起步指南

4.3

with JBoss Enterprise Application Platform

ISBN: N/A
出版日期: Sep, 2007

《起步指南》提供了 JBoss 企业级应用程序平台的后-安装信息。使用这个指南可以熟悉这个平台以及演示应用程序开发和部署的示例程序。

起步指南：with JBoss Enterprise Application Platform

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive Raleigh, NC 27606-2072 USA Phone: +1 919 754 3700 Phone: 888 733 4281 Fax: +1 919 754 3701 PO Box 13588 Resea

关于本书

本书的目标是概述 JBoss 企业级应用程序平台，以及演示它的一些特征和为企业级 Java 应用程序提供快速开发和部署环境的能力。在本书写作过程中，JBoss 最新的版本是 4.3。对于示例程序，你应该使用这个版本或更高的版本。本书使用的例程说明了 JBoss 企业级应用程序平台里的企业级 Java 应用程序的部署和配置。虽然本书的目的不是为了教你企业级的 Java 开发，但我们也将涵盖这个主题的一些基本内容，所以如果你是企业级 Java 应用程序开发的新手，本书也有帮助。

我们将引导你快速了解 JBoss 的目录结构、基本的服务器配置、关键的配置文件以及 JMX 和 Web 控制台。当我们进入示例程序章节时，你将看到运行的 JBoss 企业级应用程序平台并进行一些简单的配置和开发。我们将介绍 Seam 框架并演示它使应用程序开发变得有多么的不同。

当然，本书只涉及 JBoss 企业级应用程序平台的一些基本功能。一旦你熟悉了这里的内容，《JBoss 企业级应用程序平台：服务器配置指南》可以带你踏上精通 JBoss 企业级应用程序平台的征途。

介绍

JBoss 企业级应用程序平台易于安装，你只需要几个步骤就可以让它运行。请参考《JBoss 企业级应用程序平台：安装指南》里关于安装预备条件和安装步骤的内容。

一旦你安装了 JBoss 企业级应用程序平台，你就可以使用本指南来熟悉它的格式和演示应用程序开发和部署的示例程序。

1．反馈

如果你在《JBoss 企业级应用程序平台：起步指南》来发现了印刷错误，或者你有改进该手册的建议，我们希望听到你的声音！请提交报告到 Bugzilla [http://bugzilla.redhat.com/bugzilla/] 并指明：产品：JBoss 企业级应用程序平台，版本：<version>，组件：Getting_Started_Guide。如果是改进本文档的建议，请尽量具体化；如果是其他错误，请指出章节号以及具体的内容，这样我们就可以尽快改正。

2．其他手册

如果你在寻找详细的产品信息，请参考 http://www.redhat.com/docs/manuals/jboss 上的在线手册。

JBoss 服务器 - 快速指南

让我们探讨 JBoss 企业级应用程序平台的目录结构并帮助你理解如何进行安装。熟悉目录的结构以及关键配置文件、日志文件、部署文件等的位置是很有用的。这将帮助你理解 JBoss 服务架构，在部署自己的应用程序时能得心应手。

1. 目录结构

1.1. JBoss 顶层目录结构

如果你使用 zip 安装方法，安装 JBoss 企业级应用程序平台时会创建一个名为 jboss-eap-<version> 的顶层目录；如果你使用图形界面的安装方法，你就可以根据需要来命名。在这本指南里我们都将这个目录称为 JBOSS_DIST。在它下面有 4 个子目录：

doc：包含产品的文档。

jboss-as：包含存放服务器启动脚本、JAR 文件、服务器配置集和工作目录的子目录。你需要知道了解整个发布版本的结构来定位编译代码、更新配置和部署代码等的 JAR 文件。

seam: ：包含用于 Hibernate 和 JBoss Seam 框架的文件。

Uninstaller：包含卸载程序 uninstaller.jar。

下面是 JBoss 企业级应用程序平台 的安装目录的结构。default 服务器配置文件集如图示地展开。在全新的安装里，server/default 目录里只存在 conf、deploy 和 lib 目录。data、log、tmp 和 work 子目录由 JBoss 创建且要在服务器运行至少一次后才会存在。第 3 节 “启动和停止服务器” 将告诉你如何运行服务器。

```
jboss-eap-<version>                  // jboss.home_url
        |+ doc
        |+ jboss-as
                |+ bin
                |+ client
                |+ docs
                |+ icons
                |+ lib                          // jboss.lib.url
                |+ scripts
                |+ server
                        |+ all                  // jboss.server.name
                        |+ default              // jboss.server.home.url
                                |+ conf         // jboss.server.config.url
                                |+ deploy
                                |+ lib          // jboss.server.lib.url
                                |+ data
                                |+ log
                                |+ tmp
```

```
                        |+ work
                |+ minimal
                |+ production
        |+ seam
        |+ Uninstaller                    // jboss.uninstaller.url
```

几个位置可能会被覆盖。对于这些位置，图例里显示了接口常量 org.jboss.system.server.ServerConfig 和其对应的系统属性串。以 URL 结尾的名字对应可以用 RUL 远程访问的位置，例如，HTTP URL代表 Web 服务器。

1.2. JBOSS_DIST/jboss-as 目录结构

下表说明了 jboss-as 目录的内容。

目录	描述
bin	包含启动、关闭和其他系统相关的脚本。基本上 JBoss 发行版本所包括的所有的入口 JAR 文件和启动脚本都位于 bin 目录里。
client	存放可以被 Java 客户端程序（在 JBoss 外运行）或外部 web 容器使用的配置文件和 JAR 文件。你可以按要求选择归档文件或者使用 jbossall-client.jar。
docs	包含 JBoss 用来作为引用的 XML DTD（这也是 JBoss 特定配置的有用的文档）。这里也有设置不同数据库（如 MySQL、Oracle 和 Postgres）的数据源的示例 JCA（Java Connector Architecture）配置文件。
lib	包含 JBoss 使用的启动 JAR 文件。请不要把你自己的 JAR 文件放在这个目录里。
server	包含 JBoss 服务配置集。这里的每个子目录都是一个不同的服务器配置。JBoss 带有 minimal、default、production 和 all 配置集。default 配置集包含的子目录和关键的配置文件将在随后的章节里进行详细讨论。

表 2.1. JBOSS_DIST/jboss-as 目录里的内容

2. 服务器配置

基本上，JBoss 体系结构由 JMX MBean 服务器、微内核和一系列的可插拔的组件服务（MBean）组成。这易于组装不同的配置并让你可以灵活地进行裁减来满足需要。

你不需要一直运行一个大型的、庞大的服务器；你可以删除你不需要的组件（这也会大大缩短服务器的启动时间），你也可以通过编写自己的 MBean 将其他的服务集成到 JBoss 里。当然，如果只是运行标准的 J2EE 应用程序，你不需要作这些事情。所需的一切都已经在这里了。

要使用 JBoss，你并不需要对 JMX 有详细研究，但因为它是 JBoss 的中心工作方式，所以值得你对其基本架构有大致的了解。

JBoss 企业级应用程序平台带有 4 个不同的配置。在 JBOSS_DIST/jboss-as/server 目录下，你将发现 4 个子目录：minimal、default、production 和 all - 每个都对应一个服务器配置。它们提供不同的服务集合。启动服务器时你如果没有进行指定，production 配置将是缺省的配置。

minimal
: 具有最小的配置 - 启动 JBoss 所需的最少服务。此配置仅仅启动日志服务、JNDI 服务器和用来发现新部署的 URL 部署扫描器。如果你想用 JMX/JBoss 来启动自己的服务而不使用任何 J2EE 技术的话，此配置就是合适的选择。使用本配置的服务器仅仅是一个“裸”服务器，没有 web 容器，不支持 EJB 或 JMS。它不是兼容 J2EE 1.4 的配置。

default
: 这是包含缺省服务集的基本的 J2EE 1.4 服务器配置。它有部署 J2EE 应用程序所需的最常用的服务。但它不包括 JAXR 服务、IIOP 服务或任何群集服务。请注意虽然这个配置被称为“default”，但服务器的实际缺省配置是“production”配置。

all
: 另一方面，它配置了所有的服务来启动每个单独的组件。这是完整的 J2EE 1.4 服务器配置，并带有企业级扩展如群集和 RMI/IIOP。

production
: 它基于 "all" 配置，但根据产品进行了调整 - 减少了冗余日志、每 60 秒进行部署扫描，它也调整了内存使用来符合产品部署的需要。如果没有指定其他配置，这是服务器启动时使用的配置。

如果你想知道在这些实例里配置了哪些服务，你可以查看 JBOSS_DIST/jboss-as/server/<instance-name>/conf/ 目录下的 jboss-service.xml 文件以及 JBOSS_DIST/jboss-as/server/<instance-name>/deploy 目录里的配置文件。

```
[vsr]$ls server/default/conf
jboss-jta-properties.xml  jndi.properties   standardjbosscmp-jdbc.xml
jboss-log4j.xml           login-config.xml  standardjboss.xml
jboss-minimal.xml         props             xmdesc
jboss-service.xml         standardjaws.xml
```

注意

启动服务器时如果不进行指定，production 配置将是缺省的配置。

要用其他配置启动服务器，请参考 第 3.2 节 “用其他配置启动服务器” 。

2.1. 服务器配置目录结构

你正使用的配置的目录，实际上就是 JBoss 运行时的服务器根目录。它包含特定服务器配置所提供的服务的所有代码和配置信息。它既是日志输出、也是你部署应用程序的地方。表 2.2 “服务器配置目录结构” 展示了服务器配置目录（JBOSS_DIST/jboss-as/server/<instance-name>）里的目录结构及其功能。

目录	描述
conf	conf 目录包含 jboss-service.xml 给定服务器配置的引导描述符。它定义了服务器整个运行期间的核心服务。
data	data 目录供需要在文件系统里存放内容的服务使用。它存放在服务器重启后依然存在的持久性数据。有几个 JBoss 服务，如内嵌的 Hypersonic 数据库实例，都把数据存放在这个目录下。
deploy	deploy 目录包含可热部署的服务（亦即可以向正运行的服务器添加或删除的服务）。它也包含用于当前服务器配置的应用程序。你可以通过把应用程序软件包（JAR、WAR 和 EAR 文件）置于 deploy 目录来部署应用程序。这个目录被不断地扫描，任何被修改的组件都将被自动重新部署。你可以通过 URLDeploymentScanner URLs 属性覆盖它。
lib	这个目录包含服务器配置所需要的 JAR 文件（不应该被热部署的 Java 库）。你可以为 JDBC 驱动添加对应的库文件。这个目录下所有的 JAR 文件都会在启动时载入共享的 classpath。
log	这是日志文件写入的地方。JBoss 使用 Jakarta log4j 软件包进行日志记录，你也可以直接在自己的应用程序里使用它。它可以用 conf/log4j.xml 配置文件进行覆盖。
tmp	tmp 目录被 JBoss 服务用来存储临时文件。例如，部署者会把应用程序的归档文件在这里解压。
work	这个目录被 Tomcat 用来编译 JSP。

表 2.2. 服务器配置目录结构

2.2. "default" 服务器配置文件集

"default" 服务器配置文件集位于 JBOSS_DIST/jboss-as/server/default 目录。让我们看看

default 服务器配置文件集的内容：

```
jboss-eap-<version>                  // jboss.home_url
        |+ doc
        |+ jboss-as
                |+ bin
                |+ client
                |+ docs
                |+ icons
                |+ lib                          // jboss.lib.url
                |+ scripts
                |+ server
                        |+ all                  // jboss.server.name
                        |+ default              // jboss.server.home.url
                                |+ conf         // jboss.server.config.url
                                        |+ props
                                        |+ xmdesc
                                        -  jboss-jta-properties.xml
                                        -  jboss-minimal.xml
                                        -  jndi.properties
                                        -  standardjboss.xml
                                        -  jboss-log4j.xml
                                        -  jboss-service.xml
                                        -  login-config.xml
                                        -  standardjbosscmp-jdbc.xml
                                |+ deploy
                                        |+ ejb3.deployer
                                        |+ http-invoker.sar
                                        |+ jboss-aop-jdk50.deployer
                                        |+ jboss-bean.deployer
                                        |+ jboss-web.deployer
                                        |+ jbossws.sar
                                        |+ jboss-messaging.sar
                                        |+ jmx-console.war
                                        |+ management
                                        |+ uuid-key-generator.sar
                                        -  bsh-deployer.xml
                                        -  cache-invalidation-service.xml
                                        -  client-deployer-service.xml
                                        -  ear-deployer.xml
                                        -  ejb3-interceptors-aop.xml
                                        -  ejb-deployer.xml
                                        -  hsqldb-ds.xml
                                        -  jboss-ha-local-jdbc.rar
                                        -  jboss-ha-xa-jdbc.rar
                                        -  jbossjca-service.xml
                                        -  jboss-local-jdbc.rar
                                        -  jboss-xa-jdbc.rar
                                        -  jms-ra.rar
                                        -  jmx-invoker-service.xml
```

```
                                    -  jsr88-service.xml
                                    -  mail-service.xml
                                    -  monitoring-service.xml
                                    -  properties-service.xml
                                    -  quartz-ra.rar
                                    -  schedule-manager-service.xml
                                    -  scheduler-service.xml
                                    -  sqlexception-service.xml
                            |+ lib                  // jboss.server.lib.url
                    |+ minimal
                    |+ production
        |+ seam
        |+ Uninstaller                          // jboss.uninstaller.url
```

