Getting Started with JBoss 4.0^

Release 5

[罗时飞，于 2006 年 07 月 19 日修改稿]

翻译说明：

全文由罗时飞翻译完成。无论在企业中，还是在开发者社群中，还是在 Open Source 社群中，JBoss 应用服务器在国内外的使用已经很广泛。其中，JBoss 3.2.x 系列已经大量部署在国内外现实场景中。JBoss 4.0.x 已经通过 J2EE 1.4 认证，而且其基代码也是来自于 JBoss 3.2.x，这些都使得 JBoss 应用服务器更具企业强度。JBoss 5.x 即将推出，这是 JBoss 3.x 之后架构变化最大的服务器版本（基于 POJO 微内核，兼容于 Java EE 5）。

[罗时飞，独立Java EE顾问，《JBoss Administration and Development》(Third Edition, 3.2.x Series)中文版（《JBoss管理与开发核心技术（第三版）》）译者，该书已经由电子工业出版社 2004 年 11 月出版发行。注意，开发者可以通过如下网站联系到他：http://www.open-v.com]

版权声明：

修改日志：

- 2004.10.5，提交翻译稿。
- 2004.11.5，针对社区反馈意见，添加了书签，以便于阅读。
- 2004.12.1，因英文版的更新（Release 2）而更新中文版。比如，内置了 JSTL、新增 Hibernate 一章、修改 MySQL 4.1.17/Oracle 10g JDBC 配置等内容。
- 2005.1.21，修正几个小错误。
- 2005.3.5，同步英文版的更新（Release 3）。如果存在错误，一切以英文版为准。
- 2005.6.2，同步英文版的更新（Release 4）。如果存在错误，一切以英文版为准。
- 2006.7.19，同步英文版的更新（Release 5）。如果存在错误，一切以英文版为准

目　录

序

1 目标读者

本书的目标就是，尽快使得用户能够将J2EE 1.4 应用部署并运行在JBoss 4.0.x上。在本书写作时，JBoss最新发布版为 4.0.4。因此，用户至少应该使用该版本，或者其后续版本。同时，本书使用了Sun提供的J2EE 1.4 Tutorial中（第 7 次更新）的实例（其具体网址位于，http://java.sun.com/j2ee/1.4/docs/tutorial/doc），以阐述JBoss中J2EE应用的部署和配置。当然，本书并不是J2EE教程，但还是从最基本的层面介绍了J2EE中的各种主题，因此如果用户刚接触J2EE，则本书还是很有参考价值的。如果用户打算使用JBoss运行上述J2EE Tutorial，则恭喜您，本书就是为您准备的。用户最好能够同时阅读这两份教程。

2 本书内容

第 1 章，将涉及 JBoss 4.0.x 应用服务器的下载、安装以及运行。然后，第 2 章大体上给出 JBoss 应用服务器的目录结构、主要配置文件以及服务。最后，第 3 章介绍了本书待使用的、J2EE Tutorial 中的代码。

接下来，第 4 章将研究，Sun J2EE Tutorial 中 Duke 银行应用如何在 JBoss 上部署。这将能够使开发者很快进入角色，即熟悉 JBoss 中的简单配置和部署操作。第 5 章，研究 Web 服务。本书将分两个步骤讨论。其一，如何将 Duke 银行应用中的 EJB 方法暴露为 Web 服务；其二，如何通过 Java 客户应用访问该 Web 服务。

再然后，第 6、7 章分别给出了 JMS 和消息驱动 Bean、容器管理持久化的介绍。

第 8 章，探讨数据库的配置。我们将一步一步教会开发者，如何配置、使用 MySQL 和 Oracle 数据库。作为全书的结束，第 9 章将研究 Hibernate，即如何在 JBoss 中使用它。注意，在 Hibernate 实例中也使用了 J2EE Tutorial 中的代码。

当然，本书只是对 JBoss 作了初步介绍，可谓是 JBoss 的冰山一角。一旦用户熟悉本书的内容后，您可以参阅《The JBoss 4 Application Server Guide》一书。它将带您深入掌握 JBoss 应用服务器。

第 1 章 入门

1.1 下载和安装 JBoss

在下载和安装 JBoss 之前，请开发者确认一下自己的机器是否安装了最新版的 JVM。为运行 JBoss 4.0，开发者必须提供 Java 1.4 或 Java 5 虚拟机。原则上，我们推荐使用 Java 5 虚拟机，因为它的运行性能非常卓越，而且能够同 EJB 3、Java EE 5 技术协同工作。单凭 JRE 就可以将 JBoss 运行起来，但是如果需要编译、运行 J2EE Tutorial 实例，则需要 JDK 的支持。在我们动身之前，请再次检查一下您是否安装了合适的 JDK，而且 JAVA_HOME 环境变量是否已经设置好。

用户可以从JBoss网站（http://www.jboss.org/downloads/index）免费下载到JBoss应用服务器。我们同时提供了二进制发布版、源代码发布版以及Java Web Start发布版。如果您刚接触JBoss，则建议使用Java Web Start发布版，因为这将提升您的体验。

其中，可用的二进制版本格式分别有.zip、.tar.gz 以及.bz2。JBoss 二进制发布版的具体内容与版本格式无关，用户需要根据各自的平台选择相应的二进制版本。在您下载完成 JBoss 后，将它解压到合适的机器位置上。请务必将所有的内容解压到命名为 jboss-4.0.4 的单一目录中。但如果用户使用了 JBoss 4.0.4 后续版本，则版本号会有所不同。有一点请注意，包含解压目录的完整路径（比如，Windows 操作系统中的 Program Files 目录）上不能够含有空格，因为这将导致错误的出现。

另外，我们还提供了可执行的.jar 版本。开发者从 JBoss AS 下载页面能够浏览到 Web Installer 链接，通过它能够启动 Java Web Start 进程。随后，在安装过程中，开发者可以依据安装提示完成 JBoss 的安装。默认时，开发者应该会使用 default 配置集合。最后，请为 JMX 安全性页面的 admin 账号设置一个密码。

1.2 启动服务器

首先，来看看如何运行 JBoss 服务器。用户可以在 JBoss 主安装目录的 bin 目录中找到若干个脚本文件。请执行 run 脚本（对于 Windows，则运行 run.bat；对于 Linux、OS X、UNIX 系统，则运行 run.sh）。其中，部署和启动 JBoss 组件的具体日志信息能够在运行 JBoss 的控制台浏览到。如下消息表明，JBoss 服务器成功运行（很明显，由于启动 JBoss 的时间和目标机器的配置不同，其给出的取值会不同）：

```
13:52:00,183 INFO  [Server] JBoss (MX MicroKernel)
[4.0.4.GA (build: CVSTag=JBoss_4_0_4_GA date=200605111311)] Started in 31s:941ms
```

用户可以通过Web浏览器验证JBoss应用服务器是否在运行，其HTTP监听端口为 8080（其中，必须保证在启动JBoss时，8080 端口并没有被其他应用或服务占用）。通过Web浏览器能够找到相关有用的JBoss资源（译者注：http://localhost:8080）。

1.3 JMX 控制台

通过http://localhost:8080/jmx-console[1]，即JMX控制台应用，用户能够浏览到服务器活动视图。图 1.1 给出了示例界面。

上述界面给出了 JBoss 管理控制台，它提供了构成 JBoss 服务器的 JMX MBean 原始视图。我们暂时可以不用理会控制台的任何内容，但是需要知道控制台能够提供运行中的 JBoss 应用服务器的大量信息。另外，通过它，用户能够修改、启动、停止 JBoss 组件。

比如，请找到 service=JNDIView 链接，然后单击。该特定 MBean 提供了如下服务内容，即能够浏览服务器中 JNDI 命名空间的结构信息。接下来，请在该 MBean 显示页面底端找到 list 操作，然后单击 invoke 按钮。invoke 操作将返回绑定到 JNDI 树中的当前名字列表，这对于获得 EJB 名字很有帮助，比如当 EJB 应用客户端不能够解析 EJB 名字时。

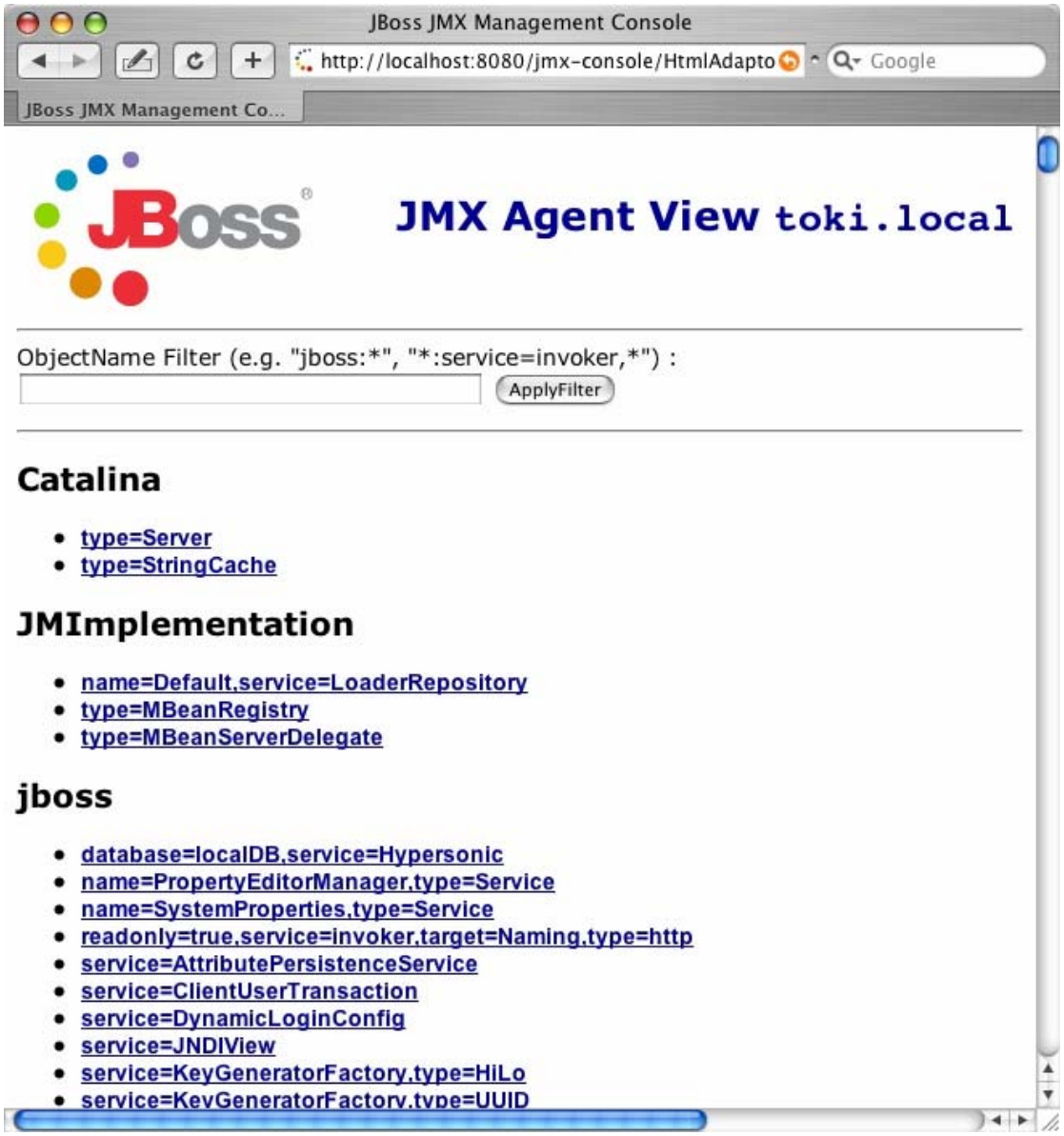

图 1-1 JMX 管理控制台 Web 应用视图

[1] 请注意，某些机器不能够正确解析localhost名。如果碰到这种情况，则应该使用本地回环地址，即 127.0.0.1。

类似地，用户也可以看看其他的 MBean 及其列举出的操作，然后试着修改一些配置属性，看看会发生什么事情。有一点请注意，即如果服务器一旦重启，从控制台所作的修改都将丢弃掉。因为重启 JBoss 时，系统将重新装载原始配置，因此用户可以大胆地尝试一切操作。

1.4 停止服务器

为了能够停止 JBoss 服务器，用户可以敲入 Ctrl-C，或者从 bin 目录运行 shutdown 脚本。甚至，用户还可以使用管理控制台（请在 jboss.system 部分找到 type=Server，然后调用 shutdown 操作。）。

1.5 运行为 Windows 服务

在实际部署场景，用户一般都不希望通过手工启动和停止 JBoss 应用服务器。因此，当目标机器启动时，用户都希望将它作为服务，或者后台应用运行。当然，具体配置细节取决于各个目标 OS 平台，并且都要求用户具有一些系统管理知识和 root 权限。

对于Linux或UNIX操作系统而言，用户需要安装启动脚本（或者通知系统管理员来完成此项任务）。其中，JBoss的bin目录中存在jboss_init_redhat.sh和jboss_init_suse.sh，这样两个脚本实例，用户也可以修改它们，以满足各自的具体需求。对于Windows操作系统而言，用户可以借助于实用工具，比如JavaService[2]，从而将JBoss安装成系统服务。

[2] 通过如下网址能够免费下载到JavaService：http://javaservice.objectweb.org/

第 2 章 JBoss 服务器--快速上手

2.1 服务器结构

至此，用户应该已完成了 JBoss 应用服务器的下载和首次运行。接下来，用户需要了解安装后的 JBoss 目录结构及其相应的内容。粗略看后，用户会发现其存在很多目录，因此很明显，用户可能会无从下手，并且不知道应该忽略哪些内容，从而不影响用户对 JBoss 应用服务器的体验。为解决这个问题，本章将深入服务器目录结构、主要配置文件的位置信息、日志文件、部署问题，等等。对于用户而言，本章的内容很有意义，因为这将帮助用户理解 JBoss 服务架构。在阅读完本章内容后，用户将能够完成 J2EE 应用的部署。

