64

TP3/3oP2
B876

Professional JSP Tag Libraries

JSP标志库编程指南

〔美〕 Simon Brown 著

邱仲潘 等译

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 提 要

本书旨在帮助新老JSP开发人员学习和充分利用JSP标志库这项激动人心的技术。它不仅为人们编写JSP应用程序带来了巨大的变革，而且通过JSP标志扩展把复杂操作简单化并使之可以随意使用。本书一方面详细介绍了标志API的所有特性，一方面围绕复用性、可读性和维护性，介绍了定制标注的开发原理，同时给出了大量实际范例。

相信有志于利用JSP进行开发的人们定会从这本指南中找到适合自己的编程方法与努力方向。

Copyright©2002 Wrox Press. All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

本书英文版由Wrox公司出版，Wrox公司已将中文版独家版权授予电子工业出版社及北京美迪亚电子信息有限公司。未经许可，不得以任何形式和手段复制或抄袭本书内容。

版权贸易合同登记号：01-2002-1755

图书在版编目（CIP）数据

JSP标志库编程指南/（美）布朗（Brown，S.）著；邱仲潘等译. - 北京：电子工业出版社，2002.10
书名原文：Professional JSP Tag Libraries
ISBN 7-5053-7994-1

Ⅰ. J… Ⅱ. ①布… ②邱… Ⅲ. JAVA语言—主页制作—程序设计 Ⅳ. TP393.092

中国版本图书馆CIP数据核字（2002）第067793号

责任编辑：李 莹
印 刷：北京天竺颖华印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
北京市海淀区翠微东里甲2号 邮编：100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：23.375 字数：590 千字
版 次：2002年10月第1版 2002年10月第1次印刷
定 价：37.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换，若书店售缺，请与本社发行部联系。联系电话：（010）68279077

前　言

欢迎阅读《JSP标志库编程指南》。本书旨在帮助新老JSP开发人员学习和充分利用这个激动人心的技术，因为这个技术使JSP应用程序编写产生了巨大的变化。

Java Servlets从一出现就成为开发可伸缩的和安全的Web组件与应用程序的重要服务器方技术。随着JavaServer Pages（JSP）的引入，这两种技术已经成为最重要和最广泛使用的服务器方技术，建立的Web应用程序不仅是可伸缩的和安全的，而且设计合理、灵活维护。

JSP标志扩展（或定制标志）是JSP规范的重要补充，意味着可以把复杂操作包装在简单的XML标志中，并在任何JSP页面中使用。把这些标志组织成标志库，更可以在其他页面、其他Web应用程序和其他组织中使用和复用。到目前为止，这类操作通常还是在页面的小脚本中用Java代码编写的。尽管也能达到目的，但在页面中嵌入Java代码（和逻辑）会造成维护性、复用性、扩展性和可读性方面的问题。

定制标志机制允许编程人员从页面中删除这些代码，将其变成可复用组件，从而增加代码的维护性、复用性，并提高页面的可读性。

本书将介绍标志API的所有特性，包括：

- 简单标志和用属性进行定制的功能
- 通过程序求值体内容的标志
- 在页面中引入脚本变量的标志
- 多次求值体内容以提供迭代的标志
- 操纵和变换体内容的标志
- 在页面上协作的标志

本书的每一章都在强调维护性与复用性，目的是显示定制标志给JSP项目带来的大多数好处。但是，除此之外，还有一些章节专门介绍标志开发的理论方面，包括：

- 何时使用定制标志，何时用JavaBeans包装JSP页面使用的功能
- 面向对象和基于组件开发与设计定制标志的关系
- 目标用户如何影响标志与标志库的设计和开发
- 与标志寿命周期有关的最佳做法
- 如何使用验证手段提供更壮实的标志库
- 部署选项和部署时常见的错误

本书介绍各个课题和技术时，给出了实际代码示例，用以演示所阐述的原理。只要你跟随我们一直到本书结束，定会使用定制标志方便地编写更可维护的、更可读的JSP页面，同时建立有用的、可复用组件库。

