Java 技术丛书

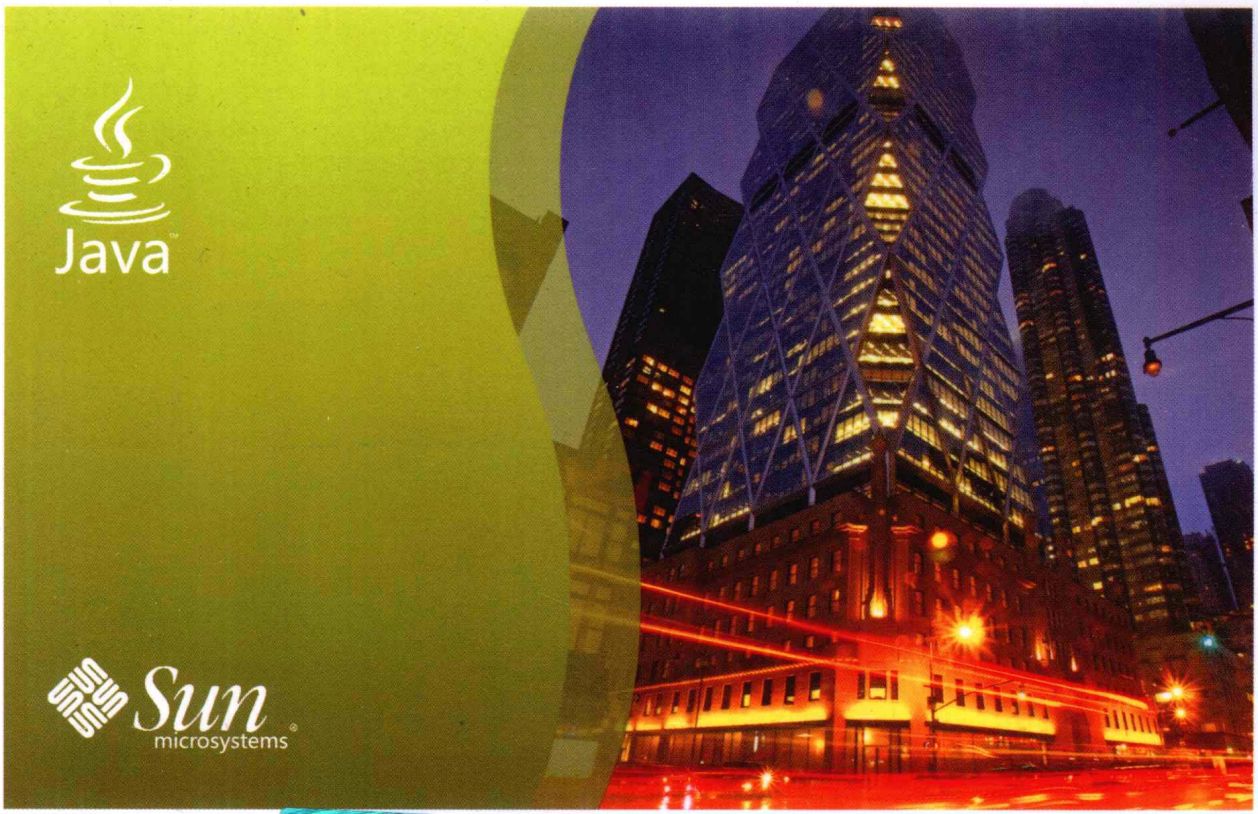
PRENTICE HALL

Servlet与JSP核心编程

（第2卷 第2版）

（美）Marty Hall Larry Brown Yaakov Chaikin 著

胡书敏 等译



PEARSON Education

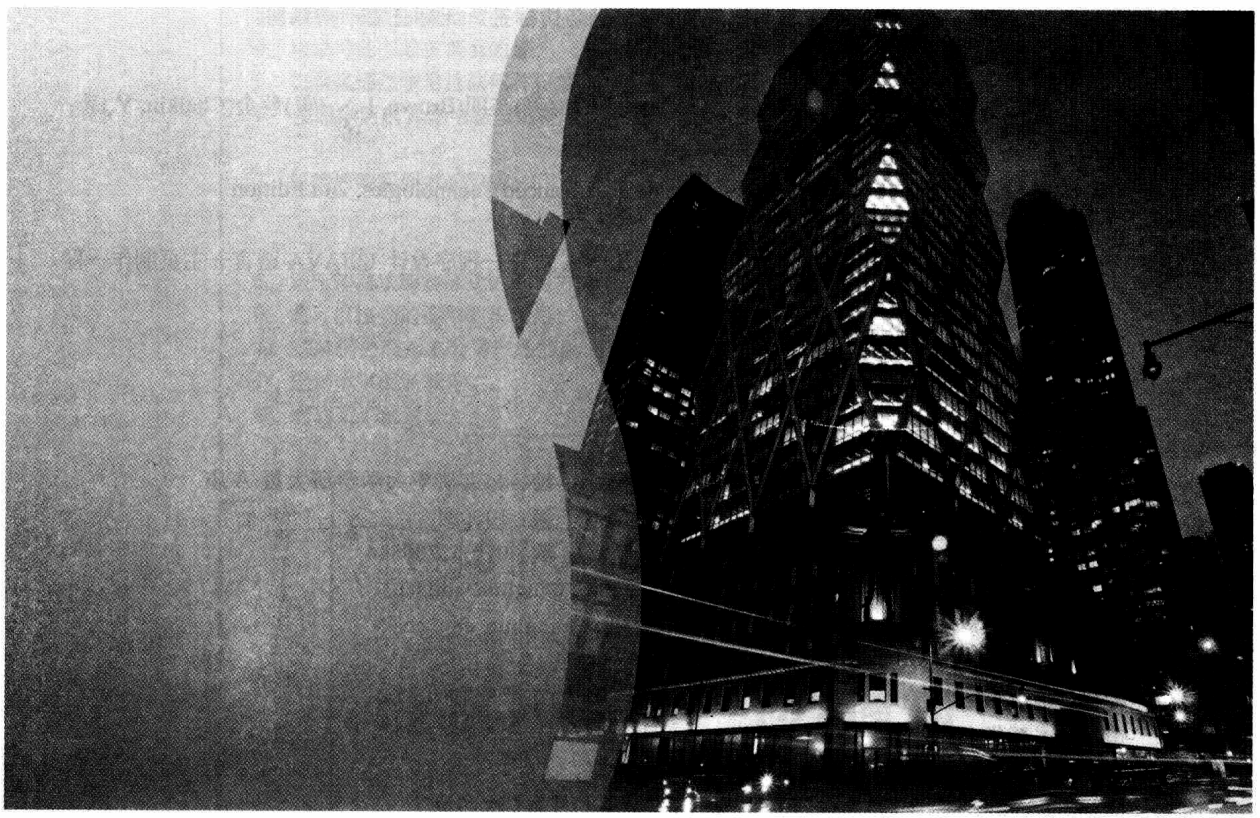
清华大学出版社

Java 技术丛书

PRENTICE HALL

Servlet与JSP核心编程（第2卷 第2版）

（美）Marty Hall Larry Brown Yaakov Chaikin 著
胡书敏 等译



清华大学出版社
北 京

内 容 简 介

Java EE 已经成为电子商务网站、动态网站和 Web 应用与服务开发的首选，作为这一平台的基础，servlet 与 JSP 的重要性日益突出，并在极短的时间内得以迅速普及。本书在第 1 卷的基础上，广泛涉及自定义标签库、过滤器、声明式安全、JSTL 和 Struts 等主题，并沿袭深受读者喜爱的写作风格，通过完整、有效、资料丰富的程序来演绎目前最流行的技术和最佳实践。

透过本书，读者可以掌握如何通过部署描述文件 web.xml 来控制 Web 应用的行为，如何通过声明式安全和编程式安全来增强 Web 应用的安全，如何使用 servlet 和 JSP 过滤器来封装 Web 应用常见的行为。同时，读者还将学会如何控制 Web 应用生命周期中的主要事件，掌握 JSTL 最佳实践，了解如何构建自定义标签库。此外，本书还用较多篇幅深入介绍了 Jakarta Struts 框架。

本书适合具有 Java 编程基本水平的读者阅读，是帮助他们提升专业技能的理想参考书。

EISBN: 978-0-13-148260-9

图书在版编目(CIP)数据

Servlet 与 JSP 核心编程(第 2 卷 第 2 版)/(美)霍尔(Hall, M.)，(美)布朗(Brown, L.)，(美)蔡金(Chaikin, Y.)著；胡书敏等译. —北京：清华大学出版社，2009.6
(Java 技术丛书)