2.1.1 主目录

将二进制发布版解压到 jboss-4.0.4 目录。该目录包含如下 5 个子目录：

- **bin**：含有启动、停止以及其他系统相关脚本。在前面，本书已经讨论过启动 JBoss 应用服务器的 run 脚本。
- **client**：存储供 Java 客户应用或者外部 Web 容器使用的配置文件和 JAR 文件。用户可以使用所需要的具体存档，或者仅仅使用 jbossall-client.jar。
- **docs**：含有 JBoss 引用的 XML DTD 文件（当然，还包括 JBoss 具体配置文件）。同时，还存在 JCA（Java Connetor Architecture，Java 连接器架构）实例配置文件，供设置不同数据库的数据源使用（比如 MySQL、Oracle、Postgres）。
- **lib**：包含运行 JBoss 微内核所需的 JAR 文件。请注意，不要往该目录添加用户自身的任何 JAR 文件。
- **server**：包含的各个子目录都是不同的服务器配置。通过往 run 脚本后添加-c <config name>参数便能够指定不同的配置。接下来，来看看 default 服务器配置。

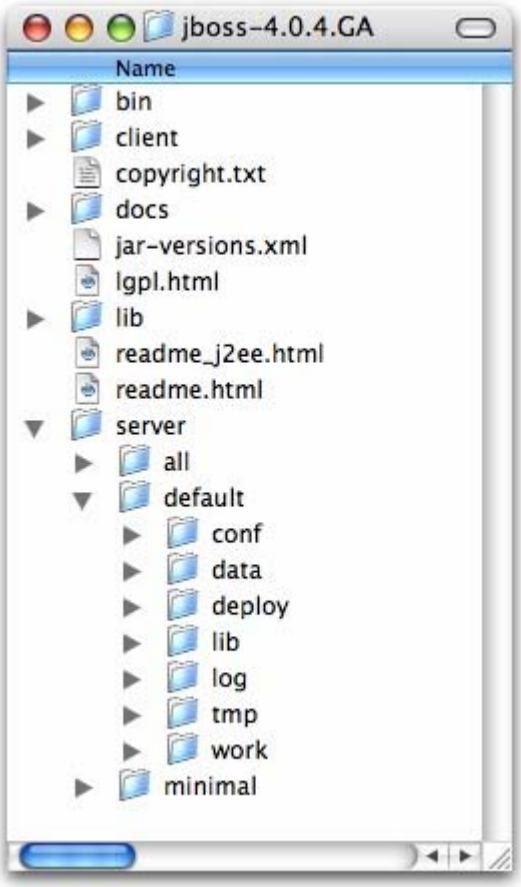

图 2-1 JBoss 目录结构

2.1.2 服务器配置

从根本上考虑，JBoss 架构是由 JMX MBean 服务器、微内核、一套可插入式组件服务以及 MBean 构成的。这种架构使得，集成不同的配置变得更加简单，并且能够很灵活地满足用户的各自需求。用户不再需要一次性运行重量级的应用服务器。同时，用户可以删除不再需要使用的组件(这将从很大程度上减少服务器的启动时间)，并且通过开发自己的 MBean 还能够集成其他服务到 JBoss 中。当然，如果是运行标准 J2EE 应用，则不用理会这些自定义工作。用户所需要的一切服务，JBoss 发布版都包括了。另外，为使用 JBoss，用户也不用掌握 JMX 的更详细技术细节，但是如果用户知道 JBoss 是将这种基于 JMX 的架构作为其核心的，则对于开发基于 JBoss 的 J2EE 而言，更有帮助。

开发者在 server 目录下能够找到若干服务器配置实例。如果 JBos 是通过安装界面完成安装的，则仅仅能看到 default 配置。在安装过程中，EJB3、EJB 3 群集、完全 J2EE 1.4 等特性可供安装者选择，最终这些特性都将配置在 default 中。如果开发者下载了二进制或源代码（构建）版本，则 server 目录下存在 3 个服务器实例配置：all、default 以及 minimal，它们各自提供了不同的服务集合。很显然，如果启动 JBoss 服务器时没有指定其他配置，则将使用 **default** 配置，这也是本书在第 1 章使用的服务器配置。各个配置的具体内容如下：

- **minimal**：这是启动 JBoss 服务器所要求的最低配置。minimal 配置将启动日志服务、JNDI 服务器以及 URL 部署扫描器，以找到待部署的（新）应用。对于那些不需要使用任何其他 J2EE 技术，而只是使用自定义服务的场合而言，则这种 JMX/JBoss

配置最适合。它仅仅是服务器，而不包含 Web 容器、不提供 EJB 和 JMS 支持。

- **default**：默认配置，它含有大部分 J2EE 应用所需的标准服务。但是，它不含有 JAXR 服务、IIOP 服务、或者其他任何群集服务。
- **all**：提供了所有可用的服务。它包含 RMI/IIOP 和群集服务，default 配置中没有提供群集服务。

用户也可以添加自身的服务器配置。最佳做法是，拷贝最接近用户需求的现有配置，然后修改其具体内容。比如，如果用户不需要使用消息服务，则只需要拷贝 default 目录，并重新命名为 myconfig，然后删除 jms 子目录。最后，启动 myconfig 配置。

```
run -c myconfig
```

包含运行 JBoss 的服务器配置是服务器根路径。它含有特定配置中所有的代码和配置信息。比如，日志输出将存储在那儿，应用也将部署在那儿。接下来，本章来看看 default 服务器配置目录的具体内容。如果用户还没有运行 JBoss 服务器，而请运行一下吧，因为初次运行后，JBoss 将创建若干个子目录。

- **conf**：含有指定 JBoss 核心服务的 jboss-service.xml 文件。同时，还包括核心服务的其他配置文件。
- **data**：这一目录存储持久化数据，即使服务器发生重启其中的数据也不会丢失。许多 JBoss 服务将数据存储在这里，比如 Hypersonic 数据库实例。
- **deploy**：用户将应用代码（JAR\WAR\EAR 文件）部署在此处。同时，deploy 目录也用于热部署服务（即，那些能够从运行服务器动态添加或删除的服务）和部署 JCA 资源适配器。因此，用户能够在 deploy 目录看到大量的配置文件。尤其是，用户能够看到 JMX 控制台应用（未打包的 WAR 文件），本书前面讨论过。JBoss 服务器将定期扫描该目录，从而查找是否有组件更新或修改，从而自动完成组件的重新部署。本书后续章节将详细阐述部署细节。
- **lib**：服务器配置所需的 JAR 文件。用户可以添加自身的库文件，比如 JDBC 驱动，等等。
- **log**：日志信息将存储到该目录。JBoss 使用 Jakarta Log4j 包作为其日志功能。同时，用户可以在应用中直接使用 Log4j 日志记录功能。
- **tmp**：供部署器临时存储未打包应用使用，也可以作为其他用途。
- **work**：供 Tomcat 编译 JSP 使用。

其中，data、log、tmp、work 目录是 JBoss 创建的。如果用户没有启动过 JBoss 服务器，则这些目录不会被创建。

本书既然提到了 JBoss 中的热部署服务主题，接下来在探讨服务器配置问题前先来看看实际例子。如果还没有启动 JBoss，则请运行它。然后，请再次查看 deploy 目录（用户必须保证运行了 default 配置），然后删除 mail-service.xml 文件。通过运行 JBoss 服务器的控制台能够浏览到如下信息：

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

然后，再次将 mail-service.xml 文件放回原处，用户将通过控制台再次发现 JBoss 重新部署了该服务。所以，这就是 JBoss 的热部署。

2.2 基本安装问题

至此，本书已经阐述了 JBoss 服务器的结构。接下来，带领用户看看其中的主要配置文件，并看看它们各自的用途。请注意，这里的目录路径都是基于 default 目录给出的。

2.2.1 核心服务

当 JBoss 服务器启动时，首先会启动 conf/jboss-service.xml 文件指定的核心服务。如果用户通过编辑器打开它，将会看到其包括了各种服务，其中包括日志、安全性、JNDI（还有本书前面讨论过的 JNDIView 服务）。将如下 JNDIView 服务入口注释掉：

```
<!-- 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>
-->
```

如果重启 JBoss，则 JNDIView 服务将不再出现在管理控制的列表中。实际场合中，用户很少会修改该文件。虽然通过 conf/jboss-service.xml 文件能够添加其他 MBean 服务，但是更好的办法是，将单独的配置文件放置在 deploy 目录中，因为这将使得用户的服务具有热部署能力。

2.2.2 日志服务

本书提到，Log4j是JBoss使用的日志功能包。如果用户还不熟悉Log4j包，并且希望在应用中使用它，则请到Jakarta网站，http://jakarta.apache.org/log4j/。通过conf/log4j.xml文件能够控制JBoss的日志功能。该文件定义了一套Appender、指定了日志文件、具体消息Category类型的存储、消息格式以及消息的过滤级别。默认时，JBoss会同时在控制台和日志文件（位于log/server.log文件中）中生成输出信息。

一共存在 5 个基本的日志级别：DEBUG、INFO、WARN、ERROR 以及 FATAL。其中，控制台的日志入口（threshold）为 INFO，即用户通过控制台能够浏览到提示信息、警告信息、错误信息，但是调试信息查看不到。相比之下，JBoss 并没有为 server.log 文件设置任何入口，因此所有生成的消息将记录到 server.log 文件中。如果 JBoss 运行过程中出现了错误，则通过控制台可能找不到用户有用的信息，因此建议通过 server.log 文件查看是否有调试信息可供解决问题所用。然而，请注意，通过调整日志入口能够在控制台查看到调试信息，但是这并没有保证所有的 JBoss 消息都将记录到 server.log 文件中。因此，用户还需要为单个的 Category 设置不同的日志级别。比如，log4j.xml 给出了如下 Category。

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

它将所有的 JBoss 相关类的日志级别限制到 INFO，即同那些含有具体日志级别的 Category 隔离开。如果将 INFO 更换为 DEBUG，则将生成更详细的日志输出。

本书将再介绍另外一个实例：将 CMP 引擎的日志输出级别设置为 DEBUG，并将其定位到 cmp.log 文件中，从而可供用户分析生成的 SQL 命令。因此，需要添加如下代码到 log4j.xml 文件中。

```
<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>
```

这将创建新的文件 Appender，并指定将它用于 org.jboss.ejb.plugins.cmp 包的 Logger（或 Category）。第 7 章讨论 CMP 时，如果使用该实例将大有益处。

由于文件 Appender 设置为每日创建新的日志文件，因此 JBoss 不会每次启动服务器时创建新的日志文件，而且也不会总是将日志信息记录到单个文件中。当前的日志文件为 cmp.log，更早的日志文件名是通过将具体日期添加到 server 而生成的。用户应该还注意到，log 目录包含了 Web 容器生成的 HTTP 请求日志。

2.2.3 安全性服务

安全性域信息存储在login-config.xml文件中，其包含了许多安全性域定义。各个安全性域指定了许多JAAS[3]登陆模块，供安全性域认证使用。当用户需要在应用中使用安全性时，需要在JBoss特定部署描述符jboss.xml或jboss-web.xml中指定待使用的安全性域名。本节将快

[3] Java认证和授权服务。JBoss使用JAAS提供插入式认证模块。用户可以使用现有的认证模块，或者可以开发更适合自身需求的认证模块。

速地带领用户分析如何保护随JBoss发布的JMX控制台和Web控制台应用。

本书在 1.3 节讨论过 JMX 控制台。通过 JMX 控制台基本上能够控制 JBoss 服务器的各个方面，因此保护该控制台很重要，至少需要通过密码来保护它。否则，任何远程用户将能够完全控制用户的 JBoss 服务器。为实现此目的，本文将安全性域添加给 JMX 控制台应用。通过 server/default/deploy/jmx-console.war/WEB-INF/目录能够找到待修改的 JMX 控制台文件，即 jboss-web.xml。将 jboss-web.xml 中 security-domain 的注释去掉，具体如下。

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

这将设置 Web 应用待使用的安全性域，但还未确定 Web 应用应该使用的安全性策略。待保护的 URL 是什么，相应的访问角色又是哪些？为配置这些内容，用户需要在同一目录中找到 web.xml 文件，然后将 security-constraint 的注释去掉。其中，该安全性约束要求登陆用户必须具有 JBossAdmin 角色。

```
<!--
    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
    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>
```

太好了，但是用户名和密码来自哪里呢？不错，它们来自 jmx-console 安全性域。通过 conf/login-config.xml 文件能够看到。

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

上述配置使用了基于简单文件的安全性策略。其中，登陆 JMX 控制台应用的用户名和密码存储在 jmx-console-users.properties 中，并且以“username=password”形式给出。为了将用户添加到 JBossAdmin 角色中，需要将“username=rolename”形式给出的用户和角色映射关系给出在 jmx-console-roles.properties 文件中。现存的文件创建了 admin 用户，其密码为 admin。用户可能删除该用户，或者更改其密码，使得 JMX 控制台应用更安全。

当更新了 web.xml 时，JBoss 会重新部署 JMX 控制台应用。用户可以通过服务器控制台检查 JBoss 保存了对 web.xml 所作的修改。如果用户正确地配置了上述各个任务，并且重新部署了 JMX 控制台应用，则下次访问它时，JBoss 会要求用户提供用户名和密码。

JMX 控制台应用不是 JBoss 唯一提供的、基于 Web 的管理界面。JBoss 还提供了另一管理应用，即 Web 控制台（参考附录 A）。尽管 Web 控制台是基于 Java Applet 形式给出的，但是对应的 Web 应用还是可以类似于 JMX 控制台的方式来保护它。其中，Web 控制台应用位于 default/deploy/management/web-console.war。注意，这同 JMX 控制器应用有所区别，因为 JMX 控制器应用是以目录形式展开的。因此，编辑 web-console.war WAR 存档更费力些。

2.2.4 其他服务

deploy 目录放置的服务不是核心服务，但具有热部署能力。用户可以通过 XML 描述符文件（*-service.xml）或 JBoss 服务存档（SAR）文件给出服务。SAR 同时含有 XML 描述符和服务所要求的其他资源（比如，类、JAR 库文件以及其他存档），而且 SAR 是以单个存档文件给出的。

