



辽宁省“十二五”普通高等教育本科省级规划教材

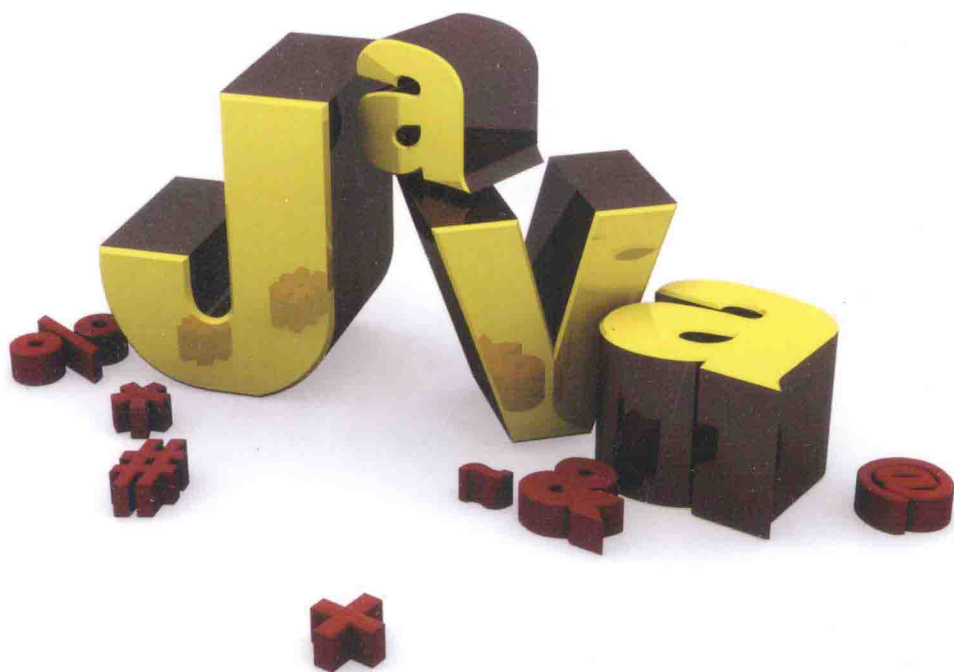


普通高等教育“十一五”国家级规划教材

高等学校Java课程系列教材

JSP 程序设计 (第2版)

耿祥义 张跃平 编著



清华大学出版社

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

备用QQ:2404062482

高等学校 Java 课程系列教材

JSP 程序设计(第 2 版)

耿祥义 张跃平 编著

清华大学出版社
北 京

内 容 简 介

JSP(Java Server Pages)是一种动态网页技术标准,它可以无缝地运行在 UNIX、Linux 和 Windows 操作平台上。利用这一技术可以建立安全、跨平台的先进动态网站。

本书详细讲解了 JSP 语法和基本的程序设计方法。全书共分 10 章,内容包括 JSP 简介,JSP 页面与 JSP 标记,Tag 文件与 Tag 标记,JSP 内置对象,JSP 文件操作,在 JSP 中如何使用 MySQL、SQL Server、Oracle 等常用数据库,JSP 与 JavaBean,Java Servlet 基础,以及 MVC 模式等重要内容。本书所有知识都结合具体实例进行介绍,力求详略得当,突出 JSP 在开发 Web 动态网站方面的强大功能及在开发商务网站方面的应用,使读者快速掌握和运用 JSP 的编程技巧。

本书不仅可以作为高等院校计算机及相关专业的选修课教材,也可作为自学者及网站开发人员的参考书。

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

JSP 程序设计/耿祥义,张跃平编著.--2 版.--北京:清华大学出版社,2015

高等学校 Java 课程系列教材

ISBN 978-7-302-37236-3

I. ①J… II. ①耿… ②张… III. ①JAVA 语言—网页制作工具 IV. ①TP312 ②TP393.092

中国版本图书馆 CIP 数据核字(2014)第 153406 号

责任编辑:魏江江 王冰飞

封面设计:杨 兮

责任校对:时翠兰

责任印制:王静怡

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

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:010-62776969, c-service@tup.tsinghua.edu.cn

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

课 件 下 载: <http://www.tup.com.cn>, 010-62795954

印 刷 者:北京富博印刷有限公司

装 订 者:北京市密云县京文制本装订厂

经 销:全国新华书店

开 本:185mm×260mm 印 张:22 字 数:536 千字

版 次:2009 年 8 月第 1 版 2015 年 1 月第 2 版 印 次:2015 年 1 月第 1 次印刷

印 数:30001~32000

定 价:39.50 元

产品编号:060267-01

前 言

JSP 是由 Sun 公司倡导、许多公司参与,于 1999 年推出的一种动态网页技术标准。JSP 是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术,利用这一技术可以建立安全、跨平台的先进动态网站,这项技术还在不断地更新和优化中。JSP 以 Java 技术为基础,又在许多方面做了改进,具有动态页面与静态页面分离,能够脱离硬件平台的束缚,以及编译后多平台运行等优点,JSP 已经成为 Internet 上的主流开发工具。

本书第 2 版对一些例子和部分内容做了适度的调整、更新和修改,考虑到数据库在 Web 设计中的重要作用,本书第 2 版加强了数据库的知识容量,特别采用了目前在 Web 设计中占主导地位的 MySQL 数据库作为主要的数据库来讲解有关知识点。本书分为 10 章,第 1 章主要介绍 Tomcat 的安装与配置,通过一个简单的 JSP 页面初识 JSP 概貌。第 2 章详细讲解 JSP 的基本语法,包括程序片,页面指令等重要内容。第 3 章主要讲解 Tag 文件与标记,特别重点强调了怎样使用 Tag 文件实现代码复用。第 4 章主要讲解 JSP 的内置对象,特别重点讲解了 session 会话对象。第 5 章讲解输入/输出流技术,重点介绍了文件的上传与下载以及怎样使用 Tag 标记实现文件的读/写操作。第 6 章涉及的内容是数据库,也是 Web 应用开发的非常重要的一部分内容,采用 MySQL 数据库讲解主要知识点,也特别介绍了各种数据库的连接方式。第 7 章讲解 JavaBean 的使用,是 JSP 技术中很重要的内容,即怎样使用 JavaBean 分离数据的显示和处理,给出了许多有一定应用价值的例子。第 8 章讲解 Servlet,对 servlet 对象的运行原理给予了详细的讲解。在第 9 章对 Servlet 在 MVC 开发模式中的地位给予了重点介绍,并按照 MVC 模式给出了易于理解 MVC 设计模式的例子,本章中的许多例子都是大多数 Web 开发中经常使用的模块。第 10 章是一个完整的网站,完全按照 MVC 模式开发设计,其目的是掌握一般 Web 应用中常用基本模块的开发方法。

希望本教材能对读者学习 JSP 有所帮助,并请读者批评指正(xygeng0629@sina.com)。

编者

2014 年 10 月

目 录

第 1 章 JSP 概述	1
1.1 什么是 JSP	1
1.2 JSP 引擎与 Tomcat 服务器	2
1.2.1 安装 JDK	2
1.2.2 安装与启动 Tomcat 服务器	3
1.3 JSP 页面与 Web 服务目录	4
1.3.1 JSP 页面	4
1.3.2 Web 服务目录	5
1.4 JSP 运行原理	7
1.5 实验：编写、保存、运行 JSP 页面	9
习题 1	10
第 2 章 JSP 页面与 JSP 标记	11
2.1 JSP 页面的基本结构	11
2.2 变量和方法的声明	12
2.2.1 声明变量	12
2.2.2 声明方法	13
2.3 Java 程序片	14
2.4 表达式	17
2.5 JSP 中的注释	17
2.6 JSP 指令标记	18
2.6.1 page 指令	18
2.6.2 include 指令标记	23
2.7 JSP 动作标记	26
2.7.1 include 动作标记	26
2.7.2 param 动作标记	27
2.7.3 forward 动作标记	28
2.7.4 plugin 动作标记	29
2.7.5 useBean 动作标记	30
2.8 实验 1：JSP 页面的基本结构	30

2.9 实验 2: JSP 指令标记	32
2.10 实验 3: JSP 动作标记	33
习题 2	37

第 3 章 Tag 文件与 Tag 标记

38

3.1 Tag 文件的结构	39
3.2 Tag 文件的存储目录	39
3.3 Tag 标记	40
3.3.1 Tag 标记与 Tag 文件	40
3.3.2 Tag 标记的使用	40
3.3.3 Tag 标记的标记体	42
3.4 Tag 文件中的常用指令	43
3.4.1 tag 指令	43
3.4.2 include 指令	45
3.4.3 attribute 指令	45
3.4.4 variable 指令	48
3.4.5 taglib 指令	52
3.5 Tag 标记的嵌套	54
3.6 实验 1: 使用标记体	55
3.7 实验 2: 使用 attribute 指令和 variable 指令	56
习题 3	59

第 4 章 JSP 内置对象

60

4.1 request 对象	61
4.1.1 获取用户提交的信息	62
4.1.2 处理汉字信息	64
4.1.3 常用方法举例	65
4.1.4 使用 Tag 文件处理有关数据	68
4.1.5 处理 HTML 标记	70
4.2 response 对象	78
4.2.1 动态响应 contentType 属性	78
4.2.2 response 的 HTTP 文件头	80
4.2.3 response 重定向	81
4.2.4 response 的状态行	81
4.3 session 对象	84
4.3.1 session 对象的 Id	85
4.3.2 session 对象与 URL 重写	86
4.3.3 session 对象存储数据	88
4.3.4 在 Tag 文件中使用 session 对象	90