书名原文：Core Servlets and JavaServer Pages Volume 2: Advanced Technologies, 2nd Edition
ISBN 978-7-302-20308-7

Ⅰ. S… Ⅱ. ①霍… ②布… ③蔡… ④胡… Ⅲ. ①JAVA 语言—程序设计 ②JAVA 语言—主页制作—程序设计 Ⅳ. TP312 TP393.092

中国版本图书馆 CIP 数据核字(2009)第 085655 号

责任编辑：文开琪
装帧设计：杨玉兰
责任印制：孟凡玉
出版发行：清华大学出版社 **地 址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京鑫海金澳胶印有限公司
装 订 者：北京市密云县京文制本装订厂
经 销：全国新华书店
开 本：185×260 **印 张**：33.75 **字 数**：816 千字
版 次：2009 年 6 月第 1 版 **印 次**：2009 年 6 月第1 次印刷
印 数：1～5000
定 价：68.00 元

致　谢

本书写作过程中，得到了很多人的帮助。没有他们的帮助，我们或许还在第 2 章停滞不前。Chuck Cavaness (Cypress Care 公司)，Bob Evans (JHU 应用物理实验室), Randal Hanford (波音)，Kalman Hazins (JHU 应用物理实验室)，Michael Kolodny (Raba 科技), Kyong Park (Raba 科技)，Eric Purcell (Lockheed-Martin)，Ylber Ramadani (George Brown 学院)和 Richard Slywczak (NASA Glenn 研究中心)针对许多章，分别提出了很多有价值的反馈意见。他们的建议使得本书得以显著完善。

Teresa Horton 为我们指正了错误的标点符号、错误的语句、录入错误和语法上的问题。她对本书进行了大量的润饰。Vanessa Moore 设计了本书的版式并制作了最终版本，不管我们有多少次最终改动，她都无怨无悔地给予支持和配合。Prentice Hall 的 Greg Doench 对本书第 1 版富有信心，并鼓励我们创作第 2 版。在此一并致谢。

