytron 中的凭证存储

6.5.2. 旧的 security 子系统和安全域删除

旧的 security 子系统和旧的安全域已从 JBoss EAP 8.0 中删除。使用 **elytron** 子系统提供的安全域。

其他资源

- 迁移身份验证配置
- 将数据库身份验证配置迁移到 Elytron
- 将 Composite Stores 迁移到 Elytron
- 将使用缓存的安全域迁移到 Elytron
- 将基于传统属性的配置迁移到 Elytron
- 将 LDAP 身份验证配置迁移到 Elytron
- 将 Kerberos 身份验证迁移到 Elytron
- 迁移 SSL 配置
- 使用身份存储保护应用程序和管理界面
- 使用多个身份存储保护应用程序和管理界面

6.5.3. PicketLink 子系统删除

PicketLink 子系统已从 JBoss EAP 8.0 中删除。使用 Red Hat build of Keycloak 而不是 PicketLink 身份提供程序，以及红帽构建的 Keycloak SAML 适配器而不是 PicketLink 服务供应商。

其他资源

- PicketLink 删除
- 红帽构建的 Keycloak

6.5.4. 从红帽构建的 Keycloak OIDC 客户端适配器迁移到 JBoss EAP 子系统

JBoss EAP 8.0 不支持 **keycloak** 子系统，它被 **elytron-oidc-client** 子系统替代。JBoss 服务器迁移工具默认执行迁移。

其他资源

- 迁移 **keycloak** 子系统
- JBoss EAP 中的 OpenID Connect 配置

6.5.5. 自定义登录模块迁移

在 JBoss EAP 8.0 中，旧的 security 子系统已被删除。要继续使用 **elytron** 子系统的自定义登录模块，请使用新的 Java 身份验证和授权服务(JAAS)安全域和 **jaas-realm**。

其他资源

- **elytron** 子系统中的 JAAS 域

6.5.6. FIPS 模式更改

从 JBoss EAP 7.1 开始，默认启用自动生成自签名证书用于开发目的。如果您以 FIPS 模式运行，请配置服务器以禁用自动自签名证书创建。否则，在启动服务器时可能会导致以下错误：

```
ERROR [org.xnio.listener] (default I/O-6) XNIO001007: A channel event listener threw an exception: java.lang.RuntimeException: WFLYDM0114: Failed to lazily initialize SSL context
...
Caused by: java.lang.RuntimeException: WFLYDM0112: Failed to generate self signed certificate
...
Caused by: java.security.KeyStoreException: Cannot get key bytes, not PKCS#8 encoded
```

其他资源

- 使用自动生成的自签名证书为应用程序启用 SSL/TLS

6.6. MOD_CLUSTER 配置更改

Red Hat JBoss Enterprise Application Platform 7.4 中更改了 mod_cluster 中静态代理列表的配置。

从 JBoss EAP 7.4 开始，**proxy-list** 属性已弃用，然后在 JBoss EAP 8.0 中删除。

它已被 **proxies** 属性替代，这个属性是出站套接字绑定名称的列表。

这个更改会影响您定义静态代理列表的方式，例如，在为 mod_cluster 禁用广告时。有关如何为 mod_cluster 禁用广告的详情，请参考 JBoss EAP 7.4 配置指南中的 mod_cluster 的禁用广告。

为确保与 JBoss EAP 8.0 的兼容性，请更新用户脚本和旧用户 CLI 脚本，如下所示：

- 将已弃用的 **ssl=configuration** 替换为 **基于 elytron** 的适当配置。
- 将 mod_cluster 配置路径从 **/mod-cluster-config=CONFIGURATION** 更新至 **/proxy=default**。
- 更新用户脚本中的动态负载提供程序路径，将已弃用的路径替换为 **provider=dynamic**。
- 已弃用 **的连接器** 属性（称为 Undertow 侦听器）已被删除。更新您的用户脚本，以使用 **listener** 属性作为替换。

有关 mod_cluster 属性的更多信息，请参阅 JBoss EAP 7.4 配置指南中的 ModCluster 子系统属性。

6.7. 查看配置更改

使用红帽 JBoss 企业应用平台 7，您可以跟踪对正在运行的服务器所做的配置更改。您还可以查看授权用户所做的配置更改历史记录。

而使用 JBoss EAP 7.0 时，您必须使用 **core-service** 管理 CLI 命令配置选项并检索最近配置更改的列表。

示例：列出 JBoss EAP 7.0 中的配置更改

```
/core-service=management/service=configuration-changes:add(max-history=10)
/core-service=management/service=configuration-changes:list-changes
```

JBoss EAP 7.1 引入了一个新的 **core-management** 子系统，可以配置为跟踪对正在运行的服务器的配置更改。这是在 JBoss EAP 7.1 及更高版本中配置和查看配置更改的首选方法。

示例：列出 JBoss EAP 7.1 及更高版本中的配置更改

```
/subsystem=core-management/service=configuration-changes:add(max-history=20)
/subsystem=core-management/service=configuration-changes:list-changes
```

有关使用 JBoss EAP 7.1 中引入的新 **core-management** 子系统的更多信息，请参阅 JBoss EAP 7.4 配置指南中的 查看配置更改。

第 7 章 了解应用程序迁移更改

本节介绍将应用程序从 JBoss EAP 6.4 或 7.x 迁移到 JBoss EAP 8.0 所需的更改。

7.1. WEB 服务应用程序更改

JBossWS 5 对 JBoss EAP 7 web 服务提供了新的特性和性能改进，主要是通过升级 Apache CXF、Apache WSS4J 和 Apache Santuario 组件。然后，JBoss EAP 8.0 使用 JBossWS 7 支持其功能。

7.1.1. JAX-RPC 支持更改

基于 XML 的 RPC Java API (JAX-RPC) 在 Java EE 6 中已被弃用，在 Java EE 7 中是可选的。从 JBoss EAP 7 开始，它不再被支持。使用 JAX-RPC 的应用必须迁移到使用 Jakarta XML Web 服务，该服务为当前的 Jakarta EE 标准 Web 服务框架。

可使用以下任一方式识别 JAX-RPC Web 服务：

- 存在 JAX-RPC 映射文件，它是带有 root 元素 < **java-wsdl-mapping** > 的 XML 文件。
- 存在 **webservices.xml** XML 描述符文件，其中包含 **<webservice-description>** 元素，其中包含一个 **<jaxrpc-mapping-file>** 子元素。以下是定义 JAX-RPC Web 服务的 **webservices.xml** 描述符文件示例。

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd" version="1.1">
 <webservice-description>
  <webservice-description-name>HelloService</webservice-description-name>
  <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
  <jaxrpc-mapping-file>WEB-INF/mapping.xml</jaxrpc-mapping-file>
  <port-component>
   <port-component-name>Hello</port-component-name>
   <wsdl-port>HelloPort</wsdl-port>
   <service-endpoint-interface>org.jboss.chap12.hello.Hello</service-endpoint-interface>
   <service-impl-bean>
    <servlet-link>HelloWorldServlet</servlet-link>
   </service-impl-bean>
  </port-component>
 </webservice-description>
</webservices>
```

- 存在 **ejb-jar.xml** 文件，该文件中包含引用一个 JAX-RPC 映射文件的 **<service-ref>**。

7.1.2. Apache CXF Spring web 服务更改

在以前的 JBoss EAP 版本中，您可以通过包含带有端点部署存档的 **jbossws-cxf.xml** 配置文件来自定义 JBossWS 和 Apache CXF 集成。一个用例是，为 Apache CXF 总线上的 Web 服务客户端和服务器端点配置拦截器链。此集成需要在 JBoss EAP 服务器中部署 Spring。

JBoss EAP 8 不再支持 Spring 集成。必须修改包含 **jbossws-cxf.xml** 描述符配置文件的任何应用程序，以替换该文件中定义的自定义配置。虽然仍然可以直接访问 Apache CXF API，但请注意，应用程序将无法可移植。

建议的方法将 Spring 自定义配置替换为新的 JBossWS 描述符配置选项。基于 JBossWS 描述符的方法提供类似功能，无需修改客户端端点代码。在某些情况下，您可以将 Spring 替换为上下文依赖注入(CDI)。

7.1.2.1. Apache CXF 拦截器

JBossWS 描述符提供新的配置选项，允许您在不修改客户端端点代码的情况下声明拦截器。反之，您可以通过为 **cxf.interceptors.in** 和 **cxf.interceptors.out** 属性指定拦截器名称列表，在预定义的客户端和端点配置中声明拦截器。

以下是使用这些属性声明拦截器的 **jaxws-endpoint-config.xml** 文件的示例。

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_0.xsd">
  <endpoint-config>
    <config-name>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointImpl</config-name>
    <property>
      <property-name>cxf.interceptors.in</property-name>
      <property-
value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointInterceptor,org.jboss.test.ws.jaxws.cxf.intercepto
s.FooInterceptor</property-value>
    </property>
    <property>
      <property-name>cxf.interceptors.out</property-name>
      <property-value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointCounterInterceptor</property-
value>
    </property>
  </endpoint-config>
</jaxws-config>
```

7.1.2.2. Apache CXF 功能

JBossWS 描述符允许您通过为 **cxf.features** 属性指定功能列表，在预定义的客户端和端点配置中声明功能。

以下是使用此属性声明功能的 **jaxws-endpoint-config.xml** 文件示例。

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_0.xsd">
  <endpoint-config>
    <config-name>Custom FI Config</config-name>
    <property>
      <property-name>cxf.features</property-name>
      <property-value>org.apache.cxf.feature.FastInfosetFeature</property-value>
    </property>
  </endpoint-config>
</jaxws-config>
```

7.1.2.3. Apache CXF HTTP 传输

在 Apache CXF 中，通过指定 **org.apache.cxf.transport.http.HTTPConduit** 选项来实现 HTTP 传输配置。JBossWS 集成允许以编程方式使用 Apache CXF API 修改，如下所示：

```
import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.transport.http.HTTPConduit;
import org.apache.cxf.transports.http.configuration.HTTPClientPolicy;

// Set chunking threshold before using a JAX-WS port client
...
HTTPConduit conduit = (HTTPConduit)ClientProxy.getClient(port).getConduit();
HTTPClientPolicy client = conduit.getClient();

client.setChunkingThreshold(8192);
...
```

您还可以通过设置系统属性来控制并覆盖 Apache CXF **HTTPConduit** 默认值。

属性	类型	描述
cxf.client.allowChunking	布尔值	指定是否使用块发送请求。
cxf.client.chunkingThreshold	整数	设置从非区块切换到块模式的阈值。
cxf.client.connectionTimeout	Long	设置连接超时的毫秒数。
cxf.client.receiveTimeout	Long	设置接收超时的毫秒数。
cxf.client.connection	字符串	指定是否使用 **Keep-Alive** 或 **close** 连接类型。
cxf.tls-client.disableCNCheck	布尔值	指定是否禁用 CN 主机名检查。

7.1.3. WS-Security 更改

本节介绍 JBoss EAP 6.4 和 JBoss EAP 7.0 中应用的各种 WS-Security 更改。

- 如果您的应用程序包含访问 **org.apache.ws.security.WSPasswordCallback** 类的自定义回调处理器，请注意，此类已移到 **org.apache.wss4j.common.ext** 软件包。
- 来自 **org.apache.ws.security.saml.ext** 软件包中的大多数 SAML bean 对象已被移到 **org.apache.wss4j.common.saml** 软件包中。
- 默认情况下，不允许使用 RSA v1.5 密钥传输和所有相关算法。
- 以前，安全令牌服务(STS)仅在 **BehalfOf** 令牌上进行验证。现在，它还会验证 **ActAs** 令牌。因此，必须在为 **ActAs** 令牌提供的 **UsernameToken** 中指定有效的用户名和密码。
- 现在，需要 SAML Bearer 令牌具有内部签名。**org.apache.wss4j.dom.SamlAsertionValidator** 类现在有一个 **setRequireBearerSignature()** 方法，用于启用或禁用签名验证。

7.1.4. JBoss 模块结构更改

cxf-api 和 **cxf-rt-core** JARs 已合并到一个 **cxf-core** JAR 中。因此，JBoss EAP 中的 **org.apache.cxf** 模块现在包含 **cxf-core** JAR，并公开比上一版本更多的类。

7.1.5. Bouncy Castle 要求和更改

如果要将 AES 加密与 Galois/Counter Mode(GCM)用于 XML/WS-Security 中的对称加密，则需要 BouncyCastle Security Provider。

从 JBoss EAP 7 开始，它包含在 **org.bouncycastle** 模块中，JBossWS 能够依赖其类加载程序来获取和使用 BouncyCastle Security Provider。因此，在当前 JVM 中静态安装 BouncyCastle 不再需要。对于在容器外运行的应用程序，可以通过向类路径添加 BouncyCastle 库来向 JBossWS 提供安全供应商。

您可以禁用此行为：把 **jaxws-endpoint-config.xml** 部署描述符文件（对于服务器）或 **jaxws-client-config.xml** 部署描述符文件（对于客户端）中的 **org.jboss.ws.cxf.noLocalBC** 属性值设置为**true**。

如果要使用与 JBoss EAP 附带的版本不同的版本，您仍然可以静态将 BouncyCastle 安装到 JVM。在这种情况下，按照类路径中的提供者选择静态安装的 BouncyCastle Security Provider。要避免任何问题，您必须使用 BouncyCastle 1.72 或更高版本。

7.1.6. Apache CXF 总线选择策略

在容器中运行的客户端的默认总线选择策略已从 **THREAD_BUS** 改为 **TCCL_BUS**。对于运行内存不足的客户端，默认的策略仍然是 **THREAD_BUS**。您可以使用以下方法之一将行为恢复到上一版本的行为。

- 使用系统属性 **org.jboss.ws.cxf.jaxws-client.bus.strategy** 值设置为 **THREAD_BUS** 来启动 JBoss EAP 服务器。
- 在客户端代码中明确设置选择策略。

7.1.7. WebServiceRef 的 Jakarta XML Web Services 2.2 要求

容器必须使用 Jakarta XML Web Services 2.2 风格构造器（在构造器中包含 WebServiceFeature 类）以构建注入 Web 服务引用的客户端。JBoss EAP 6.4 包括了 JBossWS 4，隐藏了这一要求。从包含 JBossWS 5 的 JBoss EAP 7 开始，此要求不会被隐藏。这表明，容器注入的服务类必须通过更新现有代码来实现 Jakarta XML Web 服务 2.2 或更高版本，以使用包含一个或多个 **WebServiceFeature** 参数的 **jakarta.xml.ws.Service** 构造器。

```
protected Service(URL wsdlDocumentLocation,
    QName serviceName,
    WebServiceFeature... features)
```

7.1.8. IgnoreHttpsHost CN 检查更改

在以前的版本中，您可以通过将系统属性 **org.jboss.security.ignoreHttpsHost** 设置为 **true** 来禁用对证书中提供的服务通用名称(CN)的 HTTPS URL 主机名检查。这个系统属性名称已被 **cxf.tls-client.disableCNCheck** 替代。

7.1.9. 服务器端配置和类加载

由于启用注入服务端点和服务客户端处理程序，便无法再自动加载来自 **org.jboss.as.webservices.server.integration** JBoss 模块的处理程序类。如果您的应用程序依赖于给定的预定义配置，您可能需要为部署显式定义新的模块依赖项。如需更多信息，请参阅 迁移显式模块依赖项。

7.1.10. 弃用 Java 结束的标准覆盖机制

JDK 1.8_40 中弃用了 Java 背书的标准覆盖机制，旨在将其从 JDK 9 中删除。这种机制允许开发人员通过将 JAR 放置到 JRE 中的终端目录中，使库可供所有部署的应用程序使用。

从 JBoss EAP 7 发行版本开始，如果您的应用程序使用了 Apache CXF 的 JBossWS 实施，它会确保以正确顺序添加所需的依赖项，您不应受到此更改的影响。如果您的应用程序直接访问 Apache CXF，则现在必须在 JBossWS 依赖项后提供 Apache CXF 依赖项，作为应用程序部署的一部分。

7.1.11. EAR 归档中的描述符规格

在以前的 JBoss EAP 版本中，您可以在 JAR 归档的 **META-INF/** 目录中为 Jakarta Enterprise Beans Web 服务部署配置 **jboss-webservices.xml** 部署描述符文件，或在 POJO Web 服务部署和 POJO Web 服务部署和 Jakarta Enterprise Beans Web 服务端点中捆绑在 WAR 归档的 **WEB-INF/** 目录中配置 jboss-webservices.xml 部署描述符文件。

从 JBoss EAP 7 开始，您可以在 EAR 存档的 **META-INF/** 目录中配置 **jboss-webservices.xml** 部署描述符文件。如果在 EAR 归档和 JAR 或 WAR 归档中找到 **jboss-webservices.xml** 文件，则 JAR 或 WAR 文件中的配置数据会覆盖 EAR 描述符文件中的对应数据。

7.2. 更新远程 URL 连接器和端口

从 JBoss EAP 7 开始，默认连接器已从 **remote** 改为 **http-remoting**，默认的远程连接端口已从 **4447** 改为 **8080**。默认配置的 JNDI 提供程序 URL 已从 **remote://localhost:4447** 改为 **http-remoting://localhost:8080**。

7.3. 消息传递应用程序更改

本节介绍 JBoss EAP 7 中的各种消息传递应用更改。另外，您还可以了解更多有关如何：

- 更改 Jakarta Messaging 部署描述符
- 更新外部 Jakarta Messaging 客户端
- 替换已弃用的地址设置属性
- 配置所需的消息传递应用程序更改

7.3.1. 替换或更新 Jakarta Messaging 部署描述符

从 JBoss EAP 7 开始，由命名模式 **-jms.xml** 标识的专有 HornetQ 消息传递资源部署描述符文件无法正常工作。以下是 JBoss EAP 6 中的 Java Message Service 资源部署描述符文件的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
  <hornetq-server>
    <jms-destinations>
      <jms-queue name="testQueue">
        <entry name="queue/test"/>
        <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
        <entry name="topic/test"/>
        <entry name="java:jboss/exported/jms/topic/test"/>
```

```
    </jms-topic>
   </jms-destinations>
  </hornetq-server>
</messaging-deployment>
```

如果您在上一发行版本中的应用程序中使用 **-jms.xml** Java Message Service 部署描述符，您可以转换应用程序以使用 Jakarta EE 平台的 Resource Definition 和 Configuration 部分中指定的标准部署描述符，或者您可以更新部署描述符以使用 **messaging-activemq-deployment** 模式。如果您选择更新描述符，您需要进行以下修改：

- 将命名空间从 "urn:jboss:messaging-deployment:1.0" 改为 "urn:jboss:messaging-activemq-deployment:1.0"。
- 将 **<hornetq-server>** 元素的名称改为 **<server>**。

修改的文件应类似以下示例。

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-activemq-deployment:1.0">
 <server>
  <jms-destinations>
   <jms-queue name="testQueue">
    <entry name="queue/test"/>
    <entry name="java:jboss/exported/jms/queue/test"/>
   </jms-queue>
   <jms-topic name="testTopic">
    <entry name="topic/test"/>
    <entry name="java:jboss/exported/jms/topic/test"/>
   </jms-topic>
  </jms-destinations>
 </server>
</messaging-deployment>
```

7.3.2. 替换 HornetQ API

JBoss EAP 6 包含 **org.hornetq** 模块，允许您在应用源代码中使用 HornetQ API。

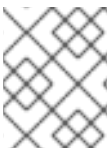
Apache ActiveMQ Artemis 取代了 JBoss EAP 7 中的 HornetQ，因此您必须迁移使用 HornetQ API 的任何代码以使用 Apache ActiveMQ Artemis API。此 API 的库包含在 **org.apache.activemq.artemis** 模块中。

ActiveMQ Artemis 是 HornetQ 的演进，因此许多概念仍然适用。

7.3.3. 替换已弃用的地址设置属性

使用 auto-create-jms-queues、 **auto-delete-jms-queues** 、**auto-create-jms-topics** 和 **auto-delete-jms-topics** 属性自动创建和自动删除主题和队列的功能仅在 JBoss EAP 7 中完全配置。这些属性已弃用，它们 仅作为技术预览功能提供，且不被支持。

您必须使用以下替换属性替换这些已弃用的属性。



注意

弃用的属性不再配置此功能，因为 JBoss EAP 8.0 无效。不支持替换属性。它们仅作为在最佳工作基础之上迁移的方法。

弃用的属性	替换属性
auto-create-jms-queues	auto-create-queues
auto-delete-jms-queues	auto-delete-queues
auto-create-jms-topics	auto-create-addresses
auto-delete-jms-topics	auto-delete-addresses

在 JBoss EAP 6 中，默认的地址设置属性设为 ***false****。从 JBoss EAP 7 开始，默认情况下替换属性设为* ***true****。*

如果您希望保留 JBoss EAP 6 行为，您必须将替换属性设置为 ***false****。*

有关这些替换属性的更多信息，请参阅 JBoss EAP 7.4 配置消息传递 指南中的 地址设置属性。

7.3.4. JBoss EAP 7 所需的消息传递应用程序更改

从 JBoss EAP 7.2 开始，如果客户端应用直接依赖于 INVENTORY 客户端 JAR，例如： ***artemis-jms-client****、****artemis-commons****、****artemis-core-client****，或* ***artemis-selector****，那么您必须在* ***pom.xml*** *文件中为* ***wildfly-client-properties*** *添加以下依赖关系。*

```
<dependency>
  <groupId>org.jboss.eap</groupId>
  <artifactId>wildfly-client-properties</artifactId>
</dependency>
```

在从旧的 JBoss EAP 7 客户端中调用 ***message.getJMSReplyTo()*** *时，避免* ***JMSRuntimeException****，如 JBEAP-15889 所述。*

7.4. JAKARTA RESTFUL WEB 服务和 RESTEASY 应用更改

JBoss EAP 6 捆绑了 RESTEasy 2，它是 JAX-RS 1.x 的实施。

JBoss EAP 7 和 JBoss EAP 7.1 包含 RESTEasy 3.0.x，这是由 JSR 339: JAX-RS 2.0 定义的 JAX-RS 2.0 实施：RESTful Web 服务规范的 Java API。

JBoss EAP 7.4 包括 RESTEasy 3.15，这是 Jakarta RESTful Web Services 2.1 的实施。此发行版本还添加了对 JDK 11 的支持。在提供一些 RESTEasy 4 关键功能时，此版本基于 RESTEasy 3.0，确保完全向后兼容。因此，您应该在从 RESTEasy 3.0 迁移到 RESTEasy 3.15 时遇到几个问题。有关 RESTEasy RESTEasy 3.15 的 Java API 的更多信息，请参阅 RESTEasy Jakarta RESTful Web Services 3.15.0.Final API 。

JBoss EAP 8.0 支持 RESTEasy 6.2，它实现了 Jakarta RESTful Web Services 3.1 规范 。

如果您要从 JBoss EAP 6.4 迁移，请注意 JBoss EAP 中包含的 Jackson 版本发生了变化。JBoss EAP 6.4 包括 Jackson 1.9.9。JBoss EAP 7 及更高版本现在包含 Jackson 2.6.3 或更高版本。

本节介绍这些更改如何影响使用 RESTEasy 或 Jakarta RESTful Web 服务的应用程序。

7.4.1. RESTEasy 弃用类

拦截器和 MessageBody 类
*JSR 311：JAX-RS：RESTful Web 服务的 Java™ API 没有包含拦截器框架，因此 RESTEasy 2 提供一个。JSR 339：JAX-RS 2.0：RESTful Web 服务的 Java API 引入了官方拦截器和过滤框架，因此 RESTEasy 2 中包含的拦截器框架现已弃用，并由 RESTEasy 3.x 中的 Jakarta REST 兼容拦截器工具替代。相关的接口在 **jakarta.ws.rs.api** 模块的 **jakarta.ws.rs.ext** 软件包中定义。*

JBoss EAP 8.0 中删除了以下提供程序：

- ***org.jboss.resteasy:resteasy-jackson-provider***
- ***org.jboss.resteasy:resteasy-jettison-provider***
- ***org.jboss.resteasy:resteasy-yaml-provider***

JBoss EAP 8.0 中删除了以下内容，因为它们现在有 Jakarta RESTful Web Services 替换。

- ***@suspend** 和 **org.jboss.resteasy.spi.AsynchronousResponse** 已被删除，并分别替换为 **@Suspended** 和 **javax.ws.rs.container.AsyncResponse**。*
- ***StringConverter** 替换为 **ParamConverter**。*
- ***org.jboss.resteasy.plugins.providers.SerializableProvider** 已被弃用，并已被删除。*
- *RESTEasy 3.x 中弃用的以下拦截器接口已被删除。*
  - ***org.jboss.resteasy.spi.interception.PreProcessInterceptor***
  - ***org.jboss.resteasy.spi.interception.PostProcessInterceptor***
  - ***org.jboss.resteasy.spi.interception.ClientExecutionInterceptor***
  - ***org.jboss.resteasy.spi.interception.ClientExecutionContext***
  - ***org.jboss.resteasy.spi.interception.AcceptedByMethod***
- ***org.jboss.resteasy.spi.interception.PreProcessInterceptor** 接口被 RESTEasy 3.x 中的 **jakarta.ws.rs.container.ContainerRequestFilter** 接口替代。*
- *以下接口和类已从 RESTEasy 3.x 和 JBoss EAP 8.0 中删除。*
  - ***org.jboss.resteasy.spi.interception.MessageBodyReaderInterceptor***
  - ***org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor***
  - ***org.jboss.resteasy.spi.interception.MessageBodyWriterContext***
  - ***org.jboss.resteasy.spi.interception.MessageBodyReaderContext***
  - ***org.jboss.resteasy.core.interception.InterceptorRegistry***
  - ***org.jboss.resteasy.core.interception.InterceptorRegistryListener***
  - ***org.jboss.resteasy.core.interception.ClientExecutionContextImpl***
- ***org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor** 接口被 **jakarta.ws.rs.ext.WriterInterceptor** 接口替代。*

- 另外，对 **jakarta.ws.rs.ext.MessageBodyWriter** 接口的一些更改可能无法向后兼容 JAX-RS 1.x。如果您的应用使用了 JAX-RS 1.x，请检查您的应用程序代码，以确保为您的端点定义了 **@Produces** 或 **@Consumes**。如果不这样做，可能会导致错误类似如下：

```
org.jboss.resteasy.core.NoMessageBodyWriterFoundFailure: Could not find
MessageBodyWriter for response object of type: <OBJECT> of media type:
```

以下是可能导致此错误的 REST 端点示例。

```
@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
          // ** DISCLAIMER **. This example is contrived.
          throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
Type.TEXT_PLAIN)
                .build());
        }
        return days;
    }
}
```

要解决这个问题，请添加 **jakarta.ws.rs.Produces** 和 **@Produces** 注解的导入，如下所示：

```
...
import jakarta.ws.rs.Produces;
...