本节内容将研究 default 配置中的 deploy 目录，并给出相应的解释。当然，这里给出的内容使得用户能够从整体上把握 JBoss 应用服务器，因此如果用户希望了解有关现有 MBean 组件的更多内容，则不要错过本节内容。通过 default/deploy 目录，用户能够发现下列文件和子目录。

- **bsh-deployer**：将 BeanShell 脚本部署成 JBoss 服务。
- **cache-invalidation-service.xml**：允许借助于 JMS，而实现对 EJB 缓存的控制。
- **client-deployer-service.xml**：部署 J2EE 应用客户。
- **ear-deployer.xml**：部署 J2EE EAR 应用。

- **ejb-deployer.xml**：部署 J2EE EJB 应用。
- **hsqldb-ds.xml**：设置嵌入式 Hypersonic 数据库服务，并将其作为默认数据源。
- **http-invoker.sar**：通过 RMI/HTTP 方式访问到 MBean 和 EJB。
- **jboss-aop.deployer**：提供 AspectManagerService，并部署 JBoss AOP 应用。
- **jboss**-hibernate.deployer：部署 Hibernate 存档（HAR 文件）。
- **jboss-local-jdbc.rar** 和 **jboss-xa-jdbc.rar**：集成 JDBC 驱动的 JCA 资源适配器，它们分别支持 DataSource 和 XADataSource。但是，这并没有提供专有 JCA 实现。
- **jboss-ws4ee.sar**：提供 J2EE Web 服务支持。
- **jbossjca-service.xml**：JBoss JCA 实现，使得在 JBoss 中部署 JCA 资源适配器成为可能。
- **jbossweb-tomcat50-sar**：含有嵌入式 Tomcat 服务的展开 SAR 文件。它为 JBoss 提供了标准的 Web 容器。
- **jms**：将 JMS 相关的服务聚集在一起，并放置在 jms 目录中。
- **hsqldb-jdbc-state-service.xml**：使用 HSQLDB 管理状态。
- **hsqldb-jdbc2-service.xml**：使用嵌入式 HSQL 数据库实现缓存和持久化。它还包含了 JMS 实现的核心服务，即 DestinationManager MBean。
- **jbossmq-destinations-service.xml**：供 JBoss 测试套件使用的 JMS Topic 和 Queue。
- **jbossmq-service.xml**：JMS 其他服务，包括拦截器配置。
- **jms-ds.xml**：将 JBoss 消息实现作为默认 JMS 提供商。并且，它还提供 JCA 配置信息，以供集成 JBoss JCA 和 JMS 资源适配器使用。
- **jms-ra.rar**：资源适配器，供 JCA 处理 JMS 连接工厂使用。
- **jbossmq-httpil.sar**：提供 JMS 调用层，从而实现 HTTP 方式使用 JMS。
- jvm-il-service.xml：配置本地 JMS 传输调用层，供本地 JVM 使用 JMS。
- uil2-service.xml：配置 JMS 版本 2 统一调用层。这是一种可靠的、自定义的、基于 Socket 的传输方式。推荐在不同 JVM 间使用它。
- **jmx-console.war**：JMX 控制台应用。前面讨论过。
- **jmx-invoker-server.xml**：为远程访问 JMX MBean 服务器提供支持。
- mail-ra.rar：为 JavaMail 提供资源适配器。
- **mail-service.xml**：允许应用和服务在 JBoss 中使用 JavaMail。请注意，邮件服务器相关信息必须由用户提供。
- **management**：含有可更换管理服务的子目录。其中，包含有改进的 Web 控制台。
- **monitoring-service.xml**：配置警告监听器，比如控制台监听器、E_mail 监听器，等等。
- **properties-service.xml**：设置 JVM 的全局系统属性（由 System.getProperties 返回）。
- **schedule-manager-service.xml** 和 **scheduler-service.xml**：定时任务服务。
- **sqlexception-service.xml**：为 JDBC 驱动提供标识一般性 SQL 异常。
- **uuid-key-generator.sar**：生成唯一的、基于 UUID 的键。

all 配置提供了其他配置没有提供的其他服务，用户可以将这些服务集成到各自的服务器配置中。具体如下：

- **cluster-service.xml**：群集服务，包括 JGroups 集成服务、HA-JNDI、有状态会话 Bean 复制、CMP2 缓存有效性服务。
- **deploy-hasingleton-service.xml**：HASingletonDeployer MBean。用于确保群集中只有单个节点在 deploy-hasingleton 目录部署了服务。

- **deploy.last/farm-service.xml**：farm 群集部署服务。用于确保它在所有其他服务部署之后才部署其本身。
- **ebxmlrr-service.xml**：JAXR 注册服务实现。
- **iiop-service.xml**：实现对 CORBA、IIOP 的支持。
- **jbossha-httpsession.sar**：遗留的 HTTP 会话复制服务。
- **remoting-service.xml**：还处于试验中的下一代分离式 Invoker 框架。
- **snmp-adaptor.sar**：将 JMX 通知转换成 SNMP 陷阱。
- **tc5-cluster-service.xml**：用于新的 HTTP 复制服务的 TressCache 配置。

有关上述各种服务更深入、全面的介绍，请用户参考《The JBoss 4 Application Server Guide》。该书还提供了服务器内核的完整信息以及服务实现（比如，JTA 和 J2EE 连接器架构）。

2.3 Web 容器—Tomcat

JBoss 4.0.0 将 Tomcat 5.5 作为其默认 Web 容器。嵌入的 Tomcat 服务是通过展开的 SAR，即 deploy/jbossweb-tomat55.sar，给出的。Tomcat 所需要的所有 JAR 文件都能在这里找到，其中还包括为 Web 应用提供默认配置集合的 web.xml 文件。如果用户熟悉 Tomcat 的配置，则可以查看 server.xml，其含有标准 Tomcat 格式化配置信息。它包含如下内容：设置 HTTP 连接器，并将 8080 端口作为默认监听端口；设置 AJP 连接器，并使用 8009 端口（比如，将 Apache 作为 Web 服务器）；给出了如何配置 SSL 连接器的实例（默认时，并没有启用）。

除非是高级用户，否则不要修改任何内容。如果用户以前使用过单独的 Tomcat 服务器，则应该注意到，它同这里的嵌入式服务还是存在差别的。JBoss 掌管了 Tomcat，因此用户根本不用去访问 Tomcat 目录。通过将 Web 应用放置在 JBoss deploy 目录中，便实现了它们的部署。同时，来自 Tomcat 的输出日志（包括内部访问和访问日志）被保存在 JBoss log 目录中。

第 3 章 关于实例应用

3.1 J2EE Tutorial

本书将使用Sun J2EE Tutorial中的应用实例，即Duke银行应用。通过如下网站能够找到J2EE Tutorial：http://java.sun.com/j2ee/1.4/docs/tutorial/doc/。用户通过J2EE Tutorial能够获得一些入门资料。http://java.sun.com/j2ee/1.4/download.html/#tutorial提供了实例代码的下载。

本章将研究在JBoss中如何运行Duke应用。并且在扩充Duke应用的同时，还需要提供JBoss特定的配置信息和部署描述符。用户还需要下载本书附带的那些代码。如果还没有下载，则赶紧去JBoss文档页面下载，http://www.jboss.org/docs/index。

该教程使用了Apache Ant（http://ant.apache.org）构建工具，用户应该下载并安装它。我们建议使用Ant 1.6.2 及后续版本。如今，Ant广泛应用于Java项目中。因此，如果用户还不熟悉它，则赶紧补上这一课。其中，Ant使用的默认构建文件名是build.xml，它含有供项目使用的一套目标（target），以自动完成构建任务。一般情况下，用户只需要在含有build.xml文件的目录中执行ant命令，这将使得默认目标会完成所需工作。

J2EE Tutorial 将解释如何在 J2EE SDK RI 服务器中运行 Duke 银行应用。请记住，本书的目标是能够在 JBoss 中运行它。

3.2 区别

J2EE 技术的设计初衷是，应用代码独立于所部署并运行的应用服务器。其中，用于 EJB 和 Web 应用的 ejb-jar.xml 和 web.xml 部署描述符是 J2EE 规定的标准描述符，因此在不同 J2EE 容器之间不用更换它们。然而，为了将应用移植到 JBoss，用户还是需要注意一些事情的。尤其是，用户需要提供 JBoss 特定描述符，并且确保数据库脚本能够正常工作。

3.2.1 容器相关部署描述符

容器相关信息通常是放置在外部 XML 描述符中，这些描述符将应用使用的逻辑信息（比如，JNDI 名字和安全性角色名）映射到服务器使用的实际值。JBoss 为应用的 EJB 和 Web 模块分别使用了 jboss.xml 和 jboss-web.xml 描述符。同时，考虑到 Java 客户应用，JBoss 还提供了它们的客户版本，以同 J2EE application-client.xml 描述符一块使用。如果实体 Bean 还使用了 CMP，则通过 jbosscmp-jdbc.xml 文件还能够配置 JBoss 持久化引擎。

3.2.2 数据库变更

J2EE SDK 自带了 Cloudscape 数据库，这也是 J2EE Tutorial 使用的数据库。本书将使用 JBoss 自带的嵌入式数据库，Hypersonic。

实际情况下，将应用直接移植到不同的数据库并不是简单过程，尤其是应用使用了数据库的特定功能，比如序列号（Sequence）、存储过程、非标准 SQL。对于简单应用，移植工作相对简单。第 4 章内容在讨论 Duke 银行应用时，用户能够看到只需要对数据库脚本做少许的语法修改就能够满足要求。

第 8 章内容，将讨论如何配置 JBoss，以使用不同的数据库。

3.2.3 安全性配置

J2EE 定义了应用中指定安全性约束的机制，但并没有给出具体服务器是如何实现和配置认证和访问控制机制（译者注：授权）的。因此，正如本书前面讨论的一样，JBoss 使用了 JAAS 以提供集成不同认证技术的可插入方式，供应用的认证和授权使用。同时，JBoss 还提供了一套标准模块，即基于文件、数据库、LDAP 的安全性机制。其中，基于文件的方式最为简单，这也是本书使用的方式。

3.3 企业应用中的 J2EE

本书给出的实例仅仅为用户提供熟悉和运行 JBoss 使用。因此，这里的应用绝对不是指导用户如何开发 J2EE 应用而言的，更何况这个主题在业界也没有统一的定论。比如，很多开发者反对使用 EJB，尤其是实体 Bean；再比如，BMP 的使用也是如此。当然，不同的 Web 技术（可以认为，并不是每个人都喜欢 JSP）的使用也是一个争执不休的主题，并且存在大量的 Model-2 框架。Struts 应用较早，而且知名度最高，但是还有很有多批评的论调。

如果用户刚启动某项目，则最好能够先评估一下现有的 Open Source 项目，然后看看它们各自的结构，最终选择适合于项目需求的框架。

最后，我们也希望用户能够意识到 JBoss 的博大精深，本书给出的仅仅是冰山一角。JBoss 还是一个持续发展的项目，对于未来，我们有很多宏伟的目标和计划。因此，请关注 JBoss 的最新动向，哪怕是用户已经打算将所有的产品运行于稳定的 JBoss 4.0.x 之上。

第 4 章 Duke 银行应用

至此，用户已经运行了 JBoss。本书将使用 J2EE Tutorial 中的 Duke 银行应用，使得它能够运行在 JBoss 中。Duke 银行应用演示了如何使用若干 J2EE 技术，以实现简单的、在线银行应用。它使用 EJB 和 Web 组件（JSP 和 Servlet），并且将数据库作为持久源。其中，通过 BMP 管理持久化，这些 BMP 中含有 SQL 语句，从而能够操纵 DB。

本书不会把重点放在 Duke 银行应用的功能，或者其实现上。而只是关注如何通过一步一步的指导，使得我们能够构建和运行基于 JBoss 的 Duke 银行应用。

4.1 构建应用

用户应该准备好 J2EE 1.4 Tutorial，因为它含有 Duke 银行应用。首先，来看看构建和部署 Duke 银行应用。然后，再来仔细分析 JBoss 相关技术细节。

4.1.1 准备文件

本文档发布的.zip 文件中包括了 jbossj2ee-src.zip。下载后，将其解压到 j2eetutorial14 目录，即添加到现有的 J2EE Tutorial 文件中。所有的 Duke 银行应用代码位于 examples/bank 子目录中。如果正常解压 jbossj2ee-src.zip，用户将看到 jboss-build.xml 文件。这是本书用于 JBoss 版的 Duke 银行应用的 Ant 构建脚本。jbossj2ee-src.zip 并没有覆盖现有的 build.xml 文件，而是创建了 jboss-build.xml 文件。因此，需要使用 ant –f jboss-build.xml 执行 Ant 命令。

在构建 Duke 银行应用之前，用户需要设置如下信息。即，编辑 j2eetutorial14 目录中的 jboss-build.properties 文件，以定位 JBoss 4.0.4 的安装目录。其中，需要将 jboss.home 属性定位到 JBoss 4.0.4 安装的完整路径。如果将 JBoss 4.0.4 解压到 Windows 中的 C:/，则用户需要设置如下信息。

```
# Set the path to the JBoss directory containing the JBoss application server
# (This is the one containing directories like "bin", "client" etc.)
jboss.home=C:/jboss-4.0.4
```

4.1.2 编译 Java 源文件

打开命令行，然后转到 bank 目录。所有的 Ant 构建命令都是在此执行的。编译工作非常直观，即敲入如下命令，以调用 compile 目标。

```
ant -f jboss-build.xml compile
```

如果没有出现错误，则用户将发现新创建的 build 目录，其中包含了.class 文件。

4.1.3 打包 EJB

该应用含有一 EJB jar，即 bank-ejb.jar。bank-ejb.jar 含有代码和描述符（ejb-jar.xml 和 jboss.xml），以用于实体 Bean 和与客户交互的控制会话 Bean。Ant 目标 package-ejb 将会把创建的 bank-ejb.jar 存放到 jar 目录。

```
ant -f jboss-build.xml package-ejb
```