本书内容

本书从简单的JSP技术入手，包括可读性、维护性、功能和易开发性等方面的优缺点，

介绍为什么需要JSP标志，标志与JavaBeans、小脚本等其他动态生成数据的机制如何选择。

书中假设读者对小服务Servlet和JSP有一定了解，但在需要时我们仍会做一简单复习。

第2章着手编写第一个标志，介绍标志存命周期，定义标志存命周期的接口和标志所处的环境，包括介绍标志可用的资源、范围变量和隐式JSP变量。

然后介绍如何定义标志库，亦即功能相互联系的标志集合，并介绍如何部署第一个标志。最后，介绍标志属性，这是向标志传递信息的机制，用以定制标志的工作，从而使标志更灵活。

第3章介绍标志与其体内容的交互；标志的体内容可能对标志来说具有特殊含义，它可以由标志操纵，在页面中生成动态内容。这样做的目的是加密站点的某个部分，重复与迭代数值，并通过程序包括和排除页面内容。

第4章介绍脚本变量。如果你熟悉JSP，则知道脚本变量就是JSP页面中可以使用的数据。JSP标志可以在页面中引入脚本变量，并可以被定制，以传入作为属性值的脚本变量。这一章将着重介绍JSP标志与页面中本地数据交互的方式。

第5章介绍使用变量进行迭代。在没有JSP标志时，对数据集合进行迭代通常会使页面变得更为混乱。而使用标志对数据集合迭代，可以使JSP页面保持整洁，使页面设计员只修改站点外观而不会破坏重要功能。

体标志是一种有条件地包括体内容并对标志体内容进行迭代的机制。第6章将介绍这种体标志机制的用途，它比迭代复杂，但具有更强大的最佳效果。

第7章介绍标志间的协作。到目前为止，本书只介绍了标志如何与所在页面进行交互。然而何时利用标志之间的相互交互技术来得到更大的灵活性，是我们在这一章将介绍的。

第8章介绍标志模式。标志的强大使一个问题可以有多种解。选择不同的解需要靠经验，因此我们力求寻找使用标志的最合适情形，更好地设计标志之间的相互交互，避免生成格式化数据时可能出现的问题，同时学会命名规则、使用资源捆绑、将标准HTML标志转换成JSP标志，最后掌握如何访问页面上的企业资源，包括访问EJB和消息结构。

第9章介绍标志设计原理、面向对象设计原理和基于组件的开发，介绍如何建立好标志，如何处理标志中的异常条件。此外，还要详细介绍如何更好地用属性提高定制、复用性和性能，介绍标志的性能取舍，同时对使用标志和传统子脚本JSP进行开发做性能比较。

第10章介绍页面上标志使用的验证和标志部署，介绍如何使不熟悉编程概念的人能使用我们的标志，介绍标志提供的功能粒度、标志名和限制页面中滥用标志的方法。本章下半部分介绍部署，包括如何减少依赖性，如何用属性文件和小服务情境初始化来定制标志的后部署，最后介绍测试多个容器和查错提示信息。

第11章将本书前面的内容综合起来，建立一个分析案例，以显示全部用标志建立完全实用的Web站点的过程。

最后，第12章介绍一些第三方库。使用第三方库非常方便，可以节省大量工作，有时还会更可靠、更稳定；其中包括Struts库、Jakarta Tag库与JSPTL推荐标准标志库。

规则

本书用几种文本和布局样式区别不同类型的信息。下面是一些样式例子及其说明。

强调项用缩排方式标出，重点词语用黑体字给出。

代码有两种样式。如果是可以作为程序输入与运行的代码块，则用普通程序体表示：

```
public void someFunction() {
  return someValue;
}
```

有时出现混合样式代码：

```
// If we haven't reached the end, return true, otherwise
// set the position to invalid, and return false.
pos++;
if (pos < 4)
   return true;
else {
   pos = -1;
   return false;
}
```

这里，未加黑的代码为已经介绍过的代码，而用粗体给出的代码是新增的代码。

建议、提示和背景信息用楷体字给出。

重要信息用黑体字给出。

方法、属性等的语法用法用下列格式演示：

```
javac [-classpath .] MyTag.java
```

其中方括号表示可选参数。

技术支持

如果读者想向专家直接了解书中的问题，可以发E-mail到support@wrox.com，并在主题字段中提供书名和ISBN号的最后四位与页号（本书最后4位为6217）。典型的E-mail应包含下列内容：

