



JBoss 企业级应用程序平台 6.4

迁移指南

适用于红帽 JBoss 企业版应用程序平台 6

JBoss 企业级应用程序平台 6.4 迁移指南

适用于红帽 JBoss 企业版应用程序平台 6

法律通告

Copyright © 2015 Red Hat, Inc..

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本书是从以前的红帽 JBoss 企业级应用程序平台移植应用程序的指南。

目录

第 1 章 介绍	5
1.1. 关于 RED HAT JBOSS 企业版应用程序平台 6	5
1.2. 关于移植指南	5
第 2 章 准备移植	6
2.1. 准备移植	6
2.2. 查看 JBOSS EAP 6 里的新功能以及不同之处	6
2.3. 复查已丢弃和不被支持的功能的列表	8
第 3 章 移植您的应用程序	10
3.1. 多数应用程序要求的修改	10
3.1.1. 复查多数程序所要求的修改	10
3.1.2. 类加载的修改	10
3.1.2.1. 根据类加载的变化更新应用程序	10
3.1.2.2. 了解模块依赖关系	10
3.1.2.3. 根据类加载的修改更新应用程序的依赖关系	11
3.1.3. 配置文件的修改	11
3.1.3.1. 创建或修改控制 JBoss EAP 6.0 里类加载的文件	11
3.1.3.2. jboss-deployment-structure.xml	15
3.1.3.3. 新模块化类加载系统的软件包资源	15
3.1.3.4. 修改 ResourceBundle 属性的位置	15
3.1.3.5. 创建自定义模块	16
3.1.4. 日志的修改	17
3.1.4.1. 修改日志依赖关系	17
3.1.4.2. 为第三方日志框架更新应用程序代码	17
3.1.4.3. 修改代码以使用新的 JBoss Logging 框架	19
3.1.5. 应用程序包的修改	20
3.1.5.1. 修改 EAR 和 WAR 的打包	20
3.1.6. 数据源和资源适配器配置的修改	21
3.1.6.1. 根据配置的变化更新应用程序	21
3.1.6.2. 更新数据源配置	21
3.1.6.3. 安装和配置 JDBC 驱动	22
3.1.6.4. 为 Hibernate 或 JPA 配置数据源	26
3.1.6.5. 更新资源适配器配置	26
3.1.7. 安全性的修改	28
3.1.7.1. 应用程序安全性的修改	28
3.1.7.2. 更新使用 PicketLink STS 和 Web Services 的应用程序	28
3.1.8. JNDI 的修改	29
3.1.8.1. 更新应用程序 JNDI 命名空间的名称	29
3.1.8.2. 可移植的 EJB JNDI 名称	30
3.1.8.3. 复核 JNDI 命名空间规则	31
3.1.8.4. 修改应用程序以遵循新的 JNDI 命名规则	31
3.1.8.5. 以前版本的 JNDI 命名空间示例和它们在 JBoss EAP 6 里是如何指定的	32
3.1.9. HTTP/HTTPS/AJP 连接器属性映射	32
3.1.9.1. HTTP/HTTPS/AJP 连接器属性映射	32
3.2. 依赖于应用程序架构和组件的修改	37
3.2.1. 复查依赖于应用程序架构和组件的修改	37
3.2.2. Hibernate 和 JPA 的修改	38
3.2.2.1. 更新使用 Hibernate 和/或 JPA 的应用程序	38
3.2.2.2. 修改使用 Hibernate 和 JPA 的应用程序的配置	38
3.2.2.3. 持久化单元属性	40
3.2.2.4. 更新 Hibernate 3 应用程序以使用 Hibernate 4	41

3.2.2.5. Hibernate 标识符自动生成值	42
3.2.2.6. 移植您的 Hibernate 3.3.x 应用程序到 Hibernate 4.x	42
3.2.2.7. 移植您的 Hibernate 3.5.x 应用程序到 Hibernate 4.x	43
3.2.2.8. 修改运行在群集环境里的移植的 Seam 和 Hibernate 应用程序的持久化属性	44
3.2.2.9. 更新您的应用程序以遵循 JPA 2.0 规格	44
3.2.2.10. 用 Infinispan 替换 JPA/Hibernate 二级缓存	45
3.2.2.11. Hibernate Cache 属性	46
3.2.2.12. 移植到 Hibernate Validator 4	47
3.2.3. JSF 的修改	48
3.2.3.1. 启用应用程序以使用更旧版本的 JSF	48
3.2.4. Web Services 的修改	49
3.2.4.1. Web Services 的修改	49
3.2.5. JAX-RS 和 RESTEasy 的修改	51
3.2.5.1. 配置 JAX-RS 和 RESTEasy 的修改	51
3.2.6. LDAP Security Realm 的修改	52
3.2.6.1. 配置 LDAP Security Realm 的修改	53
3.2.7. HornetQ 的修改	54
3.2.7.1. 关于 HornetQ 和 NFS	54
3.2.7.2. 配置 JMS 桥以移植现有的 JMS 消息到 JBoss EAP 6	54
3.2.7.3. 创建 JMS 桥	55
3.2.7.4. 移植您的应用程序以将 HornetQ 用作 JMS 提供者	59
3.2.7.5. 用 HornetQ 配置消息系统	60
3.2.8. 群集的修改	60
3.2.8.1. 修改应用程序以用于群集环境	60
3.2.8.2. 实现 HA 单点登录	64
3.2.9. 服务风格的部署的修改	70
3.2.9.1. 更新使用服务风格部署的应用程序	70
3.2.10. 远程调用的修改	70
3.2.10.1. 将进行远程调用的 JBoss EAP 5 应用程序移植到 JBoss EAP 6	70
3.2.10.2. 用 JNDI 远程调用 Session Bean	72
3.2.10.3. EJB JNDI 命名引用	75
3.2.11. EJB 2.x 的修改	76
3.2.11.1. 更新使用 EJB 2.x 的应用程序	76
3.2.12. JBoss AOP 的修改	81
3.2.12.1. 更新使用 JBoss AOP 的应用程序	81
3.2.13. 移植 Seam 2.2 应用程序	82
3.2.13.1. 移植 Seam 2.2 归档到 JBoss EAP 6	82
3.2.13.2. Seam 2.2 归档移植的问题	85
3.2.14. 移植 Spring 应用程序	88
3.2.14.1. 移植 Spring 应用程序	88
3.2.15. 其他影响移植的修改	88
3.2.15.1. 熟悉其他可能影响到移植的修改	88
3.2.15.2. 修改 Maven 插件名称	88
3.2.15.3. 修改客户端应用程序	88
第 4 章 工具和提示	89
4.1. 协助移植的资源	89
4.1.1. 可协助移植的资源	89
4.1.2. 熟悉可以协助移植的工具	89
4.1.3. 使用 Tattletale 来查找应用程序的依赖关系	89
4.1.4. 下载和安装 Tattletale	90
4.1.5. 创建和复查 Tattletale 报告	90
4.1.6. 使用 IronJacamar 工具来移植数据源和资源适配器配置	90

4.1.7. 下载和安装 IronJacamar 移植工具	91
4.1.8. 使用 IronJacamar 移植工具来转换数据源配置文件	91
4.1.9. 使用 IronJacamar 移植工具来转换资源适配器配置文件	93
4.2. 调试移植的问题	98
4.2.1. 调试和解决移植问题	98
4.2.2. 调试和解决 ClassNotFoundExceptions 和 NoClassDefFoundErrors	98
4.2.3. 查找 JBoss 模块依赖关系	98
4.2.4. 在以前的安装里查找 JAR	99
4.2.5. 调试和解决 ClassCastException	100
4.2.6. 调试和解决 DuplicateServiceException	100
4.2.7. 调试和解决 JBoss Seam 调试页面的错误	101
4.3. 复查例程的移植	102
4.3.1. 复查例程的移植	102
4.3.2. 移植 Seam 2.2 JPA 例程到 JBoss EAP 6	103
4.3.3. 移植 Seam 2.2 Booking 例程到 JBoss EAP 6	104
4.3.4. 移植 Seam 2.2 Booking 例程到 JBoss EAP 6：逐步说明	108
4.3.5. 构建和部署 Seam 2.2 Booking 例程的 JBoss EAP 5.X 版本。	108
4.3.6. 调试和解决 Seam 2.2 Booking 例程的部署错误和异常	109
4.3.7. 调试和解决 Seam 2.2 Booking 例程的运行时错误和异常	117
4.3.8. 当移植 Seam 2.2 Booking 例程时所作修改的总结	121
附录 A. 修订记录	123

第 1 章 介绍

1.1. 关于 RED HAT JBOSS 企业版应用程序平台 6

Red Hat JBoss 企业版应用程序平台 6 (JBoss EAP 6) 是一个构建在开放标准上并和 Java EE 6 规格兼容的中间件平台。它集成了 JBoss Application Server 7 和高可用性的群集、消息系统、分布式缓存以及其他技术。

JBoss EAP 6 使用了新的模块化结构，允许在有需要时才启用服务，从而提高了启动速度。

管理控制台和管理命令行界面使您不需要再编辑 XML 配置文件并增添了使用脚本和自动化任务的能力。

此外，JBoss EAP 6 包含了 API 和开放框架以用于快速开发安全和可扩充的 Java EE 应用程序。

[提交 bug 报告](#)

1.2. 关于移植指南

JBoss EAP 6 是一个快速的、轻量级的、功能强大的 Java EE 6 规格的实现。它的架构基于模块化服务容器并按需启用服务。由于这个新的架构，运行在 JBoss EAP 5 上的应用程序需要进行修改以运行在 JBoss EAP 6 上。

本指南的目的是为在 JBoss EAP 6 上成功运行和部署 JBoss EAP 5.1 应用程序需要进行的修改编写文档。它提供了如何解决部署和运行时问题的信息，以及如何防止程序行为发生变化的信息。这是移植到新平台的第一步。一旦应用程序成功地部署和运行了，您就可以开始计划升级单独的组件以使用 JBoss EAP 6 的新功能和特征。

[提交 bug 报告](#)

第 2 章 准备移植

2.1. 准备移植

因为应用服务器的结构和以前不一样，在试图移植应用程序之前，您可能想进行一些研究和规划。

1. 查看 JBoss EAP 6 里的新功能以及不同之处

这个版本里已经修改了大量的内容，可能会影响到 JBoss EAP 5 应用程序的开发。这包括对文件目录结构、脚本、部署配置、类加载和 JNDI 查找的修改。详情请参考 [第 2.2 节“查看 JBoss EAP 6 里的新功能以及不同之处”](#)。

2. 阅读关于开发应用程序起步的文档

请务必阅读《JBoss EAP 6 开发指南》里的『开发应用程序起步』章节：https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/。它包含下列重要的信息：

- Java EE 6
- 新的模块化类加载系统
- 文件结构的改变
- 如何下载和安装 JBoss EAP 6
- 如何下载和安装 JBoss Developer Studio
- 如何为您的开发环境配置 Maven
- 如何下载和运行产品附带的 Quickstart 例程。

3. 学习在 Maven 项目里如何使用 JBoss EAP 6 依赖关系

请务必阅读《JBoss EAP 6 开发指南》里的『Maven 指南』章节：https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/。『管理项目依赖关系』部分包含配置项目使用 JBoss EAP Bill of Material (BOM) artifacts 的重要信息。

4. 分析和理解您的应用程序

每个应用程序都是独特的，在移植前您必须彻底了解现有应用程序的相关组件和架构。



重要

在修改应用程序之前，请确保已进行了备份。

[提交 bug 报告](#)

2.2. 查看 JBOSS EAP 6 里的新功能以及不同之处

介绍

下面是 JBoss EAP 6 和以前版本的显著不同的列表。

基于模块的类加载

在 JBoss EAP 5 里，类加载架构是层次结构的。而在 JBoss EAP 6 里，类加载基于 JBoss 模块。这提供了真正的应用程序隔离，隐藏了服务器实现类，且只加载应用程序所需的类。类加载并行具有更

高的性能。针对 JBoss EAP 5 编写的应用程序必须进行修改以指定模块依赖关系，且在某些情况下需要重新打包归档文件。关于更多的信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的《类加载与模块》章节。

域管理

在 JBoss EAP 6 里，服务器可以作为独立服务器或者以受管域运行。在受管域里，您可以一次配置整个服务器组，从而在整个服务器网络里保持配置的同步。虽然这应该不会影响为以前版本构建的应用程序，但它可以简化对多个服务器的部署的管理。关于更多的信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 管理和配置指南》里的《关于受管域》章节。

部署配置

独立服务器和受管域

JBoss EAP 5 使用基于配置集的部署配置。这些配置集位于 **EAP_HOME/server/** 目录。应用程序经常会包含多个配置文件以用于安全性、数据库、资源适配器和其他配置。在 JBoss EAP 6 里，部署配置是通过使用一个文件实现的。这个文件用来配置部署使用的所有服务和子系统。独立服务器是使用 **EAP_HOME/standalone/configuration/standalone.xml** 文件来配置的。对于运行在受管域里的服务器，服务器使用 **EAP_HOME/domain/configuration/domain.xml** 文件来配置。包含在多个 JBoss EAP 5 配置文件里的信息必须移植到新的单个配置文件里。

部署顺序

JBoss EAP 6 的部署使用快速的、并发的初始化，从而提高了性能和效率。在多数情况下，应用服务器能够自动提前决定依赖关系并选择最有效的部署策略。然而，由多个模块组成的 JBoss EAP 5 的应用程序部署为 EAR 并使用传统的 JNDI 查找而不是 CDI 注入或 resource-ref 条目，这样就可能要求修改相关配置。

目录结构和脚本

如之前提到的，JBoss EAP 6 不再使用基于配置集的部署配置，所以不再有 **EAP_HOME/server/** 目录了。用于独立服务器的配置文件现在位于 **EAP_HOME/standalone/configuration/** 目录而部署位于 **EAP_HOME/standalone/deployments/** 目录。而对于运行在受管域里的服务器，配置文件位于 **EAP_HOME/domain/configuration/** 目录。

在 JBoss EAP 5 里，Linux 脚本 **EAP_HOME/bin/run.sh** 或 Windows 脚本 **EAP_HOME/bin/run.bat** 用来启动服务器。而在 JBoss EAP 6 里，服务器启动脚本依赖于您运行服务器的方式。Linux 脚本 **EAP_HOME/bin/standalone.sh** 或 Windows 脚本 **EAP_HOME/bin/standalone.bat** 用于启动独立服务器。Linux 脚本 **EAP_HOME/bin/domain.sh** 或 Windows 脚本 **EAP_HOME/bin/domain.bat** 用于启动受管域。

JNDI 查找

JBoss EAP 6 现在使用标准化的可移植的 JNDI 命名空间。为 JBoss EAP 5 编写的使用 JNDI 查找的应用程序必须进行修改以符合新的标准化 JNDI 命名空间格式。关于 JNDI 命名语法的详情，请参考第 3.1.8.2 节“可移植的 EJB JNDI 名称”。

关于其他的信息，请查看

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的《JBoss EAP 6 里新的和已修改的功能》。

[提交 bug 报告](#)

2.3. 复查已丢弃和不被支持的功能的列表

在您移植应用程序前，您应该意识到以前版本的 JBoss EAP 里的一些功能已被丢弃且不会被支持了。关于完整的列表，请参考客户门户

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 发行注记》里的『不被支持的功能』章节。

下面是不被支持的功能的简短概述。

服务器启动的 -b 命令行参数

在以前的版本里，JBoss EAP 自动使用启动参数 **-b** 指定的地址而不是 IP 地址。在 JBoss EAP 6 里，服务器的 `<inet-address>` 配置会寻找用匹配的 IP 地址配置的网络接口。这虽可以用于 `127.0.0.1`，却不能用于 `127.*.*.*`。如果您用 **-b** 参数启动 JBoss EAP 6 服务器并绑定 `127.*.*.*` IP 地址，您必须首先将服务器配置文件里的 `<inet-address>` 修改为 `<loopback-address>`。

关于如何用管理 CLI 配置服务器的信息，请参考客户门户

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 里《JBoss EAP 管理和配置指南》的『管理 CLI 的操作』章节。

EJB 依赖关系

在以前的 JBoss EAP 版本里，服务的 EJB 依赖关系，以及其他 EJB，都可以用 `jboss.xml` 部署描述符文件里的 `<depends>` 标签来指定。例如：

```
<depends>jboss.j2ee:jndiName=com/myorg/app/Foo,service=EJB</depends>
<depends>jboss.mq.destination:service=Queue,name=queue/HelloworldQueue</depends>
```

在 JBoss EAP 6 里，您必须使用 `@EJB` 注解来注入 EJB 引用及 `@Resource` 注解来访问数据源和其他资源。例如：

```
@EJB(lookup="java:global/MyApp/FooImpl!com.myorg.app.Foo")
@Resource(mappedName = "java:/queue/HelloworldQueue")
```

JNDI 查找也已修改，详情请参考本指南的『JNDI 的修改』章节。

关于在 EJB 引用的更多信息，请参考客户门户

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss 开发指南》里的『EJB 引用解析』章节。

HTTPInvoker

在以前的 JBoss EAP 版本里，我们可以使用 HTTPInvoker 来配置 EJB、JNDI 或 JMS 以使用 HTTP 协议。JBoss EAP 6 已不支持这种做法。

HA Singleton 部署和 BarrierController 服务

HA Singleton 服务确保了在群集里只有一个服务实例运行。

JBoss EAP 5 提供了对多重 HA Singleton 部署策略的支持，包括 HASingletonDeployer 服务、使用 HASingletonController 的 POJO 部署以及使用 BarrierController 服务的 HASingleton 部署。当群集里的不同节点启动、停止、不可用时，这些策略都依赖于 HAPartition 进行通知。

在 JBoss EAP 6 里，HA Singleton 部署已经完全改变了。Singleton Deployer 只在模块化服务容器（Modular Service Container, MSC）上操作。通过 SingletonService，目标服务被安装在群集上的每

个节点上，但在给定时刻只会一个节点上启动。这个方法简化了部署要求并最小化了在节点间移动 Singleton 主服务所需的时间。然而，它要求您编写自定义代码来实现相同的功能。本版本附带的 JBoss EAP quickstart 例程里由一个 HA Singleton 部署示例。关于 HA Singleton 的更多信息，请参考 [第 3.2.8.2 节“实现 HA 单点登录”](#)。

[提交 bug 报告](#)

第 3 章 移植您的应用程序

3.1. 多数应用程序要求的修改

3.1.1. 复查多数程序所要求的修改

JBoss EAP 6 里的类加载和配置修改将影响几乎所有的应用程序。JBoss EAP 6 也使用标准的可移植的 JNDI 语法。这些修改将影响多数程序，所以我们推荐您在移植应用程序前先复查下面的信息。

1. [第 3.1.2.1 节 “根据类加载的变化更新应用程序”](#)
2. [第 3.1.6.1 节 “根据配置的变化更新应用程序”](#)
3. [第 3.1.8.1 节 “更新应用程序 JNDI 命名空间的名称”](#)

[提交 bug 报告](#)

3.1.2. 类加载的修改

3.1.2.1. 根据类加载的变化更新应用程序

在 JBoss EAP 6 里，模块化类加载是一个明显的变化，它将影响几乎所有的程序。在移植应用程序之前，请先查看下面的信息。

1. 首先，查看您的应用程序的软件包结构及其依赖关系。详情请参考 [第 3.1.2.3 节 “根据类加载的修改更新应用程序的依赖关系”](#)。
2. 如果您的应用程序使用了日志，您需要制定正确的模块依赖关系。请查看 [第 3.1.4.1 节 “修改日志依赖关系”](#) 里的相关细节。
3. 由于模块化类加载的修改，您可能需要修改 EAR 或 WAR 的软件包结构。详情请参考 [第 3.1.5.1 节 “修改 EAR 和 WAR 的打包”](#)。

[提交 bug 报告](#)

3.1.2.2. 了解模块依赖关系

介绍

模块只能访问自己的类，以及它具有显性或隐性依赖关系的任何模块上的类。

过程 3.1. 了解模块依赖关系

1. 了解隐性依赖关系

服务器里的部署者隐性地自动添加一些常用的模块依赖关系，如 `javax.api` 和 `sun.jdk`。这使得类在运行时对于部署可见，并让开发人员不用显性地添加依赖关系。关于如何和何时添加这些隐性的依赖关系，请查看 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的『类记载和模块』章节的『隐性模块依赖关系』。

2. 了解显性依赖关系

对于其他类，模块必须显性地指定，否则缺失的依赖关系会导致部署或运行时错误。如果缺失了依赖关系，您可以在服务器日志里看到 `ClassNotFoundException` 或 `NoClassDefFoundErrors` 跟踪信息。如果多于一个模块加载了相同的 JAR 或一个模块加载的

类扩展了不同模块加载的类，您会在服务器日志里看到 **ClassCastException**。要显性地制定依赖关系，请修改 **MANIFEST.MF** 或创建一个 JBoss 专有的部署描述符文件 **jboss-deployment-structure.xml**。关于模块依赖关系的更多信息，请查看 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 《JBoss EAP 6 开发指南》里的『开发应用程序起步』章节的『类记载和模块概述』。

[提交 bug 报告](#)

3.1.2.3. 根据类加载的修改更新应用程序的依赖关系

介绍

JBoss EAP 6 里的类加载和以前版本的很不一样。类加载现在基于 JBoss Modules 项目。它不是一个加载所有 JAR 到普通 Class path 里的单一的、层级的类加载器，而是每个库都成为一个模块，链接它所依赖的模块。JBoss EAP 6 里的部署也是模块，它们不能访问应用服务器里 JAR 中定义的类，除非在这些类上显性地定义了依赖关系。应用服务器里定义的一些模块依赖关系是自动设置的。例如，如果您在部署一个 Java EE 应用程序，Java EE API 的依赖关系将被自动或隐性地添加到您的模块里。关于自动添加的依赖关系的完整列表，请参考

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》的『类加载和模块』章节里的『隐性模块依赖关系』。

任务

当您移植程序到 JBoss EAP 6 时，由于模块化类加载的变化，您可能需要执行一个或多个下列任务：

- [第 3.1.2.2 节“了解模块依赖关系”](#)
- [第 4.1.3 节“使用 Tattletale 来查找应用程序的依赖关系”](#)
- [第 3.1.3.1 节“创建或修改控制 JBoss EAP 6.0 里类加载的文件”](#)
- [第 3.1.3.3 节“新模块化类加载系统的软件包资源”](#)

[提交 bug 报告](#)

3.1.3. 配置文件的修改

3.1.3.1. 创建或修改控制 JBoss EAP 6.0 里类加载的文件

介绍

由于 JBoss EAP 6 里对于使用模块化类加载的修改，您可能需要修改或创建一个或多个文件来添加依赖关系或阻止对自动依赖关系的加载。关于类加载和类加载次序的更多信息，请参考

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的『类加载与模块』章节。

下列文件用于控制 JBoss EAP 6 的类加载。

jboss-web.xml

如果您已经在 **jboss-web.xml** 文件里定义了一个 **<class-loading>** 元素，您需要删除它。JBoss EAP 5 里使用的这个行为现在是 EAP 6 里的默认类加载行为，所以不是必需的了。如果您没有删除这个元素，您会在服务器日志里看到一个 **ParseError** 和 **XMLStreamException**。

下面是 **jboss-web.xml** 文件里已注释的 **<class-loading>** 元素的例子。

```
<!DOCTYPE jboss-web PUBLIC
```



```

"-//JBoss//DTD Web Application 4.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
  <class-loading java2ClassLoadingCompliance="false">
    <loader-repository>
      seam.jboss.org:loader=MyApplication
    </loader-repository>
  </class-loading>
</jboss-web>

```

MANIFEST.MF

手动编辑的

根据您的应用程序使用的组件或模块，您可能需要添加一个或多个以来关系到这个文件里。您可以添加它们为 **Dependencies** 或 **Class-Path** 条目。

下面是开发人员编辑的 **MANIFEST.MF** 例子：

```

Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar

```

如果您修改了这个文件，请确保在文件结尾包含一个换行符号。

用 Maven 生成的

如果您使用 Maven，您需要修改您的 **pom.xml** 文件以生成用于 **MANIFEST.MF** 的依赖关系。如果您的应用程序使用了 EJB 3.0，您可能在 **pom.xml** 文件里有如下内容：

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
  </configuration>
</plugin>

```

如果 EJB 3.0 代码使用了 **org.apache.commons.log**，您需要在 **MANIFEST.MF** 文件里有这个以来关系。要生成这个以来关系，请添加 **<plugin>** 元素到 **pom.xml** 文件里：

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
    <archive>
      <manifestFile>src/main/resources/META-

```



```
INF/MANIFEST.MF</manifestFile>
    </archive>
</configuration>
</plugin>
```

在上面的例子里，**src/main/resources/META-INF/MANIFEST.MF** 文件只需要包含以来关系条目：

```
Dependencies: org.apache.commons.logging
```

Maven 将生成完整的 **MANIFEST.MF** 文件：

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

jboss-deployment-structure.xml

这个文件是 JBoss 专有的部署描述符，它可以用来对类加载的细颗粒度控制。就像 **MANIFEST.MF** 一样，这个文件可以用来添加依赖关系。它也可以阻止添加自动依赖关系、定义额外的模块、修改 EAR 部署的隔离类加载行为，以及在模块里添加其他的资源根目录。

下面是一个 **jboss-deployment-structure.xml** 文件，它添加 JSF 1.2 模块的依赖关系且阻止了 JSF 2.0 模块的自动加载。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

关于这个文件的其他信息，请参考 [第 3.1.3.2 节 “jboss-deployment-structure.xml”](#)。

application.xml

在以前的 JBoss EAP 版本里，您通过 **jboss-app.xml** 控制 EAR 里部署的顺序。现在不是这样了。Java EE6 规格在 **application.xml** 文件里提供了 **<initialize-in-order>** 元素来控制 EAR 里的 Java EE 模块的部署顺序。

在大多数情况下，您不须要指定部署顺序。如果您的应用程序使用了依赖关系注入和 `resource-refs` 来引入外部模块的组件，多数情况下都不要要求 `<initialize-in-order>` 元素，因为应用服务器能够隐性地确定正确和优化的组件顺序。

让我们假设您有一个包含 `myBeans.jar` 且 `myApp.war` 打包在 `myApp.ear` 里的应用程序。`myApp.war` 使用 `@EJB` 注解从 `myBeans.jar` 注入一个 bean。在这个例子里，应用服务器懂得如何确保 EJB 组件在 servlet 启动之前可用，而您不需要使用 `<initialize-in-order>` 元素。

然而，如果这个 servlet 使用了如下的传统 JNDI 查找风格的远程引用来访问 bean，您可能需要指定模块的顺序。

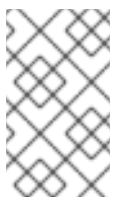
```
init() {
    Context ctx = new InitialContext();
    ctx.lookup("TheBeanInMyBeansModule");
}
```

在这种情况下，服务器不能确定 EJB 组件处于 `myBeans.jar` 里，您需要强制 `myBeans.jar` 里的组件被初始化并在 `myApp.war` 里的组件之前启动。为此，您可以设置 `<initialize-in-order>` 元素为 `true` 并指定 `application.xml` 文件里的 `myBeans.jar` 和 `myApp.war` 的顺序。

下面是一个使用 `<initialize-in-order>` 元素来控制部署顺序的例子。`myBeans.jar` 在 `myApp.war` 文件之前被部署。

```
<application xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="6"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/application_6.xsd">
    <application-name>myApp</application-name>
    <initialize-in-order>true</initialize-in-order>
    <module>
        <ejb>myBeans.jar</ejb>
    </module>
    <module>
        <web>
            <web-uri>myApp.war</web-uri>
            <context-root>myApp</context-root>
        </web>
    </module>
</application>
```