@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    @Produces(MediaType.TEXT_PLAIN)
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
          // ** DISCLAIMER **. This example is contrived.
          throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
```

```
Type.TEXT_PLAIN)
                .build());
        }
        return days;
    }
}
```

注意

来自以前版本的 RESTEasy 的所有拦截器都可以与新的 Jakarta REST 过滤器和拦截器接口并行运行。

其他资源

- RESTEasy Interceptors
- RESTEasy Jakarta RESTful Web Services 3.15.0.Final API

客户端 API

resteasy-jaxrs 中的 RESTEasy 客户端框架被 JBoss EAP 7.0 中的遵循 JAX-RS 2.0 的 **resteasy-client** 模块替代。因此，一些 RESTEasy 客户端 API 类和方法已弃用。

- 以下类已从 JBoss EAP 8.0 中删除。
  - **org.jboss.resteasy.client.ClientRequest**
  - **org.jboss.resteasy.client.ClientRequestFactory**
  - **org.jboss.resteasy.client.ClientResponse**
  - **org.jboss.resteasy.client.ProxyBuilder**
  - **org.jboss.resteasy.client.ProxyConfig**
  - **org.jboss.resteasy.client.ProxyFactory**
- **org.jboss.resteasy.client.ClientResponseFailure** 异常和 **org.jboss.resteasy.client.ClientExecutor** 和 **org.jboss.resteasy.client.EntityTypeFactory** 接口也被弃用。
- 您必须将 **org.jboss.resteasy.client.ClientRequest** 和 **org.jboss.resteasy.client.ClientResponse** 类分别替换为 **org.jboss.resteasy.client.jaxrs.ResteasyClient** 和 **jakarta.ws.rs.core.Response**。
  以下是如何在 RESTEasy 2.3.x 中使用 RESTEasy 客户端发送链接标头的示例。

```
ClientRequest request = new ClientRequest(generateURL("/linkheader/str"));
request.addLink("previous chapter", "previous", "http://example.com/TheBook/chapter2",
null);
ClientResponse response = request.post();
LinkHeader header = response.getLinkHeader();
```

以下是如何使用 RESTEasy 3 中的 RESTEasy 客户端完成相同的任务的示例：

```
ResteasyClient client = new ResteasyClientBuilder().build();
Response response = client.target(generateURL("/linkheader/str")).request()
```

```
    .header("Link", "<http://example.com/TheBook/chapter2>; rel=\"previous\";
title=\"previous chapter\"").post(Entity.text(new String()));
jakarta.ws.rs.core.Link link = response.getLink("previous");
```

如需与 Jakarta REST Web 服务交互的外部 Jakarta REST RESTEasy 客户端的示例，请参阅 **resteasy-jaxrs-client** Quickstart。

- **org.jboss.resteasy.client.cache** 软件包中的类和接口也被弃用。它们由 **org.jboss.resteasy.annotations.cache** 软件包中的等效类和接口替代。

注意

有关 **org.jboss.resteasy.client.jaxrs** API 类的更多信息，请参阅 RESTEasy Jakarta REST JavaDoc。

StringConverter
org.jboss.resteasy.spi.StringConverter 类在 RESTEasy 3.x 和 JBoss EAP 8.0 中被弃用。可以使用 Jakarta REST jakarta.ws.rs.ext.ParamConverterProvider 类替换此功能。

7.4.2. 删除或保护的 RESTEasy 类

ResteasyProviderFactory 添加方法
大多数 **org.jboss.resteasy.spi.ResteasyProviderFactory** **add （）** 方法已被删除或在 RESTEasy 3.0 中受到保护。例如，**addBuiltInMessageBodyReader()** 和 **addBuiltInMessageBodyWriter()** 方法已被删除，并且 **addMessageBodyReader()** 和 **addMessageBodyWriter()** 方法受保护。

您现在应使用 **registerProvider （）** 和 **registerProviderInstance （）** 方法。

从 RESTEasy 3 中删除额外的类
@org.jboss.resteasy.annotations.cache.ServerCached 注释指定对 Jakarta REST 方法的响应，应从 RESTEasy 3 中删除，且必须从应用程序代码中删除。

7.4.3. 其他 RESTEasy 更改

本节提供有关 JBoss EAP 的 RESTEasy 中一些其他更改的信息。

SignedInput 和 SignedOuput

- **resteasy-crypto** 的 **SignedInput** 和 **SignedOutput** 必须在 **Request** 或 **Response** 对象中将 **Content-Type** 设置为 **multipart/signed**，或使用 **@Consumes** 或 **@Produces** 注解。
- 您可以通过在 **@Produces** 或 **@Consumes** 注解中设置该类型，使用 **SignedOutput** 和 **SignedInput** 以二进制形式返回 **application/pkcs7-signature** MIME 类型格式。
- 如果 **@Produces** 或 **@Consumes** 是 **text/plain** MIME 类型，则 **SignedOutput** 将采用 base64 编码并作为 String 发送。

安全过滤器
@RolesAllowed、**@PermitAll** 和 **@DenyAll** 的安全过滤器现在返回 "403 Forbidden" 而不是 "401 Unauthorized"。

客户端过滤器
从 RESTEasy 3.0 之前，从版本使用 RESTEasy 客户端 API 时，在 JAX-RS 2.0 中引入的客户端侧过滤器不会绑定并运行。

异步 HTTP 支持
*由于 JAX-RS 2.0 规范增加了使用 **@Suspended** 注释和 **AsynResponse** 接口的异步 HTTP 支持，因此用于异步 HTTP 的 RESTEasy 专有 API 已被弃用，并可能在以后的 RESTEasy 发行版本中删除。异步 Tomcat 和异步 JBoss Web 模块也已从服务器安装中删除。如果您不使用 Servlet 3.0 容器或更高，则异步 HTTP 服务器端处理将模拟，并在同一个请求线程中异步运行。*

服务器端缓存
服务器端缓存设置已更改。如需更多信息，请参阅 RESTEasy 文档。

YAML 供应商设置更改
*在以前的 JBoss EAP 版本中，RESTEasy YAML 供应商设置被默认启用。这在 JBoss EAP 7 中有所改变。现在默认禁用 YAML 供应商。由于 RESTEasy 用于 unmarshalling 的 **SnakeYAML** 库中的安全问题，它不被支持，因此必须在应用中明确启用。有关如何在应用中启用 YAML 提供程序并添加 Maven 依赖项的信息，请参阅 JBoss EAP 7.4 开发 Web Services Applications 中的 YAML 提供程序。*

Content-Type Header 中的默认 Charset UTF-8
*自 JBoss EAP 7.1 起，**resteasy.add.charset** 参数默认设置为 **true**。当资源方法返回一个 **text/*** 或 **application/xml*** 媒介类型且没有明确的字符集时，如果您不希望 RESTEasy 将 **charset=UTF-8** 添加到返回的 content-type 头时，可以将 **resteasy.add.charset** 参数设置为 **false**。*

如需有关文本媒体类型和字符集的更多信息，请参阅 JBoss EAP 7.4 开发 Web 服务应用程序 中的文本介质类型和 字符集。

SerializableProvider
*从不受信任的源中反序列化 Java 对象是不安全的。因此，从 JBoss EAP 7 开始，默认禁用 **org.jboss.resteasy.plugins.providers.SerializableProvider** 类，我们不推荐使用此提供程序。*

将请求与资源方法匹配
在 RESTEasy 3 中，对匹配规则的实施进行了改进和更正，如 JAX-RS 规范中定义的。特别是，对子资源方法和子资源 locator 的模糊 URI 进行了更改。

在 RESTEasy 2 中，子资源 locator 可以成功执行，即使存在具有相同 URI 的另一个子资源。这个行为根据规格不正确。

*在 RESTEasy 3 中，当子资源和子资源 locator 有模糊的 URI 时，调用子资源将成功；但是，调用子资源 locator 将会导致 HTTP 状态 **405 方法 Not Allowed** 错误。*

*以下示例包含子资源方法和子资源 locator 上的模糊 **@Path** 注释。请注意，对两个端点的 URI（**anotherResource** 和 **anotherResourceLocator**）都相同。两个端点之间的区别在于 **anotherResource** 方法与 REST 动词 **POST** 关联。**anotherResourceLocator** 方法不与任何 REST 动词关联。根据规范，将始终选择具有 REST 动词的端点（本例中为 **anotherResource** 方法）。*

```
@Path("myResource")
public class ExampleSubResources {
   @POST
   @Path("items")
   @Produces("text/plain")
   public Response anotherResource(String text) {
      return Response.ok("ok").build();
   }

   @Path("items")
   @Produces("text/plain")
   public SubResource anotherResourceLocator() {
```

```
        return new SubResource();
    }
}
```

7.4.4. RESTEasy SPI 更改

JBoss EAP 8 中删除了 RESTEasy SPI 提供程序。

SPI Exceptions
所有 SPI 失败例外都已弃用，不再在内部使用。它们已被对应的 Jakarta REST 异常替代。

弃用的例外	jaxrs-api 模块中的替换例外
org.jboss.resteasy.spi.ForbiddenException	jakarta.ws.rs.ForbiddenException
org.jboss.resteasy.spi.MethodNotAllowedException	jakarta.ws.rs.NotAllowedException
org.jboss.resteasy.spi.NotAcceptableException	jakarta.ws.rs.NotAcceptableException
org.jboss.resteasy.spi.NotFoundException	jakarta.ws.rs.NotFoundException
org.jboss.resteasy.spi.UnauthorizedException	jakarta.ws.rs.NotAuthorizedException
org.jboss.resteasy.spi.UnsupportedMediaTypeException	jakarta.ws.rs.NotSupportedException

InjectoronnectionFactoryy 和 Registry
InjectorFactory 和 **Registry** SPI 已更改。如果使用 RESTEasy 作为文档和支持，这不应有问题。

7.4.5. Jackson 供应商更改

JBoss EAP 6.4 中包含的 Jackson 版本已更改。从 JBoss EAP 7 开始，Jackson 提供程序已从 **resteasy-jackson-provider** 更改为 **resteasy-jackson2-provider**。

升级到 **resteasy-jackson2-provider** 需要一些软件包更改。例如，Jackson 注解软件包已从 **org.codehaus.jackson.annotate** 改为 **com.fasterxml.jackson.annotation**。

7.4.6. Spring RESTEasy 集成更改

JBoss EAP 8.0 支持 RESTEasy 6.2。如果您计划将 Spring 6.0 框架与 JBoss EAP 8.0 搭配使用，则必须使用 Java 17。

Spring 4.0 框架引入了对 Java 8 的支持。如果您计划将 RESTEasy 3.x 与 Spring 集成，请确保将 4.2.x 指定为部署中的最低 Spring 版本，因为这是 JBoss EAP 7 支持的最早稳定版本。

7.4.7. RESTEasy Jettison JSON 提供程序更改

自 JBoss EAP 7 起，RESTEasy Jettison JSON 提供程序已被弃用，默认情况下不再添加到部署中。建议您切换到推荐的 RESTEasy Jackson 提供程序。如果您希望继续使用 Jettison 提供程序，您必须在 **jboss-deployment-descriptor.xml** 文件中定义显式依赖项，如下例所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="org.jboss.resteasy.resteasy-jackson2-provider"/>
      <module name="org.jboss.resteasy.resteasy-jackson-provider"/>
    </exclusions>
    <dependencies>
      <module name="org.jboss.resteasy.resteasy-jettison-provider" services="import"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

有关如何定义显式依赖项的更多信息，请参阅 JBoss EAP 7.4 开发指南中的 将显式模块依赖部署到部署。

7.4.8. 用于 JBoss EAP 的 MicroProfile

MicroProfile 是可供开发人员用来将应用和微服务配置为在多个环境中运行的规范名称，而无需修改或重新打包这些应用程序。在以前的版本中，MicroProfile 作为技术预览提供 JBoss EAP 7.3，但自此后已被删除。MicroProfile 现在仅适用于 JBoss EAP XP。

其他资源

- 了解 MicroProfile

7.5. CDI 应用程序更改

JBoss EAP 8.0 包括对 CDI 4.0 的支持。因此，使用旧 CDI 版本编写的应用程序可能会在迁移到 JBoss EAP 8.0 时看到一些行为更改。本节仅总结了其中的一些变化。

有关 Weld 和 CDI 4.0 的更多信息，请参阅：

- Jakarta 上下文依赖注入 4.0
- Weld 5.1.1.Final - CDI 参考实施

7.5.1. Bean 归档

已启用 Bean 的 Bean 类必须在 bean 存档中部署，以确保它们由 CDI 发现，并作为 Bean 进行处理。

CDI 1.1 引入了隐式 bean 归档，存档包含一个或多个带有定义注解的 bean 类，或者一个或多个 session Bean。隐式 Bean 归档由 CDI 扫描，在类型发现期间只发现带有 bean 定义注解的类。如需更多信息，请参阅 JSR 365: 上下文和依赖注入 Java™ 2.0 中的 Type 和 Bean Discovery。Jakarta 等同于 bean 定义注解，在 Jakarta 上下文依赖注入 2.0 规格中定义。

在 CDI 4.0 中：

- 归档不会区分 bean.xml 是否具有版本号。
- 除了构建兼容的扩展外，存档还包含没有 beans.xml 文件的存档。构建兼容扩展不是 bean 归档。
- 带有空 beans.xml 文件的存档的默认发现模式被设置为 **annotated** 而不是 **all**。例如，如果 **beans.xml** 文件为空，则它是一个隐式 bean 归档而不是显式 bean 归档。
- 在这两种情况下，bean 发现元素在带有和不使用 **beans.xml** 文件的存档之间不受影响。

有关 CDI 4.0 的更多信息，请参阅 Jakarta 上下文和依赖注入 4.0。

bean 归档具有 **all**, **annotated** 或 **none** 的 bean 发现模式。包含非空 Bean.xml 的 bean 归档必须指定 bean-discovery-mode 属性。属性的默认值被 **annotated**。

在以下情况下，存档不是 bean 归档：

- 它包含一个 **Bean.xml** 文件，**bean-discovery-mode** 为 **none**。
- 它包含可移植扩展或构建兼容扩展，且没有 **Bean.xml** 文件。

归档是 在以下情况下明确的 bean 归档：

- 归档包含一个 **beans.xml** 文件，并且 **bean-discovery-mode** 都为。

在以下情况下，存档是一个 隐式 bean 归档：

- 归档包含一个空的 beans.xml 文件。
- 存档包含一个或多个 bean 类，它们带有 bean 定义注解或一个或多个会话 Bean，即使它不包含 **Bean.xml** 文件。

CDI 1.2 限于定义以下注解 ：

- **@ApplicationScoped**,**@SessionScoped**,**@ConversationScoped**, 和 **@RequestScoped** 注释
- 所有其他常规范围类型
- **@Interceptor** 和 **@Decorator** 注解
- 所有 stereotype 注释，注释为 **@Stereotype**
- **@Dependent** scope 注释

7.5.2. 阐明对话解析的问题

CDI 1.2 中更改了对话上下文生命周期，以防止与 Servlet 规范冲突，如 CDI 规范问题 CDI-411 所述。对话范围在所有 servlet 请求期间处于活跃状态，不应阻止其他 servlet 或 servlet 过滤器设置请求正文或字符编码。如需更多信息，请参阅 Jakarta EE 中的对话上下文生命周期。

7.5.3. Observer 解析

在 CDI 1.2 中，事件解析部分被重写。在 CDI 1.0 中，如果观察器方法具有所有事件限定符，则会向观察器发送事件。在 CDI 1.2 中，如果观察器方法没有事件限定符或事件限定符的子集，则会向观察者发送事件。如需更多信息，请参阅 Observer 解析。

7.6. HTTP 会话 ID 更改

request.getSession （）.getId （） 调用返回的字符串，以获取分配给 HTTP 会话的唯一标识符，在 JBoss EAP 6.4 和 JBoss EAP 7 之间有所变化。

JBoss EAP 6.4 以 **session-id.instance-id** 格式返回会话 ID 和实例 ID。

JBoss EAP 7 和 EAP 8 仅返回会话 ID。

对于从 JBoss EAP 6 到 JBoss EAP 8 的一些升级，这个更改可能会造成无路由 Cookie 的问题。如果您的应用根据此方法调用的返回值重新创建 JSESSIONID cookies，您可能需要更新应用程序代码以提供所需的行为。

7.7. 迁移显式模块依赖项

在之前的 JBoss EAP 版本中引入模块化类加载系统和 JBoss 模块，允许对应用程序可用的类进行精细控制。此功能允许您使用应用程序的 **MANIFEST.MF** 文件或 **jboss-deployment-structure.xml** 部署描述符文件来配置显式模块依赖项。

如果您在应用程序中定义了显式模块依赖项，您应该了解 JBoss EAP 7 中的以下更改。

查看可用性依赖项
JBoss EAP 中包含的模块已更改。当您将应用程序迁移到 JBoss EAP 7 时，请查看 **MANIFEST.MF** 和 **jboss-deployment-structure.xml** 文件条目，以确保它们不引用在此发行版本中删除或已弃用的任何模块。

需要扫描的依赖关系
在之前的 JBoss EAP 版本中，如果您的依赖项包含在注解扫描期间被处理所需的注释，如声明 EJB 拦截器时，您需要在新 JAR 文件中生成并包括 Jandex 索引，然后在 **MANIFEST.MF** 或 **jboss-deployment-structure.xml** 部署描述符文件中设置标记。

JBoss EAP 7 现在为静态模块提供自动运行时的注解索引，因此您无需手动生成它们。但是，您仍然需要在应用程序的 **MANIFEST.MF** 文件或 **jboss-deployment-structure.xml** 部署描述符文件中添加 **annotations** 标志，如下所示。

示例：在 MANIFEST.MF 文件中注解标记