最重要的是，Marty 要感谢 B. J. Lindsay 和 Nathan 的耐心和鼓励。Larry 要感谢 Lee 的关爱和鼎力支持。Yaakov 希望感谢上帝的光辉、上帝的恩典和怜悯他的每一天：他的双亲 (Ilia 先生和 Galina Khaikin 女士)在 80 年代的时候带着 13 岁的他，参加了他一生中第一堂编程课；他的孩子们，Moshe 和 Esther Miriam，为他的生活带来挑战和乐趣；当然还有他的妻子 Perel，感谢她一直以来的支持和鼓励。上帝以和谐的家庭赐福于我们。

Marty Hall 是 coreservlets.com 公司的总裁。coreservlets.com 是一家提供 Java 培训和技术咨询服务的公司。同时，Marty 还在 Johns Hopkins 大学计算机科学系为在职研究生讲授 Java 和 Web 程序设计课程，并在此担任分布式计算和 Web 技术领域的主管。Marty 有 5 部著作：《servlet 与 JSP 核心编程》(第 1 版和第 2 版)、*More Servlets and JavaServer Pages* 和 *Core Web Programming*(第 1 版和第 2 版)。可通过 hall@coreservlets.com 与他联系。

Larry Brown 是美国海军研发实验室网络和系统经理。他也是 *Core Web Programming(第* 2 版)的作者。可通过 larry@lmbrown.com 和他联系。

Yaakov Chaikin 是哥伦比亚一家软件开发公司的高级顾问。除了他的本职工作以外，他还在马里兰的 Loyola 学院为计算机科学专业研究生讲授 Web 开发技术，他在此担任 Web 开发主管。他经常帮助妻子打理她的 Web/ 图形设计业务 tbiq.com。可通过 yaakov.chaikin@gmail.com 和他联系。

前　言

假设贵公司打算在网上销售产品。对您而言，数据库已经准备就绪，其中保存有各件商品的价格和库存状态。但是，数据库不会说 Web 浏览器所用的 HTTP 语言，也不会输出 Web 浏览器所需要的 HTML 格式的页面。怎么办？您如何收集“用户希望购买什么商品”这类信息？您希望针对访问者的偏好和兴趣为其定制页面，但具体如何实现？您希望在用户在贵公司网站购物时跟踪他们所选择的商品，但实现此行为需要哪些技术呢？随着网站的知名度日渐提升，您可能希望能压缩网页以减少带宽。对于所用浏览器不支持压缩格式的用户，如何在保证他们正常访问网页的情况下实现这一期望？针对所有这些情况，我们需要一个程序来充当浏览器和服务器端资源的“中介”。本书所讲的正式如何利用 Java 平台来实现这类程序。

“稍等，”您会说，“你不是已过一本这方面的书吗？”嗯，没错！在 2000 年的 5 月，Sun Microsystem Press 和 Prentice Hall 出版发行了 Marty Hall 的第二本书《Servlet 和 JSP 核心编程》。其畅销程度超乎所有人的预期，销量接近 10 万册(英文版)，被翻译为保加利亚语、简体中文、繁体中文、捷克语、法文、德文、希伯来语、日语、朝鲜语、波兰语、俄语和西班牙语，并被 Amazon.com 网站评为 2001 年最畅销的 5 本计算机编程图书之一。太开心啦！

自这本书出版以来，servlet 和 JSP 的使用以一种非常显著的速度在增长着，Java 2 平台已经成为开发电子商务应用、动态网站和 Web 应用与服务的技术首选。servlet 和 JSP 仍然是这一平台的基础，连接着 Web 客户端和服务器端应用。事实上，几乎所有主要的基于 Windows，Unix(包括 Linux)，Mac OS，VMS 和其他大型机操作系统的 Web 服务器都支持 servlet 和 JSP 技术，要么内置，要么通过插件。经过适当的配置，您就可以在 Microsoft IIS，Apache Web Server，IBM WebSphere，BEA WebLogic，Oracle Application Server 10g 等其他数十个服务器上运行 servlet 和 JSP。商业和开源的 servlet 与 JSP 引擎在性能上都已经得以显著提升。

我们深信，servlet 和 JSP 领域继续以很快的速度发展。因此，我们再也无法在单独的一本书中全面覆盖这项技术。本书第 1 卷全面概述了几乎所有实际项目中都可以用到的 servlet 和 JSP 相关特性。本书第 2 卷侧重于使用不太频繁但对安全 Web 应用至关重要的一些特性。具体如下。

- **部署描述文件**　通过合理使用部署文件 web.xml，可以控制 Web 应用的诸多方面，如 servlet 的预装载、限制资源访问以及控制 session 的失效时间。
- **Web 应用的安全性**　对于如今的任何 Web 应用，安全至关重要！我们可以通过 servlet 和 JSP 安全模型轻松创建登录页面和控制对 Web 资源的访问。
- **定制标签库**　定制标签显著改进了 JSP 页面的设计。通过定制标签，我们可以针对自己的商业应用，轻松开发自己的可重用的标签。除了介绍如何创建自定义标签库，