2.2.1. "conf" 目录的内容

下表解释了 conf 目录里的文件。

文件	描述
jboss-minimal.xml	这是 jboss-service.xml 配置文件的最简单夫的示例（它是在 minimal 配置文件集里使用的 jboss-service.xml）。
jboss-service.xml	jboss-service.xml 定义了核心的服务及其配置。
jndi.properties	jndi.properties 文件指定了当 InitialContext 使用无参数的构造函数创建时，在 JBoss 服务器里使用的 JNDI InitialContext 属性。
jboss-log4j.xml	这个文件配置了 Apache log4j 框架类别优先级和 JBoss 服务器代码使用的 appender。
login-config.xml	这个文件包含了当使用基于 JAAS 的安全性时适用的服务器端验证配置的样本。
props/*	props 目录包含了用于 jmx-console 的用户和角色属性文件。
standardjaws.xml	此文件提供了旧的 EJB 1.1 CMP 引擎的缺省配置。
standardjboss.xml	此文件提供了缺省的容器配置。
standardjbosscmp-jdbc.xml	此文件为 JBoss CMP 引擎提供了缺省的配置文件。
xmdesc/*-mbean.xml	xmdesc 目录包含了 jboss-service.xml 文件里配置的服务的 XMBean 描述符。

表 2.3. "conf" 目录的内容

2.2.2. "deploy" 目录的内容

下表解释了 deploy 目录里的文件。

文件	描述
bsh-deployer.xml	此文件配置了可将 bean shell 脚本部署为 JBoss 服务的部署者。
cache-invalidation-service.xml	这个服务允许通过 JMS 通知的 EJB 缓存的自定义 invalidation。它缺省是禁用的。
client-deployer-service.xml	此服务提供对 J2EE 应用程序客户端的支持。它为基于 application-client.xml 描述符的客户端应用程序管理 java:comp/env 企业命名上下文。
ear-deployer.xml	EAR 部署者是负责部署 J2EE EAR 文件的服务。
ejb-deployer.xml	EJB 部署者是负责部署 J2EE EJB JAR 文件的服务。
hsqldb-ds.xml	hsqldb-ds.xml 配置 Hypersonic 嵌入式数据库服务。它设置嵌入的数据库和相关的连接工厂。
http-invoker.sar	http-invoker.sar 包含支持基于 HTTP 的 RMI 的脱管调用者（detached invoker）。它也包含基于 HTTP 访问 JNDI 的代理绑定。
jboss-aop-jdk50.deployer	此服务配置 AspectManagerService 并部署 JBoss AOP 应用程序。
jboss-bean.deployer	jboss-bean.deployer 提供了 JBoss microcontainer，它部署包含在 .beans 文件里的 POJO 服务。
jboss-ha-local-jdbc.rar	jboss-ha-local-jdbc.rar 是支持数据源失效切换的 jboss-local-jdbc.rar 的实验版本。
jboss-ha-xa-jdbc.rar	jboss-ha-xa-jdbc.rar 是支持数据源失效切换的 jboss-xa-jdbc.rar 的实验版本。
jboss-local-jdbc.rar	jboss-local-jdbc.rar 是一个 JCA 资源适配器，它实现支持 DataSource 接口但不支持 JCA 的 JDBC 驱动的 JCA ManagedConnectionFactory 接口。
jboss-xa-jdbc.rar	jboss-xa-jdbc.rar 是一个 JCA 资源适配器

文件	描述
	，它实现支持 XADataSource 接口的 JDBC 驱动的 JCA ManagedConnectionFactory 接口。
jboss.jca-service.xml	jboss.jca-service.xml 是 JCA 规格的应用服务器实现。它提供了将资源适配器集成到 JBoss 服务器的连接管理工具。
jboss-web.deployer	jboss-web.deployer 目录提供 Tomcat servlet 引擎。
jbossws.sar	jbossws.sar 提供 J2EE web 服务的支持。
jboss-messaging.sar/hsqldb-persistence-service.xml	hsqldb-persistencee-service.xml 提供 Hypersonic 的 JMS 状态管理。
jboss-messaging.sar/destinations-service.xml	destinations-service.xml 配置一系列 JMS 单元测试所使用的 JMS 队列和主题。
jboss-messaging.sar/messaging-service.xml	messaging-service.xml 文件配置了核心的 JBoss Messaging JMS 服务。
jms-ra.rar	jms-ra.rar 是一个资源适配器，它为 JMS 连接工厂实现了 JCA ManagedConnectionFactory 接口。
jmx-console.war	jmx-console.war 目录提供 JMX 控制台。JMX 控制台提供管理 MBean 服务器的简单的 web 界面。
jmx-invoker-service.sar	jmx-invoker-service.sar 是一个未解压的 MBean 服务归档，它开放了 JMX MBeanServer 接口方法的一个子集作为 RMI 接口来启用对 JMX 核心功能的远程访问。它和旧的 jmx-rmi-adaptor.sar 类似，区别是传输是由脱管调用者架构来处理的。
jsr-88-service.xml	jsr-88-service.xml 提供了 JSR 88 远程部署服务。
mail-ra.rar	mail-ra.rar 是提供 JavaMail 连接器的资源适配器。
mail-service.xml	mail-service.xml 文件是一个 MBean 服务描述符，它提供在 JBoss 服务器内部使用的 JavaMail 会话。
management/console-mgr.sar	console-mgr.sar 提供 Web 控制台。它是一个 web 应用程序/applet，提供比 JMX 控制台更丰富的 JMX 服务器管理视图。你可以用 URL http://localhost:8080/web-console/ 来访问此控制台。
monitoring-service.xml	monitoring-service.xml 文件配置了警告监控器

文件	描述
	，如 JMX 通知使用的控制台侦听者和电子邮件侦听者。
properties-service.xml	properties-service.xml 文件是一个 MBean 服务描述符，它允许自定义 JavaBeans PropertyEditor 以及系统属性。
scheduler-service.xml	scheduler-service.xml 和 schedule-manager-service.xml 文件是 MBean 服务描述符，它提供一个调度类型的服务。
sqlexception-service.xml	sqlexception-service.xml 文件是一个 MBean 服务描述符，它处理和供应商相关的 SQLException。
uuid-key-generator.sar	uuid-key-generator.sar 服务提供基于 UUID 的密钥生成工具。

表 2.4. "deploy" 目录的内容

2.3. "all" 服务器配置文件集

"all" 服务器配置文件集位于 JBOSS_DIST/jboss-as/server/all 目录。除了 "default" 集里的服务，all 配置在 conf/ 目录里还包含了下面的几个其他服务。

文件	描述
cluster-service.xml	此服务为 JBoss 里的多数群集服务配置群集通信。
deploy-hasingleton-service.xml	它提供 HA 单点登录（singleton）服务，允许 JBoss 管理只能在群集系统里某一个节点活动的服务。
deploy.last/farm-service.xml	farm-service.xml 提供 farm 服务，它允许服务跨群集的部署和卸载。
httpha-invoker.sar	此服务为群集环境提供 HTTP 管道支持。
iiop-service.xml	它提供 IIOP 调用服务。
juddi-service.sar	此服务提供 UDDI 查找服务。
snmp-adaptor.sar	这是一个 JMX 到 SNMP 的适配器。它允许从 JMX 通知到 SNMP trap 的映射。
tc5-cluster.sar	提供字段级 HTTP 会话复制的 AOP 支持。

表 2.5. "all" 配置在 "conf" 目录里的其他服务

2.4. EJB3 服务

下表解释了提供 ejb3 服务的文件。

文件	描述
ejb3-interceptors-aop.xml	此服务为 EJB3 bean 提供了 AOP 拦截器栈配置。
ejb3.deployer	此服务把 EJB3 应用程序部署到 JBoss 里。
jboss-aop-jdk50.deployer	这是 AOP 部署者基于 Java 5 的版本。AOP 部署者配置 AspectManagerService 且部署 JBoss AOP 应用程序。
jbossws.sar	它提供了基于 Java EE 5 的 web 服务的支持。

表 2.6. EJB3 服务

最后，在 EJB3 的 "all" 配置里，有两个其他的服务。

文件	描述
ejb3-clustered-sfsbcache-service.xml	它为 EJB3 stateful session bean 提供了复制（replication）和失效切换（failover）。
ejb3-entity-cache-service.xml	它为 EJB3 entity bean 提供了群集缓存。

表 2.7. EJB3 "all" 配置里的其他服务

2.5. 添加你自己的配置

你也可以添加自己的配置。最好的办法就是：复制把最贴近你需要的现有配置并进行修改。例如，如果对 messaging 不感兴趣，你可以复制 production 目录，将其命名为 myconfig，然后删除 jms 子目录并用这个新配置启动服务器。

```
run -c myconfig
```

3. 启动和停止服务器

3.1. 启动服务器

要测试你的安装，转到 JBOSS_DIST/jboss-as/bin 目录并执行 run.bat（Windows 平台）或 run.sh（Linux 平台）脚本，这根据你的操作系统而定。输出应该和下面类似（除了安装目录的不同）且没有任何错误或异常信息：

```
[vrenish@vinux bin]$ ./run.sh
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/vrenish/EnterprisePlatform-4.3.0/jboss-as

  JAVA: /usr/java/jdk1.5.0_11/bin/java

  JAVA_OPTS: -Dprogram.name=run.sh -server -Xms1503m -Xmx1503m
             -Dsun.rmi.dgc.client.gcInterval=3600000
-Dsun.rmi.dgc.server.gcInterval=3600000
             -Djava.net.prefer IPv4Stack=true

  CLASSPATH: /home/vrenish/EnterprisePlatform-4.3.0/jboss-as/bin/run.jar:/u
sr/java/jdk1.5.0_11/lib/tools.jar

=========================================================================

14:52:06,251 INFO  [Server] Starting JBoss (MX MicroKernel)...
.
.
.
14:52:07,630 INFO  [ServerInfo] Java version: 1.5.0_11,Sun Microsystems Inc.
14:52:07,631 INFO  [ServerInfo] Java VM: Java HotSpot(TM) Server VM 1.5.0_11-b03 ,Sun
Microsystems Inc.
14:52:07,631 INFO  [ServerInfo] OS-System: Linux 2.6.9-42.0.3.EL,i386
14:52:11,251 INFO  [Server] Core system initialized
14:52:18,230 INFO  [WebService] Using RMI server codebase: http://127.0.0.1:8083/
14:52:18,233 INFO  [Log4jService$URLWatchTimerTask] Configuring from URL:
resource:jboss-log4j.xml
```

注意

请注意，当用 production 配置（缺省配置）启动服务器时，控制台不会显示 "Server Started" 信息。这个信息可以在 server/production/log 子目录下的 server.log 文件里找到。

3.2. 用其他配置启动服务器

不带参数使用 run.sh 来启动服务器会使用 production 服务器配置文件集。要用其他的配置文件集启动，你可以把配置文件集的名字（和 JBOSS_DIST/jboss-as/server 下的目录名相同）作为 -c 命令行选项的参数。例如，要启动 minimal 配置文件集，你应该指定：

```
[bin]$ ./run.sh -c minimal
...
...
...
14:56:30,842 INFO  [Server] JBoss (MX MicroKernel)
                [4.3.0.GA (build: SVNTag=JBPAPP_4_3_0_GA date=200711141859)] Started in
3s:930ms
```

3.3. 使用 run.sh

run 脚本支持下面的选项：

```
usage: run.sh [options]
-h, --help                          Show help message
-V, --version                       Show version information
--                                  Stop processing options
-D<name>[=<value>]      Set a system property
-d, --bootdir=<dir>             Set the boot patch directory; Must be absolute or url
-p, --patchdir=<dir>            Set the patch directory; Must be absolute or url
-n, --netboot=<url>             Boot from net with the given url as base
-c, --configuration=<name>      Set the server configuration name
-B, --bootlib=<filename>        Add an extra library to the front bootclasspath
-L, --library=<filename>        Add an extra library to the loaders classpath
-C, --classpath=<url>           Add an extra url to the loaders classpath
-P, --properties=<url>          Load system properties from the given url
-b, --host=<host or ip>         Bind address for all JBoss services
-g, --partition=<name>          HA Partition name (default=DefaultDomain)
-u, --udp=<ip>                  UDP multicast address
-l, --log=<log4j|jdk>           Specify the logger plugin type
```