`application.xml` 文件的 schema 可以在这里找到：http://java.sun.com/xml/ns/javaee/application_6.xsd。



注意

您应该意识到设置 `<initialize-in-order>` 元素为 `true` 会减慢部署速度。我们更倾向于使用依赖注入或 `resource-refs` 来定义正确的依赖关系，因为它们使容器在优化部署时更具灵活性。

jboss-ejb3.xml

`jboss-ejb3.xml` 部署描述符替换 `jboss.xml` 以覆盖和添加 Java EE 定义的 `ejb-jar.xml` 里提供的功能。这个新文件和 `jboss.xml` 兼容，而目前的部署里已经忽略了 `jboss.xml`。

login-config.xml

login-config.xml 文件不再用于安全配置。现在安全性是在服务器配置文件的 **<security-domain>** 元素里配置。这个文件是 **standalone/configuration/standalone.xml**，如果您是在受管域里运行服务器，则是 **domain/configuration/domain.xml**。

[提交 bug 报告](#)

3.1.3.2. jboss-deployment-structure.xml

jboss-deployment-structure.xml 是 JBoss EAP 6 的一个新的可选的部署描述符。这个部署描述符提供了对部署里的类加载的控制。

这个部署描述符的 XML schema 位于 **EAP_HOME/docs/schema/jboss-deployment-structure-1_2.xsd**。

[提交 bug 报告](#)

3.1.3.3. 新模块化类加载系统的软件包资源

介绍

在以前的 JBoss EAP 版本里，**WEB-INF/** 里面的所有资源都添加到了 WAR classpath。在 JBoss EAP 6 里，web 应用程序的 artifact 只从 **WEB-INF/classes** 和 **WEB-INF/lib** 目录进行加载。没有在指定位置打包应用程序 artifact 会导致 **ClassNotFoundException**, **NoClassDefError** 或运行时错误。

要解决这些类加载的错误，您必须修改应用程序归档的结构或定义一个自定义模块。

修改资源打包

要使资源只对应用程序可用，您必须将属性文件、JAR 或其他具有 WAR 的 artifact 移至 **WEB-INF/classes/** 或 **WEB-INF/lib/** 目录来进行绑定。详情请参考 [第 3.1.3.4 节“修改 ResourceBundle 属性的位置”](#)。

创建自定义模块

如果您想使自定义的资源对于运行在 JBoss EAP 服务器上的所有应用程序可用，您必须创建一个自定义的模块。关于这个途径的详情，请参考 [第 3.1.3.5 节“创建自定义模块”](#)。

[提交 bug 报告](#)

3.1.3.4. 修改 ResourceBundle 属性的位置

介绍

在以前的 JBoss EAP 版本里，**EAP_HOME/server/SERVER_NAME/conf/** 目录位于 classpath 里且可为应用程序所用。要使属性为 JBoss EAP 6 里的应用程序的 classpath 所用，您必须将其打包至您的应用程序里。

过程 3.2. 修改 ResourceBundle 属性的位置

1. 如果您在开发 WAR 归档，您必须将这些属性包裹至 WAR 的 **WEB-INF/classes/** 目录里。
2. 如果您想要 EAP 里所有组件都可以访问这些属性，那您必须将其打包到 JAR 的根目录并将 JAR 放入 EAR 的 **lib/** 目录下。

[提交 bug 报告](#)

3.1.3.5. 创建自定义模块

下面的步骤描述了如何创建自定义模块以使书性文件和其他资源对于运行在 JBoss EAP 服务器上的所有应用程序所用。

过程 3.3. 创建自定义模块

1. 创建和填充 **module/** 目录结构。

- a. 在 **EAP_HOME/module** 目录下创建一个目录结构来包含文件和 JAR。例如：

```
$ cd EAP_HOME/modules/
$ mkdir -p myorg-conf/main/properties
```

- b. 将属性文件移到您在前一步骤里创建的 **EAP_HOME/modules/myorg-conf/main/properties/** 目录里。

- c. 在 **EAP_HOME/modules/myorg-conf/main/** 目录里创建一个包含下列 XML 内容的 **module.xml** 文件：

```
<module xmlns="urn:jboss:module:1.1" name="myorg-conf">
  <resources>
    <resource-root path="properties"/>
  </resources>
</module>
```

2. 修改服务器配置文件里的 **ee** 子系统。您可以使用 JBoss CLI 或手动编辑这个文件。

- o 按照下列步骤使用 JBoss CLI 来修改服务器配置文件。

- a. 启动服务器并连接至管理 CLI。

- 对于 Linux，输入下列命令：

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

- 对于 Windows，输入下列命令：

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

您应该看到下面的结果：

```
Connected to standalone controller at localhost:9999
```

- b. 要在 **ee** 子系统里创建 **myorg-conf**<global-modules> 元素，输入下列命令：

```
/subsystem=ee:write-attribute(name=global-modules, value=
[{"name"=>"myorg-conf", "slot"=>"main"}])
```

您应该看到下面的结果：

■

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

- o 如果您向手动编辑服务器配置文件，请按照下列步骤进行。
 - a. 停止服务器并打开服务器配置文件。如果您使用的是独立服务器，这个文件是 **EAP_HOME/standalone/configuration/standalone.xml**，如果是受管域，这个文件是 **EAP_HOME/domain/configuration/domain.xml**。
 - b. 找到 **ee** 子系统并为 **myorg-conf** 添加全局模块。下面是 **ee** 子系统元素的例子，已经进行修改并包含了 **myorg-conf** 元素：

例 3.1. myorg-conf 元素

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <global-modules>
    <module name="myorg-conf" slot="main" />
  </global-modules>
</subsystem>
```

3. 假设您复制了一个名为 **my.properties** 的文件到正确的模块位置，您现在可以使用下面的代码来加载属性文件了：

例 3.2. 加载属性文件

```
Thread.currentThread().getContextClassLoader().getResource("my.pro
perties");
```

[提交 bug 报告](#)

3.1.4. 日志的修改

3.1.4.1. 修改日志依赖关系

介绍

JBoss LogManager 支持所有日志框架的前端，所以您可以保持现有的日志代码或移至新的 JBoss 日志框架。不管用哪种方法，因为模块化类加载有改变，您可能需要修改您的应用程序来添加所需的依赖关系。

过程 3.4. 更新程序日志代码

1. [第 3.1.4.2 节 “为第三方日志框架更新应用程序代码”](#)
2. [第 3.1.4.3 节 “修改代码以使用新的 JBoss Logging 框架”](#)

[提交 bug 报告](#)

3.1.4.2. 为第三方日志框架更新应用程序代码

概况

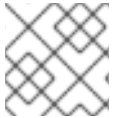
在 JBoss EAP 6 里，在默认情况下都已经添加了常见的第三方框架如 Apache Commons Logging、

Apache log4j、SLF4J 和 Java Logging 的日志依赖关系。在多数情况下，使用 JBoss EAP 容器提供的日志框架是更可取的。但是，如果您需要第三方框架提供的专有功能，您必须在部署里排除对应的 JBoss EAP 模块。请注意，虽然您的部署使用第三方的框架，服务器日志仍会继续使用 JBoss EAP 日志子系统配置。

下面的过程演示了如何从部署里排除 JBoss EAP 6 `org.apache.log4j` 模块。第一个过程适用于所有的 JBoss EAP 6 版本，而第二个过程只适用于 JBoss EAP 6.3 及以后的版本。

过程 3.5. 配置 JBoss EAP 6 以使用 log4j.properties 或 log4j.xml 文件

这个过程适用于所有的 JBoss EAP 6 版本。



注意

因为这个方法使用了 log4j 配置文件，您将无法在运行时再修改 log4j 日志配置。

1. 用下列内容创建一个 `jboss-deployment-structure.xml` 文件：

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

2. 如果您在部署 WAR，请将 `jboss-deployment-structure.xml` 文件放入 **META-INF/** 或 **WEB-INF/** 目录；如果是部署 EAR，则请放入 **META-INF/** 目录。如果您的部署包含了依赖的子部署，您必须在每个子部署里排除这些模块。
3. 在 EAR 的 **lib/** 或 WAR 部署的 **WEB-INF/classes/** 里包含 `log4j.properties` 或 `log4j.xml` 文件。如果您想将文件放到 **lib/** 目录里，您必须在 `jboss-deployment-structure.xml` 文件里指定 `<resource-root>` 路径。

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
    <resources>
      <resource-root path="lib" />
    </resources>
  </deployment>
</jboss-deployment-structure>
```

4. 用下列 runtime 参数启动 JBoss EAP 6 服务器以防止在部署应用程序时控制台出现 **ClassCastException**。

-

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. 部署您的应用程序

过程 3.6. 配置 JBoss EAP 6.3 或更新版本的日志依赖关系

在 JBoss EAP 6.3 和以后的版本里，您可以使用新的 **add-logging-api-dependencies** logging 系统属性来排除第三方的日志框架依赖关系。下面的步骤演示了在 JBoss EAP 独立服务器上如何修改这个 logging 属性。

1. 用下列 runtime 参数启动 JBoss EAP 6 服务器以防止在部署应用程序时控制台出现 **ClassCastException**。

```
-Dorg.jboss.as.logging.per-deployment=false
```

2. 打开终端窗口并连接至管理 CLI。

- o 对于 Linux，输入下列命令：

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

- o 对于 Windows，输入下列命令：

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

3. 修改日志子系统里的 **add-logging-api-dependencies** 属性。

这个属性控制容器是否添加隐性的 logging API 依赖关系到部署里。

- o 如果为 **true**（默认值），将添加所有隐性的 logging API 依赖关系。
- o 如果设置为 **false**，依赖关系不会添加到部署里。

要排除第三方的日志框架依赖关系，您必须用下列命令设置这个属性为 **false**。

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

这个命令添加了 **<add-logging-api-dependencies>** 元素到 **standalone.xml** 配置文件的 **logging** 子系统里。

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <add-logging-api-dependencies value="false"/>
  ....
</subsystem>
```

4. 部署您的应用程序

[提交 bug 报告](#)

3.1.4.3. 修改代码以使用新的 JBoss Logging 框架

介绍

要使用新的框架，请修改您的导入和代码：

过程 3.7. 修改代码和依赖关系以使用新的 JBoss Logging 框架

1. 修改您的导入和日志代码

下面是一个使用新的 JBoss Logging 框架的例子：

```
import org.jboss.logging.Level;
import org.jboss.logging.Logger;

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

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. 添加日志依赖关系

包含 JBoss Logging 类的 JAR 位于名为 **org.jboss.logging** 的模块。您的 **MANIFEST-MF** 文件应该类似要：

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

关于如何查找模块依赖关系的更多信息，请参考 [第 3.1.2.3 节“根据类加载的修改更新应用程序的依赖关系”](#) and [第 4.2.1 节“调试和解决移植问题”](#)。

[提交 bug 报告](#)

3.1.5. 应用程序包的修改

3.1.5.1. 修改 EAR 和 WAR 的打包

介绍

当您移植应用程序时，由于模块化类加载的修改，您可能需要修改 EAR 或 WAR 的打包结构。模块依赖关系是以特定的顺序加载的：

1. 系统依赖关系
2. 用户依赖关系
3. 本地资源
4. 部署间的依赖关系

过程 3.8. 修改归档的打包

1. 打包 WAR

WAR 是一个模块，它里面的所有类都用相同的类加载器加载。这意味着打包在 **WEB-INF/lib/** 目录里的类将和 **WEB-INF/classes** 里的一样被对待。

2. 打包 EAR

EAR 由多个模块组成。**EAR/lib/** 目录是一个单个的模块且 EAR 里每个 WAR 或 EJB JAR 都是一个独立的模块。除非定义了显性的依赖关系，类不能访问 EAR 里其他模块里的类。子部署总是对父部署有着自动的依赖关系，这可以让它们访问 **EAR/lib/** 目录里的类。然而，子部署并不总是由自动的依赖关系来允许它们访问彼此。这个行为是通过设置 **ee** 子系统配置里的 **<ear-subdeployments-isolated>** 元素来控制的：

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
</subsystem>
```

在默认情况下，它被设置为 **false**，这允许子部署查看属于 EAR 里其他子部署的类。

关于类加载的更多信息，请参考

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss 开发指南》里的『类加载和模块』章节。

[提交 bug 报告](#)

3.1.6. 数据源和资源适配器配置的修改

3.1.6.1. 根据配置的变化更新应用程序

在 JBoss EAP 5 里，服务和子系统是在多个文件里配置的。而在 JBoss EAP 6 里，配置主要是在一个文件里完成的。如果您的应用程序使用了任何下面的资源或服务，您就可能需要修改配置。

1. 如果您的应用程序使用了数据源，请参考：[第 3.1.6.2 节“更新数据源配置”](#)。
2. 如果您的应用程序使用了 JPA 且捆绑了 Hibernate JAR，请参考下列移植选项：[第 3.1.6.4 节“为 Hibernate 或 JPA 配置数据源”](#)。
3. 如果您的应用程序使用了资源适配器，请参考：[第 3.1.6.5 节“更新资源适配器配置”](#)。
4. 关于如何配置基本安全性，请参考：[第 3.1.7.1 节“应用程序安全性的修改”](#)。

[提交 bug 报告](#)

3.1.6.2. 更新数据源配置

概况

在以前的 JBoss EAP 版本里，JCA 数据源配置在一个后缀为 ***-ds.xml** 的文件里定义。这个文件然后部署在服务器的 **deploy/** 下或和应用程序一起打包。JDBC 驱动会被复制到 **server/lib/** 目录或打包在应用程序的 **WEB-INF/lib/** 目录里。虽然这种配置数据源的方法仍被支持，我们不推荐在产品环境里使用它，因为 JBoss 的管理根据已不支持。

在 JBoss EAP 6 里，数据源是在服务器配置文件里进行配置的。如果 JBoss EAP 实例运行在受管域里，数据源是在 **domain/configuration/domain.xml** 文件里配置。如果 JBoss EAP 实例以独立服务器运行，那么数据源是在 **standalone/configuration/standalone.xml file** 里配置。以这种方式配置的数据源可以用 JBoss 管理接口（包括 Web 管理控制台和命令行接口）进行管理和控制。这些工具使得在受管域上管理部署和配置多个服务器更为容易。

下面的内容描述了如何修改你的数据源配置，从而可以通过不同的管理工具进行管理和支持。

移植到 JBoss EAP 6 的可管理数据源配置

JDBC 4.0 兼容的驱动可以作为部署或核心模块来安装。JDBC 4.0 兼容的驱动包含一个 **META-INF/services/java.sql.Driver** 文件，它制定驱动类名。非 JDBC 4.0 兼容的驱动需要其他的步骤。关于如何兼容 JDBC 4.0 驱动和更新您的数据源配置为 Web 管理控制台和 CLI 管理的配置，请参考第 3.1.6.3 节“安装和配置 JDBC 驱动”。

如果您的应用程序使用了 Hibernate 或 JPA，它可能要求进行额外的修改。详情请参考第 3.1.6.4 节“为 Hibernate 或 JPA 配置数据源”。

使用 IronJacamar 移植工具来转换配置数据

您可以使用 IronJacamar 工具来移植数据源和资源适配器配置。这个工具将 ***-ds.xml** 风格的配置文件转换为 JBoss EAP 6 所期望的格式。更多信息请参考第 4.1.6 节“使用 IronJacamar 工具来移植数据源和资源适配器配置”。

移植执行远程数据源查找的代码

在以前的 JBoss EAP 版本里，我们可以执行数据源对象的 JNDI 远程查找。然而，基于下列原因，这从来就不是我们推荐的做法。

- 客户对服务器资源的控制不稳定，如果客户崩溃或丢失到服务器的连接还可能导致连接泄露。
- 因为所有数据源操作都通过 **MBean** 代理，性能非常差。
- 不支持事务传播。

这个功能从 JBoss EAP 6 开始就已被删除，当您移植应用程序时可能会看到 **NotSerializableException**。我们推荐的办法是创建一个 EJB 来访问数据源并远程调用 EJB。关于更多的信息，请参考本指南的『远程调用的修改』章节。其他信息请访问 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》。

[提交 bug 报告](#)

3.1.6.3. 安装和配置 JDBC 驱动

概况

JDBC 驱动可以下列两种方式之一安装到容器里。

- 作为部署
- 作为核心模块

每种方式各自的优缺点如下。

在 JBoss EAP 6 里，数据源是在服务器配置文件里进行配置的。如果 JBoss EAP 实例运行在受管域里，数据源是在 **domain/configuration/domain.xml** 文件里配置。如果 JBoss EAP 实例以独立服务器运行，那么数据源是在 **standalone/configuration/standalone.xml file** 里配置。Schema 引用信息两种模式都是一样的，您可以在 JBoss EAP 6 的 **docs/schema/** 里找到。我们在这里假设服务器是作为独立服务器运行的，而数据源是在 **standalone.xml** 文件里进行配置的。

过程 3.9. 安装和配置 JDBC 驱动

1. 安装 JDBC 驱动。

a. 将 JDBC 驱动作为部署安装。

这是我们推荐的安装驱动的方法。当 JDBC 驱动作为部署安装时，它会被部署为常规的

JAR。如果 JBoss EAP 实例作为独立服务器运行，请将兼容 JDBC 4.0 的 JAR 复制到 **EAP_HOME/standalone/deployments/** 目录。对于受管域，您必须使用管理控制台或管理 CLI 将 JAR 部署到服务器组里。

下面是一个安装为独立服务器的部署的 MySQL JDBC 驱动示例：

```
$scp mysql-connector-java-5.1.15.jar
EAP_HOME/standalone/deployments/
```

任何兼容 JDBC 4.0 的驱动都自动会被承认且根据名称和版本安装至系统里。兼容 JDBC 4.0 的 JAR 都包含一个名为 **META-INF/services/java.sql.Driver** 的文本文件，它指定驱动类的名称。如果这个驱动不兼容 JDBC 4.0，它可以用下列方法来部署：

- 创建并添加 **java.sql.Driver** 文件到 **META-INF/services/** 路径下的 JAR。这个文件应该包含驱动类的名称，例如：

```
com.mysql.jdbc.Driver
```

- 在 deployment 目录里创建一个 **java.sql.Driver** 文件。对于作为独立服务器运行的 JBoss EAP 6 实例，这个文件应该位于：**EAP_HOME/standalone/deployments/META-INF/services/java.sql.Driver**。如果服务器位于受管服务器，您必须使用管理控制台或管理 CLI 来部署这个文件。

这种方法的优点是：

- 因为不需要定义模块，所以这是最简单的方法。
- 当服务器运行在受管域里时，使用这个方法的部署将自动传播到域里的所有服务器。这意味着管理员不需要手动分发这个驱动 JAR。

这种方法的缺点是：

- 如果 JDBC 驱动由多个 JAR 组成，如驱动 JAR 以及依赖的许可证 JAR 或本地化 JAR，您不能将驱动安装为部署。您必须将其安装为核心模块。
- 如果这个驱动不兼容 JDBC 4.0，您必须创建一个包含驱动类名的文件并导入至 JAR 或覆盖到 **deployments/** 目录。

b. 将 JDBC 驱动安装为核心模块。

要将 JDBC 驱动安装为核心模块，您必须在 **EAP_HOME/modules/** 下创建一个文件路径结构。这个结构包含 JDBC 驱动 JAR，任何其他供应商许可证或本地化 JAR，以及一个定义模块的 **module.xml** 文件。

- 将 MySQL JDBC 驱动安装为核心模块

- i. 创建一个目录结构 **EAP_HOME/modules/com/mysql/main/**

- ii. 在 **main/** 子目录里，创建一个包含下列为 MySQL JDBC 驱动定义的 **module.xml** 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
```

```
<dependencies>
  <module name="javax.api"/>
</dependencies>
</module>
```

模块名 “com.mysql”，映射了这个模块的目录结构。**<dependencies>** 元素用来指定这个模块对其他模块的依赖关系。在这种情况下，所有的 JDBC 数据源都依赖于在另一个名为 **javax.api** 的模块里定义的 Java JDBC API。这个模块位于 **modules/system/layers/base/javax/api/main/** 目录。



注意

确保您在 **module.xml** 文件的开头没有留下一个空格，否则您会遇到 "New missing/unsatisfied dependencies" 错误。

- iii. 复制 MySQL JDBC 驱动 JAR 到 **EAP_HOME/modules/com/mysql/main/** 目录：

```
$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/
```

■ 将 IBM DB2 JDBC 驱动和 license JAR 安装为核心模块。

这个例子只是用来演示如何部署要求 JDBC Driver JAR 之外的 JAR 的驱动。

- i. 创建目录结构 **EAP_HOME/modules/com/ibm/db2/main/**。
- ii. 在 **main/** 子目录里，创建一个包含下列用于 IBM DB2 JDBC 驱动和许可证的模块定义的 **module.xml** 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```



注意

确保您在 **module.xml** 文件的开头没有留下一个空格，否则您会遇到 "New missing/unsatisfied dependencies" 错误。

- iii. 复制 JDBC 驱动和 license JAR 到 **EAP_HOME/modules/com/ibm/db2/main/** 目录里。

```
$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
EAP_HOME/modules/com/ibm/db2/main/
```

这种方法的优点是：

- 当 JDBC 驱动由多个 JAR 组成时，这是唯一的途径。
- 用这个方法，不兼容 JDBC 4.0 的驱动可以无需修改驱动 JAR 或创建覆盖文件就进行安装。

这种方法的缺点是：

- 通过设置模块要更为困难。
- 您必须手动将模块复制到运行在受管域里的每个服务器里。

2. 配置数据源。

a. 添加数据库驱动。

在相同的文件的 **<drivers>** 元素了添加 **<driver>** 元素。它也包含了之前的 ***-ds.xml** 文件里定义的一些数据源信息。

首先确定驱动 JAR 是否兼容 JDBC 4.0。兼容 JDBC 4.0 的 JAR 包含一个指定驱动类名的 **META-INF/services/java.sql.Driver**。这个服务器使用这个文件来在 JAR 里查找驱动类的名称。兼容 JDBC 4.0 的 JAR 不要求 **<driver-class>** 元素，因为在 JAR 里它已被指定。下面是一个 JDBC 4.0 兼容的 MySQL 驱动的 driver 元素示例：

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql"/>
```

不兼容 JDBC 4.0 的驱动要求一个 **<driver-class>** 属性来确定驱动类，这是因为没有 **META-INF/services/java.sql.Driver** 文件来指定驱动类名。下面是一个不兼容 JDBC 4.0 的 MySQL 驱动的 driver 元素示例：

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql">
<driver-class>com.mysql.jdbc.Driver</driver-class></driver>
```

b. 创建数据源。

在 **standalone.xml** 文件的 **<databases>** 部分创建一个 **<datasource>** 元素。这个文件包含之前的 ***-ds.xml** 文件里定义的大部分数据源信息。



重要

要使修改在服务器重启后仍然生效，您必须在编辑服务器配置文件前停止服务器。

下面是 **standalone.xml** 文件里的一个 MySQL 数据源元素示例：

```
<datasource jndi-name="java:/YourDatasourceName" pool-
name="YourDatasourceName">
<connection-
url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
```

```

url>
  <driver>mysql-connector-java-5.1.15.jar</driver>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
  <pool>
    <min-pool-size>100</min-pool-size>
    <max-pool-size>200</max-pool-size>
  </pool>
  <security>
    <user-name>USERID</user-name>
    <password>PASSWORD</password>
  </security>
  <statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>

```

3. 更新程序代码里的 JNDI 引用。

您必须在程序源代码里替换过时的 JNDI 查找名称以使用新的 JNDI 标准化数据源名称。详情请参考：[第 3.1.8.4 节 “修改应用程序以遵循新的 JNDI 命名规则”](#)。

您也必须让现有的访问数据源的 `@Resource` 注解使用新的 JNDI 名称。例如：

```
@Resource(name = "java:/YourDatasourceName").
```

[提交 bug 报告](#)

3.1.6.4. 为 Hibernate 或 JPA 配置数据源

如果您的应用程序时用了 JPA 且捆绑了 Hibernate JAR，您可能会想要使用 JBoss EAP 6 附带的 Hibernate。要使用这个版本的 Hibernate，您必须从应用程序里删除旧的 Hibernate 捆绑。

过程 3.10. 删除 Hibernate 捆绑

1. 从您的应用程序的库目录里将 Hibernate JAR 删除。
2. 当不再需要 `<hibernate.transaction.manager_lookup_class>` 元素时，在 `persistence.xml` 里将其删除或注释。

[提交 bug 报告](#)

3.1.6.5. 更新资源适配器配置

介绍

在以前的应用服务器版本里，资源适配器配置是在带有后缀 `*-ds.xml` 的文件里定义的。在 JBoss EAP 6 里，资源适配器是在服务器配置文件里配置的。如果您是在受管域里运行的，配置文件是 `EAP_HOME/domain/configuration/domain.xml`。如果您运行的是独立服务器，配置文件是 `EAP_HOME/standalone/configuration/standalone.xml`。Schema 引用信息两种模式都是一样的，您可以在 Iron Jacamar 网站 <http://www.ironjacamar.org/documentation.html> 的 *Schemas* 里找到。



重要

要使修改在服务器重启后仍然生效，您必须在编辑服务器配置文件前停止服务器。

定义资源适配器

资源适配器描述符信息是在服务器配置文件中的下列 subsystem 元素里定义的：

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

您将使用资源适配器 *-ds.xml 文件里定义的相同的一些信息。

下面是服务器配置文件里的资源适配器元素示例：

```
<resource-adapters>
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
        class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory1"
        enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
        pool-name="MultipleConnectionFactory1">
      <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
    </connection-definition>
    <connection-definition
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory2"
      enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
      pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
    </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
      jndi-name="java:/eis/MultipleAdminObject1">
    <config-property name="Name">MultipleAdminObject1Value</config-
property>
    </admin-object>
    <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
      jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-
property>
    </admin-object>
```

```

    </admin-objects>
  </resource-adapter>
</resource-adapters>

```

[提交 bug 报告](#)

3.1.7. 安全性的修改

3.1.7.1. 应用程序安全性的修改

配置基本验证的安全性

在以前的 JBoss EAP 版本里，放在 **EAP_HOME/server/SERVER_NAME/conf/** 下的属性文件位于 classpath 上且可以轻易地被 **UsersRolesLoginModule** 找到。在 JBoss EAP 6 里，目录结构已经发生了变化。属性文件必须打包在应用程序里，使其在 classpath 上可用。



重要

要使修改在服务器重启后仍然生效，您必须在编辑服务器配置文件前停止服务器。

要配置基本验证的安全性，请在 **standalone/configuration/standalone.xml** 或 **domain/configuration/domain.xml** 服务器配置文件里的 **security-domains** 下添加一个新的安全域：

```

<security-domain name="example">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties"
        value="{jboss.server.config.dir}/example-
users.properties"/>
      <module-option name="rolesProperties"
        value="{jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
  </authentication>
</security-domain>

```

如果 JBoss EAP 6 实例作为独立服务器运行，**{jboss.server.config.dir}** 指向 **EAP_HOME/standalone/configuration/** 目录。如果它运行在受管域里，**{jboss.server.config.dir}** 指向 **EAP_HOME/domain/configuration/** 目录。