```
Dependencies: com.company.my-ejb annotations, com.company.other
```

示例：在 jboss-deployment-structure.xml 文件中注解标记

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.company.my-ejb" annotations="true"/>
      <module name="com.company.other"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

7.8. HIBERNATE 更改

JBoss EAP 8 包括对 Hibernate ORM 6.2 的支持，这是 Java 编程语言的对象关系映射工具。有关 Hibernate ORM 6.2 文档的更多信息，请参阅 Hibernate ORM 6.2。

从 JBoss EAP 7.4 迁移到 JBoss EAP 8.0 时，请参考 Hibernate ORM 版本的特定 Hibernate ORM 迁移文档。

- 要从 JBoss EAP 7.4 迁移到 JBoss EAP 8，您必须完成以下步骤。
  - 从 Hibernate ORM 5.3 迁移到 5.4
  - 从 Hibernate ORM 5.4 迁移到 5.5

- 从 Hibernate ORM 5.5 迁移到 5.6
- 从 Hibernate ORM 5.6 迁移到 6.0
- 从 Hibernate ORM 6.0 迁移到 6.1
- 从 Hibernate ORM 6.1 迁移到 6.2
- Hibernate ORM dialects
- 弃用的 Hibernate ORM 类
- 推出 Hibernate ORM 类
- Hibernate ORM 内部

- 要从 JBoss EAP 和 Hibernate 的旧版本迁移，您必须完成以下步骤。
  - 从 Hibernate ORM 4.3 迁移到 Hibernate ORM 5.0
  - 从 Hibernate ORM 5.0 迁移到 Hibernate ORM 5.1
  - 从 Hibernate ORM 5.1 和 Hibernate ORM 5.2 迁移到 Hibernate ORM 5.3

其他资源

- Hibernate ORM 6.2 Dialects
- 弃用的 Hibernate ORM 类
- 推出 Hibernate ORM 类
- Hibernate ORM 内部

7.8.1. 从 Hibernate ORM 5.3 迁移到 5.4

本节重点介绍从 Hibernate ORM 版本 5.3 迁移到 5.4 时所需的更改。有关 Hibernate ORM 5.3 和 Hibernate ORM 5.4 之间实施的更改的更多信息，请参阅 Hibernate ORM 5.4 迁移指南。

已知的更改
下面描述了从 Hibernate ORM 版本 5.3 迁移到 5.4 时的一些变化。

7.8.1.1. 覆盖延迟身份管理行为

- 在 Hibernate 5.3 中，为基于 **FlushMode** 或 **FlushModeType** 值的 **DelayedPostInsertIdentifier** 行为提供支持，以缩短增强 **扩展 PersistenceContext** 支持。不幸的是，此更改中包含了一些问题。
- 在 Hibernate 5.4 中，它决定保留尽可能多的 Hibernate 5.3 行为，并且只为所选用例恢复非常具体的 **DelayedPostInsertIdentifier** 行为。
- 为了让 Hibernate 5.4 更灵活，需要将配置选项用作完全禁用 Hibernate 5.3 行为的临时解决方案，将其还原回 Hibernate 5.2 及更早版本。

7.8.1.2. SQL Server JDBC 驱动程序版本升级到至少 6.1.2

由于修复 HHH-12973，您必须将 JDBC 驱动程序版本升级到 6.1.2。因此，在没有关闭数据库连接的情况下，SQL Server JDBC 驱动程序的旧版本无法内省 **INFORMATION_SCHEMA.SEQUENCES**。

7.8.2. 从 Hibernate ORM 5.4 迁移到 5.5

本节重点介绍从 Hibernate ORM 版本 5.4 迁移到 5.5 时所需的更改。有关 Hibernate ORM 5.4 和 Hibernate ORM 5.5 之间实施的更改的更多信息，请参阅 Hibernate ORM 5.5 迁移指南。

已知的更改

Hibernate ORM 5.5 版本与 Hibernate ORM 5.4 类似，因为它包括应用到 5.4 维护版本的所有 bug 修复，并引入对 Jakarta Persistence API 的支持。

7.8.2.1. 基于 Dom4J 的 XML 映射

Hibernate 的 XML 映射定义的实施基于 JAXB 而不是 DOM4J 进行重组，以确保删除此依赖的持续进度。

7.8.2.2. 删除了禁用 "enhanced 代理" 的功能

"增强型代理"功能已作为 Hibernate 5.3 的可选性能改进功能。此功能现已永久启用。

7.8.3. 从 Hibernate ORM 5.5 迁移到 5.6

本节重点介绍从 Hibernate ORM 版本 5.5 迁移到 5.6 时所需的更改。有关 Hibernate ORM 5.5 和 Hibernate ORM 5.6 之间实施的更改的更多信息，请参阅 Hibernate ORM 5.6 迁移指南。

已弃用的功能

Hibernate 5.6 版本与之前的 Hibernate 5.5 版本非常相似，但从之前的 Hibernate 版本中删除一些已弃用的功能除外。

7.8.3.1. 删除 Javassist

您无法选择 **javassist** 作为用于实体字节代码增强的实施。**byte Buddy** 是默认值，在一段时间后，javassist 已被弃用，现在被删除。这对应用程序没有任何功能影响；唯一的例外是，它不再可用于配置 **hibernate.bytecode.provider=javassist** 属性。如果您使用这个 **feautre**，可以删除此属性。这可能导致 **Hibenate ORM** 不再在其依赖项中列出 **javassist** 的问题。

7.8.4. 从 Hibernate ORM 5.6 迁移到 6.0

本节重点介绍从 **Hibernate ORM** 版本 **5.6** 迁移到 **6.0** 时所需的更改。有关 **Hibernate ORM 5.6** 和 **Hibernate ORM 6.0** 之间实施的更改的更多信息，请参阅 **Hibernate ORM 6.0 迁移指南**。

Hibernate 6.0 发行版本包括以下更改：

- **Java 11 是 Hibernate 6.0 的最低兼容基准版本。**
- **Jakarta Persistence：Hibernate ORM 6.0** 发行版中的另一项重要更改包括从 **Java Persistence**（如 **Java EE specs** 定义）移到 **Jakarta Persistence**，如 **Jakarta EE spec** 定

义。此更改产生的最重要的影响包括使用 Jakarata Persistence 类 jakarta.persistence prerequisites 而不是 Java Persistence ones javax.persistence prerequisites。

- 从 JDBC 读取： Hibernate ORM 6.0 版本开发的另一个原因是从 JDBC ResultSet 中按名称(read-by-name)读取结果的另一方面，以根据位置（读取）读取结果。进行了此更改，通过实施吞吐量测试来提高扩展。

- 生成的 SQL：此功能会导致以下改进：

  - 不再生成列别名

  - 列引用是"unique-d"。

  - 更好地定义并更好地确定不必要的加入（次要表、继承表）

- 标识符为 Object - 早期版本的 Hibernate 需要所有标识符类型都实现 Serializable，Hibernate 6.0 已被删除，因为标识符可以是任何 对象。这个更改会影响之前使用 Serializable 定义的许多 API 和 SPI 方法。

- @IdGeneratorType：在这个发行版本中，您可以使用 @IdGeneratorType 注释来获得更好的 type-safe 方法来定义标识符生成的自定义生成器。

- 隐式标识符序列和表名称 ：在 Hibernate 6.0 中修改了与标识符生成关联的序列和表的隐式名称，这可能会影响用户迁移应用程序。在本发行版本中，Hibernate 会为每个实体层次结构创建一个序列，而不是默认单个序列 hibernate_sequence。

- 隐式序列生成器的默认值：Implicit 序列，如前面 hibernate_sequence，现在遵循 JPA @SequenceGenerator 注释的默认值，这意味着序列的分配大小为 50。

- 类型 system: 因为 Hibernate 6.0 是一个主要版本，另一个重要的改变是修改 Hibernate 的映射注解，并使它们更适合类型安全。这个功能已决定在此发行版本中提供，因为与类型相关的合同已更改。

- query：在 Query 的功能中引入了很多更改。查询功能，如迁移到专用树结构以模型 HQL 和标准查询，改进了批量 SQM DML 语句（如 insert、update 和 delete）的实现，以及更改

hibernate.criteria.copy_tree 属性的行为，并包括了传递令牌。

- *#onSave 方法的签名更改：AonSave 方法的签名已从保存 (Object 实体, Serializable id, Object[] state, String[] propertyNames, String[] type) 改为 boolean onSave (Object entity, Object id, Object[] state, String[] propertyNames, Type[] type)，将预期标识符类型从 Serializ 改为 Object。*

- *获取循环确定：以前的 Hibernate 版本确定使用深度优先方法获取，偶尔会导致奇数的"循环"确定。从 Hibernate 6.0 开始，现在首先使用宽度首先获取确定。*

- *重组 org.hibernate.loader： loader.collection 软件包的内容被重构为 loader.ast.spi 和 loader.ast.internal，并适应 SQM API。*

- *重组 SQL 软件包： sql.ordering 的内容被移到 metamodel.mapping.ordering.ast。*

- *hbm.xml 映射的弃用： Legacy hbm.xml 映射格式已弃用，并将不再支持 6.x。*

- *lazy 关联广告：在 Hibernate 6.0 之前，使用 fetch="join" 或 @Fetch (FetchMode.JOIN) 的 lazy 关联 在通过 id i.e 加载时被视为 eager。从 Hibernate 6.0 开始，无论获取机制如何，这些关联都被正确遵守。通过指定 lazy="false" 或 @ManyToOne (fetch = EAGER)/@OneToOne (fetch = EAGER)/@OneToMany (fetch = EAGER)/@ManyToMany (fetch = EAGER)来实现向后兼容。*

- *对于 hbm.xml <return-join/> 的行为变化：根据 Hibernate 6.0， &lt ;return-join /> 会导致获取关联，而不是添加选择项目。*

有关这些功能的更多信息，请参阅 Hibernate ORM 6.0 迁移指南。

Hibernate 6.0 发行版本还包含从以前的 Hibernate 版本中删除的许多功能，如下所示：

- *现在不允许使用 hbm.xml 多个 < column /> - 在 6.0 中删除了对带有多个列的基本属性映射的支持。组件类属性现在支持正确解释 CompositeUserType 类。*

- *传统 Hibernate 标准 API - Hibernate 6.0 中已被弃用的传统 Hibernate 标准 API 已在 Hibernate 6.0 中删除。*

- 通过 NativeQuery 的调用 - 使用 NativeQuery 调用 SQL 功能和程序不再被支持。使用 org.hibernate.procedure.ProcedureCall 或 jakarta.persistence.StoredProcedureQuery 等方法。

- HQL 获取所有 properties 子句 - 获取所有 properties 子句已从 HQL 语言中删除。

- JMX 集成 - Hibernate 不再为将其与 JMX 环境集成提供内置支持。

- JACC 集成 - Hibernate 不再为将其与 JACC 环境集成提供内置支持。

有关 Hibernate 6.0 中删除的功能的更多信息，请参阅 Hibernate ORM 6.0 迁移指南。

7.8.5. 从 Hibernate ORM 6.0 迁移到 6.1

本节重点介绍从 Hibernate ORM 版本 6.0 迁移到 6.1 所需的更改。有关 Hibernate ORM 6.0 和 Hibernate ORM 6.1 间实施的更改的更多信息，请参阅 Hibernate ORM 6.1 迁移指南。

Hibernate 6.1 发行版本包含以下更改：

- 基本数组：除 byte[]/Byte[] 和 char[]/Character[] 以外的基本数组，基本集合（仅子类型集合）映射到类型代码 SqlTypes.ARRAY，它映射到由新方法 getArrayTypeName 决定的 SQL 标准数组类型，并支持 org.hibernate.dialect.Dialect.Dialect。

- Enum 映射更改：现在默认映射到类型代码 SqlType.SMALLINT，就像它映射到 TINYINT 之前。这个映射不正确，因为 Java 有效地允许 32K 枚举条目，但 TINYINT 只是一个 1 字节类型。

有关 Hibernate 6.1 中包含的功能的详细信息，请参阅 Hibernate ORM 6.1 迁移指南。

7.8.6. 从 Hibernate ORM 6.1 迁移到 6.2

本节重点介绍从 Hibernate ORM 版本 6.1 迁移到 6.2 所需的更改。有关 Hibernate ORM 6.1 和 Hibernate ORM 6.2 之间实现的更改的更多信息，请参阅 Hibernate ORM 6.2 迁移指南。

Hibernate 6.2 发行版本包含以下改进，如下所示：

DDL 类型更改：

- OffsetTime 映射更改 - 在此发行版本中，OffsetTime 依赖于 @TimeZoneStorage 和 hibernate.timezone.default_storage 设置。由于默认设置为 TimeZoneStorageType.DEFAULT，这意味着此类列的 DDL 期望已更改。
- MariaDB 上的 UUID 映射更改 - 在 MariaDB 上，类型代码 SqlTypes.UUID 默认引用 DDL 类型 uuid，与使用 binary (16) 的位置相比。由于这个变化，在现有数据库中可能会出现模式验证错误。
- SQL Server - On SQL Server 上的 UUID 映射更改，类型代码 SqlTypes.UUID 默认指的是 DDL 类型 唯一identifier，与使用 二进制(16) 的位置相比。由于这个变化，在现有数据库中可能会出现模式验证错误。
- 与使用 clob 之前相比，Oracle 12.1+ 上的 JSON 映射更改，类型代码 SqlTypes.JSON 默认指的是 DDL 类型 blob 和 21+ 到 json。由于这个变化，在现有数据库中可能会出现模式验证错误。
- H2 上的 JSON 映射更改 - On H2 1.4.200+，类型代码 SqlTypes.JSON 默认引用 DDL 类型 JSON，与使用 clob 的位置相比。由于这个变化，在现有数据库中可能会出现模式验证错误。
- enums 的 datatype - 从 Hibernate 6.2 开始，选择隐式 SQL datatype 以存储枚举类上定义的条目数量敏感。
- timezone 和 offset 存储 - hibernate.timezone.default_storage 现在默认为 DEFAULT。
- byte[]/Character[] 映射更改 - Hibernate 6.2 使其可配置，以处理域模型中的 Byte[] 和 Character[] 映射更改的映射。
- 用于可选一对一映射的 UNIQUE 约束 - Hibernate 的 Earlier 版本没有在数据库中为标记为可选 的逻辑一对一关联创建一个 UNIQUE 约束。从 Hibernate 6.2 开始，这些 UNIQUE 约束现

已创建。

- Oracle 上的原生 SQL 查询的列类型(n,0) - Since Hibernate 6.0，根据精度，在 Oracle 上带有 scale 0 的列 type ,tinyint,smallint , int, 或 bigint。现在，根据精度，带有 scale 0 的类型为 number 的列被解释为 int 或 bigint。

- 删除了对旧数据库版本的支持 - Hibernate 6.2 引入了对 Hibernate 支持的大多数数据库版本的最低支持数据库版本的概念。

- CDI 处理的更改 - 当 CDI 可用并配置了 CDI 时，Hibernate 可以使用 CDI BeanManager 解析各种 Bean 引用。从 Hibernate 6.2 开始，只有在 hibernate.cdi.extensions 设为 true 时，这些扩展才会从 CDI BeanManager 解析。

- 更改默认和弃用 - enableLazyInitialization 和 enableDirtyTracking 增强工具选项、全局属性 hibernate.bytecode.use_reflection_optimizer 以及相应的 hibernate.enhancer.enableLazyInitialization 和 hibernate.enhancer.enableDirtyTracking 配置设置，将其默认值切换到 true，这些设置现已弃用。

- org.hibernate.cfg 和 org.hibernate.loader 的软件包更新： org.hibernate.cfg 和 org.hibernate.loader 软件包已更新，以明确显示被视为 API、SPI 和内部的合同之间的区别。

- 集成合同(SPI)的变化 - 在 Hibernate ORM 6.2 的开发过程中，以下 SPIs 已被修改： EntityPersister#lock,EntityPersister#multiLoad,Executable#afterDeserialize, 和 JdbcType#getJdbcRecommendedJavaTypeMapping （）。

- 查询路径比较 ：根据 Hibernate 6.2，在早期检查的路径比较。

- batch Fetching 和 LockMode - 当 LockMode 大于 READ 时，Hibernate 不会执行批处理获取，因此不会初始化现有的未初始化代理。这是因为锁定模式与批处理获取队列中的一个代理不同。

7.8.7. 从 Hibernate ORM 4.3 迁移到 Hibernate ORM 5.0

JBoss EAP 7.0 包括 Hibernate ORM 5.0。本节重点介绍从 Hibernate ORM 版本 4.3 迁移到版本 5 时所需的更改。有关 Hibernate ORM 4 和 Hibernate ORM 5 之间实施的更改的更多信息，请参阅 Hibernate ORM 5.0 迁移指南。

删除和弃用的类

以下已弃用的类已从 Hibernate ORM 5 中删除：

- *org.hibernate.cfg.AnnotationConfiguration*
- *org.hibernate.id.TableGenerator*
- *org.hibernate.id.TableHiLoGenerator*
- *org.hibernate.id.SequenceGenerator*

对类和软件包的其他更改

- *org.hibernate.integrator.spi.Integrator 接口已针对重新设计的 bootstrap 进行了相应的改变。*
- *创建了新的软件包 org.hibernate.engine.jdbc.env.spi。它包含 org.hibernate.engine.jdbc.env.spi.JdbcEnvironment 接口，该接口是从 org.hibernate.engine.jdbc.spi.JdbcServices 接口中提取的。*
- *引入了一个新的 org.hibernate.boot.model. relational.ExportableProducer 接口，它将影响 org.hibernate.id.PersistentIdentIdentifierGenerator 实现。*
- *org.hibernate.id.Configurable 的签名被修改为接受 org.hibernate.service.ServiceRegistry 而不是 org.hibernate.dialect.Dialect。*
- *org.hibernate.metamodel.spi.TypeContributor 接口已迁移到 org.hibernate.boot.model.TypeContributor。*
- *org.hibernate.metamodel.spi.TypeContributions 接口已迁移到 org.hibernate.boot.model.TypeContributions。*

类型处理

-

内置 org.hibernate.type.descriptor.sql.SqlTypeDescriptor 实现不再使用 org.hibernate.type.descriptor.sql.SqlTypeDescriptorRegistry 自动注册。使用自定义 SqlTypeDescriptor 实现来扩展内置实现并依赖于该行为的应用程序必须更新，以调用 SqlTypeDescriptorRegistry.addDescriptor （） 本身。

- 对于定义为生成的 UUID 的 ID，一些数据库需要您明确设置 @Column (length=16) 以便生成 BINARY (16)，以便比较可以正常工作。

- 对于 hbm.xml 中定义的 EnumType 映射，您希望 javax.persistence.EnumType.STRING name-mapping，必须使用 useNamed (true) 设置或指定 12 的 VARCHAR 设置来显式声明此配置。

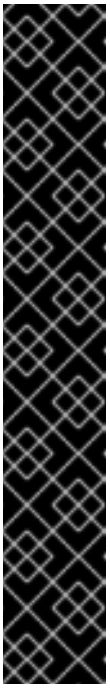
事务管理

- 事务下的 SPI 在 Hibernate ORM 5 中进行了重大重新设计。在 Hibernate ORM 4.3 中，您使用 org.hibernate.Transaction API 来直接访问不同的后端事务策略。Hibernate ORM 5 引入了一定程度的间接性。在后端， org.hibernate.Transaction 实现现在与 org.hibernate.resource.transaction.TransactionCoordinator 对话，它代表基于后端策略给定会话的事务上下文。虽然这对开发人员没有直接影响，但可能会影响 bootstrap 配置。之前的应用程序会指定 hibernate.transaction.factory_class 属性，它现已被弃用，并指代一个 org.hibernate.engine.transaction.spi.TransactionFactory FQN（完全限定名称）。使用 Hibernate ORM 5 时，您可以指定 hibernate.transaction.coordinator_class 设置，并引用 org.hibernate.resource.transaction.TransactionCoordinatorBuilder。详情请查看 org.hibernate.cfg.AvailableSettings.TRANSACTION_COORDINATOR_STRATEGY。

- 现在可识别以下短名称：

  - JDBC ：使用 JDBC java.sql.Connection 管理事务。这是非 Jakarta Persistence 事务的默认值。

  - jta: 使用 Jakarta 交易管理事务。



重要

如果 Jakarta Persistence 应用没有为 hibernate.transaction.coordinator_class 属性提供设置，则 Hibernate 将自动根据持久性单元的事务类型构建正确的事务协调器。

如果非 Jakarta Persistence 应用程序不提供 hibernate.transaction.coordinator_class 属性的设置，则 Hibernate 将默认为 jdbc 管理事务。如果应用程序实际使用 Jakarta 事务，则此默认将引发问题。使用 Jakarta Transactions 的非 Jakarta Persistence 应用应明确将 hibernate.transaction.coordinator_class 属性值设置为 jta，或提供自定义 org.hibernate.resource.transaction.TransactionCoordinatorBuilder，它构建了一个 org.hibernate.resource.transaction.TransactionCoordinator，它与 Jakarta Transactions 正确协调。

其他 Hibernate ORM 5 更改

- cfg.xml 文件再次被完全解析，并与事件、安全和其他功能集成。
- 使用 EntityManagerFactory 从 cfg.xml 加载的属性没有之前使用 hibernate 的前缀名称。现在，已保持一致。
- 配置不再是连续的。
- org.hibernate.dialect.Dialect.getQuerySequencesString （） 方法现在检索目录、模式和递增值。
- AuditConfiguration 修饰符已从 org.hibernate.envers.boot.internal.EnversService 中删除。
- AuditStrategy 方法参数被修改为删除过时的 AuditConfiguration，并使用新的 EnversService。
- org.hibernate.hql.spi 软件包和子软件包中的各种类和接口已移到新的 org.hibernate.hql.spi.id 软件包中。这包括 MultiTableBulkIdStrategy 类和 AbstractTableBasedBulkIdHandler、TableBasedDeleteHandlerImpl、TableBasedUpdateHandlerImpl 接口及其子类。

- 对属性访问合同进行完全重新设计。

- 现在，使用 org.hibernate.cache.spi.access.cache.spi.access.AccessType .getExternalName （）方法而不是 org.hibernate.cache.spi.access.AccessType enum constants 来定义有效的 hibernate.cache.default_cache_concurrency_strategy 设置值。这与其他 Hibernate 设置一致。

7.8.8. 从 Hibernate ORM 5.0 迁移到 Hibernate ORM 5.1

JBoss EAP 7.1 包括 Hibernate ORM 5.1。本节重点介绍从 Hibernate ORM 版本 5.0 迁移到版本 5.1 所需的区别和所需更改。

Hibernate ORM 5.1 功能

Hibernate 的这个版本包括性能改进和程序错误修复。如需更多信息，请参阅 7.1.0 的 JBoss EAP 发行注记中的 Hibernate ORM 5.1 功能。有关在 Hibernate ORM 5.0 和 Hibernate ORM 5.1 之间实施的更改的更多信息，请参阅 Hibernate ORM 5.1 迁移指南。

模式管理工具更改
JBoss EAP 7 中的模式管理工具更改

Hibernate ORM 5.1 中的模式管理工具更改专注于以下区域：

- 统一处理 hbm2ddl.auto，并支持 hibernate 的 Jakarta Persistence 模式-generation。

- 从 SPI 中删除 JDBC 顾虑，以促进 Hibernate OGM（一种为 NoSQL 数据存储提供 Jakarta Persistence 支持）的持久性引擎。

模式管理工具更改只是直接使用以下类的应用程序的迁移问题：

- org.hibernate.tool.hbm2ddl.SchemaExport

- org.hibernate.tool.hbm2ddl.SchemaUpdate

- org.hibernate.tool.hbm2ddl.SchemaValidator

- org.hibernate.tool.schema.spi.SchemaManagementTool，或其委派

JBoss EAP 7.1 中的模式管理工具更改

Hibernate ORM 5.1.10 包含在 JBoss EAP 7.1 中，引入了检索数据库表的策略，用于改进 SchemaMigrator 和 SchemaValidator 性能。此策略执行单个 java.sql.DatabaseMetaData#getTables (String, String, String[]) 调用来确定每个 javax.persistence.Entity 是否具有映射的数据库表。这是默认策略，它使用 hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=grouped 属性设置。此策略可能要求提供 hibernate.default_schema 和/或 hibernate.default_catalog。

要使用执行 java.sql.DatabaseMetaData#getTables (String, String, String[]) 调用 的每个 javax.persistence.Entity 的旧策略，请使用 hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=indivivi ally 属性设置。

7.8.9. 从 Hibernate ORM 5.1 和 Hibernate ORM 5.2 迁移到 Hibernate ORM 5.3

JBoss EAP 7.4 包括 Hibernate ORM 5.3。本节重点介绍从 Hibernate ORM 5.1 迁移到 Hibernate ORM 5.2 时所需的一些更改，然后介绍 Hibernate ORM 5.3。

Hibernate ORM 5.2 功能

Hibernate ORM 5.2 使用 Java 8 JDK 构建，在运行时需要 Java 8 JRE。以下是本发行版本中进行的一些更改列表：

- hibernate-java8 模块已合并到 hibernate-core 中，Java 8 日期/时间数据类型现已原生支持。
- hibernate-entitymanager 模块合并到 hibernate-core 中。HibernateEntityManager 和 HibernateEntityManagerFactory 已被弃用。
- Session、StatelessSession 和 SessionFactory 类层次结构被重构为移除已弃用的类，并与 Jakarta Persistence Metamodel API 更好地保持一致。
- org.hibernate.persister 和 org.hibernate.tuple 软件包中的 SPIs 已更改。任何使用这些 SPI 的自定义类都需要检查和更新。
- LimitHandler 更改引入了一个新的 hibernate.legacy_limit_handler 设置，该设置默认设置为 false，它允许您启用传统的 Hibernate 4.3 限制处理程序行为。这会影响有限问题列表。

- 引入了一个新的检索数据库表的策略，改进了 SchemaMigrator 和 SchemaValidator 性能。

- 此发行版本更改了 String、字符[] 和 Character[] 属性的 CLOB 值，在使用 PostgreSQL81Dialect 及其子类时，会如何处理 @Lob。

- @TableGenerator 和 @SequenceGenerator 名称的范围已从 global 改为 local。

有关 Hibernate 5.2 中实施的更改的完整列表，请参阅 Hibernate ORM 5.2 迁移指南。

Hibernate ORM 5.3 功能

Hibernate ORM 5.3 添加了对 Jakarta Persistence 2.2 规范的支持。此发行版本包含遵守此规格的更改以及其他改进。以下是其中一些更改的列表：

- 对位置查询参数处理的更改会导致以下更改：
  - 在 HQL/JPQL 查询中删除了对 JDBC 样式参数声明的支持。
  - Jakarta Persistence 位置位置参数的行为与命名参数更为相似。
  - 原生查询中的 JDBC 样式参数声明使用基于一键的参数声明，而不是零参数绑定与 Jakarta Persistence 保持一致。您可以通过将 hibernate.query.sql.jdbc_style_params_base 属性设置为 true 来恢复到基于零的绑定。

- 为遵守 Jakarta Persistence 规格，由 @TableGener 存储的值存储的序列值是最后生成的值。在以前的版本中，Hibernate 存储下一个序列值。您可以使用 hibernate.id.generator.stored_last_used 属性启用传统的 Hibernate 行为。使用 @TableGenerator 和迁移到 Hibernate 5.3 的现有应用程序必须将 hibernate.id.generator.stored_last_used 配置 属性设置为 false。

- org.hibernate.query.QueryParameter 类中的 getType() 方法被重命名为 getHibernateType()。

- Hibernate 的第二个缓存 SPI 被重新设计，以更好地满足各种缓存供应商的要求。可在

HHH-11356 中找到详情。

- HHH-11356 的更改还需要更改消费者，这会影响 Hibernate Statistics 系统。

- 有些方法被临时添加到 org.hibernate.Query 类中，以便更轻松地将原生应用程序从 Hibernate ORM 5.1 迁移到 5.3，并维护 Hibernate 5.1 分页行为。这些方法已弃用，要使用休眠版本来可移植，应该更新应用程序以使用 Jakarta Persistence 方法。

- 对将 Infinispan 用作 Hibernate 第二级缓存提供程序的支持已移到 Infinispan 项目中。因此， hibernate-infinispan 模块已被丢弃。

- org.hibernate.tool.enhance.EnhancementTask Ant 任务的 API 已更改。使用 setBase （） 和 setDir （） 方法丢弃了 addFileset （） 方法。详情可在 HHH-11795 中找到。

- Hibernate 4.3 中引入了一个错误，导致嵌入的集合元素和复合 ID 中的多到一关联，即使在显式映射为 lazy 时也是如此。在 Hibernate 5.3.2 中，这个程序错误已被解决。因此，这些关联会根据其映射指定。详情可在 HHH-12687 中找到。

- 本发行版本中， Jakarta Persistence 和 Hibernate 事件监听器的原生实现。因此， JpaIntegrator 类已过时。扩展 org.hibernate.jpa.event.spi.JpaIntegrator 的类必须进行修改，以更改这些类以实施 org.hibernate.integrator.spi.Integrator 接口。可在 HHH-11264 中找到详细信息。

- org.hibernate.persister 软件包中的 SPI 已更改。任何使用这些 SPI 的自定义类都需要检查和更新。

有关 Hibernate 5.3 中所实施的其他更改的完整列表， 请参阅 Hibernate ORM 5.3 迁移指南。

Hibernate 5.1 和 Hibernate 5.3 之间的更改异常

在 Hibernate 5.2 和 5.3 中，使用 Hibernate 的原生 Bootstrap 构建的 SessionFactory 异常处理，按照 Jakarta Persistence 规格嵌套或转换 HibernateException。这个行为的唯一例外是在操作是特定于 Hibernate 时，如 Session.save （） 或 Session.saveOrUpdate （）。

在 Hibernate 5.3.3 中，添加了 hibernate.native_exception_handling_51_compliance 属性。此属性指示使用 Hibernate 的原生 bootstrap 构建的 SessionFactory 异常处理是否与 Hibernate ORM 5.1 中的原生异常处理行为相同。当设置为 true 时，会根据 Jakarta Persistence 规范，不会嵌套或转

换HibernateException。对于使用 Jakarta Persistence bootstrapping 构建的 SessionFactory，会忽略此设置。

兼容性转换器

JBoss EAP 7.4 包含一个兼容性转换器，它解决了 Hibernate ORM 5.3 API 方法，它们不再与 Hibernate ORM 5.1 兼容。转换程序是一种临时措施，允许使用 Hibernate ORM 5.1 构建的应用程序在 JBoss EAP 7.4 中表现出与 Hibernate 5.3 相同的行为。这是一个临时解决方案，您应该将这些方法调用替换为推荐的 Jakarta Persistence 方法调用。

您可以使用以下方法之一启用转换器：

- *您可以通过将 Hibernate51CompatibilityTransformer 系统属性设为 true 来为所有应用程序全局启用转换器。*
- *您可以使用 jboss-deployment-structure.xml 文件在应用程序级别启用转换程序。*

```
<jboss-deployment-structure>
  <deployment>
    <transformers>
      <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
    </transformers>
  </deployment>
  <sub-deployment name="main.war">
    <transformers>
      <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
    </transformers>
  </sub-deployment>
</jboss-deployment-structure>
```

下表列出了正在转换的 Hibernate 5.1 方法，并将其转换为 Hibernate 5.3 方法：

Hibernate 5.1 参考或方法	转换为 Hibernate 5.3 参考或方法
org.hibernate.BasicQueryContract.getFlushMode()	org.hibernate.BasicQueryContract.getHibernateFlushMode()
org.hibernate.Session.getFlushMode()	org.hibernate.Session.getHibernateFlushMode()
Enum org.hibernate.FlushMode.NEVER (0)	Enum org.hibernate.FlushMode.MANUAL (0)
org.hibernate.Query.getMaxResults()	org.hibernate.Query.getHibernateMaxResults()

Hibernate 5.1 参考或方法	转换为 Hibernate 5.3 参考或方法
org.hibernate.Query.setMaxResults(int)	org.hibernate.Query.setHibernateMaxResults(int)
org.hibernate.Query.getFirstResult(int)	org.hibernate.Query.getHibernateFirstResult()
org.hibernate.Query.setFirstResult(int)	org.hibernate.Query.setHibernateFirstResult(int)

7.9. HIBERNATE 搜索更改

JBoss EAP 7 中包含的 Hibernate Search 版本已更改。之前的 JBoss EAP 版本包括 Hibernate Search 4.6.x。Hibernate Search 5.5.x 中包括了 JBoss EAP 7。

Hibernate Search 5.5 基于 Apache Lucene 5.3.1 构建。如果您使用任何原生 Lucene API，请确保与此版本一致。Hibernate Search 5.5.8.Final 包装并隐藏版本 3 和版本 5 之间的许多 Lucene API 更改的复杂性；但是，一些类现已弃用、重命名或重新打包。本节描述了这些更改如何影响应用程序代码。

在 JBoss EAP 8.0 中，Hibernate Search 5 API 已被删除，并被 Hibernate Search 6 API 替代。

其他资源

- Hibernate 搜索更改

7.9.1. Hibernate Search 6 替换 Hibernate Search 5 API

Hibernate Search 5 API 已被删除，并被 JBoss EAP 8.0 中的 Hibernate Search 6 API 替换。

要查看删除的功能列表，请参阅 JBoss EAP 7.4 中的 Hibernate Search 5 API 已弃用，并在 EAP 8.0 中删除。

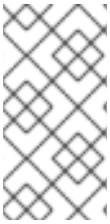
注意

Hibernate Search 6 API 与 Hibernate Search 5 API 向后兼容。您需要将您的应用程序迁移到 Hibernate Search 6。

JBoss EAP 8.0 中包含的 Hibernate Search 6 的最新版本是 6.2。如果您要从 Hibernate Search 5 迁移，您应该考虑迁移到 6.0、6.1 和 6.2 版本。

如需更多信息，请参阅以下迁移指南：

- 要从 Hibernate Search 5 迁移应用程序，请查看 Hibernate Search 6.0 迁移指南。
- 要将应用程序从 Hibernate Search 6.0 迁移到 6.1，请参阅 Hibernate Search 6.1 迁移指南。
- 要将应用程序从 Hibernate Search 6.1 迁移到 6.2，请参阅 Hibernate Search 6.2 迁移指南



注意

Hibernate Search 6.2 与 Hibernate ORM 6.2 兼容。如需更多信息，请参阅 Hibernate Search 6.2 参考文档中的 Hibernate ORM 6 部分。

7.9.2. Hibernate Search 6 支持 Elasticsearch

JBoss EAP 8.0 还支持在 Hibernate Search 6 中使用 Elasticsearch 后端，将数据索引到远程 Elasticsearch 或 OpenSearch 集群中。

要查看可能的 Hibernate 搜索架构和后端列表，请参阅 Table 2。构架的比较 中的 Hibernate Search 6.2 参考文档。

有关配置 Hibernate Search 6 的更多信息，请参阅 WildFly Developer 指南中的使用 Hibernate Search。

其他资源

- Hibernate 更改

7.10. 将实体 BEAN 迁移到 JAKARTA PERSISTENCE

Java EE 8 中对 Enterprise Java Beans 实体 Bean 的支持是可选的，从 JBoss EAP 7 开始不支持它们。

在以前的 JBoss EAP 版本中，实体 Bean 通过扩展 javax.ejb.EntityBean 类和实施必要的方法，在应用源代码中创建。然后，在 ejb-jar.xml 文件中配置它们。CMP entity bean 由一个 <entity> 元素来指定，该元素包含值为 Container 的 <persistence-type> 子元素。BMP entity bean 由一个 <entity> 元素来指定，其中包含一个 <persistence-type> 子元素，值为 Bean。

从 JBoss EAP 7 开始，您必须将代码中的任何 CMP 和 BMP 实体 Bean 替换为 Jakarta Persistence 实体。Jakarta Persistence 实体使用 jakarta.persistence prerequisites 类创建，并在 persistence.xml 文件中定义。

以下是 Jakarta Persistence 实体类的示例：

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
// User is a keyword in some SQL dialects!
@Table(name = "MyUsers")
public class MyUser {
    @Id
    @GeneratedValue
    private Long id;

    @Column(unique = true)
    private String username;
    private String firstName;
    private String lastName;

    public Long getId() {
        return id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
```

```
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

以下是 persistence.xml 文件的示例。

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
          https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
        version="3.0">
  <persistence-unit name="my-unique-persistence-unit-name">
    <properties>
      // properties...
    </properties>
  </persistence-unit>
</persistence>
```