3.4. 停止服务器

要关闭服务器，你只要简单地在 JBoss 启动的控制台按 Ctrl-C 键组合就可以了。或者，你可以使用 shutdown.sh 命令。

```
[bin]$ ./shutdown.sh -S
```

shutdown 脚本支持下面的选项：

```
A JMX client to shutdown (exit or halt) a remote JBoss server.
```

```
usage: shutdown [options] <operation>

options:
-h, --help                Show this help message (default)
-D<name>[=<value>]        Set a system property
--                        Stop processing options
-s, --server=<url>        Specify the JNDI URL of the remote server
-n, --serverName=<url>    Specify the JMX name of the ServerImpl
-a, --adapter=<name>      Specify JNDI name of the MBeanServerConnection to use
-u, --user=<name>         Specify the username for authentication
-p, --password=<name>     Specify the password for authentication

operations:
-S, --shutdown            Shutdown the server
-e, --exit=<code>         Force the VM to exit with a status code
-H, --halt=<code>         Force the VM to halt with a status code
```

使用 shutdown 命令要求服务器配置包含有 jmx-invoker-service.xml 服务。因此你不能在 minimal 配置下使用 shutdown 命令。

3.5. 在 Microsoft Windows 里作为服务运行

你可以配置服务器在 Microsoft Windows 里作为服务运行，如果需要也可以配置为自动启动。

从 http://forge.objectweb.org/projects/javaservice/ 里下载 JavaService 软件包。

解压软件包并使用 JBossInstall.bat 文件来安装 JBoss 服务。在运行 JBossInstall.bat 之前，你必须设置环境变量 JAVA_HOME 和 JBOSS_HOME 分别指向 jdk 和 jboss-as 目录。使用下面的语法运行 JBossInstall.bat：

```
JBossInstall.bat <depends> [-auto | -manual]
```

这里的 <depends> 是 JBoss AS 服务器所依赖的任何服务的名字，如 mysql 数据库服务。

一旦安装了这个服务，JBoss AS 服务器就可以用 net start JBoss 命令来启动，或 net stop JBoss 命令停止。

请参考 JavaService 软件包里包括的文档来获得进一步的信息。

4. JMX 控制台

当 JBoss 服务器运行时，你可以通过 http://localhost:8080/jmx-console 访问 JMX 控制台来获得服务器的活动视图。你应该可以看到和 图 2.1 “JMX 管理控制台 Web 应用程序视图” 类似的页面。

JMX 控制台是 JBoss 的管理控制台，它提供构成服务器的 JMX MBeans 的原始视图。它可以提供

运行的服务器的大量信息并允许你修改它的配置、启动和停止组件等等。

例如，在页面上找到 service=JNDIView 链接并点击它。这个 MBean 提供允许你查看服务器内 JNDI 命名空间结构的服务。然后在 MBean 视图页面底部找到名为 list 的操作并点击 invoke 按钮。这个操作将返回绑定到 JNDI 树的当前名称的视图，这对于诊断在部署应用程序时不能解析某个 EJB 名称很有用。

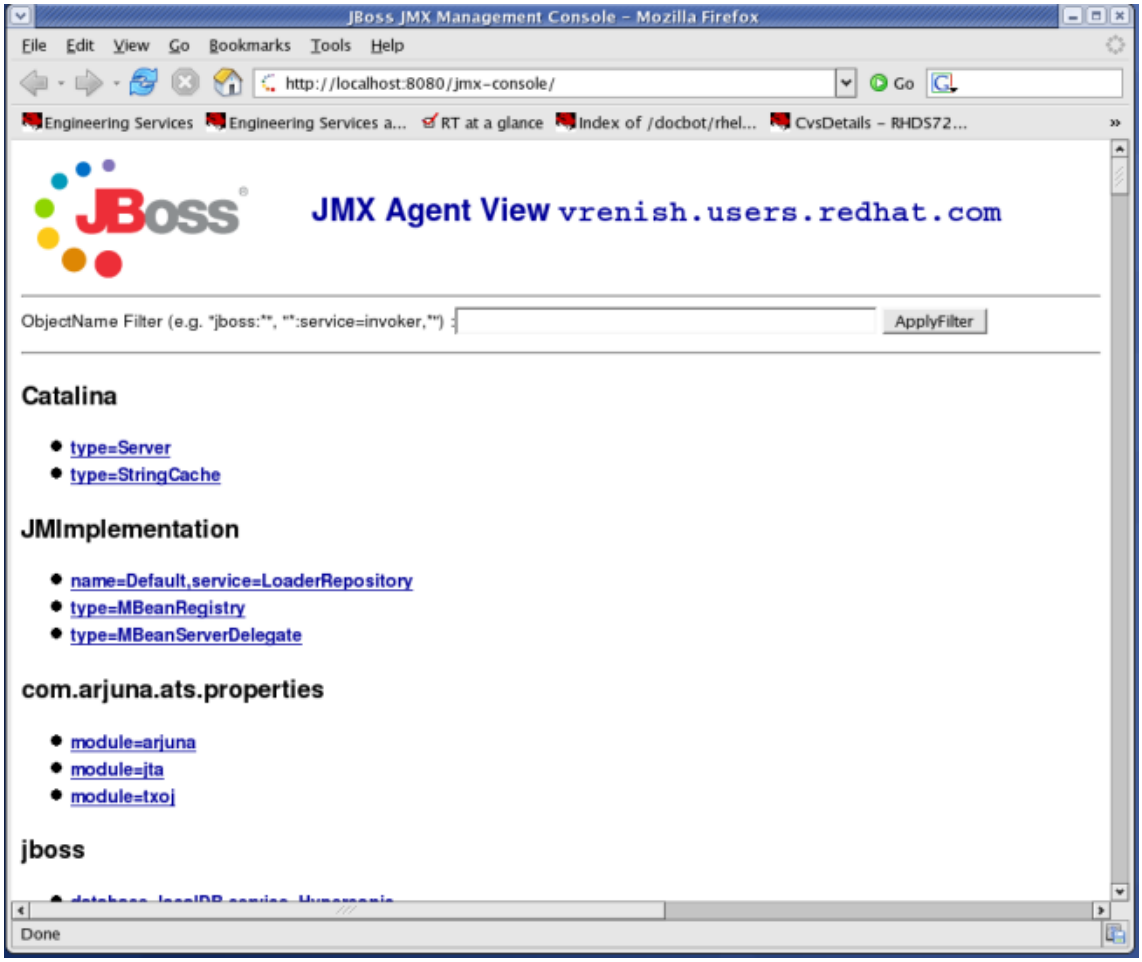

图 2.1. JMX 管理控制台 Web 应用程序视图

看看其他一些 MBean 和它们的操作：尝试改变某些配置属性并查看结果。绝大部分情况下，这种修改都不是持久的。当你重启 JBoss 时，原始的配置将被重新载入。所以，你可以自由尝试而不会造成永久的损害。

注意

如果你是用图形安装程序安装的 JBoss，JMX 控制台将在访问前提示你输入用户名和密码。如果你是用其他方法进行的安装，你仍然可以手工配置 JMX 的安全性

。我们将在 第 6.3 节 “安全服务” 章节向你演示如果设置控制台的安全性。

5. JBoss 里服务的热部署

可热部署的服务是那些可以向运行中的服务器添加和删除的服务。它们位于 JBOSS_DIST/jboss-as/server/<instance-name>/deploy 目录。在了解服务器配置细节之前，让我们看看一个 JBoss 热部署的示例。

启动 JBoss 并进入 server/default/deploy 目录，删除 mail-service.xml 文件并观察服务器的输出：

```
13:10:05,235 INFO  [MailService] Mail service 'java:/Mail' removed from JNDI
```

然后再将这个文件还原，我们可以看到 JBoss 重新安装了这个服务：

```
13:58:54,331 INFO  [MailService] Mail Service bound to java:/Mail
```

。这就是热部署的一个实例。

6. 基本的配置问题

我们已经检查了 JBoss 服务器，现在让我们看看一些主要的配置文件以及用处。所有的路径都是性对于服务器配置目录的（如，server/production）。

6.1. 核心服务

服务器启动时，conf/jboss-service.xml 文件里指定的核心服务将首先启动。如果你用一个编辑器查看这个文件，你将看到不同服务的 MBean，如日志、安全性、JNDI、JNDIView 等等。注释关于 JNDIView 服务的条目。

注意因为这个 MBean 的定义已经嵌入了注释，所我们必须把它分两部分进行注释，并保留原来的注释。

```
<!-- Section 1 commented out
<mbean code="org.jboss.naming.JNDIView"
    name="jboss:service=JNDIView"
    xmbean-dd="resource:xmdesc/JNDIView-xmbean.xml">
-->
    <!-- The HANamingService service name -->
<!-- Section two commented out
    <attribute name="HANamingService">jboss:service=HAJNDI</attribute></mbean>
-->
```

如果你重启 JBoss，你将看到 JNDIView 服务器不再出现在 JMX Management Console（JMX 控制台）列表里。虽然你可以添加额外的 MBean 条目，但实际上，你应该很少甚至根本不需要修改这个文件。替代的办法就是在 deploy 目录里使用一个单独的文件，这允许你热部署自己的服务。

6.2. 日志服务

在 JBoss 里，log4j 被用于日志记录。如果你不熟悉 log4j 软件包但要在应用程序里使用它，你可以在 Jakarta 的网站（http://jakarta.apache.org/log4j/）上获得更多信息。

日志是通过 conf/log4j.xml 文件进行集中控制的。这个文件定义了一系列的 appender、日志文件、各个类别的信息应该记录的位置、信息格式以及过滤级别。在缺省情况下，JBoss 把输出同时发送到控制台和一个日志文件（log 目录下的 server.log）。

我们可以使用 5 个日志级别：DEBUG、INFO、WARN、ERROR 和 FATAL。控制台上的日志级别极限是 INFO，这意味着你将看到信息性的消息、警告消息和错误消息，但你看不到通常的调试消息。相反，server.log 就没有设置日志极限，所以所有产生的日志消息都会在此记录。

如果出现了问题但控制台上却看不到任何有用的信息，请记得查看 server.log 文件里的调试信息，这可以帮助你追踪问题。然而，正是因为设置的日志极限允许调试信息的显示，这意味着所有的 JBoss 服务都将在这个日志文件里输出详细调试信息。你不得不为单独的类别设定日志限制。我们以下面的类别为例。

```
<!-- Limit JBoss categories to INFO -->
<category name="org.jboss">
    <priority value="INFO"/>
</category>
```

除非进行了其他特定的定义，这将所有 JBoss 类的日志级别限制为 INFO。如果你要修改为 DEBUG，它将产生更为详细的日志输出。

作为另外一个例子：假设为了分析所产生的 SQL 命令，你想把容器管理的持久化引擎的输出设置为 DEBUG 级别并重定向到一个单独的文件 cmp.log。你需要把下列的代码添加到 log4j.xml 文件里：

```
<appender name="CMP" class="org.jboss.logging.appender.RollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/cmp.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
```

```
</appender>

<category name="org.jboss.ejb.plugins.cmp">
    <priority value="DEBUG" />
    <appender-ref ref="CMP"/>
</category>
```

这会创建一个新的文件 appender 并指定它应该被软件包 org.jboss.ejb.plugins.cmp 的 logger（或 category）所使用。

这个文件 appender 被设置为每天产生一个新的日志文件，而不是每次重启服务器时产生或者一直写入到单一的文件里。当前的日志文件是 cmp.log，而旧的文件名里含有写入的日期。你将注意到 log 目录也包含 web 容器产生的 HTTP 请求日志。

6.3. 安全服务

安全域信息作为命名的安全域列表存放在 login-config.xml 文件里，每个指定了许多 JAAS [1] 和在该域里用于验证目的的登录模块。当你希望在应用程序里使用安全性时，你可以在该程序的和 JBoss 相关的部署描述符 jboss.xml 和/或 jboss-web.xml 里指定域的名字。我们将快速浏览一下如何用这个办法来设置 JMX 控制台应用程序的安全性。