修改安全域的名称

在 JBoss EAP 6 里，安全域名称里不再使用前缀 **java:/jaas/**。

- 对于 Web 应用程序，您必须从 **jboss-web.xml** 里的安全域配置里删除这个前缀。
- 对于企业级应用程序，您必须从 **jboss-ejb3.xml** 里的安全域配置里删除这个前缀。这个文件已经替换了 JBoss EAP 6 里的 **jboss.xml**。

[提交 bug 报告](#)

3.1.7.2. 更新使用 PicketLink STS 和 Web Services 的应用程序

介绍

如果您的 JBoss EAP 6.1 应用程序使用了 PicketLink STS 和 Web service，在移植到 JBoss EAP 6.2 或更新版本时您需要进行一些修改。我们修复了 JBoss EAP 以解决 [CVE-2013-2133](#)，在运行任何附加到基于 EJB3 的 WS 端点的 JAXWS 处理程序前它强制执行容器的授权检查。因此，一些 PicketLink STS 功能会受到影响，因为 PicketLink **SAML2Handler** 会建立一个安全性主体在以后使用。您可能在服务器日志里看到 **NullPointerException**，因为当 **HandlerAuthInterceptor** 访问 **SAML2Handler** 时这个主体是 **NULL**。要修复这个问题，您必须要禁用安全检查。

过程 3.11. 禁用额外的授权检查

- 您可以禁用额外的授权检查并通过下列方法之一坚持使用现有的 PicketLink 部署。
 - **设置系统属性**
您可以通过设置 **org.jboss.ws.cxf.disableHandlerAuthChecks** 属性为 **true** 来禁用服务器级别的额外授权检查。这个方法会影响应用服务器上的所有部署。

关于设置系统属性的信息，请参考《管理和配置指南》里的『用管理 CLI 配置系统属性』。
 - **在部署的 Web 服务描述文件里创建一个属性**
您可以通过设置 **jboss-webservices.xml** 文件里的 **org.jboss.ws.cxf.disableHandlerAuthChecks** 属性为 **true** 来禁用部署级别的额外授权检查。这个方法只会影响专有的部署。
 - a. 在您要禁用额外授权检查的部署的 **META-INF/** 目录里创建一个 **jboss-webservices.xml** 文件。
 - b. 添加下列内容：

```
<?xml version="1.1" encoding="UTF-8"?>
<webservices xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.2"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee">
  <property>
    <name>org.jboss.ws.cxf.disableHandlerAuthChecks</name>
    <value>true</value>
  </property>
</webservices>
```



注意

启用 **org.jboss.ws.cxf.disableHandlerAuthChecks** 属性会让系统易受 [CVE-2013-2133](#) 攻击。如果应用程序期望应用 EJB 方法上声明的安全限制且不独立于 JAX-WS 处理程序进行应用，那这个属性不应该启用。这个属性只能用于向后兼容的目的来避免破坏应用程序。

[提交 bug 报告](#)

3.1.8. JNDI 的修改

3.1.8.1. 更新应用程序 JNDI 命名空间的名称

介绍

EJB 3.1 引入了一个标准化的全局 JNDI 命名空间和一系列相关的映射到不同 Java EE 应用程序作用域的命名空间。可移植 EJB 名称只能绑定其中三种：**java:global**、**java:module** 和 **java:app**。使用 JNDI 查找的应用程序必须进行修改以遵循新的标准化 JNDI 命名空间规则。

过程 3.12. 修改 JNDI 查找

1. 了解更多的 [第 3.1.8.2 节 “可移植的 EJB JNDI 名称”](#)
2. [第 3.1.8.3 节 “复核 JNDI 命名空间规则”](#)
3. [第 3.1.8.4 节 “修改应用程序以遵循新的 JNDI 命名规则”](#)

JNDI 映射示例

前一版本里的 JNDI 命名空间示例以及如何在 JBoss EAP 6 里指定可以在这里找到：[第 3.1.8.5 节 “以前版本的 JNDI 命名空间示例和它们在 JBoss EAP 6 里是如何指定的”](#)

[提交 bug 报告](#)

3.1.8.2. 可移植的 EJB JNDI 名称

介绍

Java EE 6 规格定义了 4 种逻辑命名空间，每种都有自己的作用域，但可移植的 EJB 名称只能绑定到其中三种。下表详述了何时和如何使用每种命名空间。

表 3.1. 可移植的 JNDI 命名空间

JNDI 命名空间	描述
java:global	<p>这个命名空间里的名称是应用程序服务器实例里部署的所有应用程序共享的。您可以使用这个命名空间里的名称来查找部署到相同服务器里的 EJB 外部归档。</p> <p>下面是一个 java:global 命名空间的示例：java:global/jboss-seam-bookings/jboss-seam-bookings-jar/HotelBookingAction</p>
java:module	<p>这个命名空间里的名称是由部署在模块里的所有组件共享的，例如单一 EJB 模块或 web 模块中所有组件里的所有 EJB。</p> <p>下面是一个 java:module 命名空间的示例：java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking</p>
java:app	<p>这个命名空间里的名称是由单个应用程序里的所有模块里的所有组件共享的，例如相同 EJB 文件里的 WAR 或 EJB JAR 文件将可以访问 java:app 命名空间里的资源。</p> <p>下面是一个 java:app 命名空间的示例：java:app/jboss-seam-bookings-jar/HotelBookingAction</p>

您在 EE.5.2.2 章节里可以找到关于 JNDI 命名上下文的更多信息，如 "JSR 316: Java™ Platform, Enterprise Edition (Java EE) Specification, v6" 里的 "Application Component Environment Namespaces"。您也可以在这里下载相关规格：<http://jcp.org/en/jsr/detail?id=316>。

[提交 bug 报告](#)

3.1.8.3. 复核 JNDI 命名空间规则

介绍

JBoss EAP 6 已经改进了 JNDI 命名空间，不只是为应用服务器里绑定的每个名称提供可预测和一致的规则，也防止了将来的兼容性问题。这意味着如果名称不遵循新的规则，应用程序里的当前命名空间也会出现问题。

命名空间应该遵循下列规则：

1. 未限定的相对名称如 **DefaultDS** 或 **jdbc/DefaultDS** 应该根据上下文限定于 **java:comp/env**、**java:module/env** 或 **java:jboss/env**。
2. 未限定的绝对名称如 **/jdbc/DefaultDS** 应该限定于 **java:jboss/root**。
3. 限定的绝对名称如 **java:/jdbc/DefaultDS** 应该和上面未限定的绝对名称采用相同的方式进行限定。
4. **java:jboss** 命名空间在整个 AS 服务器实例间进行共享。
5. 带有 **java:** 前缀的相对名称必须位于下列 5 个命名空间中的一个：**comp**、**module**、**app**、**global** 或私有的 **jboss**。任何以 **java:xxx** 开始的名称里的 **xxx** 不匹配上面的 5 个命名空间都将导致无效名称错误。

[提交 bug 报告](#)

3.1.8.4. 修改应用程序以遵循新的 JNDI 命名规则

- 下面是一个 JBoss EAP 5.1 里的 JNDI 查找示例。这些代码是在一个初始方法里找到的。

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch (Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

请注意查找名称是 **OrderManagerApp/ProductManagerBean/local**。

- 下面是在 JBoss EAP 6 里如何用依赖关系注入编写相同查找的示例：

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

查找值现在被定义为成员变量并使用新的可移植的 **java:app** JNDI 命名空间名 **java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager**。

- 如果您不想使用依赖关系注入，您仍可以像上面那样创建新的 **InitialContext** 并修改查找以使用新的 JNDI 命名空间名称。

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

[提交 bug 报告](#)

3.1.8.5. 以前版本的 JNDI 命名空间示例和它们在 JBoss EAP 6 里是如何指定的

表 3.2. JNDI 命名空间映射表

JBoss EAP 5.x 里的命名空间	JBoss EAP 6 里的命名空间	其他注释
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Java EE6 标准绑定。作用域为当前模块，只可以在相同模块里访问。
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE6 标准绑定。作用域为当前应用程序，只可以在相同应用程序里访问。
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE6 标准绑定。作用域为应用服务器，可全局访问。
java:comp/UserTransaction	java:comp/UserTransaction	命名空间作用域为当前组件。非 Java EE 6 的线程不可访问，例如您的应用程序直接创建的线程。
java:comp/UserTransaction	java:jboss/UserTransaction	可全局访问，如果 java:comp/UserTransaction 不可用则使用它。
java:/TransactionManager	java:jboss/TransactionManager	
java:/TransactionSynchronizationRegistry	java:jboss/TransactionSynchronizationRegistry	

[提交 bug 报告](#)

3.1.9. HTTP/HTTPS/AJP 连接器属性映射

3.1.9.1. HTTP/HTTPS/AJP 连接器属性映射

下表展示了如何将 HTTP、HTTPS 和 AJP 连接器属性从之前的版本映射到 Red Hat JBoss EAP 6。

表 3.3. 连接器属性映射

以前版本里的属性名	JBoss EAP 6 里的对等属性	详情
maxThreads	max-connections	<p>这个属性设置 JBossWeb 级别的线程/连接的大小。它是通过 Web 子系统的 connector 来设置的。默认值是每个 CPU 核心对应 512。</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" max-connections="200" /></pre>
minSpareThreads maxSpareThreads	不适用	<p>既然 JBoss EAP 6 不鼓励回收线程，minSpareThreads 或 maxSpareThreads 没有对等的属性。如果您使用执行器而不是默认的工作节点线程池，最相近的属性是 core-threads 和 keepalive-time。</p>
proxyName proxyPort	proxy-name proxy-port	<p>这个属性通过 web 子系统的 connector 来设置。</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" proxy-name="proxy.com" proxy- port="80"/></pre>
redirectPort	redirect-port	<p>这个属性通过 web 子系统的 connector 来设置。</p> <pre>connector name="http" protocol="HTTP/1.1" scheme="https" secure="true" socket- binding="http" redirect-port="8443" proxy- name="loadbalancer.hostname.com" proxy- port="443"</pre>
enableLookups	enable-lookups	<p>这个属性通过 web 子系统的 connector 来设置。</p> <pre><connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enable- lookups="true" /></pre>
MaxHttpHeaderSize	System Property	<p>这个属性是用系统属性设置的。默认值是 8KB。</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol. MAX_HEADER_SIZE" value="8192"/> </system-properties></pre>

以前版本里的属性名	JBoss EAP 6 里的对等属性	详情
maxKeepAliveRequests	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol. MAX_KEEP_ALIVE_REQUESTS" value="1"/> </system-properties></pre>
connectionTimeout	System Property	<p>这个属性是用系统属性设置的。下面的配置设置 AJP connectionTimeout 为 600000 毫秒（10 分钟）及 HTTP connectionTimeout 为 120000 毫秒（2 分钟）。</p> <pre><system-properties> <!-- connectionTimeout for AJP connector. Default value is "-1" (no timeout). --> <property name="org.apache.coyote.ajp.DEFAULT_CONNECTION _TIMEOUT" value="600000"/> <!-- connectionTimeout for HTTP connector. Default value is "60000". --> <property name="org.apache.coyote.http11.DEFAULT_CONNECT ION_TIMEOUT" value="120000"/> </system-properties></pre>
compression	System Property	<p>这个属性启用压缩。您可以指定内容类型，其默认值为 text/html, text/xml, text/plain。您也可以指定将进行压缩的内容的最小尺寸，其默认值为 2048 字节。压缩将通过 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol. COMPRESSION" value="on"/> <property name="org.apache.coyote.http11.Http11Protocol. COMPRESSION_MIN_SIZE" value="4096"/> <property name="org.apache.coyote.http11.Http11Protocol. COMPRESSION_MIME_TYPES" value="text/javascript, text/css, text/html"/> </system-properties></pre>

以前版本里的属性名	JBoss EAP 6 里的对等属性	详情
URIEncoding	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.catalina.connector.URI_ENCODING" value="UTF-8"/> </system-properties></pre>
useBodyEncodingForURI	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.catalina.connector.USE_BODY_ENCODING_FOR_QUERY_STRING" value="true"/> </system-properties></pre>
server	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.coyote.http11.Http11Protocol.SERVER" value="NewServerHeader"/> </system-properties></pre>
allowTrace	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.catalina.connector.ALLOW_TRACE " value="true"/> </system-properties></pre>
xpoweredby	System Property	<p>这个属性将用 System Property 进行设置。</p> <pre><system-properties> <property name="org.apache.catalina.connector.X_POWERED_BY " value="true"/> </system-properties></pre>

以前版本里的属性名	JBoss EAP 6 里的对等属性	详情
keepAliveTimeout	不适用	<p>在 JBoss EAP 6.4 之前没有对等的参数，它在内部被默认为 connectionTimeout 的值。</p> <p>In JBoss EAP 6.4, a new system property, org.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT, was introduced to control keepAliveTimeout. The -Dorg.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT argument takes affect when used with the -Dorg.apache.coyote.http11.DEFAULT_DISABLE_UPLOAD_TIMEOUT=true argument.</p>
disableUploadTimeout connectionUploadTimeout	不适用	JBoss EAP 6 里目前没有对等的参数。 disableUploadTimeout 默认为 true 且 connectionUploadTimeout 在内部使用 connectionTimeout 的值。
packetSize	System Property	<p>这个属性是用系统属性设置的。下面的配置设置 AJP packetSize 为 20000。</p> <pre><system-properties> <property name="org.apache.coyote.ajp.MAX_PACKET_SIZE" value="20000"/> </system-properties></pre>
maxPostSize maxSavePostSize	max-post-size max-save-post-size	<p>0 表示无限制。请注意，只有当 Content-Type 为 application/x-www-form-urlencoded 时，这个参数才会限制数据大小。关于更多的信息，请参考客户门户里的解决方案：How to limit data size of HTTP POST method from a client to JBoss。</p>

以前版本里的属性名	JBoss EAP 6 里的对等属性	详情
tomcatAuthentication	System Property	<p>根据 JBoss EAP 6 的版本，这个属性将用系统属性 <code>org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION</code> 或 <code>org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION</code> 设置。服务器的所有版本都需要安装补丁或升级。</p> <p>在 JBoss EAP 6.0.1 里，tomcatAuthentication 使用下列属性配置的。</p> <pre><system-properties> <property name="org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION" value="false"/> </system-properties></pre> <p>在 JBoss EAP 6.1 及更高版本里，tomcatAuthentication 是这样配置的：</p> <pre><system-properties> <property name="org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION" value="false"/> </system-properties></pre> <p>详情请参考客户门户里的 How to configure tomcatAuthentication in JBoss EAP 6。</p>

关于连接器的更多信息，请参考客户门户里的 [Equivalent HTTP/HTTPS/AJP connector attributes mapping between JBoss EAP 5.x and JBoss EAP 6.x](#)。

[提交 bug 报告](#)

3.2. 依赖于应用程序架构和组件的修改

3.2.1. 复查依赖于应用程序架构和组件的修改

如果您的应用程序使用了下面的技术或组件，在移植到 JBoss EAP 6 时您可能需要修改应用程序。

Hibernate 和 JPA

如果您的应用程序使用了 Hibernate 或 JPA，您的应用程序可能需要进行修改。详情请参考：[第 3.2.2.1 节“更新使用 Hibernate 和/或 JPA 的应用程序”](#)。

REST

如果您的应用程序使用了 JAX-RS，您应该意识到 JBoss EAP 6 会自动设置 RESTEasy，所以您不需要再进行配置。详情请参考 [第 3.2.5.1 节“配置 JAX-RS 和 RESTEasy 的修改”](#)。

LDAP

在 JBoss EAP 6 里，LDAP 安全区的配置是不一样的。如果您的应用程序使用了 LDAP，请参考：[第 3.2.6.1 节“配置 LDAP Security Realm 的修改”](#)。

Messaging

JBoss EAP 6 里不再包含 JBoss Messaging。如果您的应用程序使用了 JBoss Messaging，您需要用 HornetQ 替换 JBoss Messaging 代码。下面是相关的信息：[第 3.2.7.4 节“移植您的应用程序以将 HornetQ 用作 JMS 提供者”](#)。

群集

在 JBoss EAP 6 里启用群集的方式已经修改了，详情请参考：[第 3.2.8.1 节“修改应用程序以用于群集环境”](#)。

服务风格的部署

虽然 JBoss EAP 6 不再使用服务风格的描述符，容器还是这种风格的部署而无需进行修改。关于部署的信息，请参考：[第 3.2.9.1 节“更新使用服务风格部署的应用程序”](#)。

远程调用

如果您的应用程序进行了远程调用，您仍可以使用 JNDI 为您的 Bean 查找代理并在返回的代理上进行调用。关于语法和命名空间的修改，请参考：[第 3.2.10.1 节“将进行远程调用的 JBoss EAP 5 应用程序移植到 JBoss EAP 6”](#)。

Seam 2.2

如果您的应用程序使用了 Seam 2.2，对于可能需要的修改，请参考：[第 3.2.13.1 节“移植 Seam 2.2 归档到 JBoss EAP 6”](#)。

Spring

如果您的应用程序使用了 Spring，请参考：[第 3.2.14.1 节“移植 Spring 应用程序”](#)。

其他可能影响应用程序移植的修改

对于 JBoss EAP 6 里可能影响您的应用程序的其他修改，请参考：[第 3.2.15.1 节“熟悉其他可能影响到移植的修改”](#)。

[提交 bug 报告](#)

3.2.2. Hibernate 和 JPA 的修改

3.2.2.1. 更新使用 Hibernate 和/或 JPA 的应用程序

介绍

如果您的应用程序使用了 Hibernate 或 JPA，请阅读下列章节并进行修改以移植到 JBoss EAP 6。

- [第 3.2.2.2 节“修改使用 Hibernate 和 JPA 的应用程序的配置”](#)
- [第 3.2.2.4 节“更新 Hibernate 3 应用程序以使用 Hibernate 4”](#)
- [第 3.2.2.9 节“更新您的应用程序以遵循 JPA 2.0 规格”](#)
- [第 3.2.2.10 节“用 Infinispan 替换 JPA/Hibernate 二级缓存”](#)
- [第 3.2.2.12 节“移植到 Hibernate Validator 4”](#)

[提交 bug 报告](#)

3.2.2.2. 修改使用 Hibernate 和 JPA 的应用程序的配置

介绍

如果您的应用程序包含一个 `persistence.xml` 文件或代码使用了注解 `@PersistenceContext` 或 `@PersistenceUnit`，JBoss EAP 6 会在部署时进行检测并假设应用程序使用了 JPA。这会隐性地添加 Hibernate 4 以及一些其他的依赖关系到应用程序的 classpath。

如果您的应用程序使用了 Hibernate 3，在大多数情况下，您将可以切换到 Hibernate 4 且成功运行。然而，如果您在部署应用程序时看到了 `ClassNotFoundExceptions`，您可以用下列方法来解决：



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 `org.hibernate` 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 3.13. 配置应用程序

1. 复制所需的 Hibernate 3 JAR 到您的应用程序库。

您可以复制包含缺失类的特定 Hibernate 4 JAR 到应用程序的 `lib/` 目录，或者使用其他方法添加它们到 classpath 来解决这个问题。在某些情况下，由于混合使用 Hibernate 版本，这可能会导致 `ClassCastException` 或其他类加载问题。如果发生了这种情况，您需要使用下一个方法。

2. 设置服务器只使用 Hibernate 3 库。

JBoss EAP 6 允许您将 Hibernate 3.5 (或更高版本) 持久化提供者 JAR 和应用程序打包。为了让服务器只使用 Hibernate 3 库而排除 Hibernate 4 库，您需要在 `persistence.xml` 里设置 `jboss.as.jpa.providerModule` 为 `hibernate3-bundled`：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="plannerdatasource_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
source>
    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
  </persistence-unit>
</persistence>
```

Java Persistence API (JPA) 开发人员将检测到应用程序里持久性提供者的存在并使用 Hibernate 3 库。关于 JPA 持久性属性的更多信息，请参考第 3.2.2.3 节“持久化单元属性”。

3. 禁用 Hibernate 的二级缓存

Hibernate 3 的二级缓存在 JBoss EAP 6 没有展现和以前版本里一样的行为。如果您在应用程序里是了 Hibernate 二级缓存，您必须禁用它，直至您升级到 Hibernate 4。要禁用二级缓存，您需要在 `persistence.xml` 文件里将 `<hibernate.cache.use_second_level_cache>` 设置为 `false`。

3.2.2.3. 持久化单元属性

Hibernate 4.x 配置属性

JBoss EAP 6 自动设置下列 Hibernate 4.x 配置属性：

表 3.4. Hibernate 持久化单元属性

属性名	默认值	目的
<code>hibernate.id.new_generator_mappings</code>	<code>true</code>	如果您使用 <code>@GeneratedValue(AUTO)</code> 为新条目生成唯一的索引值，这个设置就是相关的。新的应用程序应该保持默认的值 true 。现有的使用 Hibernate 3.3.x 的应用程序可能需要将其修改为 false 以继续使用序列对象或基于表的生成器并保持向后的兼容性。应用程序可以在 <code>persistence.xml</code> 文件里覆盖这个值。 下面提供了关于这种行为的更多信息。
<code>hibernate.transaction.jta.platform</code>	<code>org.hibernate.service.jta.platform.spi.JtaPlatform</code> 接口的实例	这个类将事务管理者、用户事务、事务同步注册表传入 Hibernate。
<code>hibernate.ejb.resource_scanner</code>	<code>org.hibernate.ejb.packaging.Scanner</code> 接口的实例	这个类知道如何使用 JBoss EAP 注解 indexer 来提供更快的部署。
<code>hibernate.transaction.manager_lookup_class</code>		如果在 <code>persistence.xml</code> 里找到这个属性，它将被删除，因为它和 <code>hibernate.transaction.jta.platform</code> 相冲突
<code>hibernate.session_factory_name</code>	<code>QUALIFIED_PERSISTENCE_UNIT_NAME</code>	这将设置为应用程序名称 + 持久化单元名称。应用程序可以指定不同的值但它必须在 JBoss EAP 实例上的所有应用程序部署里都是唯一的。
<code>hibernate.session_factory_name_is_jndi</code>	<code>false</code>	只有应用程序没有为 <code>hibernate.session_factory_name</code> 指定值时它才会被设置。
<code>hibernate.ejb.entitymanager_factory_name</code>	<code>QUALIFIED_PERSISTENCE_UNIT_NAME</code>	这将设置为应用程序名称 + 持久化单元名称。应用程序可以指定不同的值但它必须在 JBoss EAP 实例上的所有应用程序部署里都是唯一的。

在 Hibernate 4.x 里，如果 `new_generator_mappings` 被设置为 `true`：

- `@GeneratedValue(AUTO)` 映射 `org.hibernate.id.enhanced.SequenceStyleGenerator`。

- `@GeneratedValue(TABLE)` 映射 `org.hibernate.id.enhanced.TableGenerator`。
- `@GeneratedValue(SEQUENCE)` 映射 `org.hibernate.id.enhanced.SequenceStyleGenerator`。

在 Hibernate 4.x 里，如果 `new_generator_mappings` 被设置为 `false`：

- `@GeneratedValue(AUTO)` 映射 Hibernate "native"。
- `@GeneratedValue(TABLE)` 映射 `org.hibernate.id.MultipleHiLoPerTableGenerator`。
- `@GeneratedValue(SEQUENCE)` 映射 Hibernate "seqhilo"。

关于这些属性的更多信息，请访问 <http://www.hibernate.org/docs> 并查看 [Hibernate 4.1 Developer Guide](#)。

JPA 持久化属性

`persistence.xml` 文件里的持久化单元定义支持下面的 JPA 属性：

表 3.5. JPA 持久化单元属性

属性名	默认值	目的
<code>jboss.as.jpa.providerModule</code>	<code>org.hibernate</code>	持久化提供者模块的名称。 如果 Hibernate 3 JAR 位于应用程序归档里，这个值应该为 hibernate3-bundled 。 如果这个应用程序和持久化提供者一起打包，这个值应该为 application 。
<code>jboss.as.jpa.adapterModule</code>	<code>org.jboss.as.jpa.hibernate:4</code>	帮助 JBoss EAP 和持久化提供者一起使用的集成类的名称。 目前的有效值是： <ul style="list-style-type: none"> • <code>org.jboss.as.jpa.hibernate:4</code>：用于 Hibernate 4 集成类 • <code>org.jboss.as.jpa.hibernate:3</code>：用于 Hibernate 3 集成类

[提交 bug 报告](#)

3.2.2.4. 更新 Hibernate 3 应用程序以使用 Hibernate 4

介绍

当您更新应用程序以使用 Hibernate 4 时，某些更新是通用的，而不管您目前使用的是哪个版本的 Hibernate。对于其他的更新，您必须确定您当前使用的版本。

过程 3.14. 更新应用程序以使用 Hibernate 4

1. 自动增量序列生成器的默认行为在 JBoss EAP 6 已经进行了修改。详情请参考 [第 3.2.2.5 节“Hibernate 标识符自动生成值”](#)。

2. 确定应用程序当前使用的 Hibernate 版本并选择下面正确的更新过程。
 - [第 3.2.2.6 节 “移植您的 Hibernate 3.3.x 应用程序到 Hibernate 4.x”](#)
 - [第 3.2.2.7 节 “移植您的 Hibernate 3.5.x 应用程序到 Hibernate 4.x”](#)
3. See [第 3.2.2.8 节 “修改运行在群集环境里的移植的 Seam 和 Hibernate 应用程序的持久化属性”](#) 如果您准备在群集环境里运行您的应用程序。

[提交 bug 报告](#)

3.2.2.5. Hibernate 标识符自动生成值

Hibernate 3.5 引入了名为 `hibernate.id.new_generator_mappings` 的核心属性，它指定在使用 `@GeneratedValue` 时如何生成标识符或序列号。在 JBoss EAP 6 里，这个属性的默认值是：

- 当您部署原声 Hibernate 应用程序时，这个默认值是 **false**。
- 当您部署 JPA 应用程序时，这个默认值是 **true**。

新应用程序准则

使用 `@GeneratedValue` 的新应用程序应该设置 `hibernate.id.new_generator_mappings` 属性为 **true**。这是首选的设置，因为对于不同的数据库它更容易移植。在多数情况下，它的效率更高；在某些情况下，它解决了和 JPA 2 规格的兼容问题。

- 对于新的 JPA 应用程序，JBoss EAP 6 默认 `hibernate.id.new_generator_mappings` 属性为 **true** 且不应该被改变。
- 对于新的原生 Hibernate 应用程序，JBoss EAP 6 默认 `hibernate.id.new_generator_mappings` 属性为 **false**。您应该设置它为 **true**。

现有的 JBoss EAP 5 应用程序的准则

当移植到 JBoss EAP 6 时，现有的使用 `@GeneratedValue` 注解的应用程序应该确保使用相同的生成器来为新条目创建主键值。

- 对于现有的 JPA 应用程序，JBoss EAP 6 默认 `hibernate.id.new_generator_mappings` 属性为 **true**。您应该在 `persistence.xml` 里将这个属性设置为 **false**。
- 对于现有的原生 Hibernate 应用程序，JBoss EAP 6 默认 `hibernate.id.new_generator_mappings` 属性为 **false** 且不应该被改变。

关于这些属性设置的更多信息，请参考 [第 3.2.2.3 节 “持久化单元属性”](#)。

[提交 bug 报告](#)

3.2.2.6. 移植您的 Hibernate 3.3.x 应用程序到 Hibernate 4.x

1. 映射 Hibernate text 类型为 JDBC LONGVARCHAR

在 Hibernate 3.5 以前的版本里，`text` 类型被映射为 **JDBC CLOB**。Hibernate 4 里引入了一个新的 Hibernate 类型 `materialized_clob`，将 Java `String` 属性映射为 **JDBC CLOB**。如果您的应用程序里有属性需要将 `type="text"` 映射为 **JDBC CLOB**，您必须这样：

- a. 如果您的应用程序使用了 hbm 映射文件，请将这个属性修改为 `type="materialized_clob"`。

- b. 如果您的应用程序使用了注解，您应该将 `@Type(type = "text")` 替换为 `@Lob`。
- 2. 复查代码以找出返回值类型的修改
数字型组合标准的 `Projection` 现在返回的是 HQL 对应类型的值。因此，`org.hibernate.criterion` 里的下列 `projection` 的返回类型已经有了改动。
 - a. `CountProjection`、`Projections.rowCount()`、`Projections.count(propertyName)` 和 `Projections.countDistinct(propertyName)` 里的修改，`count` 和 `count distinct` `projection` 现在返回的是 `Long` 型值。
 - b. 由于 `Projections.sum(propertyName)` 里的修改，`sum` `projection` 现在返回依赖于属性类型的值。



注意
修改程序代码失败可能导致 `java.lang.ClassCastException`。

- i. 对映射为 `Long`、`Short`、`Integer` 或原始类型的属性，将返回 `Long` 值。
- ii. 对映射为 `Float`、`Double` 或原始浮点类型的属性，将返回 `Double` 值。

[提交 bug 报告](#)

3.2.2.7. 移植您的 `Hibernate 3.5.x` 应用程序到 `Hibernate 4.x`

- 1. 合并 `AnnotationConfiguration` 到配置里。

`AnnotationConfiguration` 已废弃，它不应该影响应用程序的移植。

如果您还在使用 `hbm.xml` 文件，您应该意识到 JBoss EAP 6 现在在 `AnnotationConfiguration` 里使用 `org.hibernate.cfg.EJB3NamingStrategy`，而不是以前版本里的 `org.hibernate.cfg.DefaultNamingStrategy`。这可能导致命名不匹配。如果您依赖于这个命名策略来设置关系（多对多和元素集合）表的默认名称，您可能会遇到这个现象。要解决这个问题，您可以调用 `Configuration#setNamingStrategy` 传入 `org.hibernate.cfg.DefaultNamingStrategy#INSTANCE` 让 `Hibernate` 使用旧的 `org.hibernate.cfg.DefaultNamingStrategy`。

- 2. 修改这个命名空间以遵循下表里注明的新的 `Hibernate DTD` 文件名称规则。

表 3.6. DTS 命名空间映射表

以前的 DTD 命名空间	新的 DTD 命名空间
<code>http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd</code>
<code>http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd</code>

- 3. 修改环境变量。
 - a. 如果您在使用 `Oracle` 以及 `materialized_clob` 或 `materialized_blob` 属性，全局环境变量 `hibernate.jdbc.use_streams_for_binary` 必须设置为 `true`。

- b. 如果您在使用 PostgreSQL 以及 **CLOB** 或 **BLOB** 属性，全局环境变量 **hibernate.jdbc.use_streams_for_binary** 必须设置为 **false**。

[提交 bug 报告](#)

3.2.2.8. 修改运行在群集环境里的移植的 Seam 和 Hibernate 应用程序的持久化属性

如果您在移植 JPA 容器管理的应用程序，影响扩展持久化上下文序列化的属性会自动传递给容器。

然而，由于 Hibernate 的修改，如果移植的 Seam 或 Hibernate 应用程序在群集环境里运行，您可能会遇到序列化问题。您会看到类似于这样的错误日志消息：

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

要纠正这些错误，您需要修改配置文件里的属性。在多数情况下是 **persistence.xml** 文件。对于原生的 Hibernate API 应用程序来说是 **hibernate.cfg.xml** 文件。

过程 3.15. 设置持久化属性以运行在群集环境里

1. 设置 **hibernate.session_factory_name** 为唯一的名字。这个名字必须在 JBoss EAP 实例里的所有应用程序部署里是唯一的。例如：