- 在主题字段中提供书名和ISBN号的最后四位与有问题的页号
- 在消息体中包含你的姓名、联系方式和有关问题

我们不会向你发垃圾邮件，但我们需要细节，以节省双方的时间。发送E-mail消息时，它会经过下列支持链：

此为试读，需要完整PDF请访问：www.ertongbook.com

- **客户支持**——消息发送给客户支持人员，他们是第一读者，具有常见问题的文档，会立即回答基于该书或Web站点的问题。
- **编辑**——较深的问题会转交给负责该书的技术编辑。他们具有编程语言或特定产品的经验，可以回答该主题的详细技术问题。问题解决之后，编辑会把勘误发表到Web站点。
- **作者**——最后，如果编辑无法回答你的问题，则会把它转交给作者。我们尽量不分散作者的创作精力，但我们乐意把特定请求转交给他们。所有Wrox作者都会对自己的书提供帮助与支持。他们会向客户和编辑发送响应E-mail，使所有读者受益。

Wrox支持过程只能直接支持与所出版图书有关的问题。关于正常支持范围以外的问题，见http://p2p.wrox.com/论坛的社区清单。

p2p.wrox.com

作者和对等体讨论请加入P2P邮件清单。这个独特系统负责提供编程人员之间联系的邮件清单、论坛和新闻组，是一对一E-mail支持系统的补充。请相信，你的询问会得到许多Wrox作者和许多行业专家的审查。在p2p.wrox.com站点，可以找到不同的帮助名单，它不仅在读者阅读本书时有用，而且在以后各位开发自己的应用程序时也有用。

预订邮件清单的步骤如下：

1. 访问http://p2p.wrox.com/。
2. 从左边菜单栏中选择相应的类别。
3. 单击想要加入的邮件清单。
4. 按照指令来预订和填写E-mail地址与口令。
5. 答复收到的确认E-mail。
6. 用预订管理器加入多个清单和设置E-mail选项。

目　录

第1章 JSP与标志扩展

从表示角度看，Java 2平台企业版（J2EE）包括两种生成动态Web内容的技术——Java小服务（servlet）与JavaServer Pages（JSP）。小服务技术首先出现，随着JSP的引入，这两个功能可以得到相同的结果，两者的差别在于用不同的机制建立组件，一个建立动态Web页面以进行业务处理，一个提供Web服务。小服务在逻辑中嵌入内容并夹带服务器方处理逻辑，而JSP页面则在内容中嵌入逻辑。

JSP页面使用的语法与HTML相似，因此很容易生成动态Web页面。相反，小服务是用Java类和Java代码写成的，因此适合生成高度编程的内容。尽管我们的兴趣点是开发，但了解JSP页面和小服务这些产生动态内容的优秀技术也非常必要。但是，把应用程序层代码和要显示的内容混合起来很容易影响维护性和可读性。

本书将介绍JSP标志扩展，这是个强大的机制，能够把JSP页面中的可复用功能包装起来，用以编写维护性和可读性更好的页面。我们首先一步一步地介绍第一个标志的建立，然后介绍更高级的课题，如标志与页面间的共享信息、控制页面、访问企业资源等，最后介绍一些设计准则和思考内容。

下面首先介绍几个JSP术语：

- **定制标志**

  定制标志指的是JSP页面中使用的一种标志（由JSP开发人员定义和建立），与驱动逻辑的幕后代码相关联。

- **标志库**

  标志库是一组定制标志的集合。

介绍第一个标志的建立过程之前，先要介绍什么是标志，为什么需要标志。

本章要介绍两种产生动态内容的机制，即Java小服务和JSP页面，二者可以方便地产生动态内容。在介绍这些例子时，我们还要探讨这两种方法的一些设计与实现问题。最后，介绍如何用JavaBeans来处理其中的一些问题，如何扩展JSP标志。

JSP标志扩展通常是高级JSP课题，因此要求读者对Java小服务和JSP页面有一定的了解。在本书相关部分，我们会从标志的角度简要复习Java小服务和JSP页面问题，但这方面的更详细信息请参考Wrox公司的Professional系列丛书：

- 《Professional JavaServer Pages》第2版（ISBN 1-861004-95-8）
- 《Professional Java Servlets 2.3》（ISBN 1-861005-61-X）