4.3.5	session 对象的生存期限	92
4.3.6	使用 session 设置时间间隔	93
4.3.7	计数器	94
4.4	out 对象	96
4.5	application 对象	97
4.5.1	application 对象的常用方法	98
4.5.2	用 application 制作留言板	98
4.6	实验 1: request 对象	101
4.7	实验 2: response 对象	103
4.8	实验 3: session 对象	104
习题 4	107
第 5 章	JSP 中的文件操作	108
5.1	File 类	109
5.1.1	获取文件的属性	109
5.1.2	创建目录	110
5.1.3	删除文件和目录	111
5.2	使用字节流读/写文件	112
5.2.1	FileInputStream 类和 FileOutputStream 类	113
5.2.2	BufferedInputStream 类和 BufferedOutputStream 类	114
5.3	使用字符流读/写文件	115
5.3.1	FileReader 类和 FileWriter 类	116
5.3.2	BufferedReader 类和 BufferedWriter 类	116
5.4	RandomAccessFile 类	118
5.5	文件上传	122
5.6	文件下载	127
5.7	实验 1: 使用文件字节流读/写文件	128
5.8	实验 2: 使用文件字符流加密文件	132
习题 5	136
第 6 章	在 JSP 中使用数据库	137
6.1	MySQL 数据库管理系统	138
6.1.1	下载、安装与启动 MySQL	138
6.1.2	建立数据库	140
6.2	JDBC	145
6.3	连接 MySQL 数据库	146
6.3.1	加载 JDBC-数据库驱动程序	146
6.3.2	建立连接	147
6.3.3	MySQL 乱码解决方案	148

6.4 查询记录	150
6.4.1 顺序查询	152
6.4.2 随机查询	154
6.4.3 条件查询	158
6.4.4 排序查询	161
6.4.5 模糊查询	163
6.5 更新记录	165
6.6 添加记录	168
6.7 删除记录	170
6.8 用结果集操作数据库中的表	172
6.8.1 更新记录中的列值	172
6.8.2 插入记录	173
6.9 预处理语句	175
6.9.1 预处理语句的优点	175
6.9.2 使用通配符	177
6.10 事务	180
6.11 常见数据库连接	182
6.11.1 连接 Microsoft SQL Server 数据库	182
6.11.2 连接 Oracle 数据库	184
6.11.3 连接 Microsoft Access 数据库	184
6.12 实验 1: 查询记录	188
6.13 实验 2: 更新记录	193
6.14 实验 3: 删除记录	195
习题 6	197
第 7 章 JSP 与 JavaBean	199
7.1 编写 JavaBean 和使用 JavaBean	200
7.1.1 bean 的编写与保存	200
7.1.2 使用 bean	201
7.2 获取和修改 bean 的属性	205
7.2.1 getProperty 动作标记	205
7.2.2 setProperty 动作标记	206
7.3 bean 的辅助类	210
7.4 使用 bean 的简单例子	211
7.4.1 三角形	211
7.4.2 猜数字	213
7.4.3 日历	215
7.4.4 四则运算	218
7.4.5 浏览图片	219

7.5	JavaBean 与文件操作	221
7.5.1	读文件	221
7.5.2	写文件	223
7.5.3	上传文件	225
7.6	JavaBean 与数据库操作	228
7.6.1	查询记录	228
7.6.2	分页显示记录	231
7.7	标准化考试	235
7.8	实验 1: 有效范围为 request 的 bean	238
7.9	实验 2: 有效范围为 session 的 bean	240
7.10	实验 3: 有效范围为 application 的 bean	242
	习题 7	246
第 8 章	Java Servlet 基础	247
8.1	Servlet 类与 servlet 对象	248
8.2	编写 web.xml	249
8.3	servlet 对象的创建与运行	251
8.4	servlet 对象的工作原理	251
8.4.1	servlet 对象的生命周期	252
8.4.2	init 方法	252
8.4.3	service 方法	252
8.4.4	destroy 方法	253
8.5	通过 JSP 页面访问 servlet	253
8.5.1	通过表单向 servlet 提交数据	253
8.5.2	通过超链接访问 servlet	254
8.6	共享变量	256
8.7	doGet 和 doPost 方法	257
8.8	重定向与转发	259
8.8.1	sendRedirect 方法	260
8.8.2	RequestDispatcher 对象	260
8.9	使用 session	263
8.10	实验: 使用 servlet 读取文件	265
	习题 8	268
第 9 章	MVC 模式	269
9.1	MVC 模式介绍	270
9.2	JSP 中的 MVC 模式	270
9.3	模型的生命周期与视图更新	271
9.3.1	request 周期的 JavaBean	271

9.3.2	session 周期的 JavaBean	273
9.3.3	application 周期的 JavaBean	274
9.4	MVC 模式的简单实例	275
9.4.1	JavaBean 和 Servlet 与配置文件	275
9.4.2	计算三角形和梯形的面积	276
9.5	MVC 模式与注册登录	279
9.5.1	JavaBean 与 Servlet 管理	279
9.5.2	配置文件管理	280
9.5.3	数据库设计与连接	281
9.5.4	注册	281
9.5.5	登录与验证	285
9.6	MVC 模式与数据库操作	290
9.6.1	JavaBean 与 Servlet 管理	290
9.6.2	配置文件与数据库连接	290
9.6.3	MVC 设计细节	291
9.7	MVC 模式与文件操作	297
9.7.1	模型(JavaBean)	297
9.7.2	控制器(servlet)	298
9.7.3	视图(JSP 页面)	299
9.8	实验: 计算等差、等比数列的和	300
习题 9	304
第 10 章	手机销售网	305
10.1	系统模块构成	305
10.2	数据库设计与连接	305
10.2.1	数据库设计	305
10.2.2	数据库连接	307
10.3	系统管理	307
10.3.1	页面管理	308
10.3.2	JavaBean 与 Servlet 管理	309
10.3.3	配置文件管理	310
10.3.4	图像管理	311
10.4	会员注册	312
10.4.1	视图(JSP 页面)	312
10.4.2	模型(JavaBean)	313
10.4.3	控制器(servlet)	314
10.5	会员登录	316
10.5.1	视图(JSP 页面)	316
10.5.2	模型(JavaBean)	317

10.5.3	控制器(servlet)	317
10.6	浏览手机	320
10.6.1	视图(JSP 页面)	320
10.6.2	模型(JavaBean)	325
10.6.3	控制器(servlet)	325
10.7	查看购物车	328
10.7.1	视图(JSP 页面)	328
10.7.2	模型(JavaBean)	329
10.7.3	控制器(servlet)	330
10.8	查询手机	332
10.8.1	视图(JSP 页面)	332
10.8.2	模型(JavaBean)	333
10.8.3	控制器(servlet)	334
10.9	查询订单	336
10.9.1	视图(JSP 页面)	336
10.9.2	模型(JavaBean)	337
10.9.3	控制器(servlet)	337
10.10	退出登录	337

第 1 章

JSP 概述

本章导读

主要内容

- 什么是 JSP
- JSP 引擎与 Tomcat 服务器
- JSP 页面与 Web 服务目录
- JSP 运行原理

难点

- JSP 的运行原理
- 设置 Web 服务目录

关键实践

- 上机编写、保存、运行一个简单的 JSP 页面

1.1 什么是 JSP

网络应用中最常见的模式是 B/S 模式,即需要获取信息的用户使用浏览器向服务器发出请求,服务器对此做出响应,将有关信息发送给用户的浏览器。在 B/S 模式中,服务器上必须有所谓的 Web 应用程序,服务器通过运行这些 Web 应用程序来响应用户的请求。因此,基于 B/S 模式的网络程序的核心就是设计服务器端的 Web 应用程序。

随着网络的迅速发展,服务器不仅要和用户动态、安全地交互更多的信息,而且对 Web 应用程序的规模、难度和维护都提出了更高的要求。JSP(Java Server Pages)正是在这一背景下诞生的优秀的 Web 开发技术,利用这一技术可以建立安全、跨平台、易维护的 Web 应用程序。JSP 的跨平台、易维护 and 安全性得益于 Java 语言,这是因为 Java 语言具有不依赖于平台、面向对象、安全等优良特性,已经成为网络程序设计的佼佼者,许多和 Java 相关的技术得到了广泛的认可和应用,JSP 就是其中之一。读者可能对 Microsoft 的 ASP(Active Server Pages)比较熟悉,ASP 也是一项 Web 开发技术,可以开发出动态的、高性能的 Web 应用程序。JSP 和 ASP 技术非常相似,ASP 使用的是 VBScript 脚本语言,而 JSP 使用的是 Java 编程语言。与 ASP 相比,JSP 以 Java 技术为基础,又在许多方面做了改进,具有动态页面与静态页面分离,能够脱离硬件平台的束缚,以及编译后运行等优点,完全克服了 ASP 的脚本级执行的缺点。JSP 已经成为开发动态网站的主流技术。

需要强调的一点是:要想真正地掌握 JSP 技术,必须有较好的 Java 语言基础,以及较

好的 HTML 语言方面的知识。

注意：在 Web 设计中，用户一词通常代表客户端计算机上驻留的浏览器，即使浏览器在客户端计算机的多个窗口中被打开，服务器也认为这是一个用户。

1.2 JSP 引擎与 Tomcat 服务器

一个服务器上可以有很多基于 JSP 的 Web 应用程序，以满足各种用户的需求。这些 Web 应用程序必须有一个软件来统一管理和运行，将这样的软件称作 JSP 引擎或 JSP 容器，将安装 JSP 引擎的计算机称作一个支持 JSP 的 Web 服务器。

JSP 的核心内容之一就是编写 JSP 页面(有关 JSP 页面的内容见 1.3 和 2.1 节)，JSP 页面是 Web 应用程序的重要组成部分之一。一个简单 Web 应用程序可能只有一个 JSP 页面，而一个复杂的 Web 应用程序可能由许多 JSP 页面、JavaBean 和 servlet 组成(见第 7 章、第 8 章、第 9 章)。当用户请求 Web 服务器上的 JSP 页面时，JSP 引擎负责运行 JSP，并将运行结果返回给用户，有关 JSP 的运行原理将在 1.4 节讲解。