```
<property name="hibernate.session_factory_name" value="jboss-seam-
booking.ear_session_factory"/>
```

2. 设置 **hibernate.ejb.entitymanager_factory_name** 为唯一的名字。这个名字必须在 JBoss EAP 实例里的所有应用程序部署里是唯一的。例如：

```
<property name="hibernate.ejb.entitymanager_factory_name"
value="seam-booking.ear_PersistenceUnitName"/>
```

关于 Hibernate JPA 持久化单元属性设置的更多信息，请参考 [第 3.2.2.3 节“持久化单元属性”](#)。

[提交 bug 报告](#)

3.2.2.9. 更新您的应用程序以遵循 JPA 2.0 规格

介绍

JPA 2.0 规格要求持久化上下文不能在 JTA 事务外部传播。如果您的应用程序只使用事务范围的持久化上下文，JBoss EAP 6 里这个行为和之前版本的一样，无需进行修改。然而，如果您的应用程序使用了扩展的持久化上下文（XPC）以允许数据修改的排队或批处理，您可能需要修改您的应用程序。

持久化上下文的传播行为

如果您的应用程序有一个 stateful session bean **Bean1**，它具有扩展的持久化上下文，并调用另一个使用事务范围的持久化上下文的 stateless session bean **Bean2**，您会看到下列结果：

- 如果 **Bean1** 启动了一个 JTA 事务并在 JTA 事务活动时调用了 **Bean2** 的方法，JBoss EAP 6 里的行为将和之前版本一样，无需进行修改。
- 如果 **Bean1** 没有启动 JTA 事务且调用 **Bean2** 的方法，JBoss EAP 6 则不会将扩展的持久化上下文传播到 **Bean2**。这个行为和会将持久化上下文传播至 **Bean2** 的以前版本不一样。如果您的应用程序期望用事务性实体管理者传播扩展持久化上下文，您需要修改应用程序以在活动的 JTA 事务里进行调用。

[提交 bug 报告](#)

3.2.2.10. 用 Infinispan 替换 JPA/Hibernate 二级缓存

介绍

对于二级缓存（2LC）而言，JBoss Cache 已经被 Infinispan 所替代。这要求修改 **persistence.xml** 文件。其语法稍有不同，这取决于您使用 JPA 还是 Hibernate 二级缓存。下面的例子假设您在使用 Hibernate。

这是 JBoss EAP 5.x 里用 **persistence.xml** 文件指定二级缓存属性的例子。

```
<property name="hibernate.cache.region.factory_class"
value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

下面的步骤将使用这个例子来配置 JBoss EAP 6 里的 Infinispan。

过程 3.16. 修改 persistence.xml 文件以使用 Infinispan

1. 为 JBoss EAP 6 里的 JPA 应用程序配置 Infinispan

这是如何用 JBoss EAP 6 里的 Infinispan 来指定属性以实现用于 JPA 应用程序的相同配置：

```
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

此外，您需要指定一个值为 **ENABLE_SELECTIVE** 或 **ALL** 的 **shared-cache-mode**：

- **ENABLE_SELECTIVE** 是默认的推荐值。它表示实体不会被缓存，除非您显性地将它标记为可缓存的。

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

- **ALL** 表示实体总是被缓存，即使您将其标记为不可缓存的。

```
<shared-cache-mode>ALL</shared-cache-mode>
```

2. 为 JBoss EAP 6 里的原生 Hibernate 应用程序配置 Infinispan

这是如何用 JBoss EAP 6 里的 Infinispan 来指定用于原生 Hibernate 应用程序的相同配置：

```
<property name="hibernate.cache.region.factory_class"
value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactor
y"/>
<property name="hibernate.cache.infinispan.cachemanager"
value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

您也必须将下列依赖关系添加至 MANIFEST.MF 文件：

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

关于 Hibernate 缓存属性的更多信息，请参考：[第 3.2.2.11 节 “Hibernate Cache 属性”](#)。

[提交 bug 报告](#)

3.2.2.11. Hibernate Cache 属性

表 3.7. 属性

属性名	描述
hibernate.cache.region.factory_class	自定义 CacheProvider 的类名。
hibernate.cache.use_minimal_puts	布尔值。优化二级缓存操作，通过更频繁的读操作来最小化写操作。这个设置对于群集缓存更有用，且在 Hibernate3 里，群集缓存实现是默认启用它的。
hibernate.cache.use_query_cache	布尔值。启用队列缓存。单个的对列仍然需要设置为 cacheable。
hibernate.cache.use_second_level_cache	布尔值。用来完全禁用二级缓存，而它对于指定了<amplt;缓存>>映射的类来说默认是启用的。
hibernate.cache.query_cache_factory	自定义 QueryCache 接口的类名。默认值是内置的 StandardQueryCache 。
hibernate.cache.region_prefix	用于二级缓存区名的前缀。
hibernate.cache.use_structured_entries	布尔值。强制 Hibernate 以更友好的格式在二级缓存里保存数据。

属性名	描述
<code>hibernate.cache.default_cache_concurrency_strategy</code>	当使用 <code>@Cacheable</code> 或 <code>@Cache</code> 时为使用的默认 <code>org.hibernate.annotations.CacheConcurrencyStrategy</code> 设置的名称。而 <code>@Cache(strategy="..")</code> 用来覆盖这个默认值。

[提交 bug 报告](#)

3.2.2.12. 移植到 Hibernate Validator 4

介绍

Hibernate Validator 4.x 采用了完全新的代码，它实现了 [JSR 303 - Bean Validation](#)。从 Validator 3.x 到 4.x 的移植过程非常直观，但在移植时您必须进行一些修改。

过程 3.17. 您可能需要执行一个或多个下面的任务：

1. 访问默认的 `ValidatorFactory`

JBoss EAP 6 捆绑了一个默认的 `ValidatorFactory` 到 `java:comp/ValidatorFactory` 下的 JNDI 上下文。

2. 理解生命周期触发的检验

和 Hibernate Core 4 一起使用时，Hibernate Core 将自动启用基于生命周期的检验。

a. 检验发生在实体 **INSERT**、**UPDATE** 和 **DELETE** 操作上。

b. 您可以配置组通过使用下列属性的事件类型来检验：

- `javax.persistence.validation.group.pre-persist`,
- `javax.persistence.validation.group.pre-update`, 和
- `javax.persistence.validation.group.pre-remove`。

这些属性的值是用逗号隔开的、要检验的组的全限定类名。

Validation 组是 Bean Validation 规格的一个新功能。如果您不想利用这个功能，移植到 Hibernate Validator 4 时无需进行任何修改。

c. 您可以设置 `javax.persistence.validation.mode` 属性为 `none` 来禁用基于生命周期的检验。这个属性的其他值还有 `auto`（默认值），`callback` 和 `ddl`。

3. 配置您的应用程序以使用手动检验

a. 如果您想手动控制检验，您可以用下列方法创建一个 `Validator`：

- 用 `getValidator()` 方法从 `ValidatorFactory` 创建一个 `Validator` 实例。
- 注入 `Validator` 实例到 EJB、CDI Bean 或其他 Java EE 可注入的资源。

b. 您可以使用 `ValidatorFactory.usingContext()` 返回的 `ValidatorContext` 来定制

您的 Validator 实例。使用这个 API，您可以配置一个自定义的 **MessageInterpolator**、**TraversableResolver** 和 **ConstraintValidatorFactory**。Bean Validator 规格里指定了这些接口，在 Hibernate Validator 4 里它们是新的接口。

4. 修改代码以使用新的 Bean Validation 约束

当移植到 Hibernate Validator 4 时新的 Bean 级别的检验约束需要修改代码。

- a. 要升级到 Hibernate Validator 4，您必须使用下列包里的约束：

- `javax.validation.constraints`
- `org.hibernate.validator.constraints`

- b. Hibernate Validator 3 里所有的约束在 Hibernate Validator 4 里仍然可用。要使用它们，您需要导入指定的类，且在某些情况下，修改其约束参数的名称或类型。

5. 使用自定义的约束

在 Hibernate Validator 3 里，自定义的约束需要实现

org.hibernate.validator.Validator 接口。在 Hibernate Validator 4 里，您需要实现 **javax.validation.ConstraintValidator** 接口。这个接口包含了和以前接口相同的 **initialize()** 和 **isValid()** 方法，但方法签名有了变动。此外，Hibernate Validator 4 不再支持 **DDL** 修改。

[提交 bug 报告](#)

3.2.3. JSF 的修改

3.2.3.1. 启用应用程序以使用更旧版本的 JSF

介绍

如果您的应用程序使用了更旧版本的 JSF，您不需要升级到 JSF 2.0。相反，您可以创建一个 **jboss-deployment-structure.xml** 文件来请求 JBoss EAP 6 对您的应用程序部署使用 JSF 1.2 而不是 JSF 2.0。这个 JBoss 专有的部署描述符用来控制类加载并放在您的 WAR 的 **META-INF/** 或 **WEB-INF/** 目录里，或者 EAR 的 **META-INF/** 目录里。

下面是一个 **jboss-deployment-structure.xml** 文件，它添加 JSF 1.2 模块的依赖关系且排斥或阻止了 JSF 2.0 模块的自动加载。

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

```
</sub-deployment>
</jboss-deployment-structure>
```

[提交 bug 报告](#)

3.2.4. Web Services 的修改

3.2.4.1. Web Services 的修改

JBoss EAP 6 包含了对部署 JAX-WS Web 服务器端点的支持。这种支持是 JBossWS 提供的。关于 Web Service 的更多信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《部署指南》里的《JAX-WS Web Services》章节。

JBossWS 4 包含了下列可能影响移植的修改。

JBossWS API 的修改

JBossWS 4 里引用了 SPI 和公用组件。下表列出了可能影响应用程序移植的 API 和软件包修改。

表 3.8. JBossWS API 的修改

旧的 JAR	旧的软件包	新的 JAR	新的软件包
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*

旧的 JAR	旧的软件包	新的 JAR	新的软件包
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig
JBossWS 框架	org.jboss.wsf.framework.invocation.RecordingServerHandler	JBossWS Common	org.jboss.ws.common.invocation.RecordingServerHandler

@WebContext Annotation

在 JBossWS 3.4.x 里，这个注解在 JBossWS SPI JAR 被打包为 **org.jboss.wsf.spi.annotation.WebContext**。在 JBossWS 4.0 里，这个注解被移至 JBossWS API JAR 里的 **org.jboss.ws.api.annotation.WebContext**。如果你的应用程序包含了过时的依赖关系，你必须替换源代码里的导入和依赖关系并根据新的 JBossWS API JAR 重新编译。

本版里对于不向后兼容的属性也有改动。**String[] virtualHosts** 属性已经改为 **String virtualHost**。在 JBoss EAP 6 里，对于每个部署您可以只指定一个虚拟主机。如果多个 web service 使用了 **@WebContext** 注解，对于部署归档里定义的所有端点，virtualHost 值都必须是相等的。

端点配置

JBossWS 4.0 提供了 JBoss Web Service 栈和多数 Apache CXF 模块的集成。集成层允许使用标准 webservice API，包括 JAX-WS。它也允许在 JBoss EAP 6 容器上使用 Apache CXF 的高级功能而无需复杂的配置或设置。

JBoss EAP 6 的域配置里的 **webservice** 子系统包含了预定义的端点配置。您也可以定义自己的其他端点配置。**@org.jboss.ws.api.annotation.EndpointConfig** 注解被用来引用给定的端点配置。

关于配置 JBoss 服务器里 webservice 端点的更多信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的『JAX-WS Web Services』章节。

jboss-webservices.xml 部署描述符

JBossWS 4.0 引入了新的部署描述符来配置 web service。**jboss-webservices.xml** 文件为给定的部署提供了其他信息并部分地替换了过时的 **jboss.xml** 文件。

对于 EJB webservice 部署，**jboss-webservices.xml** 描述符文件期望的位置是 **META-INF/** 目录。对于 POJO 和 WAR 里捆绑的 EJB webservice 端点，**jboss-webservices.xml** 文件期望的位置是 **META-INF/** 目录。

下面是 **jboss-webservices.xml** 描述符文件和描述元素的表的例子。

```
<webservices>
  <context-root>foo</context-root>
  <config-name>Standard WSSecurity Endpoint</config-name>
  <config-file>META-INF/custom.xml</config-file>
  <property>
```

```
<name>prop.name</name>
<value>prop.value</value>
</property>
<port-component>
  <ejb-name>TestService</ejb-name>
  <port-component-name>TestServicePort</port-component-name>
  <port-component-uri>/*</port-component-uri>
  <auth-method>BASIC</auth-method>
  <transport-guarantee>NONE</transport-guarantee>
  <secure-wsdl-access>true</secure-wsdl-access>
</port-component>
<webservice-description>
  <webservice-description-name>TestService</webservice-
description-name>
  <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
</webservice-description>
</webservices>
```

表 3.9. jboss-webservice.xml 文件元素描述

名称	描述
context-root	用来自定义 webservises 部署的根上下文。
config-name config-file	用来关联端点部署和给定的端点配置。端点配置是在引用的配置文件或域配置的 webservices 子系统里指定的。
property	用来设置简单属性的名称/值对以配置 webservice 的栈行为。
port-component	用来自定义 EJB 端点目标 URI 以配置和安全性相关的属性。
webservice-description	用来自定义或覆盖 webservice WSDL 的发布位置。

[提交 bug 报告](#)

3.2.5. JAX-RS 和 RESTEasy 的修改

3.2.5.1. 配置 JAX-RS 和 RESTEasy 的修改

JBoss EAP 6 会自动设置 RESTEasy，所以您不需要自己来配置。因此，您应该从 **web.xml** 文件里删除所有现有的 RESTEasy 配置并替换为下列三个选项之一：

- 1. 子类化 **javax.ws.rs.core.Application** 并使用 **@ApplicationPath** 注解。

这是最简单的选项且不要求任何 XML 配置。在您的应用程序里简单地将 **javax.ws.rs.core.Application** 作为子类并用您想将 JAX-RS 类可用的路径来进行注解。例如：


```
@ApplicationPath("/mypath")
public class MyApplication extends Application {
}
```

在上面的例子里，JAX-RS 资源位于路径 `/MY_WEB_APP_CONTEXT/mypath/`。



注意

注意，路径应该指定为 `/mypath` 而不是 `/mypath/*`。

2. 将 `javax.ws.rs.core.Application` 作为子类并使用 `web.xml` 文件来设立 JAX-RS 映射。

如果您想使用 `@ApplicationPath` 注解，您仍需要将 `javax.ws.rs.core.Application` 子类化。然后您可以在 `web.xml` 文件里设置 JAX-RS 映射。例如：

```
public class MyApplication extends Application {
}

<servlet-mapping>
  <servlet-name>com.acme.MyApplication</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

在上面的例子里，JAX-RS 资源位于路径 `/MY_WEB_APP_CONTEXT/hello`。



注意

您也可以使用这个方法覆盖用 `@ApplicationPath` 注解设置的应用程序路径。

3. 修改 `web.xml` 文件。

如果您不想将 `Application` 子类化，您可以像下面这样在 `web.xml` 文件里设置 JAX-RS 映射：

```
<servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

在上面的例子里，JAX-RS 资源位于路径 `/MY_WEB_APP_CONTEXT/hello`。



注意

当您选择这个选项时，您只需要添加映射。您不需要添加对应的 servlet。服务器将负责自动添加对应的 servlet。

[提交 bug 报告](#)

3.2.6. LDAP Security Realm 的修改

3.2.6.1. 配置 LDAP Security Realm 的修改

在 JBoss EAP 5 里, LDAP 安全区是在 `login-config.xml` 文件里的 `<application-policy>` 元素中配置的。而在 JBoss EAP 6 里, LDAP 安全区是在服务器配置文件里的 `<security-domain>` 里进行配置。对于独立服务器, 配置文件是 `standalone/configuration/standalone.xml`; 如果是在受管域里运行服务器, 这个文件是 `domain/configuration/domain.xml`。

下面是一个 JBoss EAP 5 里 `login-config.xml` 文件中的 LDAP 安全区配置：

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</modul
e-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
      ....
    </login-module>
  </authentication>
</application-policy>
```

这是一个 JBoss EAP 6 里服务器配置文件里的 LDAP 配置的例子：

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          ...
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```



注意

JBoss EAP 6 里修改了 XML parser。在 JBoss EAP 5 里，您像这样指定模块选项为元素内容：

```
<module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</module-option>
```

现在，模块选项必须被指定为带有 "value=" 的元素属性：

```
<module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

[提交 bug 报告](#)

3.2.7. HornetQ 的修改

3.2.7.1. 关于 HornetQ 和 NFS

在多数情况下，如果日志类型使用 NIO，由于同步的锁机制，NFS 并不是一个存储和 HornetQ 一起使用的 JMS 数据的合适方式。然而，在某些情况下 NFS 可以用于红帽企业版 Linux 服务器。这是因为红帽企业版 Linux 使用的 NFS 实现所致。

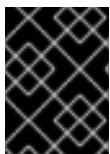
红帽企业版 Linux NFS 实现支持直接 I/O（设置 O_DIRECT 标记并打开文件）以及基于内核的异步 I/O。使用这些功能，依据严格的配置规则，您可以将 NFS 做为一个共享存储选项使用。

- 红帽企业版 Linux NFS 客户缓存必须被禁用。



重要

您应该在 JBoss EAP 6 启动后检查服务器日志，以确保成功加载原生库且使用 ASYNCIO 日志类型。如果原生库加载失败，HornetQ 将使用 NIO 日志类型并在服务器日志里注明。



重要

实现异步 I/O 的原生库要求将 **libaio** 安装在 JBoss EAP 6 所运行的红帽企业版 Linux 系统上。

[提交 bug 报告](#)

3.2.7.2. 配置 JMS 桥以移植现有的 JMS 消息到 JBoss EAP 6

JBoss EAP 6 将 JBoss Messaging 替换为 HornetQ 以作为默认的 JMS 实现。从一个环境移植 JMS 消息到另外一个环境的最简单的方法是使用 JMS 桥。JMS 桥的功能是从源 JMS 目的地消费消息并发送到目标 JMS 目的地。您可以配置和部署 JMS 桥到 JBoss EAP 5.x 或 JBoss EAP 6.1 服务器及更高版本里。

关于如何把 JMS 消息从 JBoss EAP 5.x 移植到 JBoss EAP 6.x 的细节，请参考[第 3.2.7.3 节“创建 JMS 桥”](#)

[提交 bug 报告](#)

3.2.7.3. 创建 JMS 桥

概况

JMS 桥消费源 JMS 队列或主题里的消息并发送到目标 JMS 队列或主题（通常位于不同的服务器上）。它可以用于桥接任何 JMS 服务器间的消息，只要这些消息是兼容 JMS 1.1 的。源和目的 JMS 资源通过 JNDI 查找，用于 JNDI 查找的客户类必须捆绑在模块里。然后在 JMS 桥配置里声明模块名。

过程 3.18. 创建 JMS 桥

这个过程演示了如何配置 JMS 桥从 JBoss EAP 5.x 移植消息到 JBoss EAP 6 服务器。

1. 配置源 JBoss EAP 5.x 服务器上的 JMS 桥

为了避免版本间类的冲突，您必须使用下列过程配置 JBoss EAP 5.x 上的 JMS 桥。SAR 目录和桥的名称可以是任意的，也可以进行修改。

- a. 在 JBoss EAP 5 的 deployment 目录里创建一个子目录来包含 SAR，如 `EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar`。
- b. 在 `EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/` 里创建一个名为 `META-INF` 的子目录。
- c. 在 `EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/META-INF/` 目录里创建一个包含类似于下列信息的 `jboss-service.xml` 文件。

```
<server>
  <loader-repository>
    com.example:archive=unique-archive-name
    <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
  </loader-repository>

  <!-- JBoss EAP 6 JMS Provider -->
  <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAp
plicationPlatform6JMSProvider">
    <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</at
tribute>
    <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapte
r</attribute>
    <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute name="Properties">

    java.naming.factory.initial=org.jboss.naming.remote.client.Initia
lContextFactory

    java.naming.provider.url=remote://EnterpriseApplicationPlatform6h
ost:4447

    java.naming.security.principal=jbossuser
```

```

        java.naming.security.credentials=jbosspass
    </attribute>
</mbean>

<mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
    <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=JMSProvider</depends>
    <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=EnterpriseApplicationPlatform6JMSProvider</depends>
    <attribute
name="SourceDestinationLookup">/queue/A</attribute>
    <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
    <attribute name="QualityOfServiceMode">1</attribute>
    <attribute name="MaxBatchSize">1</attribute>
    <attribute name="MaxBatchTime">-1</attribute>
    <attribute name="FailureRetryInterval">60000</attribute>
    <attribute name="MaxRetries">-1</attribute>
    <attribute name="AddMessageIDInHeader">false</attribute>
    <attribute name="TargetUsername">jbossuser</attribute>
    <attribute name="TargetPassword">jbosspass</attribute>
</mbean>
</server>

```



注意

使用 **load-repository** 是为了确保 SAR 具有独立的类加载器。请注意，JNDI 查找和桥 "target" 都包括用户 "jbossuser"（密码 "jbosspass"）的安全性凭证。这是因为 JBoss EAP 6 默认是设置了安全性的。用户 "jbossuser" 和密码 "jbosspass" 都是在具有 **guest** 角色的 **ApplicationRealm** 里使用 **EAP_HOME/bin/add_user.sh** 脚本创建的。

- d. 将下列 JAR 从 **EAP_HOME/modules/system/layers/base/** 目录复制到 **EAP5_HOME/server/PROFILE_NAME/deploy/myBridge.sar/** 目录。用实际的版本号替换每个 **VERSION_NUMBER**。

- **org/hornetq/main/hornetq-core-VERSION_NUMBER.jar**
- **org/hornetq/main/hornetq-jms-VERSION_NUMBER.jar**
- **org/jboss/ejb-client/main/jboss-ejb-client-VERSION_NUMBER.jar**
- **org/jboss/logging/main/jboss-logging-VERSION_NUMBER.jar**
- **org/jboss/logmanager/main/jboss-logmanager-VERSION_NUMBER.jar**
- **org/jboss/marshalling/main/jboss-marshalling-VERSION_NUMBER.jar**

- `org/jboss/marshalling/river/main/jboss-marshalling-
river-VERSION_NUMBER.jar`
- `org/jboss/remote-naming/main/jboss-remote-
naming-VERSION_NUMBER.jar`
- `org/jboss/remoting3/main/jboss-remoting-VERSION_NUMBER.jar`
- `org/jboss/sasl/main/jboss-sasl-VERSION_NUMBER.jar`
- `org/jboss/netty/main/netty-VERSION_NUMBER.jar`
- `org/jboss/remoting3/remote-jmx/main/remoting-
jmx-VERSION_NUMBER.jar`
- `org/jboss/xnio/main/xnio-api-VERSION_NUMBER.jar`
- `org/jboss/xnio/nio/main.xnio-nio-VERSION_NUMBER.jar`



注意

请不要简单地复制 `EAP_HOME/bin/client/jboss-client.jar`，因为 javax API 类会和 JBoss EAP 5.x 里的类相冲突。

2. 配置目的 JBoss EAP 6 服务器上的 JMS 桥

在 JBoss EAP 6.1 以后的版本里，JMS 桥可以用于从任何兼容 JMS 1.1 的服务器上桥接消息。因为源和目标 JMS 资源是用 JNDI 进行查找的，源消息供应商或消息中介的 JNDI 查找类必须捆绑在 JBoss 模块里。下面的过程使用了虚拟的 'MyCustomMQ' 消息中介作为例子。

a. 为消息供应商创建 JBoss 模块。

- i. 在 `EAP_HOME/modules/system/layers/base/` 下为新的模块创建一个目录结构。`main/` 子目录将包含客户 JAR 和 `module.xml` 文件。下面是为消息供应商 MyCustomMQ 创建的一个目录结构示例：`EAP_HOME/modules/system/layers/base/org/mycustommq/main/`。