有关 Jakarta Persistence 实体的工作示例，请参阅 JBoss EAP 8.0 中包含的 cmt quickstart。

7.11. JAKARTA PERSISTENCE 属性更改

本节介绍 JBoss EAP 7.0 和 7.1 中引入的 Jakarta Persistence 属性更改。

JBoss EAP 7.0 中引入的 Jakarta Persistence 属性更改

添加了一个新的持久性属性 jboss.as.jpa.deferdetach，以便与之前的 JBoss EAP 版本中的持久性行为兼容。

jboss.as.jpa.deferdetach 属性控制在每个实体 EntityManager 调用后，在非Jakarta 事务性环境中使用的事务范围持久性上下文以及是否等待持久性上下文关闭时（例如，会话 bean 调用结束时）。属性值默认为 false，即实体在各个 EntityManager 调用后被分离或清除。这是 Jakarta Persistence 规范中定义的正确默认行为。如果属性值设为 true，则实体在持久性上下文关闭前不会被分离。

在 JBoss EAP 5 中，持久性的行为如同 jboss.as.jpa.deferdetach 属性设为 true。要在将应用从 JBoss EAP 5 迁移到 JBoss EAP 7 时获得同样的行为，您必须在 persistence.xml 中将 jboss.as.jpa.deferdetach 属性值设置为 true，如下例所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
   <persistence-unit name="EAP5_COMPAT_PU">
   <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
   <properties>
       <property name="jboss.as.jpa.deferdetach" value="true" />
```

```
    </properties>
  </persistence-unit>
</persistence>
```

在 JBoss EAP 6 中，持久性的行为如同 jboss.as.jpa.deferdetach 属性设为 false。这与 JBoss EAP 7 中看到的行为相同，因此迁移应用程序时不需要任何更改。

JBoss EAP 7.1 中引入的 Jakarta Persistence 属性更改

在 JBoss EAP 7.0 中，未同步持久性上下文错误检查不严格，因为它应该在以下区域：

- 同步的容器管理的持久性上下文允许使用与 Jakarta 事务关联的未同步扩展持久性上下文。相反，应该已经抛出一个 IllegalStateException，以防止使用未同步持久性上下文。
- 部署描述符中指定的未同步持久性上下文被视为同步。

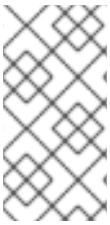
另外，JBoss EAP 7.0 中错误地忽略 @PersistenceContext 中的 PersistenceProperty 提示。

这些问题已在 JBoss EAP 7.1 及更高版本中解决和修复。由于这些更新可能会导致应用行为不需要更改，因此 JBoss EAP 7.1 中引入了两个新的持久性单元属性，以提供向后兼容性并保留之前的行为。

属性	描述
wildfly.jpa.skipmixedsynctypechecking	此属性禁用错误检查。它应该仅用作临时措施，以便在 JBoss EAP 7.0 中工作并在 JBoss EAP 7.1 及更高版本中出现故障时向后兼容。因为此属性可能在以后的发行版本中弃用，因此建议您尽快更正应用程序代码。
wildfly.jpa.allowjoinedunsync	此属性是 **wildfly.jpa.skipmixedsynctypechecking** 的替代选择。它允许应用程序对待与 Jakarta 事务关联的未同步持久性上下文，就像它们同步持久性上下文一样。

7.12. 迁移 JAKARTA ENTERPRISE BEANS 客户端代码

本节讨论 JBoss EAP 7.0 中的 Jakarta Enterprise Beans 客户端的更改。它还解释了如何修改客户端代码，以便在 JBoss EAP 7.0 中使用新的默认远程端口和连接器。此外，它描述了 JBoss EAP 7.1 和 JBoss EAP 7.0 中引入的所需的 JBoss EJB 客户端更改。



注意

从 JBoss EAP 7.0 开始，不支持企业实体 Bean。如需更多信息，请参阅将 实体 Bean 迁移到 Jakarta Persistence。

7.12.1. Jakarta Enterprise Beans 在 JBoss EAP 7 中的客户端变化

从 JBoss EAP 7 开始，默认的远程连接器和端口已更改。有关此更改的详情，请参阅更新 远程 URL 连接器和端口。

如果您使用 JBoss 服务器迁移工具迁移服务器配置，旧的设置会被保留，您不需要在此处进行详细更改。但是，如果您使用新的 JBoss EAP 8.0 默认配置，您必须进行以下更改。

7.12.1.1. 更新默认远程连接端口

在 jboss-ejb-client.properties 文件中，将远程连接端口值从 4447 更改为 8080。以下是上一个和当前版本中的 jboss-ejb-client.properties 文件的示例：

示例：JBoss EAP 6 jboss-ejb-client.properties 文件

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

示例：JBoss EAP 8 jboss-ejb-client.properties 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
```

7.12.1.2. 更新默认连接器

如果您使用新的 JBoss EAP 7 配置，则默认连接器已从 remote 改为 http-remoting。此更改会影响使用来自 JBoss EAP 的一个库的客户端连接到不同发行版中的服务器。

- 如果客户端应用程序使用 JBoss EAP 6 中的 Jakarta Enterprise Beans 客户端库并希望连接到 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 远程 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 客户端库并想连接到 JBoss EAP 6 服务器的客户端应用程序必须意识到服务器实例不使用 http-remoting 连接器，而是使用 远程 连接器。这可以通过定义新的客户端侧连接属性来实现。

  示例： 远程连接 属性

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

7.12.2. 迁移远程命名客户端代码

如果您使用新的默认 JBoss EAP 7 配置运行，您必须修改客户端代码以使用新的默认远程端口和连接器。

以下是如何在 JBoss EAP 6 中的客户端代码中指定远程命名属性的示例。

```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.provider.url=remote://localhost:4447
```

以下是如何在 JBoss EAP 7 中的客户端代码中指定远程命名属性的示例。

```
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory
java.naming.provider.url=http-remoting://localhost:8080
```

7.12.3. JBoss EAP 7.1 中引入的其他 JBoss EJB 客户端更改

JBoss EAP 7.0 包含在 JBoss Enterprise Java Beans 客户端 2.1.4 中，JBoss EAP 7.1 及更高版本包含在 JBoss Enterprise Java Beans 客户端 4.0.x 中，其中包括很多对 API 的更改。

注意

从 JBoss EAP 7 开始，不支持企业实体 Bean。有关如何将实体 Bean 迁移到 Jakarta Persistence 的详情，请参考 将实体 Beans 迁移到 Jakarta Persistence。

- *org.ejb.client.ejbClientInvocationContext 类添加以下新方法：*

方法	类型	描述
isBlockingCaller()	布尔值	确定这个调用当前是否阻塞了调用线程。
isClientAsync()	布尔值	确定方法是否标记为客户端同步，这意味着调用必须是异步的，无论服务器端方法是否为异步方法。
isIdempotent()	布尔值	确定方法是否标记为 idempotent，即方法多次被调用，且没有额外的效果。
setBlockingCaller(boolean)	void	确定这个调用目前阻止调用线程。
setLocator(EJBLocator<T>)	**<T> void**	为调用目标设置 locator。

- *org.ejb.client.ejbLocator 类添加了以下新方法：*

方法	类型	描述
asStateful()	**StatefulEJBLocator<T>**	将此 locator 返回为有状态的 locator（如果是）。
asStateless()	**StatelessEJBLocator<T>**	将此 locator 返回为无状态的 locator（如果是）。
isEntity()	布尔值	确定这是否是实体 locator。
isHome()	布尔值	确定这是否是家 locator。
isStateful()	布尔值	确定这是否是状态 locator。

方法	类型	描述
isStateless()	布尔值	确定这是否是无状态 locator。
withNewAffinity(Affinity)	**抽象 EJBLocator<T>**	创建这个 locator 的一个副本，但具有新的给定关联性。

- *引入了一个新的 org.ejb.client.ejbClientPermission 类，它是 java.security.Permission 的子类，用于控制对特权 Enterprise Java Beans 操作的访问。它提供以下构造器：*
  - *EJBClientPermission (String name)*
  - *EJBClientPermission (String name, String action)*
- *它提供以下方法：*

方法	类型	描述
equals(EJBClientPermission obj)	布尔值	检查两个 **EJBClientPermission** 对象是否相等。
equals(Object obj)	布尔值	检查两个 **Permission** 对象是否相等。
equals (Permission obj)	布尔值	检查两个 **Permission** 对象是否相等。
getActions()	字符串	以字符串形式返回操作。
hashcode()	int	返回此 **Permission** 对象的散列代码值。
means (EJBClientPermission permissions)	布尔值	检查指定权限的行为是否由 **EJBClientPermission** 对象的行为*决定*。
implies (Permission permissions)	布尔值	检查指定权限的操作 *是否由这个* **Permission** 对象的操作表示。

- *引入了一个新的 org.ejb.client.ejbMethodLocator 类，以查找特定的 Enterprise Java Beans 方法。它提供以下构造器：*

- *EJBMethodLocator (String methodName, String… parameterTypeNames)*

- *它提供以下方法：*

方法	类型	描述
equals (EJBMethodLocator other)	布尔值	确定此对象是否等于另一个对象。
equals (Object other)	布尔值	确定此对象是否等于另一个对象。
forMethod (Method method)	**静态 EJBMethodLocator**	获取给定反映方法的 locator。
getMethodName()	字符串	获取方法名称。
getParameterCount()	int	获取参数数。
getParameterTypeName(int index)	字符串	在给定索引中获取参数名称。
hashCode()	int	获取哈希代码。

- *在故障情况下，引入了一个新的 org.jboss.ejb.client.ejbReceiverInvocationContext.ResultProducer.Failed 类。它提供以下构造器：*

  - *failed (Exception cause)*

- *它提供以下方法：*

方法	类型	描述
discardResult()	void	丢弃结果，表示不会使用它。
getResult()	**对象**	获取结果。

- *引入了一个新的 org.jboss.ejb.client.ejbReceiverInvocationContext.ResultProducer.Immediate 类来获取即*

时结果。它提供以下构造器：

- *failed (Exception cause)*

- *它提供以下方法：*

方法	类型	描述
discardResult()	void	丢弃结果，表示不会使用它。
getResult()	**对象**	获取结果。

- *为 URI 关联性规格引入了一个新的 org.jboss.ejb.client.URIAffinity 类，它是 org.jboss.ejb.client.Affinity 的子类。它使用 Affinity.forUri (URI) 创建。*

- *它提供以下方法：*

方法	类型	描述
equals (Affinity other)	布尔值	指明另一个对象是否等于这个对象。
equals (Object other)	布尔值	指明另一个对象是否等于这个对象。
equals (URIAffinity other)	布尔值	指明另一个对象是否等于这个对象。
getURI()	URI	获取关联的 URI。
hashCode()	int	获取哈希代码。
toString()	字符串	返回对象的字符串表示。

- *org.jboss.ejb.client.ejbMetaData Impl 类弃用以下方法：*

  - *toAbstractEJBMetaData()*

  - *EJBMetaDataImpl(AbstractEJBMetaData<?,?>)*

7.13. 迁移客户端以使用 WILDFLY 配置文件

在发布 JBoss EAP 7.1 之前，JBoss EAP 客户端库（如企业 Java Beans 和命名）使用不同的配置策略。JBoss EAP 7.1 引入了 wildfly-config.xml 文件，其目的是将所有客户端配置统一到一个配置文件中，其方式与处理服务器配置的方式类似。

例如，在 JBoss EAP 7.1 之前，您可以使用 jboss-ejb-client.properties 文件为企业 Java Beans 客户端创建一个新的 InitialContext，或使用 Properties 类编程设置属性。

示例： jboss-ejb-client.properties 属性文件

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=one
remote.connection.one.port=8080
remote.connection.one.host=127.0.0.1
remote.connection.one.username=quickuser
remote.connection.one.password=quick-123
```

在 JBoss EAP 7.1 及更高版本中，您可以在客户端存档的 META-INF/ 目录中创建 wildfly-config.xml 文件。这是使用 wildfly-config.xml 文件相同的配置。

示例：使用 wildfly-config.xml 文件评估配置

```
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.7">
        <authentication-rules>
            <rule use-configuration="ejb"/>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="ejb">
                <set-user-name name="quickuser"/>
                <credentials>
                    <clear-password password="quick-123"/>
                </credentials>
            </configuration>
        </authentication-configurations>
    </authentication-client>
    <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.2">
        <connections>
            <connection uri="remote+http://127.0.0.1:8080" />
```

```
        </connections>
    </jboss-ejb-client>
</configuration>
```

其他资源

- 使用 Elytron 客户端配置客户端身份验证。
- 使用 wildfly-config.xml 文件配置客户端。

7.14. 迁移部署计划配置

Java EE 应用部署规范(concurrency-88) 旨在定义标准合同，使多个提供程序的工具能够在任何 Java EE 平台产品上配置和部署应用。Java EE 产品供应商需要合同来实施 DeploymentManager 和其他 javax.enterprise.deploy.spi 接口，供工具提供程序访问。对于 JBoss EAP 6，部署计划由名为 deployment-plan.xml 的 XML 描述符来标识，该描述符捆绑在存档或 JAR 存档中。

这种规格非常少地采用，因为大多数应用服务器产品都提供自己的"功能丰富的"部署解决方案。因此，Java EE 7 丢弃了 JSR-88 支持，并随后从 JBoss EAP 7 中丢弃了 JSR-88 的支持。

如果您使用 JSR-88 来部署应用，您现在必须使用另一种方法来部署应用。JBoss EAP 管理 CLI 部署命令提供了将存档部署到单机服务器或受管域中的服务器组的标准方式。有关管理 CLI 的更多信息，请参阅 管理 CLI 指南。

7.15. 迁移自定义应用程序 VALVES

您必须手动迁移 jboss-web.xml XML 文件中定义的任何 valves 或 valves。这包括通过扩展 jboss-web.xml 描述符文件的 <valve> 元素配置的 org.apache.catalina.valves.ValveBase 创建的值。

迁移在部署中配置的 Valves

在 JBoss EAP 6 中，您可以在应用程序级别定义自定义 valves，方法是在 jboss-web.xml web 应用描述符文件中配置它们。自 JBoss EAP 7 起，也可以使用 Undertow 处理程序进行此操作。

以下是 JBoss EAP 6 中 jboss-web.xml 文件中配置的 valve 示例。

-

```
<jboss-web>
  <valve>
    <class-name>org.jboss.examples.MyValve</class-name>
    <param>
      <param-name>myParam</param-name>
      <param-value>foobar</param-value>
    </param>
  </valve>
</jboss-web>
```

有关如何在 JBoss EAP 中创建和配置自定义处理程序的更多信息，请参阅 JBoss EAP 7.4 开发指南中的 创建自定义 处理程序。

迁移自定义 Authenticator Valves

有关如何迁移验证器 valves 的详情，请参考 迁移验证器 valves。

7.16. 安全应用程序更改

使用 Undertow 替换 JBoss Web 需要更改 JBoss EAP 7 的安全性配置。从 JBoss EAP 8.0 开始，您必须使用 Elytron 作为旧安全性，因为 PicketBox 不再可用。

7.16.1. 迁移验证器 valves

如果您创建了在 JBoss EAP 6.4 中扩展 AuthenticatorBase 的自定义验证器 valve，则必须手动将它替换为 JBoss EAP 7 中的自定义 HTTP 身份验证实施。HTTP 身份验证机制是在 elytron 子系统中创建的，然后注册到 undertow 子系统。有关如何实现自定义 HTTP 身份验证机制的详情，请参考 JBoss EAP 7.4 开发指南中的 自定义 HTTP 机制。

7.16.2. PicketLink 删除

PicketLink 已从 JBoss EAP 8.0 中删除。

PicketLink SP

使用 Keycloak SAML 适配器而不是 PicketLink 服务供应商。

要通过配置 Keycloak SAML 适配器从 PicketLink 迁移，请执行以下任务：

- 将 Keycloak SAML 客户端安装到 JBoss EAP 8.0。如需更多信息，请参阅

- 使用 jboss-eap-installation-manager 安装 JBoss EAP 8.0
- 用于使用 SAML 保护应用程序的 Keycloak SAML 适配器功能 pack

- 如果需要，配置 Keycloak SAML 而不是 PicketLink IdP。要使用 Keycloak SAML 保护 SP 应用程序，需要创建一个 SAML 客户端。有关创建 Keycloak SAML 客户端的更多信息，请参阅 JBoss EAP 7.5 服务器管理指南中的创建 SAML 客户端。

- 更新应用程序以使用 Keycloak SAML 适配器。有关更新应用程序的更多信息，请参阅使用 SAML 保护 Web 应用程序。

PicketLink IDP

从 JBoss EAP 8.0 开始，PicketLink IDP 不可用，您可以配置红帽构建的 Keycloak。如需更多信息，请参阅配置红帽构建的 Keycloak。

PicketLink STS

在以前的版本中，您可以将 PicketLink STS 配置为 Apache CXF 安全令牌服务实现的替代选择。PicketLink STS 配置涉及旧的安全域。需要删除对 STS 应用程序中的旧安全域和 PicketLink 的任何引用，因此您必须配置 Apache CXF STS。

有关如何配置 Apache CXF STS 的更多信息，请参阅 JBoss EAP 7.4 开发 Web 服务应用程序 中的安全令牌服务(STS)。

7.16.3. Vault 删除

Vault 已从 JBoss EAP 8.0 中删除。如果您的应用程序使用传统的 vault 表达式，则必须迁移和使用 Elytron 加密表达式。

检查部署文件中的 ${VAULT:: 实例，它们可能位于注解或部署描述符中，并将它们替换为对应的加密表达式。

其他资源

- Elytron 中加密的表达式

7.16.4. OIDC 客户端迁移

JBoss EAP 8.0 不支持 Keycloak OIDC 客户端适配器，并被原生 Elytron OIDC 客户端替代，提供类似的功能和配置。

要从 Keycloak OIDC 客户端适配器迁移到原生 Elytron OIDC 客户端，请按照以下步骤执行：

- 在应用程序的 web.xml 文件中检查 <auth-method>KEYCLOAK </auth-method>，并将它替换为部署的 web.xml 文件中的 <auth-method>OIDC </auth-method>。
- 检查是否存在 WEB-INF/keycloak.json，并将它重命名为 WEB-INF/oidc.json。

其他资源

- JBoss EAP 中的 OpenID Connect 配置
- 迁移 keycloak 子系统

7.16.5. 自定义登录模块迁移

在 JBoss EAP 8.0 中，旧的 security 子系统已被删除。要继续使用 elytron 子系统的自定义登录模块，请使用新的 Java 身份验证和授权服务(JAAS)安全域和 jaas-realm。

其他资源

- elytron 子系统中的 JAAS 域

7.16.6. 其他安全应用程序更改

JBoss EAP 7.2 或更高版本和更早的版本之间存在一些显著区别：

- jboss-web.xml 中不需要 NegotiationAuthenticator valve，但仍必须是 web.xml 中定义的 < security-constraint> 和 &lt ;login-config > 元素。它们用于决定保护哪些资源。
-

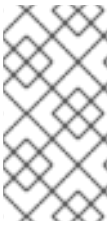
< login-config& gt; 元素中的 auth-method 元素现在是一个用逗号分开的列表。确切的值 SPNEGO 必须在此处，应首先显示在该列表中。如果需要 FORM 身份验证作为回退，则确切的值为 SPNEGO,FORM。

- *jboss-deployment-structure.xml 文件不是必需的。*

7.17. JBOSS LOGGING 更改

从 JBoss EAP 7 开始，如果应用程序使用 JBoss Logging，请注意 org.jboss.logging 软件包中的注解已弃用。它们已移到 org.jboss.logging.annotations 软件包中，因此您必须更新源代码以导入新软件包。

注解也移到单独的 Maven groupId:artifactId:version (GAV) ID 中，因此您需要在项目 pom.xml 文件中为 org.jboss.logging:jboss-logging-annotations 添加新的项目依赖项。



注意

只有日志记录注解已移动。org.jboss.logging.BasicLogger 和 org.jboss.logging.Logger 仍存在于 org.jboss.logging 软件包中。

下表列出了已弃用的注解类和对应的替换。

表 7.1. 弃用的日志记录注解替换

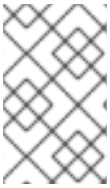
弃用的类	替换类
org.jboss.logging.Cause	org.jboss.logging.annotations.Cause
org.jboss.logging.Field	org.jboss.logging.annotations.Field
org.jboss.logging.FormatWith	org.jboss.logging.annotations.FormatWith
org.jboss.logging.LoggingClass	org.jboss.logging.annotations.LoggingClass
org.jboss.logging.LogMessage	org.jboss.logging.annotations.LogMessage
org.jboss.logging.Message	org.jboss.logging.annotations.Message
org.jboss.logging.MessageBundle	org.jboss.logging.annotations.MessageBundle

弃用的类	替换类
org.jboss.logging.MessageLogger	org.jboss.logging.annotations.MessageLogger
org.jboss.logging.Param	org.jboss.logging.annotations.Param
org.jboss.logging.Property	org.jboss.logging.annotations.Property

7.18. JAKARTA FACES 代码更改

本节介绍将应用迁移到 JBoss EAP 时 Jakarta Faces 代码更改的影响。

4.0 之前丢弃了对 Jakarta Server Faces 的支持



注意

Jakarta Server Faces 是 JavaServer Faces 的新名称。

使用 JBoss EAP 6.4，您可以通过创建 jboss-deployment-structure.xml 文件，在应用程序部署中继续使用 Jakarta Server Faces 1.2。JBoss EAP 7.4 包含 Jakarta Server Faces 2.3，不再支持 Jakarta Server Faces 1.2 API。如果您的应用程序使用 Jakarta Server Faces 1.2，则必须重写它以使用 Jakarta Server Faces 2.3。

JBoss EAP 8.0 不再支持 4.0 之前的任何 JSF 版本。

7.19. 在其他方面集成 MYFACES

您可以通过引入 Galleon 功能软件包 eap-myfaces-feature-pack 来简化其他 Jakarta Faces 实施的安装 MyFaces，作为 JBoss EAP 中的默认 Mojarra Jakarta Faces 实施的替代选择。您可以使用此功能 pack 在 JBoss EAP 中置备不同的 Jakarta Faces 实施。

您可以使用 eap-myfaces-feature-pack 来使用 MYFACES_VERSION 环境变量选择 MyFaces 版本。此功能软件包引入了一个名为 MyFaces 的单个层，提供安装选项，并选择 MyFaces 作为替代方案。如需更多信息，请参阅如何在 EAP 8 中配置 Multi-JSF 功能。

注意

与 JBoss EAP 8.0 的兼容性仅限于 4.x 版本。

7.20. 模块类加载更改

在 JBoss EAP 7 中，当多个模块包含多个类或软件包时，类加载行为已更改。

假设有两个模块 MODULE_A 和 MODULE_B，它们相互依赖，并包含其中一些相同的软件包。在 JBoss EAP 6 中，从依赖项加载的类或软件包优先于 module.xml 文件的 resource-root 中指定的类或软件包。这意味着 MODULE_A 看到 MODULE_B 和 MODULE_B 的软件包会看到 MODULE_A 的软件包。这个行为令人混淆，可能会导致冲突。JBoss EAP 7 中已更改此行为。现在，module.xml 文件中的 resource-root 指定的类或软件包优先于由依赖项指定的那些。这意味着 MODULE_A 查看 MODULE_A 和 MODULE_B 的软件包是 MODULE_B 的软件包。这可防止冲突，并提供更适当的行为。

如果您定义了自定义模块，其中包含 resource-root 库或包含在其模块依赖项中重复的类的软件包，您可能会看到 ClassCastException、LinkageError、类加载错误或其他在迁移到 JBoss EAP 7 时的行为更改。要解决这个问题，您必须配置 module.xml 文件，以确保只使用一个类版本。这可以通过以下方法之一来实现。

- 您可以避免指定在模块依赖项中重复类的 resource-root。
- 您可以使用 imports and exports 元素的 include 和 exclude 子元素来控制 module.xml 文件中的类加载。以下是排除类在指定软件包中的导出元素。