在《安装指南》里我们提及了 JMX 控制台。JBoss 服务器差不多每一个方面都可以通过 JMX 控制台来控制，所以你至少要确保它是有密码保护的。否则，任何远程用户都可以完全地控制你的服务器。为了保护它，我们将添加一个安全域。[2] 这可以通过 deploy/jmx-console.war/WEB-INF/ 目录里 JMX 控制台的 jboss-web.xml 文件里进行设置。如下所示，取消 security-domain 部分的注释。

```
<jboss-web>
    <security-domain>java:/jaas/jmx-console</security-domain>
</jboss-web>
```

这把安全域链接至 web 应用程序，但它没有告诉 web 应用程序执行哪个安全策略、保护哪个 URL 以及允许谁访问。要配置这些，可以打开同一目录下的 web.xml 文件并取消 security-constraint 部分的注释。这个安全性约束将要求 JBossAdmin 组里用户的有效用户名和密码。

```
<!--
   A security constraint that restricts access to the HTML JMX console
   to users with the role JBossAdmin. Edit the roles to what you want and
   uncomment the WEB-INF/jboss-web.xml/security-domain element to enable
```

[1] Java 验证和授权服务。 JBoss 使用 JAAS 来提供可插拔的验证模块。你可以使用这些或是根据自己的特定需要编写自己的模块。

[2] 如果你的 JBoss 是用图形化安装程序安装的并设置了 JMX 安全性，你将不需要取消这些部分的注释，因为它们已经是没有注释的。此外，admin 的密码将设置为任何你指定的密码。

```
   secured access to the HTML JMX console.
-->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HtmlAdaptor</web-resource-name>
        <description>
            An example security config that only allows users with the
            role JBossAdmin to access the HTML JMX console web application
        </description>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>JBossAdmin</role-name>
    </auth-constraint>
</security-constraint>
```

这很不错，但用户名和密码从哪而来呢？它们来自应用程序链接到的 jmx-console 安全域。我们已经在 conf/login-config.xml 里提供了相应的配置。

```
<application-policy name="jmx-console">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                      flag="required">
            <module-option name="usersProperties">
                props/jmx-console-users.properties
            </module-option>
            <module-option name="rolesProperties">
                props/jmx-console-roles.properties
            </module-option>
        </login-module>
    </authentication>
</application-policy>
```

这个配置使用了一个简单的基于文件的安全策略。配置文件可以在你的服务器配置的 conf/props 目录里找到。用户名和密码存放在 jmx-console-users.properties 文件里，格式为 "username=password"。要分配一个用户给 JBossAdmin 组，可以在 jmx-console-roles.properties 文件里添加 "username=JBossAdmin"。现有的文件会创建一个密码为 admin 的用户 admin。基于安全考虑，请删除这个用户或是设置更复杂的密码。

任何时候只要你更新 JMX 控制台的 web.xml，JBoss 将重新部署它。你可以检查服务器控制台来验证 JBoss 是否已经检测到这些变化。如果你已经配置好了一切且重新部署了应用程序，下次你访问 JMX 控制台的时候，JBoss 将要求输入用户名和密码。[3]

JMX 控制台不是 JBoss 唯一的基于 web 的管理界面。Web 控制台也是其中之一。虽然它是一个 Java applet，对应的 web 应用程序也可以用和 JMX 控制台相同的方法进行安全设定。Web 控制台对于的应用程序是 deploy/management/web-console.war。唯一的区别是，Web 控制台是用简单的 WAR 文件形式来提供的，而不是使用 JMX 控制台所采用的展开的目录结构。而真正的区别是编辑 WAR 文件里的文件会更为麻烦。

6.4. 其他服务

非核心的、可热部署的服务也被添加到 deploy 目录里。它们可以是 XML 描述符文件 *-service.xml，或是 JBoss 服务归档文件（Service Archive，SAR）。SAR包含 XML 描述符文件和该服务需要的其他资源（如类、库 JAR 文件或其他归档文件），所有东西都打包至单一归档文件。

这些服务的详细信息可以在《JBoss 企业版应用程序平台：服务器配置指南》里找到，这本书也提供服务器内部以及服务的实施如 JTA 和 J2EE Connector Architecture（JCA）等方面的综合信息。

7. Web 容器 - Tomcat

JBoss 企业级应用程序平台把 Tomcat 作为缺省的 web 容器。内嵌的 Tomcat 服务位于 deploy/jboss-web.deployer 目录。所有 Tomcat 需要的 jar 文件都可以在这里找到，这里还有一个 web.xml 文件，它为 web 应用程序提供缺省的配置集。

如果你已经熟悉 Tomcat 的配置，你可以看看包含标准 Tomcat 格式的配置子集的 server.xml 文件。它包含对缺省端口 8080 的 HTTP 连接器、端口 8009 的 AJP 连接器（如果你通过 web 服务器如 Apache 连接时使用）的设置以及怎样配置 SSL 连接器的示例（缺省是被注释的）。

除了一些高级应用，你应该不需要修改任何内容。如果你以前曾把 Tomcat 作为独立服务器使用，你应该意识到和作为内嵌服务使用有所不同。JBoss 负责所有事情，你应该根本不需要访问 Tomcat 目录。Web 应用程序部署在 JBoss 的 deploy 目录，Tomcat 的日志输出可以在 JBoss 的 log 目录下找到。

[3] 既然用户名和密码是 web 浏览器里的会话变量，你可能需要关闭浏览器并重新打开才可以看到登录对话框。

JBoss 企业级应用程序平台4.3 里的 EJB3 Caveat

为 JBoss 企业级应用程序平台4.3 进行应用程序开发时，你应该意识到许多未实现的故能。

1. 未实现的特征

JBoss 企业级应用程序平台4.3 的发行注记包含还没有实现或部分实现的 EJB3 特征的信息。发行注记包含 JIRA 里关于这些问题的链接，如绕开的技巧或进一步的细节。

2. 从非 EJB3 Bean 里引用 EJB3 Session Bean

JBoss 企业级应用程序平台5 将完全支持完整的 Java 5 EE 规格。同时，通过将 EJB MBean 容器作为插件在 JBoss 应用服务器里运行，JBoss 企业级应用程序平台4.3 实现了 EJB3 功能。对于应用程序开发而言，这代表着某些含意。

EJB3 插件注入了对 EntityManager 引用和从一个 EJB 对象到另一个对象的 @EJB 引用。然而，这种支持仅限于 EJB3 MBean 及其管理的 JAR 文件。任何从 WAR（如 Servlets、JSF backing bean 等）加载的 JAR 文件都不能采取这种处理方式。Java 5 EE 标准规定了 Servlet 可以通过 @EJB 注解来引用 Session Bean，JBoss 企业级应用程序平台4.3 还没有实现这点。

要从 Servlet 或 JSF Backing Bean 访问 EJB3 Session Bean，你需要做下面的两件事情：

1.
如没有 Seam - JNDI 查找 .

如果没有使用作为 JBoss 企业级应用程序平台4.3 一部分的 Seam 框架，你将需要使用显性的 JNDI 查找来访问 EJB3 Session Bean。如 第 5 章 JSF-EJB3 示例程序所描述的，你可以看到 jsfejb3 示例程序里的 TodoBean.java 使用了这个方法。

```
private TodoDaoInt getDao () {
        try {
        InitialContext ctx = new InitialContext();
        return (TodoDaoInt) ctx.lookup("jsfejb3/TodoDao/local");
        } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException("couldn't lookup Dao", e);
        }
        }
```

ctx.lookup("jsfejb3/TodoDao/local"); 是用来引用 EJB3 Session Bean 的方法。其格式是：AppName/SessionBeanName/local。

2.

如果使用 Seam – 把它留给 Seam 框架来解决 .

当你正在使用 Seam 框架时，你不需要担心这个。因为 Seam 框架管理 Bean 的交互，它已经将这种交互自动化了。

请参考 第 6 章 使用 Seam 里关于用 Seam 框架来实现的细节。

关于示例程序

在本指南里，我们利用一个简单的 web 应用程序来演示 JSF-EJB3 组件的使用。然后我们将说明如何使用 Seam 来集成 JSF 和 EJB3 组件。你可以在 JBOSS_DIST/doc/examples/gettingstarted 目录里找到本指南附带的示例程序源码。你也可以从 http://www.redhat.com/docs/manuals/jboss 下载这些例程。我们在本书里使用两个示例：

简单的创建、查看和编辑任务的 "TODO" 应用程序 - 用 JSF 和 EJB3 实现；

使用 SEAM 框架实现相同功能的程序。

如果你在硬盘里安装了文档，第一个示例可以在 JBOSS_DIST/doc/examples/gettingstarted/jsfejb3 里找到。我们将学习如何使用这里的 build.xml 文件构建这个例程并部署它。我们也将讨论 .java、.xml 和 .properties 文件的细节。

本书使用的第二个例程可以在 JBOSS_DIST/doc/examples/gettingstarted/seamejb3 里找到。我们将利用简单的 "TODO" 应用程序来演示 Seam 怎样将数据库、web 接口和 EJB3 商业逻辑集成在一个 web 应用程序里。我们将用这里的 build.xml 文件来编译和构建这个 Seam 应用程序。

在 JBOSS_DIST/doc/examples/gettingstarted/<seamejb3|jsfejb3> 目录里，你将找到如下的子目录：

src：包含 Java 源码文件。

view：包含 web 页面文件。

resources：包含所有的配置文件。

1. 安装 Ant

要编译并将这些例程打包，你必须安装 Apache Ant 1.6+。你可以从 http://ant.apache.org 下载它并用下面几个步骤进行安装：

把下载的文件解压到你选择的目录。

创建一个名为 ANT_HOME 且指向 Ant 安装目录的环境变量。你可以在 .bashrc 文件里加入类似下面的一行（替换 ant 的实际位置）来进行设置：

```
export ANT_HOME=/home/user/apache-ant-1.7.0
```

在 Windows 平台里，你可以从「开始」菜单里选择「控制面板」（如果需要切换至经典视图），然后打开「系统」 >「高级」 >「环境变量」。创建一个新的变量，命名为 ANT_HOME 并

设置为 ant 的安装目录。

把 $ANT_HOME/bin 加入到系统路径，这样你就可以从命令行运行 ant。你可以通过在 .bashrc 文件里加入下列的行进行设置：

```
export PATH=$PATH:$ANT_HOME/bin
```

在 Windows 平台里，你可以从「开始」菜单里选择「控制面板」（如果可能切换至经典视图），然后打开「系统」 >「高级」 >「环境变量」 >「系统变量」里编辑 PATH 环境变量，加入一个分号和 ant 的 bin 目录。

验证 Ant 的安装。在命令提示下输入 ant -version。输出应该和下面类似：

```
Apache Ant version 1.7.0 compiled on December 13 2006
```

JSF-EJB3 示例程序

我们用一个简单的 "TODO" 应用程序来展示在 web 程序里怎样使用 JSF 和 EJB3。"TODO" 应用程序的功能是：你可以用 "Create" web 表单创建一个新的 'todo' 任务。每个任务都有一个 'title' 和 'description'。当你提交这个表单时，应用程序保存你的任务到一个关系型数据库里。使用这个应用程序，你可以查看所有 'todo' 任务，编辑/删除现有的 'todo' 项目并更新数据库里的任务。

这个示例应用程序由下面的组件组成：

实体对象 - 这些对象代表数据模型；对象的属性映射关系型数据库表里的字段。

JSF web 页面 - 捕获输入数据和显示结果的 web 界面。这些 web 页面上的数据字段通过 JSF Expression Language (EL) 映射至数据模型。

EJB3 Session Bean - 这是功能的实现部分。我们使用一个无状态的会话 Bean (Stateless Session Bean) 。

1. 数据模型

让我们看看 Todo.java 文件里的 Todo 类所代表的数据模型的内容。Todo 类的每个实例都对应数据库表的一行记录。'Todo' 类有 3 个属性：id、title 和 description。每个属性都对应数据库表里的一个字段。

'Entity class' 到 'Database Table' 的映射信息是在 'Todo' 里用 EJB3 注解指定的。这消除了对 XML 配置文件的需要并使程序更加清晰。@Entity 注解表示 Todo 类是一个实体 Bean (Entity Bean) 。id 属性的 @Id 和 @GeneratedValue 注解表示 id 字段是主键，且它的值由服务器为保存到数据库里的每个 Todo 对象自动生成。

```
@Entity
public class Todo implements Serializable {

  private long id;
  private String title;
  private String description;

  public Todo () {
    title ="";
    description ="";
  }

  @Id @GeneratedValue
  public long getId() { return id;}
  public void setId(long id) { this.id = id; }
```

```
  public String getTitle() { return title; }
  public void setTitle(String title) {this.title = title;}

  public String getDescription() { return description; }
  public void setDescription(String description) {
    this.description = description;
  }

}
```