- ii. 在 `main/` 子目录里，创建一个包含消息供应商的模块定义的 `module.xml` 文件。下面是为消息供应商 MyCustomMQ 创建的 `module.xml` 示例。

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <!-- Insert resources required to connect to the
source or target -->
    <resource-root path="mycustommq-1.2.3.jar" />
    <resource-root path="mylogapi-0.0.1.jar" />
  </resources>

  <dependencies>
    <!-- Add the dependencies required by JMS Bridge code
-->
```

```

        <module name="javax.api" />
        <module name="javax.jms.api" />
        <module name="javax.transaction.api"/>
        <!-- Add a dependency on the org.hornetq module since
we send      -->
        <!-- messages to the HornetQ server embedded in the
local EAP instance -->
        <module name="org.hornetq" />
    </dependencies>
</module>

```

- iii. 从源资源复制消息供应商用于 JNDI 查找的 JAR 到模块的 **main/** 子目录。
MyCustomMQ 模块的目录结构应该类似于：

```

modules/
  -- system
    -- layers
      -- base
        -- org
          -- mycustommq
            -- main
              -- mycustommq-1.2.3.jar
              -- mylogapi-0.0.1.jar
              -- module.xml

```

- b. 配置部署到 JBoss EAP 6 服务器的 **messaging** 子系统的 JMS 桥。

- i. 在开始之前，请先停止服务器并备份当前的服务器配置文件。如果是作为独立服务器运行，它是 **EAP_HOME/standalone/configuration/standalone-full-ha.xml**。如果是运行的受管域，请备份 **EAP_HOME/domain/configuration/domain.xml** 以及 **EAP_HOME/domain/configuration/host.xml**。
- ii. 在服务器配置文件里的 **messaging** 子系统里添加 **jms-bridge** 元素。**source** 和 **target** 元素提供的用于 JNDI 查找的 JMS 资源的名称。如果指定了 **user** 和 **password** 凭证，当创建 JMS 连接时它们会作为参数传入。

下面是一个为消息供应商 MyCustomMQ 配置的 **jms-bridge** 元素例子：

```

<subsystem xmlns="urn:jboss:domain:messaging:1.3">
    ...
    <jms-bridge name="myBridge" module="org.mycustommq">
        <source>
            <connection-factory name="ConnectionFactory"/>
            <destination name="sourceQ"/>
            <user>user1</user>
            <password>pwd1</password>
            <context>
                <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
                <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
            </context>
        </source>
        <target>

```

```

        <connection-factory name="java:/ConnectionFactory"/>
        <destination name="/jms/targetQ"/>
    </target>
    <quality-of-service>DUPLICATES_OK</quality-of-service>
    <failure-retry-interval>500</failure-retry-interval>
    <max-retries>1</max-retries>
    <max-batch-size>500</max-batch-size>
    <max-batch-time>500</max-batch-time>
    <add-messageID-in-header>true</add-messageID-in-header>
    </jms-bridge>
</subsystem>

```

在上面的例子里，JNDI 属性是在 **source** 的 **context** 元素里定义的。如上面的 **target** 例子，如果忽略了 **context** 元素，JMS 资源将在本地实例里进行查找。

[提交 bug 报告](#)

3.2.7.4. 移植您的应用程序以将 HornetQ 用作 JMS 提供者

JBoss EAP 6 里不再包含 JBoss Messaging。如果您的应用程序将 JBoss Messaging 用作消息提供者，您需要用 HornetQ 替换 JBoss Messaging。

过程 3.19. 在开始之前

1. 关闭客户和服务端。
2. 备份任何 JBoss Messaging 数据。消息数据保存在数据库中前缀为 **JBM_** 的表里。

过程 3.20. 修改提供者 为 HornetQ

1. 转移配置

请转移现有的 JBoss Messaging 配置到 JBoss EAP。下面的配置可在位于 JBoss Messaging 服务器的上的部署描述符里找到：

- 连接工厂服务的配置

这个配置描述了和 JBoss Messaging 服务器一起部署的 JMS 连接工厂。JBoss Messaging 用一个名为 **connection-factories-service.xml** 的文件来配置连接工厂，这个文件位于应用服务器的 **deployment** 目录里。

- 目的地配置

这个配置描述了和 JBoss Messaging 一同部署的 JMS 队列和主题。在默认情况下，JBoss Messaging 在一个名为 **destinations-service.xml** 的文件里配置目的地，这个文件位于应用服务器的 **deployment** 目录下。

- 消息桥服务配置

这个配置描述了和 JBoss Messaging 服务器一起部署的桥服务。在默认情况下不会部署桥，所以部署文件的名称根据 JBoss Messaging 安装而有所不同。

2. 修改您的程序代码

如果应用程序代码使用了标准的 JMS，您不需要修改代码。然而，如果应用程序将连至群集，您必须小心地查看 HornetQ 文档里关于群集模式的内容。群集超出了 JMS 规格和 HornetQ 的范围，JBoss Messaging 采用了很不一样的方法来实现群集功能。

如果应用程序使用 JBoss Messaging 专有的功能，您必须修改代码以使用 HornetQ 里相等的功能。

关于如何用 HornetQ 配置消息系统，请参考 [第 3.2.7.5 节“用 HornetQ 配置消息系统”](#)。

3. 移植现有的消息

使用 JMS 桥移动 JBoss Messaging 数据库里的任何消息到 HornetQ 日志。配置 JMS 桥的说明可以在这里找到：[第 3.2.7.2 节“配置 JMS 桥以移植现有的 JMS 消息到 JBoss EAP 6”](#)。

[提交 bug 报告](#)

3.2.7.5. 用 HornetQ 配置消息系统

我们推荐在 JBoss EAP 6 里配置消息系统的方法是通过管理控制台或管理 CLI。您可以用这些工具来进行持久性的修改而无需手动编辑 `standalone.xml` 或 `domain.xml` 文件。但熟悉默认配置文件的消息组件是有用处的，而使用管理工具的示例的文档里包含了配置文件的片段以供参考。

[提交 bug 报告](#)

3.2.8. 群集的修改

3.2.8.1. 修改应用程序以用于群集环境

1. 启动启用了群集的 JBoss EAP 6

要在 JBoss EAP 5.x 里启用群集，您需要使用 **all** 配置集或其衍生配置集来启动服务器实例，如：

```
$ EAP5_HOME/bin/run.sh -c all
```

在 JBoss EAP 6 里，启用群集的方法取决于服务器是独立的还是运行在受管域里。

a. 为运行在受管域里的服务器启用群集

要为使用域控制器启动的服务器的群集，请更新您的 `domain.xml` 并指定一个服务器组来使用 **ha** 配置集和 **ha-sockets** 套接字绑定组。例如：

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-group>
```

b. 为独立服务器启用群集

要为独立服务器启用群集，用下列配置文件启动服务器：

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME
```

2. 指定绑定地址

在 JBoss EAP 5.x 里，您通常要用 **-b** 命令行参数来指定用于群集的绑定地址：


```
$ EAP5_HOME/bin/run.sh -c all -b 192.168.0.2
```

JBoss EAP 6 绑定套接字到 **standalone.xml**、**domain.xml** 和 **host.xml** 文件里 **<interfaces>** 元素的 IP 地址和接口。JBoss EAP 6 附带的标准配置包含两个接口配置：

```
<interfaces>
  <interface name="management">
    <inet-address
value="{jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="{jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

这些接口配置使用了系统属性 **jboss.bind.address.management** 和 **jboss.bind.address** 的值。如果没有设置这些系统属性，默认的 **127.0.0.1** 将被使用。

您也可以在启动服务器时指定绑定地址为命令行参数，或者您可以在 JBoss EAP 6 服务器配置文件里显性地定义它。

- 在启动 JBoss EAP 独立服务器时在命令行指定绑定参数。

下面是一个如何在命令行为独立服务器指定绑定地址的例子：

```
EAP_HOME/bin/standalone.sh -Djboss.bind.address=127.0.0.1
```

注意

您也可以使用 **-b** 参数，它是 **-Djboss.bind.address=127.0.0.1** 的快捷方式。

```
EAP_HOME/bin/standalone.sh -b=127.0.0.1
```

JBoss EAP 5 的语法格式仍被支持：

```
EAP_HOME/bin/standalone.sh -b 127.0.0.1
```

请注意 **-b** 参数只会修改 **public** 接口。它不会影响 **management** 接口。

- 在服务器配置文件里指定绑定地址。

对于运行在受管域里的服务器，请在 **domain/configuration/host.xml** 文件里指定绑定地址。对于独立服务器，请在 **standalone-ha.xml** 文件里指定绑定地址。

在下面的例子里，**public** 接口被指定为 **ha-sockets** 套接字绑定组里的所有套接字的默认接口。

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
  </interface>
  <interface name="public">
```

```
<inet-address value="192.168.0.2"/>
</interface>
</interfaces>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
    interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```



注意

如果您用硬编码值而不是配置文件里的系统属性来指定绑定地址，那您无法用命令行参数来覆盖它。

3. 配置 `jvmRoute` 以支持 `mod_jk` 和 `mod_proxy`

在 JBoss EAP 5 里，web 服务器 `jvmRoute` 是用 `server.xml` 文件里的一个属性进行配置的。在 JBoss EAP 6 里，`jvmRoute` 属性使用 `instance-id` 属性在服务器配置文件的 web subsystem 部分进行配置。

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
  server="default-host" native="false" instance-id="
    {JVM_ROUTE_SERVER}">
```

上面的 `{JVM_ROUTE_SERVER}` 应该用 `jvmRoute` 服务器 ID 替换。

`instance-id` 也可以用管理控制台来设置。

4. 指定多点传送地址和端口

在 JBoss EAP 5.x 里，您可以使用命令行参数 `-u` 和 `-m` 分别指定用于群集间通讯的多点传送地址和端口，如：

```
$ EAP5_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688
```

在 JBoss EAP 6 里，用于群集间通讯的多点传送地址和端口是用相关的 JGroups 协议栈引用的 `socket-binding` 定义的。

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <!-- ... -->
  </stack>
</subsystem>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
    interface="public">
    <!-- ... -->
    <socket-binding name="jgroups-udp" port="55200" multicast-
      address="228.11.11.11" multicast-port="45688"/>
    <!-- ... -->
```

```
</socket-binding-group>
</socket-binding-groups>
```

如果您想在命令行指定多点传送地址和端口，您可以将多点传送地址和端口定义为系统属性并在启动服务器时在命令行使用这些属性。在下面的例子里，`jboss.mcast.addr` 是多点传送地址的变量名，而 `jboss.mcast.port` 是端口的变量名。

```
<socket-binding name="jgroups-udp" port="55200"
  multicast-address="${jboss.mcast.addr:230.0.0.4}" multicast-
  port="${jboss.mcast.port:45688}"/>
```

您可以用下列命令行参数启动服务器：

```
$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -
  Djboss.mcast.port=45688
```

5. 使用其他的协议栈

在 JBoss EAP 5.x 里，您可以使用 `jboss.default.jgroups.stack` 系统属性来操纵用于群集服务的默认协议栈。

```
$ EAP5_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp
```

在 JBoss EAP 6 里，默认的协议栈是由 `domain.xml` 或 `standalone-ha.xml` 里的 JGroups 子系统定义的：

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <!-- ... -->
  </stack>
</subsystem>
```

6. 替换 Buddy Replication

JBoss EAP 5.x 使用 JBoss Cache Buddy Replication 来抑制将数据复制到群集里的所有实例。

在 JBoss EAP 6 里，Buddy Replication 已被 Infinispan 的分布式缓存（也称为 **DIST** 模式）所替代。Distribution 是一个强大的群集模式，它允许 Infinispan 在添加更多服务器到群集里时进行线性扩充。下面是配置服务器使用 DIST 缓存模式的例子。

- a. 打开命令行窗口并用 HA 或 Full 配置集启动服务器，例如：

```
EAP_HOME/bin/standalone.sh -c standalone-ha.xml
```

- b. 打开另外一个命令行窗口并连接至管理 CLI。

- 对于 Linux，输入下列命令：

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

- 对于 Windows，输入下列命令：

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

您应该看到下面的结果：

```
Connected to standalone controller at localhost:9999
```

c. 运行下列命令：

```
/subsystem=infinispan/cache-container=web/:write-attribute(name=default-cache,value=dist)
/subsystem=infinispan/cache-container=web/distributed-cache=dist/:write-attribute(name=owners,value=3)
:reload
```

您应该在每条命令后看到下面的结果：

```
"outcome" => "success"
```

这些命令修改了 `standalone-ha.xml` 文件的 `infinispan` 子系统的 `web <cache-container>` 配置里的 `dist <distributed-cache>` 元素：

```
<cache-container name="web" aliases="standard-session-cache"
default-cache="dist"
module="org.jboss.as.clustering.web.infinispan">
  <transport lock-timeout="60000"/>
  <replicated-cache name="repl" mode="ASYNC" batching="true">
    <file-store/>
  </replicated-cache>
  <replicated-cache name="sso" mode="SYNC" batching="true"/>
  <distributed-cache name="dist" owners="3" l1-lifespan="0"
mode="ASYNC" batching="true">
    <file-store/>
  </distributed-cache>
</cache-container>
```

更多的信息请参考客户门户

https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的『Web 应用程序里的群集』章节。

[提交 bug 报告](#)

3.2.8.2. 实现 HA 单点登录

总结

下面的过程演示了如何部署用 `SingletonService decorator` 包裹并用作群集内的单点登录服务的 `Service`。这个服务激活了群集里只启动一次的定时器。

过程 3.21. 实现 HA 单点登录服务

1. 编写 HA 单点登录服务应用程序。

下面是一个将部署为单点登录服务的用 `SingletonService decorator` 包裹的 `Service` 的简单例子。完整的例子请参考 JBoss EAP 6 附带的 `cluster-ha-singleton quickstart`。这个 Quickstart 包含构建和部署应用程序的所有说明。

a. 创建服务。

下面是一个服务的示例：

```
package org.jboss.as.quickstarts.cluster.hsingleton.service.ejb;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jboss.logging.Logger;
import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class HATimerService implements Service<String> {
    private static final Logger LOG =
        Logger.getLogger(HATimerService.class);
    public static final ServiceName SINGLETON_SERVICE_NAME =
        ServiceName.JBOSS.append("quickstart", "ha", "singleton",
            "timer");

    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
        AtomicBoolean(false);

    /**
     * @return the name of the server node
     */
    public String getValue() throws IllegalStateException,
        IllegalArgumentException {
        LOG.infof("%s is %s at %s",
            HATimerService.class.getSimpleName(), (started.get() ? "started"
                : "not started"), System.getProperty("jboss.node.name"));
        return "";
    }

    public void start(StartContext arg0) throws StartException {
        if (!started.compareAndSet(false, true)) {
            throw new StartException("The service is still
started!");
        }
        LOG.info("Start HASingleton timer service '" +
            this.getClass().getName() + "'");

        final String node =
            System.getProperty("jboss.node.name");
```

```

        try {
            InitialContext ic = new InitialContext();
            ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).initialize("HASingleton timer @" +
node + " " + new Date());
        } catch (NamingException e) {
            throw new StartException("Could not initialize
timer", e);
        }
    }

    public void stop(StopContext arg0) {
        if (!started.compareAndSet(true, false)) {
            LOGGER.warn("The service '" +
this.getClass().getName() + "' is not active!");
        } else {
            LOGGER.info("Stop HASingleton timer service '" +
this.getClass().getName() + "'");
            try {
                InitialContext ic = new InitialContext();
                ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).stop();
            } catch (NamingException e) {
                LOGGER.error("Could not stop timer", e);
            }
        }
    }
}

```

b. 创建一个将 **Service** 安装为群集单点登录服务的激活器。

下面的列表是一个服务激活器的例子，它将 **HATimerService** 安装为群集单点登录服务：

```

package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.logging.Logger;
import org.jboss.msc.service.DelegatingServiceContainer;
import org.jboss.msc.service.ServiceActivator;
import org.jboss.msc.service.ServiceActivatorContext;
import org.jboss.msc.service.ServiceController;

/**
 * Service activator that installs the HATimerService as a
 * clustered singleton service
 * during deployment.
 *
 * @author Paul Ferraro
 */
public class HATimerServiceActivator implements ServiceActivator
{

```

```

private final Logger log = Logger.getLogger(this.getClass());

@Override
public void activate(ServiceActivatorContext context) {
    log.info("HATimerService will be installed!");

    HATimerService service = new HATimerService();
    SingletonService<String> singleton = new
SingletonService<String>(service,
HATimerService.SINGLETON_SERVICE_NAME);
    /*
     * To pass a chain of election policies to the singleton,
    for example,
     * to tell JGroups to prefer running the singleton on a
    node with a
     * particular name, uncomment the following line:
     */
    // singleton.setElectionPolicy(new
PreferredSingletonElectionPolicy(new
SimpleSingletonElectionPolicy(), new
NamePreference("node2/cluster")));

    singleton.build(new
DelegatingServiceContainer(context.getServiceTarget(),
context.getServiceRegistry()))
        .setInitialMode(ServiceController.Mode.ACTIVE)
        .install()
    ;
}
}

```



注意

上面的代码示例使用了

org.jboss.as.clustering.singleton.SingletonService 类，它是 JBoss EAP 私有 API 的一部分。JBoss EAP 7 里将有可用的公共 API，私有 API 会被废弃，但在 EAP 6.x 发行周期里我们仍将维护这个类。

c. 创建 **ServiceActivator** 文件

在应用程序的 **resources/META-INF/services/** 目录里创建一个名为

org.jboss.msc.service.ServiceActivator 的文件。请添加包含之前步骤里创建的 **ServiceActivator** 类的全限定名称的一行内容。

```
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerServiceActivator
```

d. 创建一个 **Singleton bean**，它实现了用作群集内 **Singleton Timer** 的 **Timer**。

这个 **Singleton bean** 不能有 **remote** 接口，而且您不能从任何应用程序里的其他 **EJB** 引用其 **local** 接口。这阻止了客户或其他组件的查找，从而确保 **SingletonService** 可以完全控制单点登录。

i. 创建 **Scheduler** 接口

```

package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 Fink</a>
 */
public interface Scheduler {

    void initialize(String info);

    void stop();

}

```

ii. 创建实现群集内 Singleton Timer 的 Singleton bean。

```

package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import javax.annotation.Resource;
import javax.ejb.ScheduleExpression;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerConfig;
import javax.ejb.TimerService;

import org.jboss.logging.Logger;

/**
 * A simple example to demonstrate a implementation of a
 cluster-wide singleton timer.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 Fink</a>
 */
@Singleton
public class SchedulerBean implements Scheduler {
    private static Logger LOGGER =
Logger.getLogger(SchedulerBean.class);
    @Resource
    private TimerService timerService;

    @Timeout
    public void scheduler(Timer timer) {
        LOGGER.info("HASingletonTimer: Info=" +
timer.getInfo());
    }

    @Override
    public void initialize(String info) {
        ScheduleExpression sexpr = new ScheduleExpression();
        // set schedule to every 10 seconds for demonstration
    }
}

```



```

        sexpr.hour("*").minute("*").second("0/10");
        // persistent must be false because the timer is
        started by the HASingleton service
        timerService.createCalendarTimer(sexpr, new
TimerConfig(info, false));
    }

    @Override
    public void stop() {
        LOGGER.info("Stop all existing HASingleton timers");
        for (Timer timer : timerService.getTimers()) {
            LOGGER.trace("Stop HASingleton timer: " +
timer.getInfo());
            timer.cancel();
        }
    }
}

```

2. 启动每个启用了群集的 JBoss EAP 6 实例。

要为独立服务器启用群集，您必须用 **HA** 配置集启动每个服务器并对每个实例使用唯一的节点名称和端口偏移量。

- 对于 Linux，请使用下列命令行语法启动服务器：

```

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

```

例 3.3. 在 Linux 上启动多个独立服务器实例

```

$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node1
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100

```

- 对于 Microsoft Windows，请使用下列命令行语法启动服务器：

```

EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET

```

例 3.4. 在 Microsoft Windows 上启动多个独立服务器实例

```

C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node1
C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-
offset=100

```



注意

如果您不想使用命令行参数，您可以为每个服务器实例配置 **standalone-ha.xml** 文件以绑定到单独的接口上。

3. 将应用程序部署到服务器上

下面的 Maven 命令部署应用程序到运行在默认端口的独立服务器上。

```
mvn clean install jboss-as:deploy
```

要部署到其他的服务器，请传入服务器名称。如果位于不同的主机，在命令行上要指定主机名和端口号码：

```
mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -
Djboss-as.port=10099
```

设置 JBoss EAP 6 附带的 **cluster-ha-singleton** quickstart 的 Maven 配置和部署细节。

[提交 bug 报告](#)

3.2.9. 服务风格的部署的修改

3.2.9.1. 更新使用服务风格部署的应用程序

概况

MBean 是以前 Red Hat JBoss EAP 版本的核心架构的一部分。应用服务器通过使用 JBoss 专有的 **jboss-service.xml** 和 **jboss-beans.xml** 服务风格的部署描述符文件的 JBoss Service Archive (SAR) 部署来创建基于 JBoss Bean 的 MBean。JBoss EAP 6 已改变内部的架构，不再基于 MBean JMX 架构，MBean 也不再是核心架构的一部分。它们现在是管理 API 的 Wrapper。

如果您的应用程序使用服务风格的部署描述符，只要它依赖于应用程序定义的 MBean 且不依赖于 JBoss 管理 API，它就可能可以继续运行。在 JBoss EAP 6 里，SAR 只能声明其他 SAR 部署创建的 MBean 的依赖关系。这意味着如果你的应用程序依赖于 JBoss EAP 创建的 MBean，例如用于 EJB 或消息组件的 MBean，它们将无法运行。您可以依赖的 MBean 只能是在其他 **jboss-service.xml files** 里定义的其他 MBean。

以前的 JBoss EAP 版本里使用的 JBoss Service Archive (SAR) 和服务风格的描述符不是 Java EE 6 规格的一部分，我们不推荐将其用于 JBoss EAP 6。我们推荐您按照 Java EE 6 规格修改应用程序。对于 MBean Singleton，您应该修改代码以使用 Java EE 6 **@Singleton**。关于创建和部署 MBean 服务的更多信息，请参考客户门户

https://access.redhat.com/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》的『JBoss MBean 服务』章节。

[提交 bug 报告](#)

3.2.10. 远程调用的修改

3.2.10.1. 将进行远程调用的 JBoss EAP 5 应用程序移植到 JBoss EAP 6

介绍

在 JBoss EAP 5 里, EJB 远程接口绑定在 JNDI 里, 在默认情况下, 本地接口是 "ejbName/local", 而远程接口是 "ejbName/remote"。客户应用程序可以用 "ejbName/remote" 来查找 Bean。

在 JBoss EAP 6 里, 我们引入了新的专有 EJB 客户 API 来执行调用。然而, 如果您不希望重写代码来使用新的 API, 您可以修改现有代码, 使用 `ejb:BEAN_REFERENCE` 语法来远程访问 EJB。

对于 stateless bean, `ejb:BEAN_REFERENCE` 语法是 :

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>
```

对于 stateful bean, `ejb:BEAN_REFERENCE` 语法是 :

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>?stateful
```

在上面的语法里要替换的值是 :

- **<app-name>** - 部署的 EJB 的应用程序名称。这通常是 EJB 部署的 EAR 名称, 不带 .ear 后缀, 但可以在 application.xml 里进行覆盖。如果应用程序没有部署为 .EAR, 这个值是一个空字符串。我们假设这个例子没有部署为 EAR。
- **<module-name>** - 服务器上部署的 EJB 名称。这通常是 EJB 部署的 JAR 名称, 不带 .jar 后缀, 但可以在 ejb-jar.xml 里进行覆盖。在这个例子里, 假设 EJB 是部署在 jboss-ejb-remote-app.jar 里, 所以其模块名是 jboss-ejb-remote-app。
- **<distinct-name>** - EJB 的一个可选的不同名称。这个例子没有使用不同名称, 所以它使用空字符串。
- **<bean-name>** - 默认是 bean 实现类的简单名称。
- **<fully-qualified-classname-of-the-remote-interface>** - the remote view fully qualified class name.

更新客户端代码

假设您已经将下列 stateless EJB 部署到了 JBoss EAP 6 服务器。请注意, 它开放了 bean 的一个远程视图。

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

在 JBoss EAP 5 里，使用上面的信息，客户 EJB 查找和调用的代码类似于：

```
InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);
```

在 JBoss EAP 6 里，使用上面的信息，客户查找和调用的代码类似于：

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-ejb-remote-app";
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);
```

如果您的客户应用程序访问 stateful EJB，您必须在上下文查找后面附加 “?stateful”：

```
final RemoteCalculator statefulRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")
```

您可以在 QuickStarts 里找到一个完整的、包含服务器和客户端代码的可运行的例程。相关的更多信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 部署指南》里的『开发企业级应用程序起步』章节中的『回顾 Quickstart 教程』。

关于使用 JNDI 进行远程调用的更多信息，请参考 第 3.2.10.2 节“用 JNDI 远程调用 Session Bean”。

[提交 bug 报告](#)

3.2.10.2. 用 JNDI 远程调用 Session Bean

这个任务描述了如何为远程客户添加支持以使用 JNDI 调用 Session Bean。这个任务假设该项目是用 Maven 构建的。

ejb-remote Quickstart 包含了演示这个功能的 Maven 项目。这个 Quickstart 包含了要部署的 Session Bean 和远程客户项目。下面的代码示例是从远程客户端程序抽取的。

这个任务假设 Session Bean 不要求进行验证。



警告

Red Hat 推荐您在所有受影响的软件包里显性地禁用 TLSv1.1 或 TLSv1.2 上的 SSL。

先决条件

在开始之前，下列预备条件必须被满足：

- 您必须有一个可用的 Maven 项目。
- 已经添加了用于 JBoss EAP 6 Maven Repository 的配置。
- 已经部署了您要调用的 Session Bean。
- 部署的 Session Bean 实现了远程商业接口。
- Session Bean 的远程商业接口作为 Maven 依赖关系可用。如果远程商业接口只作为 JAR 文件可用，那么我们推荐您将 JAR 添加到 Maven 库作为 artifact 使用。请参考 <http://maven.apache.org/plugins/maven-install-plugin/usage.html> 里的 `install:install-file` 说明。
- 您需要知道这个 session bean 所在的服务器的主机名和 JNDI 端口号。

要从远程客户调用 session bean，您首先必须正确配置这个项目。

过程 3.22. 为 session bean 的远程商业接口添加 Maven 项目依赖关系。

1. 添加所需的项目依赖关系

必须更新项目的 `pom.xml` 以包含必要的依赖关系。

2. 添加 `jboss-ejb-client.properties` 文件

JBoss EJB 客户 API 期望在项目根目录找到名为 `jboss-ejb-client.properties` 的文件，它包含 JNDI 服务的连接信息。添加这个文件到项目的 `src/main/resources/` 目录并包含下列内容。

```
# In the following line, set SSL_ENABLED to true for SSL
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
# Uncomment the following line to set SSL_STARTTLS to true for SSL
#
remote.connection.default.connect.options.org.xnio.Options.SSL_STARTTLS=true
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
# Add any of the following SASL options if required
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY
```

```

CY_NOANONYMOUS=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
CY_NOPLAINTEXT=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
LLOWED_MECHANISMS=JBoss-LOCAL-USER