4.1.4 打包 WAR 文件

下个 Ant 目标是提供前端支持的 Web 应用，从而实现用户与业务组件（EJB）的交互。其中，src/web 目录含有的 Web 资源（JSP、图片、等等）原封不动地添加到存档中。Ant war 任务也能够添加 WEB-INF 目录，Web 浏览器并不能直接与该目录中的内容进行交互，但是其内容是 Web 应用的组成部分。Web 应用具体包含的内容有部署描述符（web.xml 和 jboss-web.xml）、类文件（比如，Servlet 和 EJB 接口）、其他 WAR 要求的库和 JSP 标签库描述符。Ant 目标 package-web 能够构建 Web 客户，即 WAR 文件。

```
ant -f jboss-build.xml package-web
```

4.1.5 打包 Java 客户

除了 Web 界面外，还存在单独运行的 Java 客户应用，用于管理顾客和账号。使用 Ant 目标 package-client 如下：

```
ant -f jboss-build.xml package-client
```

其生成的 app-client.jar 文件含有 application-client.xml 和 jboss-client.xml 描述符。另外，还包括 jndi.properties 文件。同时，客户 JAR 文件也将包含在 EAR 文件中，以作为附加的模块。

4.1.6 集成 EAR

EAR 文件是完整的应用，即含有 3 个 EJB 模块和 1 个 Web 模块。而且，它还必须包含 application.xml 描述符。当然，可以单独部署 EJB 和 Web 应用模块，但是 EAR 提供了单一的、便于操作的部署单元。Ant 目标 assemble-app 将创建 JBossDukesBank.ear。

```
ant -f jboss-build.xml assemble-app
```

4.1.7 数据库

在部署应用之前，用户需要完成其数据库配置。如果应用中使用了 CMP，则可以通过配置 CMP 引擎，使得在部署时自动创建表。否则，用户需要创建 SQL 脚本。对于存在初始化数据的 DB 而言，手工创建表也是较好的办法。

4.1.7.1 启用 HSQL MBean 和 TCP/IP 连接

HSQL 数据库能够以如下两种方式运行：进程内或客户-服务器（HSQL 文档称之为，服务器模式）。由于用户需要使用能够连接到 DB 的工具，以运行 SQL 脚本，因此必须保证数据库以客户-服务器模式运行，并能接受 TCP/IP 连接请求。默认时，最新的 JBoss 发布版将客户-服务器模式禁用了，这主要考虑到安全性要素，因为如果默认登陆账号没有修改将导致安全性问题。因此，用户需要打开设置了默认数据源的 deploy/hsqldb-ds.xml 文件。在文件的开始部分，能够看到 connection-url 元素。其取值必须是 jdbc:hsqldb:hsql://localhost:1701，而且其他 connection-url 元素值必须注释掉。

```
<!-- The jndi name of the DataSource, it is prefixed with java:/ -->
<!-- Datasources are not available outside the virtual machine -->
<jndi-name>DefaultDS</jndi-name>

<!-- for tcp connection, allowing other processes to use the hsqldb
database. This requires the org.jboss.jdbc.HypersonicDatabase mbean. -->
<connection-url>jdbc:hsqldb:hsql://localhost:1701</connection-url>

<!-- for totally in-memory db, not saved when jboss stops.
The org.jboss.jdbc.HypersonicDatabase mbean is unnecessary
<connection-url>jdbc:hsqldb:.</connection-url>
-->
<!-- for in-process db with file store, saved when jboss stops. The
org.jboss.jdbc.HypersonicDatabase is unnecessary

<connection-url>jdbc:hsqldb:${jboss.server.data.dir}/hypersonic/localDB
</connection-url>
-->
```

然后，定位到文件的底端部分，用户应该可以找到 Hypersonic 服务的 MBean 声明。

```
<mbean code="org.jboss.jdbc.HypersonicDatabase" name="jboss:service=Hypersonic">
    <attribute name="Port">1701</attribute>
    <attribute name="Silent">true</attribute>
    <attribute name="Database">default</attribute>
    <attribute name="Trace">false</attribute>
```

```
    <attribute name="No_system_exit">true</attribute>
</mbean>
```

用户必须确保，这部分内容没有被注释掉，因此 JBoss 才能够正确地启动 DB。

4.1.7.2 创建数据库模式

用户通过 sql 目录能够找到适合 HSQL 的 SQL 脚本。其中，jboss-build.xml 中的数据库任务能够同 HSQL 数据库进行交互。如果还未运行 JBoss，则请立即启动它吧，这样才能使用数据库。

首先，可以使用 Ant 目标 db-create-table 创建所需的表。

```
ant -f jboss-build.xml db-create-table
```

然后，运行 db-insert 目标，以初始化所需的数据。

```
ant -f jboss-build.xml db-insert
```

最后，如果上述过程很顺利，则使用 Ant 目标 db-list 能够浏览到部分数据。其中，db-list 目标列举出了某特定账号的交易信息。

```
ant -f jboss-build.xml db-list
```

4.1.7.3 HSQL 数据库管理工具

另外，如果用户启动 JMX 控制台 Web 应用，能够在域名 jboss 下看到 service=Hypersonic 链接。如果看不到，则请用户务必确保 4.1.7.1 节中确实生效了 Hypersonic 服务。

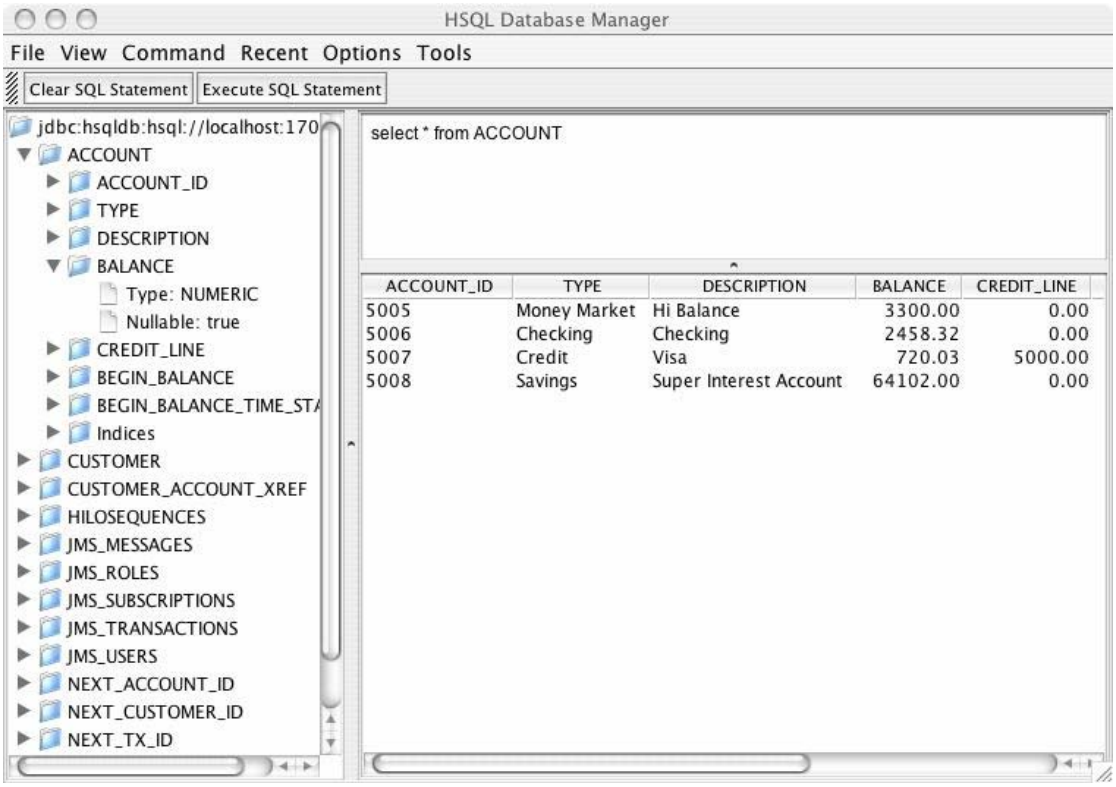

图 4-1 HSQL 数据库管理器

通过单击上述链接，用户能够使用到 Hypersonic MBean 服务提供的 HSQL 数据库管理功能。具体步骤如下：将屏幕滚动到页面底端，然后单击用于 startDatabaseManager()操作的 invoke 按钮，进而启动了 HSQL 管理器，即用户能够直接操作数据库的、基于图形用户界面的 Java 应用。

4.1.8 部署应用

部署基于 JBoss 的应用很简单，用户只需要将 EAR 文件拷贝到 deploy 目录即可。通过 Ant 目标 deploy 能够完成应用的部署工作。

```
ant -f jboss-build.xml deploy
```

如下给出了示例（作了部分裁减，而且同用户所看到的实际输出会有所差别）。

```
18:07:53,923 INFO  [EARDeployer] Init J2EE application:
file:/private/tmp/jboss-4.0.4/server/default/deploy/JBossDukesBank.ear
18:07:55,024 INFO  [EjbModule] Deploying CustomerBean
18:07:55,103 INFO  [EjbModule] Deploying AccountBean
18:07:55,142 INFO  [EjbModule] Deploying TxBean
18:07:55,403 INFO  [EjbModule] Deploying NextIdBean
18:07:55,439 INFO  [EjbModule] Deploying AccountControllerBean
18:07:55,478 INFO  [EjbModule] Deploying CustomerControllerBean
```

```
18:07:55,503 INFO  [EjbModule] Deploying TxControllerBean
18:07:56,950 INFO  [EJBDeployer] Deployed: file:/private/tmp/jboss-4.0.4/server/default/t
mp/deploy/tmp15097JBossDukesBank.ear-contents/bank-ejb.jar
18:07:57,267 INFO  [TomcatDeployer] deploy, ctxPath=/bank, warUrl=file:/private/tmp/jboss
-4.0.4/server/default/tmp/deploy/tmp15097JBossDukesBank.ear-contents/web-client.war/
18:08:00,784 INFO  [EARDeployer] Started J2EE application: file:/private/tmp/jboss-4.0.4/
server/default/deploy/JBossDukesBank.ear
```

如果存在错误或异常信息，则请注意错误信息，并确认导致错误的原因（比如，部署某特定 EJB 时，或 Web 应用等等）。请检查 EAR、WAR、EJB jar 文件的完整性，比如所有所需的组件（类、部署描述符等等）是否都存在。

如果应用已经部署，则用户可以放心地再次部署它。如果需要卸载应用，则只需要将相应的存档从 deploy 目录删除即可。当然，JBoss 服务器始终不需要重启。如果一切都正常，并无异常抛出，则通过 Web 浏览器打开如下 URL：

http://localhost:8080/bank/main

用户将浏览到应用登陆界面。同 J2EE Tutorial 给出的一样，用户通过顾客 ID（200）和密码（j2ee）能够登陆进入系统，见图 4-2。如果出现错误，则请检查 4.1.7.1 节设置的数据库是否正常工作。尤其是，需要确保 connection-url 没问题。然后，需要确保正确地初始化了数据库。

当然，用户使用 Ant 目标 run-client 还能够启动单独的 Java 客户应用。

```
ant -f jboss-build.xml run-client
```

这是基于 Swing 的 Java 应用，以实现对顾客和账号的管理。

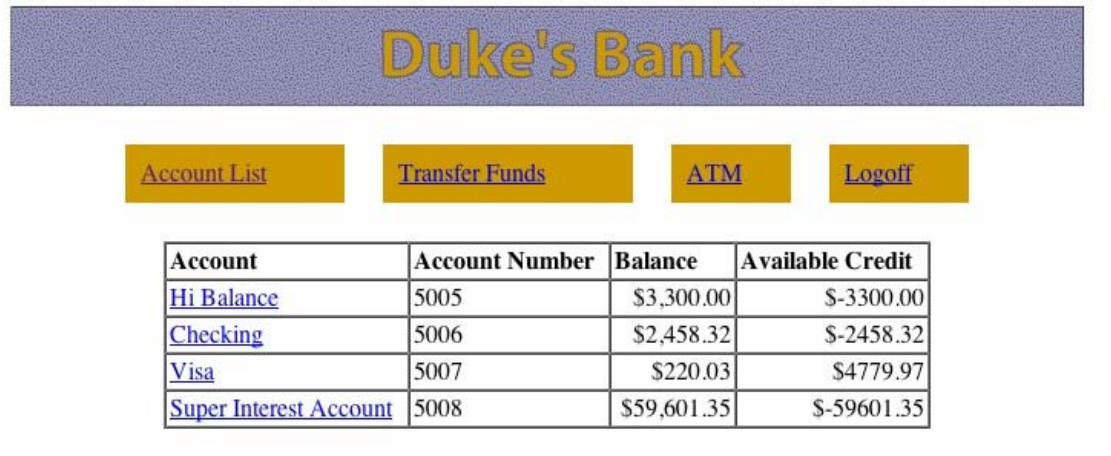

图 4-2 Duke 银行

4.2 JNDI 和 Java 客户

另外，花些精力研究单独 Java 客户应用是如何使用 JNDI，还是很值得的。本实例使用了 J2EE 应用客户框架，它引入了客户端本地环境命名上下文的概念，即能够解析前缀为 java:/comp/env 的 JNDI 名字。这同服务器端使用 JNDI 很类似，因此用户能够避免在客户应用中硬编码 JNDI 名。用户通过使用专有 jboss-client.xml 部署描述符能够解析标准 application-client.xml 部署描述符中的 JNDI 引用名。详情请参考，3.2.1 节。

4.2.1 jndi.properties 文件

为查找 JNDI 服务器，用户需要使用标准 Java 属性。具体细节及其工作机制，请用户参考 JDK API 文档对 javax.naming.Context 类的解析。Java 属性可以通过两种方式提供。其一，硬编码；其二，将 jndi.properties 文件放置在应用类路径中。比如，本实例使用的 jndi.properties 文件如下：

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming.client
j2ee.clientName=bank-client
```

其中，前三个是标准属性，供 JBoss JNDI 实现使用。通过 j2ee.clientName 属性能够标识客户应用部署信息，并告知服务器端。j2ee.clientName 名必须匹配 jboss-client.xml 描述符中 jndi-name 指定的名字。