```
<exports>
  <exclude path="com/mycompany/duplicateclassespath/"/>
</exports>
```

如果要保留现有行为，则必须使用 filter 元素过滤 module.xml 文件中的依赖 resource-root 的依赖项软件包。这样，您可以保留现有的行为，而无需在 JBoss EAP 6 下看到的奇数循环。以下是一个 root-resource 示例，它可过滤指定软件包中的类。

```
<resource-root path="mycompany.jar">
  <filter>
    <exclude path="com/mycompany/duplicateclassespath"/>
  </filter>
</resource-root>
```

有关模块和类加载的更多信息，请参阅 JBoss EAP 7.4 开发指南中的 类加载和模块。

7.21. 应用程序集群更改

本节概述将应用从 JBoss EAP 6 迁移到 JBoss EAP 8 所需的集群更改。此外，本节描述了集群更改可能会影响您的应用程序迁移到 JBoss EAP 8.0。

7.21.1. 新集群功能概述

下表介绍了将应用从 JBoss EAP 6 迁移到 JBoss EAP 8.0 时应注意的一些新集群功能。

- JBoss EAP 7 引入了一个新的公共 API，用于构建可显著简化流程的单例服务。有关单例服务的信息，请参阅 JBoss EAP 7.4 开发指南中的 HA 单例服务

- 可将单例部署配置为一次仅在集群中的一个节点上部署和启动。如需更多信息，请参阅 JBoss EAP 7.4 开发指南中的 HA 单例部署。

- 现在，您可以定义集群单例 MDB。如需更多信息，请参阅 JBoss EAP 7.4 开发 Jakarta Enterprise Beans 应用 中的 Clustered Singleton MDB。

- JBoss EAP 8.0 包括 Undertow mod_cluster 实施。这提供了一个纯 Java 负载平衡解决方案，不需要 httpd Web 服务器。如需更多信息，请参阅 JBoss EAP 7.4 配置指南中的 将 JBoss EAP 配置为 Front-end Load Balancer。

7.21.2. Web Session 集群更改

JBoss EAP 7 引入了一个新的 Web 会话集群实施。它取代了以前的实施，它与旧的 JBoss Web 子系统源代码紧密耦合。

新的 Web 会话集群实施会影响如何在 jboss-web.xml JBoss EAP 专有 Web 应用程序 XML 描述符文件中配置应用程序。以下是此文件中唯一保留的集群配置元素。

```
<jboss-web>
  ...
  <max-active-sessions>...</max-active-sessions>
  ...
  <replication-config>
```

```
    <replication-granularity>...</replication-granularity>
    <cache-name>...</cache-name>
  </replication-config>
  ...
</jboss-web>
```

distributable-web 子系统弃用了 jboss-web.xml 的 <replication-config> 元素。它通过生成临时分布式 web 会话配置集来提高 <replication-config> 的使用。

您可以通过按名称或提供特定于部署的会话管理配置来覆盖默认的可分布式会话管理行为。如需更多信息，请参阅 覆盖默认的可分布式会话管理行为。

下表论述了如何为 jboss-web.xml 文件中的元素实现类似行为，这些行为现已过时。

配置元素	更改描述
<max-active-sessions/>	在以前的版本中，如果会话创建的结果超过 < **max-active-sessions/> 指定的值，则会话创建会失败**。 在新的实现中，**<max-active-sessions/>** 用于启用会话传递。如果会话创建将导致活跃会话数量超过 &lt **;max-active-sessions** />，则会话管理器最旧的会话将传递新会话的空间。
<passivation-config/>	从 JBoss EAP 7 开始，不再使用此配置元素及其子元素。
<use-session-passivation/>	在以前的版本中，使用此属性启用 passivation。 在新的实现中，通过为 **<max-active-sessions/>** 指定非负值来启用 passivation。
<passivation-min-idle-time/>	在以前的版本中，在成为传递候选者前，会话需要最少处于活跃状态。这可能会导致会话创建失败，即使启用了 passivation。 新的实现不支持这个逻辑，因此避免了这个拒绝服务(DoS)漏洞。
<passivation-max-idle-time/>	在以前的版本中，会话会在闲置特定时间后被传递。 新实施只支持 lazy passivation。它不支持强制传递。只有在需要符合 **<max-active-sessions/>** 时，会话才会被传递。
<replication-config/>	**distributable-web** 子系统弃用此元素。如需更多信息，请参阅 JBoss EAP 7.4 *开发指南中的*用于分布式 Web 会话配置的 distribut-web 子系统和 覆盖默认的可分布式会话管理 行为。
<replication-trigger/>	在以前的版本中，这个元素用于决定何时触发会话复制。新的实现将这个配置选项替换为一个可靠的策略。如需更多信息，请参阅 JBoss EAP 7.4 *开发指南中的* 不可变会话属性。

配置元素	更改描述
<use-jk/>	在以前的版本中，处理给定请求的节点的 **instance-id** 附加到 **jsessionid**，供负载均衡器（如 mod_jk、mod_proxy_balancer, mod_cluster）使用，具体取决于为 < **use-jk/>** 指定的值。 在新的实现中，如果定义，**instance-id** 始终附加到 **jsessionid**。
<max-unreplicated-interval/>	在以前的版本中，这个配置选项旨在优化，在没有会话属性更改时防止复制会话的时间戳。虽然这种声音 nice，在实践中，但实际上它不会阻止任何 RPC，因为会话访问需要缓存事务 RPC，无论是否更改了会话属性。 在新的实现中，会话的时间戳在每次请求中都会复制。这可防止在故障切换后进行过时的会话元数据。
<snapshot-mode/>	在以前的版本中，可以将 **<snapshot-mode/>** 配置为 **INSTANT** 或 **INTERVAL**。Infinispan 的异步复制使该配置选项变得过时。
<snapshot-interval/>	这只适用于 **<snapshot-mode>INTERVAL</snapshot-mode>**。因为 **<snapshot-mode** /> 已过时，这个选项也会过时。
<session-notification-policy/>	在以前的版本中，此属性指定的值定义了一个用于触发会话事件的策略。 在新的实现中，此行为是规范驱动的，不可配置。

这个新实施还支持直写缓存存储和仅传递缓存存储。通常，直写缓存存储与无效缓存结合使用。与无效缓存一起使用时，JBoss EAP 6 中的 Web 会话集群实施不正确。

7.21.3. 覆盖默认的可分布式会话管理行为

您可以使用以下方法之一覆盖默认的可分布式会话管理行为：

- *按名称引用会话管理配置集*
- *提供特定于部署的会话管理配置*

引用现有的会话管理配置集

- *要使用现有的分布式会话管理配置集，请在应用程序的 /WEB-INF 目录中包含一个 distributable-web.xml 部署描述符。例如：*

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
    <session-management name="foo"/>
</distributable-web>
```

- 或者，在现有的 **jboss-all.xml** 部署描述符中定义目标分布式会话管理配置集：

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
    <distributable-web xmlns="urn:jboss:distributable-web:1.0">
        <session-management name="foo"/>
    </distributable-web>
</jboss>
```

使用特定于 Deployment 的 Session Management 配置集

如果只有一个 **Web** 应用使用自定义会话管理配置，您可以在部署描述符本身中定义配置。临时配置与 **distributable-web** 子系统使用的配置相同。

- 在部署描述符中定义自定义会话管理配置。例如：

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
    <infinispan-session-management cache-container="foo" cache="bar" granularity="SESSION">
        <primary-owner-affinity/>
    </infinispan-session-management>
</distributable-web>
```

- 或者，在现有的 **jboss-all.xml** 部署描述符中定义会话管理配置：

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
```

```
    <distributable-web xmlns="urn:jboss:distributable-web:1.0">
      <infinispan-session-management cache-container="foo" cache="bar" granularity="ATTRIBUTE">
        <local-affinity/>
      </infinispan-session-management>
    </distributable-web>
  </jboss>
```

7.21.4. 有状态会话 EJB 集群更改

在 JBoss EAP 6 中，您需要使用以下方法之一为有状态会话 Bean (SFSB)启用集群行为：

- 您可以在会话 bean 中添加 org.jboss.ejb3.annotation.Clustered 注解。

```
@Stateful
@Clustered
public class MyBean implements MySessionInt {

  public void myMethod() {
    //
  }
}
```

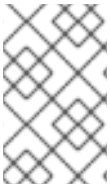
- 您可以将 < clustered> 元素添加到 jboss-ejb3.xml 文件中。

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

从 JBoss EAP 7 开始，您不再需要启用集群行为。默认情况下，如果服务器使用 HA 配置文件启动，SFSB 的状态将自动复制。您可以使用以下方法之一禁用此默认行为：

- 您可以使用 @Stateful (passivationCapable=false) 禁用单个有状态会话的默认行为，这是 Enterprise Java Beans 3.2 规范的新。

- 您可以在服务器配置中 ejb3 子系统的配置全局禁用此行为。



注意

如果 @Clustered 注释没有从应用中删除，它将被简单地忽略，不会影响应用的部署。

7.21.5. 集群服务更改

在 JBoss EAP 6 中，集群服务的 API 位于私有模块中，且不被支持。

JBoss EAP 7 引入了应用程序使用的公共集群服务 API。新服务旨在轻便、易注入，不需要外部依赖项。

- 新的 org.wildfly.clustering.group.Group 接口提供对当前集群状态的访问，并允许侦听集群成员资格更改。
- 新的 org.wildfly.clustering.dispatcher.CommandDispatcher 接口允许在所有这些或所选节点子集中运行代码。

这些服务取代了之前版本中提供的类似 API，即 JBoss EAP 5 中的 HAPartition，以及 JBoss EAP 6 中的 GroupCommunicationService, GroupMembershipNotifier, 和 GroupRpcDispatcher

如需更多信息，请参阅 JBoss EAP 7.4 开发指南中的 集群服务的公共 API。

7.21.6. 迁移集群 HA 单例

在 JBoss EAP 6 中，没有可用于集群范围的 HA 单例服务的公共 API。如果您使用私有 org.jboss.as.clustering.singleton prerequisites 类，您必须在将应用程序迁移到 JBoss EAP 8 时更改您的代码以使用新的公共 org.wildfly.clustering.singleton prerequisites 软件包。

有关 HA 单例服务的更多信息，请参阅 JBoss EAP 7.4 开发指南中的 HA 单例服务。有关 HA 单例部署的信息，请参阅 JBoss EAP 7.4 开发指南中的 HA 单例部署。

7.22. 使用上下文类型进行 CONTEXTSERVICE 自定义

作为 Jakarta EE Concurrency 3.0 的一部分，您可以使用上下文类型自定义 ContextService 属性。Transaction 上下文是一个此类类型。Transaction 上下文取代了 use-transaction-setup-provider resource-definition 属性的使用。当 use-transaction-setup-provider 属性设置为 true 时，事务上下文会被清除，当此属性设置为 false 时，事务上下文将保持不变。

红帽不再支持特定于供应商的配置，因此此类资源定义属性已弃用。在 JBoss EAP 7 中，定义 use-transaction-setup-provider 属性为 false 的默认配置，这意味着在线程上运行上下文任务时事务上下文

保持不变。默认情况下，在 JBoss EAP 8 中，默认的 ContextService 属性与 Jakarta EE Concurrency 3.0 规范一致，并在执行上下文任务前清除事务上下文。

要使用不同的 ContextService，您必须使用 ContextServiceDefinition 注解或在 XML 中指定它，在部署时定义它。

7.23. 删除已弃用的 INITIALCONTEXT 类

在 JBoss EAP 8 中删除了 org.jboss.naming.remote.client.InitialContextFactory 类。在 JBoss EAP 7 中，org.jboss.naming.remote.client.InitialContextFactory 类已弃用，并替换为 org.wildfly.naming.client.WildFlyInitialContextFactory 类。您必须迁移源代码或配置文件以反映这个更改。

命名配置更改：

- 如果用户应用程序使用 system 或 environment 属性，则 java.naming.factory.initial 属性必须从 java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory 迁移到 java.naming.factory.initial=org.wildfly.client.wildfly.client.WildFlyInitialContextFactory。

- 如果用户应用程序使用 WSDL 合同，其中包含 < soapjms:jndiInitialContextFactory >，则必须从 < soapjms:jndiInitialContextFactory>org.jboss.naming.remote.client.InitialContextFactory< soapjms:jndiInitialContextFactory> 迁移其值 要 <soapjms:jndiInitialContextFactory>org.wildfly.naming.client.WildFlyInitialContextFactory<soapjms:jndiInitialContextFactory>.

- 如果用户应用程序使用 Java 代码来配置远程命名，则必须从 Properties env = new Properties （）;env.put (Context.put （Context.INITIAL_CONTEXT_FACTORY, org.jboss.naming.remote.client.InitialContextFactory.class.getName （)）更新为 env.put (Context.INITIAL_CONTEXT_FACTORY, org.jboss.naming.remote.client.InitialContextFactory.class.getName （)）； org.wildfly.naming.client.WildFlyInitialContextFactory.class.getName （）；

以下列出的方法（来自 org.wildfly.naming.client.ProviderEnvironment 类）已在 JBoss EAP 7 中弃用，现已在 JBoss EAP 8 中删除，作为红帽承诺在我们的代码、文档和 Web 属性中替换有问题的语言的一部分。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

包含删除方法的任何代码都必须使用对应的替换来重构：

- *getBlackList （） 替换为 getBlockList （）*
- *updateBlacklist (URI) 被 updateBlockList (URI)替代*
- *dropFromBlacklist (URI) 被 dropFromBlocklist (URI)替代*

7.24. 资源适配器

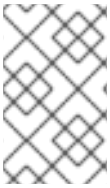
Jakarta Connectors 资源适配器可让您的应用程序与任何消息传递供应商通信。它配置 Jakarta EE 组件（如 MDB 和其他 Jakarta Enterprise Beans）如何发送或接收消息。

7.24.1. 部署 IBM MQ 资源适配器

IBM MQ 是 IBM 提供的消息传递导向中间件(MOM)产品，允许分布式系统上的应用程序相互通信。这可以通过使用消息和消息队列来实现。IBM MQ 负责将消息发送到消息队列，并使用消息通道将数据传送到其他队列管理器。有关 IBM MQ 的更多信息，请参阅 IBM 产品网站上的 IBM MQ。

概述

IBM MQ 可以配置为 JBoss EAP 8.0 的外部 Java 消息服务供应商。本节介绍在 JBoss EAP 中部署和配置 IBM MQ 资源适配器的步骤。此部署和配置可以通过管理 CLI 工具或基于 Web 的管理控制台来完成。有关 IBM MQ 支持的配置，请参阅 JBoss EAP 支持的配置。



注意

在配置 IBM MQ 资源适配器后，您必须重启系统才能使配置更改生效。

JBoss EAP 8.0 是一个 Jakarta EE 10 实现，因此用于所有 EE API 的软件包已从 javax 改为 jakarta，这需要 Jakarta EE 10 兼容资源适配器。如果您在 JBoss EAP 7.x 或更早版本中使用 IBM MQ 资源适配器，则必须使用 wmq.jakarta.jakarta.rar，使用这个 jakarta 命名空间的 IBM MQ 资源适配器。

- *为 wmq.jmsra.rar 删除和取消部署以前的资源适配器配置，并使用 wmq.jakarta.jmsra.rar*
- *根据本节中介绍的步骤，部署 wmq.jakarta.jmsra.rar 并配置。*

先决条件

开始之前，您必须验证 IBM MQ 资源适配器的版本并了解其配置属性。

- IBM MQ 资源适配器作为名为 wmq.jakarta.jmsra.rar 的资源存档(RAR)文件提供。您可以从 /opt/mqm/java/lib/jca/ wmq.jakarta.jmsra.rar 获取 wmq.jakarta.jmsra.rar 文件。如需有关 JBoss EAP 的每个发行版本支持 的特定版本的信息，请参阅 JBoss EAP 支持的配置。

- 您必须知道以下 IBM MQ 配置值。有关这些值的详情，请参阅 IBM MQ 产品文档。

  - MQ_QUEUE_MANAGER: IBM MQ 队列管理器的名称
  - MQ_HOST_NAME ：用于连接 IBM MQ 队列管理器的主机名
  - MQ_CHANNEL_NAME ：用于连接到 IBM MQ 队列管理器的服务器频道
  - MQ_QUEUE_NAME ：目标队列的名称
  - MQ_TOPIC_NAME ：目标主题的名称
  - MQ_PORT ：用于连接 IBM MQ 队列管理器的端口
  - MQ_CLIENT ：传输类型

- 对于出站连接，还必须熟悉以下配置值：

  - MQ_CONNECTIONFACTORY_NAME ：将提供远程系统连接的连接工厂实例的名称

流程

以下是 IBM 提供的默认配置，可能随时更改。如需更多信息，请参阅 IBM MQ 文档。

1. 首先，通过将 wmq.jakarta.jmsra.rar 文件复制到 EAP_HOME/standalone/deployments/ 目录中来手动部署资源适配器。

2. 接下来，使用管理 CLI 添加资源适配器并进行配置。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-support=XATransaction)
```

请注意，transaction-support 元素被设置为 XATransaction。在使用事务时，请务必提供 XA 恢复数据源的安全域，如下例所示。

```
/subsystem=resource-adapters/resource-adapter=test/connection-definitions=test:write-attribute(name=recovery-security-domain,value=myDomain)
```

有关 XA 恢复的更多信息，请参阅 JBoss EAP 7.4 配置指南中的配置 XA 恢复。

对于非事务部署，将 transaction-support 的值改为 NoTransaction。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-support=NoTransaction)
```

3. 现在创建了资源适配器，您可以在其中添加必要的配置元素。

   a. 为队列添加 admin-object 并配置其属性。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao:add(class-name=com.ibm.mq.jakarta.connector.outbound.MQQueueProxy, jndi-name=java:jboss/MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueName:add(value=MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueManagerName:add(value=MQ_QUEUE_MANAGER)
```

b.

*为主题添加 **admin-object** 并配置其属性。*

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao:add(class-name=com.ibm.mq.jakarta.connector.outbound.MQTopicProxy, jndi-name=java:jboss/MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao/config-properties=baseTopicName:add(value=MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-objects=topic-ao/config-properties=brokerPubQueueManager:add(value=MQ_QUEUE_MANAGER)
```

c.

为受管连接工厂添加连接定义并配置其属性。

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd:add(class-name=com.ibm.mq.jakarta.connector.outbound.ManagedConnectionFactoryImpl, jndi-name=java:jboss/MQ_CONNECTIONFACTORY_NAME, tracking=false)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=hostName:add(value=MQ_HOST_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=port:add(value=MQ_PORT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=channel:add(value=MQ_CHANNEL_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=transportType:add(value=MQ_CLIENT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-definitions=mq-cd/config-properties=queueManager:add(value=MQ_QUEUE_MANAGER)
```

4.

如果要将 JBoss EAP 中的 EJB3 消息传递系统的默认提供程序从 JBoss EAP 8.0 消息改为 IBM MQ，请使用管理 CLI 修改 ejb3 子系统，如下所示：

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name,value=wmq.jakarta.jmsra.rar)
```

5.

在 MDB 代码中配置 @ActivationConfigProperty 和 @ResourceAdapter 注释，如下所示：

```
@MessageDriven(name="IbmMqMdb", activationConfig = {
```

```
    @ActivationConfigProperty(propertyName = "destinationType",propertyValue =
"jakarta.jms.Queue"),
    @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
    @ActivationConfigProperty(propertyName = "hostName", propertyValue =
"MQ_HOST_NAME"),
    @ActivationConfigProperty(propertyName = "port", propertyValue = "MQ_PORT"),
    @ActivationConfigProperty(propertyName = "channel", propertyValue =
"MQ_CHANNEL_NAME"),
    @ActivationConfigProperty(propertyName = "queueManager", propertyValue =
"MQ_QUEUE_MANAGER"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue =
"MQ_QUEUE_NAME"),
    @ActivationConfigProperty(propertyName = "transportType", propertyValue =
"MQ_CLIENT")
})

@ResourceAdapter(value = "wmq.jakarta.jmsra-VERSION.rar")
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class IbmMqMdb implements MessageListener {
}
```

务必将 @ResourceAdapter 值中的 VERSION 替换为 RAR 名称中的实际版本。

6. 激活资源适配器：

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:activate()
```

7.24.1.1. IBM MQ 资源适配器的限制和已知问题

下表列出了 IBM MQ 资源适配器的已知问题。版本列中的复选标记(mvapich)表示问题是该资源适配器的版本有问题。

表 7.2. IBM MQ 资源适配器的已知问题

JIRA	问题描述	IBM MQ 9
JBEAP-503	IBM MQ 资源适配器返回 **Queue.toString （）** 和 **QueueBrowser.getQueue （）.toString （）** 方法的不同 String 值。队列 是 **com.ibm.mq.connector.outbound.MQQueueProxy** 类的实例，它与 **com.ibm.mq.jms.MQQueue** 类不同，后者由 **QueueBrowser.htmlQueueBrowser.getQueue （）** 方法返回。这些类包含 **toString （）** 方法的不同实现。请注意，您无法依赖这些 **toString （）** 方法返回相同的值。	✔

JIRA	问题描述	IBM MQ 9
JBEAP-511, JBEAP-550, JBEAP-3686	以下限制适用于 IBM MQ 的消息属性名称。 • 在部署描述符的 **activation-config** 部分中，您无法使用特殊字符（如 _、**&** amp; 或 | ）配置 **destinationName** 属性。使用这些字符会导致 MDB 部署失败，并显示 **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 异常。 • 在部署描述符的 **activation-config** 部分中，您不能使用 **java:/** 前缀配置 **destinationName** 属性。使用此前缀会导致 MDB 部署失败，并显示 **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 异常。 • 属性不能以"JMS"或"usr.JMS"开头，因为它们保留供 IBM MQ JMS 类使用。IBM 知识库网站上有例外说明。 有关消息属性名称限制的完整列表，请参阅 IBM 知识库网站上的 Property 名称限制。	✔
JBEAP-549	使用 **@ActivationConfigProperty** 注释为 MDB 指定 **destination** 属性值时，您必须使用所有大写字母。例如： `@ActivationConfigProperty(propertyName = "destination", propertyValue = "QUEUE")`	✔
JBEAP-624	如果使用 IBM MQ 资源适配器在 Jakarta EE 部署中使用 **@JMSConnectionFactoryDefinition** 注解创建连接工厂，您必须指定 **resourceAdapter** 属性。否则，部署将失败。 `@JMSConnectionFactoryDefinition( name = "java:/jms/WMQConnectionFactory", interfaceName = "javax.jms.ConnectionFactory", resourceAdapter = "wmq.jmsra", properties = { "channel=<channel>", "hostName=<hostname_wmq_broker>", "transportType=<transport_type>", "queueManager=<queue_manager>" } )`	✔
JBEAP-2339	IBM MQ 资源适配器可以从队列和主题读取消息，即使在连接启动前也是如此。这意味着消费者可以在连接启动前使用消息。要避免遇到这个问题，请使用远程 IBM MQ 代理使用 **external-context** 而不是 IBM MQ 资源适配器创建的连接工厂。	✔

JIRA	问题描述	IBM MQ 9

JBEAP-3685	设置了 **<transaction-support>XATransaction</transaction-support>** 后，一个 **JMSContext** 始终为 **JMSContext.SESSION_TRANSACTED**，无论是使用注入还是手动创建。
	
	在以下代码示例中，**@JMSSessionMode (JMSContext.DUPS_OK_ACKNOWLEDGE)** 被忽略，**JMSContext** 仍然保持在 **JMSContext.SESSION_TRANSACTED**。
	
	`@Inject`
	`@JMSConnectionFactory("jms/CF")`
	`@JMSPasswordCredential(userName="myusername", password="mypassword")`
	`@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)`
	`transient JMSContext context3;`
		✔

JBEAP-14633	根据 JMS 规范，**QueueSession** 接口无法用于创建特定于 publish/subscribe 域的对象，以及从 **Session** 继承的特定方法应抛出 **javax.jms.IllegalStateException**。一个这样的方法是 **QueueSession.createTemporaryTopic （）**。IBM MQ 资源适配器不抛出 **javax.jms.IllegalStateException**，而是抛出 **java.lang.NullPointerException**。	✔
JBEAP-14634	**MQTopicProxy.getTopicName （）** 返回不同的主题名称，而不是由 IBM MQ 代理设置。例如，如果主题名称被设置为 **topic://MYTOPIC?XMSC_WMQ_BROKER_PUBQ_QMGR=QM**，则 **MQTopicProxy** 返回 **topic://MYTOPIC**。	✔
JBEAP-14636	**JMSContext** 的默认 **autoStart** 设置为 **false**，这意味着在创建消费者时不自动启动 **JMSContext** 使用的底层连接。此设置应默认为 **true**。	✔

JIRA	问题描述	IBM MQ 9
JBEAP-14640	当使用无效凭证时，IBM MQ 资源适配器会抛出 **DetailedJMSException** 而不是 **JMSSecurityException**，并将以下错误记录到服务器控制台。 > WARN [org.jboss.jca.core.connectionmanager.pool.strategy.PoolByCri] (EJB default - 7) IJ000604: Throwable while attempting to get a new connection: null: com.ibm.mq.connector.DetailedResourceException: MQJCA1011: Failed to allocate a JMS connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure. 以下是可能导致此问题的代码示例。 > QueueConnection qc = queueConnectionFactory.createQueueConnection("invalidUserName", "invalidPassword");	✔
JBEAP-14642	由于 **MQMessageProducer.send (Destination destination, Message message)** 和 **MQMessageProducer.send (Destination destination, Message message, int deliveryMode, int priority, CompletionListener completionListener)** 方法中的资源适配器的无效类 cast 转换，并将以下错误消息记录到服务器控制台。 > **SVR-ERROR: Expected JMSException, received com.ibm.mq.connector.outbound.MQQueueProxy cannot be cast to com.ibm.mq.jms.MQDestination** 这是因为队列或主题查找中使用的 **JNDI 名称是 com.ibm.mq.connector.outbound.MQQueueProxy/MQTopicProxy**。	✔
JBEAP-14643	**JMSProducer** 接口上的 **setDeliveryDelay (expDeliveryDelay)** 方法不会改变设置。调用此方法后，它保留在默认设置 0。	✔