自从 JSP 发布以后，出现了各式各样的 JSP 引擎。目前，比较常用的 JSP 引擎包括 Tomcat、JRun 和 Resin，其中以 Tomcat 的使用最为广泛。Tomcat 由 Apache 和 Sun 公司共同开发而成，是一个免费的开源 JSP 引擎。可以登录 <http://tomcat.apache.org/> 免费下载 Tomcat。目前，Tomcat 的最新版本是 Tomcat 8.0，登录之后，可以在 Download 里选择 Tomcat 8.0，然后在 Binary Distributions 的 Code 中选择 Zip 或 32-bit/64-bit Windows Service Installer 即可。如果选择 Zip 将下载：apache-tomcat-8.0.3.zip；如果选择 32-bit/64-bit Windows Service Installer 将下载：apache-tomcat-8.0.3.exe。

本章重点讲述在 Windows 7/Windows XP 操作系统下 Tomcat 的安装与配置，将安装了 Tomcat 的计算机称作一个 Tomcat 服务器。

1.2.1 安装 JDK

安装 Tomcat 之前，首先安装 JDK，这里安装 JDK1.7。假设 JDK 的安装目录是：

D:\jdk1.7

安装 JDK 之后需要进行几个环境变量的设置。对于 Windows 7/Windows XP，用鼠标右键单击“计算机”/“我的电脑”，在弹出的快捷菜单中选择“属性”命令，弹出“系统特性”对话框，再单击该对话框中的“高级系统设置”/“高级选项”，然后单击“环境变量”按钮，分别添加如下的系统环境变量：

变量名：Java_home，变量值：D:\jdk1.7

变量名：Path，变量值：D:\jdk1.7\bin

如果曾经设置过环境变量 Java_home 和 Path，可单击该变量进行编辑操作，将环境变量需要的值加入即可。需要注意的是，在编辑环境变量的值时，新加入的值，如果不是环境变量取值范围中的第一个值或最后一个值，那么新加入的值要和已有的其他值用分号分隔，如果是最后一个值需要和前面的值用分号分隔，如果是第一个值需要和后面的值用分号分隔。如图 1-1 和图 1-2 所示。

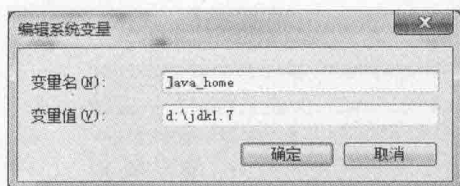


图 1-1 设置 Java_home

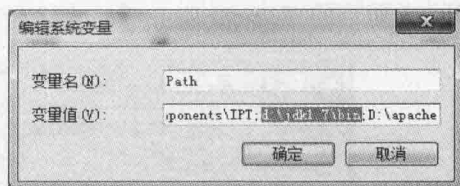


图 1-2 编辑 Path

1.2.2 安装与启动 Tomcat 服务器

1. apache-tomcat-8.0.3.zip 的安装

将下载的 apache-tomcat-8.0.3.zip 解压到磁盘某个分区,比如解压到 D:\,解压缩后将出现如图 1-3 所示的目录结构。

执行 Tomcat 安装根目录中 bin 文件夹中的 startup.bat 启动 Tomcat 服务器。执行 startup.bat 启动 Tomcat 服务器会占用一个 MS-DOS 窗口(如图 1-4 所示的界面),如果关闭当前 MS-DOS 窗口,将关闭 Tomcat 服务器。使用 startup.bat 启动 Tomcat 服务器,Tomcat 服务器将使用 Java_home 环境变量设置的 JDK。

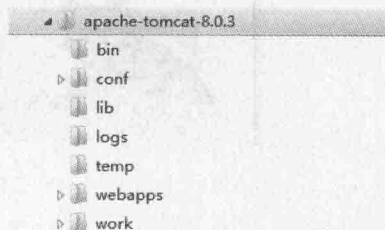


图 1-3 Tomcat 服务器的目录结构

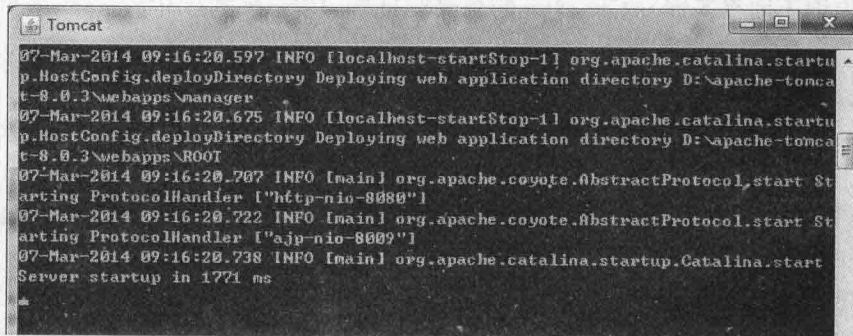


图 1-4 启动 Tomcat 服务器

2. apache-tomcat-8.0.3.exe 的安装

apache-tomcat-8.0.3.exe 文件是针对 Windows 的安装版,安装后形成的目录结构和 apache-tomcat-8.0.3.zip 安装目录相同。

单击下载的 apache-tomcat-8.0.3.exe,将出现“安装向导”界面,单击其中的 Next 按钮,接受授权协议后,将出现选择“安装方式”的界面。在“安装方式”界面中选择 Normal、Minimum、Custom 和 Full 之一,然后按着安装向导的提示进行安装即可,不再赘述。

3. 测试 Tomcat 服务器

在浏览器的地址栏中输入: <http://localhost:8080> 或 <http://127.0.0.1:8080>,会出现

如图 1-5 所示的 Tomcat 服务器的测试页面。

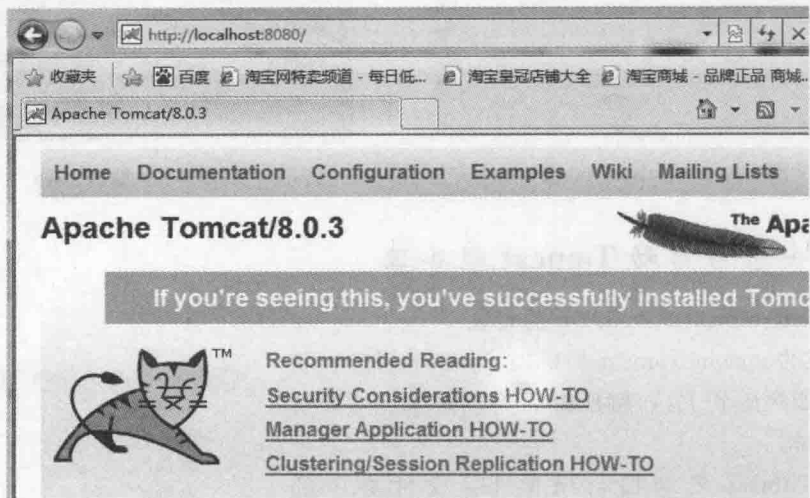


图 1-5 测试 Tomcat 服务器

注意：Tomcat 服务器默认地占用 8080 端口，如果 Tomcat 所使用的端口已经被占用，Tomcat 服务器将无法启动，有关端口号的配置稍后讲解。

4. 配置端口

8080 是 Tomcat 服务器默认占用的端口。可以通过修改 Tomcat 服务器安装目录中 conf 文件下的主配置文件 server.xml 来更改端口号。用记事本打开 server.xml 文件，找到出现

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

的部分，将其中的 port="8080" 更改为新的端口号，并重新启动 Tomcat 服务器即可，例如将 8080 更改为 9090 等。

如果 Tomcat 服务器所在的计算机没有启动占用 80 端口号的其他网络程序，也可以将 Tomcat 服务器的端口号设置为 80，这样用户在访问 Tomcat 服务器时可以省略端口号。例如：

```
http://127.0.0.1/
```

1.3 JSP 页面与 Web 服务目录

1.3.1 JSP 页面

在后面的章节里将详细地学习编写 JSP 页面的语法。简单地说，一个 JSP 页面除了普通的 HTML 标记符外，还可以使用标记符号“<%”，“%>”加入 Java 程序片。一个 JSP 页面按文本文件保存，扩展名是 .jsp。例如，使用文本编辑器“记事本”编辑 JSP 页面，在保存

JSP 页面时,必须将“保存类型”选择为“所有文件”,将“编码”选择为“ANSI”。如果保存 JSP 页面时,计算机系统总是给文件名额外加上“.txt”,那么在保存 JSP 页面时可以将文件名用双引号括起,如图 1-6 所示。

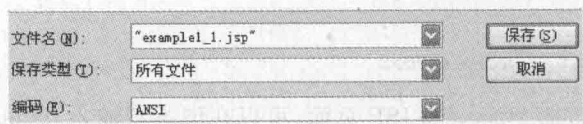


图 1-6 JSP 页面的保存

在保存 JSP 页面时,文件的名字必须符合标识符规定(名字可以由字母、下划线、美元符号和数字组成,并且第一个字符不能是数字)。需要注意的是,JSP 技术基于 Java 语言,名字区分大小写,Boy.jsp 和 boy.jsp 是名字不相同的 JSP 文件。

为了明显地区分普通的 HTML 标记和 Java 程序片段以及 JSP 标签,用大写字母书写普通的 HTML 标记符号(这不是必须的,HTML 标记不区分大小写)。可以用记事本或更好的文本编辑器来编辑 JSP 页面的源文件。

下面的例 1-1 是一个简单的 JSP 页面,页面的运行效果如图 1-7 所示。

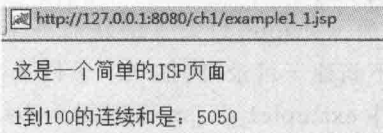


图 1-7 简单的 JSP 页面

例 1-1
example1_1.jsp

```
<% @ page contentType = "text/html;charset = GB2312" %>
<HTML>
<BODY BGCOLOR = yellow>
<FONT Size = 3>
<P>这是一个简单的 JSP 页面
<%      //这是 Java 程序片
    int i,sum = 0;
    for(i=1;i<=100;i++){
        sum = sum + i;
    }
%>
<P>1 到 100 的连续和是: <% = sum %>
</FONT>
</BODY>
</HTML>
```