```

根据您的服务器来修改主机名和端口号。**4447** 是默认的端口号码。对于安全的连接，请将 **SSL_ENABLED** 一行设置为 **true** 并取消 **SSL_STARTTLS** 行的注释。容器里的 Remoting 接口支持使用相同端口的安全和不安全的连接。

3. 为远程商业接口添加依赖关系

在 **pom.xml** 里为 session bean 的远程商业接口添加 Maven 依赖关系。

```

<dependency>
  <groupId>org.jboss.as.quickstarts</groupId>
  <artifactId>jboss-ejb-remote-server-side</artifactId>
  <type>ejb-client</type>
  <version>${project.version}</version>
</dependency>

```

现在已经正确配置了这个项目，您可以添加代码来访问并调用 session bean。

过程 3.23. 用 JNDI 获取 Bean 代理并调用 Bean 的方法

1. 处理 checked 异常

下面代码里的两个方法 (**InitialContext()** 和 **lookup()**) 都会抛出类型为 **javax.naming.NamingException** 的 checked 异常。这些方法调用必须包括在捕获 **NamingException** 的 try/catch 块里，或者是再声明抛出 **NamingException** 的方法里。**ejb-remote** quickstart 使用了第二种方法。

2. 创建一个 JNDI 上下文

JNDI 上下文对象提供了从服务器请求资源的机制。请使用下列代码创建 JNDI 上下文：

```

final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);

```

JNDI 服务的连接属性是从 **jboss-ejb-client.properties** 文件里读取的。

3. 使用 JNDI Context's lookup() 方法来获取 bean 代理

调用 bean 代理的 **lookup()** 方法并将您请求的 session bean 的 JNDI 名称传入。这将返回一个对象，它必须转换为包含您要调用的方法的远程商业接口的类型。

```

final RemoteCalculator statelessRemoteCalculator =
(RemoteCalculator) context.lookup(
"ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
RemoteCalculator.class.getName());

```

Session bean 的 JNDI 名称是用特殊语法定义的。详情请参考 [第 3.2.10.3 节“EJB JNDI 命名引用”](#)。

4. 调用方法

现在您已有一个代理 bean 对象，您可以调用远程商业接口里包含的任何方法了。

```
int a = 204;
int b = 340;
System.out.println("Adding " + a + " and " + b + " via the remote
stateless calculator deployed on the server");
int sum = statelessRemoteCalculator.add(a, b);
System.out.println("Remote calculator returned sum = " + sum);
```

代理 bean 将方法调用请求传入服务器上的 session bean。其结果被返回到代理 bean，然后再返回给调用者。代理 bean 和远程 session bean 间的通讯对于调用者来说是透明的。

您现在应该可以配置一个 Maven 项目来支持调用远程服务器上的 session bean 并编写代码，使用通过 JNDI 从服务器获取的代理 bean 来调用 session bean 方法。

[提交 bug 报告](#)

3.2.10.3. EJB JNDI 命名引用

Session Bean 的 JNDI 查找名称的语法是：

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

<appName>

如果 Session Bean 的 JAR 文件已经部署在 EAR 里，那 JNDI 名称就是 EAR 的名称。在默认情况下，EAR 的名称是不带 **.ear** 后缀的文件名。应用程序名称也可以在 **application.xml** 文件里进行覆盖。如果 Session Bean 没有部署在 EAR 里，那应将它留空。

<moduleName>

这个模块名是 Session Bean 部署所在的 JAR 文件的名称。在默认情况下，JAR 文件的名称是其带 **.jar** 后缀的文件名。模块名也可以在 JAR 的 **ejb-jar.xml** 文件里进行覆盖。

<distinctName>

JBoss EAP 6 允许每个部署指定可选的标识名。如果部署没有标识名，则请留空。

<beanName>

这是要调用的 Session Bean 的类名。

<viewClassName>

这是远程接口的权限定类名。它包括了接口的软件包名。

?stateful

当 JNDI 名称引用 stateful session bean 时，**?stateful** 后缀是必需的。其他 bean 类型没有包含这个后缀。

[提交 bug 报告](#)

3.2.11. EJB 2.x 的修改

3.2.11.1. 更新使用 EJB 2.x 的应用程序

JBoss EAP 6 构建于开放式标准，它和 Java EE 6 规格兼容。虽然应用程序服务器支持 EJB 2.x，但它不会再支持超过这个规格的功能。请记住，Java EE 7 规格已经将 EJB 2.x 标记为可选的，所以我们强烈推荐您根据 EJB 3.x 规格重写您的应用程序。

如果您仍想移植 EJB 2.x 代码，多数情况下您需要修改代码以在 JBoss EAP 6 里运行。本节描述了其中一些修改。

在 JBoss EAP 6 上运行 EJB 2.x 所需的配置修改

用 Full 配置集启动服务器

EJB 2.x Container Managed Persistence (CMP) bean 要求 Java EE 6 Full 配置集。这个配置集包含运行 CMP EJB 所需的配置元素。

这个配置集包含了 `org.jboss.as.cmp` 扩展模块：

```
<extensions>
    ...
    <extension module="org.jboss.as.cmp"/>
    ...
</extensions>
```

它也包含 `cmp` 子系统：

```
<profiles>
    ...
    <subsystem xmlns="urn:jboss:domain:cmp:1.1"/>
    ...
</profiles>
```

要用 Full 配置集启动 JBoss EAP 6 独立服务器，在用命令行启动服务器时请使用 `-c standalone-full.xml` or `-c standalone-full-ha.xml` 参数。

容器配置不再被支持了

在以前的 JBoss EAP 版本里，我们可以为 CMP 实体和其他 bean 配置不同的容器并通过在 `jboss.xml` 部署描述符文件里设置引用来使用它。例如，对于 SLSB 对 Session Bean 的引用通常有不同的配置。

在 JBoss EAP 6.x 里，我们可以将 EJB 2 Entity Bean 和标准容器一起使用。然而，它不再支持不同的容器配置。推荐的做法是移植 EJB2 Stateful Session Bean (SFSB)、Stateless Session Beans (SLSB)、Message Driven Beans (MDB) 到 EJB3，而 Container-Managed Persistence (CMP) 和 Bean-Managed Persistence (BMP) Entity Beans 则按照 EJB3 规格使用 Java Persistence API (JPA)。

JBoss EAP 6 里的默认容器配置包含几个对 EJB2 CMP Bean 的修改：

- 默认情况下悲观锁是活动的。这可能导致死锁。

- JBoss EAP 6 里不再有 JBoss EAP 5.x 里的 CMP 层的死锁检测代码。

在 JBoss EAP 5.x 里，我们也可以在定义缓存、池、**commit-options** 和拦截器栈。而在 JBoss EAP 6 里这是不可能的，它只有一个类似于带有 **commit-option C** 的 **Instance Per Transaction** 策略的实现。如果您移植使用了 **cmp2.x jdbc2 pm** entity bean 容器配置的应用程序（使用了兼容 CMP2.x 的基于 JDBC 持久化管理者），这会对性能有影响。这个容器为性能作了优化。我们推荐您在移植应用程序之前将这些实体移植到 EJB3。

服务器端连接器的配置

JBoss EAP 6 支持使用 **@Interceptors** 和 **@AroundInvoke** 注解的标准 Java EE **Interceptor**。然而，这并不会允许安全性或事务之外的操作。

在以前的 JBoss EAP 版本里，我们可以修改拦截器栈为每个 EJB 调用定制拦截器。这通常用来实现自定义的安全性或安全检查、事务检查或创建前的重试机制。JBoss EAP 6.1 引入了容器拦截器来提供类似的功能。关于容器拦截器的更多信息，请参考《JBoss EAP 开发指南》里的『容器拦截器』章节。

在事务的提交阶段之前、期间、之后提供更多控制且遵循 Java EE 规格的方法是使用事务同步注册表（Transaction Synchronization Registry）来添加 listener。

资源可以用下列方法之一来获取：

- 使用 **InitialContext**

```
TransactionSynchronizationRegistry tsr =
    (TransactionSynchronizationRegistry)
        new
    InitialContext().lookup("java:jboss/TransactionSynchronizationRegi-
        stry");
tsr.registerInterposedSynchronization(new MyTxCallback());
```

- 使用注入

```
@Resource(mappedName =
    "java:comp/TransactionSynchronizationRegistry")
TransactionSynchronizationRegistry tsr;
...
tsr.registerInterposedSynchronization(new MyTxCallback());
```

回调 Routine 必需实现 **javax.transaction.Synchronization** 接口。请在事务提交或回滚前用 **beforeCompletion{}** 方法来执行任何检查。如果这个方法抛出 **RuntimeException**，事务会被回滚且用 **EJBTransactionRolledbackException** 通知客户。如果是 XA-Transaction，所有的资源将按照 XA 合约回滚。我们也可以根据事务状态用 **afterCompletion(int txStatus)** 方法启用商业逻辑。如果这个方法抛出 **RuntimeException**，事务将保持之前的状态，提交或回滚，且客户不会得到通知。只有事务管理者会在服务器日志文件里显示一个警告信息。

客户端拦截器的服务器端配置

在以前的 JBoss EAP 版本里，我们可以在服务器配置文件里配置客户拦截器并只提供带有客户 API 的类。

在 JBoss EAP 6 里这已不可能了，因为服务器端不会再创建客户代理并再查找后传输到客户端。这个代理现在是在客户端生成。这次优化避免了查找和类上传时对服务器的调用。

Entity Bean 池配置

我们不推荐在 JBoss EAP 6 里进行 Entity bean 的池配置。因为它受限于 `<strict-max-pool>` 元素的配置，如果池过小而无法加载结果集里的所有实体，死锁和其他问题就可能发生。Entity Bean 在初始化时没有大型的生命周期方法，所以创建实例和使用容器并不会比池化的 Entity Bean 实例慢。

替换 jboss.xml 部署描述符文件

`jboss-ejb3.xml` 部署描述符文件替换 `jboss.xml` 以覆盖和添加 Java EE 定义的 `ejb-jar.xml` 里提供的功能。这个新文件和 `jboss.xml` 兼容，而目前的部署里已经忽略了 `jboss.xml`。

例如，在以前的 JBoss EAP 版本里，如果您在 `ejb-jar.xml` 文件里定义了 `<resource-ref>`，`jboss.xml` 里的 JNDI 名称需要有对应的资源定义。XDoclet 自动生成这两个部署描述符文件。在 JBoss EAP 6，`jboss-ejb3.xml` 文件里现在已定义了 JNDI 映射信息。我们假定 Java 源代码里的数据源是如下这样定义的。

```
DataSource ds1 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource1");
DataSource ds2 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource2");
```

`ejb-jar.xml` 定义了下列资源引用。

```
<resource-ref >
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
<resource-ref >
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

`jboss-ejb3.xml` 文件用下列 XML 语法映射 JNDI 名称和引用

```
<resource-ref>
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
```

JBoss EAP 6 没有实现 JBoss EAP 5.x `jboss.xml` 文件里可用的一些配置选项。下表描述了 `jboss.xml` 文件里常用的属性以及在 JBoss EAP 6 是否有替代的方法。

- **method-attribute** 元素用来配置单独的 Entity 和 Session Bean 方法。
 - **read-only** 和 **idempotent** 配置选项没有移植到 JBoss EAP 6 里。
 - **transaction-timeout** 选项现在在 `jboss-ejb3.xml` 文件里进行配置。
- **missing-method-permission-exclude-mode** 属性修改了方法的行为，而不用实现 secured bean 上的显性安全元数据。在 JBoss EAP 6 里，`@PermitAll` 注解也会依据和 `@RolesAllowed` 注解类似的方式处理。

数据源类型映射配置

在以前的 JBoss EAP 版本里，我们可以在 `*-ds.xml` 数据源部署配置文件里配置数据源的类型映射。

在 JBoss EAP 6 里，这必须在 `jbosscmp-jdbc.xml` 部署描述符文件里进行。

```
<defaults>
  <datasource-mapping>mySQL</datasource-mapping>
  <create-table>true</create-table>
  ....
</defaults>
```

在以前的 JBoss EAP 版本里，自定义的映射是在 `standardjbosscmp-jdbc.xml` 文件里完成的。这个文件已取消，现在映射是通过 `jbosscmp-jdbc.xml` 部署描述符文件进行的。

其他 Container-Managed Persistence (CMP) 和 Container-Managed Relationship (CMR) 的修改

Container Managed Relationship (CMR) Iterator 和 Collection 的修改

在以前的 JBoss EAP 版本里，一些容器如 `cmp2.x jdbc2 pm` 可以迭代 CMR 容器并删除或添加关系。JBoss EAP 6 不支持这种容器配置，所以无法再这样做了。关于如何在程序代码实现相同功能的信息，请参考客户门户的『Support Knowledgebase Solutions』部分的 [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#)。

用于 Finder 的 Container Managed Relationship (CMR) 重复条目

在以前的 JBoss EAP 版本里，我们可以选择使用了不同持久化策略的 CMP 容器。JBoss EAP 5.x 里的 `cmp2.x jdbc2 pm` 容器使用优化的 `SQL-92` 来为 Finder 生成优化的 LEFT OUTER JOIN 语法。因为 JBoss EAP 6.x 只支持标准的 CMP 和 CMR 容器，其实现没有包含这些优化。Finder 应该在 `SELECT` 语句里包含关键字 `DISTINCT` 来避免结果集出现笛卡尔集合。更多的信息请参考客户门户的『Support Knowledgebase Solutions』部分的 [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#)。

对 CMP Entity Beans 的 Cascade Delete 默认值的修改

Cascade Delete 的默认值已改成 `false`。这可能导致 JBoss EAP 6 里出现删除失败。如果实体关系被标记为 `cascade-delete`，您必须在 `jbosscmp-jdbc.xml` 文件里显性地设置 `batch-cascade-delete` 为 `true`。更多的信息请参考客户门户的『Support Knowledgebase Solutions』部分的 [cascade delete fail for EJB2 CMP Entities after migration to EAP6](#)。

用于自定义字段的 CMP 自定义映射器

如果您的 JBoss EAP 5.x 应用程序里使用了自定义的映射器类，如 `JDBCParameterSetter`、`JDBCResultSetReader` 和 `Mapper`，在部署到 JBoss EAP 6 里时，您可能会看到 `java.lang.ClassNotFoundException`。这是因为这些接口的软件包名从 `org.jboss.ejb.plugins.cmp.jdbc.Mapper` 改成了 `org.jboss.as.cmp.jdbc.Mapper`。更多的信息请参考客户门户的『Support Knowledgebase Solutions』部分的 [How to use Field mapping for custom classes in an EJB2 CMP application in EAP6](#)。

使用 entity-commands 生成主键

如果您的 JBoss EAP 5 应用程序使用了 `entity-commands` 来生成主键，如 `Sequence` 或 `Auto-increment`，当移植应用程序到 JBoss EAP 6 时，您可以看到 `JDBCOracleSequenceCreateCommand` 抛出 `ClassNotFoundException`。这是因为类软件包从

`org.jboss.ejb.plugins.cmp.jdbc` 改成了 `org.jboss.as.cmp.jdbc.keygen`。如果您在 JBoss EAP 6 应用程序里使用了这个类，您必须添加对 `EAP_HOME/modules/system/layers/base/org/jboss/as/cmp` 模块的依赖关系。

应用程序的修改

修改代码以使用新的 JNDI 命名空间规则。

和 EJB 3.0 一样，对 EJB 2.x 您必须使用完整的 JNDI 前缀。关于新的 JNDI 命名空间规则和代码示例，请参考 [第 3.1.8.1 节“更新应用程序 JNDI 命名空间的名称”](#)。

关于如何更新以前版本的 JNDI 命名空间的例子，您可以参考 [第 3.1.8.5 节“以前版本的 JNDI 命名空间示例和它们在 JBoss EAP 6 里是如何指定的”](#)。

修改 `jboss-web.xml` 文件描述符

对每个 `<ejb-ref>` 的 `<jndi-name>` 进行修改以使用新的 JNDI 全限定查找格式。

使用 XDoclet 来映射内部 Local 接口的 JNDI 名称

对于 EJB2，使用 **Locator** 模式来查找 Bean 是非常常见的。如果您在程序里使用了这个模式，不需要修改代码，您可以使用 [XDoclet](#) 来生成新 JNDI 名称的映射。

典型的 XDoclet 注解类似于：

```
@ejb.bean name="UserAttribute" display-name="UserAttribute" local-jndi-name="ejb21/UserAttributeEntity" view-type="local" type="CMP" cmp-version="2.x" primkey-field="id"
```

上述例子中的 JNDI 名称 `ejb21/UserAttributeEntity` 在 JBoss EAP 6 里不再有效。您可以用服务器配置和 XDoclet 补丁里的 **naming** 子系统将这个名称映射到有效的 JNDI 名称。

如上面的『[用于自定义字段的 CMP 自定义映射器](#)』里说提及的，您可以创建自定义的映射器，或者您可以按照下面过程描述的那样修改代码。

过程 3.24. 修改 XDoclet Generated Code 并使用 Naming 子系统

1. 解压 `ejb-module.jar` 里的 XDoclet `lookup.xdt` 模板并修改 `lookupHome` 里的 `lookup()`：

```
private static Object lookupHome(java.util.Hashtable environment,
String jndiName, Class narrowTo) throws
javax.naming.NamingException {
    // Obtain initial context
    javax.naming.InitialContext initialContext = new
    javax.naming.InitialContext(environment);
    try {
        // Replace the existing lookup
        // Object objRef = initialContext.lookup(jndiName);
        // This is the new mapped lookup
        Object objRef;
        try {
            // try JBoss EAP mapping
            objRef = initialContext.lookup("global/"+jndiName);
        } catch (java.lang.Exception e) {
            objRef = initialContext.lookup(jndiName);
        }
    }
}
```

```

    }
    // only narrow if necessary
    if (java.rmi.Remote.class.isAssignableFrom(narrowTo))
        return javax.rmi.PortableRemoteObject.narrow(objRef,
narrowTo);
    else
        return objRef;
    } finally {
        initialContext.close();
    }
}

```

2. 运行 Ant, 对于 **ejbdoclet** 任务设置 **template** 属性以使用改动的 **lookup.xdt**。
3. 修改服务器配置文件里的 **naming** 子系统来映射旧的 JNDI 名称到新的有效 JNDI 名称。

```

<subsystem xmlns="urn:jboss:domain:naming:1.2">
    <bindings>
        <lookup name="java:global/ejb21/UserAttributeEntity"
lookup="java:global/ejb2CMP/ejb/UserAttribute!de.wfink.ejb21.cmp.c
mr.UserAttributeLocalHome"/>
    </bindings>
    <remote-naming/>
</subsystem>

```

废弃的文件概述

JBoss EAP 6 不再支持下列文件。

jboss.xml

Jboss EAP 6 不再支持 **jboss.xml** 部署描述符文件且没有在部署归档里包含它。

standardjbosscmp-jdbc.xml

Jboss EAP 6 不再支持 **standardjbosscmp-jdbc.xml** 配置文件。这个配置信息现在包含在 **org.jboss.as.cmp** 模块里且不再是可以定制的了。

standardjboss.xml

JBoss EAP 6 不再支持 **standardjboss.xml** 配置文件。运行独立服务器时配置信息现在包含在 **standalone.xml** 文件里, 而运行在受管域时包含在 **domain.xml** 文件里。

提交 bug 报告

3.2.12. JBoss AOP 的修改

3.2.12.1. 更新使用 JBoss AOP 的应用程序

JBoss EAP 6 里不再包含 JBoss AOP (Aspect Oriented Programming, 面向方面编程)。在以前的版本里, EJB 容器使用了 JBoss AOP。然而, 在 JBoss EAP 6 里, EJB 容器使用了新的机制。如果您的应用程序使用 JBoss AOP, 您需要象下面这样修改您的代码。

重构应用程序

- **ejb3-interceptors-aop.xml** 里以前的标准 EJB3 配置现在已经在服务器配置文件里完成了。对于独立的服务器，就是 **standalone/configuration/standalone-full.xml** 文件。如果您在受管域里运行服务器，这个文件将是 **domain/configuration/domain.xml**。
- 您应该修改服务器端的 AOP 拦截器来使用标准的 Java EE 拦截器（**Interceptor**）。关于容器拦截器的更多信息以及如何使用客户端拦截器，请参考客户门户 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 开发指南》里的《容器拦截器》章节。

使用 JBoss AOP 库

- 如果您无法重构代码，您可以获取 JBoss AOP 库的拷贝并将其与应用程序捆绑。AOP 库可以在 JBoss EAP 6 里运行，但不能进行部署。您可以使用下列命令行参数启动服务器来进行手动部署：**Djboss.aop.path=PATH_TO_AOP_CONFIG**。



注意

虽然 JBoss AOP 库可以在 JBoss EAP 6 里运行，但它并非受支持的配置。

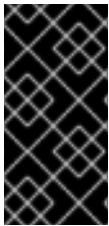
[提交 bug 报告](#)

3.2.13. 移植 Seam 2.2 应用程序

3.2.13.1. 移植 Seam 2.2 归档到 JBoss EAP 6

介绍

当您移植 Seam 2.2 应用程序时，您需要配置数据源并指定任何模块化依赖关系。您也需要确定应用程序是否依赖于 JBoss EAP 6 不附带的归档并将依赖的 JAR 复制到应用程序的 **lib/** 目录。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 3.25. 移植 Seam 2.2 归档

1. 更新数据源配置

某些 Seam 2.2 示例使用名为 **java:/ExampleDS** 的默认 JDBC 数据源。JBoss EAP 6 里已经修改了这个默认的数据源为 **java:jboss/datasources/ExampleDS**。如果您的应用程序使用了 example 数据库，您可以：

- 如果您想使用 JBoss EAP 6 附带的示例数据库，请修改 **META-INF/persistence.xml** 文件以替换现有的 **jta-data-source** 元素为示例数据库数据源 JNDI 名：

```
<!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```


- 如果您像保留现有的数据库，您可以在 `EAP_HOME/standalone/configuration/standalone.xml` 文件里添加数据源定义。



重要

要使修改在服务器重启后仍然生效，您必须在编辑服务器配置文件前停止服务器。

下面是在 JBoss EAP 6 里定义的默认 HSQL 数据源定义：

```
<datasource name="ExampleDS" jndi-name="java:/ExampleDS"
enabled="true" jta="true" use-java-context="true" use-ccm="true">
  <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
  <driver>h2</driver>
  <security>
    <user-name>sa</user-name>
    <password>sa</password>
  </security>
</datasource>
```

- 您也可以用管理 CLI 命令行接口添加数据源定义。下面是您添加数据源所必须使用的语法。结尾的 "\" 表示下一行仍是命令行的一部分。

例 3.5. 添加数据源定义的语法示例

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
--connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
--user-name=sa --password=sa
```

关于如何配置数据源的更多信息，请参考 第 3.1.6.2 节“更新数据源配置”。

2. 添加所需的依赖关系

因为 Seam 2.2 应用程序使用了 JSF 1.2，您需要添加 JSF 1.2 模块的依赖关系并排除 JSF 2.0 模块。为了实现这一点，您需要在包含下列数据的 EAR 的 `META-INF/` 目录下创建一个 `jboss-deployment-structure.xml` 文件：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
  </sub-deployment>
</jboss-deployment-structure>
```

```

    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```

如果您的应用程序使用了第三方的日志框架，您需要添加如下这些依赖关系：[第 3.1.4.1 节“修改日志依赖关系”](#)。

3. 如果您的应用程序使用了 Hibernate 3.x，请首先用 Hibernate 4 库来运行应用程序。

如果您的应用程序没有使用 Seam Managed Persistence Context、Hibernate search、validation 或其他 Hibernate 4 里已修改的功能，您应该可以用 Hibernate 4 库来运行它。然而，如果您看到指向 Hibernate 类的 **ClassNotFoundException** 或 **ClassCastException**，或者看到类似于下面的错误信息，您就可能需要遵循下一步的说明并修改应用程序以使用 Hibernate 3.x 库。

```

    Caused by: java.lang.LinkageError: loader constraint
    violation in interface itable initialization: when resolving method
    "org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hibernate/EntityMode;)Lorg/hibernate/Session;" the class loader
    (instance of org/jboss/modules/ModuleClassLoader) of the current
    class, org/jboss/seam/persistence/HibernateSessionProxy, and the
    class loader (instance of org/jboss/modules/ModuleClassLoader) for
    interface org/hibernate/Session have different Class objects for the
    type org/hibernate/Session used in the signature

```

4. 从外部框架或其他位置复制依赖的归档

如果您的应用程序使用了 Hibernate 3.x 且您无法成功使用 Hibernate 4，您需要像下面这样复制 Hibernate 3.x JAR 到 `/lib` 目录并在 `META-INF/jboss-deployment-structure.xml` 里的 deployment 部分排除 Hibernate 库：

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <exclusions>
      <module name="org.hibernate"/>
    </exclusions>
  </deployment>
</jboss-deployment-structure>

```

当您的应用程序捆绑 Hibernate 3.x 时您还必须采取一些其他的步骤。相关的信息，请参考[第 3.2.2.2 节“修改使用 Hibernate 和 JPA 的应用程序的配置”](#)。

5. 调试和解决 Seam 2.2 JNDI 错误

当您移植 Seam 2.2 应用程序时，您可能在日志里使用 `javax.naming.NameNotFoundException` 错误：

```

javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
found in context ''

```

如果您不想修改代码里的 JNDI 查找，您可以修改应用程序的 `components.xml` 文件：

a. 替换现有的 core-init 元素

首先，您需要替换现有的 `core-init` 元素：

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init debug="true" distributable="false"/>
```

b. 找到服务器日志里的 JNDI 绑定 INFO 消息

然后，在服务器日志里找到应用程序部署时的 JNDI 绑定 INFO 消息。JNDI 绑定消息应该类似于：

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

    java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction
```

c. 添加组件元素

对于日志里的每个 JNDI 绑定 INFO 消息，添加匹配的 `component` 元素 `components.xml` 文件：

```
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />
```

关于如何调试和解决移植问题的更多信息，请参考 [第 4.2.1 节“调试和解决移植问题”](#)。

关于 Seam 2 归档的已知移植问题列表，请参考 [第 3.2.13.2 节“Seam 2.2 归档移植的问题”](#)。

结果

Seam 2.2 归档成功地在 JBoss EAP 6 里运行。

[提交 bug 报告](#)

3.2.13.2. Seam 2.2 归档移植的问题

Seam 2.2 Drools 和 Java 7 是不兼容的

Seam 2.2 Drools 和 Java 7 是不兼容的且会抛出异常 `org.drools.RuntimeDroolsException: value '1.7' is not a valid language level error`。

签注了 `cglib.jar` 的 `Seam 2.2.5` 阻止了 `Spring` 例程的运行

当 `Spring` 例程使用 `JBoss EAP 5` 里 `Seam 2.2.5` 附带的签注的 `cglib.jar` 时，它会运行失败并带有以下错误：

```
java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package
```

解决办法是取消 `cglib.jar` 的签注：

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

`Seambay` 例程会抛出异常 `NotLoggedInException`。

造成这个结果的原因是当处理 `SOAPRequestHandler` 里的消息时，`SOAP` 消息头部为空，从而未设置会话 ID。

变通办法是像 <https://issues.jboss.org/browse/JBPAPP-8376> 里描述的那样，覆盖 `org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound`。