```
<jboss-client>
    <jndi-name>bank-client</jndi-name>
    <ejb-ref>
        <ejb-ref-name>ejb/customerController</ejb-ref-name>
        <jndi-name>MyCustomerController</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>ejb/accountController</ejb-ref-name>
        <jndi-name>MyAccountController</jndi-name>
    </ejb-ref>
</jboss-client>
```

当然，如果用户只需要构建简单的 Web 应用，则不需要顾及远程客户的情形。

4.2.2 JMX 控制台中的应用 JNDI 信息

既然我们在此阐述 JNDI 主题，则让我们再次回到 JBoss JMX 控制台，来看看通过它能够浏览到应用的哪些信息。请单击 service=JNDIView 链接，然后单击 list()操作，即显示服

务器的 JNDI 树。用户应用能够浏览到如下几方面的信息。其一，位于列表顶端，为 Duke 银行应用提供的 EJB 模块；其二，各个 EJB 的各自私有环境命名上下文；其三，各个 EJB 连接到 JBoss 服务器的入口名。下面给出了整理过的 JNDI 列表：

```
Ejb Module: bank-ejb.jar

java:comp namespace of the CustomerBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the AccountBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the TxBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the NextIdBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the AccountControllerBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)
   |    +- ejb (class: org.jnp.interfaces.NamingContext)
   |    |    +- tx[link -> MyTx] (class: javax.naming.LinkRef)
   |    |    +- nextId[link -> MyNextId] (class: javax.naming.LinkRef)
   |    |    +- account[link -> MyAccount] (class: javax.naming.LinkRef)
   |    |    +- customer[link -> MyCustomer] (class: javax.naming.LinkRef)

java:comp namespace of the CustomerControllerBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)
   |    +- ejb (class: org.jnp.interfaces.NamingContext)
   |    |    +- tx[link -> MyTx] (class: javax.naming.LinkRef)
   |    |    +- nextId[link -> MyNextId] (class: javax.naming.LinkRef)
   |    |    +- account[link -> MyAccount] (class: javax.naming.LinkRef)
   |    |    +- customer[link -> MyCustomer] (class: javax.naming.LinkRef)

java:comp namespace of the TxControllerBean bean:
   +- env (class: org.jnp.interfaces.NamingContext)
   |    +- ejb (class: org.jnp.interfaces.NamingContext)
   |    |    +- tx[link -> MyTx] (class: javax.naming.LinkRef)
   |    |    +- nextId[link -> MyNextId] (class: javax.naming.LinkRef)
   |    |    +- account[link -> MyAccount] (class: javax.naming.LinkRef)
   |    |    +- customer[link -> MyCustomer] (class: javax.naming.LinkRef)
```

```
java: Namespace
  +- XAConnectionFactory (class: org.jboss.mq.SpyXAConnectionFactory)
  +- DefaultDS (class: org.jboss.resource.adapter.jdbc.WrapperDataSource)
  +- SecurityProxyFactory (class: org.jboss.security.SubjectSecurityProxyFactory)
  +- DefaultJMSProvider (class: org.jboss.jms.jndi.JNDIProviderAdapter)
  +- comp (class: javax.naming.Context)
  +- JmsXA (class: org.jboss.resource.adapter.jms.JmsConnectionFactoryImpl)
  +- ConnectionFactory (class: org.jboss.mq.SpyConnectionFactory)
  +- jaas (class: javax.naming.Context)
  |    +- HsqlDbRealm (class: org.jboss.security.plugins.SecurityDomainContext)
  |    +- jmx-console (class: org.jboss.security.plugins.SecurityDomainContext)
  |    +- jbossmq (class: org.jboss.security.plugins.SecurityDomainContext)
  |    +- JmsXARealm (class: org.jboss.security.plugins.SecurityDomainContext)
  +- timedCacheFactory (class: javax.naming.Context)
          Failed to lookup: timedCacheFactory, errmsg=org.jboss.util.TimedCachePolicy
  +- TransactionPropagationContextExporter
          (class: org.jboss.tm.TransactionPropagationContextFactory)
  +- StdJMSPool (class: org.jboss.jms.asf.StdServerSessionPoolFactory)
  +- Mail (class: javax.mail.Session)
  +- TransactionPropagationContextImporter
          (class: org.jboss.tm.TransactionPropagationContextImporter)
  +- TransactionManager (class: org.jboss.tm.TxManager)
```

如果将屏幕滚动到底端，即java:命名空间[4]之下，用户能够浏览到活动安全性管理器列表。如下给出了绑定的安全性域名字。

```
+- jaas (class: javax.naming.Context)
| +- dukesbank (class: org.jboss.security.plugins.SecurityDomainContext)
| +- JmsXARealm (class: org.jboss.security.plugins.SecurityDomainContext)
| +- jbossmq (class: org.jboss.security.plugins.SecurityDomainContext)
| +- HsqlDbRealm (class: org.jboss.security.plugins.SecurityDomainContext)
```

请注意，只有需要某安全性域时，它才会被创建，因此如果应用使用了 dukesbank 域，并试图登陆到系统中，则才能看到 dukesbank 入口。

4.3 安全性

初次访问 Duke 银行应用时，用户需要依据应用提供的简单登陆表单给出用户名和密码。与此同时，然而为使用 J2EE 安全性，开发者需要在应用服务器中配置相应的信息。J2EE 规范本身并不关注认证逻辑的具体行为。实际上，安全性域控制了认证机制。本节将研究 Duke

[4] java:命名空间中的名字仅供运行JBoss的JVM内部使用。远程客户不能够使用它们，除非使用了全局命名空间中的JNDI名字。

银行应用中安全性域的配置。

4.3.1 配置安全性域

通常，用于 Web 和 EJB 层的、标准的 J2EE 安全性声明需要借助于 web.xml 和 ejb-jar.xml 部署描述符指定。然而，为配置使用 JBoss 安全性，开发者还需要提供 JBoss 专有的部署描述符。

通过 JBoss 具体部署描述符能够完成应用的安全性配置。为保护 Web 应用，用户需要将 security-domain 元素包含在 jboss-web.xml 中。

```
<jboss-web>
    <security-domain>java:/jaas/dukesbank</security-domain>
    ...
</jboss-web>
```

如果需要在 EJB 层实现访问控制，则用户也可以对 jboss.xml 文件添加同样的 security-domain 元素。

```
<jboss>
    <security-domain>java:/jaas/dukesbank</security-domain>

    <enterprise-beans>
        ...
    </enterprise-beans>
</jboss>
```

这将意味着，JBoss 会在 JNDI 名“java:/jaas/dukesbank”下为 Duke 银行应用绑定安全性管理器实例。所有的安全性域配置在 java:/jaas 上下文，因此 Duke 银行应用实际上使用了 dukesbank 安全性域。

用户通过 conf/login-config.xml 文件能够配置它。但是，如果浏览 Duke 银行应用，并不能够找到 dukesbank 安全性域。一旦 JBoss 寻找不到相应的安全性域，则它会使用 other 域。其中，other 域的具体配置如下。

```
<application-policy name="other">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                        flag="required" />
    </authentication>
</application-policy>
```

此处，登陆模块使用本地属性文件来认证应用。JBoss 借助于两个文件进行认证工作。

其一，提供用户名、密码；其二，提供角色。比如，如下是 Duke 银行应用的 users.properties。

```
# users.properties file for use with the UsersRolesLoginModule
# Format is:
#
# username=password

200=j2ee
```

属性文件的格式非常简单。各行采用 username=password 的形式。因此，上述文件定义了 200 用户，其密码为 j2ee。这就是用户访问 Duke 银行应用的帐号。如果用户修改了上述密码，则需要重新部署 Web 应用。

当然，用户名和密码不是驱动 J2EE 应用安全性的唯一因素。部署者需要将用户指定角色，因此应用会根据用户的角色信息来决定用户是否有权访问目标资源。只有是 Duke 银行应用的客户才有权访问它，即通过 bankCustomer 角色。下面给出了 roles.properties 文件，用于指定 200 用户的角色。

```
# A roles.properties file for use with the UsersRolesLoginModule
#
# Format is
#
# username=role1,role2,role3

200=bankCustomer
```

为让 Duke 银行应用使用 dukesbank 安全性域，而不是使用服务器提供的默认安全性域，开发者还需定义出 dukesbank 安全性域。因此，开发者需要往 conf/login-config.xml 文件中添加如下内容：

```
<application-policy name="dukesbank">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                       flag="required" />
    </authentication>
</application-policy>
```

注意，需要重启 JBoss，否则修改后的 login-config.xml 不会生效。

4.3.2 使用 RDBMS 实现安全性

实际应用中，将用户、角色信息存储在数据库中很常见。JBoss 提供了称之为

DatabaseServerLoginModule 的登陆模块，用户只需要做少许的配置即可使用它。用户需要提供如下内容：

- 获得用户密码的 SQL 查询语句
- 获得用户角色的 SQL 查询语句
- 待使用数据源的 JNDI 名

因此，可以使用现有的数据库模式。现假定：使用如下 SQL 语句创建安全性相关的数据库表。

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), userRoles VARCHAR(32))

INSERT INTO Users VALUES ('200','j2ee')
INSERT INTO UserRoles VALUES ('200','bankCustomer')
```

然后，将它们作为 Duke 银行应用的安全性数据库。需要对 login-config.xml 作如下修改：

```
<application-policy name="dukesbank">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
                       flag="required">
            <module-option name="dsJndiName">java:/DefaultDS</module-option>
            <module-option name="principalsQuery">
                select passwd from Users where username=?
            </module-option>
            <module-option name="rolesQuery">
                select userRoles,'Roles' from UserRoles where username=?
            </module-option>
        </login-module>
    </authentication>
</application-policy>
```

其中，获得密码的SQL语句很好理解。但是，对于获得角色的SQL语句而言，多出了称之为“Roles”的角色组域。用户可以添加自身需要的、处于该角色组中的其他角色。JBoss本身要求，其取值是“Roles”。本实例的数据库中，仅仅存在一套角色，并无角色组信息[5]。

上述登陆模块中，使用了默认数据源。如果用户在使用 Hypersonic，则可以使用数据库管理器工具创建数据库表、添加数据（Duke 银行应用一章曾给出介绍）。

[5] 用户可以使用默认数据库模式，即创建含有PrincipalID和Password的表Principals；含有PrincipalID、Role、RoleGroup的表Roles。对于默认模式而言，用户不需要在登陆模块中指定SQL查询语句。其中，RoleGroup的取值必须是Roles。

4.3.3 使用密码散列

到目前为止，本书使用到的登陆模块都支持密码散列，即不是以明文存储密码，而是存储单工密码（比如，使用 MD5）。这同 UNIX 系统中的/etc/passwd 文件类似。因此，用户即使浏览到密码，也不能够登陆到系统中。然而，这种方式也有缺点。其一，万一用户忘记密码，则不能够恢复。其二，管理工作变得较复杂，因为用户需要计算密码散列，并放置在安全性数据库中。当然，这并不妨碍其使用。为生效密码散列，用户需要添加如下模块选项。

```
<module-option name="hashAlgorithm">MD5</module-option>
<module-option name="hashEncoding">base64</module-option>
```

这表明使用了 MD5 散列、base64 编码，从而将二进制散列值转换为字符串。JBoss 在认证用户之前，会使用上述选项计算出密码的散列，因此用户必须确保数据库中存储了正确的散列信息。如果用户使用 UNIX 系统，或者在 Windows 上安装了 Cygwin，则可以使用 openssl 计算散列。

```
$ echo -n "j2ee" | openssl dgst -md5 -binary | openssl base64
glcikLhvxq1BwPBZN0EGMQ==
```

用户应该将“glcikLhvxq1BwPBZN0EGMQ==”插入到数据库中，而不是明文，“j2ee”。或者，可以使用 jbosssx.jar 中的 org.jboss.security.Base64Encoder 类。

```
$ java -classpath ./jbosssx.jar org.jboss.security.Base64Encoder j2ee MD5
[glcikLhvxq1BwPBZN0EGMQ==]
```

如果用户只提供单个参数，则上述类只会以 base64 对它进行编码。但如果将加密算法名作为第二个参数，则它将计算出第一个参数的散列

第 5 章 J2EE 之 Web 服务

从一开始，Web 服务就承诺：使用平台、语言独立协议（比如，基于 HTTP 的 SOAP）传输 XML 数据，以实现应用的真正互操作性。当初，由于存在多个竞争标准和开发者对 Web 服务的误解使得这种承诺不能够成为现实。现在，Web 服务已经成熟，并且富有标准化。J2EE 1.4 规范已经将 Web 服务集成进来了。

秉承本书的初衷，还是假设用户已经有开发Web服务的一些经验。如果用户还未涉足过，本书建议用户补上这一课。JBoss wiki，即http://www.jboss.org/wiki/Wiki.jsp?JBossWS，是学习Web服务较好的切入点，因为它深入地介绍了开发基于JBoss的Web服务。另外，本书还推荐Richard Monson-Haefel的《J2EE Web Services》，一本对J2EE Web服务的更一般性介绍。

5.1 JBoss 中的 Web 服务

JBossWS是JBoss 4.0 中用于提供Web服务的JBoss模块，它替换了以前的JBoss.NET包。同以前一样，JBossWS还是基于Apache Axis（http://ws.apache.org/axis）。然而，JBossWS提供了J2EE 1.4 Web服务的完整技术栈，其中包括SOAP、SAAJ、JAX-RPC以及JAXR。

J2EE Web 服务提供了两种 Web 服务端点（endpoint）。如果将 Web 服务作为平台独立的调用层看待，则端点就是用户暴露的对象、也是能够调用其提供的操作的对象。自然地，J2EE Web 服务支持将 EJB 暴露为 Web 服务，但只有无状态会话 Bean 能够如此。因此，这也说明 Web 服务请求也具有无状态的特质。另外，J2EE Web 服务还提供 JAX-RPC 服务端点（JAX-RPC Service Endpoint，JSE），它仅仅是简单 Java 类。本实例只会涉及到 EJB 端点。