1.3.2 Web 服务目录

必须将编写好的 JSP 页面文件保存到 Tomcat 服务器的某个 Web 服务目录中,只有这样,远程的用户才可以访问该 Tomcat 服务器上的 JSP 页面。人们常说一个网站,实际上

就是一个 Web 服务目录。

1. 根目录

如果 Tomcat 服务器的安装目录是 D:\apache-tomcat-8.0.3,那么 Tomcat 的 Web 服务目录的根目录是:

```
D:\apache-tomcat-8.0.3\webapps\Root
```

用户如果准备访问根目录中的 JSP 页面,可以在浏览器输入 Tomcat 服务器的 IP 地址(或域名)、端口号和 JSP 页面的名字即可(必须省略 Web 根目录的名字),例如, Tomcat 服务器的 IP 地址是 192.168.1.100,根目录中存放的 JSP 页面的名字是 example1_1.jsp,那么用户在浏览器输入的内容是:

```
http://192.168.1.100:8080/example1_1.jsp
```

也许没有为 Tomcat 服务器所在的机器设置过一个有效的 IP 地址,那么为了调试 JSP 页面,可以打开 Tomcat 服务器所在机器上的浏览器,在浏览器的地址栏中输入:

```
http://127.0.0.1:8080/example1_1.jsp
```

2. webapps 下的 Web 服务目录

Tomcat 服务器安装目录的 webapps 目录下的任何一个子目录都可以作为一个 Web 服务目录。可以在 webapps 下新建子目录,例如 ch1 子目录,那么 ch1 就成为一个 Web 服务目录。如果将 JSP 页面文件 example1_1.jsp 保存到 webapps 下的 Web 服务目录中,那么应当在浏览器的地址栏中输入 Tomcat 服务器的 IP 地址(或域名)、端口号、Web 服务目录和 JSP 页面的名字,比如,example1_1.jsp 保存到 ch1 中,输入的内容为:

```
http://127.0.0.1:8080/ch1/example1_1.jsp
```

将例 1-1 中的 example1_1.jsp 保存到 D:\apache-tomcat-8.0.3\webapps\ch1 中,用浏览器访问 example1_1.jsp 的效果如图 1-7 所示。

注意: 安装 Tomcat 服务器后,webapps 下默认已有: examples、host-manager、manager 子目录;在 webapps 下新建一个 Web 服务目录后不必重新启动 Tomcat 服务器。

3. 新建 Web 服务目录

可以将 Tomcat 服务器所在计算机的某个目录(非 webapps 下的子目录)设置成一个 Web 服务目录,并为该 Web 服务目录指定虚拟目录,即隐藏 Web 服务目录的实际位置,用户只能通过虚拟目录访问 Web 服务目录中的 JSP 页面。

通过修改 Tomcat 服务器安装目录下 conf 文件夹中的 server.xml 文件来设置新的 Web 服务目录。假设要将 D:\MyBook\zhang 以及 C:\wang 作为 Web 服务目录,并让用户分别使用 apple 和 cloud 虚拟目录访问 Web 服务目录 D:\MyBook\zhang 和 C:\wang 下的 JSP 页面。首先用记事本打开 conf 文件夹中的主配置文件: server.xml,找到出现 </Host> 的部分(接近 server.xml 文件尾部处)。然后在 </Host> 的前面加入:

```
<Context path = "/apple" docBase = "D:\MyBook\zhang" debug = "0" reloadable = "true" />
<Context path = "/cloud" docBase = "C:\wang" debug = "0" reloadable = "true" />
```

注意: xml 文件是区分大小写的,不可以将 <Context> 写成 <context>。

主配置文件 server.xml 修改后,必须重新启动 Tomcat 服务器。这样,就可以将 JSP 页面存放到 D:\MyBook\zhang 或 C:\wang 中,用户可以通过虚拟目录 apple 或 cloud 访问 JSP 页面,例如,example1_1.jsp 保存到 D:\MyBook\zhang 或 C:\wang 中,在浏览器地址栏中输入:

```
http://127.0.0.1:8080/apple/example1_1.jsp
```

或

```
http://127.0.0.1:8080/cloud/example1_1.jsp
```

4. 相对目录

Web 服务目录下的目录称为该 Web 服务目录下的相对 Web 服务目录。例如,可以在 Web 服务目录 D:\MyBook\zhang(虚拟目录是 apple)下再建立一个子目录 image,将 example1_1.jsp 文件保存到 image 中。那么可以在浏览器的地址栏中输入:

```
http://127.0.0.1:8080/apple/image/example1_1.jsp
```

来访问 example1_1.jsp。

1.4 JSP 运行原理

当服务器上的一个 JSP 页面被第一次请求执行时,服务器上的 JSP 引擎首先将 JSP 页面文件转译成一个 java 文件,并编译这个 java 文件生成字节码文件,然后执行字节码文件响应用户的请求。而当这个 JSP 页面再次被请求执行时,JSP 引擎将直接执行字节码文件来响应用户。根据 Tomcat 执行 JSP 页面的这一特点,Web 程序设计完毕后,可以由管理者首次访问 JSP 页面,这样一来,后续用户再访问该 JSP 页面时,JSP 引擎将直接执行已经驻留在内存中的字节码文件来响应用户,提高了 JSP 页面的访问速度,这也正是 JSP 技术比 ASP 技术速度快的一个主要原因。

JSP 引擎对每个 JSP 页面都对应地产生一个字节码文件,以下阐述 Tomcat 服务器执行 JSP 页面所对应的字节码文件的原理,即给出 JSP 页面的运行原理:

(1) 把 JSP 页面中的 HTML 标记(页面的静态部分)发送给用户的浏览器,由浏览器中的 HTML 解释器负责解释执行 HTML 标记。

(2) 负责处理 JSP 标记,并将有关的处理结果发送到用户的浏览器。

(3) 执行“<%”和“%>”之间的 Java 程序片(JSP 页面中的动态部分),并把执行结果交给用户的浏览器显示。

(4) 当多个用户请求一个 JSP 页面时,Tomcat 服务器为每个用户启动一个线程,该线程负责执行常驻内存的字节码文件来响应相应用户的请求。这些线程由 Tomcat 服务器来管理,将 CPU 的使用权在各个线程之间快速切换,以保证每个线程都有机会执行字节码文件,这与传统的 CGI 为每个用户启动一个进程相比较,效率要高得多。

注意: 如果对 JSP 页面进行了修改、保存,那么 Tomcat 服务器会生成新的字节码。

下面是 JSP 引擎生成的 example1_1.jsp 的 java 文件: example1_005f1_jsp.java,文件把 JSP 引擎交给用户端负责显示的内容做了注释(见/***/)。如果 JSP 页面保存在 Root

目录中,可以在 Tomcat 服务器下的: work\Catalina\localhost_org\apache\jsp 目录中找到 Tomcat 服务器生成的 JSP 页面对应的 Java 文件以及编译 Java 文件得到的字节码文件。

example1_005f1_jsp.java

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class example1_005f1_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
    private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();
    private static java.util.List _jspx_dependants;
    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.AnnotationProcessor _jsp_annotationprocessor;
    public Object getDependants() {
        return _jspx_dependants;
    }
    public void _jspInit() {
        _el_expressionfactory = _jspxFactory.getJspApplicationContext(
            getServletConfig().getServletContext()).getExpressionFactory();
    }
    public void _jspDestroy() {
    }
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;
        try {
            response.setContentType("text/html;charset = GB2312");
            pageContext =
                _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;
            /**/out.write("\r\n");
            /**/out.write("<HTML>< BODY BGCOLOR = yellow>< FONT Size = 3>\r\n");
            /**/out.write("<P>这是一个简单的 JSP 页面\r\n");
            /**/out.write(" ");
            int i, sum = 0;
            for(i = 1; i <= 100; i++){
                if(i % 2 == 0)
                    sum = sum + i;
            }
        }
    }
}
```

```

    }
    /**/out.write("\r\n");
    /**/out.write("<P>1 到 100 的偶数之和是: ");
    /**/out.print(sum);
    /**/out.write("\r\n");
    /**/out.write("</FONT></BODY></HTML>\r\n");
    } catch (Throwable t) {
        if (! (t instanceof SkipPageException)){
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                try { out.clearBuffer(); } catch (java.io.IOException e) {}
            if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        }
    } finally {
        _jspxFactory.releasePageContext(_jspx_page_context);
    }
}
}
}

```

1.5 实验：编写、保存、运行 JSP 页面

1. 实验目的

本实验的目的是让学生掌握怎样设置 Web 服务目录,怎样修改 Tomcat 服务器的端口号,怎样访问 Web 服务目录下的 JSP 页面。

2. 实验步骤

(1) 安装 Tomcat 服务器。

将下载的 apache-tomcat-8.0.3.zip 解压到硬盘某个分区,比如 C。

(2) 设置 Web 服务目录。

在硬盘分区 C 下新建一个 Web 服务目录,名字为 student。

打开 Tomcat 安装目录中 conf 文件夹里的 server.xml 文件,找到出现</Host>的部分(server.xml 文件尾部)。然后在</Host>的前面加入:

```
<Context path = "/friend" docBase = "c:/student" debug = "0" reloadable = "true" />
```

将 student 设置为 Web 服务目录,并为该 Web 服务目录指定名字为 friend 的虚拟目录。

(3) 修改端口号。

在 server.xml 文件中找到修改端口号部分,将端口号修改为 9999。

(4) 启动 Tomcat 服务器。

如果已经启动,必须关闭 Tomcat 服务器,并重新启动。

(5) 编写 JSP 页面。

用文本编辑器编写一个简单的 JSP 页面 number.jsp(见后面的参考代码),该 JSP 页面可以显示 1 至 100 之间的完数。将 JSP 页面保存到 Web 服务目录 student 中。

(6) 访问 JSP 页面。

用浏览器访问 Web 服务目录 student 中的 JSP 页面 number.jsp,在浏览器地址栏

输入:

`http://127.0.0.1:9999/friend/number.jsp`

3. 参考代码

number.jsp