本书还介绍了标准的 Java 标签库(JSTL)。

- **事件处理** 通过事件处理框架，我们可以控制 Web 应用的初始化和关闭，识别 HTTP 会话的销毁，设置应用范围的值。
- **servlet 和 JSP 过滤器** 有了过滤器，我们可以应用一些预处理和后处理动作。例如，注册进入请求，阻止访问和修改 servlet 和 JSP 响应。
- **Apache Struts** 这个框架大大增强了可用于 servlet 和 JSP 的模型-视图-控制器(MVC)架构。更重要的是，Apache Struts 仍然是行业内应用最广的框架。

读者对象

本书的主要受众是熟悉 servlet 和 JSP 基础、希望提升的开发人员。本书讨论了许多主题，如部署描述文件、安全、监听程序、自定义标签、JSTL、Struts 以及 Ant，有的读者可能希望先选择自己最感兴趣的技术，然后再阅读其他的主题。大多数商用 servlet 和 JSP Web 应用几乎都采用了本书所介绍的这些技术，所以建议读完整本书。

对于 servlet 与 JSP 新手，建议阅读本书第 1 卷。本书不仅介绍了如何安装和配置 servlet 容器，还很好地阐述了 servlet 与 JSP 规范。它是本书第 2 卷的基础。

本书第 1 卷和第 2 卷都假设读者已经具有一些基本的 Java 编程知识。本书不要求读者一定要是专业的 Java 开发人员，但如果对 Java 编程一无所知，建议选择其他书入门。毕竟，servlet 与 JSP 技术应用的是 Java 语言。如果不了解 Java 语言，就无法使用 servelt 与 JSP。因此，对于不了解 Java 编程基础，建议阅读一些优秀的入门书，比如《Java 编程思想》，《Core Java》或《Core Web Programming》，这些书都是 Prentice Hall 出版发行的。

本书配套网站

本书有一个配套网站，网址为 *http://volume2.coreservlet.com*。这个免费的配套网站包括如下内容。

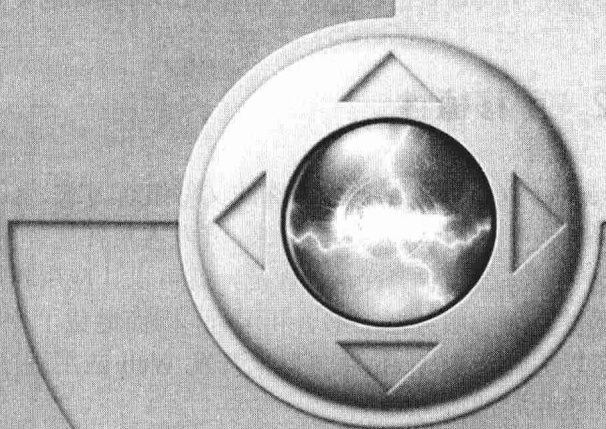
- 书中所有实例的源代码，供读者下载，并随意使用。
- 书中所提到的所有 URL 资源链接地址。
- servlet 和 JSP 软件的最新下载站点。

目　录

使用和部署 Web 应用

本章主题

- Web 应用的用途
- Web 应用的结构
- Web 应用的注册
- 开发策略和部署策略
- WAR 文件
- Web 应用的数据共享



通过 Web 应用程序，我们可以把一系列 servlet、JSP 页面、标记(tag)类库、HTML 文档、图像、样式表和其他的 Web 内容打包到一个单独的集合中，此集合可以在任何一个支持 servlet 规范的服务器上使用。如果经过精心设计，Web 应用程序可以在不同的服务器或者同一台服务器的不同位置上成功运行，无需修改 Web 应用程序中的任何 servlet、JSP 页面、标签类库和 HTML 文档。

通过这种特性，我们可以轻而易举地转向复杂的应用程序，有效促进应用程序的重用。另外，每一个 Web 应用程序都有自己的目录结构、会话、ServletContext(上下文环境)和类装载器，如此一来，我们甚至还能使用 Web 应用程序简化最初的部署，因为它减少了整个系统的各个部分之间所需要的协调工作。

1.1 Web 应用的用途

Web 应用程序主要可以在三个方面提供帮助：组织资源、轻松部署应用程序和避免不同应用程序之间相互影响。接下来让我们详细看一下每一个好处。

1.1.1 组织资源

Web 应用程序的第一个优点就是我们可以清楚地知道每一个资源在哪里。针对每个资源，Web 应用程序都有一个标准的存放位置。各个单独的 Java 类文件总是放在一个名为 WEB-INF/classes 的目录中，JAR 文件(一组被压缩的 Java 类文件)总是存放在 WEB-INF/lib 目录中，web.xml 配置文件总是放在 WEB-INF 目录中，等等。Web 应用程序的顶级目录或者其中子目录(WEB-INF 目录除外)中的文件可以直接被客户端(如 Web 浏览器)访问。

另外，开发人员从一个项目把文件移动到其他项目是非常常见的。得益于这样一种标准的 Web 应用程序资源组织方法，我们能够在启动一个新项目时无需手动建立程序结构，而且在有新成员加入项目时，他也无需重新学习特殊的文件组织方式，因为他也熟悉这个标准。