5.2 将 Duke 银行运行为 Web 服务

本章将继续开发第 4 章的 Duke 银行应用，并创建简单 Web 服务，即查询账号和收支情况。会话 Bean，即 AccountController 能够实现账号和收支情况的查询。但是，它是有状态的。因此，不能够直接暴露 AccountController。本章将创建新的无状态会话 Bean，即 TellerBean，一更适合于 Web 服务端点的无状态会话 Bean。

再进入操作之前，用户必须保证，确实根据第 4 章给出的操作步骤编译并部署了 Duke 银行应用。同前述一样，将 examples/bank 作为工作目录。在部署 Duke 银行应用时，可能已经编译了 TellerBean，但是再次调用 Ant 目标 compile 并无坏处。

```
ant -f jboss-build.xml compile
```

然后，让我们看看部署描述符。用户已经掌握了如何部署会话 Bean。将会话 Bean 部署为 Web 服务很简单，只需要在 ejb-jar.xml 的会话 Bean 定义中添加 service-endpoint 元素。其中，service-endpoint 元素指定的类提供了对应于会话 Bean 中方法的接口，而相应的会话 Bean 暴露为 Web 服务。

```
<session>
    <ejb-name>TellerBean</ejb-name>
```

```
    <service-endpoint>com.jboss.ebank.TellerEndpoint</service-endpoint>
    <ejb-class>com.jboss.ebank.TellerBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <ejb-ref>
        <ejb-ref-name>ejb/account</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.sun.ebank.ejb.customer.CustomerHome</home>
        <remote>com.sun.ebank.ejb.customer.Customer</remote>
    </ejb-ref>
    <security-identity>
        <run-as>
            <role-name>bankCustomer</role-name>
        </run-as>
    </security-identity>
</session>
```

用户可能已经注意到，这里并没有为 TellerBean 声明 Home 或远程接口。如果某会话 Bean 仅供 Web 服务接口访问，则不需要声明 Home 或远程接口，因此在此省略了它们。其中，将 TellerEndpoint 类声明为端点接口。该接口暴露了两个操作，它们均对应于 TellerBean 中提供的方法。

```
public interface TellerEndpoint
    extends Remote
{
    public String[] getAccountsOfCustomer(String customerId)
        throws RemoteException;

    public BigDecimal getAccountBalance(String accountID)
        throws RemoteException;
}
```

接下来，需要借助于上述接口，并使用 JBossWS 附带的 wstools 工具生成 Web 服务相关部署描述府（WSDL、JAX-RPC 映射文件、webservices 部署描述府），即互操作 Web 服务定义。Ant 目标 wstools 操作如下：

```
ant -f jboss-build.xml wstools
```

注意，wstools 使用 wstools-config.xml 文件控制上述内容的生成。

```
<configuration xmlns="http://www.jboss.org/jbossws-tools">
```

```
    <java-wsdl>
        <service name="TellerService" style="rpc" endpoint="com.jboss.ebank.TellerEndpoint"/>
        <namespaces target-namespace="http://ebank.jboss.com/teller"
                    type-namespace="http://ebank.jboss.com/teller/types"/>
        <mapping file="jaxrpc-mapping.xml"/>
        <webservices ejb-link="TellerBean"/>
    </java-wsdl>
</configuration>
```

上述 Ant 任务生成了如下配置信息：

- TellerService.wsdl：这是描述 Web 服务的 WSDL 定义。注意，在运行期，JBossWS 会替换其中的一些取值。
- jaxrpc-mapping.xml：它描述 WSDL 类型与底层 Java 实现间的映射关系。由于这里的客户程序也是采用 Java 编写的，因此开发者在完成客户端映射时能够重用它。
- webservices.xml：描述 webservice。它将 Endpoint 连接到服务器端组件（比如，会话 Bean）。同时，它还给出了相关部署文件的位置信息。

这里的 Web 服务是简单会话 Bean，因此只需要将相关类和部署描述符打包到 EJB jar 文件中。其中，package-ws 任务能够完成打包工作，deploy-ws 目标能够将 EJB jar 部署到 JBoss。

```
ant -f jboss-build.xml package-ws
ant -f jboss-build.xml deploy-ws
```

一旦上述 Web 服务部署完成，用户通过如下 URL 能够查看到 WSDL 文件。

http://localhost:8080/bankws-ejb/TellerService?wsdl

在本实例中，创建了 WSDL 文件。反之，用户也可以手工编写用于 Web 服务的 WSDL 文件，然后借助于 JBossWS 附带的 wstools 工具生成 Java 端点接口。

5.3 运行 Web 服务客户

同时，本书还提供了从非 J2EE 环境中访问该 Web 服务的 Java 客户应用。

```
public class WSClient {
    public static void main(String[] args)
        throws Exception
    {
        URL url =
                new URL("http://localhost:8080/bankws-ejb/TellerService?wsdl");

        QName qname = new QName("http://ebank.jboss.com",
```

```
                                        "TellerService");

        ServiceFactory factory = ServiceFactory.newInstance();
        Service        service = factory.createService(url, qname);

        TellerEndpoint endpoint = (TellerEndpoint)
            service.getPort(TellerEndpoint.class);

        String     customer = "200";
        String[] ids        = endpoint.getAccountsOfCustomer(customer);

        System.out.println("Customer: " + customer);
        for (int i=0; i<ids.length; i++) {
            System.out.println("account[" + ids[i] +   "]   " +
                                        endpoint.getAccountBalance(ids[i]));
        }
    }
}
```

用户能够使用 Ant 目标 run-ws 运行上述 Java 客户。

```
ant -f jboss-build.xml run-ws
```

客户端将显示出属于某给定顾客的账号收支情况。

```
    [java] Customer: 200
    [java] account[5005]   3300.00
    [java] account[5006]   2458.32
    [java] account[5007]   220.03
    [java] account[5008]   59601.35
```

5.4 监控 Web 服务请求

在处理 Web 服务请求时，开发者都希望能够观察到客户同服务器间传递的实际消息。JBoss 通过 org.jboss.ws.server.ServiceEndpoint Category 记录了这类信息。为了能够记录它们，需要往 log4j.xml 文件新增如下内容：

```
<category name="org.jboss.ws.server.ServiceEndpoint">
    <priority value="DEBUG"/>
</category>
```

一旦设置好后，所有的 SOAP 请求和响应都将记录到 server.log 文件中。比如，在调用 getAccountOfCustomer 方法后，能够在 server.log 中发现如下内容：

```
2005-05-16 17:50:43,479 DEBUG [org.jboss.axis.transport.http.AxisServlet]
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <ns1:getAccountsOfCustomer xmlns:ns1="http://ebank.jboss.com">
     <in0>200</in0>
   </ns1:getAccountsOfCustomer>
 </soapenv:Body>
</soapenv:Envelope>
...
2005-05-16 17:50:44,240 DEBUG [org.jboss.axis.transport.http.AxisServlet]
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <ns1:getAccountsOfCustomerResponse xmlns:ns1="http://ebank.jboss.com">
     <getAccountsOfCustomerResponse>
       <accounts>5005</accounts>
       <accounts>5006</accounts>
       <accounts>5007</accounts>
       <accounts>5008</accounts>
     </getAccountsOfCustomerResponse>
   </ns1:getAccountsOfCustomerResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

第 6 章 JMS 和消息驱动 Bean

Duke 银行应用并没有使用到 JMS 消息，因此本章将再次参考 J2EE Tutorial 中有关消息驱动 Bean（Message Driven Bean，MDB）的实例，从而研究如何在 JBoss 中使用消息。本书仍然假定用户已经熟悉 JMS 和 MDB 基本概念。其中，J2EE Tutorial 中用于 MDB 的代码位于 j2eetutorial14/examples/ejb/simplemessage。位于 simplemessage 目录下存在 jboss-build.xml 文件，供从头编译和运行 JBoss 版本的 MDB 使用。

实例代码很简单。仅仅有两个类，一个用于客户端，一个用于 MDB（同其他 EJB 不同，它不需要其他接口）。客户将消息发布到 JMS Queue 中，MDB 借助于标准的 onMessage 方法处理这些消息。其中，这些消息都是 javax.jms.TextMessage 类型，该 MDB 只是简单地打印出各个消息含有的文本信息。

同容器相关的任务有两个。其一，在 JBoss 中设置 Queue。其二，配置 MDB，以接收来自该 Queue 的消息。

6.1 构建实例

6.1.1 编译并打包 MDB 和客户

从 simplemessage 目录调用 compile-mdb 目标，能够完成代码的编译工作。

```
ant -f jboss-build.xml compile-mdb
```

然后，依次运行如下 Ant 目标，能够在 jar 目录生成 MDB 存档、客户存档以及由它们合并的 EAR 文件。

```
ant -f jboss-build.xml package-mdb
ant -f jboss-build.xml package-mdb-client
ant -f jboss-build.xml assemble-mdb
```

请注意，这同 Duke 银行应用建立的目录结构相同，即 dd 目录含有部署描述符、jar 目录含有构建过程生成的存档。

6.1.1.1 指定 MDB 监听的 Queue

同 JBoss 具体容器其他信息一样，通过 jboss.xml 能够指定 MDB 使用的 Queue 名。

```
<jboss>
    <enterprise-beans>
        <message-driven>
            <ejb-name>SimpleMessageBean</ejb-name>
```

```
            <destination-jndi-name>queue/MyQueue</destination-jndi-name>
        </message-driven>
    </enterprise-beans>
</jboss>
```

因此，该 MDB 将处理来自 JNDI 名为“queue/MyQueue”的 Queue 中的消息。

6.2 部署和运行实例

将 SimpleMessage.ear 文件拷贝到 JBoss 的 deploy 目录，就完成了 MDB 的部署。其中，Ant 目标 deploy-mdb 能够完成其部署。

```
ant -f jboss-build.xml deploy-mdb
```

如下给出了成功部署 SimpleMessage.ear 的示例。

```
08:20:08,188 INFO  [EARDeployer] Init J2EE application:
file:/tmp/jboss-4.0.4.GA/server/default/deploy/SimpleMessage.ear
08:20:08,370 INFO  [EjbModule] Deploying SimpleMessageEJB
08:20:08,565 INFO  [ClientDeployer] Client ENC bound under: SimpleMessageClient
08:20:08,712 WARN  [JMSContainerInvoker] Could not find the queue
destination-jndi-name=queue/MyQueue
08:20:08,719 WARN  [JMSContainerInvoker] destination not found: queue/MyQueue
reason: javax.naming.NameNotFoundException: MyQueue not bound
08:20:08,719 WARN  [JMSContainerInvoker] creating a new temporary destination: queue/MyQueue
08:20:08,772 INFO  [MyQueue] Bound to JNDI name: queue/MyQueue
08:20:08,952 INFO  [EJBDeployer] Deployed:
file:/tmp/jboss-4.0.4.GA/server/default/tmp/deploy/tmp51464SimpleMessage.ear-contents
/simplemessage.jar
08:20:08,996 INFO  [EARDeployer] Started J2EE application:
file:/tmp/jboss-4.0.4.GA/server/default/deploy/SimpleMessage.ear
```

如果用户仔细的话，能够看到一些警告信息，即部署描述符中指定的 JMS Queue 不存在。对于 JBoss 而言，会为应用创建临时的 Queue，并绑定到相同的 Queue 名。用户可以通过 JNDIView MBean 验证，位于全局 JNDI 命名空间中。接下来，来研究如何显示地创建 JMS 目的地。

6.2.1 运行客户应用

通过 Ant 目标 run-mdb 能够运行客户应用。

```
ant -f jboss-build.xml run-mdb
```

用户能够在客户和服务器控制台浏览到它们各自发送和接收消息的输出信息。

6.3 管理 JMS 目的地

同 JBoss 其他架构组件一样，JMS Topic 和 Queue 也是基于 MBean 实现的。用户可以通过两种方式创建它们。其一，将 MBean 声明添加给相应的配置文件。其二，使用 JMX 控制台应用动态创建它们。然而，对于第二种方式，如果服务器重启，则动态创建的 MBean 将丢弃掉。

6.3.1 jbossmq-destinations-service.xml 文件

通过 deploy/jms 目录能够找到 jbossmq-destinations-service.xml 文件。它含有 JMS 目的地列表，并且配置了供测试目的的 Topic 和 Queue 列表。为添加 SimpleMessage 需要的 Queue，用户只需简单地将如下 MBean 声明添加到文件中。