JIRA	问题描述	IBM MQ 9
JBEAP-14670	如果在 **UserTransaction.begin （）** 之前创建的 **QueueSession** 上完成工作，则该工作不被视为事务的一部分。这意味着，使用此会话发送到队列的任何消息都不会由 **UserTransaction.commit （）** 提交，在 **UserTransaction.rollback （）** 后，消息将保留在队列中。	✔
JBEAP-14675	如果您关闭连接，然后立即使用相同的 **clientID** 创建 **JMSContext**，IBM MQ 资源适配器会间歇性将以下错误记录到服务器控制台。 > ERROR [io.undertow.request] (default task-1) UT005023: Exception handling request to /jmsServlet-1.0-SNAPSHOT/: com.ibm.msg.client.jms.DetailedJMSRuntimeException: MQJCA0002: An exception occurred in the IBM MQ layer. See the linked exception for details. A call to IBM MQ classes for Java(tm) caused an exception to be thrown. 当与同一 **clientID** 的连接关闭后创建新的 **JMSContext** 时，不会出现这个问题。	✔
JBEAP-15535	如果有状态会话 bean 会尝试在容器管理事务(CMT)中发送消息到主题，则消息发送会失败，并显示以下信息： > SVR-ERROR: com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination '*MDB_NAME TOPIC_NAME*' 堆栈追踪显示由以下异常导致的： > com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED') reason '2072' ('MQRC_SYNCPOINT_NOT_AVAILABLE')	
JBEAP-25561	当消息通过 **JMSReplyTo** 标头集发送到目的地时，会在到达 IBM MQ 9 代理后修改该消息。因此，响应的回复消息会被定向到修改后的 **JMS_REPLY** 标头中定义的目的地。 例如，如果 **JMSReplyTo** 标头被设置为队列，名称为 **queue:///MYQUEUE**，使用标头 **message.setJMSReplyTo (queue);** 并发送到名为 **QM** 的队列管理器的 IBM MQ 9.3 代理，其名称更改为 **queue://QM/MYQUEUE**。	✔

7.24.2. 删除 Apache Log4j 版本 1 API

从 JBoss EAP 8 开始，停止了对 Apache Log4j 版本 1 API 的支持。任何未打包 log4j.jar 和 log4j 配置的应用程序都必须更新。

影响：

日志消息将不再基于 logging 子系统进行路由。如果应用程序没有打包 log4j.jar，且以下任何语句都为 true，则需要迁移更改：

- *如果您在部署中使用 log4j，且没有包含 log4j 配置文件，则必须迁移到新的日志 facade 或向部署中添加 log4j 配置。*
- *如果您在部署中使用 log4j.xml、log4j.properties 或 jboss-log4j.xml 文件，且没有打包应用程序中的 log4j.jar。如果它是一个 jboss-log4j.xml 文件，您必须将该文件重命名为 log4j.xml。*
- *如果您在 custom-handler 中的 JBoss EAP Logging 子系统中使用 log4j v1 附加器，它将不再被支持。*
- *如果应用类导入类，如 org.apache.log4j.Logger。*
- *如果应用程序包含 jboss-deployment-structure.xml 或在 MANIFEST . MF 中指定依赖项，则需要删除这些依赖项。*

迁移：

- *更新应用类以使用 Apache Log4jv2 类，或使用 JBoss EAP 8 提供的其他日志记录 API 之一。*
- *将 org.apache.log4j.Logger (log4j v1) 类更改为 org.apache.logging.log4j.Logger (log4j v2)。*
- *如果应用程序软件包 log4j.properties、log4j.xml 或 jboss-log4j.xml，您必须：*
  - *在 JBoss EAP 配置中配置日志。*

- 在应用程序中配置 logging.properties，因为应用程序不支持 log4jv2 配置文件。

或者

- 在应用中打包 Apache Log4j 版本 1 JAR，而不是根据 Logging API 的 JBoss EAP 8 打包。您还可以通过 logging 子系统上的 jboss-deployment-structure.xml exclude-subsystems 从应用程序中排除 JBoss Logging API。

额外详情：

为特定部署禁用隐式日志记录依赖项

- 在应用程序的 jboss-deployment-structure.xml 中，配置 exclude-subsystems 以排除 logging 子系统，例如：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <deployment>
    <exclude-subsystems>
      <subsystem name="logging"/>
    </exclude-subsystems>
  </deployment>
</jboss-deployment-structure>
```

- 如果应用程序是 EAR 文件，并且有一个名为 example.war 的子部署，则 jboss-deployment-structure.xml 文件位于 EAR 文件位置 / META-INF/jboss-deployment-structure.xml，在子部署中声明它将排除该子系统，例如：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <sub-deployment name="example.war">
    <exclude-subsystems>
      <subsystem name="logging"/>
    </exclude-subsystems>
  </sub-deployment>
</jboss-deployment-structure>
```

为所有部署禁用隐式日志记录依赖项

要使日志记录 API 默认对部署不可用，请使用以下 CLI 命令将 add-logging-api-dependencies 设置为 false ：

```
/subsystem=logging:write-attribute(name="add-logging-api-dependencies", value="false")
```

要将 JBoss 模块和日志记录 API 设置为依赖项，请修改 jboss-deployment-structure.xml 或 MANIFEST.MF 配置文件：

```
<subsystem xmlns="urn:jboss:domain:logging:8.0">
    <add-logging-api-dependencies value="false"/>
  ...
</subsystem>
```

注意

如果应用程序软件包了 Apache Log4j v1 JARs 和 log4j 配置，则应用不再由 EAP 管理。* 应用程序不应该尝试因为意外结果写入 server.log，因为应该将日志框架写入特定的日志文件。

如需更多信息，请参阅 JBoss EAP 8.0 中不再提供 Apache Log4j 版本 1。

第 8 章 其它更改

本节概述本发行版本中发生的各种各种各种更改。

8.1. JBOSS EAP 原生和 APACHE HTTP 服务器的交付更改

JBoss EAP 8.0 原生的在此发行版本中与 JBoss EAP 6 中的不同。有些组件包括 Red Hat JBoss Core Services 产品，这是很多 Red Hat JBoss Middleware 产品通用的补充软件。新产品支持更快地发布更新以及更一致的更新体验。JBoss Core Services 产品可下载红帽客户门户网站中的专用位置。

- 下表列出了版本之间交付方法的不同。

软件包	JBoss EAP 6	JBoss EAP 8.0
用于消息传递的 AIO 原生	在单独的 "Native 实用程序" 下载中交付产品	包括在 JBoss EAP 分发中。
Apache HTTP 服务器	在单独的 "Apache HTTP Server" 下载中交付产品	提供了新的 JBoss Core Services 产品
mod_cluster, mod_jk, isapi, 和 nsapi 连接器	在单独的 "Webserver Connector Natives" 下载中交付产品	提供了新的 JBoss Core Services 产品
JSVC	在单独的 "Native 实用程序" 下载中交付产品	提供了新的 JBoss Core Services 产品
OpenSSL	在单独的 "Native 实用程序" 下载中交付产品	提供了新的 JBoss Core Services 产品
tcnatives	在单独的"Native Components" 下载中交付产品	JBoss EAP 7 中删除了对 tcnatives 的支持

JBoss EAP 原生和 Apache HTTP 服务器的其他更改

- 您还应注意以下更改：
  - 已取消将 mod_cluster 和 mod_jk 连接器用于来自 Red Hat Enterprise Linux RPM 频道的 Apache HTTP 服务器的支持。如果您从 Red Hat Enterprise Linux RPM 频道运行 Apache HTTP 服务器，且需要为 JBoss EAP 8.0 服务器配置负载均衡，您可以执行以下操作之一：

- 使用 JBoss Core Services 提供的 Apache HTTP 服务器。
- 您可以将 JBoss EAP 8.0 配置为充当前端负载均衡器。如需更多信息，请参阅 JBoss EAP 7.4 配置指南中的 将 JBoss EAP 配置为 Front-end Load Balancer。
- 您可以在支持并认证的机器上部署 Apache HTTP 服务器，然后在该计算机上运行负载均衡器。有关支持的配置列表，请参阅 JBoss EAP 7.4 配置指南中的 HTTP Connectors 概述。

- 您可以在 Apache HTTP 服务器安装指南中找到有关 JBoss 核心服务 的更多信息。

8.2. 对 AMAZON EC2 上的部署的更改

在 JBoss EAP 7 中对 Amazon Machine Images (AMI)进行了一些更改。本节简要总结了其中一些更改。

- 在 Amazon EC2 中启动非集群和集群 JBoss EAP 实例和域的方式发生了重大变化。
- 在 JBoss EAP 6 中，配置取决于 User Data: 字段。在 JBoss EAP 7 中，解析 User Data: 字段中配置的 AMI 脚本，并在实例启动时自动启动服务器。
- 在 JBoss EAP 6 中安装 Red Hat JBoss Operations Network 代理。从 JBoss EAP 7.0 开始，您必须单独安装它。

有关在 Amazon EC2 上部署 JBoss EAP 7 的详情，请参考在 Amazon Web Services 上部署 JBoss EAP。

8.3. 删除包含共享模块的应用程序

在 JBoss EAP 7.1 服务器中引入的更改，在尝试删除应用时，Maven 插件可能会导致以下失败：如果您的应用程序包含相互交互或依赖的模块，则可能会出现这个错误。

```
WFLYCTL0184:    New missing/unsatisfied dependencies
```

例如，假设您有一个包含两个 Maven WAR 项目模块 application-A 和 application-B 的应用，该应用程序共享由 data-sharing 模块管理的数据。

部署此应用程序时，您必须首先部署 共享数据共享 模块，然后部署依赖于它的模块。部署顺序在父 pom.xml 文件的 & lt;modules > 元素中指定。通过 JBoss EAP 8.0，JBoss EAP 6.4 在 JBoss EAP 6.4 中也是如此。

在 JBoss EAP 7.1 之前的发行版本中，您可以使用以下命令从父项目的根目录取消此应用的所有存档：

```
$ mvn wildfly:undeploy
```

在 JBoss EAP 7.1 和更高版本中，您必须首先取消部署使用共享模块的存档，然后取消部署共享模块。由于无法在 pom.xml 文件中指定未部署的顺序，因此您必须手动取消部署模块。您可以从父目录的根目录中运行以下命令来完成此操作。

```
$ mvn wildfly:undeploy -pl application-A,application-B
$ mvn wildfly:undeploy -pl data-shared
```

此更新的取消行为更为准确，并确保您不会处于不稳定的部署状态。

8.4. 对 ADD-USER 脚本的更改

由于密码策略的变化，在 JBoss EAP 7 中更改了 add-user 脚本的行为。JBoss EAP 6 具有严格的密码策略。因此，add-user 脚本会拒绝不满足最低要求的弱密码。从 JBoss EAP 7 开始，接受弱密码并发出警告。如需更多信息，请参阅 JBoss EAP 7.4 配置指南中的 设置 Add-User 实用程序密码限制。

8.5. 删除 OSGI 支持

当 JBoss EAP 6.0 GA 首次发布时，JBoss OSGi 是 OSGi 规范的实施，它作为技术预览功能包括在内。随着 JBoss EAP 6.1.0 的发布，JBoss OSGi 从技术预览降级为不支持。

在 JBoss EAP 6.1.0 中，单机服务器的 configadmin 和 osgi 扩展模块和子系统配置已移到单独的 EAP_HOME/standalone/configuration/standalone-osgi.xml 配置文件中。由于您不应该迁移这个不支持的配置文件，因此删除 JBoss OSGi 支持不会影响单机服务器配置的迁移。如果您修改了任何其他独立配置文件来配置 osgi 或 configadmin，则必须删除这些配置。

对于受管域，JBoss EAP 6.1.0 发行版中的 osgi 扩展和子系统配置已从

EAP_HOME/domain/configuration/domain.xml 文件中删除。但是，configadmin 模块扩展和子系统配置保留在 EAP_HOME/domain/configuration/domain.xml 文件中。从 JBoss EAP 7 开始，不再支持此配置，必须被删除。

8.6. 使用 JAVA 的附件 API 中的 SOAP 的变化

更新用户定义的 SOAP 处理程序，使其在迁移到 JBoss EAP 8.0 时遵守 SAAJ 3.0 规格。

其他资源

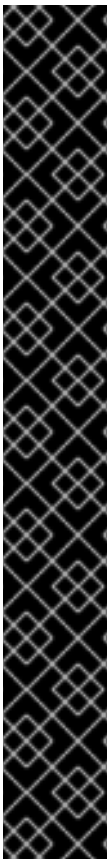
- jakarta Soap with Attachments

第 9 章 迁移到 ELYTRON

JBoss EAP 7.1 引入了 Elytron，它提供了一个统一的框架，可以管理和配置单机服务器和托管域的访问。它还可用于为部署到 JBoss EAP 服务器的应用配置安全访问。

从 JBoss EAP 8.0 开始，您必须使用 Elytron 来迁移应用程序，因为旧的 security 子系统不可用。

9.1. ELYTRON 概述



重要

Elytron 的架构和传统的安全子系统基于 PicketBox 非常不同。使用 Elytron 时，尝试创建一个可让您在当前操作的同一安全环境中运行的解决方案，但这并不意味着 每个 PicketBox 配置选项在 Elytron 中都有相同的配置选项。

如果您无法在文档中查找信息，以帮助您使用传统安全实施时的 Elytron 获得类似的功能，您可以通过以下方式之一找到帮助。

- 如果您有红帽开发订阅，请访问红帽客户门户上的 Support Cases, Solutions, 和 Knowledge Articles您还可以创建一个 技术支持 的问题单，并从 WildFly 社区获得帮助，如下所述。
- 如果您没有红帽开发订阅，您仍然可以访问红帽客户门户网站中的 知识库文章。您还可以加入 用户论坛和实时聊天，以询问 WildFly 社区的问题。WildFly 社区产品由 Elytron 工程团队积极监控。

有关 elytron 子系统中可用的新资源的概述，请参阅 JBoss EAP 7.4 安全架构 中的 Elytron 子系统中的资源。

9.2. 迁移安全 VAULT 和属性

要使用 Elytron，您必须迁移安全库来保护凭证存储，并将旧的安全属性迁移到 Elytron 安全属性。

9.2.1. 将安全 Vault 迁移到安全凭证存储

在 JBoss EAP 7.0 中，用来存储纯文本字符串加密的 安全 库与 JBoss EAP 7.1 或更高版本中的

Elytron 不兼容。JBoss EAP 7.1 或更高版本使用凭证存储来存储字符串。凭据将加密凭据存储在 JBoss EAP 配置文件之外的存储文件中。您可以使用 Elytron 提供的实现，也可以使用凭证存储 API 和 SPI 自定义配置。每个 JBoss EAP 服务器都可以包含多个凭据存储。

注意

如果您之前使用 vault 表达式对非敏感数据进行参数化，您必须将数据替换为 Elytron 安全属性。有关 Elytron 安全属性的更多信息，请参阅 Elytron 安全属性。

有关凭证存储的更多信息，请参阅 JBoss EAP 7.4 如何配置服务器安全性 中的凭据存储。https://access.redhat.com/documentation/zh-cn/red_hat_jboss_enterprise_application_platform/7.4/html-single/how_to_configure_server_security/#credential_store

9.2.1.1. 使用 WildFly Elytron 工具迁移 vault 数据

JBoss EAP 附带的 WildFly Elytron 工具提供了 vault 命令，可帮助您将 vault 内容迁移到凭据存储中。要执行 vault 命令，请在 EAP_HOME/bin 目录中运行 elytron-tool 脚本。

```
$ EAP_HOME/bin/elytron-tool.sh vault VAULT_ARGUMENTS
```

您可以使用以下命令获取所有可用参数的描述。

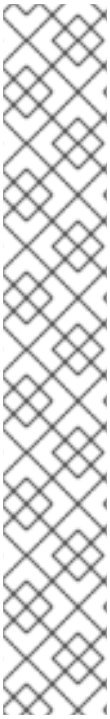
```
$ EAP_HOME/bin/elytron-tool.sh vault --help
```

重要

凭证存储仅用于保护密码。不支持管理模型中使用的 vault 表达式功能。如需更多信息，请参阅在 Elytron 中创建一个加密表达式

选择以下迁移选项之一：

- 将单个安全 Vault 迁移到凭证存储
- 将多个安全库迁移到批量的凭证存储



注意

- *WildFly Elytron 工具无法处理安全库数据文件的第一个版本。*
- *您可以使用 masked 格式输入 --keystore-password 参数，如下例所示迁移单个 vault 或明文。*
- *提供的 --salt 和 --iteration 参数用于提供解密已屏蔽密码的信息，或在输出中生成屏蔽的密码。如果省略了 --salt 和 --iteration 参数，则使用默认值。*
- *--summary 参数生成格式化的管理 CLI 命令，该命令可用于将转换的凭据存储添加到 JBoss EAP 配置中。纯文本密码在概述输出中被屏蔽。*

9.2.1.1.1. 将单个安全 Vault 迁移到凭证存储

使用以下示例将单个安全库迁移到凭证存储中：

示例：将单个安全库转换为凭证存储命令

```
$ EAP_HOME/bin/elytron-tool.sh vault --enc-dir vault_data/ --keystore vault-jceks.keystore --
keystore-password MASK-2hKo56F1a3jYGnJwhPmiF5 --iteration 34 --salt 12345678 --alias test --
location cs-v1.store --summary
```

此命令将安全库转换为凭据存储，并打印输出中用于转换的管理 CLI 命令的摘要。

```
Vault (enc-dir="vault_data/";keystore="vault-jceks.keystore") converted to credential store "cs-
v1.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="cs-v1.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-
2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
```

9.2.1.1.2. 将多个安全库迁移到批量的凭证存储

您可以使用 --bulk-convert 参数将多个库转换为凭证存储，并指向批量转换描述符文件。

先决条件

本节中的示例使用以下批量转换描述符文件。

示例： bulk-vault-conversion-descriptor.txt File

```
keystore:vault-v1/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1/vault_data/
salt:12345678
iteration:34
location:v1-cs-1.store
alias:test

keystore:vault-v1/vault-jceks.keystore
keystore-password:secretsecret
enc-dir:vault-v1/vault_data/
location:v1-cs-2.store
alias:test

# different vault vault-v1-more
keystore:vault-v1-more/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1-more/vault_data/
salt:12345678
iteration:34
location:v1-cs-more.store
alias:test
```

当遇到每个新 keystore: 行时，会启动新的转换。除 salt, iteration, 和 properties 外，所有选项均是必需的。

流程

1. 要执行批量转换并生成格式化管理 CLI 命令的输出，请执行以下命令：

```
$ EAP_HOME/bin/elytron-tool.sh vault --bulk-convert path/to/bulk-vault-conversion-descriptor.txt --summary
```

此命令将文件中指定的所有安全库转换为凭据存储，并打印用于在输出中转换的管理 CLI 命令的摘要。

```
Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential store "v1-cs-1.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-1.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
---------------------------------------

Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential store "v1-cs-2.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-2.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="secretsecret"})
---------------------------------------

Vault (enc-dir="vault-v1-more/vault_data/";keystore="vault-v1-more/vault-jceks.keystore") converted to credential store "v1-cs-more.store"
Vault Conversion summary:
---------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-more.store",implementation-properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
---------------------------------------
```

9.2.2. 将安全属性迁移到 Elytron

以下示例假定 group.name 和 encoding.algorithm 安全属性在传统的 security 子系统中被定义为 security-properties。

在 security 子系统中定义的安全属性：

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
    ...
    <security-properties>
        <property name="group.name" value="engineering-group" />
        <property name="encoding.algorithm" value="BASE64" />
    </security-properties>
</subsystem>
```

要在 elytron 子系统中定义这些安全属性，请使用以下管理 CLI 命令设置 elytron 子系统的 security-properties 属性：

```
/subsystem=elytron:write-attribute(name=security-properties, value={ group.name = "engineering-group", encoding.algorithm = "BASE64" })
```

这会在服务器配置文件中的 elytron 子系统中定义 security-properties。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
    <security-properties>
        <security-property name="group.name" value="engineering-group"/>
        <security-property name="encoding.algorithm" value="BASE64"/>
    </security-properties>
    ...
</subsystem>
```

上一命令中的 write-attribute 操作会覆盖现有属性。要在不影响其他安全属性的情况下添加或更改安全属性，请在管理 CLI 命令中使用 映射 操作：

```
/subsystem=elytron:map-put(name=security-properties, key=group.name, value=technical-support)
```

同样，您可以使用 map-remove 操作删除特定的安全属性：

```
/subsystem=elytron:map-remove(name=security-properties, key=group.name)
```

9.3. 迁移身份验证配置

本节提供有关将基于属性的身份验证和授权迁移到 Elytron 的信息。此外，还包括将 LDAP 身份验证配置、kerberos 身份验证配置、kerberos 身份验证、复合存储、JACC 安全性和使用缓存到 Elytron 的安全域的信息。

9.3.1. 将基于 PicketBox Properties 的配置迁移到 Elytron

使用以下示例，将基于 PicketBox 属性的身份验证迁移到 Elytron。

示例：基于 PicketBox Properties 的配置命令

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=UsersRoles, flag=Required, module-options=
{usersProperties=file://${jboss.server.config.dir}/example-users.properties,
rolesProperties=file://${jboss.server.config.dir}/example-roles.properties}}])
```

这会生成以下服务器配置。

示例：基于 PicketBox Properties 的安全域配置

```
<security-domain name="application-security">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties" value="file://${jboss.server.config.dir}/example-
users.properties"/>
      <module-option name="rolesProperties" value="file://${jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
  </authentication>
</security-domain>
```

9.3.1.1. 将基于属性的身份验证迁移到 Elytron

按照以下步骤将基于 PicketBox 属性的身份验证迁移到 Elytron。

前提条件

您计划迁移的部署的 Web 应用程序必须配置为需要基于表单的身份验证。应用程序正在引用 PicketBox 安全域，并使用 UsersRolesLoginModule 从 example-users.properties 和 example-

roles.properties 文件中加载用户信息。此流程还假设安全域在传统的 security 子系统中使用以下管理 CLI 命令定义。

确保您从基于 PicketBox 属性的身份验证开始。

流程

1. 在 elytron 子系统中定义引用 PicketBox 属性文件的新安全域。

```
/subsystem=elytron/properties-realm=application-properties:add(users-properties={path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-realm-name="Application Security"}, groups-properties={path=example-roles.properties, relative-to=jboss.server.config.dir}, groups-attribute=Roles)
```

2. 在 elytron 子系统中定义安全域子系统。

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-properties}], default-realm=application-properties, permission-mapper=default-permission-mapper)
```

这会在服务器配置文件中生成以下 elytron 子系统配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="application-properties" permission-mapper="default-permission-mapper">
      <realm name="application-properties"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="application-properties" groups-attribute="Roles">
      <users-properties path="example-users.properties" relative-to="jboss.server.config.dir" digest-realm-name="Application Security" plain-text="true"/>
      <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
  </security-realms>
  ...
</subsystem>
```

3. 将部署引用的应用安全域映射到 undertow 子系统中新定义的 HTTP 身份验证工厂。

```
/subsystem=undertow/application-security-domain=application-security:add(security-domain=application-security)
```

这会在服务器配置文件中生成以下 undertow 子系统配置：

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  ...
  <application-security-domains>
    <application-security-domain name="application-security" security-domain="application-security"/>
  </application-security-domains>
  ...
</subsystem>
```

4. 您必须重新加载服务器或重新部署应用程序，以便新的应用程序安全域映射生效。

身份验证现在配置为等同于 PicketBox 配置。

9.3.2. 将基于旧安全域的属性配置迁移到 Elytron

这部分论述了如何在 JBoss EAP 7.4 及更早版本中将用户、密码和组信息从属性文件加载到 Elytron 的传统安全域。这种类型的传统安全域通常用于保护管理接口或远程连接器。

在 JBoss EAP 8.0 中，filesystem-realm 优先于 properties-realm。

先决条件

您计划迁移的部署的 Web 应用程序必须配置为需要基于表单的身份验证。应用程序正在引用 PicketBox 安全域，并使用 UsersRolesLoginModule 从 example-users.properties 和 example-roles.properties 文件中加载用户信息。此流程还假设安全域在传统的 security 子系统中使用以下管理 CLI 命令定义。

示例：传统的安全 Realm 命令

```
/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-realm=ApplicationSecurity/authentication=properties:add(relative-to=jboss.server.config.dir, path=example-users.properties, plain-text=true)
/core-service=management/security-realm=ApplicationSecurity/authorization=properties:add(relative-to=jboss.server.config.dir, path=example-roles.properties)
```

这会生成以下服务器配置。

示例：传统安全 Realm 配置

```
<security-realm name="ApplicationSecurity">
  <authentication>
    <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-text="true"/>
  </authentication>
  <authorization>
    <properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
  </authorization>
</security-realm>
```

在应用服务器中添加 Elytron 安全性的原因之一是允许跨服务器使用一致的安全解决方案。将基于属性的传统安全域迁移到 Elytron 的初始步骤与用于将 PicketBox 属性验证迁移到 Elytron 的验证类似。按照以下步骤，将基于属性的传统安全域迁移到 Elytron。

流程

1. 在 elytron 子系统中定义引用属性文件的新安全域。

```
/subsystem=elytron/properties-realm=application-properties:add(users-properties=
{path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-
realm-name="Application Security"}, groups-properties={path=example-roles.properties,
relative-to=jboss.server.config.dir}, groups-attribute=Roles)
```

2. 在 elytron 子系统中定义安全域子系统。

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-
properties}], default-realm=application-properties, permission-mapper=default-permission-
mapper)
```

这会生成以下 Elytron 配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
 ...
 <security-domains>
  ...
  <security-domain name="application-security" default-realm="application-properties" permission-mapper="default-permission-mapper">
   <realm name="application-properties"/>
  </security-domain>
 </security-domains>
 <security-realms>
  ...
  <properties-realm name="application-properties" groups-attribute="Roles">
   <users-properties path="example-users.properties" relative-to="jboss.server.config.dir" digest-realm-name="Application Security" plain-text="true"/>
   <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
  </properties-realm>
 </security-realms>
  ...
</subsystem>
```