2. JSF Web 页面

在本节，我们将向你展示 web 界面是怎样用 JSF 页面定义的。我们也将看到数据模型是怎样用 JSF EL 映射到 web 表单的。使用 #{...} 注解来引用 Java 对象被称为 JSF EL (JSF Expression Language)。让我们看看在这个应用程序里使用的页面：

index.xhtml：此页面显示两个选项：1. 创建新的 Todo 2. 显示所有的 Todo 任务。当你点击「提交」按钮时，相应的动作将被调用。

```
<h:form>
<ul>
  <li><h:commandLink type="submit" value="Create New Todo" action="create"/></li>
  <li><h:commandLink type="submit" value="Show All Todos" action="todos"/></li>
</ul>
</h:form>
```

create.xhtml：当你试图创建一个新的任务时，此 JSF 页面将捕获输入数据。我们使用 todoBean 来处理表单里的输入字段。#{todoBean.todo.title} 符号引用 "todo" 对象的 "title" 属性；#{todoBean.todo.description} 符号引用 "todo" 对象的 "description" 属性。#{todoBean.persist} 符号引用 "TodoBean" 对象的 "persist" 方法。这个方法用输入的数据（标题和描述）创建 "Todo" 实例并将数据持久化。

```
<h:form id="create">
<table>
  <tr>
    <td>Title:</td>
    <td>
      <h:inputText id="title" value="#{todoBean.todo.title}" size="15">
        <f:validateLength minimum="2"/>
      </h:inputText>
    </td>
  </tr>
```

```
  <tr>
    <td>Description:</td>
    <td>
      <h:inputTextarea id="description" value="#{todoBean.todo.description}">
        <f:validateLength minimum="2" maximum="250"/>
      </h:inputTextarea>
    </td>
  </tr>
</table>
<h:commandButton type="submit" id="create" value="Create"
                 action="#{todoBean.persist}"/>
</h:form>
```

图 5.1 “"Create Todo" web 页面 ” 用映射到数据模型的输入字段显示 "Create Todo" web 页面。

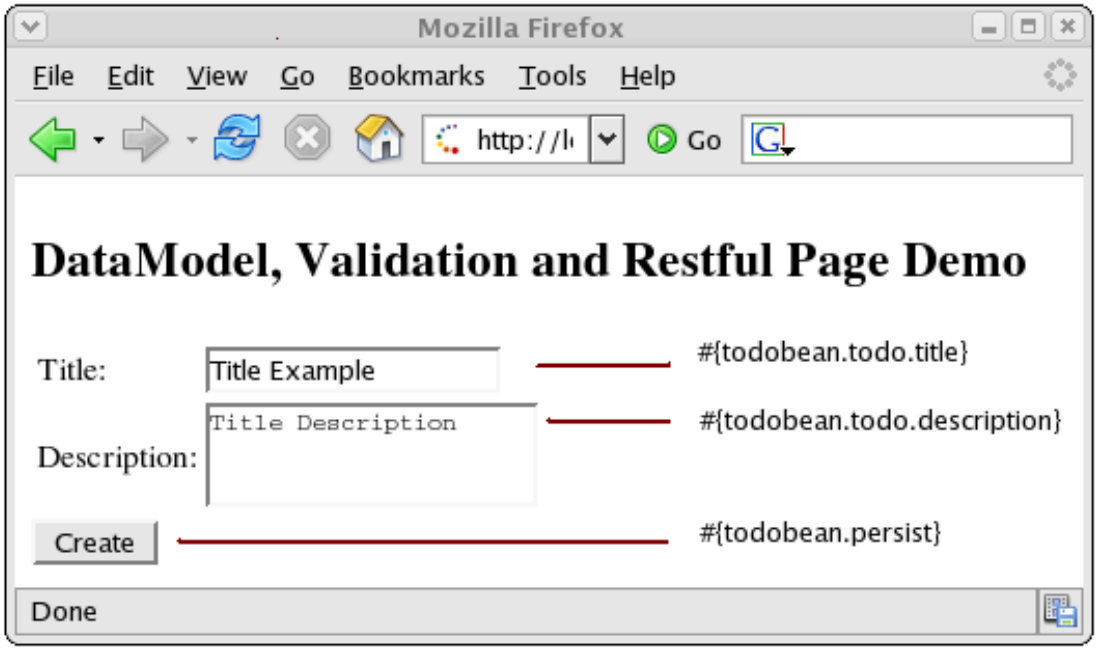

图 5.1. "Create Todo" web 页面

todos.xhtml：此页面显示所有创建的 "todo" 任务的列表。这里也有编辑或删除 "todo" 任务的选项。

所有 'todo' 任务的列表通过引用 'TodoBean' 类的 'getTodos()' 属性的 #{todoBean.todos} 符号来获取。JSF dataTable 遍历这个列表并在一行里显示每个 Todo 对象。每一行都有可用的 'Edit' 选项。#{todo.id} 符号代表 "todo" 对象的 "id" 属性。

```
<h:form>
```

```
<h:dataTable value="#{todoBean.todos}" var="todo">
  <h:column>
    <f:facet name="header">Title</f:facet>
    #{todo.title}
  </h:column>
  <h:column>
    <f:facet name="header">Description</f:facet>
    #{todo.description}
  </h:column>
  <h:column>
    <a href="edit.faces?tid=#{todo.id}">Edit</a>
  </h:column>
</h:dataTable>
<center>
  <h:commandButton action="create"
            value="Create New Todo" type="submit"/>
</center>
</h:form>
```

图 5.2 “"Show All Todos" web 页面 ” 用映射到数据模型的数据字段显示 "Show All Todos" web 页面。

图 5.2. "Show All Todos" web 页面

edit.xhtml：此页面运行你编辑 "todo" 项的 'title' 和 'description' 属性。#{todoBean.update} 和 #{todoBean.delete} 属性代表 "TodoBean" 类里的 "update" 和 "delete" 方法。

```
<h2>Edit #{todoBean.todo.title}</h2>
<h:form id="edit">
<input type="hidden" name="tid" value="#{todoBean.todo.id}"/>
<table>
  <tr>
    <td>Title:</td>
    <td>
      <h:inputText id="title" value="#{todoBean.todo.title}" size="15">
        <f:validateLength minimum="2"/>
      </h:inputText>
    </td>
  </tr>
  <tr>
    <td>Description:</td>
    <td>
      <h:inputTextarea id="description" value="#{todoBean.todo.description}">
        <f:validateLength minimum="2" maximum="250"/>
      </h:inputTextarea>
    </td>
  </tr>
</table>
<h:commandButton type="submit" id="update" value="Update"
                 action="#{todoBean.update}"/>
<h:commandButton type="submit" id="delete" value="Delete"
                 action="#{todoBean.delete}"/>
</h:form>
```

图 5.3 “"Edit Todo" web 页面 ” 用数据模型的映射显示 "Edit Todo" web 页面。

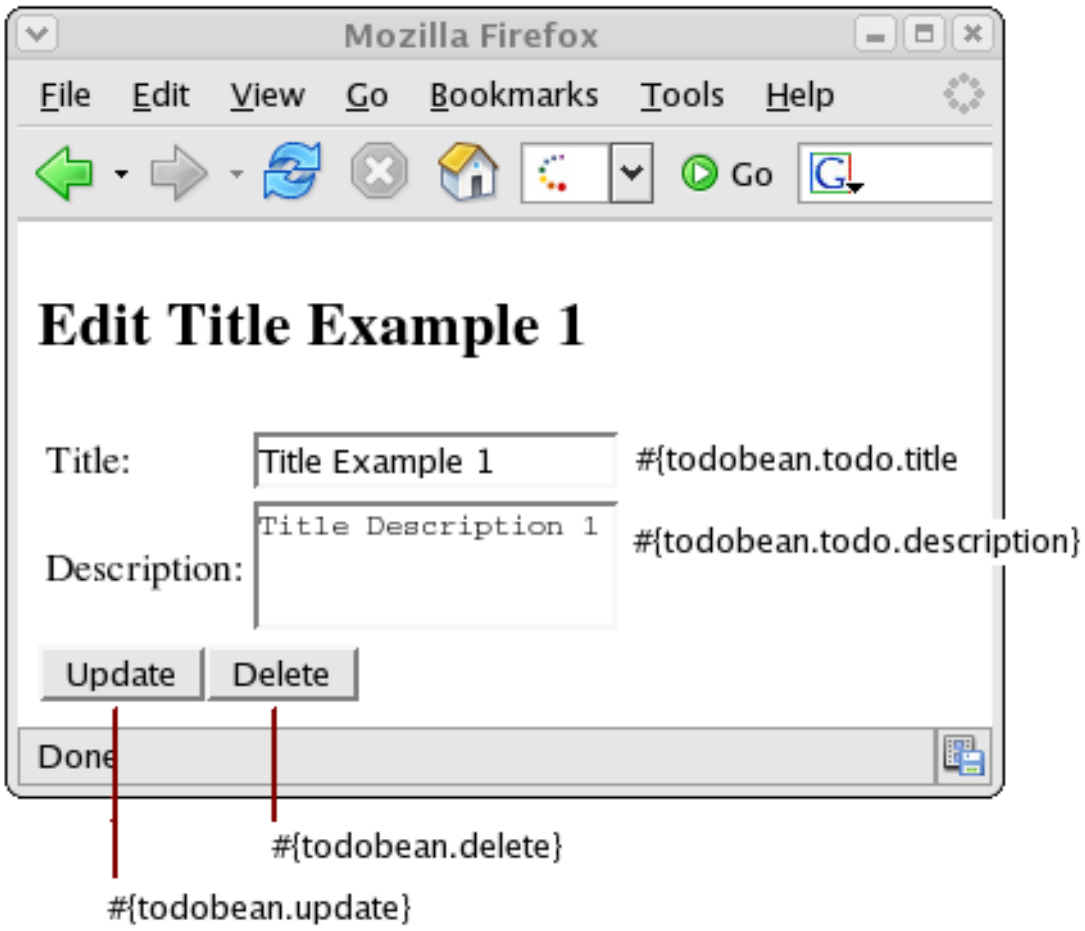

图 5.3. "Edit Todo" web 页面

注意

在示例程序里我们使用了 XHTML 页面，这是因为我们推荐用 Facelets 代替 JSP 来显示 JSF 视图页面。

3. EJB3 Session Bean

EJB 3.0 是 Java EE 5.0 里最重要的改进之一。它的目的是减少旧版本 EJB 的复杂度并简化企业级 Java 开发和部署。你将注意到，要把一个类声明为 'Session Bean'，你只需要简单地注解（annotate）就可以了。使用注解消除了大量部署描述符文件所带来的复杂性。EJB3 Session Bean 仅需的接口是一个商业接口，它声明这个 bean 必须实现的所有的商业方法。

我们将探讨和 Bean 的实现相关联的两个重要的源文件：TodoDaoInt.java 和 TodoDao.java。

Business interface：TodoDaoInt.java

我们在这里定义了 bean 实现类需要实现的方法。基本上，本应用程序里用到的商业方法都在这里定义。

```
public interface TodoDaoInt {

  public void persist (Todo todo);
  public void delete (Todo todo);
  public void update (Todo todo);

  public List <Todo> findTodos ();
  public Todo findTodo (String id);
}
```

Stateless Session Bean：TodoDao.java

@Stateless 注解把此 bean 标记为一个 stateless session bean。在这个类里，我们需要访问之前定义的 Entity bean 'Todo'。因此我们需要一个 EntityManager。@PersistenceContext 注解告诉 JBoss 服务器在部署时注入（inject）一个 entity manager。

```
@Stateless
public class TodoDao implements TodoDaoInt {

  @PersistenceContext
  private EntityManager em;

  public void persist (Todo todo) {
    em.persist (todo);
  }

  public void delete (Todo todo) {
    Todo t = em.merge (todo);
    em.remove( t );
  }

  public void update (Todo todo) {
    em.merge (todo);
  }

  public List <Todo> findTodos () {
    return (List <Todo>) em.createQuery("select t from Todo t")
                                .getResultList();
  }

  public Todo findTodo (String id) {
    return (Todo) em.find(Todo.class, Long.parseLong(id));
  }
```

```
}
```