```
<mbean code="org.jboss.mq.server.jmx.Queue"
        name="jboss.mq.destination:service=Queue,name=MyQueue">
</mbean>
```

6.3.2 从 JMX 控制台使用 DestinationManager

当 JBoss 处于运行状态时，将 Web 浏览器定位到 JMX 控制台应用，然后请用户找到 jboss.mq 域名。然后，单击 service=DestinationManager。其中，DestinationManager MBean 是 JBossMQ 中的主要 JMS 服务，用户能够使用它在运行时创建、销毁 Topic 和 Queue。请找到 createQueue 操作。用户需要为它提供两个参数。其一，用于 Queue MBean 的名字（一般情况下，与应用代码无关）。其二，JNDI 名。请分别输入 MyQueue 和 queue/MyQueue。本书采用了标准的 JBoss 约定，即将 Queue 绑定在 JNDI 名 queue 中、Topic 绑定在 JNDI 名 topic 中。用户可以不遵循该约定，而使用任何名字。请注意，如果某名字已被占用，则将失败（比如，如果用户已根据上述方式部署了 SimpleMessage 应用，或者使用 XML 配置文件添加了 Queue MBean。）。如果遇到这种失败，则用户需要删除现有的 Queue，或者尝试使用其他的名字。

6.3.3 管理目的地

用户能够访问表示 Queue 或 Topic 的 MBean 的属性和操作，而且它们是借助于 JMX 暴露的。请用户再次查看 JMX 控制台主视图，应该能够找到 jboss.mq.destination 部分，其中存在供 SimpleMessage 使用的 Queue 入口（无论 Queue 是通过何种方式创建的）。单击后，用户能够看到该 Queue 的属性。其中，QueueDepth 表示当前处于 Queue 中的消息数量。

作为练习，用户可以暂时停止对 MDB 的消息分发。具体停止步骤如下：用户需要在 JMX 控制台中定位到 jboss.j2ee 域名，然后用户能够看到 MBean 列表，其中有负责调用 MDB 的

MBean 实例。其名字为 binding=message-driven-bean,jndiName=local/SimpleMessageEJB, plugin=invoker,service=EJB。

用户可以使用对应 MBean 支持的操作来启动或停止消息的分发。请调用 stopDelivery 方法，然后运行客户应用几次。用户然后能够看到 QueueDepth 也会相应增长。如果调用 startDelivery 方法，则立即能够看到消息的到来。

第 7 章 容器管理持久化

Duke 银行应用使用 Bean 管理持久化（Bean-Managed Persistence，BMP）。然而，EJB 2.0 引入的容器管理持久化（Container-Managed Persistence，CMP）改进了 BMP，使得实际应用中很少使用 BMP。本章内容将使用 J2EE Tutorial 中的 RosterApp 实例应用，它涉及到 CMP 和 CMR 的使用。在用户继续深入本章之前，请务必阅读 CMP 相关教程，从而对 CMP 及其关系有较好的理解。

用户通过 j2eetutorial14/examples/cmproster 能够找到代码。该应用实现了俱乐部队员的花名册。一共存在 3 个实体 Bean：PlayerEJB、TeamEJB、LeagueEJB；1 个会话 Bean：RosterEJB，它操作上述 3 个实体 Bean，并且位客户应用提供业务方法。只有该会话 Bean 提供了远程接口。

7.1 构建实例

将上述 EJB 打包成两个单独的 jar 文件，一个是实体 Bean，另一个是会话 Bean。同以前一样，需要为各个 jar 提供 ejb-jar.xml 文件。本实例不需要 jboss.xml 文件。所有构建数据库模式的 CMP 信息都包含在标准描述符中。本章将在后续内容中讨论 JBoss 具体的自定义工作。

将 examples/ejb/cmproster 目录作为工作目录，然后运行 compile-cmp 目标，从而完成所有代码的编译工作。

```
ant -f jboss-build.xml compile-cmp
```

然后，运行 package-team，以构建包含实体 Bean 的 EJB jar 文件。

```
ant -f jboss-build.xml package-team
```

运行 package-roster 目标，以构建包会话 Bean 的 EJB jar 文件。

```
ant -f jboss-build.xml package-roster
```

通过 jar 目录能够找到它们。运行 package-roster-client 目标能够构建客户 jar 文件。

```
ant -f jboss-build.xml package-roster-client
```

最后，使用 assemble-roster 目标，以集成 RosterApp EAR 存档。

```
ant -f jboss-build.xml assemble-roster
```

7.2 部署和运行实例

使用 Ant 目标 deploy-cmp 能够应用的部署。

```
ant -f jboss-build.xml deploy-cmp
```

从 jar 目录将 RosterApp.ear 文件拷贝到 JBoss deploy 目录（或者使用 Ant 目标 deploy-cmp），然后查看服务器的输出信息。

```
13:49:11,044 INFO  [EARDeployer] Init J2EE application: file:/private/tmp/jboss-4.0.4/serv
er/default/deploy/RosterApp.ear
13:49:11,884 INFO  [EjbModule] Deploying RosterBean
13:49:13,366 INFO  [EjbModule] Deploying TeamBean
13:49:13,751 INFO  [EjbModule] Deploying LeagueBean
13:49:13,842 INFO  [EjbModule] Deploying PlayerBean
13:49:14,377 INFO  [EJBDeployer] Deployed: file:/private/tmp/jboss-4.0.4/server/default/tm
p/deploy/tmp29312RosterApp.ear-contents/roster-ejb.jar
13:49:17,931 INFO  [EJBDeployer] Deployed: file:/private/tmp/jboss-4.0.4/server/default/tm
p/deploy/tmp29312RosterApp.ear-contents/team-ejb.jar
13:49:17,991 INFO  [EARDeployer] Started J2EE application: file:/private/tmp/jboss-4.0.4/s
erver/default/deploy/RosterApp.ear
```

其中，有几点值得用户注意。在 Duke 银行应用中，本书在 jboss.xml 文件中指定了 EJBHome 引用的 JNDI 名。如果没有提供这些 JNDI 名，JBoss 默认使用 EJB 名代替。因此，会话 Bean 绑定到 RosterBean，等等。通过 JMX 控制台能够确认这些信息。

初次部署该应用时，JBoss 会自动创建要求的数据库表。如果用户查看数据库模式（第 4.1.7.3 有介绍），则能够看到 JBoss 为每个实体 Bean 创建了表。而且还创建了一张关系表，以处理队员（player）和对（team），这个多队多关系。其中，表名和列名对应于实体 Bean 名和各个属性。如果这些名字能够满足要求，则用户不用进一步指定数据库模式。现在，用户需要手工将 PlayerBean 中的 position 域映射到 pos 列，因此默认列名 position 是 HSQL 中的关键字。通过 jbosscmp-jdbc.xml 文件能够定义模式。

请注意，如果将 org.jboss.ejb.plugins.cmp 包（2.2.2 节）的日志级别调整到 DEBUG，则 CMP 引擎将把执行的 SQL 命令都打印出来。这对于理解 CMP 引擎的工作机制以及不同调优参数是如何影响数据装载很有帮助。

7.2.1 运行客户应用

客户应用借助于会话 Bean 接口能够完成数据的创建和获得。它创建俱乐部、队以及队员，并将它们添加到数据库中。其中，用户获取数据的会话 Bean 方法主要是封装了对 EJB finder 方法的调用。运行客户应用和输出日志信息如下。

```
$ ant -f jboss-build.xml run-cmp
```

```
Buildfile: jboss-build.xml

run-cmp:
[java] P10 Terry Smithson midfielder 100.0
[java] P6 Ian Carlyle goalkeeper 555.0
[java] P7 Rebecca Struthers midfielder 777.0
[java] P8 Anne Anderson forward 65.0
[java] P9 Jan Wesley defender 100.0

[java] T1 Honey Bees Visalia
[java] T2 Gophers Manteca
[java] T5 Crows Orland

[java] P2 Alice Smith defender 505.0
[java] P22 Janice Walker defender 857.0
[java] P25 Frank Fletcher defender 399.0
[java] P5 Barney Bold defender 100.0
[java] P9 Jan Wesley defender 100.0

[java] L1 Mountain Soccer
[java] L2 Valley Basketball
```

请注意，客户应用没有删除数据。因此如果如果运行客户应用两次以上，应用将抛出异常，因为实体已经存在。如果再次运行，则需要删除这些数据。其中，最简单的方式（如果使用 HSQL）是删除当前工作服务器配置中 data/hypersonic 目录中的内容（假定，用户没有保存其他重要的数据在这里），然后重启服务器。本书同时也提供了简单的 SQL 脚本，以复位数据库，即运行 db-delete 目标。

```
ant -f jboss-build.xml db-delete
```

用户也可以直接使用 HSQL 管理器工具，通过 SQL 命令删除数据。

7.3 自定义 CMP

用户通过 jbosscmp-jdbc.xml 文件能够进一步自定义 CMP 引擎的行为。该文件用于基本信息配置，比如数据源名、类型映射（Hypersonic、Oracle 等等）、在部署时是否自动创建数据库表、在关闭服务器时是否自动删除数据库表。用户能够自定义 EJB 映射到的数据库表名和列名，还可以根据自身要求调整 CMP 引擎装载数据的方式。比如，通过使用 read-ahead 元素，用户能够通过单个 SQL 调用读取并缓存多个 EJB 的数据块，供后续访问使用。用户可以指定 eager 装载组，即初始时，只装载实体 Bean 的某些域数据。相反，通过指定 lazy 装载组，能够实现延时装载。使用类似的机制能够实现 EJB 间关系的访问。如果用户使用 BMP，则不可能有这么灵活，因为每个 Bean 必须使用单个 SQL 调用装载。因此，用户需要

开发自身的 SQL，并手工实现关系管理。选择 CMP，还是 BMP，已经很明显了。极少情况下，用户需要在应用中使用 BMP。

当然，调整 CMP 引擎的详细内容并不是本书的讨论主题。通过阅读 docs/dtd/jbosscmp-jdbc_4_0.dtd（提供了详细注释），用户能够较好地了解 CMP 引擎。在 conf 目录中还存在标准设置，即 standardjbosscmp-jdbc.xml 文件，它提供了默认设置，并且提供了常见数据库的类型映射列表。该文件的开头部分如下。

```
<jbosscmp-jdbc>
    <defaults>
        <datasource>java:/DefaultDS</datasource>
        <datasource-mapping>Hypersonic SQL</datasource-mapping>

        <create-table>true</create-table>
        <remove-table>false</remove-table>
        <read-only>false</read-only>
        <read-time-out>300000</read-time-out>
        <row-locking>false</row-locking>
        <pk-constraint>true</pk-constraint>
        <fk-constraint>false</fk-constraint>
        <preferred-relation-mapping>foreign-key</preferred-relation-mapping>
        <read-ahead>
            <strategy>on-load</strategy>
            <page-size>1000</page-size>
            <eager-load-group>*</eager-load-group>
        </read-ahead>
        <list-cache-max>1000</list-cache-max>
...
```

用户可以看到：standardjbosscmp-jdbc.xml 设置了数据源，并将嵌入式 Hypersonic 数据库作为默认数据库，将 table-creation 设置为 true，将 removal 设置为 false。因此，如果表不存在，CMP 引擎会创建它。read-only 和 read-time-out 元素分别指定数据是否只读和数据的有效时间（单位：毫秒）。请注意，大部分元素都能够用于不同粒度，比如每实体 Bean 级别，或者实体 Bean 的域级别（详细内容，请参考 DTD）。使用了 on-load 策略的 read-ahead 元素表明，在访问实体 Bean（而不是调用 finder 方法）时才装载 EJB 数据。page-size 表明，SQL 命令一次最多能够装载 1000 个实体。用户能够在 jbosscmp-jdbc.xml 文件中覆盖默认设置，或者直接修改 standardjbosscmp-jdbc.xml 文件。

《The JBoss 4 Application Server Guide》一书给出了 CMP 引擎及其不同装载策略的详细阐述。

7.3.1 XDoclet

开发和维护部署描述符很费时间，而且容易出错。尤其是对于那些需要完整自定义 CMP

引擎行为的应用，即需要维护内容很多的、复杂的文件，则更容易出错。如果用户在使用 CMP（或者，EJB），则学习使用 XDoclet 代码生成引擎（http://xdoclet.sourceforge.net）很有必要。XDoclet 使用 JavaDoc 风格属性标签，并能够为用户生成部署描述符、EJB 接口以及其他产出物。XDoclet 对 JBossCMP 提供了完整支持，并且学习曲线非常平滑，因此本书强烈推荐（事实上，确实如此）将 XDoclet 用于实际项目。

第 8 章 使用其他数据库

在以前的章节，本书的应用仅仅使用了 JBoss 默认数据源。其中，默认数据源由嵌入式 HSQL 数据库实例提供，并且绑定到 JNDI 名，“java:/DefaultDS”。在 JBoss 中集成数据库对于运行实例带来了很多便利，而且 HSQL 适合于大部分场合。然而，用户可能需要使用其他数据库，或者替换默认数据源，或者从同一服务器中访问多个数据库。

8.1 配置数据源

JBoss 中的数据库连接管理完全由 JBoss JCA 实现处理。因此，借助于 JCA 资源适配器能够访问数据库，并且能够处理连接池、安全性和事务。

8.1.1 包裹 JDBC 的资源适配器

如果没有为数据库提供专有适配器，则用户需要配置使用 JBoss 提供的、包裹了 JDBC 的资源适配器（2.2.4 节曾给出过讨论）。很明显，用户需要提供 JDBC 驱动，并提供相应的类（仅仅需要将驱动 jar 或 zip 文件拷贝到当前工作的服务器配置中的 lib 目录）。其中，不同数据源配置的主要区别在于，它们是使用本地，还是 XA 事务的 JDBC 适配器。如果某 JDBC 驱动提供了 javax.sql.XADataSource 实现，则支持 XA 事务的数据源可用。有一点请注意，即使 XADataSource 实现可用，用户还是可以选择使用本地事务（比如，请参考 JBoss 提供的两份 Oracle 配置文件）。同时，还存在 no-transaction 配置，用户应该很少以这种方式使用数据库。

8.1.2 数据源配置文件

数据源配置文件的后缀为-ds.xml，因此 JCA 部署器能够正确处理它们。其中，目录 docs/example/jca 包含了用于各种数据库的配置实例，用户最好能够以其中一个为切入点。数据源配置文件格式的完整描述请参考 docs/dtd/jboss-ds_1_5.dtd。其他有关该文件和 JBoss JCA 实现，用户可以参考《The JBoss 4 Application Server Guide》一书。

本地事务数据源可使用 local-tx-datasource 元素配置、XA 事务数据源可使用 xa-tx-datasource 元素配置。示例文件 generic-ds.xml 展示了如何同时使用这两种类型，以及其他可用元素（比如，连接池）的配置。对于同时提供了本地和 XA 配置的数据库实例有 Oracle、DB2、Informix。

如果用户查看 firebird-ds.xml、facets-ds.xml、sap3-ds.xml 实例文件，则将注意到它们的格式完全不同，即根元素为 connection-factories，而不是 datasources。这些文件是基于通用的 JCA 配置语法，供配置了相应的 JCA 资源适配器使用。正如本书于 2.2.4 节阐述的一样，该语法不是针对数据源配置的，比如用于配置 JMS 资源适配器的 jmx-ds.xml 文件。

接下来，本书将给出具体配置的详细步骤。

8.2 将 MySQL 作为默认数据源

MySQL 是世界范围内使用最为流行的开源数据库之一，并且许多大型公司一直都在使

用它，比如 Yahoo 和 NASA。官方提供的 JDBC 驱动称之为 Connector/J。在本实例中，使用了 MySQL 4.1.17 和 Connector/J 3.0.15。通过 http://www.mysql.com 能够下载到它们。

8.2.1 创建数据库和用户

本书假定用户已经熟悉 MySQL 的安装，而且知道如何运行它和其基本知识。通过命令行运行 mysql 客户应用能够执行一些管理工作。用户应该以具有足够权限的用户连接到 MySQL 服务器（比如，通过指定-uroot 选项，使得以 MySQL root 用户运行）。

首先，创建数据库 jboss，供 JBoss 使用。

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

然后，验证是否成功创建。

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

接下来，创建用户名为 jboss、密码为 password 的用户，以访问 jboss 数据库。

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

最后，再次验证一切是否就绪。