3. 定义 **sasl-authentication-factory**，以便传统安全域也可以用于简单身份验证安全层**(SASL)**身份验证。

```
/subsystem=elytron/sasl-authentication-factory=application-security-sasl:add(sasl-server-factory=elytron, security-domain=application-security, mechanism-configurations=[{mechanism-name=PLAIN}])
```

这会生成以下 **Elytron** 配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
 ...
 <sasl>
  ...
  <sasl-authentication-factory name="application-security-sasl" sasl-server-factory="elytron" security-domain="application-security">
   <mechanism-configuration>
    <mechanism mechanism-name="PLAIN"/>
   </mechanism-configuration>
  </sasl-authentication-factory>
  ...
 </sasl>
</subsystem>
```

4. 为 **SASL** 身份验证配置 **remoting connector**，并删除与传统安全域的关联。

```
/subsystem=remoting/http-connector=http-remoting-connector:write-attribute(name=sasl-authentication-factory, value=application-security-sasl)
```

- 这会在服务器配置文件的 remoting 子系统中生成以下配置。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  ...
  <http-connector name="http-remoting-connector" connector-ref="default" sasl-
authentication-factory="application-security-sasl"/>
</subsystem>
```

5. 添加两个身份验证工厂，以使用 Elytron 保护 http-interface。

```
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-
mechanism-factory=global, security-domain=application-security, mechanism-
configurations=[{mechanism-name=BASIC}])
/core-service=management/management-interface=http-interface:write-attribute(name=http-
upgrade.sasl-authentication-factory, value=application-security-sasl)
```

这会生成以下配置。

```
<management-interfaces>
  <http-interface http-authentication-factory="application-security-http">
    <http-upgrade enabled="true" sasl-authentication-factory="application-security-sasl"/>
    <socket-binding http="management-http"/>
  </http-interface>
</management-interfaces>
```

注意

在保护管理界面时，您应该将这些名称替换为这些示例中使用的名称。

现在完成将基于旧属性的配置迁移到 Elytron。

9.3.3. 使用 filesystem-realm 命令迁移到基于文件系统的安全 Realm

使用 elytron.sh 工具的 filesystem-realm 命令将基于文件系统的安全域迁移到 Elytron 中的基于文件系统的域。

基于文件系统的域是 Elytron 用于存储用户身份的基于文件系统的身份存储。filesystem-realm 命令将 properties-realm 文件转换为 filesystem-realm。它还生成将这个域和安全域添加到 elytron 子系统

的命令。

流程

1. *迁移属性文件。*

   您可以一次迁移单个 user-properties 文件，或者批量迁移属性文件。以下示例演示了两种类型的迁移的步骤。

   - *要迁移单个属性文件，请执行以下操作：*

     以下示例将具有关联 roles-properties 文件的单个 users-properties 文件转换为 filesystem-realm。示例假定旧的安全域具有以下 user-properties 和 role-properties 文件：

     ```
     example-users.properties
     example-roles.properties
     ```

     示例： Single user-property 文件迁移

     ```
     $./bin/elytron-tool.sh filesystem-realm --users-file example-users.properties --roles-file example-roles.properties --output-location realms/example
     ```

     这将创建 filesystem-realm 文件以及包含管理 CLI 命令的脚本。该脚本存储在 realms/example 目录中。

   - *要迁移多个属性文件，请执行以下操作：*

     以下示例将含有关联 roles-properties 文件的用户/操作文件转换为 filesystem-realm。示例假定旧的安全域具有以下属性文件：

     ```
     users-1.properties
     users-2.properties
     roles-1.properties
     roles-2.properties
     ```

要批量转换用户角色文件，必须创建一个描述符文件，以便与 filesystem-realm 命令搭配使用。在本例中，使用以下内容创建一个 /bin 目录中的描述符文件 example-descriptor-file ：

示例：描述符文件

```
users-file:/full/path/to/users-1.properties
roles-file:/full/path/to/roles-1.properties
output-location:./realms/bulk-1-example
filesystem-realm-name:exampleFileSystemRealm1
security-domain-name:exampleSecurityDomain1

users-file:/full/path/to/users-2.properties
roles-file:/full/path/to/roles-2.properties
output-location:./realms/bulk-2-example
filesystem-realm-name:exampleFileSystemRealm2
security-domain-name:exampleSecurityDomain2
```

描述符文件中的空白行用于分隔每个 users-properties 文件的操作。

以下示例使用描述符文件将两个带有关联 roles-properties 文件的 users-properties 文件转换为 filesystem-realm。

示例：模拟迁移

```
$./bin/elytron-tool.sh filesystem-realm --bulk-convert example-descriptor-file
```

这将创建 filesystem-realm 文件和包含管理 CLI 命令的脚本。脚本存储在描述符文件 output-location 属性中指定的目录中。

2. 使用 Elytron 工具生成的 CLI 命令，将新的安全域和安全域添加到 elytron 子系统。

示例：添加 filesystem-realm

```
/subsystem=elytron/filesystem-realm=converted-properties-filesystem-realm:add(path=/full/path/to/realms/example)

/subsystem=elytron/security-domain=converted-properties-security-domain:add(realms=[{realm=converted-properties-filesystem-realm}],default-realm=converted-properties-filesystem-realm,permission-mapper=default-permission-mapper)
```

9.3.4. 将 LDAP 身份验证配置迁移到 Elytron

将旧的 LDAP 身份验证迁移到 Elytron，以便它可以作为身份属性管理信息。

先决条件

在将旧的 LDAP 身份验证迁移到 Elytron 之前，您必须阅读 **Migrate Properties-based Authentication and Authorization to Elytron** 部分中的内容。这也适用于此处。您必须专注于如何定义安全域和身份验证工厂，以及如何映射它们以进行身份验证。本节不会重复这些指令，因此请确保在继续前通过该部分阅读。

以下示例假定组或角色信息直接从 LDAP 加载，并且传统的 LDAP 身份验证配置如下。

流程

1. LDAP 服务器包含以下用户和组条目：

示例：LDAP 服务器用户条目

```
dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
objectClass: top
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
userPassword: {SSHA}UG8ov2rnrnBKakcARVvraZHqTa7mFWJZlWt2HA==
```

-

示例：LDAP 服务器组条目

```
dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
```

为了进行身份验证，用户名与 **uid** 属性匹配，生成的组名则取自组条目的 **uid** 属性。

2. 与 **LDAP** 服务器和相关安全域的连接是使用以下管理 **CLI** 命令定义的：

示例：LDAP Security Realm Configuration 命令

```
batch
/core-service=management/ldap-
connection=MyLdapConnection:add(url="ldap://localhost:10389", search-
dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=LDAPRealm:add
/core-service=management/security-
realm=LDAPRealm/authentication=ldap:add(connection="MyLdapConnection", username-
attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")

/core-service=management/security-
realm=LDAPRealm/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=LDAPRealm/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org")
/core-service=management/security-realm=LDAPRealm/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", iterative=true, prefer-original-connection=true, principal-
attribute=uniqueMember, search-by=DISTINGUISHED_NAME, group-name=SIMPLE,
group-name-attribute=uid)
run-batch
```

这会生成以下服务器配置。

示例：LDAP Security Realm 配置

```
<management>
  <security-realms>
    ...
    <security-realm name="LDAPRealm">
      <authentication>
        <ldap connection="MyLdapConnection" base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
          <username-filter attribute="uid"/>
        </ldap>
      </authentication>
      <authorization>
        <ldap connection="MyLdapConnection">
          <username-to-dn>
            <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org"
attribute="uid"/>
          </username-to-dn>
          <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
            <group-to-principal search-by="DISTINGUISHED_NAME" base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org" prefer-original-connection="true">
              <membership-filter principal-attribute="uniqueMember"/>
            </group-to-principal>
          </group-search>
        </ldap>
      </authorization>
    </security-realm>
  </security-realms>
  <outbound-connections>
    <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-
dn="uid=admin,ou=system" search-credential="secret"/>
  </outbound-connections>
  ...
</management>
```

3. *以下管理 CLI 命令用于配置 PicketBox 安全域，它使用 LdapExtLoginModule 来验证用户名和密码。*

示例：安全域配置命令

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

这会生成以下服务器配置。

示例：安全域配置

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
 ...
 <security-domains>
  ...
  <security-domain name="application-security">
   <authentication>
    <login-module code="LdapExtended" flag="required">
     <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
     <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
     <module-option name="java.naming.security.authentication" value="simple"/>
     <module-option name="bindDN" value="uid=admin,ou=system"/>
     <module-option name="bindCredential" value="secret"/>
     <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
     <module-option name="baseFilter" value="(uid={0})"/>
     <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
     <module-option name="roleFilter" value="(uniqueMember={1})"/>
     <module-option name="roleAttributeID" value="uid"/>
    </login-module>
   </authentication>
  </security-domain>
 </security-domains>
</subsystem>
```

9.3.4.1. 将传统 LDAP 身份验证迁移到 Elytron

按照以下步骤，在 JBoss EAP 7.4 及更早版本中将前面的 LDAP 身份验证示例配置迁移到 Elytron。本节适用于迁移 传统安全 LDAP 域以及 PicketBox LDAP 安全域。

流程

1. 在 elytron 子系统中定义与 LDAP 的连接。

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin, ou=system", credential-reference={clear-text=secret})
```

2. 创建一个安全域来搜索 LDAP 并验证提供的密码。

```
/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
verification=true, identity-mapping={search-base-dn="ou=users, dc=group-to-principal,
dc=wildfly, dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,
dc=group-to-principal, dc=wildfly, dc=org", filter="(uniqueMember={1})", from="uid",
to="Roles"}]})
```

这些步骤会在服务器配置文件中生成以下 elytron 子系统配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
          <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
        </attribute-mapping>
      </identity-mapping>
    </ldap-realm>
  </security-realms>
  ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```

注意

默认情况下，如果没有为给定 security-domain 定义 role-decoder，则 "Roles" identity 属性映射到 identity 角色。

从 LDAP 加载的信息现在可以将身份作为属性关联。这些属性可以映射到角色，但也可以加载并用于其他目的。新创建的安全域可以在安全域中使用，方式与在本指南的 Migrate Properties-authentication 和 Authorization to Elytron 部分中描述。

9.3.5. 将数据库身份验证配置迁移到 Elytron

将基于 JDBC 数据源的 PicketBox 身份验证迁移到 Elytron。有关定义安全域、身份验证工厂和映射用于身份验证的说明，请参阅 迁移基于属性的身份验证和授权到 Elytron。

以下示例假定用户身份验证数据存储在使用类似以下示例的语法创建的数据库表中。

示例：创建数据库用户表的语法

```
CREATE TABLE User (
    id BIGINT NOT NULL,
    username VARCHAR(255),
    password VARCHAR(255),
    role ENUM('admin', 'manager', 'user'),
    PRIMARY KEY (id),
    UNIQUE (username)
)
```

出于身份验证目的，用户名与用户名列中存储的数据匹配，密码应以十六进制编码的 MD5 哈希存储在 password 列中，以及用于授权目的的用户角色存储在 role 列中。

PicketBox 安全域配置为使用 JBDC 数据源从数据库表中检索数据，然后使用它验证用户名和密码，并分配角色。使用以下管理 CLI 命令，假设 PicketBox 安全域已配置：

示例：PicketBox Database LoginModule Configuration Commands

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add( login-
modules=[ { code=Database, flag=Required, module-options={
dsJndiName="java:jboss/datasources/ExampleDS", principalsQuery="SELECT password FROM
User WHERE username = ?", rolesQuery="SELECT role, 'Roles' FROM User WHERE username =
?", hashAlgorithm=MD5, hashEncoding=base64 } } ] )
```

这会在传统的 security 子系统中生成以下 login-module 配置。

示例：PicketBox LoginModule Configuration

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security">
      <authentication>
        <login-module code="Database" flag="required">
          <module-option name="dsJndiName" value="java:jboss/datasources/ExampleDS"/>
          <module-option name="principalsQuery" value="SELECT password FROM User WHERE
username = ?"/>
          <module-option name="rolesQuery" value="SELECT role, 'Roles' FROM User WHERE
username = ?"/>
          <module-option name="hashAlgorithm" value="MD5"/>
          <module-option name="hashEncoding" value="base64"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

9.3.5.1. 将旧的数据库身份验证迁移到 Elytron

对于 JBoss EAP 7.4 及更早版本，您必须定义一个 JDBC 域，以启用 Elytron 的 JDBC 数据源访问，以便将前面的数据库身份验证示例配置迁移到 Elytron。

流程

1. 使用以下管理命令定义 **jdbc-realm**。

```
/subsystem=elytron/jdbc-realm=jdbc-realm:add(principal-query=[ { data-source=ExampleDS,
sql="SELECT role, password FROM User WHERE username = ?", attribute-mapping=[{index=1,
to=Roles } ] simple-digest-mapper={algorithm=simple-digest-md5, password-index=2} } ] )
```

这会在服务器配置文件的 elytron 子系统中生成以下 jdbc-realm 配置：

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <jdbc-realm name="jdbc-realm">
      <principal-query sql="SELECT role, password FROM User WHERE username = ?" data-
source="ExampleDS">
        <attribute-mapping>
          <attribute to="Roles" index="1"/>
        </attribute-mapping>
        <simple-digest-mapper password-index="2"/>
      </principal-query>
    </jdbc-realm>
    ...
  </security-realms>
  ...
</subsystem>
```

Elytron 现在使用 JDBC 域配置管理数据库身份验证。Elytron 比 PicketBox 更高效，因为它使用一个 SQL 查询来获取所有用户属性和凭证，然后从 SQL 结果中提取数据，并创建用于身份验证的属性映射。

9.3.6. 将 Kerberos 身份验证迁移到 Elytron

在使用 Kerberos 配置时，JBoss EAP 服务器可以依赖环境中的配置信息，或者使用系统属性来指定密钥配置。

这些系统属性适用于传统配置和迁移的 Elytron 配置。

示例：Kerberos 系统属性管理 CLI 命令

```
# Enable debugging
/system-property=sun.security.krb5.debug:add(value=true)
# Identify the Kerberos realm to use
/system-property=java.security.krb5.realm:add(value=ELYTRON.ORG)
# Identify the address of the KDC
/system-property=java.security.krb5.kdc:add(value=kdc.elytron.org)
```

示例：Kerberos 系统属性服务器配置

```
<system-properties>
  <property name="sun.security.krb5.debug" value="true"/>
  <property name="java.security.krb5.realm" value="ELYTRON.ORG"/>
  <property name="java.security.krb5.kdc" value="kdc.elytron.org"/>
</system-properties>
```

根据您的身份验证机制，选择以下迁移选项之一：

- *迁移 Kerberos HTTP 身份验证*
- *迁移 Kerberos 远程 SASL 身份验证*

9.3.6.1. 迁移 Kerberos HTTP 身份验证

在传统的安全配置中，您可以定义一个安全域来为 HTTP 管理界面启用 SPNEGO 身份验证，如下所示：

前提条件

以下示例假设 Kerberos 是使用系统属性配置的。

示例：为 HTTP 管理界面启用 SPNEGO 身份验证

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=HTTP\/test-server.elytron.org@ELYTRON.ORG:add(path=/path/to/test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-realm=true)
```

*示例：**Kerberos Security Realm** 配置*

```
<security-realms>
 ...
 <security-realm name="Kerberos">
  <server-identities>
   <kerberos>
    <keytab principal="HTTP/test-server.elytron.org@ELYTRON.ORG" path="/path/to/test-
server.keytab" debug="true"/>
   </kerberos>
  </server-identities>
  <authentication>
   <kerberos remove-realm="true"/>
  </authentication>
 </security-realm>
</security-realms>
```

*您还可以定义一对传统安全域，以允许应用程序使用 **Kerberos HTTP** 身份验证。*

示例：定义多个安全域

```
# Define the first security domain
/subsystem=security/security-domain=host:add
/subsystem=security/security-domain=host/authentication=classic:add
/subsystem=security/security-domain=host/authentication=classic/login-
module=1:add(code=Kerberos, flag=Required, module-options={storeKey=true, useKeyTab=true,
principal=HTTP/test-server.elytron.org@ELYTRON.ORG, keyTab=path/to/test-server.keytab,
debug=true}

# Define the second SPNEGO security domain
/subsystem=security/security-domain=SPNEGO:add
/subsystem=security/security-domain=SPNEGO/authentication=classic:add
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=1:add(code=SPNEGO, flag=requisite,  module-options={password-stacking=useFirstPass,
serverSecurityDomain=host})
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-module=1:write-
attribute(name=module, value=org.jboss.security.negotiation)
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=2:add(code=UsersRoles, flag=required, module-options={password-stacking=useFirstPass,
```

```
usersProperties=  /path/to/kerberos/spnego-users.properties, rolesProperties=
/path/to/kerberos/spnego-roles.properties, defaultUsersProperties=  /path/to/kerberos/spnego-
users.properties, defaultRolesProperties=  /path/to/kerberos/spnego-roles.properties})
```

示例：使用安全域对进行配置

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="host">
      <authentication>
        <login-module name="1" code="Kerberos" flag="required">
          <module-option name="storeKey" value="true"/>
          <module-option name="useKeyTab" value="true"/>
          <module-option name="principal" value="HTTP/test-server.elytron.org@ELYTRON.ORG"/>
          <module-option name="keyTab" value="/path/to/test-server.keytab"/>
          <module-option name="debug" value="true"/>
        </login-module>
      </authentication>
    </security-domain>
    <security-domain name="SPNEGO">
      <authentication>
        <login-module name="1" code="SPNEGO" flag="requisite"
module="org.jboss.security.negotiation">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="serverSecurityDomain" value="host"/>
        </login-module>
        <login-module name="2" code="UsersRoles" flag="required">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="usersProperties" value="path/to/kerberos/spnego-users.properties"/>
          <module-option name="rolesProperties" value="  /path/to/kerberos/spnego-roles.properties"/>
          <module-option name="defaultUsersProperties" value="  /path/to/kerberos/spnego-
users.properties"/>
          <module-option name="defaultRolesProperties" value="  /path/to/kerberos/spnego-
roles.properties"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

旧应用程序会部署引用 SPNEGO 安全域并使用 SPNEGO 机制的安全。

9.3.6.1.1. 将 Kerberos HTTP 身份验证迁移到 Elytron

使用安全域和 Kerberos 安全工厂保护 Elytron 中的管理界面和应用程序。

前提条件

以下示例假设 Kerberos 是使用系统属性配置的。

流程

1. 定义用于加载身份信息的安全域。

```
/subsystem=elytron/properties-realm=spnego-properties:add(users-properties=
{path=path/to/spnego-users.properties, plain-text=true, digest-realm-
name=ELYTRON.ORG}, groups-properties={path=path/to/spnego-roles.properties})
```

2. 定义 Kerberos 安全工厂，允许服务器加载自己的 Kerberos 身份。

```
/subsystem=elytron/kerberos-security-factory=test-server:add(path=path/to/test-
server.keytab, principal=HTTP/test-server.elytron.org@ELYTRON.ORG, debug=true)
```

3. 定义安全域以拉取策略以及身份验证策略的 HTTP 身份验证工厂。

```
/subsystem=elytron/security-domain=SPNEGODomain:add(default-realm=spnego-
properties, realms=[{realm=spnego-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=spnego-http-authentication:add(security-
domain=SPNEGODomain, http-server-mechanism-factory=global,mechanism-
configurations=[{mechanism-name=SPNEGO, credential-security-factory=test-server}])
```

这会在服务器配置文件的 elytron 子系统中生成以下配置：

示例：迁移的 Elytron 配置

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
  ...
    <security-domain name="SPNEGODomain" default-realm="spnego-properties"
```

```
permission-mapper="default-permission-mapper">
      <realm name="spnego-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="spnego-properties">
      <users-properties path="path/to/spnego-users.properties" digest-realm-
name="ELYTRON.ORG" plain-text="true"/>
      <groups-properties path="path/to/spnego-roles.properties"/>
    </properties-realm>
  </security-realms>
  <credential-security-factories>
    <kerberos-security-factory name="test-server" principal="HTTP/test-
server.elytron.org@ELYTRON.ORG" path="path/to/test-server.keytab" debug="true"/>
  </credential-security-factories>
  ...
  <http>
    ...
    <http-authentication-factory name="spnego-http-authentication" http-server-mechanism-
factory="global" security-domain="SPNEGODomain">
      <mechanism-configuration>
        <mechanism mechanism-name="SPNEGO" credential-security-factory="test-server"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
</subsystem>
```

4. 为保护应用，请在 **undertow** 子系统中定义应用安全域，将安全域映射到此 **http-authentication-factory**。可以更新 **HTTP** 管理界面，以引用此配置中定义的 **http-authentication-factory**。此过程记录在 **Migrate Properties- authentication and Authorization to Elytron** 中。

9.3.6.2. 迁移 Kerberos 远程 SASL 身份验证

使用以下信息迁移 **Kerberos** 远程 **SASL** 身份验证：

流程

1. 为用于远程身份验证的 **Kerberos / GSSAPI SASL** 身份验证定义旧的安全域，如原生管理接口。

示例：用于删除管理 CLI 命令的 Kerberos 身份验证

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=remote\/test-server.elytron.org@ELYTRON.ORG:add(path=path/to/remote-test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-realm=true)
```

示例：Kerberos Remoting Security Realm 配置

```
<management>
  <security-realms>
    ...
    <security-realm name="Kerberos">
      <server-identities>
        <kerberos>
          <keytab principal="remote/test-server.elytron.org@ELYTRON.ORG" path="path/to/remote-test-server.keytab" debug="true"/>
        </kerberos>
      </server-identities>
      <authentication>
        <kerberos remove-realm="true"/>
      </authentication>
    </security-realm>
  </security-realms>
  ...
</management>
```

9.3.6.2.1. 将 Kerberos 远程 SASL 身份验证迁移到 Elytron

使用以下步骤将 Kerberos 远程 SASL 身份验证迁移到 Elytron。

流程

1. 定义用于加载身份信息的安全域。

```
/path=kerberos:add(relative-to=user.home, path=src/kerberos)
/subsystem=elytron/properties-realm=kerberos-properties:add(users-properties=
```

```
{path=kerberos-users.properties, relative-to=kerberos, digest-realm-name=ELYTRON.ORG},
groups-properties={path=kerberos-groups.properties, relative-to=kerberos})
```

2. 为服务器的身份定义 **Kerberos** 安全工厂。

```
/subsystem=elytron/kerberos-security-factory=test-server:add(relative-to=kerberos,
path=remote-test-server.keytab, principal=remote/test-server.elytron.org@ELYTRON.ORG)
```

3. 定义安全域和 **SASL** 身份验证工厂。

```
/subsystem=elytron/security-domain=KerberosDomain:add(default-realm=kerberos-
properties, realms=[{realm=kerberos-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
/subsystem=elytron/sasl-authentication-factory=gssapi-authentication-factory:add(security-
domain=KerberosDomain, sasl-server-factory=elytron, mechanism-configurations=
[{mechanism-name=GSSAPI, credential-security-factory=test-server}])
```

这会在服务器配置文件的 **elytron** 子系统中生成以下配置：

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="KerberosDomain" default-realm="kerberos-properties" permission-
mapper="default-permission-mapper">
      <realm name="kerberos-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="kerberos-properties">
      <users-properties path="kerberos-users.properties" relative-to="kerberos" digest-realm-
name="ELYTRON.ORG"/>
      <groups-properties path="kerberos-groups.properties" relative-to="kerberos"/>
    </properties-realm>
  </security-realms>
  <credential-security-factories>
    <kerberos-security-factory name="test-server" principal="remote/test-
server.elytron.org@ELYTRON.ORG" path="remote-test-server.keytab" relative-to="kerberos"/>
  </credential-security-factories>
  ...
  <sasl>
    ...
    <sasl-authentication-factory name="gssapi-authentication-factory" sasl-server-factory="elytron"
security-domain="KerberosDomain">
      <mechanism-configuration>
        <mechanism mechanism-name="GSSAPI" credential-security-factory="test-server"/>
      </mechanism-configuration>
```

```
    </sasl-authentication-factory>
    ...
  </sasl>
</subsystem>
```

验证

现在可以更新管理界面或远程连接器来引用 SASL 身份验证工厂。

这里定义的两个 Elytron 示例也可以合并为使用共享安全域和安全域，并且只使用 特定于协议的身份验证因素，各自引用自己的 Kerberos 安全工厂。

其他资源

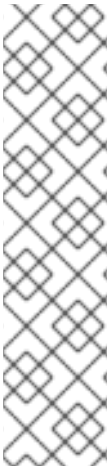
这些步骤用于定义等同的 Elytron 配置，与 迁移 Kerberos HTTP 身份验证 中所述的步骤非常相似。

9.3.7. 将 Composite Stores 迁移到 Elytron

本节论述了如何将使用多个身份存储的 PicketBox 或 传统的安全域 配置迁移到 Elytron Aggregate Security Realm 配置。

使用 PicketBox 或旧的安全域时，可以在其中定义一个配置，当用于授权的信息从不同的存储加载时。迁移到 Elytron 时，可以使用聚合安全域来实现。

以下示例使用 example-users.properties 属性文件执行用户身份验证，然后查询 LDAP 来加载组和角色信息。



注意

显示的配置基于以下部分中的示例，它提供了额外的背景信息：

- 将基于 PicketBox Properties 的配置迁移到 Elytron
- 将 LDAP 身份验证配置迁移到 Elytron

9.3.7.1. PicketBox Composite Store 配置

此情境的 PicketBox 安全域使用以下管理 CLI 命令进行配置。

示例：PicketBox Configuration Commands

```
/subsystem=security/security-domain=application-security:add

/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[ {code=UsersRoles, flag=Required, module-options={ password-stacking=useFirstPass,
usersProperties=file://${jboss.server.config.dir}/example-users.properties}} {code=LdapExtended,
flag=Required, module-options={ password-stacking=useFirstPass,
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org",roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

这会生成以下服务器配置。

示例：PicketBox Security Domain Configuration