4. 配置和打包

我们将用 Ant 构建示例应用程序并探讨配置和打包的细节。如果你还没有安装 Ant，现在就开始吧。

4.1. 构建应用程序

让我们先看看此示例程序的构建，然后再探讨配置文件的细节。

在 第 4 章 关于示例程序里，我们说明了 jsfejb3 示例程序的目录结构。在命令行上，进入 jsfejb3 目录。你将看到一个 build.xml 文件。这是用来编译和打包归档文件的 Ant 构建脚本。要构建这个应用程序，你只要输入 ant 命令就可以了，你应该看到和下面类似的输出：

```
[vrenish@vinux jsfejb3]$ ant
Buildfile: build.xml

compile:
    [mkdir] Created dir:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/build/classes
    [javac] Compiling 4 source files to
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3
    /build/classes
    [javac] Note:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/src/TodoDao.java uses
    unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

war:
    [mkdir] Created dir:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/build/jars
      [war] Building war:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/build/jars/
      app.war

ejb3jar:
      [jar] Building jar:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/build/jars/
      app.jar

ear:
      [ear] Building ear:
/home/vrenish/jboss-eap-4.2/doc/examples/gettingstarted/jsfejb3/build/jars/
      jsfejb3.ear
```

```
main:

BUILD SUCCESSFUL
Total time: 2 seconds
(vrenish@vinux jsfejb3)$
```

如果你看到了 BUILD SUCCESSFUL 信息，你将在这里发现新创建的带有 2 个子目录的 build 目录：

classes：包含已编译的类文件。

jars包含 3 个规定文件 – app.jar、app.war 和 jsfejb3.ear。

app.jar：EJB 代码和描述符。

app.war：web 应用程序，它提供前端界面来让用户和商业组件（EJB）进行交互。jsfejb3/view 目录里包含的 web 源码（HTML、images 等）不经修改地加入了这个规定文件。Ant 也添加 WEB-INF 目录，它包含不直接通过 web 浏览器访问却也是应用程序的一部分的文件。这包含了部署描述符（web.xml）和 web 应用程序所需的额外的 jar 文件。

jsfejb3.ear：此 EAR 文件是完整的应用程序，它包含 EJB 模块和 web 模块。它也包含一个其他的描述符：application.xml。你也可能单独部署 EJB 和 web 模块，但 EAR 文件提供了方便的单一方法。

4.2. 配置文件

现在我们已经构建了这个应用程序，让我们看看一些重要的配置文件。我们已经构建了可以部署的最后的归档文件 – jsfejb3.ear。你的 EAR 文件的内容应该类似于：

```
jsfejb3.ear
|+ app.jar   // contains the EJB code
    |+ import.sql
    |+ Todo.class
    |+ TodoDao.class
    |+ TodoDaoInt.class
    |+ META-INF
        |+ persistence.xml
|+ app.war   // contains web UI
    |+ index.html
    |+ index.xhtml
    |+ create.xhtml
    |+ edit.xhtml
    |+ todos.xhtml
    |+ TodoBean.class
    |+ style.css
```

```
    |+ META-INF
    |+ WEB-INF
       |+ faces-config.xml
       |+ navigation.xml
       |+ web.xml
|+ META-INF  // contains the descriptors
    |+ application.xml
    |+ jboss-app.xml
```

application.xml：此文件列出了 EAR 里的 JAR 文件（这里是 app.jar）并告诉 JBoss 服务器该在哪寻找哪些文件。此文件里的 'context-root' 指定了应用程序的根 URL。

```
<application>
  <display-name>Sample Todo</display-name>
  <module>
    <web>
      <web-uri>app.war</web-uri>
      <context-root>/jsfejb3</context-root>
    </web>
  </module>
  <module>
    <ejb>app.jar</ejb>
  </module>
</application>
```

jboss-app.xml：每个 EAR 应用程序应该为类加载器指定一个唯一的字符串。在这里，我们把应用程序名 'jsfejb3' 作为类加载器的名字。

```
<jboss-app>
  <loader-repository>
    jsfejb3:archive=jsfejb3.ear
  </loader-repository>
</jboss-app>
```

app.jar：它包含 EJB3 Session Bean 和 Entity Bean 类以及相关的配置文件。此外，persistence.xml 文件为 EntityManager 配置了后台数据源（在这里是缺省的 HSQL 数据源）。

```
<persistence>
   <persistence-unit name="helloworld">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/DefaultDS</jta-data-source>
      <properties>
```

```
        <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      </properties>
   </persistence-unit>
</persistence>
```

app.war：它包含了按照 Web Application aRchive (WAR) 规格打包的 Web UI 文件。它包含所有的 web 页面和必需的配置文件。对于所有的 JAVA EE web 应用程序来说，web.xml 文件是一个重要的文件。它是 web 部署描述符文件。faces-config.xml 文件是 JSF 的配置文件。navigation.xml 文件包含了 JSF 页面导航的规则。

```
//faces-config.xml
<faces-config>
  <application>
    <view-handler>
      com.sun.facelets.FaceletViewHandler
    </view-handler>
  </application>
  <managed-bean>
    <description>Dao</description>
    <managed-bean-name>todoBean</managed-bean-name>
    <managed-bean-class>TodoBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>
```

5. 数据库

5.1. 创建数据库 Schema

为了预填充（pre-populate）数据库，我们已经在 examples/gettingstarted/jsfejb3/resources 目录里提供了 SQL 代码（import.sql）来运行 HSQL。当你用 Ant 构建应用程序时，它被打包至 jsfejb3.ear 的 app.jar 文件里。当部署这个应用程序时，你应该能够查看到预填充的数据。

5.2. HSQL 数据库管理者工具

启动 JMX 控制台程序并点击 jboss 部分下的 service=Hypersonic 链接。如果你找不到这个链接，请检查 hsqldb-ds.xml 文件里是否启用了 Hypersonic 服务。

这将带你进入 Hypersonic 服务 MBean 的信息页。向下滚动到这个页面的底部并点击 startDatabaseManager() 操作的 invoke 按钮。这将启动 HSQL 管理者，它是一个你可以用来直接操纵数据库的 Java GUI 应用程序。

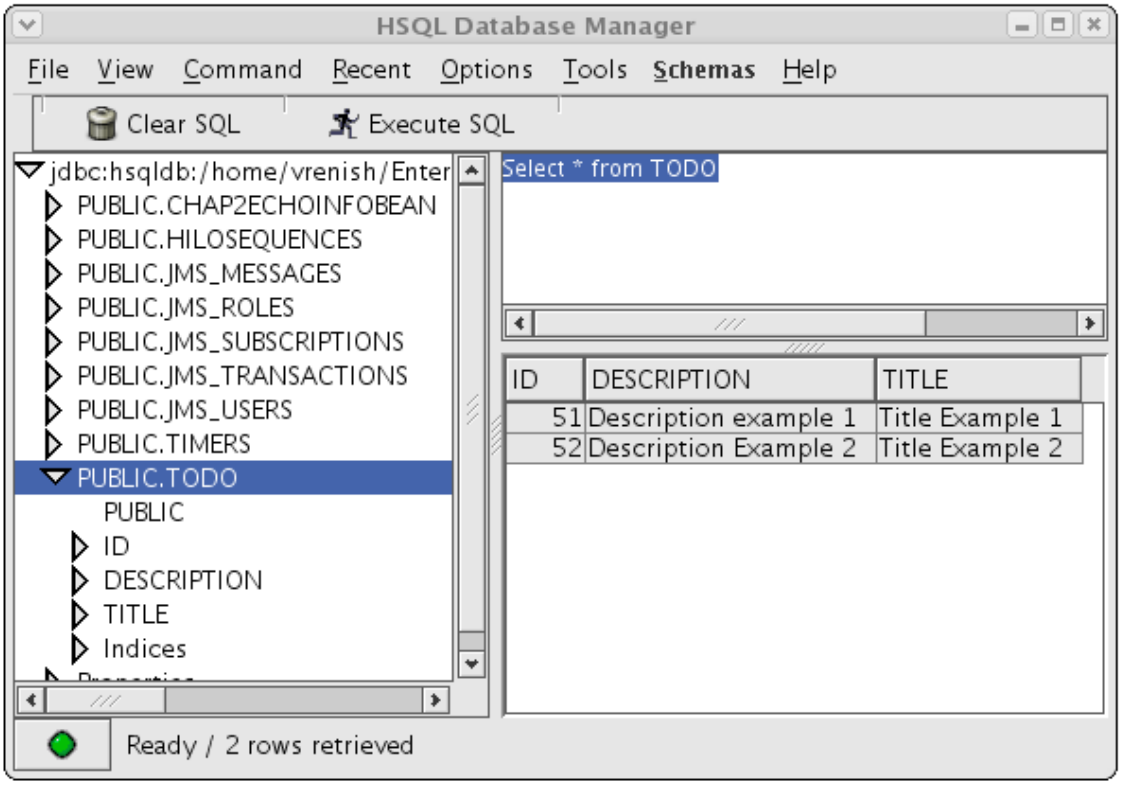

图 5.4. HSQL 数据库管理者

6. 部署这个应用程序

在 JBoss 里部署应用程序是件简单而容易的事情。你只需要把 EAR 文件复制到服务器配置目录的 deploy 子目录下就可以了。在这里我们要把它部署到 'default' 配置里，所以我们需要把这个 EAR 文件复制到 JBOSS_DIST/jboss-as/server/default/deploy。

你应该从服务器里看到类似下面的输出：

```
15:32:23,997 INFO  [EARDeployer] Init J2EE application: file:/home/vrenish/jboss-eap-4.2
/jboss-as/server/default/deploy/jsfejb3.ear
15:32:24,212 INFO  [JmxKernelAbstraction] creating wrapper delegate for: org.jboss.ejb3.
entity.PersistenceUnitDeployment
15:32:24,213 INFO  [JmxKernelAbstraction] installing MBean: persistence.units:ear=
jsfejb3.ear,jar=app.jar,unitName=helloworld with dependencies:
15:32:24,213 INFO  [JmxKernelAbstraction]       jboss.jca:name=DefaultDS,service=
DataSourceBinding
15:32:24,275 INFO  [PersistenceUnitDeployment] Starting persistence unit persistence.
units:ear=jsfejb3.ear,jar=app.jar,unitName=helloworld
15:32:24,392 INFO  [Ejb3Configuration] found EJB3 Entity bean: Todo
15:32:24,450 WARN  [Ejb3Configuration] Persistence provider caller does not implements
the EJB3 spec correctly. PersistenceUnitInfo.getNewTempClassLoader() is null.
15:32:24,512 INFO  [Configuration] Reading mappings from resource : META-INF/orm.xml
```

```
15:32:24,512 INFO  [Ejb3Configuration] [PersistenceUnit: helloworld] no META-INF/orm.xml
found
15:32:24,585 INFO  [AnnotationBinder] Binding entity from annotated class: Todo
15:32:24,586 INFO  [EntityBinder] Bind entity Todo on table Todo
.
.
.
.
15:32:26,311 INFO  [SchemaExport] Running hbm2ddl schema export
15:32:26,312 INFO  [SchemaExport] exporting generated schema to database
15:32:26,314 INFO  [SchemaExport] Executing import script: /import.sql
15:32:26,418 INFO  [SchemaExport] schema export complete
15:32:26,454 INFO  [NamingHelper] JNDI InitialContext properties:{java.naming.factory.
initial=org.jnp.interfaces.NamingContextFactory, java.naming.factory.url.pkgs=org.jboss.
naming:org.jnp.interfaces}
15:32:26,484 INFO  [JmxKernelAbstraction] creating wrapper delegate for: org.jboss.ejb3.
stateless.StatelessContainer
15:32:26,485 INFO  [JmxKernelAbstraction] installing MBean: jboss.j2ee:ear=jsfejb3.ear,
jar=app.jar,name=TodoDao,service=EJB3 with dependencies:
15:32:26,513 INFO  [JmxKernelAbstraction]       persistence.units:ear=jsfejb3.ear,
jar=app.jar,unitName=helloworld
15:32:26,557 INFO  [EJBContainer] STARTED EJB: TodoDao ejbName: TodoDao
15:32:26,596 INFO  [EJB3Deployer] Deployed: file:/home/vrenish/jboss-eap-4.2/jboss-as/
server/default/tmp/deploy/
tmp33761jsfejb3.ear-contents/app.jar
15:32:26,625 INFO  [TomcatDeployer] deploy, ctxPath=/jsfejb3, warUrl=.../tmp/deploy/
tmp33761jsfejb3.ear-contents/app-exp.war/
15:32:26,914 INFO  [EARDeployer] Started J2EE application: file:/home/vrenish/jboss-eap-
4.2/jboss-as/server/default/deploy/jsfejb3.ear
```