```
<% @ page contentType = "text/html;charset = GB2312" %>
```

```
<HTML>
```

```
<BODY BGCOLOR = yellow>
```

```
<FONT Size = 3>
```

如果一个正整数刚好等于它的真因子之和,这样的正整数为完数,

例如, $6 = 1 + 2 + 3$, 因此 6 就是一个完数。

1 到 1000 内的完数有:

```
<% int i,j,sum;
    for(i=1,sum=0;i<=1000;i++){
        for(j=1;j<i;j++){
            if(i%j==0)
                sum=sum+j;
        }
        if(sum==i)
            out.print(" "+i);
    }
    %>
```

```
</FONT></BODY></HTML>
```

习 题 1

1. 怎样启动和关闭 Tomcat 服务器?
2. 请在 C:\ 下建立一个名字为 book 的目录,并将该目录设置成一个 Web 服务目录,然后编写一个简单的 JSP 页面,保存到该目录中,让用户使用虚拟目录 red 访问该 JSP 页面。
3. 怎样访问 Web 服务目录子目录中的 JSP 页面?
4. 如果想修改 Tomcat 服务器的端口号,应当修改哪个文件? 能否将端口号修改为 80。

第 2 章

JSP 页面与 JSP 标记

本章导读

主要内容

- JSP 页面的基本结构
- 变量和方法的声明
- Java 程序片
- 表达式
- JSP 中的注释
- JSP 指令标记
- JSP 动作标记

难点

- Java 程序片的运行原理
- include 指令标记与 include 动作标记

关键实践

- 编写一个 JSP 页面, 让该 JSP 页面包含 5 种基本的元素
- 编写含有 JSP 指令标记的 JSP 页面
- 编写含有 JSP 动作标记的 JSP 页面

2.1 JSP 页面的基本结构

在传统的 HTML 页面文件中加入 Java 程序片和 JSP 标记就构成了一个 JSP 页面文件。一个 JSP 页面可由 5 种元素组合而成:

- 普通的 HTML 标记。
- JSP 标记, 如指令标记、动作标记。
- 变量和方法的声明。
- Java 程序片。
- Java 表达式。

当服务器上的一个 JSP 页面被第一次请求执行时, 服务器上的 JSP 引擎首先将 JSP 页面文件转译成一个 Java 文件, 再将这个 Java 文件编译生成字节码文件, 然后通过执行字节码文件响应用户的请求, 执行原理是:

(1) 把 JSP 页面中普通的 HTML 标记交给用户的浏览器执行显示。

(2) JSP 标记、数据和方法声明、Java 程序片由服务器负责执行,将需要显示的结果发送给用户的浏览器。

(3) Java 表达式由服务器负责计算,并将结果转化为字符串,然后交给用户的浏览器负责显示。


Tomcat 服务器的 webapps 目录的子目录都可以作为一个 Web 服务目录,在 webapps 目录下新建一个 Web 服务目录: ch2,除非特别约定,本章例子中的 JSP 页面均保存在 ch2 中。

例 2-1 中, example2_1.jsp 页面包含了 5 种元素,页面效果如图 2-1 所示。

例 2-1

example2_1.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %> <!-- jsp 指令标记 -->
<% @ page import = "java.util.Date" %> <!-- jsp 指令标记 -->
<% !
    Date date; // 数据声明
    int sum;
    public int getFactorSum(int n) { // 方法声明
        for(int i = 1; i < n; i++) {
            if(n % i == 0)
                sum = sum + i;
        }
        return sum;
    }
%>
<HTML><BODY bgcolor = cyan> <!-- html 标记 -->
<FONT Size = 4><P>程序片创建 Date 对象:
    <% date = new Date(); //Java 程序片
        out.println("<br>" + date + "<br>");
        int m = 100;
    %>
    <% = m %> <!-- Java 表达式 -->
        的因子之和是(不包括<% = m %>):
    <% = getFactorSum(m) %> <!-- Java 表达式 -->
</FONT></BODY></HTML>
```

地址  http://127.0.0.1:8080/ch2/example2_1.jsp

程序片创建Date对象:
Mon Jan 19 18:20:08 CST 2009
100 的因子之和是(不包括100): 117

图 2-1 包含了 5 种元素的 JSP 页面

2.2 变量和方法的声明

在“<%!”和“%>”标记符号之间声明变量和方法。

2.2.1 声明变量

在“<%!”和“%>”标记符之间声明变量,即在“<%!”和“%>”之间放置 Java 的变量声明语句,变量的类型可以是 Java 语言允许的任何数据类型。将“<%!”和“%>”之间声

明的变量称为 JSP 页面的成员变量。例如：

```
<%!  
    int x,y=100,z;  
    String tom=null,jerry="love JSP";  
    Date date;  
%>
```

“<%!”和“%>”之间声明的变量在整个 JSP 页面内都有效，与标记符号<%!、%>所在的位置无关，但习惯将标记符号<%!、%>写在 Java 程序片的前面。JSP 引擎将 JSP 页面转译成 Java 文件时，将“<%!”和“%>”之间声明的变量作为类的成员变量，这些变量的内存空间直到服务器关闭才被释放。当多个用户请求一个 JSP 页面时，JSP 引擎为每个用户启动一个线程，这些线程由 JSP 引擎服务器来管理。这些线程共享 JSP 页面的成员变量，因此任何一个用户对 JSP 页面成员变量操作的结果，都会影响到其他用户。

例 2-2 利用成员变量被所有用户共享这一性质，实现了一个简单的计数器，页面效果如图 2-2 所示。

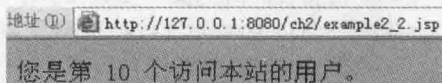


图 2-2 JSP 页面的成员变量

例 2-2

example2_2.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>  
<HTML>< BODY BGCOLOR = cyan>  
< FONT Size = 4>  
    <%!    int i = 0;  
    %>  
    <%    i++;  
    %>  
<P>你是第 <% = i> 个访问本站的用户。  
</FONT></BODY></HTML>
```

由于成员变量的有效范围与标记符号<%!、%>所在的位置无关，因此例 2-2 中的 example2_2.jsp 等价于下面的 JSP 页面(将<%!、%>写在 JSP 页面的后面)：

```
<% @ page contentType = "text/html; charset = GB2312" %>  
<HTML>< BODY BGCOLOR = cyan>< FONT Size = 4>  
    <% i++;  
    %>  
<P>你是第 <% = i> 个访问本站的用户。  
    <%! int i = 0;  
    %>  
</FONT></BODY></HTML>
```

2.2.2 声明方法

在“<%!”和“%>”之间声明方法，该方法在整个 JSP 页面有效(与标记符号<%!、%>所在的位置无关)，但是该方法内定义的变量只在该方法内有效。这些方法将在 Java 程序片中被调用。当方法被调用时，方法内定义的变量被分配内存，调用完毕即可释放所占的内

存。在例 2-3 中, example2_3.jsp 在“<%!”和“%>”之间声明了两个方法: getArea(double a)和 getLength(double a),在程序片中调用这两个方法,分别计算圆的面积和周长。example2_3.jsp 页面效果如图 2-3 所示。

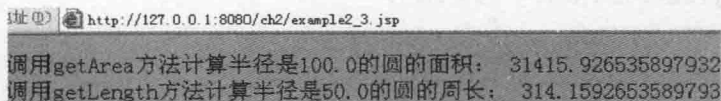


图 2-3 JSP 页面中方法的声明与调用

例 2-3

example2_3.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>< BODY bgcolor = cyan >< FONT size = 4 >
    <% !    final double PI = Math.PI;
            double r;
            double getArea(double a){
                return PI * a * a;
            }
            double getLength(double a) {
                return 2 * PI * a;
            }

    %>
    <%    r = 100;
            out.println("调用 getArea 方法计算半径是" + r + "的圆的面积: ");
            double area = getArea(r);
            out.println(area);
            r = 50;
            out.println("<BR>调用 getLength 方法计算半径是" + r + "的圆的周长: ");
            double length = getLength(r);
            out.println(length);

    %>
</FONT></BODY></HTML>
```

2.3 Java 程序片

可以在“<%”和“%>”之间插入 Java 程序片。一个 JSP 页面可以有許多 Java 程序片,这些 Java 程序片将被 JSP 引擎按顺序执行。在 Java 程序片中声明的变量称作 JSP 页面的局部变量。局部变量的有效范围与其声明的位置有关,即局部变量在 JSP 页面后继的所有 Java 程序片以及 Java 表达式部分内都有效。JSP 引擎将 JSP 页面转译成 Java 文件时,将各个 Java 程序片的这些变量作为类中某个方法的变量,即局部变量。

现在已经知道,当多个用户请求一个 JSP 页面时,JSP 引擎为每个用户启动一个线程,该线程负责执行字节码文件响应用户的请求。因此当多个用户访问一个 JSP 页面时,该 JSP 页面中的 Java 程序片将被运行多次,分别运行在不同的线程中。

Java 程序片的执行有如下特点(如图 2-4 所示):

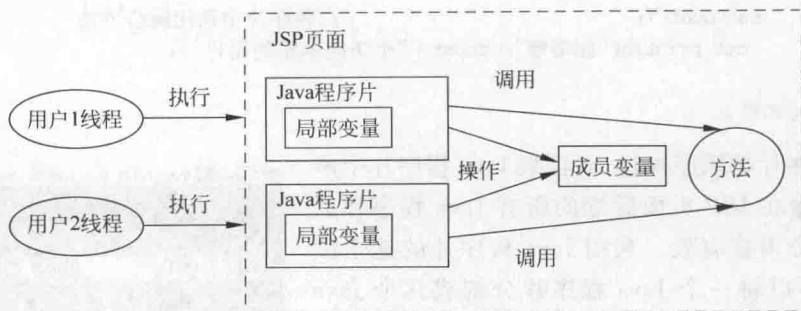


图 2-4 程序片的执行

1. 操作 JSP 页面的成员变量

Java 程序片中操作的成员变量是各个线程共享的变量,任何一个线程对 JSP 页面成员变量操作的结果,都会影响到其他线程。

2. 调用 JSP 页面的方法

Java 程序片中可以出现方法调用语句,该方法必须是 JSP 页面的方法(在“<%!”和“%>”之间声明方法)。

3. 声明操作局部变量

当一个线程享用 CPU 资源时,JSP 引擎让该线程执行 Java 程序片,这时,Java 程序片中的局部变量被分配内存空间,当轮到另一个线程享用 CPU 资源时,JSP 引擎让这个线程执行 Java 程序片,那么,Java 程序片中的局部变量会再次被分配内存空间。也就是说,Java 程序片已经被执行了两次,分别运行在不同的线程中,即运行在不同的时间片内。运行在不同线程中的 Java 程序片的局部变量互不干扰,即一个用户改变 Java 程序片中的局部变量的值不会影响其他用户的 Java 程序片中的局部变量。当一个线程将 Java 程序片执行完毕,运行在该线程中的 Java 程序片的局部变量释放所占的内存。