1.1.2 可移植性

因为 servlet 规范提供了一个特殊的文件组织方式，任何遵循此规范的服务器都应该能够立刻部署和运行应用程序。这样一来，开发人员在选择 Web 服务器提供商时就有很大的自由。只要这个服务器遵循 servlet 规范，就可以让应用程序在无需修改的情况下在不同的 Web 服务器供应商的服务器上部署和运行，因而避免了令人生畏的“供应商垄断行为”[①]。例如，我们可以先在免费 Web 服务器上开发 Web 应用程序，到部署的时候再将其部署到供应商提供的更稳定的服务器上。

① 原文为 verdor lock-in，这里的意思是指，任何一个供应商都可以根据标准组织制定的 API 标准来开发自己的产品。

1.1.3　相互独立

部署在同一个服务器上的不同 Web 应用程序之间不会相互影响。每一个应用程序都有自己可供访问的统一资源定位地址(URL)和 ServletContext 对象等。在同一个服务器上部署的两个应用程序在行为上各自独立，如同被部署在不同的服务器上一样。两者不需要知道彼此的存在。

这进一步简化了 Web 应用程序的开发和部署。开发人员不必关心他开发的应用程序如何与服务器上现有的 Web 应用程序集成。现在，Web 应用程序之间可以通过几种方法进行彼此间的交互，本章稍后将对此进行详述。然而，最重要的是，我们可以独立开发各个 Web 应用程序。

1.2　Web 应用的结构

如前所述，Web 应用程序有一个标准的格式，并且能够在所有兼容的 Web 或应用程序服务器上运行。Web 应用程序的顶级目录可以是开发人员命名的一个目录。此目录内包含特定的目录结构，分别存储着各种类型的内容。本小节将详细介绍内容的类型及其合适的存储位置。

1.2.1　不同文件类型的存储位置

图 1.1 显示了一个典型的 Web 应用程序层次结构。我们提供了一个循序渐进的(step-by-step)案例来指导你动手创建一个 Web 应用程序。可以从 *http://volume2.coreservlets.com/*下载 app-blank Web 应用程序，并按照 1.6 节的步骤建立一个简单的 Web 应用程序。

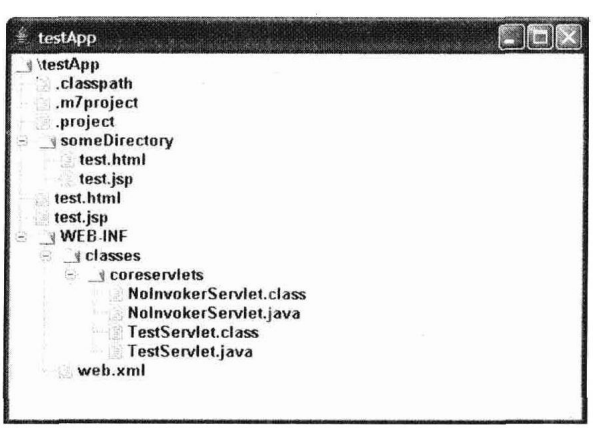


图 1.1　一个典型的 Web 应用程序

1.2.2 JSP 页面

JSP 页面应该放在 Web 应用程序的顶级目录中或者任何子目录中(WEB-INF 或 META-INF 除外)。服务器禁止用户从客户端访问 WEB-INF 或 META-INF 目录资源。在注册一个 Web 应用程序时(详见 1.3 节)，需要将 URL 前缀告诉服务器，此前缀指定了 Web 应用程序并定义了 Web 应用程序所在的目录。将 Web 应用程序主目录的名称用作 URL 前缀，这是很常见的，但系统并不强制这样做。一旦注册前缀，我们即可通过外部的 URL 地址 http://*host/WebAppPrefix/filename*.jsp(如果 *filename*.jsp 在 Web 应用程序的顶级目录中)或 http://*host/WebAppPrefix/subdirectory/filename*.jsp 访问 JSP 页面。