如果这里有任何错误或异常，请记下错误信息。然后检查 EAR 文件是否完整并查看 WAR 和 EJB jar 文件，确保它们包含了所有必需的组件（类、描述符等）。

如果这个应用程序已经被部署了，你可以安全地重新部署它。你只需要把归档文件从 deploy 目录删除就可以卸载它。不管是部署还是卸载，你都不需要重启服务器。如果一切都正常，你就可以用浏览器访问应用程序的 URL 了。

http://localhost:8080/jsfejb3

你将被转到这个应用程序的主页。图 5.5 “Sample TODO” 展示了正运行的示例程序。

图 5.5. Sample TODO

使用 Seam

JBoss Seam 是一个框架，它集成了 Java EE 5.0 标准下的新的 EJB3 和 JSF 框架。事实上，Seam 这个名字指的就是让开发人员能够以无缝的方式集成这两个框架。Seam 自动化了许多共同的任务，它大量运用注解来减少需要编写的 xml 代码的数量。而总的效果就是显著地减少了代码量。

我们已经包含了示例程序的两个版本，一个是用 EJB3 / JSF 而不使用 Seam 编写的，而另外一个则使用了 Seam。这可以清楚地演示在程序开发时使用 Seam 框架的区别。

注意

关于 Seam 示例程序的部署和应用程序开发的详情，请参考文档集里的《Seam 参考指南》（JBOSS_DIST/doc/seam/Seam_Reference_Guide.pdf）。

1. 数据模型

在前一章里，我们涉及了在例程的 EJB3/JSF 实现里使用的数据模型。让我们看看使用 Seam 框架时，数据模型怎样被实现。这是在 Todo.java 文件里被完成的。

```
@Entity
@Name("todo")
public class Todo implements Serializable {

  private long id;
  private String title;
  private String description;

  public Todo () {
    title ="";
    description ="";
  }

  @Id @GeneratedValue
  public long getId() { return id;}
  public void setId(long id) { this.id = id; }

  @NotNull
  public String getTitle() { return title; }
  public void setTitle(String title) {this.title = title;}

  @NotNull
  @Length(max=250)
  public String getDescription() { return description; }
```

```
  public void setDescription(String description) {
    this.description = description;
  }

}
```

@Entity 注解把这个类定义为 EJB3 session bean，并告诉容器把 Todo 类映射到关系型数据库的表里。这个类的每个属性都将成为表里的一个字段。每个实例都将成为表里的一行记录。既然我们还没有使用 @Table 注解，Seam 的 "configuration by exception" 缺省将在这个类后给这个表命名。

@Entity 和 @Table 都是 EJB3 注解，都不是 Seam 特定的语法。我们有可能不用任何 EJB3 相关的注解而全部用 POJOs (Plain Old Java Objects) 来使用 Seam。然而，EJB3 为表带来了许多好处，如容器管理的安全性、消息驱动的组件、事务和组件级的持久化上下文以及我们即将遇到的 @PersistenceContext 注入。

@Name 注解是 Seam 特定的语法，它定义 Seam 用来注册 Entity Bean 的字符串。它将成为关系型数据库里表的缺省名称。Seam 应用程序里的每个组件都必须有一个唯一的名字。在 Seam 框架里的其他组件里，如 JSF web 页和 session bean 里，你可以用这个名字来引用被管理的 Todo bean。如果当从其他组件引用时，这个类的实例不存在，Seam 将初始化一个实例。

@Id 注解为组件定义了一个主键 id 字段。@GeneratedValue 指明服务器在组件保存至数据库时将自动地生成它的值。

Seam 提供对 Hibernate Validator 定义的基于模型的约束的支持，虽然 Hibernate 并不需要是被使用的对象持久者。@NotNull 注解是一个验证约束（validation constraint），它要求这个属性在持久化到数据库之前已经被赋值。使用这个注解允许视图级的 JSF 代码进行强制验证，而不需要在 JSF 代码里指定确切的验证约束。

到目前为止，这个 Seam 版本和 EJB3/JSF 版本应用程序之间唯一明显的区别是 @NotNull 和 @Name 注解。然而，EJB3/JSF 版本的应用程序要求进一步手工编写和管理 TodoBean 类来处理 Todo 类和 web 界面之间的交互，而使用 Seam 框架的这个 Seam 版本却替我们完成这一切。在我们研究用户界面的实现时，我们将看到这些是如何完成的。

2. JSF Web 页面 - index.xhtml 和 create.xhtml

index.xhtml 文件和 EJB3/JSF 例程里的是一样的。

create.xhtml 开始展现了用 Seam 框架进行编码的不同之处。

```
<h:form id="create">

<f:facet name="beforeInvalidField">
  <h:graphicImage styleClass="errorImg" value="error.png"/>
</f:facet>
<f:facet name="afterInvalidField">
```

```
  <s:message styleClass="errorMsg" />
</f:facet>
<f:facet name="aroundInvalidField">
  <s:div styleClass="error"/>
</f:facet>

<s:validateAll>

<table>

  <tr>
    <td>Title:</td>
    <td>
      <s:decorate>
        <h:inputText id="title" value="#{todo.title}" size="15"/>
      </s:decorate>
    </td>
  </tr>

  <tr>
    <td>Description:</td>
    <td>
      <s:decorate>
        <h:inputTextarea id="description" value="#{todo.description}"/>
      </s:decorate>
    </td>
  </tr>

</table>

</s:validateAll>

<h:commandButton type="submit" id="create" value="Create"
                 action="#{todoDao.persist}"/>
</h:form>
```

第一个不同的地方是开始的 Java Server Facelet 代码，它和 todo 类的 @NotNull 验证约束一起可以指出无效的用户输入。

也请注意，不是象在 EJB3/JSF 例程里使用 TodoBean 类，我们直接用 Todo entity bean 来处理表单。当这个页面被调用时，JSF 请求 Seam 解析 JSF EL 所引用的变量 todo 如 #{todo.title}。既然还没有值绑定到这个变量名，在把它存入 Seam 的上下文后，Seam 将实例化 todo 类的一个 entity bean 并返回给 JSF。Seam 的上下文可以替代中介的 bean。

表单里的输入根据 todo 类里指定的 Hibernate Validator 约束进行验证。如果违反了该约束，JSF 将重新显示此页面，否则它将表单里的输入值存入 Todo entity bean。

Entity bean 不应该进行数据库访问或事务管理，所以我们不能把 Todo entity bean 当作 JSF action listener。因此，我们通过调用 TodoDao session bean 的 persist 方法来在数据库里创建一个新的 todo 项。当 JSF 请求 Seam 通过 JSF EL 表达式 #{todoDao.persist} 解析变量 todoDao 时，Seam 将实例化一个对象（对象不存在时）或是把现有的 stateful todoDao 对象从 Seam 上下文里传递出来。Seam 将拦截 persist 方法的调用并从会话上下文中注入（inject）todo entity。

让我们看看 TodoDao 类（在 TodoDao.java 定义的）这种注入（injection）能力是怎样被实现的。

3. 用 Session Bean 访问数据

让我们来看看 TodoDao 类的代码清单。

```
@Stateful
@Name("todoDao")
public class TodoDao implements TodoDaoInt {

  @In (required=false) @Out (required=false)
  private Todo todo;

  @PersistenceContext (type=EXTENDED)
  private EntityManager em;

  // Injected from pages.xml
  Long id;

  public String persist () {
    em.persist (todo);
    return "persisted";
  }

  @DataModel
  private List <Todo> todos;

  @Factory("todos")
  public void findTodos () {
    todos = em.createQuery("select t from Todo t")
                                  .getResultList();
  }

  public void setId (Long id) {
    this.id = id;

    if (id != null) {
      todo = (Todo) em.find(Todo.class, id);
    } else {
      todo = new Todo ();
```

```
    }
  }

  public Long getId () {
    return id;
  }

  public String delete () {
    em.remove( todo );
    return "removed";
  }

  public String update () {
    return "updated";
  }

  @Remove @Destroy
  public void destroy() {}

}
```

首先，请注意这个是一个 stateful session bean。Seam 可以使用 stateful 和 stateless session bean，它们都是最常用的 EJB3 bean。

@In 和 @Out 注解定义了 Seam 注入的一个属性。这个属性被注入到这个对象，或者通过 Seam 上下文变量 todo 从这个对象注入到另外一个对象，这个变量是对 Todo.java 里定义的 Todo 类的 Seam 注册名称的引用。

@PersistenceContext 注解注入了 EJB3 Entity manager，它允许这个对象将对象持久化至数据库。因为这是一个 stateful session bean 且 PersistenceContext 类型被设置为 EXTENDED，相同的 Entity Manager 实例将被使用，一直到这个 session bean 的 Remove 方法被调用。resources/META-INF/persistence.xml 文件里定义了所使用的数据库（persistence-unit）。

请注意，这个 session bean 对和 web 请求（todo 对象的表单值）相关联的上下文进行了并行访问，它也具有保持在事务性资源（EntityManager）里的状态。这违背了传统的 J2EE 体系结构，但 Seam 并不强求你使用这种方式。如果你愿意，你可以采用更为传统的应用程序层次结构。

@DataModel 注解初始化 todos 属性，它将被开放（exposed）给视图层。@Factory 注解的方法执行生成 todos 列表的操作，如果 Seam 视图访问被开放的 DataModel 属性并发现这个属性为空时，它将被调用。请注意，todos 属性并没有属性访问方法。Seam 会自动替你完成这些工作。

让我们来看看显示和编辑 todo 列表的 JSF 代码，学习在实践中如何使用这些界面。

4. JSF Web 页面 - todos.xhtml 和 edit.xhtml

使用这个 Session Bean 开放的 DataModel 属性将产生一个 todo 列表：

```
<h:form>

<h:dataTable value="#{todos}" var="todo">
  <h:column>
    <f:facet name="header">Title</f:facet>
    #{todo.title}
  </h:column>
  <h:column>
    <f:facet name="header">Description</f:facet>
    #{todo.description}
  </h:column>
  <h:column>
    <a href="edit.seam?tid=#{todo.id}">Edit</a>
  </h:column>
</h:dataTable>

<center>
  <h:commandButton action="create"
            value="Create New Todo" type="submit"/>
</center>

</h:form>
```

当 JSF 变量解析器遇到 {#todos} 和 todos 请求，Seam 发现当前作用域没有 "todos" 组件时，它将调用 @Factory("todos") 方法来创建组件。既然 todos 对象是用 @DataModel 进行注解的，一旦 factory 方法完成了，todos 对象就会被注出（outject）。

构建编辑页面的视图很简单：

```
<h:form id="edit">

<f:facet name="beforeInvalidField">
  <h:graphicImage styleClass="errorImg" value="error.png"/>
</f:facet>
<f:facet name="afterInvalidField">
  <s:message styleClass="errorMsg" />
</f:facet>
<f:facet name="aroundInvalidField">
  <s:div styleClass="error"/>
</f:facet>

<s:validateAll>

<table>

  <tr>
    <td>Title:</td>
    <td>
```

```
      <s:decorate>
        <h:inputText id="title" value="#{todo.title}" size="15"/>
      </s:decorate>
    </td>
  </tr>

  <tr>
    <td>Description:</td>
    <td>
      <s:decorate>
        <h:inputTextarea id="description" value="#{todo.description}"/>
      </s:decorate>
    </td>
  </tr>

</table>

</s:validateAll>

<h:commandButton type="submit" id="update" value="Update"
                 action="#{todoDao.update}"/>

<h:commandButton type="submit" id="delete" value="Delete"
                 action="#{todoDao.delete}"/>
</h:form>
```

这里我们可以看到相同的东西。JSF 的验证代码利用了在 Entity Bean 里定义的验证约束，并使用todoDao Session Bean 的 update 和 delete 方法来更新数据库。

todos.xhtml 里的调用：edit.seam?tid=#{todo.id} 导致 Seam 创建一个 todoDao 并将它的 id 属性设置为 tid。设置 id 属性可以使 todoDao 从数据库里获取合适的记录。

pages.xml 实现了允许带参数调用编辑页面的功能。让我们看看 Seam 应用程序是怎样使用 pages.xml 文件的。

5. 构建应用程序

从命令行进入 JBOSS_DIST/doc/examples/gettingstarted/seamejb3 目录。你将看到一个 build.xml 文件。这是我们用来编译和打包的 Ant 构建脚本。要构建这个应用程序，键入 ant 命令，你的输出应该类似于：

```
[vrenish@vinux jsfejb3]$ ant

Buildfile: build.xml

compile:
```

```
  [mkdir] Created dir:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/classes
  [javac] Compiling 3 source files to
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/classes
  [javac] Note:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/src/TodoDao.java
                    uses unchecked or unsafe operations.
  [javac] Note: Recompile with -Xlint:unchecked for details.

war:
  [mkdir] Created dir:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/jars
  [war] Building war:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/jars/app. war

ejb3jar:
  [jar] Building jar:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/jars/app. jar

ear:
  [ear] Building ear:
/home/vrenish/jboss-eap-4.3/doc/examples/gettingstarted/seamejb3/build/jars/seam ejb3.ear

main:

BUILD SUCCESSFUL
Total time: 7 seconds
```