根据 Java 程序片的上述特点,对于某些特殊情形必须给予特别注意。例如,如果一个用户在执行 Java 程序片时调用 JSP 页面的方法操作成员变量时,可能不希望其他用户也调用该方法操作成员变量,以免对其产生不利的影响(成员变量被所有的用户共享),那么就应该将操作成员变量的方法用 synchronized 关键字修饰。当一个线程在执行 Java 程序片期间调用 synchronized 方法时,其他线程想在 Java 程序片中调用这个 synchronized 方法时必须等待,直到调用 synchronized 方法的线程将该方法执行完毕。

在例 2-4 中,通过 synchronized 方法操作一个成员变量来实现一个简单的计数器。

例 2-4

example2_4.jsp

```
<% @ page contentType = "text/html; Charset = GB2312" %>
<HTML>< BODY>
    <%!    int count = 0;                //被用户共享的 count
           synchronized void setCount() { //synchronized 修饰的方法
               count++;
           }
    %>
```

```

<%      setCount();                      //程序片中调用同步方法
        out.println("你是第" + count + "个访问本站的用户");
    %>
</BODY></HTML>

```

Java 程序片按顺序执行,而且某 Java 程序片中声明的局部变量在 JSP 页面后继的所有 Java 程序片以及表达式部分内都有效。利用 Java 程序片的这个性质,有时候可以将一个 Java 程序片分割成几个 Java 程序片,然后在这些 Java 程序片之间再插入其他标记元素。例 2-5 通过将程序片分割成几部分,来验证用户输入的 E-mail 地址中是否含有非法的字符,页面效果如图 2-5 所示。

例 2-5

example2_5.jsp

```

<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY bgcolor = cyan><FONT Size = 3>
<P> 请输入 E-mail: <BR>
    <FORM action = "" method = get name = form>
        <INPUT type = "text" name = "client" value = "请输入 E-mail">
        <INPUT TYPE = "submit" value = "送出" name = submit>
    </FORM>
<% String str = request.getParameter("client");
    if(str!=null){
        int index = str.indexOf("@");
        if(index == -1){
            %>      <BR>你的 E-mail 地址中没有@。
        <%      }
        else{
            int space = str.indexOf(" ");
            if(space!= -1){
                %>      <BR>你的 E-mail 地址含有非法的空格。
            <%      }
            else{
                int start = str.indexOf("@");
                int end = str.lastIndexOf("@");
                if(start!=end){
                    %>      <BR>你的 E-mail 地址有两个以上的符号: @。
                <%      }
                else{
                    out.print("<BR>" + str);
                    %>      <BR>你的 E-mail 地址书写正确。
                <%      }
            }
        }
    }
    %>
</FONT></BODY></HTML>

```

地址 @ http://127.0.0.1:8080/ch2/example2_5.jsp

请输入E-mail:

请输入E-mail

送出

sahngzhai@yahoo.com
您的E-mail地址书写正确。

图 2-5 分割程序片

2.4 表 达 式

可以在“<%=”和“%>”之间插入一个表达式,例如: <%= x+y %>,不可以在“<%=”和“%>”之间插入语句。例如: <%= x=100; %>是错误的。需要特别注意的是,“<%=”是一个完整的符号,“<%”和“=”之间不要有空格。表达式的值由服务器负责计算,并将计算结果以字符串形式发送至用户端显示。如果表达式无法求值, Tomcat 引擎将给出编译错误。例 2-6 计算了表达式的值,页面效果如图 2-6 所示。

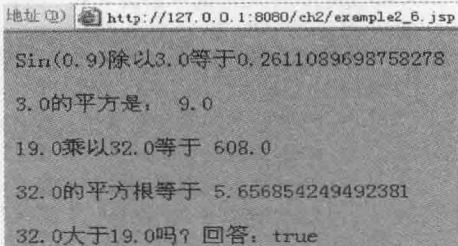


图 2-6 计算表达式的值

例 2-6

example2_6.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY bgcolor = cyan><FONT Size = 3>
<% double x = 0.9,y = 3;
%>
<P> Sin(<% = x %>)除以<% = y %>等于<% = Math.sin(x)/y %>
  <p><% = y %>的平方是: <% = Math.pow(y,2) %>
<%   x = 19;
      y = 32;
%>
<P><% = x %>乘以<% = y %>等于 <% = x * y %>
<P><% = y %>的平方根等于 <% = Math.sqrt(y) %>
<P><% = y %>大于<% = x %>吗? 回答: <% = y > x %>
</FONT></BODY></HTML>
```

2.5 JSP 中的注释

注释可以增强 JSP 文件的可读性,并易于 JSP 文件的维护。JSP 中的注释可分为两种。

1. HTML 注释

在标记符号“<!--”和“-->”之间加入注释内容。例如:

<!-- 下面是一个提交密码的表单 -->

JSP 引擎把 HTML 注释交给用户,用户通过浏览器查看 JSP 的源文件时,能够看到 HTML 注释。

2. JSP 注释

在标记符号“<!--”和“--%>”之间加入注释内容。例如:

<%-- 利用 for 循环输出用户的全部信息 --%>

JSP 引擎忽略 JSP 注释,即在编译 JSP 页面时忽略 JSP 注释。

例 2-7 中的 JSP 页面使用了 HTML 注释和 JSP 注释。

例 2-7

example2_7.jsp

```

<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY>
  <P> 请输入三角形的三条边 a,b,c 的长度:
  <!-- 以下是 HTML 表单,向服务器发送三角形的三条边的长度 -->
  <!-- 要特别注意 action="" 的引号中不要有空格 -->
  <FORM action="" method = post name = form>
    <P>请输入三角形边 a 的长度: <INPUT type = "text" name = "a">
    <P>请输入三角形边 b 的长度: <INPUT type = "text" name = "b">
    <P>请输入三角形边 c 的长度: <INPUT type = "text" name = "c">
    <INPUT TYPE = "submit" value = "送出" name = submit>
  </FORM>
  <% -- 获取用户提交的数据 -- %>
  <%   String string_a = request.getParameter("a"),
        string_b = request.getParameter("b"),
        string_c = request.getParameter("c");
        double a = 0,b = 0,c = 0;

    %>
    <% -- 判断字符串是否是空对象,如果是空对象就初始化 -- %>
    <% if(string_a == null){
        string_a = "0";
        string_b = "0";
        string_c = "0";
    }

    %>
    <% -- 求出边长,并计算面积 -- %>
    <% try{ a = Double.valueOf(string_a).doubleValue();
        b = Double.valueOf(string_b).doubleValue();
        c = Double.valueOf(string_c).doubleValue();
        if(a + b > c && a + c > b && b + c > a){
            double p = (a + b + c)/2.0;
            double mianji = Math.sqrt(p * (p - a) * (p - b) * (p - c));
            out.print("<BR>" + "三角形面积: " + mianji);
        }
        else
            out.print("<BR>" + "你输入的三边不能构成一个三角形");
        }
        catch(NumberFormatException e){
            out.print("<BR>" + "请输入数字字符");
        }
    %>
  </BODY></HTML>

```

2.6 JSP 指令标记

2.6.1 page 指令

page 指令用来定义整个 JSP 页面的一些属性和这些属性的值。page 指令标记可以指

定如下属性的值：

contentType、import、language、session、buffer、autoFlush、isThreadSafe、pageEncoding

例如，可以用 page 指令指定 JSP 页面的 contentType 属性的值是 text/html；charset = GB2312，这样，JSP 页面就可以显示标准汉语。例如：

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

page 指令对整个页面有效，与其书写的位置无关，但习惯把 page 指令写在 JSP 页面的最前面。page 指令的格式：

```
<% @ page 属性 1 = "属性 1 的值" 属性 2 = "属性 2 的值" ..... %>
```

属性值需用单引号或双引号括起来，如果为一个属性指定几个值的话，这些值用逗号分隔。例如：

```
<% @ page import = "java.util. * ", "java.io. * " %>
```

可以用一个 page 指令指定多个属性的值，例如：

```
<% @ page 属性 1 = "属性 1 的值" 属性 2 = "属性 2 的值" ..... %>
```

也可以使用多个 page 指令分别为每个属性指定值。例如：

```
<% @ page 属性 1 = "属性 1 的值" %>
```

```
<% @ page 属性 2 = "属性 2 的值" %>
```

⋮

```
<% @ page 属性 n = "属性 n 的值" %>
```

当为 import 属性指定多个属性值时，JSP 引擎把 JSP 页面转译成的 Java 文件中会有如下 import 语句：

```
import java.util. * ;
```

```
import java.io. * ;
```

```
import java.sql. * ;
```

在一个 JSP 页面中，也可以使用多个 page 指令来指定属性及其值。需要注意的是：可以使用多个 page 指令指定 import 属性几个值，但其他属性只能使用 page 指令指定一个值。例如：

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

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

```
<% @ page import = "java.util. * ", "java.awt. * " %>
```

不允许两次使用 page 指令给 contentType 属性指定不同的属性值，下列用法是错误的：

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