如果开发人员没有在 WEB-INF/web.xml 文件中定义网站的默认主页，那么在用户仅仅输入网站主目录的 URL(如 http://host/WebAppPrefix/)访问页面的时候，默认主页将依赖于服务器的默认主页文件设置，比如 index.jsp。如果希望将 index.jsp 设置为默认主页名称，我们强烈建议开发人员在网站配置文件 web.xml 中明确定义一个 welcome-file-list。例如，在下面的 web.xml 文件中，明确定义了如果用户只输入目录的 URL 而不是文件的 URL，那么服务器应该首先尝试查找 index.jsp，如果找不到，再尝试查找 index.html 作为默认主页。如果两个文件都无法找到，则返回服务器设置(如目录列表)。

```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

关于 web.xml 的使用，请参见第 2 章。

1.2.3 HTML 文档、图像和其他常规的 Web 内容

就像 servlet 和 JSP 页面一样，HTML 文件、GIF 和 JPEG 图像，样式表和其他 Web 文档同样遵循与 JSP 页面相同的规则。它们被准确地放在同样的位置，而且可以通过同一种形式的 URL 来访问。

1.2.4 单独的 servlet，bean 和 Helper 类

servlet 和其他后缀为.class 的类文件都放在 WEB-INF/classes 目录或者在 WEB-INF/classes 的子目录中，而且目录名称和它们的 package 名相匹配。

为了访问其中的 servlet，需要我们在 Web 应用程序的 WEB-INF 目录中的 web.xml 部署配置文件的 servlet-mapping 元素中为其指明 URL。具体内容可以参考 1.3 节。

除了上述方法以外，还有一种方法可以无需指定 URL 就能访问 servlet。URL 的访问形式

为 http://*host*/*webAppPrefix*/servlet/packageName.Servlet-Name。在打算进行兼容性测试或制定一个快速测试用例的时候，用这种方法访问 servlet 是没有问题的。尽管如此，我们仍然建议不要针对实际应用程序采用这种方法。有四大原因。第一，如果同时指明这个 servlet 的 servlet-mapping，我们将有两种不同的方法访问同一个 servlet。这样做会给以后的维护带来很大的麻烦。第二，因为公开的安全性依赖于访问资源的 URL。这也可能成为 Web 应用程序的潜在漏洞。第三，这样做将会强制用户在浏览器中输入 servlet 所对应的符合完全合格名称的 URL 地址。这样的 URL 地址看起来很费劲，记起来更费劲。因此，这种方法会降低 Web 应用程序的易用性。第四，一旦要改变类名或者 package 名，这个 URL 也不得不跟着改变，迫使开发人员更新 Web 应用程序中所有对此 servlet 的地址引用。除了维护成本问题，这个问题还会让已经将此 URL 地址添加到收藏夹的用户很困惑，如此一来，进一步降低了 Web 应用程序的易用性。

事实上，我们建议开发人员明确阻止用户在没有映射自定义 URL 的时候通过原始地址访问 servlet。这个映射可以借助于 web.xml 文件的 servlet-mapping 元素进行指定。这样的例子可以参见 app-blank 示例程序中的 web.xml 文件。可以从 *http://volume2.coreservlets.com/* 下载 app-blank 示例程序。

1.2.5　servlet、bean 和 Helper 类(打包到 JAR 文件中)

如果 servlet 和其他的类被一同打包在 JAR 文件中，那么该 JAR 文件应该放在 WEB-INF/lib 中。如果所有的类都在 package 中，那么当它们要打包到 JAR 文件中的时候，应该放在与其 package 名同名的目录中。在大部分服务器上，JAR 文件可以被多个 Web 应用程序共享。尽管如此，这种特性并不标准，而且在不同的服务器上，具体实现细节可能有所差别。在 Tomcat 服务器上，应该把共享的 JAR 文件放在 tomcat_dir/shared/lib 目录中(其他服务器可能有点差别，请参照服务器说明)。

1.2.6　部署描述文件

部署描述文件，即 web.xml，应该放在网站主目录的 WEB-INF 子目录中。关于 web.xml 的详细用法，请参考第 2 章。注意，有一些服务器有一个应用于所有 Web 应用的全局 web.xml 文件。该文件是整个服务器的配置说明。每一个 Web 应用中唯一标准的 web.xml 文件是每一个 Web 应用独有的，都被放在 WEB-INF 目录中。

1.2.7　Tag Library Descriptor 文件

Tag Library Descriptor (TLD，**标记类库描述文件**)文件应该放在 WEB-INF 目录或其子目录中。然而，我们强烈建议把 TLD 文件放在 WEB-INF 目录中的 tlds 目录中。把它们单独放在一个公共的目录中(如 tlds 目录)可以简化对它们的管理。JSP 页面可以通过一个 taglib 指

令来访问 WEB-INF 目录中的 TLD 文件，如下所示：

```
<%@ taglib uri="/WEB-INF/tlds/myTaglibFile.tld" ...%>
```

因为是访问 TLD 的服务器，而不是客户端的 Web 浏览器，所以对服务器而言，对客户端通过外部 URL 访问 WEB-INF 目录的限制无效。

如果要将.tld 文件部署到一个 JAR 文件中，那么.tld 文件应该放在 META-INF 目录或者其子目录中。之所以要在打包前把.tld 文件从 WEB-INF 移到 META-INF，是因为 JAR 文件不是 Web 应用程序的结构，因此没有名为 WEB-INF 的目录。如果要了解 TLD 文件的详细内容，请参考第 7 章。

1.2.8 tag 文件

tag 文件应该放在 WEB-INF/tags 目录或其子目录中。虽然 tag 文件放在受保护的 WEB-INF 目录中，但它仍然可以像 TLD 文件一样，可以由 JSP 页面从内部访问。tag 文件在 JSP 页面中也是通过 taglib 指令声明的。不过仍然有一些变化。tagdir 属性替代了原来的 uri 属性。例如，如果我们把 myTagFile.tag 文件放在 Web 应用的 WEB-INF/tags 中，那么 JSP 页面的 taglib 指令看起来就应该像下面这样：