如果你看到了 BUILD SUCCESSFUL 信息，你将发现一个新创建的 build 目录，它包含两个子目录：

classes：存放编译好的 .class 文件。

jars：存放 3 个归档文件 - app.jar、 app.war 和 seamejb3.ear。

app.jar

app.war

seamejb3.ear

关于这些文件的更多信息请参考 第 4 节 “配置和打包”。

6. Xml 文件

Seam 极大地减少了 xml 编码。它感兴趣的文件是 app.war 里 WEB-INF 目录下的 pages.xml。这个文件可以在源码包的 resources/WEB-INF 目录下找到。pages.xml 被用来定义页面描述，如

Seam 页面参数（HTTP GET 参数）、页面动作、页面导航规则、出错页面等。Seam 应用程序也可以用它来定义异常处理程序和重定向。

在我们的示例程序里，我们使用它来定义 Seam 页面参数。例程里的 pages.xml 包含下面的代码：

```
<page view-id="/edit.xhtml">
    <param name="tid" value="#{todoDao.id}"
           converterId="javax.faces.Long"/>
</page>
```

这定义了 edit.xhtml 页面的一个名为 tid 的参数。当 edit.xhtml 页面被载入时，HTTP GET 请求参数 tid 被转换为 Long 型值并分配给 todoDao 对象的 id 属性。无论多少，你可以按需要用使用页面参数来把 HTTP GET 请求参数绑定到应用程序后台的组件。

7. 进一步的信息

我们已经完成了这个 Seam 示例程序。关于进一步的使用 Seam 框架开发应用程序的详细信息，请参考 《Seam Reference Guide》。

使用其他数据库

在前面的章节里，我们在应用程序里已经使用了 JBoss 缺省的数据源。这是由内嵌的 HSQL 数据库实例所提供并绑定在 JNDI 名 java:/DefaultDS。JBoss 里包含的数据库对于运行示例非常方便，HSQL 对于许多用途来说也已足够。然而，某些情况下你可能希望使用其他数据库，你可以替换缺省的数据源或是访问多个数据库。

1. 数据源配置文件

数据源配置名以后缀 -ds.xml 结尾，这样才能被 JCA 部署者正确识别。docs/example/jca 目录里包含了许多数据库的样本文件，以它们为起点是个好主意。而对于配置格式的完整描述，最好的参考是 DTD 文件 docs/dtd/jboss-ds_1_5.dtd。这些文件和 JBoss JCA 实现的其他文档也可以在《JBoss 4 应用服务器指南》里找到。

Local transaction 数据源可以用 local-tx-datasource 元素配置，而 XA 兼容的数据源可以用 xa-tx-datasource 进行配置。示例文件 generic-ds.xml 展示了如何使用这两种类型以及其他一些元素如连接池的配置。local 和 XA 配置的示例也可用于 Oracle、DB2 和 Informix。

如果你比较示例文件 firebird-ds.xml、facets-ds.xml 和 sap3-ds.xml，你将注意到它们具有完全不同的格式，根元素是 connection-factories 而不是datasources。它们使用了另外一种更通用的预先打包了 JCA 资源适配器的配置语法。这种语法不是针对于数据源配置，而是用在 jms-ds.xml 文件里来配置 JMS 资源适配器。

下面，我们将使用一些按部就班的例程来说明如何设置针对特定数据库的数据源。

2. 把 MySQL 用作缺省的数据源

MySQL 是某流行的开源数据库之一，它被 Yahoo 和 NASA 等许多重要组织所使用。它的官方 JDBC 驱动名为 Connector/J。例如，我们已经使用了 MySQL 4.1.7 和 Connector/J 3.0.15。你可以从 http://www.mysql.com 下载它们。

2.1. 创建数据库和用户

我们将假设你已经安装了 MySQL 并已启动，且你熟悉基本的操作。从命令行运行 mysql 客户程序来执行一些管理命令。你应该确保你在使用有足够权限的用户（例如，指定 -u root 选项来以 MySQL 根用户运行）。

首先，在 MySQL 里创建一个名为 jboss 的数据库供 JBoss 使用。

```
mysql> CREATE DATABASE jboss;
Query OK, 1 row affected (0.05 sec)
```

然后检查它是否已经被创建。

```
mysql> SHOW DATABASES;
```

```
+----------+
| Database |
+----------+
| jboss    |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)
```

其次，创建一个名为 jboss、密码为 password 的用户来访问这个数据库。

```
mysql> GRANT ALL PRIVILEGES ON jboss.* TO jboss@localhost IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.06 sec)
```

你需要再次检查是否一切正常。

```
mysql> select User,Host,Password from mysql.User;
+-------+-----------+-------------------+
| User  | Host      | Password          |
+-------+-----------+-------------------+
| root  | localhost |                   |
| root  | %         |                   |
|       | localhost |                   |
|       | %         |                   |
| jboss | localhost | 5d2e19393cc5ef67 |
+-------+-----------+-------------------+
5 rows in set (0.02 sec)
```

2.2. 安装 JDBC 驱动并部署数据源

要使 JDBC 驱动为 JBoss 所用，你可以把 mysql-connector-java-3.0.15-ga-bin.jar 文件从 Connector/J 分发包里复制到 default 服务器配置（当然，得假定它是你正运行的配置）的 lib 目录里。然后在 deploy 目录里创建一个名为 mysql-ds.xml 并含有下面的数据源配置的文件。数据库用户名和密码分别对应我们在前面创建的 MySql 用户和密码。

```
<datasources>
    <local-tx-datasource>
        <jndi-name>MySqlDS</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/jboss</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>jboss</user-name>
        <password>password</password>
    </local-tx-datasource>
</datasources>
```

因为我们在 lib 目录里已经添加了一个新的 JAR 文件，你需要确保服务器能够发现这个 MySQL 驱动类。

2.3. 测试 MySQL 数据源

我们将使用 CMP roster 应用程序来测试这个新的数据库连接。为了在我们的应用程序里使用 MySql，我们将需要在 CMP roster 应用程序的 dd/team 目录下的 jbosscmp-jdbc.xml 文件里设置数据源名称和类型映射（type-mapping）。编辑这个文件并在 defaults 元素里加入下面的 datasource 和 datasource-mapping 元素。

```
<jbosscmp-jdbc>
    <defaults>
        <datasource>java:/MySqlDS</datasource>
        <datasource-mapping>mySQL</datasource-mapping>
    </defaults>

    <enterprise-beans>
...
    </enterprise-beans>
</jbosscmp-jdbc>
```

在重启 JBoss 后，你应该能够部署这个应用程序并看到表被创建。这些表应该可以用 MySQL 客户端查看。

```
mysql> show tables;
+-----------------------------------+
| Tables_in_jboss                   |
+-----------------------------------+
| LeagueBean                        |
| PlayerBean                        |
| PlayerBean_teams_TeamBean_players |
| TeamBean                          |
+-----------------------------------+
4 rows in set (0.00 sec)
```

既然我们把 MySQL 作为缺省数据源使用，在这里你也可以看到 JMS 持久化表。

3. 设置 Oracle 9i 的 XADataSource

Oracle 是商业数据库领域的主要供应商之一，大多数读者可能已经和它打过交道。如用于非商业目的，你可以从 http://www.oracle.com 免费下载它。

安装和配置 Oracle 不是件容易的事。它实在不仅是一个简单的数据库，你可能并不需要它的一些额外的功能和技术（如 web 服务器、多重 JDK、Orbs 等），虽然它们通常都会被安装。所以，我们假定你已经安装好了 Oracle。在这个例子里，我们使用 Oracle 10g。

3.1. Oracle 兼容性的 Padding Xid 值

如果你打开 default/conf 目录下的 jboss-service.xml 文件，你将发现下面的服务 MBean。

```
<!-- The configurable Xid factory. For use with Oracle, set pad to true -->
<mbean code="org.jboss.tm.XidFactory"
       name="jboss:service=XidFactory">
    <!--attribute name="Pad">true</attribute-->
</mbean>
```

事务服务使用它来创建 XA 事务标识符。这里的注释解释了：要使用 Oracle，你必须包含设置了属性 Pad 为真的行。这使标识符填充至最大的长度 - 64 个字节。请记住，你将需要重启 JBoss 来使这个改变生效，但请等到你安装了 JDBC 驱动类（我们将在后面讨论）后再重新启动。

3.2. 安装 JDBC 驱动并部署数据源

Oracle JDBC 驱动可以在 $ORACLE_HOME/jdbc/lib 里找到。有些客户所熟悉的旧版本的名字不具有任何信息，如 classes12.zip，但最新的驱动版本可以在文件 ojdbc14.jar 里找到。当你遇到问题时，类名称里附带 _g 的调试版本可能有用。你应该复制其中一个到 JBoss default 配置的 lib 目录里。用于非 XA 设定的基本的驱动类名为 oracle.jdbc.driver.OracleDriver。这里使用的 XADataSource 类，被称为 oracle.jdbc.xa.client.OracleXADataSource。

复制一份 oracle-xa-ds.xml 示例文件作为配置文件，编辑它并设置正确的 URL、用户名和密码。

```
<datasources>
    <xa-datasource>
        <jndi-name>XAOracleDS</jndi-name>
        <track-connection-by-tx>true</track-connection-by-tx>
        <isSameRM-override-value>false</isSameRM-override-value>
        <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
        <xa-datasource-property name="URL">
            jdbc:oracle:thin:@monkeymachine:1521:jboss
        </xa-datasource-property>
        <xa-datasource-property name="User">jboss</xa-datasource-property>
        <xa-datasource-property name="Password">password</xa-datasource-property>
        <exception-sorter-class-name>
            org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter
        </exception-sorter-class-name>
        <no-tx-separate-pools/>
    </xa-datasource>

    <mbean code="org.jboss.resource.adapter.jdbc.vender.oracle.OracleXAExceptionFormatter"
           name="jboss.jca:service=OracleXAExceptionFormatter">
        <depends optional-attribute-name="TransactionManagerService">
            jboss:service=TransactionManager
        </depends>
    </mbean>
</datasources>
```

我们已经使用了 Oracle 的 thin（纯 java）驱动并假设数据库运行在主机 monkeymachine 上，数据库名（Oracle 术语为 SID）为 jboss。我们也假定你已经创建了具有足够权限的用户 jboss。在这个例子里，你可以使用 dba 权限。

```
SQL> connect / as sysdba
Connected.
SQL> create user jboss identified by password;
User created.
SQL> grant dba to jboss;
Grant succeeded.
```

现在把这个文件复制到 deploy 目录里。你应该得到下面的输出。

```
11:33:45,174 INFO  [WrapperDataSourceService] Bound connection factory for resource adapter
for ConnectionManager 'jboss.jca:name=XAOracleDS,service=DataSourceBinding to JNDI name
'java:XAOracleDS'
```

如果你从 JMX 控制台使用 JNDIView 服务，你应该可以看到 java:/XAOracleDS。

3.3. 测试 Oracle 数据源

我们将再次使用 CMP 例程来测试这个新的数据库连接。jbosscmp-jdbc.xml 文件应该包含下面的内容。

```
<jbosscmp-jdbc>
    <defaults>
        <datasource>java:/XAOracleDS</datasource>
        <datasource-mapping>Oracle9i</datasource-mapping>
    </defaults>
</jbosscmp-jdbc>
```

这里也有其他可用的 Oracle 类型映射。如果你在使用老的版本，请在 conf/standardjbosscmp-jdbc.xml 文件里找到正确的名字。

象以前一样部署这个应用程序，检查输出是否有错误，并在命令行运行 Oracle SQLPlus 来查看表是否已经被创建。

```
SQL> select table_name from user_tables;

TABLE_NAME
------------------------------
TEAMBEAN
LEAGUEBEAN
```

```
PLAYERBEAN
PLAYERBEAN_TEAMS_TEAM_1OFLZV8
```

附录 A. 进一步的信息来源

对于 JBoss 的进一步介绍，请参考《JBoss: A Developer's Notebook》（O'Reilly, 2005. Norman Richards, Sam Griffith）。

关于更多 JBoss 的高级主题方面的文档，请参考 http://www.redhat.com/docs/manuals/jboss 上的在线手册。

关于一般的 EJB 说明，请参考《Enterprise JavaBeans, 4th Edition》（O'Reilly, 2004. Richard Monson-Haefu1, Bill Burke, Sacha Labourey）。

要进一步学习 Hibernate，请参考《Java Persistence With Hibernate》（Manning, 2007. Christian Bauer, Gavin King）。

要学习完整的 JBoss Seam 框架，我们推荐《JBoss Seam: Simplicity And Power Beyond Java EE》（Prentice Hall, 2007. Michael Yuan, Thomas Heute）。