```
<% @ page contentType = "application/msword" %>
```

以下将分别讲述 page 指令可以指定的属性及其作用。

1. language 属性

定义 JSP 页面使用的脚本语言，该属性的值目前只能取 java。

为 language 属性指定值的格式：

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

language 属性的默认值是 java,即如果在 JSP 页面中没有使用 page 指令指定该属性的值的话,那么 JSP 页面默认有如下 page 指令:

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

2. import 属性

该属性的作用是为 JSP 页面引入 Java 核心包中的类,这样就可以在 JSP 页面的程序片部分、变量及方法声明部分、表达式部分使用核心包中的类。可以为 import 属性指定多个值,该属性的值可以是 Java 某核心包中的所有类或一个具体的类。例如:

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

JSP 页面默认 import 属性已经有如下的值:

```
" java.lang. * ","javax.servlet. * ","javax.servlet.jsp. * ","javax.servlet.http. * "
```

3. contentType 属性

contentType 属性值确定 JSP 页面响应的 MIME(Multipurpose Internet Mail Extention)类型和 JSP 页面字符的编码。属性值的一般形式是 MIME 类型; charset=编码。例如:

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

如果不使用 page 指令为 contentType 指定一个值,那么 contentType 属性的默认值是 text/html; charset=ISO-8859-1。

当用户请求一个 JSP 页面时, Tomcat 服务器负责解释执行 JSP 页面,并将某些信息发送到用户的浏览器,以使用户浏览这些信息。Tomcat 服务器同时负责通知用户的浏览器使用怎样的方式来处理所接收到的信息,这就要求 JSP 页面必须设置响应的 MIME 类型和 JSP 页面字符的编码。例如,如果希望用户的浏览器启用 HTML 解析器来解析执行所接收到的信息(即所谓的网页形式),就可以如下设置 contentType 属性的值:

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

如果希望用户的浏览器启用本地的 MS-Word 应用程序来解析执行收到的信息,就可以如下设置 contentType 属性的值:

```
<% @ page contentType = "application/msword" %>
```

JSP 页面使用 page 指令只能为 contentType 指定一个值,不允许两次使用 page 指令给 contentType 属性指定不同的属性值,下列用法是错误的:

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

```
<% @ page contentType = "application/msword" %>
```

可以使用 page 指令为 contentType 属性指定的值有: text/html、text/plain、image/gif、image/x-bitmap、image/jpeg、image/pjpeg、application/x-shockwave-flash、application/vnd.ms-powerpoint、application/vnd.ms-excel、application/msword 等。

如果用户的浏览器不支持某种 MIME 类型,那么用户的浏览器就无法用相应的手段处理所接收到的信息。例如,使用 page 指令设置 contentType 属性的值是 application/msword,

如果用户浏览器所驻留的计算机没有安装 MS-Word 应用程序,那么浏览器就无法处理所接收到的信息。

例 2-8 中有两个 JSP 页面,其中的 first.jsp 页面使用 page 指令设置 contentType 属性的值是 text/html; charset = GB2312,当用户请求 first.jsp 页面时,用户的浏览器启用 HTML 解析器来解析执行收到的信息;second.jsp 页面使用 page 指令设置 contentType 属性的值是 application/msword,当用户请求 second.jsp 页面时,用户的浏览器将启动本地的 MS-Word 应用程序来解析执行收到的信息,页面效果如图 2-7 所示。

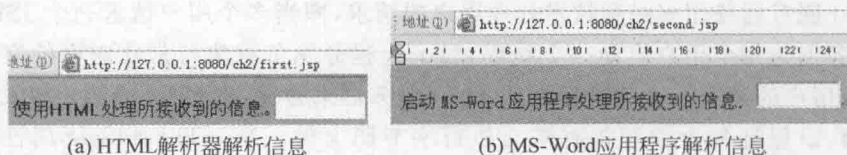


图 2-7 两个页面的效果图

例 2-8

first.jsp

```
<% @ page contentType = "text/html; Charset = GB2312" %>
<HTML>< BODY BGCOLOR = cyan>
< FONT Size = 3>
< P>使用 HTML 处理所接收到的信息。
    < input type = text size = 10>
</FONT></BODY></HTML>
```

second.jsp

```
<% @ page contentType = "application/msword" %>
<HTML>< BODY BGCOLOR = cyan>
< FONT Size = 8>
< P>启动 MS-Word 应用程序处理所接收到的信息。
    < input type = text size = 10>
</FONT></BODY></HTML>
```

4. session 属性

用于设置是否需要使用内置的 session 对象。

session 属性的属性值可以是 true 或 false,默认的属性值是 true。

5. buffer 属性

内置输出流对象 out 负责将服务器的某些信息或运行结果发送到用户端显示,buffer 属性用来指定 out 设置的缓冲区的大小或不使用缓冲区。buffer 属性可取值 none 来设置 out 不使用缓冲区。buffer 属性的默认值是 8KB。例如:

```
<% @ page buffer = "24KB" %>
```

6. autoFlush 属性

指定 out 的缓冲区被填满时,缓冲区是否自动刷新。autoFlush 可以取值 true 或 false。autoFlush 属性的默认值是 true。当 autoFlush 属性取值 false 时,如果 out 的缓冲区被填

满,就会出现缓存溢出异常。当 buffer 的值是 none 时,autoFlush 的属性值就不能设置成 false。

7. isThreadSafe 属性

isThreadSafe 的属性值取 true 或 false,用来设置 JSP 页面是否可多线程访问。当 isThreadSafe 属性值设置为 true 时,JSP 页面能同时响应多个用户的请求;当 isThreadSafe 属性值设置成 false 时,JSP 页面同一时刻只能处理响应一个用户的请求,其他用户需排队等待。isThreadSafe 属性的默认值是 true。

Tomcat 服务器使用多线程技术处理用户的请求,即当多个用户请求一个 JSP 页面时, Tomcat 服务器为每个用户启动一个线程,每个线程分别负责执行常驻内存的字节码文件来响应相应用户的请求。这些线程由 Tomcat 服务器来管理,将 CPU 的使用权在各个线程间快速切换,以保证每个线程都有机会执行字节码文件。当 isThreadSafe 属性值为 true 时,CPU 的使用权在各个线程间快速切换,也就是说,即使一个用户的线程没有执行完毕,CPU 的使用权也可能要切换给其他的线程,如此轮流,直到各个线程执行完毕;当 JSP 使用 page 指令将 isThreadSafe 属性值设置成 false 时,该 JSP 页面同一时刻只能处理响应一个用户的请求,其他用户需排队等待,也就是说,CPU 要保证一个线程将 JSP 页面执行完毕才会把 CPU 使用权切换给其他线程。

下列 JSP 页面 computer.jsp 将 isThreadSafe 属性的值设置为 false。

computer.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<% @ page isThreadSafe = "false" %>
<HTML><BODY>
  <%!    int i = 1;           //被所有用户共享
  %>
  <%
    for(int k = 1; k <= 100; k++){
      out.println(i);
      i++;
    }
  %>
</BODY></HTML>
```

首先注意到,上述 computer.jsp 页面中的成员变量 *i* 被所有用户共享。假设有两个用户访问上述 computer.jsp 页面,那么当第一个用户访问该页面时, Tomcat 引擎要保证该用户的线程执行完该页面,才会让第 2 个用户线程执行该 JSP 页面。因此,第一个用户看到的页面运行效果是输出 1 至 100 的整数,第 2 个用户看到的页面运行效果是输出 101 至 200 的整数。但是,如果 computer.jsp 将 isThreadSafe 属性的值设置为 true, Tomcat 引擎就会将 CPU 的使用权在各个线程间快速切换,这样一来,就可出现第 1 个用户的线程在没有执行完 JSP 页面中的程序片中的循环时,例如,当循环到第 50 次时, Tomcat 引擎就可能将 CPU 的使用权切换给第 2 个用户的线程,那么第 2 个用户看到的第一个整数将是 51。

8. info 属性

info 属性的属性值是一个字符串,其目的是为 JSP 页面准备一个常用且可能需要经常修改的字符串。例如:

```
<% @ page info = "we are students" %>
```

可以在 JSP 页面中使用方法：

```
getServletInfo();
```

获取 info 属性的属性值。

注意：当 JSP 页面被转译成 Java 文件时，转译成的类是 Servlet 的一个子类，所以在 JSP 页面中可以使用 Servlet 类的方法：getServletInfo()。

2.6.2 include 指令标记

如果需要在 JSP 页面的某处整体插入一个文件，就可以考虑使用这个指令标记。该指令标记语法如下：

```
<% @ include file = "文件的 URL" %>
```

该指令标记的作用是在 JSP 页面出现该指令的位置处，静态插入一个文件。被该指令插入的文件必须是可访问和可使用的，如果该文件和当前 JSP 页面在同一 Web 服务目录中，那么“文件的 URL”就是文件的名字；如果该文件在 JSP 页面所在的 Web 服务目录的一个子目录中，例如 fileDir 子目录中，那么“文件的 URL”就是“fileDir/文件的名字”。

静态插入就是当前 JSP 页面和插入的文件合并成一个新的 JSP 页面，然后 JSP 引擎再将这个新的 JSP 页面转译成 Java 文件。因此，一个 JSP 页面使用该指令插入文件后，必须保证新合并成的 JSP 页面符合 JSP 语法规则，即能够成为一个 JSP 页面文件。例如，JSP 页面 A.jsp 已经使用 page 指令为 contentType 属性设置了值：

```
<% @ page contentType = "text/html; charset = GB2312" %>
```

如果 A.jsp 使用 include 指令标记插入一个 JSP 页面：B.jsp，而 B.jsp 页面使用 page 指令为 contentType 属性设置的值是：

```
<% @ page contentType = "application/msword" %>
```

那么，合并后的 JSP 页面就两次使用 page 指令为 contentType 属性设置了不同的属性值，导致出现语法错误，因为 JSP 页面中的 page 指令只能为 contentType 指定一个值。