```
<%@ taglib tagdir="/WEB-INF/tags" ...%>
```

在这个场景中，服务器自动为 tag 文件生成 TLD，所以不需要自定义映射。

也可以把 tag 文件打包到 JAR 中。如前所述，JAR 文件本身应该放在 WEB-INF/lib 目录中。但是，tag 文件应该放到 JAR 文件的 META-INF/tags 目录中。在这个例子中，服务器没有自动生成 TLD。开发人员必须手动声明 tag 文件，并在.tld 文件中定义它的路径。注意：.tld 文件还可以包含其他自定义类型的 tag。关于 tag 文件的详细内容，请参考第 7 章。

1.2.9 WAR Manifest 文件

在创建一个 WAR 文件(参见 1.5 节)的时候，有一个 MANIFEST.MF 文件会被放入 META-INF 子目录。一般情况下，jar 工具会自动创建 MANIFEST.MF，并将其放在 META-INF 目录下，而且我们可以在解压 WAR 文件的时候忽略它。不过在个别情况下，开发人员会明确修改 MANIFEST.MF 文件，所以知道其存储位置还是很有用的。

1.3 在服务器上注册 Web 应用

如前所述，Web 应用程序是可以移植的。即使更换了服务器，开发人员也可以把文件存储在同一个目录结构中，而且仍然可以通过同一种形式的 URL 来访问它们。例如，图 1.2 展示了一个简单的 Web 应用程序 myWebApp 的目录结构以及访问其资源的 URL 地址。本小

节将举例说明如何在不同的平台上安装和执行这个简单的 Web 应用程序 myWebApp。

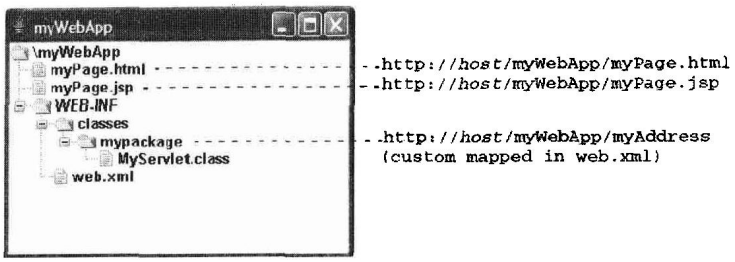


图 1.2　myWebApp Web 应用程序的目录结构

虽然 Web 应用程序本身是完全可移植的，但它在服务器上的注册过程却依赖于服务器。比如，在把 myWebApp 从一个服务器移到另一个服务器上的时候，开发人员不需要修改图 1.2 所示目录内的任何东西。然而，顶级目录(在本例中是 myWebApp)的位置可能根据服务器的不同而不同。类似的，需要使用服务器特定的过程来告诉当前系统 http://*host*/*myWebApp*/ 打头的 URL 应该应用于 Web 应用程序。

本小节假设读者已经知道如何安装和配置服务器。要了解如何配置服务器，可以阅读服务器帮助文档，查看本书第 1 卷的“前言”，或者访问 *http://www.coreservlets.com/*，获取 Tomcat 安装和配置指导。这里要通过一个简短的例子讲述 Tomcat 服务器的详细信息。如果想学习一个在 Tomcat 服务器上开发和部署 Web 应用程序的循序渐进(step-by-step)完整案例，请阅读 1.6 节。

正如 1.4 节将展示的一样，通常的策略是在个人开发环境中创建和开发 Web 应用程序，然后定期将它们复制到不同服务器的各个部署目录中进行测试。我们的建议是，尽量避免将开发目录直接复制到服务器的部署目录上进行测试。这样做的后果是：很难进行多服务器部署；在 Web 应用程序正在运行时无法部署；很难组织文件等。推荐的做法是，应该将其部署到一个单独的部署目录中，并且按照 1.4 节介绍的策略进行部署。最简单的方法是保存一个到不同服务器的部署目录的快捷方式(Windows 系统)或者符号链接(UNIX/Linux)，此后在打算部署测试的时候，只需将整个开发目录复制过去即可。例如，在 Windows 系统上，可以按住 Ctrl 键，然后在开发文件夹上单击鼠标左键，将其拖到要部署的其他服务器部署目录的快捷方式，或者使用鼠标右键将开发文件夹拖到快捷方式，松开鼠标按键，然后选择 Copy(复制)命令即可。

1.3.1　在 Tomcat 上注册一个 Web 应用

在 Tomcat 上，要创建一个 Web 应用程序，我们可以创建一个合适的目录结构，然后把它复制到 tomcat_dir/Webapps 目录中。余下的工作由 Tomcat 完成。这种直接复制整个目录结构到服务器目录的做法通常叫“热部署”(hot-deployment)或“自动部署”(auto-deployment)。服务器上允许这种功能的目录被成为“热部署目录”(hot-deploy directory)或“自动部署目录”(auto-deploy directory)。但是，并不是所有的服务器都有这样的功能。所以，有时还需

要手动修改与要部署的 Web 应用程序相关的 tomcat_dir/conf/server.xml(一个 Tomcat 说明文件)。

下面的步骤介绍了如何创建一个可以通过 http://host/myWebApp/来访问的 Web 应用程序。

(1) **创建一个 Web 应用程序的目录结构，其顶级目录为 myWebApp 目录** 因为这是我们个人的开发目录结构，所以可以随意将其放在计算机的任何目录中。myWebApp 目录中的结构应该像 1.2 节中展示的那样。如果不想手动创建这些内容，可以从 *http://volume2.coreservlets.com/*下载名为 app-blank 的 Web 应用程序。它已经包含所需要的所有目录结构和一个 web.xml 部署描述示例。我们只需要将顶级目录名称重命名为 myWebApp。

然而，如果决定自己手动创建这样的目录结构，请按照下面的步骤来做。首先，在服务器系统安装目录以外的任何地方创建一个名为 myWebApp 的目录(如果是 Windows 系统，则可以在“开始”|“运行”中输入**%systemroot%**查看系统的安装目录)。然后，在根目录 myWebApp 中创建一个名为 WEB-INF 的目录。在 WEB-INF 目录中创建一个名为 classes 的目录。创建一个 web.xml 部署描述文件，然后把它放到 WEB-INF 的目录中。我们将在第 2 章详细讨论部署描述文件。现在只需要从 *tomcat_dir* /Webapps/ROOT/WEB-INF 目录中或者从 app-blank 项目中复制现有的 web.xml 文件。

创建合适的目录结构之后，请在 myWebApp 目录中放一个名为 myPage.jsp 的简单的 JSP 页面，然后把一个名为 MyServlet.class 的简单的 servlet 文件放到 WEB-INFO/classes 目录中。

(2) **通过修改 web.xml 部署描述符文件来声明一个 servlet 并将其映射到一个 URL** 与 JSP 文件不同的是，servlet 需要显式声明。对于这个 servlet，我们需要通过提供其完全合格类名来告诉服务器此 servlet 是存在的。另外，我们需要通知服务器用户应该使用哪些 URL 来调用 MyServlet.class 类。这两个步骤都可以通过修改 web.xml 的相关元素节点来实现：