Java小服务

Java小服务是Java的公用网关接口（CGI，Common Gateway Interface），通常是用Perl、C、C++与UNIX shell脚本之类的语言编写的。这些CGI程序服务于通常在Web服务器

此为试读，需要完整PDF请访问：www.ertongbook.com

外部进程中的HTTP请求，并把产生的内容发送回客户端Web浏览器。通常，每个请求都会派生新进程，并要求大量处理功能和内存资源。结果，几乎必然使得性能和伸缩性下降。

要解决这个问题，应把Java小服务放在Web容器中运行，用多个线程同时服务于许多请求。由于使用了Web容器来管理为线程服务的请求，因此也就大大减少了高负荷可能对容器所在的机器造成影响。

内容、表示与业务逻辑

不管选择何种语言，CGI程序和Java小服务通常都是用过程性方法编写的，用一个长方法（或函数）产生整个响应。当然，还可能用到其他函数、方法和对象，但通常要由单个方法进行协调和产生响应。这样，编写用Web浏览器作为表示媒介的程序时，就可以在源代码中包括下列特征：

- **静态内容**
  例如HTML，包括格式化与布局标记
- **表示逻辑**
  影响表示的代码，如变换与格式信息
- **业务逻辑**
  查找信息和进行处理的代码，如验证

简单Java小服务

为了演示上述内容，我们先看一个简单的Java小服务，寻找客户机场景并产生符合场景格式的当前日期的HTML页面。这是个非常简单的Java小服务，因此演示时不会受方案细节的影响。

用户向这个小服务发送请求时，会得到图1.1所示的HTML页面。

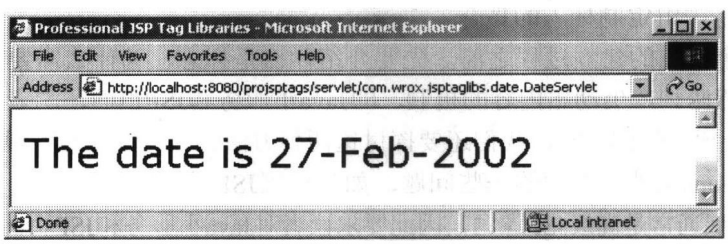

图1.1

要显示其如何用小服务实现，可以看看下列代码：

```
package com.wrox.jsptaglibs.date;

import java.io.IOException;
import java.io.PrintWriter;
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;

public class DateServlet extends HttpServlet {

  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    PrintWriter out = res.getWriter();
```

首先要从HTTP请求中寻找客户机场景，用DateFormat类本身的IT方法getDateInstance()生成适当的java.text.DateFormat实例。由于我们要使用客户机场景，因此这个小服务的输出随用户所在地的不同而不同：

```
    // get the locale of the client and create an appropriate date format
    Locale loc = req.getLocale();
    DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM, loc);
```

然后，嵌入println()语句中的静态和动态内容。<h1></h1>之类的标记用以提供格式化和布局，也放在这些语句中：

```
    out.println("<html>");

    out.println("<head>");
    out.println("<title>Professional JSP Tag Libraries</title>");
    out.println("<link rel=\"stylesheet\" href=\"../page.css\">");
    out.println("</head>");

    out.println("<body>");
    out.println("<h1>The date is ");
    out.println(df.format(new Date()));
    out.println("</h1>");
    out.println("</body>");

    out.println("</html>");

    out.flush();
    out.close();
  }
}
```

从这个简单的例子可以看出，仅用一个方法即可协调响应的生成，且其中嵌入和交织了不同类型的内容和逻辑。

注意上述小服务代码中引用了page.css。这个文件的代码可以从www.wrox.com下载。它为本例和书中其他例子的输出提供了漂亮的效果，但这里不准备展开介绍，只是告诉你可以像在文本中一样省略这些引用，也可以像在下载代码中一样包括这些引用。关于级联样式单（CSS，Cascading Style Sheets）的细节，可以参阅有关联机文档和图书。

维护性

尽管上述代码能够运行，并达到了按所要格式输出日期的目的，但仍有几个问题：

- 如果要改变内容呢？
- 如果要改变内容样式和布局呢？
- 如果要改变内容日期格式呢？

所有这些问题的答案是相同的：生成代码中的任何修改都要求修改、重新编译和重新部署小服务。尽管这一切对这个简单的例子来说不需要太长时间，但对于大型复杂页面而言，修改这类属性则需要投入大量的精力。换句话说，维护Web应用程序的难度大大增加。