`Seambay` 例程抛出异常 `UnsupportedOperationException`：没有事务。

这个程序错误是由于修改 `JBoss EAP 6` 里的 `UserTransaction` 的 `JNDI` 名称而引起的。

变通办法是像 <https://issues.jboss.org/browse/JBPAPP-8322> 里描述的那样，覆盖 `org.jboss.seam.transaction.Transaction.getUserTransaction`。

`Tasks` 例程抛出 `org.jboss.resteasy.spi.UnhandledException`：无法解码请求的主体。

这个程序错误是由于 `JBoss EAP 5.1.2` 里包含的 `seam-resteasy-2.2.5` 和 `JBoss EAP 6` 里的 `RESTEasy 2.3.1.GA` 不兼容而引起的。

解决办法是按照 <https://issues.jboss.org/browse/JBPAPP-8315> 里所描述的，使用 `jboss-deployment-structure.xml` 从主部署里排除 `resteasy-jaxrs`、`resteasy-jettison-provider` 和 `resteasy-jaxb-provider`，以及从 `jboss-seam-tasks.war` 里排除 `resteasy-jaxrs`、`resteasy-jettison-provider`、`resteasy-jaxb-provider` 和 `resteasy-yaml-provider`。然后您需要在 `EAR` 里包含 `Seam 2.2` 附带的 `RESTEasy` 库。

在 `AJAX` 请求过程中 `org.jboss.seam.core.SynchronizationInterceptor` 和 `stateful` 组件实例 `EJB` 锁之间发生死锁。

显示错误页面 "Caused by javax.servlet.ServletException with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": org.jboss.seam.core.LockTimeoutException: could not acquire lock on @Synchronized component: hotelSearch" 或类似的错误信息。

问题是 `Seam 2` 在 `stateful session bean (SFSB)` 锁外部进行锁定且使用不同的作用域。这意味着如果某个线程在相同的事务里两次访问了同一个 `EJB`，在第一次调用后，它将具有 `SFSB` 锁，而不是 `Seam` 的锁。然后另外一个线程可能获得 `Seam` 锁，这样将和 `EJB` 锁冲突并阻塞。当第一个线程开始第二次调用时，它将阻塞在 `Seam 2` 拦截器上并形成死锁。在 `Java EE 5` 里，`EJB` 遇到并发访问时将立即抛出异常。而在 `JBoss EAP 6` 里已经修改了这个行为。

解决办法是在 `EJB` 里添加 `@AccessTimeout(0)`。这会使它在发生这种情况时立即抛出 `ConcurrentAccessException`。

`Dvdstore` 例程创建订单失败并抛出 `javax.ejb.EJBTransactionRolledbackException`。

dvdstore 例程显示了下列错误：

```
JBAS011437: Found extended persistence context in SFSB invocation call
stack but that cannot be used because the transaction already has a
transactional context associated with it. This can be avoided by
changing application code, either eliminate the extended persistence
context or the transactional context. See JPA spec 2.0 section 7.6.3.1.
```

这是由于 JPA 规格的改变而引起的。

解决办法时修改 **CheckoutAction** 和 **ShowOrdersAction** 类里的持久化上下文为 **transactional** 并在 **cancelOrder** 和 **detailOrder** 方法里使用实体管理者的合并操作。

在 JBoss EAP 6 里无法使用 Seam Cache 的提供者 JBoss Cache

在 JBoss EAP 6 里不支持 JBoss Cache。这导致应用服务器上的一个 Seam 程序里使用 JBoss Cache 时抛出异常：

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

Hibernate 3.3.x 自动扫描 JBoss EAP 6 里 JPA 实体的问题。

解决办法时在 persistence.xml 文件里手动列出所有的实体类。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="example_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
    <properties>
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
    <class>com.acme.Foo</class>
    <class>com.acme.Bar</class>
  </persistence-unit>
</persistence>
```

从非 EJB 线程调用 EJB Seam 组件导致了 javax.naming.NameNotFoundException

这个问题是 JBoss EAP 里为实现新的模块化类加载系统和采用新的标准化 JNDI 命名空间规则而进行修改的结果。**java:app** 命名空间是用于单个应用程序里所有组件共享的名称。非 EE 线程，如 Quartz 异步线程，必须使用 **java:global** 命名空间，它是部署在应用服务器实例里的所有应用程序共享的。

如果您试图从 Quartz 异步方法里调用 EJB Seam 组件时遇到 **javax.naming.NameNotFoundException**，您必须修改 **components.xml** 文件来使用全局 JNDI 名称，例如：

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

For more information on JNDI changes, refer to the following topic: [第 3.1.8.1 节 “更新应用程序 JNDI 命名空间的名称”](#) . For more information on this specific issue, refer to [BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods](#) in the *2.2.0 Release Notes for Red Hat JBoss Web Framework Kit* on the Red Hat Customer Portal.

[提交 bug 报告](#)

3.2.14. 移植 Spring 应用程序

3.2.14.1. 移植 Spring 应用程序

关于移植 Spring 应用程序的信息可以在 Customer Portal <https://access.redhat.com/site/documentation/> 的 *Red Hat JBoss Web Framework Kit* 文档里找到。找到 **Red Hat JBoss Middleware**, 然后点击 *Red Hat JBoss Web Framework Kit* 链接。《*Spring Installation Guide*》和《*Spring Developer Guide*》都有多种格式的文档。

[提交 bug 报告](#)

3.2.15. 其他影响移植的修改

3.2.15.1. 熟悉其他可能影响到移植的修改

下面是 JBoss EAP 6 里可能会影响移植的修改的列表。

- [第 3.2.15.2 节 “修改 Maven 插件名称”](#)
- [第 3.2.15.3 节 “修改客户端应用程序”](#)

[提交 bug 报告](#)

3.2.15.2. 修改 Maven 插件名称

jboss-maven-plugin 还没有更新, 无法在 JBoss EAP 6 里使用。您必须使用 **org.jboss.as.plugins:jboss-as-maven-plugin** 来部署到正确的目录里。

[提交 bug 报告](#)

3.2.15.3. 修改客户端应用程序

如果您计划移植连接至 JBoss EAP 6 的客户端程序, 您应该注意捆绑了客户库的 JAR 的位置和名称。这个 JAR 现在的名称为 **jboss-client.jar** 且位于 **EAP_HOME/bin/client/** 目录。它代替了 **EAP_HOME/client/jbossall-client.jar** 并包含所有从远程客户连接 JBoss EAP 6 所需的依赖关系。

[提交 bug 报告](#)

第 4 章 工具和提示

4.1. 协助移植的资源

4.1.1. 可协助移植的资源

下面是在移植应用程序到 JBoss EAP 6 时可能会由帮助的资源列表。

工具

这是几个自动化某些配置修改的工具。详情请参考：[第 4.1.2 节“熟悉可以协助移植的工具”](#)。

调试提示

关于您可能遇到的问题和错误的常见原因和解决办法，请参考：[第 4.2.1 节“调试和解决移植问题”](#)。

例程移植

关于已经移植到 JBoss EAP 6 里的例程，请参考：[第 4.3.1 节“复查例程的移植”](#)。

[提交 bug 报告](#)

4.1.2. 熟悉可以协助移植的工具

介绍

有些工具可以协助您移植应用程序。下面是这些工具以及相关的描述。

Tattletale

由于采用模块化类加载，您需要找到并修订应用程序的依赖关系。Tattletale 可以帮助您确定依赖的模块名称并为应用程序生成配置 XML 内容。

[第 4.1.3 节“使用 Tattletale 来查找应用程序的依赖关系”](#)

IronJacamar 移植工具

在 JBoss EAP 6 里，数据源和资源适配器都不再在单独的文件里配置了。它们现在在服务器配置文件里进行配置且使用新的 schema。IronJacamar 移植工具可以帮助转换旧的配置为 JBoss EAP 6 需要的格式。

[第 4.1.6 节“使用 IronJacamar 工具来移植数据源和资源适配器配置”](#)

[提交 bug 报告](#)

4.1.3. 使用 Tattletale 来查找应用程序的依赖关系

概况

由于 JBoss EAP 6 里的模块化类加载的改动，当您移植应用程序时，您可能会在 JBoss 日志里看到 **ClassNotFoundException** 或 **ClassCastException** 跟踪信息。要解决这些错误，您需要找到包含这些异常指定的类的 JAR。

Tattletale 是一个优秀的第三方工具，它递归地扫描您的应用程序并提供其内容的详细报告。Tattletale 1.2.0.Beta2 或更新版本包含额外的支持以帮助新的 JBoss 模块化类加载。Tattletale 的 "JBoss AS 7" 报告可以用于自动确定和生成依赖模块的名称来包含应用程序的 `jboss-deployment-structure.xml`

文件。

过程 4.1. 安装并运行 Tattletale 来查找应用程序的依赖关系

1. 第 4.1.4 节 “下载和安装 Tattletale”
2. 第 4.1.5 节 “创建和复查 Tattletale 报告”



注意

Tattletale 是一个第三方的工具，它并不是 JBoss EAP 6 的一部分。关于如何安装和使用 Tattletale 的最新文档，请访问其网站：<http://tattletale.jboss.org/>。

[提交 bug 报告](#)

4.1.4. 下载和安装 Tattletale

过程 4.2. 下载和安装 Tattletale

1. <http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale> 下载 Tattletale 1.2.0.Beta2 或更新版本。
2. 解压文件到您指定的目录。
3. 修改 `TATTLETALE_HOME/jboss-tattletale.properties` 文件：
 - a. 添加 `ee6` 和 `as7` 到 `profiles` 属性。

```
profiles=java5, java6, ee6, as7
```

- b. 取消 `scan` 和 `reports` 属性的注释。

[提交 bug 报告](#)

4.1.5. 创建和复查 Tattletale 报告

1. 用下列命令创建 Tattletale 报告：`java -jar TATTLETALE_HOME/tattletale.jar APPLICATION_ARCHIVEOUTPUT_DIRECTORY`
 例如：`java -jar tattletale-1.2.0.Beta2/tattletale.jar ~/applications/jboss-seam-booking.ear ~/output-results/`
2. 在浏览器里打开 `OUTPUT_DIRECTORY/index.html` 文件并点击 "Reports" 下的 JBoss AS 7"。
 - a. 左侧的列列出了应用程序使用的归档。点击 `ARCHIVE_NAME` 链接来查看归档的细节，如位置、Manifest 信息和它包含的类。
 - b. 右侧列上的 `jboss-deployment-structure.xml` 链接显示了如何为左侧列命名的归档指定模块依赖关系。点击这个链接来查看如何为这个归档定义部署依赖关系模块信息。

[提交 bug 报告](#)

4.1.6. 使用 IronJacamar 工具来移植数据源和资源适配器配置

介绍

在以前的应用服务器版本里，数据源和资源适配器都是使用后缀为 ***-ds.xml** 的文件来配置和部署的。IronJacamar 1.1 版本包含了一个移植工具，它可以用来将这些文件转换为 JBoss EAP 6 所要求的格式。这个工具解析了之前版本的源配置文件，然后创建 XML 配置并写入新格式的输出文件。这个 XML 文件可以在 JBoss EAP 6 服务器配置文件里的正确的子系统下复制和粘贴。这个工具尽量将旧的属性和元素转换为新的格式，然而，您还是有必要对生成的文件进行额外的修改。

过程 4.3. 安装和运行 IronJacamar 移植工具

1. 第 4.1.7 节 “下载和安装 IronJacamar 移植工具”
2. 第 4.1.8 节 “使用 IronJacamar 移植工具来转换数据源配置文件”
3. 第 4.1.9 节 “使用 IronJacamar 移植工具来转换资源适配器配置文件”

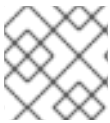


注意

IronJacamar 移植工具是一个第三方的工具，它不会以 JBoss EAP 6 的一部分而被支持。关于 IronJacamar 的更多信息，请访问 <http://www.ironjacamar.org/>。关于如何安装和使用这个工具的最新文档，请访问：<http://www.ironjacamar.org/documentation.html>。

[提交 bug 报告](#)

4.1.7. 下载和安装 IronJacamar 移植工具



注意

移植工具只能用于 IronJacamar 1.1 或更高版本且要求 Java 7 或更高版本。

1. 在这里下载最新年的 IronJacamar：<http://www.ironjacamar.org/download.html>
2. 解压下载的文件到您指定的目录。
3. 在 IronJacamar 里找到转换脚本。
 - Linux 脚本在这里：**`IRONJACAMAR_HOME/doc/as/converter.sh`**
 - Windows 批处理文件在这里：**`IRONJACAMAR_HOME/doc/as/converter.bat`**

[提交 bug 报告](#)

4.1.8. 使用 IronJacamar 移植工具来转换数据源配置文件



注意

IronJacamar 转换器脚本要求 Java 7 或更高版本。

过程 4.4. 转换数据源配置文件

1. 打开命令行并进入 **`IRONJACAMAR_HOME/doc/as/`** 目录。
2. 请输入以下命令来运行转换脚本：

- 对于 Linux : `./converter.sh -ds SOURCE_FILE TARGET_FILE`
- 对于 Microsoft Windows : `./converter.bat -ds SOURCE_FILE TARGET_FILE`

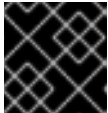
SOURCE_FILE 是以前版本的 `-ds.xml` 文件。**TARGET_FILE** 包含新的配置。

例如, 要转换位于当前目录里的 `jboss-seam-booking-ds.xml` 数据源配置文件, 您可以输入:

- 对于 Linux : `./converter.sh -ds jboss-seam-booking-ds.xml new-datasource-config.xml`
- 对于 Microsoft Windows : `./converter.bat -ds jboss-seam-booking-ds.xml new-datasource-config.xml`

请注意数据源转换的参数是 `-ds`。

3. 从目标文件复制 `<datasource>` 元素并粘贴到 `<subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources>` 元素下的服务器配置文件。



重要

要使修改在服务器重启后仍然生效, 您必须在编辑服务器配置文件前停止服务器。

- 如果服务器运行在受管域里, 请将 XML 复制到 `EAP_HOME/domain/configuration/domain.xml` 文件里。
- 如果运行的是独立服务器, 请将 XML 复制到 `EAP_HOME/standalone/configuration/standalone.xml` 文件里。

4. 修改新配置文件里生成的 XML。

下面是 JBoss EAP 5.x 里附带的 Seam 2.2 Booking 例程里的 `jboss-seam-booking-ds.xml` 数据源配置文件:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDataSource</jndi-name>
    <connection-url>jdbc:hsqldb:./</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

下面是运行转换器脚本生成的配置文件。生成的文件包含 `<driver-class>` 元素。首选的方式是定义 JBoss EAP 6 里的驱动类使用 `<driver>` 元素。下面是注释 `<driver-class>` 元素并添加对应的 `<driver>` 元素的 JBoss EAP 6 配置文件里的 XML 内容:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDataSource" jta="true"
```



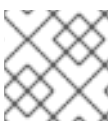
```

        pool-name="bookingDatasource" use-ccm="true" use-java-
context="true">
    <connection-url>jdbc:hsqldb:./</connection-url>
    <!-- Comment out the following driver-class element
        since it is not the preferred way to define this.
    -->
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
->
    <!-- Specify the driver, which is defined later in the
datasource -->
    <driver>h2</driver>
    <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
    <pool>
        <prefill>>false</prefill>
        <use-strict-min>>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
        <user-name>sa</user-name>
        <password/>
    </security>
    <validation>
        <validate-on-match>>false</validate-on-match>
        <background-validation>>false</background-validation>
        <use-fast-fail>>false</use-fast-fail>
    </validation>
    <timeout/>
    <statement>
        <track-statements>>false</track-statements>
    </statement>
</datasource>
<drivers>
    <!-- The following driver element was not in the
XML target file. It was created manually. -->
    <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
    </driver>
</drivers>
</datasources>
</subsystem>

```

[提交 bug 报告](#)

4.1.9. 使用 IronJacamar 移植工具来转换资源适配器配置文件



注意

IronJacamar 转换器脚本要求 Java 7 或更高版本。

1. 打开命令行并进入 **IRONJACAMAR_HOME/docs/as/** 目录。
2. 请输入以下命令来运行转换脚本：

- 对于 Linux : `./converter.sh -ra SOURCE_FILE TARGET_FILE`
- 对于 Microsoft Windows : `./converter.bat -ra SOURCE_FILE TARGET_FILE`

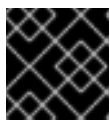
SOURCE_FILE 是以前版本的资源适配器 `-ds.xml` 文件。**TARGET_FILE** 包含新的配置。

例如, 要转换位于当前目录里的 `mttestadapter-ds.xml` 资源适配器配置文件, 您可以输入 :

- 对于 Linux: `./converter.sh -ra mttestadapter-ds.xml new-adapter-config.xml`
- 对于 Microsoft Windows : `./converter.bat -ra mttestadapter-ds.xml new-adapter-config.xml`

请注意资源适配器转换的参数是 `-ra`。

3. 从目标文件将整个 `<resource-adapters>` 元素复制并粘贴到 `<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">` 元素下的服务器配置文件里。



重要

要使修改在服务器重启后仍然生效, 您必须在编辑服务器配置文件前停止服务器。

- 如果服务器运行在受管域里, 请将 XML 复制到 `EAP_HOME/domain/configuration/domain.xml` 文件里。
- 如果运行的是独立服务器, 请将 XML 复制到 `EAP_HOME/standalone/configuration/standalone.xml` 文件里。

4. 修改新配置文件里生成的 XML。

下面是 JBoss EAP 5.x TestSuite 里的 `mttestadapter-ds.xml` 资源适配器配置文件的一个例子 :

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=====
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=====
-->
  <connection-factories>
    <tx-connection-factory>
      <jndi-name>JBossTestCF</jndi-name>
      <xa-transaction/>
      <rar-name>jbosstestadapter.rar</rar-name>
      <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
      <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
```

```

    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCFByTx</jndi-name>
    <xa-transaction/>
    <track-connection-by-tx>true</track-connection-by-tx>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
</connection-factories>

```

下面是运行转换器脚本生成的配置文件。用正确的受管连接工厂的类名替换生成的 XML 里的 class-name 属性值 "FIXME_MCF_CLASS_NAME", 在这个例子是

"org.jboss.test.jca.adapter.TestManagedConnectionFactory"。下面是已修改了 **<class-name>** 元素值的 JBoss EAP 6 配置文件的 XML 内容：

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
        <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
        <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
      jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
      use-ccm="true" use-java-context="true"> -->
        <connection-definition
          class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
          enabled="true"
          jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
          use-ccm="true" use-java-context="true">
          <config-property name="IntegerProperty">2</config-property>
          <config-property name="sleepInStart">200</config-property>
          <config-property name="sleepInStop">200</config-property>
          <config-property name="BooleanProperty">>false</config-property>
          <config-property
name="UrlProperty">http://www.jboss.org</config-property>
          <config-property name="DoubleProperty">5.5</config-property>
        </pool>
          <prefill>>false</prefill>
          <use-strict-min>>false</use-strict-min>
          <flush-strategy>FailingConnectionOnly</flush-strategy>
        </pool>
        <security>
          <application/>
        </security>
        <timeout/>
        <validation>
          <background-validation>>false</background-validation>
          <use-fast-fail>>false</use-fast-fail>
        </validation>
      </connection-definition>
    </connection-definitions>
  </resource-adapter>
  <resource-adapter>
    <archive>jbosstestadapter.rar</archive>
    <transaction-support>XATransaction</transaction-support>
    <connection-definitions>
      <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
      <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
      jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
      use-ccm="true" use-java-context="true"> -->
    </connection-definition>
    <connection-definition
      class-
```

```

name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
  enabled="true"
  jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
  use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
  <pool>
    <prefill>false</prefill>
    <use-strict-min>false</use-strict-min>
    <flush-strategy>FailingConnectionOnly</flush-strategy>
  </pool>
  <security>
    <application/>
  </security>
  <timeout/>
  <validation>
    <background-validation>false</background-validation>
    <use-fast-fail>false</use-fast-fail>
  </validation>
</connection-definition>
  </connection-definitions>
</resource-adapter>
<resource-adapter>
  <archive>jbosstestadapter.rar</archive>
  <transaction-support>XATransaction</transaction-support>
  <connection-definitions>
    <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
    <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
      jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
      use-ccm="true" use-java-context="true"> -->
    <connection-definition
      class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
      enabled="true"
      jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
      use-ccm="true" use-java-context="true">
        <config-property name="IntegerProperty">2</config-property>
        <config-property name="sleepInStart">200</config-property>
        <config-property name="sleepInStop">200</config-property>
        <config-property name="BooleanProperty">false</config-property>
        <config-property
name="UrlProperty">http://www.jboss.org</config-property>
        <config-property name="DoubleProperty">5.5</config-property>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>

```

```

    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
</connection-definitions>
</resource-adapter>
</resource-adapters>
</subsystem>

```

[提交 bug 报告](#)

4.2. 调试移植的问题

4.2.1. 调试和解决移植问题

由于应用服务器里的类加载、JNDI 命名规则以及其他修改，如果您试图就地部署应用程序，您可能会遇到异常抛出或其他错误。下面的内容描述了如何解决一些常见的异常和错误。

- [第 4.2.2 节 “调试和解决 ClassNotFoundExceptions 和 NoClassDefFoundErrors”](#)
- [第 4.2.5 节 “调试和解决 ClassCastExceptions”](#)
- [第 4.2.6 节 “调试和解决 DuplicateServiceException”](#)
- [第 4.2.7 节 “调试和解决 JBoss Seam 调试页面的错误”](#)

[提交 bug 报告](#)

4.2.2. 调试和解决 ClassNotFoundExceptions 和 NoClassDefFoundErrors

介绍

ClassNotFoundExceptions 通常是由未解决的依赖关系引起的。这意味着您必须显性地定义对其他模块的依赖关系或从外部源复制 JAR。

1. 首先，找到缺失的依赖关系。详情请参考 [第 4.2.3 节 “查找 JBoss 模块依赖关系”](#)
2. 如果缺失的类没有模块，请找到前一个版本里的 JAR 文件。详情请访问 [第 4.2.4 节 “在以前的安装里查找 JAR”](#)

[提交 bug 报告](#)

4.2.3. 查找 JBoss 模块依赖关系

要解决依赖关系，首先，通过查找 `EAP_HOME/modules/system/layers/base/` 目录找到包含 **ClassNotFoundException** 指定的类的模块。如果您找到这个类的模块，您必须在 manifest 条目里添加一个依赖关系。

例如，如果您在日志里看到 `ClassNotFoundException` 跟踪信息：

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module
Loader]
    at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

通过下列步骤来查找包含这个类的 JBoss 模块：

过程 4.5. 找到依赖关系

1. 首先确定这个类是否有明显的模块。
 - a. 进入 `EAP_HOME/modules/system/layers/base/` 目录并查找对应 `ClassNotFoundException` 里命名的类的模块路径。

找到模块路径 `org/apache/commons/logging/`。
 - b. 打开 `EAP_HOME/modules/system/layers/base/org/apache/commons/logging/main/module.xml` 文件并找到模块名称。在这个例子里是 "org.apache.commons.logging"。
 - c. 添加模块名到 `MANIFEST.MF` 文件里的依赖关系：

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

2. 如果这个类没有明显的模块路径，您可能需要在另一个位置寻找依赖关系。
 - a. 在 Tattletale 报告里找到名为 `ClassNotFoundException` 的类。
 - b. 在 `EAP_HOME/modules` 目录里找到包含 JAR 的模块以及和前一步骤里相同的模块名。

[提交 bug 报告](#)

4.2.4. 在以前的安装里查找 JAR

如果在服务器定义的模块里的 JAR 中没有找到这个类，请到 `EAP5_HOME` 版本或之前服务器的 `lib/` 目录里查找这个 JAR。

例如，如果您在日志里看到 `ClassNotFoundException` 跟踪信息：

```
Caused by: java.lang.NoClassDefFoundError:
org.hibernate.validator.ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

通过下列步骤来查找包含这个类的 JAR：

1. 打开一个终端窗口并进入 `EAP5_HOME/` 目录。
2. 执行这个命令：


```
grep 'org.hibernate.validator.ClassValidator' `find . \-name '*.jar'`
```

3. 您可能会看到多个结果。在这个例子里，下面的结果就是我们需要的 JAR：

```
Binary file ./jboss-eap-5.1/seam/lib/hibernate-validator.jar matches
```

4. 将这个 JAR 复制到 **lib/** 目录。

如果您发现您需要大量的 JAR，为这些类定义一个模块可能更加便利。关于更多的信息，请参考 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 部署指南》里的『开发企业级应用程序起步』章节中的『模块』。

5. 重建和重部署这个应用程序。

[提交 bug 报告](#)

4.2.5. 调试和解决 **ClassCastException**

ClassCastException 通常是类被不同于它扩展的类加载器加载而引起的，它也可能是因为在多个 JAR 里存在相同的类而引起的。

1. 搜索应用程序来查找包含 **ClassCastException** 里命名的类的所有 JAR。如果为这个类定义了模块，请从应用程序的 WAR 或 EAR 里查找并删除重复的 JAR。
2. 找到包含这个类的 JBoss 模块并显性地在 **MANIFEST.MF** 文件或 **jboss-deployment-structure.xml** 文件里定义依赖关系。关于更多的信息，请查看 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ 上的《JBoss EAP 6 部署指南》里『类加载和模块』章节中的『类加载和子部署』。
3. 如果您无法使用上面的步骤解决这个问题，您可以将类加载器的信息输出到日志里确定原因。例如，您可以在日志里看下列 **ClassCastException**：

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be
cast to com.example2.CustomClass2
```

- a. 在您的代码里，将 **ClassCastException** 里出现的类的类加载信息输出到日志。例如：

```
logger.info("Class loader for CustomClass1: " +
com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +
com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

- b. 日志里的信息显示哪些模块在加载类，且您需要根据您的应用程序删除或移走冲突的 JAR。

[提交 bug 报告](#)

4.2.6. 调试和解决 **DuplicateServiceException**

当您在 JBoss EAP 6 里部署 EAR 时，得到一个 JAR 子部署的 `DuplicateServiceException` 或显示 WAR 应用程序已经安装的消息，这可能是由于 JBossWS 处理部署的方式的改变而引起的。

JBossWS 3.3.0 对于基于端点的 servlet 引入了一个新的 Context Root 映射算法以允许它和 TCK6 无缝兼容。如果应用程序 EAR 归档包含具有相同名称的 WAR 和 JAR，JBossWS 可能会创建一个具有相同名字的 WAR 上下文和 web 上下文。Web 上下文和 WAR 上下文相冲突，这会导致部署错误。请用下列方法来解决部署问题：

- 将 JAR 文件重命名为与 WAR 不同的名称，这样生成的 Web 和 WAR 上下文就是唯一的。
- 在 `jboss-web.xml` 文件里提供一个 `<context-root>` 元素。
- 在 `jboss-webservices.xml` 文件里提供一个 `<context-root>` 元素。
- 在 `application.xml` 文件为 WAR 自定义 `<context-root>` 元素。

[提交 bug 报告](#)

4.2.7. 调试和解决 JBoss Seam 调试页面的错误

在您移植和成功部署应用程序后，您可能遇到一个运行时错误，并重定向到“JBoss Seam 调试页面”。这个页面的 URL 是 `"http://localhost:8080/APPLICATION_CONTEXT/debug.seam"`。这个页面允许您查看和检查与当前登陆会话关联的任何 Seam 上下文里的 Seam 组件。

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

+ Component
+ Conversation Context (None selected)
+ Business Process Context
+ Session Context
+ Application Context

图 4.1. JBoss Seam 调试页面

重定向到这个页面的最可能的原因是 Seam 已经捕获了应用程序里没有处理的异常。异常的根本原因常常可以在“JBoss Seam 调试页面”上的链接里找到。

1. 展开页面上的 **Component** 部分并查找 `org.jboss.seam.caughtException` 组件。
2. 其原因和跟踪信息应该指向缺失的依赖关系。

JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	Select Destroy

- Component

Select a component from one of the contexts below

[- Component \(org.jboss.seam.caughtException\)](#)

cause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
class	class javax.servlet.ServletException
localizedMessage	Servlet execution threw an exception
message	Servlet execution threw an exception
rootCause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
stackTrace	[org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:346), org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:248), org.jboss.seam.servlet.SeamFilter\$FilterChainImpl.doFilter(SeamFilter.java:83), org.jboss.seam.web.IdentityFilter.doFilter(IdentityFilter.java:40), [... rest of stacktrace omitted for display purposes]
toString()	javax.servlet.ServletException: Servlet execution threw an exception

+ Conversation Context (None selected)

+ Business Process Context

+ Session Context

+ Application Context

图 4.2. 组件 org.jboss.seam.caughtException 的信息

3. 使用 第 4.2.2 节 “调试和解决 ClassNotFoundExceptions 和 NoClassDefFoundErrors” 里描述的技术来解决模块依赖关系。

在上面的例子里，最简单的办法是添加 `org.slf4j` 到 `MANIFEST.MF` 里。