```
<servlet>
  <servlet-name>MyName</servlet-name>
  <servlet-class>mypackage.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyName</servlet-name>
  <url-pattern>/MyAddress</url-pattern>
</servlet-mapping>
```

servlet 元素及其子元素将告诉服务器我们自定义的 servlet 名和 servlet 类的完全合格类名。servlet-mapping 元素及其子元素将告诉服务器当用户请求一个与 url-pattern 元素内容相匹配的 URL 的时候，应该调用哪一个 servlet。因此，在这个例子中，我们可以使用 http://*host*/myWebApp/MyAddress 来调用被声明为 MyName 的 servlet。

(3) **复制 myWebApp 目录到 tomcat_dir/Webapps** 例如，假设我们把 Tomcat 安装在

C:\tomcat_dir。然后把 myWebApp 目录复制到 Webapps 目录，会导致 C:\tomcat_dir\Webapps\myWebApp\HelloWebApp.jsp, C:\tomcat_dir\Webapps\myWebApp\WEB-INF\classes\HelloWebApp.class 和 C:\tomcat_dir\Webapps\myWebApp\WEB-INF\web.xml 这样的目录。也可以把这个目录打包到一个 WAR 文件中，然后把这个文件直接放到 C:*tomcat_dir*\Webapps 目录中。

(4) **(可选)：在 *tomcat_dir*/conf/server.xml 中添加一个 Context 元素**　默认情况下，Tomcat 服务器将自动为 Web 应用程序配置一个完全匹配于其顶级目录名称的一个 URL 前缀。如果满意这样的默认设置，则可以忽略这一步。然而，如果想自己控制 Web 应用程序的注册过程，则可以在 *tomcat-dir*/conf/server.xml 文件中提供一个 Context 元素。如果确定要编辑 server.xml 文件，那么务必牢记一点，在真正修改这个文件之前做好一个备份——因为 server.xml 文件的一个小小的语法错误足以使 Tomcat 完全不能运行，所以最好做一个备份，万一将来修改错了，还可以用备份文件替换回来。然而，对于后来新版本的 Tomcat，推荐的方法是把 Context 元素及其子元素都放在 context.xml 文件中。然后把这个文件连同 web.xml 文件一起放在 Web 应用程序的 WEB-INF 目录中。

Context 元素有一些常用属性，可以访问 *http://jakarta.apache.org/tomcat/tomcat-5.0-doc/config/context.html*，从中了解具体包含的属性信息。例如，可以决定是否使用 cookie 或者在 session 跟踪的时候是否重写 URL，也可以启用或者禁用 servlet 重新载入(系统可以自动监视类的修改，并且在这个 servlet 被访问之前重新编译和载入这个修改过的 servlet)，而且可以设置调试级别。然而，对于最基本的 Web 应用程序，只需要处理两个属性：path(URL 前缀)和 docBase(Web 应用程序的基本安装目录，如 tomcat_dir/Webapps)。关于这两个属性的具体用法，可以参见下述代码：

```
<Context path="/some-Web-app" docBase="myWebApp" />
```

注意：不应该使用/examples 作为 URL 前缀；Tomcat 已经在内置的 Web 应用程序示例中使用了此 URL 前缀。

核心警告：不要将/examples 用作 Tomcat 服务器上 *Web* 应用程序的 URL 前缀！

(5) **访问 JSP 页面和 servlet**　http://*host*/myWebApp/myPage.jsp 调用 JSP 页面，http://*host*/myWebApp/MyAddress 调用 servlet。这些 URL 假设开发人员已经修改了 Tomcat 配置文件((tomcat_dir/conf/server.xml)，从而将 80 端口用作默认的 Web 应用访问端口(默认安装的时候应该是 8080 端口，以防止与 IIS 服务器的 80 端口冲突，假设服务器上没有安装 IIS，就可以放心地将 Tomcat 默认的 8080 端口设置为 80 端口)。关于如何修改端口设置，可以访问 *http://www.coreservlets.com/*，从中获取 Tomcat 安装和配置指南。如果没有将 8080 端口修改为 80 端口，则应该使用 *http://host:**8080***/myWebApp/myPage.jsp 来访问 JSP 页面，使用 *http://host:**8080***/myWebApp/MyAddress 来访问 servlet。