如果还要修改Web站点的图形元素，那么就难以想像更新站点时修改每个小服务需要多少工作。

可读性

不仅如此，在源代码中（这里是println()语句中）嵌入内容还会影响另一个重要因素——可读性。尽管对这个简单例子来说问题不大，但那些能够产生许多复杂响应的小服务通常包含许多代码行，分别负责生成、格式化和写出HTML页面，对其进行一行一行的添加、接合和写入，会使得到的Java代码非常混乱，很难阅读。

这不是新问题，典型的CGI程序都有这个问题。Java小服务虽然解决了性能和伸缩性问题，但没有解决维护性与可读性问题。当然，小服务不是不适合建立Web应用程序，事实正好相反，它们适合作为模型/视图/控制器（MVC，Model-View-Controller）体系结构中的控制器，作为处理单元或作为过滤非验证请求的入口点。

维护性与可读性呢？这两个问题将在下一节解决。

JavaServer Pages

本章开头曾介绍过，Java小服务负责在逻辑中嵌入内容，带行服务器方处理逻辑，而JSP页面则在内容中嵌入逻辑。尽管JSP也是产生动态内容的技术，但它被写成标记的形式（如HTML、XML，等等），包含下列元素中的一个或几个：

- 小脚本
- 表达式
- 声明
- 指令
- 操作（或标志）

下面我们来一一介绍。

小脚本

小脚本是<%和%>之间的小段源代码：

```
<%
  String name = request.getParameter("name");
%>
```

尽管JSP规范目前要求JSP引擎只提供对用Java语言编写的小脚本的支持，但它将来也可能支持其他语言。

表达式

表达式与小脚本相似，但返回一个值。在<%=与%>之间嵌入Java表达式等于将表达式结果写回页面和写回客户机：

```
<%= name %>
```

表达式还可以向页面上的其他操作提供请求时的值，但我们把这个问题放到下一章介绍。

声明

就像在类中声明方法与变量一样，在JSP页面中也可以声明方法与变量。例如，在JSP页面中声明一个多处使用的方法，以避免复制和粘贴小脚本。声明应放在<%!与%>之间：

```
<%!
  private int calculate(int a, int b) {
    ...
  }
%>
```

指令

表达式、小脚本和声明定义的是运行时要执行的逻辑，而指令则负责在转换时，即把JSP页面转换成小服务时向页面提供信息。指令可以指定要导入的类，指定捕获异常的错误页以及导入标志库和让页面使用。尽管指令多种多样，但都要放在<%@与%>之间：

```
<%@ page import="java.util.Date" %>
```

操作（或标志）

JSP页面中可以包括的最后一种元素是操作（或标志），它分为定制标志与标准标志。JSP规范目前定义了几个标准操作，由符合JSP的容器提供。标准操作（或标志）用以包装可复用功能，以页面上的标志形式提供给开发人员。这些标准标志的前面均冠以jsp前缀，如<jsp:include/>与<jsp:useBean/>。稍后将会介绍使用标准标志的例子。

另一方面，定制标志是自己编写或第三方提供的JSP标志。

JSP寿命周期

JSP技术实际上是Java小服务技术的扩展，尽管JSP页面被写成标记的形式，并在其中嵌入了所有这些元素，但它们只有在转换成相应的小服务以后才能处理请求。编写JSP页面并部署到Web服务器之后，第一个请求会启动JSP寿命周期，如图1.2所示。