```
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)
```

8.2.2 安装 JDBC 驱动和部署数据源

为将 MySQL JDBC 驱动供 JBoss 使用，用户需要将 Connector/J 发布版中的 mysql-connector-java-3.0.15-ga-bin.jar 拷贝到 default 服务器配置中的 lib 目录（当然，用户必须在使用 default 服务器配置）。然后，在 deploy 目录创建称之为 mysql-ds.xml 的数据源配置。另外，相应的 username 和 password 在上述过程已经给出。

```
<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>
```

由于新加了 jar 文件到 lib 目录（译者注：default/lib）中，因此用户需要告知 JBoss，因此如果在 JBoss 运行期间加入 jar 到 lib 目录，则需要重启 JBoss 服务器。

8.2.3 测试 MySQL 数据源

本章将使用 CMP roster 应用测试 MySQL 数据源。其中，唯一需要改变的是将类型映射从 Hypersonic 修改为 MySQL。用户可以添加新的 jbosscmp-jdbc.xml 给 EJB 部署单元，或者修改全局默认 conf/standardjbosscmp-jdbc.xml 设置。当然，第二种方法更简单，因为不用重新打包应用。其缺点是，需要重启 JBoss，否则修改不能够生效。编辑该文件，然后将 datasource-mapping 元素修改为 mySQL。

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

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

重启 JBoss 后，用户应该能够部署应用，并同 7.2 节一样，查看表信息。同 MySQL 客户端能够看到它们。

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

用户还能够看到 JMS 持久化数据库表也在这里，因为本书已经将 MySQL 作为默认数据源了。

8.3 设置 Oracle9i 的 XADataSource

在商业数据库领域，Oracle是主要的竞争者之一。因此，大部分用户都会涉及到Oracle的使用。用户可以通过http://www.oracle.com免费下载，供非商业目的使用。安装和配置Oracle并不是很关键的问题。Oracle是功能强大的数据库，而且它还提供了很多额外的功能和技术（Apache Web服务器、多个JDK、ORB等等）。这些方面可能并不用户所需要的，但通常都会随Oracle数据库一同安装。因此，本书假定用户已安装了Oracle，比如，本文使用Oracle 10g。

8.3.1 出于 Oracle 兼容性而设置 Pad 值

如果用户查看 default/conf 目录中的 jboss-service.xml 文件，将发现如下 MBean 服务。

```
<!-- 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>
```

事务服务需要使用 XidFactory MBean 服务创建 XA 事务标识。其中的注释表明：如果使用 Oracle 数据库，则用户需要将属性 Pad 设置为 true 的那行也包括进来。这使得 XA 事务标识能够达到 64 位的最大长度。记得去重启 JBoss 应用服务器，否则修改不会生效。当然，还是等安装 JDBC 驱动后再说吧！

8.3.2 安装 JDBC 驱动和部署数据源

通过$ORACLE_HOME/jdbc/lib 目录能够找到 Oracle JDBC 驱动。其中，用户可能都比较熟悉旧版本的 Oracle JDBC 驱动，比如 classes12.zip。在写作本书时，其最新版为 ojdbc14.jar。同时，为解决 Oracle JDBC 使用过程中的一些问题，用户可以使用后缀为_g 的调试版本。同样，用户需要将相应的 jar 文件拷贝到 JBoss default 配置的 lib 目录中。用于非 XA 的基本驱动类是 oracle.jdbc.driver.OracleDriver。本书将使用的 XADataSource 类称之为 oracle.jdbc.xa.client.OracleXADataSource。

用户需要拷贝一份 oracle-xa-ds.xml 配置文件，然后修改它，比如需要设置正确的 URL、用户名、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.xa.oracle.OracleXAExceptionFormatter"
name="jboss.jca:service=OracleXAExceptionFormatter">
        <depends optional-attribute-name="TransactionManagerService">
            jboss:service=TransactionManager
        </depends>
    </mbean>
</datasources>
```

本书使用了 Oracle Thin（纯 Java）驱动，并假定数据库运行在主机 monkeymachine 上。同时，数据库名（用 Oracle 行话说，就是 SID）为 jboss。本书还假定用户创建的 jboss 用户具有足够的权限。对于本实例而言，只需要给它赋予 dba 的权限。

```
SQL> connect / as sysdba
Connected.
```

```
SQL> create user jboss identified by password;
User created.
SQL> grant dba to jboss;
Grant succeeded.
```

好了，请将配置文件拷贝到 deploy 目录。通过 JBoss 服务器控制台，用户应该可以看到如下输出信息。

```
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'
```

同以前一样，如果用户从 JMX 控制台使用 JNDIView MBean 服务，将从列表中看到 java:/XAOracleDS。

8.3.3 测试 Oracle 数据源

本节内容将再次使用 CMP 实例，以试用新创建的数据源。因此，用户需要将如下内容添加到 jbosscmp-jdbc.xml 文件中。

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

另外，还存在其他可用的 Oracle 类型映射。如果用户使用了旧版本的 Oracle，则查看 conf/standardjbosscmp-jdbc.xml 文件能够找到相应的类型映射。同上一样，用户可以修改其默认取值，即为所有 CMP 部署应用设置全局值，而且用户不用重新打包 EAR 文件。

同以前部署应用一样，检查是否有错误信息输出。然后，使用 Oracle SQL Plus 工具再次验证是否有数据库表创建。

```
SQL> select table_name from user_tables;

TABLE_NAME
------------------------------
TEAMBEAN
LEAGUEBEAN
PLAYERBEAN
PLAYERBEAN_TEAMS_TEAM_1OFLZV8
```

第 9 章 使用 Hibernate

Hibernate，非常流行的持久化引擎。无论是在功能上，还是使用的简单性，它都能够替代实体 Bean。它对运行环境没有任何要求，无论是在应用服务器中，还是在应用服务器之外。然而，当在 JBoss 服务器中运行时，开发者可以选择将应用部署成 HAR 存档，使得 Hibernate 的应用更简单。JBoss 能够管理 Hibernate Session 及其他配置细节。通过少许配置，开发者即可在 JBoss 中使用到 Hibernate 技术了。

本章将重温第七章，CMP。其中，给出了 CMP roster 应用。本文将使用 Hibernate 访问数据库表。另外，我们还将创建 HAR 文件，并从 Web 应用中访问到 HAR 中的 Hibernate 对象。当然，整个应用将打包成 EAR 文件。

其中，examples/hibernate 目录存放了相应的源代码。当然，这需要同 CMP roster 应用配合使用。因为，HAR 存档需要使用 CMP roster 中的表和数据。

我们也将研究 JBoss 中如何部署 Hibernate 应用的详细步骤。至于 Hibernate 更完整的介绍，请参考《Hibernate in Action》一书，由 Christian Bauer 和 Gavin King 于 Manning 在 2004 年出版发行。

9.1 创建 Hibernate 存档

我们待开发的 Web 应用中，只会涉及到单个 org.jboss.roster.Player Java 类同 Hibernate 相关。它能够映射到 CMP roster 应用中的 PlayerBean 实体 Bean。Player 对象只是简单的 POJO 对象，同 Hibernate 并无直接联系。对应的 Hibernate 映射文件 Player.hbm.xml 如下：

```
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
      <class name="org.jboss.roster.Player" table="PlayerBean">
          <id name="id" type="string" column="playerID">
                <generator class="assigned" />
          </id>

          <property name="position" type="string" column="POS"       />
          <property name="name"         type="string" column="name"     />
          <property name="salary"     type="float"    column="salary" />
      </class>
</hibernate-mapping>
```

除了 Palyer 对象和映射文件外，开发者还需要准备 hibernate-service.xml 文件。它能够创建 MBean，从而管理 Hibernate 的运行期配置。具体如下：

```
<server>
    <mbean code="org.jboss.hibernate.jmx.Hibernate"
           name="jboss.har:service=Hibernate">
        <attribute name="DatasourceName">java:/DefaultDS</attribute>
        <attribute name="Dialect">org.hibernate.dialect.HSQLDialect</attribute>
        <attribute name="SessionFactoryName">java:/hibernate/SessionFactory</attribute>
        <attribute name="CacheProviderClass">
            org.hibernate.cache.HashtableCacheProvider
        </attribute>
        <!-- <attribute name="Hbm2ddlAuto">create-drop</attribute> -->
    </mbean>
</server>
```

Hibernate 开发者应该很熟悉这一配置信息。在运行期，JBoss 会将 java:/DefaultDS 数据源提供给 Hibernate，并且会将 Hibernate SessionFactory 绑定到 JNDI 树上。

编译和打包 Hibernate 存档的 Ant 任务如下。

```
ant -f jboss-build.xml compile
ant -f jboss-build.xml package-har
```

HAR 文件的内容如下。

```
$ jar tf jar/roster.har
META-INF/
META-INF/MANIFEST.MF
META-INF/hibernate-service.xml
org/
org/jboss/
org/jboss/roster/
org/jboss/roster/Player.class
org/jboss/roster/Player.hbm.xml
```

开发者可能会问，Hibernate 将会持久化哪些对象。Hibernate 部署器会在 HAR 存档中搜索 Hibernate 映射文件。因此，JBoss 对 Hibernate 集成做了大量的工作。

9.2 使用 Hibernate 对象

在部署完成 HAR 存档后，开发者可以在应用的其他地方使用 SessionFactory。作为实例，本书创建了简单的 JSP 页面，它从 SessionFactory 创建了 Session，然后做了一次 Hibernate 查询。在实际开发应用中，开发者一般都是通过 Servlet 或 Session Bean 访问 SessionFactory。JSP 中访问 Hibernate SessionFactory 的代码如下。

```
InitialContext ctx          = new InitialContext();
SessionFactory factory    = (SessionFactory)
                                    ctx.lookup("java:/hibernate/SessionFactory");
Session             hsession = factory.openSession();
try {
      request.setAttribute("players",
            hsession.find("from org.jboss.roster.Player order by name"));
} finally {
      hsession.close();
}
```

运行 Ant package-web 任务，能够完成 Web 应用的打包。

```
ant -f jboss-build.xml package-web
```

最后，roster.war 存档将放置到 jar 目录下。

9.3 打包完整的应用

接下来，开发者需要将应用打包成 EAR 文件。具体使用的 Ant 任务是 assemble。

```
ant -f jboss-build.xml assemble
```

即，创建 HibernateRoster.ear 文件。该 EAR 文件包括 roster.har 和 roster.war 文件。另外，还有一些部署描述符。

```
$ jar tf jar/HibernateRoster.ear
META-INF/
META-INF/MANIFEST.MF
META-INF/application.xml
META-INF/jboss-app.xml
roster.har
roster.war
```

正如本文在 application.xml 文件声明 WAR 文件一样，开发者还需要声明 HAR 文件。由于 Hibernate 存档不是标准的 J2EE 部署单元，因此需要借助于 jboss-app.xml 部署描述符实现对 Hibernate 存档的声明。

```
<!DOCTYPE jboss-app PUBLIC "-//JBoss//DTD J2EE Application 1.4//EN"
            "http://www.jboss.org/j2ee/dtd/jboss-app_4_0.dtd">
```

```
<jboss-app>
    <module>
        <har>roster.har</har>
    </module>
</jboss-app>
```

至此，可以开始部署 EAR 应用了。

9.4 部署、运行应用

一旦 EAR 文件创建成功，开发者可以使用 Ant deploy 任务部署它，即将 EAR 文件拷贝到 JBoss deploy 目录。

```
ant -f jboss-build.xml deploy
```

通过http://localhost:8080/roster/players.jsp能够访问到上述应用。其中，开发者可以浏览到依据队员名排序的队员列表。如果开发者没有看到任何数据，则请检查是否在第 7 章成功部署了CMP roster应用。因此，EAR需要使用该应用创建的表及其数据。

附录 A 其他信息资料

- 进一步信息，请参考《JBoss: A Developer's Notebook》。它将由 Oreilly、于 2005 年发行。Norman Richards 和 Sam Griffith 是该书的作者。
- 有关JBoss的高级主题，请参考如下文档：《The JBoss 4 Application Server Guide》，其网址位于，http://www.jboss.org/docs/index。（译者注：中文版已经由电子工业出版社于 2004 年 11 月出版，第 3 版）。您通过Sams出版社也可以购买到它，书名是《JBoss 4.0: The Official Guide》，于 2005 年出版发行。
- 为实现运行的良好性能和高可用性，需要使用群集JBoss服务器。相关文档，请参考，《JBoss Clustering》(Sacha Labourey 和 Bill Burke)，其网址也是位于，http://www.jboss.org/docs/index。（译者注：开发者通过JBoss网站能够下载到它。）
- 有关结合 JBoss 的 EJB 教程，请参考《Enterprise JavaBeans》第四版，由 Oreilly 于 2004 年出版。该书由 Richard Monson-Haefel、Bill Burke、Sacha Labourey 共同完成。
- 至于其他的EJB教程，本书还将推荐经典的《Mastering Enterprise JavaBeans》第二版，由Wiley于 2001 年出版。该书由Ed. Roman等人写作完成。通过如下网址能够下载到其免费电子版，http://www.theserverside.com/books/masteringEJB/index.jsp。
- 有关 J2EE 1.4 Web 服务的其他内容，请参考《J2EE Web Services》，由 Addison-Wesley 于 2003 年出版。该书由 Richard Monson-Haefel 完成。
- 有关如何使用 XDoclet 以简化 J2EE 开发的资料，则请参考《XDoclet in Action》，由 Manning 于 2003 年出版。该书由 Craig Walls、Norman Richards 完成。
- 有关 Hibernate 的更多内容，请参考《Hibernate in Action》，由 Manning 于 2004 年出版。该书由 Christian Bauer、Gavin King 完成。