```
<security-domain name="application-security">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="usersProperties" value="file://${jboss.server.config.dir}/example-
users.properties"/>
    </login-module>
    <login-module code="LdapExtended" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
      <module-option name="java.naming.security.authentication" value="simple"/>
      <module-option name="bindDN" value="uid=admin,ou=system"/>
      <module-option name="bindCredential" value="secret"/>
      <module-option name="baseCtxDN" value="ou=users,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="baseFilter" value="(uid={0})"/>
      <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="roleFilter" value="(uniqueMember={1})"/>
      <module-option name="roleAttributeID" value="uid"/>
    </login-module>
  </authentication>
</security-domain>
```

如需更多信息，请参阅 Elytron Aggregate Security Realm Configuration 以了解如何配置聚合安全域。

9.3.7.2. 传统的安全性 Realm Composite Store 配置

对于 JBoss EAP 7.4 及更早版本的 releses，使用以下管理 CLI 命令配置旧的安全域配置：

注意

因为旧的安全命令不适用于 JBoss EAP 8，这些命令仅适用于 JBoss EAP 7.4 及更早版本。

示例：传统的安全 Realm 配置命令

```
/core-service=management/ldap-connection=MyLdapConnection:add(url="ldap://localhost:10389",
search-dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-
realm=ApplicationSecurity/authentication=properties:add(path=example-users.properties, relative-
to=jboss.server.config.dir, plain-text=true)

batch
/core-service=management/security-
realm=ApplicationSecurity/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",
iterative=true, prefer-original-connection=true, principal-attribute=uniqueMember, search-
by=DISTINGUISHED_NAME, group-name=SIMPLE, group-name-attribute=uid)
run-batch
```

这会生成以下服务器配置。

示例：传统安全 Realm 配置

```
<security-realms>
  ...
  <security-realm name="ApplicationSecurity">
    <authentication>
      <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-
text="true"/>
    </authentication>
    <authorization>
      <ldap connection="MyLdapConnection">
        <username-to-dn>
          <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org" attribute="uid"/>
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
          <group-to-principal search-by="DISTINGUISHED_NAME" base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org" prefer-original-connection="true">
            <membership-filter principal-attribute="uniqueMember"/>
          </group-to-principal>
        </group-search>
      </ldap>
    </authorization>
  </security-realm>
</security-realms>
<outbound-connections>
  <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-dn="uid=admin,ou=system"
search-credential="secret"/>
</outbound-connections>
```

有关如何在 **elytron** 子系统中配置聚合安全域来完成此操作，请参阅 **Elytron Aggregate Security Realm 配置**。

9.3.7.3. Elytron Aggregate Security Realm 配置

此情境对应的 **Elytron** 配置使用以下管理 **CLI** 命令进行配置。

示例： **Elytron** 配置命令

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin,ou=system", credential-reference={clear-text=secret})

/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-verification=true,
identity-mapping={search-base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org", rdn-
```

```
identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",filter="(uniqueMember={1})",from="uid",to="Roles"}]})

/subsystem=elytron/properties-realm=application-properties:add(users-properties={path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-realm-name="Application Security"})

/subsystem=elytron/aggregate-realm=combined-realm:add(authentication-realm=application-properties, authorization-realm=ldap-realm)

/subsystem=elytron/security-domain=application-security:add(realms=[{realm=combined-realm}], default-realm=combined-realm, permission-mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-mechanism-factory=global, security-domain=application-security, mechanism-configurations=[{mechanism-name=BASIC}])
```

这会生成以下服务器配置。

示例： Elytron 配置

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="combined-realm" permission-mapper="default-permission-mapper">
      <realm name="combined-realm"/>
    </security-domain>
  </security-domains>
  <security-realms>
    <aggregate-realm name="combined-realm" authentication-realm="application-properties" authorization-realm="ldap-realm"/>
      ...
      <properties-realm name="application-properties">
        <users-properties path="example-users.properties" relative-to="jboss.server.config.dir" digest-realm-name="Application Security" plain-text="true"/>
      </properties-realm>
      <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
        <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org">
          <attribute-mapping>
            <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
          </attribute-mapping>
        </identity-mapping>
      </ldap-realm>
```

```
  </security-realms>
  ...
  <http>
    ...
    <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="BASIC"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```

在 elytron 子系统中，定义了 aggregate-realm，用于指定用于身份验证的安全域以及用于授权决策的安全域。

9.3.8. 将使用缓存的安全域迁移到 Elytron

使用 PicketBox 时，可以定义一个安全域并启用内存缓存来访问。这样，您可以在内存中访问身份数据，并避免额外的直接访问身份存储。可以通过 Elytron 实现类似的配置。本节显示在使用 Elytron 时示例 PicketBox 配置和等同的安全域缓存配置。

9.3.8.1. PicketBox Cached Security Domain Configuration

以下命令显示 PicketBox 安全域配置，以便在 JBoss EAP 7.4 及更早版本中启用缓存。

示例：PicketBox Cached Security Domain Commands

```
/subsystem=security/security-domain=application-security:add(cache-type=default)
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
```

```
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

这会生成以下服务器配置。

示例：PicketBox Cached Security Domain Configuration

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security" cache-type="default">
      <authentication>
        <login-module code="LdapExtended" flag="required">
          <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
          <module-option name="java.naming.security.authentication" value="simple"/>
          <module-option name="bindDN" value="uid=admin,ou=system"/>
          <module-option name="bindCredential" value="secret"/>
          <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="baseFilter" value="(uid={0})"/>
          <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="roleFilter" value="(uniqueMember={1})"/>
          <module-option name="roleAttributeID" value="uid"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

注意

此命令和生成的配置与将 LDAP 身份验证配置迁移到 Elytron 中的示例类似，但此处的 cache-type 属性由 default 的值来定义。默认缓存类型是内存缓存。使用 PicketBox 时，您还可以指定 infinispan 的 cache-type，但这种类型不支持 Elytron。

9.3.8.2. 配置 Elytron 缓存安全域

按照以下步骤创建使用 Elytron 时缓存安全域的类似的配置。

流程

1. **定义安全域，并将安全域嵌套在缓存域中。然后，缓存域可以在安全域中使用，然后在身份验证工厂中使用。**

示例： Elytron Security Realm Configuration 命令

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin,ou=system", credential-reference={clear-text=secret})
/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
verification=true, identity-mapping={search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",filter="(uniqueMember=
{1})",from="uid",to="Roles"}]})
/subsystem=elytron/caching-realm=cached-ldap:add(realm=ldap-realm)
```

2. **定义安全域和 HTTP 身份验证工厂，它使用上一步中定义的 cached-ldap 域。**

示例：Elytron Security Domain and Authentication Factory Configuration Commands

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=cached-ldap}],
default-realm=cached-ldap, permission-mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-
mechanism-factory=global, security-domain=application-security, mechanism-
configurations=[{mechanism-name=BASIC}])
```

注意

您必须引用 caching-realm 而不是原始域。否则，会绕过缓存。

这些命令会在服务器配置中添加以下内容：

示例： Elytron Cached Security Domain configuration

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="cached-ldap" permission-
mapper="default-permission-mapper">
      <realm name="cached-ldap"/>
    </security-domain>
  </security-domains>
  ...
  <security-realms>
    ....
  <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
          <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
        </attribute-mapping>
      </identity-mapping>
    </ldap-realm>
    <caching-realm name="cached-ldap" realm="ldap-realm"/>
  </security-realms>
  ...
  <http>
    ...
    <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="BASIC"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
   ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
  ...
```

9.3.9. 将 Jakarta 授权安全性迁移到 Elytron

默认情况下，JBoss EAP 7.4 及更早版本使用传统的 security 子系统配置 Jakarta 授权策略提供程序和工厂。默认配置映射从 PicketBox 映射到实施。

elytron 子系统根据容器的 Java 授权合同(JACC)规范提供内置的策略提供程序。

有关如何在 elytron 子系统中为容器策略提供程序启用和定义 Java 授权合同的信息，请参阅 JBoss EAP 7.4 开发指南中的 定义 Jakarta 身份验证策略提供程序。

9.4. 迁移应用程序客户端

本节介绍如何将客户端应用程序迁移到 Elytron。

将 Naming Client 配置迁移到 Elytron

本节论述了如何使用 org.jboss.naming.remote.client.InitialContext 类（由 org.jboss.naming.remote.client.InitialContext 类支持）将执行远程 JNDI 查找的客户端应用程序迁移到 Elytron。

以下示例假定通过指定用户凭据以及它所连接的命名供应商的 URL 来创建 InitialContextFactory 类。

示例：之前版本中使用的 InitialContext 代码

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
properties.put(Context.PROVIDER_URL,"http-remoting://127.0.0.1:8080");
properties.put(Context.SECURITY_PRINCIPAL, "bob");
properties.put(Context.SECURITY_CREDENTIALS, "secret");
InitialContext context = new InitialContext(properties);
Bar bar = (Bar) context.lookup("foo/bar");
...
```

您可以从以下迁移方法之一进行选择：

- 使用配置文件方法迁移 Naming 客户端
- 使用编程方法迁移命名客户端

9.4.1. 使用配置文件方法迁移命名客户端

使用配置方法将您的命名客户端迁移到 Elytron。

流程

1. 在客户端应用程序 META-INF/ 目录中创建 wildfly-config.xml 文件。该文件应包含建立与命名提供程序连接时使用的用户凭据。

   示例： wildfly-config.xml 文件

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.7">
    <authentication-rules>
      <rule use-configuration="namingConfig">
        <match-host name="127.0.0.1"/>
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="namingConfig">
        <set-user-name name="bob"/>
        <credentials>
          <clear-password password="secret"/>
        </credentials>
      </configuration>
    </authentication-configurations>
  </authentication-client>
</configuration>
```

2. 如下例所示，创建一个 InitialContext。请注意，InitialContext 由 org.wildfly.naming.client.WildFlyInitialContextFactory 类支持。

   示例： InitialContext 代码

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextFactory");
properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
InitialContext context = new InitialContext(properties);
Bar bar = (Bar) context.lookup("foo/bar");
...
```

9.4.2. 使用编程方法迁移命名客户端

使用此方法，您可以提供用于在应用程序代码中直接建立连接的用户凭证。

示例：使用编程方法的代码

```
// Create the authentication configuration
AuthenticationConfiguration namingConfig =
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), namingConfig);

// Create a callable that creates and uses an InitialContext
Callable<Void> callable = () -> {
    Properties properties = new Properties();

properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextFactory");
    properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
    InitialContext context = new InitialContext(properties);
    Bar bar = (Bar) context.lookup("foo/bar");
    ...
    return null;
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

9.4.3. 将 Jakarta Enterprise Beans 客户端迁移到 Elytron

此迁移示例假定客户端应用已配置为通过 jboss-ejb-client.properties 文件调用部署到远程服务器的 Jakarta Enterprise Beans。此文件位于客户端应用程序 META-INF/ 目录中，包含连接到远程服务器所需的以下信息：

示例： jboss-ejb-client.properties 文件

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=127.0.0.1
remote.connection.default.port = 8080
remote.connection.default.username=bob
remote.connection.default.password=secret
```

客户端使用类似以下示例的代码查找 Jakarta Enterprise Beans 并调用其方法之一。

示例：调用远程 Jakarta Enterprise Bean 的客户端代码

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

// Look up the Jakarta Enterprise Beans and invoke one of its methods
RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
    "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
int sum = statelessRemoteCalculator.add(101, 202);
```

您可以从以下迁移方法之一进行选择：

- 使用配置文件迁移 Jakarta Enterprise Beans 客户端
- 以编程方式迁移 Jakarta Enterprise Beans 客户端

9.4.3.1. 使用配置文件迁移 Jakarta Enterprise Beans 客户端

按照以下步骤，使用配置方法将您的命名客户端迁移到 Elytron。

流程

1. 在客户端应用程序 META-INF/ 目录中配置 wildfly-config.xml 文件。该文件应包含建立与命名提供程序连接时使用的用户凭据。

示例： wildfly-config.xml 文件

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.7">
    <authentication-rules>
      <rule use-configuration="ejbConfig">
        <match-host name="127.0.0.1"/>
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="ejbConfig">
        <set-user-name name="bob"/>
        <credentials>
          <clear-password password="secret"/>
        </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>
```

2. 如下例所示，创建一个 InitialContext。请注意，InitialContext 由 org.wildfly.naming.client.WildFlyInitialContextFactory 类支持。

示例： InitialContext 代码

```
// Create an InitialContext
```

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextFactory");
InitialContext context = new InitialContext(properties);

// Look up an Jakarta Enterprise Beans and invoke one of its methods
// Note that this code is the same as before
RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
    "ejb:/ejb-remote-server-side//CalculatorBean!" +
RemoteCalculator.class.getName());
int sum = statelessRemoteCalculator.add(101, 202);----
```

3. 现在，您可以删除过时的 jboss-ejb-client.properties 文件，因为不再需要该文件。

9.4.3.2. 以编程方式迁移 Jakarta Enterprise Beans 客户端

使用以下步骤以编程方式迁移 Jakarta Enterprise Beans 客户端。

流程

- 提供直接在应用程序代码中直接连接到远程服务器所需的信息。

示例：使用编程方法的代码

```
// Create the authentication configuration
AuthenticationConfiguration ejbConfig =
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), ejbConfig);

// Create a callable that invokes the Jakarta Enterprise Beans
Callable<Void> callable = () -> {

   // Create an InitialContext
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
   properties.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
   InitialContext context = new InitialContext(properties);

   // Look up the Jakarta Enterprise Beans and invoke one of its methods
```

```
    // Note that this code is the same as before
    RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
        "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
    int sum = statelessRemoteCalculator.add(101, 202);
    ...
    return null;
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

现在，您可以删除过时的 jboss-ejb-client.properties 文件，因为不再需要该文件。

9.5. 迁移 SSL 配置

将应用程序中的 SSL 配置迁移到将 Elytron 与以下信息搭配使用：

将简单 SSL 配置迁移到 Elytron

如果您使用安全域保护到 JBoss EAP 服务器的 HTTP 连接，请使用本节中提供的信息将 SSL 配置迁移到 Elytron。

先决条件

使用安全域，具有到 JBoss EAP 服务器的安全 HTTP 连接。

以下示例假定您在 security-realm 中配置了以下 keystore ：

示例：使用安全域密钥存储进行 SSL 配置

```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
password="keystore_password" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
</security-realm>
```

完成以下步骤，使用 Elytron 实现相同的配置。

流程

1. 在 elytron 子系统中创建一个 key-store，用于指定密钥存储的位置及其加密的密码。此命令假定密钥存储是使用密钥tool 命令生成的，其类型是 JKS。

```
/subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
```

2. 在 elytron 子系统中创建一个 key-manager，用于指定上一步中定义的 key-store、别名和密码。

```
/subsystem=elytron/key-manager=LocalhostKeyManager:add(key-store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-text="key_password"})
```

3. 在 elytron 子系统中创建 server-ssl-context，它引用了上一步中定义的 key-manager。

```
/subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-manager=LocalhostKeyManager)
```

4. 将 https-listener 从旧的 security-realm 切换到新创建的 Elytron ssl-context。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context,value=LocalhostSslContext)
run-batch
```

5. 重新加载服务器。

```
reload
```

这会在服务器配置文件中生成以下 elytron 子系统配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" ...>
  ...
  <tls>
    <key-stores>
      <key-store name="LocalhostKeyStore">
        <credential-reference clear-text="keystore_password"/>
        <implementation type="JKS"/>
        <file path="server.keystore" relative-to="jboss.server.config.dir"/>
      </key-store>
    </key-stores>
    <key-managers>
      <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore"  alias-
filter="server">
        <credential-reference clear-text="key_password"/>
      </key-manager>
    </key-managers>
    <server-ssl-contexts>
      <server-ssl-context name="LocalhostSslContext" key-manager="LocalhostKeyManager"/>
    </server-ssl-contexts>
  </tls>
</subsystem>
```

这会在服务器配置文件中生成以下 undertow 子系统配置：

```
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
```

如需更多信息，请参阅 JBoss EAP 7.4 如何配置 服务器安全性 中的 Elytron 子系统 以及如何保护管理接口。

9.5.1. 将 CLIENT-CERT SSL 身份验证迁移到 Elytron

要启用 CLIENT-CERT SSL 身份验证，请在 authentication 元素中添加 truststore 元素。

```
<security-realm name="ManagementRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
password="KEYSTORE_PASSWORD" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
  <authentication>
    <truststore path="server.truststore" relative-to="jboss.server.config.dir" keystore-
password="TRUSTSTORE_PASSWORD" />
    <local default-user="$local"/>
    <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
  </authentication>
</security-realm>
```

注意

如果没有发生 CLIENT-CERT 身份验证，客户端可以回退到使用本地机制 或用户名/密码 身份验证机制。要使基于 CLIENT-CERT 的身份验证强制使用，请删除 local 和 properties 元素。

传统的 truststore 可以通过两种方式使用：

- 传统的 truststore 只包括 CA
- 传统的 truststore 包括客户端的证书

9.5.1.1. 传统的 truststore 只包括 CA

按照以下步骤配置服务器，以防止没有有效证书和私钥的用户使用 Elytron 访问服务器。

流程

1. 在 elytron 子系统中创建一个 key-store，用于指定密钥存储的位置及其加密的密码。此命令假定密钥存储是使用密钥tool 命令生成的，其类型是 JKS。

```
/subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
```

2. 在 elytron 子系统中创建一个 key-store，用于指定信任存储的位置以及它加密的密码。此命令假定密钥存储是使用密钥tool 命令生成的，其类型是 JKS。

```
/subsystem=elytron/key-store=TrustStore:add(path=server.truststore,relative-to=jboss.server.config.dir,credential-reference={clear-text="truststore_password"},type=JKS)
```

3. 在 elytron 子系统中创建 key-manager，用于指定之前定义的 LocalhostKeyStore 密钥存储、别名和密码。

```
/subsystem=elytron/key-manager=LocalhostKeyManager:add(key-store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-text="key_password"})
```

4.

4. 在 elytron 子系统中创建 trust-manager，用于指定之前创建的信任存储的 key-store。

```
/subsystem=elytron/trust-manager=TrustManager:add(key-store=TrustStore)
```

5. 在 elytron 子系统中创建 server-ssl-context，它引用之前定义的 key-manager，设置 trust-manager 属性并启用客户端身份验证。

```
/subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-manager=LocalhostKeyManager,trust-manager=TrustManager,need-client-auth=true)
```

6. 将 https-listener 更新至新创建的 Elytron ssl-context。

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context,value=LocalhostSslContext)
```

7. 重新加载服务器。

```
reload
```

这会在服务器配置文件中生成以下 elytron 子系统配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0"...>
 ...
 <tls>
  <key-stores>
   <key-store name="LocalhostKeyStore">
    <credential-reference clear-text="keystore_password"/>
    <implementation type="JKS"/>
    <file path="server.keystore" relative-to="jboss.server.config.dir"/>
   </key-store>
   <key-store name="TrustStore">
    <credential-reference clear-text="truststore_password"/>
    <implementation type="JKS"/>
    <file path="server.truststore" relative-to="jboss.server.config.dir"/>
   </key-store>
  </key-stores>
  <key-managers>
   <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore" alias-filter="server">
    <credential-reference clear-text="key_password"/>
   </key-manager>
  </key-managers>
  <trust-managers>
   <trust-manager name="TrustManager" key-store="TrustStore"/>
  </trust-managers>
```

```
    <server-ssl-contexts>
      <server-ssl-context name="LocalhostSslContext" need-client-auth="true" key-
manager="LocalhostKeyManager" trust-manager="TrustManager"/>
    </server-ssl-contexts>
  </tls>
</subsystem>
```

这会在服务器配置文件中生成以下 **undertow** 子系统配置：

```
<subsystem xmlns="urn:jboss:domain:undertow:14.0">
...
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
...
</subsystem>
```

9.5.1.2. 安全域和域

安全域用于两个情况：

- 当证书身份验证失败时，安全域将用于密码回退。
- 为密码和证书完成授权后，域将提供单个用户的角色。

要允许使用预定义的 **Elytron ManagementDomain** 安全域和管理 **Realm** 安全域，用户存储在标准属性文件中。

```
<security-domains>
    <security-domain name="ManagementDomain" default-realm="ManagementRealm" permission-
mapper="default-permission-mapper">
        <realm name="ManagementRealm" role-decoder="groups-to-roles"/>
        <realm name="local"/>
    </security-domain>
</security-domains>
<security-realms>
    <properties-realm name="ManagementRealm">
        <users-properties path="mgmt-users.properties" relative-to="jboss.server.config.dir" digest-
realm-name="ManagementRealm"/>
        <groups-properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
</security-realms>
```

因此，对于任何客户端证书，用户必须存在于安全域中。

9.5.1.3. 主体 Decoder

当使用证书身份验证且安全域接受用于解析身份的用户名时，必须定义从客户端证书获取 用户名 的方法。

在这种情况下，在证书主题中使用 CN 属性。

```
/subsystem=elytron/x500-attribute-principal-decoder=x500-decoder:add(attribute-name=CN)
```

9.5.1.4. HTTP 身份验证工厂

对于 HTTP 连接，会使用之前定义的资源来定义 HTTP 身份验证工厂。它被配置为支持 CLIENT_CERT 和 DIGEST 身份验证。

由于属性域只验证密码，且无法验证客户端证书，因此您需要首先添加配置机制工厂。这禁用对安全域的证书验证。

```
/subsystem=elytron/configurable-http-server-mechanism-factory=configured-cert:add(http-server-mechanism-factory=global, properties={org.wildfly.security.http.skip-certificate-verification=true})
```

HTTP 验证可以创建：

```
./subsystem=elytron/http-authentication-factory=client-cert-digest:add(http-server-mechanism-factory=configured-cert,security-domain=ManagementDomain,mechanism-configurations=[{mechanism-name=CLIENT_CERT,pre-realm-principal-transformer=x500-decoder},{mechanism-name=DIGEST, mechanism-realm-configurations=[{realm-name=ManagementRealm}]}])
```

以上命令会产生：

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
  ...
  <http>
    ...
    <http-authentication-factory name="client-cert-digest" http-server-mechanism-factory="configured-cert" security-domain="ManagementDomain">
      <mechanism-configuration>
        <mechanism mechanism-name="CLIENT_CERT" pre-realm-principal-transformer="x500-decoder"/>
        <mechanism mechanism-name="DIGEST">
          <mechanism-realm realm-name="ManagementRealm"/>
        </mechanism>
```

```
    </mechanism-configuration>
  </http-authentication-factory>
  ...
  <configurable-http-server-mechanism-factory name="configured-cert" http-server-mechanism-
factory="configured-cert">
      <properties>
          <property name="org.wildfly.security.http.skip-certificate-verification" value="true"/>
      </properties>
  </configurable-http-server-mechanism-factory>
  ...
 </http>
 ...
</subsystem>
```

9.6. LDAP 中的旧安全行为更改

在 Red Hat JBoss Enterprise Application Platform 8.0 中，对 LDAP 有显著变化。这些更改包括对不可访问 LDAP 域的 HTTP 状态更改，为从 DN 进行角色解析启用 LDAP 安全域，以及将 JBoss EAP SSL 证书发送到 LDAP 服务器的更改。

- 在利用 Elytron 子系统的 JBoss EAP 8.0 中，当 LDAP 域无法访问时，会返回 "500 Internal Error"，表示无法访问域的 HTTP 状态更改。

- 在 JBoss EAP 8.0 中，您可以启用 LDAP 安全域来从 DN 解析角色。通过从 LDAP 加载信息作为与身份关联的属性，这些属性可使用 attribute-mapping 元素映射到角色。

  配置示例

  ```
  <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
    <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
  principal,dc=wildfly,dc=org">
      <attribute-mapping>
        <attribute from="dn" to="Roles" filter="(uniqueMember={1})" filter-base-
  dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      </attribute-mapping>
    </identity-mapping>
  </ldap-realm>
  ```

- 在 JBoss EAP 8.0 中，您可以选择配置出站 LDAP 连接，以使用双向或 mutual TLS 将 JBoss EAP SSL 证书发送到 LDAP 服务器。如需更多信息，请参阅为管理界面和应用程序启用

双向 SSL/TLS。

附录 A. 参考材料

作为 JBoss EAP 用户，您可以期望在不同版本间的无缝兼容性和互操作性。支持从各种客户端连接到服务器，只有特定情况下需要额外的注意事项。

A.1. 版本间的兼容性和互操作性

本节介绍 JBoss EAP 6、JBoss EAP 7 和 JBoss EAP 8.0 版本之间的客户端和服务器企业 Bean 和消息传递组件的兼容性和互操作性。

A.1.1. Enterprise Bean 通过互联网间移动协议

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 6 服务器

A.1.2. 使用 Java 命名和目录接口进行企业 Beans

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器

JBoss EAP 6 支持 Enterprise Beans 3.1 规范，并引进了使用标准化全局 Java 命名和目录接口命名空间，它们仍然在 JBoss EAP 8.0 中使用。Java Naming 和 Directory Interface 命名空间名称的更改不

会为以下配置引入不兼容：

- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 或 JBoss EAP 7 服务器
- 从 JBoss EAP 8.0 或 JBoss EAP 7 客户端连接到 JBoss EAP 6 服务器

A.1.3. 使用 @WebService 进行企业 Bean 远程

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 6 服务器

A.1.4. 消息传递独立客户端

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器

JBoss EAP 8.0 内置消息传递无法连接到 JBoss EAP 6 附带的 HornetQ 2.3.x，因为协议兼容性问题。因此，以下配置不兼容：

- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 6 服务器

- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 服务器

注意

要使此连接成为可能，您必须创建一个传统的连接工厂，可通过 Java 命名和目录接口访问。

A.1.5. 消息传递 MDB

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器
- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器

JBoss EAP 8.0 内置消息传递无法连接到 JBoss EAP 6 附带的 HornetQ 2.3.x，因为协议兼容性问题。因此，以下配置不兼容：

- 从 JBoss EAP 8.0 客户端连接到 JBoss EAP 6 服务器
- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 服务器

注意

要使此连接成为可能，您必须创建一个传统的连接工厂，可通过 Java 命名和目录接口访问。

A.1.6. 消息传递网桥

以下配置必须在没有错误的情况下运行：

- 从 JBoss EAP 6 客户端连接到 JBoss EAP 8.0 服务器

- *从 JBoss EAP 7 客户端连接到 JBoss EAP 8.0 服务器*
- *从 JBoss EAP 8.0 客户端连接到 JBoss EAP 7 服务器*
- *从 JBoss EAP 8.0 客户端连接到 JBoss EAP 6 服务器*