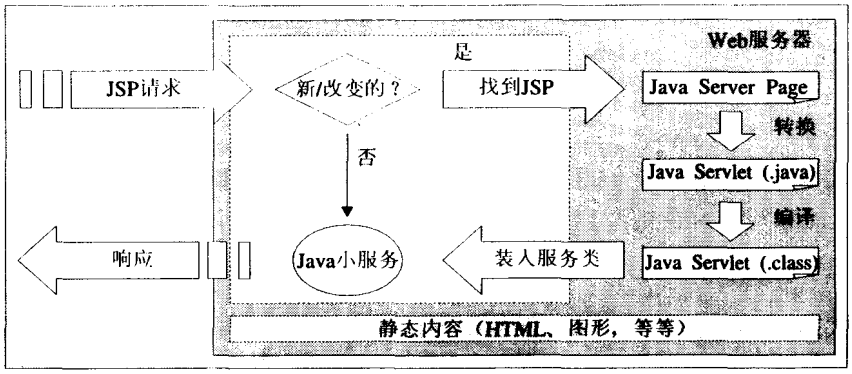

图1.2

从图1.2可以看出，请求新的或修改的JSP页面时，Web服务器要将其转换成表示相应Java小服务的Java代码。这个步骤可能相当复杂，但实际上是将整个JSP页面变成结果小服务_jspService()的一个方法。静态内容包装在out.println()调用中，而小脚本和表达式中包含的源代码可直接插入。

产生Java代码之后，要将其编译成标准类文件。然后由Web服务器装入这个类，服务于请求，最后向客户机返回响应。

尽管图中没有显示，但有些JSP兼容服务器厂家提供了将JSP页面预编译成相应Java小服务的工具。在进行部署时，实际部署的是这些包装起来的小服务类，而不是JSP页面本身。这在处理许可证和知识产权之类的问题时非常有用。

另外，Java编译器在生产环境中通常有安全风险，因为恶意代码可能在服务器中编译和执行。如果要消除转换与编译JSP页面的初始性能影响，则可以发布编译的小服务。

简单JSP

下面来看看本章前面介绍的Java小服务对应的JSP。这个JSP页面和Java小服务一样会产生响应，但这时格式化代码、寻找客户机场景的逻辑和日期格式生成代码嵌入在HTML中：

```
<%@ page language="java" isErrorPage="false" %>
<html>

  <head>
    <title>Professional JSP Tag Libraries</title>
  </head>
```

由于我们使用的是java.lang包之外的类，因此首先要用page指令进行导入：

```
<%@ page import="java.text.DateFormat" %>
<%@ page import="java.util.Date" %>
<%@ page import="java.util.Locale" %>
```

javax.servlet与javax.servlet.jsp包中的类由得到的小服务导入，因此不必显式导入。

完成之后，可以用Java小脚本从HTTP请求寻找客户机场景，并和前面的做法一样，生成相应的DateFormat实例：

```
<body>
  <%
    // get the locale of the client and create an appropriate date format
    Locale loc = request.getLocale();
    DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM, loc);
  %>
```

最后，产生页面的动态部分，为此，我们用一个简短的表达式格式化和输出日期：

```
    <h1>The date is <%= df.format(new Date()) %></h1>
  </body>
</html>
```

从Web服务器中请求这个JSP页面时，得到的HTTP页面如图1.3所示。

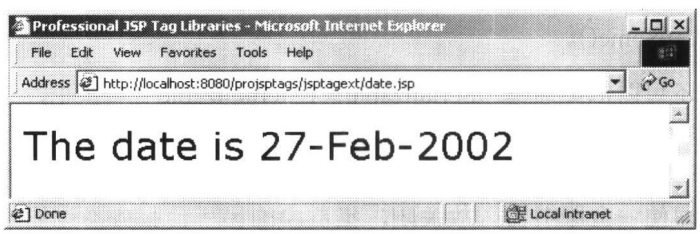

图1.3

可以看出，结果是相同的，但JSP代码不仅更短，而且更可读。

易写易破

JSP的一个优点是页面很容易编写，只需生成扩展名为.jsp的HTML文件，导入所要的Java类，并编写一些Java代码。但反过来说，JSP页面也很容易被破坏，只要不知道Java的人或工具（例如页面写作工具或Web设计工具）删除一小段Java代码，就可能使一些重要业务处理或表示逻辑丢失。此外，维护性与可读性等设计与实现问题如何呢？我们来看一下。

维护性

修改内容和表示时，JSP页面比小服务更简单。原因有两个，第一，内容的形式更合适（如HTML），第二，JSP页面的寿命周期使其很容易修改和重新编译。但是，JSP页面比小服务更容易修改并不说明维护性与可读性更好。

例如，包含大量嵌入Java代码的复杂JSP页面很容易削弱页面的维护性与可读性，这也回到了与Java小服务相同的维护性问题——应用程序代码与内容交织在一起。

此为试读，需要完整PDF请访问：www.ertongbook.com