```
Manifest-Version: 1.0
Dependencies: org.slf4j
```

另一个选择是添加模块依赖关系到 `jboss-deployment-structure.xml` 文件：

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.slf4j" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

[提交 bug 报告](#)

4.3. 复查例程的移植

4.3.1. 复查例程的移植

介绍

下面是一个已经从 JBoss EAP 5.x 移植到 JBoss EAP 6 的例程列表。要查看特定程序的修改细节，请点击下列链接。

- [第 4.3.2 节 “移植 Seam 2.2 JPA 例程到 JBoss EAP 6”](#)
- [第 4.3.3 节 “移植 Seam 2.2 Booking 例程到 JBoss EAP 6”](#)
- [第 4.3.4 节 “移植 Seam 2.2 Booking 例程到 JBoss EAP 6 : 逐步说明”](#)

[提交 bug 报告](#)

4.3.2. 移植 Seam 2.2 JPA 例程到 JBoss EAP 6

介绍

下面的任务列表总结了移植 Seam 2.2 JPA 例程到 JBoss EAP 6 所需的修改。这个例程可以在最新的 JBoss EAP 5.1 的 [EAP5.x_HOME/jboss-eap-5.x/seam/examples/jpa/](#) 目录下找到。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 4.6. 移植 Seam 2.2 JPA 例程

1. 删除 jboss-web.xml 文件

从 **jboss-seam-jpa.war/WEB-INF/** 目录删除 **jboss-web.xml** 文件。**jboss-web.xml** 里定义的类加载现在是默认的行为。

2. 像下面这样修改 jboss-seam-jpa.jar/META-INF/persistence.xml 文件：

- 删除或注释 **jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml** 文件里的 **hibernate.cache.provider_class** 属性：

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

- 在 **jboss-seam-booking.jar/META-INF/persistence.xml** 文件里添加 Provider Module 属性：

```
<property name="jboss.as.jpa.providerModule" value="hibernate3-
bundled" />
```

- 修改 **jta-data-source** 属性以使用默认的 JDBC 数据源 JNDI 名称：

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
```

3. 添加 Seam 2.2 依赖关系

从 Seam 2.2 的 **SEAM_HOME/lib/** 目录复制下列 JAR 到 **jboss-seam-jpa.war/WEB-INF/lib/** 目录：

- antlr.jar
- slf4j-api.jar
- slf4j-log4j12.jar
- hibernate-entitymanager.jar
- hibernate-core.jar
- hibernate-annotations.jar
- hibernate-commons-annotations.jar
- hibernate-validator.jar

4. 创建一个 **jboss-deployment-structure** 文件来添加剩下的依赖关系

在 **jboss-seam-jpa.war/WEB-INF/** 目录里创建一个包含下列数据的 **jboss-deployment-structure.xml** 文件：

```
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="org.apache.log4j" />
      <module name="org.dom4j" />
      <module name="org.apache.commons.logging" />
      <module name="org.apache.commons.collections" />
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

结果：

Seam 2.2 JPA 例程在 JBoss EAP 6 上成功部署和运行。

[提交 bug 报告](#)

4.3.3. 移植 Seam 2.2 Booking 例程到 JBoss EAP 6

介绍

Seam 2.2 Booking EAR 移植比 Seam 2.2 JPA WAR 更为复杂。在 [第 4.3.2 节“移植 Seam 2.2 JPA 例程到 JBoss EAP 6”](#) 里可以找到 Seam 2.2 JPA WAR 例程移植的文档。要移植这个程序，您必须：

1. 初始化 JSF 1.2 而不是默认的 JSF 2。
2. 捆绑旧版的 Hibernate JAR 而不是使用 JBoss EAP 6 附带的版本。
3. 修改 JNDI 绑定以使用新的 Java EE 6 JNDI 可移植的语法。

上面的头两个步骤是在 Seam 2.2 JPA WAR 例程移植里完成的。第三个步骤是新的，因为 EAR 包含了 EJB，它也是必需的。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 4.7. 移植 Seam 2.2 Booking 例程

1. 创建 jboss-deployment-structure.xml 文件

在 **jboss-seam-booking.ear/META-INF/** 里创建一个名为 **jboss-deployment-structure.xml** 的文件并添加下列内容：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.log4j" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
    <exclusions>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

2. 像下面这样修改 jboss-seam-booking.jar/META-INF/persistence.xml 文件。

a. 删除或注释 cache provider 类的 hibernate 属性：

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

- b. 在 **jboss-seam-booking.jar/META-INF/persistence.xml** 文件里添加 Provider Module 属性：

```
<property name="jboss.as.jpa.providerModule" value="hibernate3-bundled" />
```

- c. 修改 **jta-data-source** 属性以使用默认 JDBC 数据源 JNDI 名称：

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

3. 从 Seam 2.2 复制 JAR 文件

从 Seam 2.2 的 **EAP5.x_HOME/jboss-eap5.x/seam/lib/** 目录复制下列 JAR 到 **jboss-seam-booking.ear/lib** 目录：

```
antlr.jar
slf4j-api.jar
slf4j-log4j12.jar
hibernate-core.jar
hibernate-entitymanager.jar
hibernate-validator.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar
```

4. 修改 JNDI 查找名称

修改 **jboss-seam-booking.war/WEB-INF/components.xml** 文件里的 JNDI 查找字符串。由于新的 JNDI 可移植规则，JBoss EAP 6 现在使用 JNDI 可移植语法规则绑定 EJB，您无法使用 JBoss EAP 5 里使用的单个的 `jndiPattern`。应用程序的 EJB JNDI 查找字符串必须修改成：

```
java:global/jboss-seam-booking/jboss-seam-booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:app/jboss-seam-booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:module/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:global/jboss-seam-booking/jboss-seam-booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction
```

Seam 2.2 框架 EJB 的 JNDI 查找字符串必须修改成：

```
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:app/jboss-seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
```

```

java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations

```

您可以使用下列方式中的一个：

a. 添加组件元素

您可以为每个 EJB 添加一个 **jndi-name** 至 **WEB-INF/components.xml**：

```

<component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
<component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
<component
class="org.jboss.seam.example.booking.BookingListAction" jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
<component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
<component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
<component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />
<component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />

```

- b. 您可以修改代码，添加 **@JNDIName(value="")** 注解指定 JNDI 路径。下面是一个已修改的 stateless session bean 代码示例。关于这个产品的详细说明可以在 Seam 2.2 的参考文档里找到。

```

@Stateless
@Name("authenticator")
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
{
    ...
}

```

结果：

Seam 2.2 Booking 例程在 JBoss EAP 6 上成功部署和运行。

[提交 bug 报告](#)

4.3.4. 移植 Seam 2.2 Booking 例程到 JBoss EAP 6 : 逐步说明

这是一个将 Seam 2.2 Booking 从 JBoss EAP 5.X 移植到 JBoss EAP 6 的逐步指南。虽然移植应用程序有更好的方法，许多开发人员可能会冒险直接将归档部署到 JBoss EAP 6，并看看会有什么发生。这个文档的目的是展示您可能遇到的各种问题，并如何调试和解决这些问题。

对于这个例子，应用程序 EAR 部署在 **EAP6_HOME/standalone/deployments** 目录，对现有归档无需进行修改。这允许您在遇到问题时可以轻易地修改归档里的 XML 文件从而解决问题。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 4.8. 移植应用程序

1. 第 4.3.5 节 “构建和部署 Seam 2.2 Booking 例程的 JBoss EAP 5.X 版本。”
2. 第 4.3.6 节 “调试和解决 Seam 2.2 Booking 例程的部署错误和异常”
3. 第 4.3.7 节 “调试和解决 Seam 2.2 Booking 例程的运行时错误和异常”

此时您应该可以通过 <http://localhost:8080/seam-booking/> 成功访问应用程序了。用 demo/demo 登陆您就会看到 Booking 的欢迎页面。

复查修改的总结

第 4.3.8 节 “当移植 Seam 2.2 Booking 例程时所作修改的总结”

[提交 bug 报告](#)

4.3.5. 构建和部署 Seam 2.2 Booking 例程的 JBoss EAP 5.X 版本。

在移植这个应用程序前，您应该构建 JBoss EAP 5.X 下的 Seam 2.2 Booking 例程，展开归档文件，并复制到 JBoss EAP 6 的部署目录下。

过程 4.9. 构建和部署 EAR :

1. 构建 EAR :

```
$ cd /EAP5_HOME/jboss-eap5.x/seam/examples/booking
$ ANT_HOME/ant explode
```

用实际移植的 JBoss EAP 版本替换 *jboss-eap5.x*。

2. 复制 EAR 到 EAP6_HOME 部署目录 :

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. 启动 JBoss EAP 6 服务器并检查日志。您会看到：

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. 创建一个名为 **jboss-seam-booking.ear.dodeploy** 的空文件并将其复制到 **EAP6_HOME/standalone/deployments** 目录。在移植应用程序时，您需要多次将这个文件复制到部署目录，所以请将它放在一个容易找到的目录里。在日志里，您现在应该看到下列信息，表示它正在进行部署：

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

此时，您会遇到第一个部署错误。在下一步骤里，您将遇到每个问题并学习如何进行调试和解决。

要学习如何调试和解决部署问题，点击这里：[第 4.3.6 节“调试和解决 Seam 2.2 Booking 例程的部署错误和异常”](#)

返回上一个主题，请点击：[第 4.3.4 节“移植 Seam 2.2 Booking 例程到 JBoss EAP 6：逐步说明”](#)

[提交 bug 报告](#)

4.3.6. 调试和解决 Seam 2.2 Booking 例程的部署错误和异常

在上面的步骤 [第 4.3.5 节“构建和部署 Seam 2.2 Booking 例程的 JBoss EAP 5.X 版本。”](#) 里，您构建了 JBoss EAP 5.X 的 Seam 2.2 Booking 例程且将其部署到了 JBoss EAP 6 的部署目录里。在这个步骤，您可以调试和解决遇到的每个部署错误。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 4.10. 调试和解决部署错误和异常

1. 问题 - java.lang.ClassNotFoundException: javax.faces.FacesException

当您部署应用程序时，日志会包含下列错误：

```
ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
MSC00001: Failed to start service jboss.deployment.subunit."jboss-
seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
org.jboss.msc.service.StartException in service
```

```
jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
Failed to process phase POST_MODULE of subdeployment "jboss-seam-booking.war" of deployment "jboss-seam-booking.ear"
(.. additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
javax.faces.FacesException from \[Module "deployment.jboss-seam-booking.ear:main" from Service Module Loader\]
    at
    org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:191)
```

它表示：

`ClassNotFoundException` 表示缺失的依赖关系。在这个例子里是无法找到 `javax.faces.FacesException`，您需要显性地添加这个依赖关系。

如何解决这个问题：

在 `EAP6_HOME/modules/system/layers/base/` 目录里通过查找匹配缺失的类的路径找到这个类的模块名。在这个例子里，您会找到两个匹配的模块：

```
javax/faces/api/main
javax/faces/api/1.2
```

这两个模块都有相同的模块名：`javax.faces.api`，但位于 `main` 目录里的是用于 JSF 2.0 的，而位于 `1.2` 目录里的另外一个则是用于 JSF 1.2 的。如果只有一个可用的模块，您可以简单地创建一个 `MANIFEST.MF` 并添加模块依赖关系。在这个例子里，您像使用 JSF 1.2 版本而不是 `main` 里的 JSF 2.0，所以您需要指定一个并排斥另外一个。为此，您可以在 EAR 的 `META-INF/` 目录里创建一个包含下列数据的 `jboss-deployment-structure.xml`：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

在 `deployment` 部分，您可以添加用于 JSF 1.2 模块的 `javax.faces.api` 的依赖关系。您也可以在 WAR 的 `subdeployment` 部分添加用于 JSF 1.2 模块的依赖关系并排除用于 JSF 2.0 的模块。

通过删除 `EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed` 文件并在相同目录里创建一个空白的 `jboss-seam-booking.ear.dodeploy` 文件来重新部署应用程序。

2. 问题 - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

当您部署应用程序时，日志会包含下列错误：

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]
```

它表示：

ClassNotFoundException 指示了一个缺失的依赖关系。在这个例子里是无法找到 **org.apache.commons.logging.Log** 且您需要显性地添加依赖关系。

如何解决这个问题：

通过查找匹配缺失类的路径在 **EAP6_HOME/modules/system/layers/base/** 里找到这个类的模块名。在这个例子里，您要找到匹配路径 **org/apache/commons/logging/** 的模块。这个模块名是 "org.apache.commons.logging"。

修改 **jboss-deployment-structure.xml** 文件，添加模块依赖关系至部署部分。

```
<module name="org.apache.commons.logging" export="true"/>
```

jboss-deployment-structure.xml 应该类似于：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

3. 问题 - java.lang.ClassNotFoundException: org.dom4j.DocumentException

当您部署应用程序时，日志会包含下列错误：

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-host].[/seam-booking]] (MSC service thread 1-3) Exception sending context initialized event to listener instance of class org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError: org/dom4j/DocumentException
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException: org.dom4j.DocumentException from [Module "deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

它表示：

ClassNotFoundException 表示缺失的依赖关系。在这个例子里是无法找到 **org.dom4j.DocumentException** 类。

如何解决这个问题：

在 **EAP6_HOME/modules/system/layers/base/** 目录里通过查找 **org/dom4j/DocumentException** 来查找模块名。这个模块名是“org.dom4j”。修改 **jboss-deployment-structure.xml** 文件，添加模块依赖关系至文件的部署部分。

```
<module name="org.dom4j" export="true"/>
```

jboss-deployment-structure.xml 文件应该类似于：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

4. 问题 - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

当您部署应用程序时，日志会包含下列错误：

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-host].[/seam-booking]] (MSC service thread 1-6) Exception sending context initialized event to listener instance of class org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException: Could not create Component: org.jboss.seam.international.statusMessages (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue from [Module "deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

它表示：

ClassNotFoundException 表示缺失的依赖关系。在这个例子里是无法找到 **org.hibernate.validator.InvalidValue** 类。

如何解决这个问题：

存在名为 “org.hibernate.validator” 的模块，但 JAR 不包含 **org.hibernate.validator.InvalidValue** 类，所以添加模块依赖关系不会解决这个问题。在这个例子里，JAR 包含这个类，将其作为 JBoss EAP 5.X 部署的一部分。在 **EAP5_HOME/seam/lib/** 目录里查找包含缺失类的 JAR。为此，打开控制台并输入：

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

其结果显示：

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

在这个例子里，复制 **hibernate-validator.jar** 到 **jboss-seam-booking.ear/lib/** 目录：

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-booking.ear/lib
```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

5. 问题 - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

当您部署应用程序时，日志会包含下列错误：

```
INFO [javax.enterprise.resource.webcontainer.jsf.config] (MSC service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory 'javax.faces.application.ApplicationFactory' was not configured properly.
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactoryExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
```

```

jbossorg-4.jar:2.0.4-b09-jbossorg-4]
(... additional logs removed ...)
Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
    at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
(... additional logs removed ...)
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactoryExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-jbossorg-4.jar:2.0.4-b09-jbossorg-4]
... 11 more
Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
    at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
    at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
    at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
... 16 more

```

它表示：

com.sun.faces.config.ConfigurationException 和 **java.lang.InstantiationException** 表示存在依赖关系的问题。在这个例子里，原因并不明显。

如何解决这个问题：

您需要找到包含 **com.sun.faces** 类的模块。虽然这里没有 **com.sun.faces** 模块，但却有两个 **com.sun.jsf-impl** 模块。在 1.2 目录里快速检查 **jsf-impl-1.2_13.jar**，您会发现它包含 **com.sun.faces** 类。如您对 **javax.faces.FacesExceptionClassNotFoundException** 所做的一样，您希望在 main 里使用 JSF 1.2 版本而不是 JSF 2.0，所以您需要指定一个且排除另外一个。您需要修改 **jboss-deployment-structure.xml** 以添加模块依赖关系到这个文件的 deployment 部分。您也需要添加它到 WAR 子部署并排除 JSF 2.0 模块。这个文件现在应该类似于：

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```



```

        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

6. 问题 - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

当您部署应用程序时，日志会包含下列错误：

```

ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]

```

它表示：

ClassNotFoundException 表示缺失的依赖关系。在这个例子里是无法找到 **org.apache.commons.collections.ArrayStack**。

如何解决这个问题：

在 **EAP6_HOME/modules/system/layers/base/** 目录里通过查找 **org/apache/commons/collections** 路径来查找模块名。这个模块名是“org.apache.commons.collections”。修改 **jboss-deployment-structure.xml** 文件，添加模块依赖关系至文件的部署部分。

```

<module name="org.apache.commons.collections" export="true"/>

```

jboss-deployment-structure.xml 文件应该类似于：

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>

```

```

        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

7. 问题 - Services with missing/unavailable dependencies

当您部署应用程序时，日志会包含下列错误：

```

ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]\",\"jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]\",\"
(... additional logs removed ...)
\"jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]\",\"jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}

```

它表示：

当您遇到“Services with missing/unavailable dependencies”错误，请查看“missing”后面括号里的文本。在这里例子是：

```

missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]

```

“/em”表示这是一个 Entity Manager 和数据源的问题。

如何解决这个问题：

在 JBoss EAP 6 里，数据源配置已经进行了修改，它需要在

EAP6_HOME/standalone/configuration/standalone.xml 文件里进行定义。因为 JBoss EAP 6 附带的示例数据库已经在 **standalone.xml** 文件里进行了定义，请修改 **persistence.xml** 文件来使用这个示例数据库。请看 **standalone.xml** 文件，您可以看到示例数据库的 **jndi-name** 是 **java:jboss/datasources/ExampleDS**。修改 **jboss-seam-booking.jar/META-INF/persistence.xml** 文件，注释现有的 **jta-data-source** 元素并用下列内容来替代：

```
<!-- <jta-data-source>java:/bookingDataSource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

通过删除 **EAP6_HOME/standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

8. 此时，应用程序应该已部署无误了，但当您访问 <http://localhost:8080/seam-booking/> 并尝试 "Account Login" 时，您会遇到一个运行时错误 "The page isn't redirecting properly"。在下一步里，您会学习如何调试和解决运行时错误。

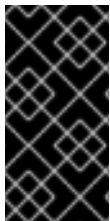
要学习如何调试和解决运行时的问题，点击这里：[第 4.3.7 节 “调试和解决 Seam 2.2 Booking 例程的运行时错误和异常”](#)

返回上一个主题，请点击：[第 4.3.4 节 “移植 Seam 2.2 Booking 例程到 JBoss EAP 6：逐步说明”](#)

[提交 bug 报告](#)

4.3.7. 调试和解决 Seam 2.2 Booking 例程的运行时错误和异常

在前面的步骤 [第 4.3.6 节 “调试和解决 Seam 2.2 Booking 例程的部署错误和异常”](#) 里，您学习了如何调试部署错误，在这个步骤里，您会调试和解决您遇到的运行时错误。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

过程 4.11. 调试和解决运行时错误和异常

目前，当您部署这个应用程序时，您没有在日志里看到任何错误。但是，当您访问这个程序的 URL 是，日志里就出现了错误。

1. 问题 - javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context "

当您通过浏览器访问 URL <http://localhost:8080/seam-booking/> 时，您会看到 "The page isn't redirecting properly" 且日志会包含下列错误：

```
SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-1) swallowing exception: java.lang.IllegalStateException: Could not start transaction
    at
    org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:598) [jboss-seam.jar:]
```

```
(... log messages removed ...)
Caused by: org.jboss.seam.InstantiationException: Could not
instantiate Seam component:
org.jboss.seam.transaction.synchronizations
    at org.jboss.seam.Component.newInstance(Component.java:2170)
[jboss-seam.jar:]
(... log messages removed ...)
Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-
booking' not found in context ''
    at
org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUtil
s.java:109)
(... log messages removed ...)
```

它表示：

NameNotFoundException 表示了 JNDI 命名问题。JNDI 命名规则在 JBoss EAP 6 里已经进行了修改，所以您需要修改查找名称以遵循新的规则。

如何解决这个问题：

为了调集试，

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUn
itProcessor] (MSC service thread 1-1) JNDI bindings for session bean
named RegisterAction in deployment unit subdeployment "jboss-seam-
booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:app/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:module/RegisterAction!org.jboss.seam.example.booking.Register
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction
    java:app/jboss-seam-booking.jar/RegisterAction
    java:module/RegisterAction
[JNDI bindings continue ...]
```

日志里一共列出了 8 个 INFO JNDI 绑定，每个都对应一个会话 bean：RegisterAction、BookingListAction、HotelBookingAction、AuthenticatorAction、ChangePasswordAction、HotelSearchingAction、EjbSynchronizations 和 TimerServiceDispatcher。您需要修改 WAR 的 **lib/components.xml** 文件来使用新的 JNDI 绑定。在日志里，注意 EJB JNDI 绑定都以 "java:app/jboss-seam-booking.jar" 开始。请替换 **core:init** 元素：

```
<!--      <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

然后，您需要添加 EjbSynchronizations 和 TimerServiceDispatcher JNDI 绑定。请添加下列 component 元素到这个文件里：

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
```

```
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

components.xml 文件应该类似于：

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.2.xsd
    http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
    http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
    http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

  <!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
  <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
  <core:manager conversation-timeout="120000"
    concurrent-request-timeout="500"
    conversation-id-parameter="cid"/>
  <transaction:ejb-transaction/>
  <security:identity authenticate-method="#
{authenticator.authenticate}"/>
  <component
class="org.jboss.seam.transaction.EjbSynchronizations"
    jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
  <component class="org.jboss.seam.async.TimerServiceDispatcher"
    jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```

通过删除 **standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

2. 问题 - 应用程序部署和运行无误。当您访问 <http://localhost:8080/seam-booking/> 并试图登陆时失败，错误消息为 "Login failed. Transaction failed."。您应该在服务器日志里看到如下异常跟踪信息：

```
13:36:04,631 WARN [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
```

```

....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
  at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...

```

它表示：

ClassNotFoundException 表示缺失的 Hibernate 库。在这个例子里是 **hibernate-core.jar**。

如何解决这个问题：

从 **EAP5_HOME/seam/lib/** 目录复制 **hibernate-core.jar** JAR 到 **jboss-seam-booking.ear/lib** 目录。

通过删除 **standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

3. 问题 - 应用程序部署和运行无误。当您访问 <http://localhost:8080/seam-booking/>，您可以成功登录。然而，当您试图预订酒店时，您会看到如下异常跟踪信息：

为了进行调试，您必须首先删除 **jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar**，因为它标记了真正的错误。此时，您应该看到如下错误：

```

java.lang.NoClassDefFoundError:
org/hibernate/annotations/common/reflection/ReflectionManager

```

它表示：

ClassNotFoundException 表示有缺失的 Hibernate 库。

如何解决这个问题：

从 **EAP5_HOME/seam/lib/** 目录复制 **hibernate-annotations.jar** 和 **hibernate-commons-annotations.jar** 到 **jboss-seam-booking.ear/lib** 目录。

通过删除 **standalone/deployments/jboss-seam-booking.ear.failed** 文件并在相同目录里创建一个空白的 **jboss-seam-booking.ear.dodeploy** 文件来重新部署应用程序。

4. 运行时和应用程序错误应该被解决

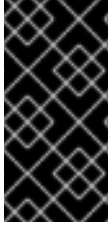
此时，应用程序已部署且运行无误。

返回上一个主题，请点击：[第 4.3.4 节“移植 Seam 2.2 Booking 例程到 JBoss EAP 6：逐步说明”](#)

[提交 bug 报告](#)

4.3.8. 当移植 Seam 2.2 Booking 例程时所作修改的总结

尽管提前决定依赖关系并在一个步骤里添加隐性的依赖关系具有更高的效率，这种方式还是展示了问题是如何出现在日志里并提供如何调试和解决的信息。下面是在移植到 JBoss EAP 6 时所作修改的总结。



重要

直接使用 Hibernate 的 Seam 2.2 应用程序可以使用包裹在应用程序里的一个 Hibernate 3 版本。而通过 JBoss EAP 6 的 org.hibernate 模块提供的 Hibernate 4，不被 Seam 2.2 支持。这个例子将帮助您在 JBoss EAP 6 运行应用程序。请注意，将 Hibernate 3 包裹在 Seam 2.2 应用程序不是被支持的配置。

1. 您在 EAR 的 **META-INF/** 目录里创建了一个 **jboss-deployment-structure.xml** 文件。您添加了 **<dependencies>** 和 **<exclusions>** 来解析 **ClassNotFoundExceptions**。这个文件包含下列数据：

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

2. 您可以将下列 JAR 从 **EAP5_HOME/jboss-eap-5.X/seam/lib/** 目录（用实际移植的 EAP 5 版本替换 5.X）复制到 **jboss-seam-booking.ear/lib/** 目录以解决 **ClassNotFoundExceptions** 问题：
 - o hibernate-core.jar
 - o hibernate-validator.jar
3. 您修改了 **jboss-seam-booking.jar/META-INF/persistence.xml** 文件。
 1. 您修改了 **jta-data-source** 元素以使用 JBoss EAP 6 附带的示例数据库。

■


```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -  
->  
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-  
source>
```

2. 您注释了 `hibernate.cache.provider_class` 属性：

```
<!-- <property name="hibernate.cache.provider_class"  
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

4. 您修改了 WAR 的 `lib/components.xml` 文件来使用新的 JNDI 绑定。

1. 您替换了现有的 `core:init` 元素：

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#  
{ejbName}/local" debug="true" distributable="false"/> -->  
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#  
{ejbName}" debug="true" distributable="false"/>
```

2. 您为 "EjbSynchronizations" 和 "TimerServiceDispatcher" JNDI 绑定添加了 component 元素：

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"  
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>  
  <component class="org.jboss.seam.async.TimerServiceDispatcher"  
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

[提交 bug 报告](#)

附录 A. 修订记录

修订 6.4.0-11

Tuesday April 14 2015

Lucas Costi

Red Hat JBoss 企业级应用程序平台 6.4.0.GA