Tomcat 5.0 版本以后的服务器每次都要检查 include 指令标记插入的文件是否被修改过，因此，JSP 页面成功静态插入一个文件后，如果对插入的文件进行了修改，那么 Tomcat 服务器会重新编译 JSP 页面，即将当前的 JSP 页面和修改后的文件合并成一个 JSP 页面，然后 Tomcat 服务器再将这个新的 JSP 页面转译成 Java 类文件。

使用 include 指令可以实现代码的复用。例如，每个 JSP 页面上都可能需要一个导航条，以使用户在各个 JSP 页面之间方便地切换，那么每个 JSP 页面都可以使用 include 指令在页面的适当位置整体插入一个相同的文件。

需要特别注意的是，在 Tomcat 4 版本中，被插入的文件不允许使用 page 指令指定 contentType 属性的值，但是，在 Tomcat 5 版本中，被插入的文件允许使用 page 指令指定 contentType 属性的值，但指定的值必须和插入该文件的 JSP 页面中的 page 指令指定的

contentType 属性的值相同。

例 2-9 中的 example2_9.jsp 页面使用 include 指令标记静态插入一个文本文件 Hello.txt, 该文本文件的内容如下:

```
<% @ page contentType = "text/html; charset = GB2312" %>
很高兴认识你们!
nice to meet you.
```

Hello.txt 文件和当前 example2_9.jsp 页面在同一 Web 服务目录中, 页面效果如图 2-8 所示。

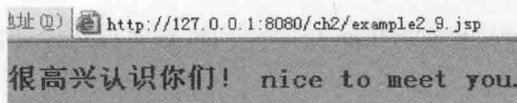


图 2-8 使用 include 指令标记(a)

例 2-9

example2_9.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY bgcolor = cyan>
<H3><% @ include file = "Hello.txt" %>
</H3>
</BODY></HTML>
```

上述 example2_9.jsp 等价于下面的 JSP 文件: example2_9_1.jsp。

example2_9_1.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<html><BODY>
<H3> 很高兴认识你们!
nice to meet you.
</H3>
</BODY></HTML>
```

例 2-10 中的 JSP 页面 example2_10.jsp 使用 include 指令标记静态插入一个 JSP 文件 computer.jsp。computer.jsp 和 example2_10.jsp 均保存在 Web 服务目录 ch2 中, example2_10.jsp 页面的效果如图 2-9 所示。

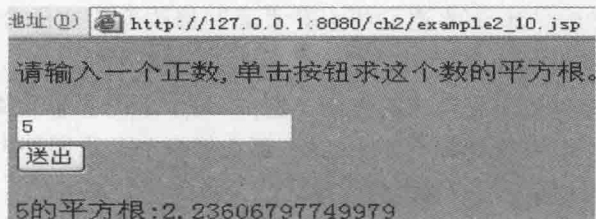


图 2-9 使用 include 指令标记(b)

例 2-10

example2_10.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>< BODY Bgcolor = cyan>< FONT size = 3>
<P>请输入一个正数,单击按钮求这个数的平方根。
<% @ include file = "computer.jsp" %>
</FONT></BODY></HTML>
```

computer.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<FORM action = "" method = post name = form>
< INPUT type = "text" name = "ok">
< BR>< INPUT TYPE = "submit" value = "送出" name = submit>
</FORM>
<% String a = request.getParameter("ok");
    if(a == null){
        a = "1";
    }
    try{ double number = Integer.parseInt(a);
        out.print(a + "的平方根: " + Math.sqrt(number));
    }
    catch(NumberFormatException e){
        out.print("< BR>" + "请输入数字字符");
    }
%>
```

上述 example2_10.jsp 等同于下述 JSP 文件: example2_10_1.jsp。

example2_10_1.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>< BODY Bgcolor = cyan>< FONT size = 1>
<P>请输入一个正数,单击按钮求这个数的平方根。
<FORM action = "" method = post name = form>
< INPUT type = "text" name = "ok">
< BR>< INPUT TYPE = "submit" value = "送出" name = submit>
</FORM>
<% String a = request.getParameter("ok");
    if(a == null){
        a = "1";
    }
    try{ double number = Integer.parseInt(a);
        out.print(a + "的平方根: " + Math.sqrt(number));
    }
    catch(NumberFormatException e){
        out.print("< BR>" + "请输入数字字符");
    }
%>
</FONT></BODY></HTML>
```

2.7 JSP 动作标记

动作标记是一种特殊的标记,它影响 JSP 运行时的功能。

2.7.1 include 动作标记

include 动作标记语法格式为:

```
<jsp: include page = "文件的 URL"/>
```

或

```
<jsp: include page = "文件的 URL">
    param 子标记
</jsp: include>
```

需要注意的是,当 include 动作标记不需要 param 子标记时,必须使用上述第一种形式。

include 动作标记告诉 JSP 页面动态加载一个文件。与静态插入文件的 include 指令标记不同,当 JSP 引擎把 JSP 页面转译成 Java 文件时,不把 JSP 页面中动作指令 include 所指定的文件与原 JSP 页面合并一个新的 JSP 页面,而是告诉 Java 解释器,这个文件在 JSP 运行时(JSP 页面对应的字节码文件被加载执行)才被处理。如果包含的文件是普通的文本文件,就将文件的内容发送到用户端,由用户端负责显示;如果包含的文件是 JSP 文件,JSP 引擎就执行这个文件,然后将执行的结果发送到用户端,并由用户端负责显示这些结果。

使用 include 动作标记也可以实现代码的复用,尽管 include 动作标记和 include 指令标记的作用都是处理所需要的文件,但是处理方式和处理时间上是不同的。include 指令标记是在编译阶段就处理所需要的文件,被处理的文件在逻辑和语法上依赖于当前 JSP 页面,其优点是页面的执行速度快;而 include 动作标记是在 JSP 页面运行时才处理文件,被处理的文件在逻辑和语法上独立于当前 JSP 页面,其优点是可以使用 param 子标记更加灵活地处理所需要的文件(见后面的 param 标记),缺点是执行速度要慢一些。

书写 include 动作标记<jsp: include/>时要注意:jsp:、include 三者之间不要有空格。

例 2-11 中的 example2_11.jsp 页面动态加载两个文件:imageCar.html 和 car.txt。example2_11.jsp 页面保存在 Web 服务目录 ch2 中。example2_11.jsp 页面要动态加载的 imageCar.html 文件以及 imageCar.html 文件所使用的图像文件 car.jpg 均保存在 ch2 中;example2_11.jsp 页面要动态加载的 car.txt 文件保存在 ch2 的子目录 Myfile 中。

在浏览器的地址栏输入: http://127.0.0.1:8080/ch2/example2_11.jsp,页面的效果如图 2-10 所示。

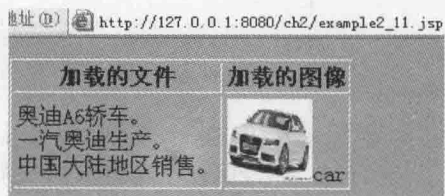


图 2-10 使用 include 动作标记

例 2-11

car.txt

奥迪 A6 轿车。

一汽奥迪生产。

中国大陆地区销售。

imageCar.html

```
< image src = "car.jpg" width = 60 height = 60 > car </iame>
```

example2_11.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY BGCOLOR = Cyan><FONT Size = 2>
  <table border = 1>
    <tr><th>加载的文件</th>
      <th>加载的图像</th>
    </tr>
    <tr><td><jsp: include page = "Myfile/car.txt" /></td>
      <td><jsp: include page = "imageCar.html" /></td>
    </tr>
  </table>
</FONT></BODY></HTML>
```

2.7.2 param 动作标记

param 标记以“名字-值”对的形式为其他标记提供附加信息,这个标记与 jsp: include、jsp: forward、jsp: plugin 标记一起使用。

param 动作标记:

```
<jsp: param name = "属性名字" value = "属性的值" />
```

param 标记不能独立使用,需作为 jsp: include、jsp: forward、jsp: plugin 标记的子标记来使用。

当该标记与 jsp: include 标记一起使用时,可以将 param 标记中的值传递到 include 指令要加载的文件中去。也就是说,JSP 页面在使用 include 动作标记加载文件时,可以动态地向所加载文件传递数据。相对使用 include 指令标记,JSP 页面通过使用 include 动作标记能更为灵活地处理所需要的文件。

例 2-12 中,example2_12.jsp 页面使用 include 动作标记动态加载文件 tom.jsp,当 tom.jsp 文件被加载时获取 example2_12.jsp 页面中 include 动作标记的 param 子标记中 name 属性的值(tom.jsp 文件使用 Tomcat 服务器提供的 request 内置对象获取 param 子标记中 name 属性的值,有关内置对象见后面的第 4 章)。

example2_12.jsp 和 tom.jsp 页面保存在 Web 服务目录 ch2 中。在浏览器的地址栏输入: http://127.0.0.1:8080/ch2/example2_12.jsp,页面的效果如图 2-11 所示。

地址 http://127.0.0.1:8080/ch2/example2_12.jsp

加载文件效果:

从1到300的连续和是: 45150

图 2-11 动态加载文件计算连续和

例 2-12

example2_12.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>< BODY>
<P>加载文件效果:
    <jsp: include page = "tom.jsp">
        <jsp: param name = "computer" value = "300" />
    </jsp: include>
</BODY></HTML>
```

tom.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>< BODY>
    <% String str = request.getParameter("computer"); //获取 example2_12.jsp 中 param 子标记中
        int n = Integer.parseInt(str);                //name 属性的值
        int sum = 0;
        for(int i = 1; i <= n; i++)
            sum = sum + i;
    %>
<P> 从 1 到<% = n %>的连续和是: <% = sum %>
</BODY></HTML>
```

2.7.3 forward 动作标记

forward 动作标记的语法格式:

```
<jsp: forward page = "要转向的页面" />
```


或

```
<jsp: forward page = "要转向的页面" >
    param 子标记
</jsp: forward>
```

该指令的作用是: 从该指令处停止当前页面的继续执行, 转向执行该指令中 page 属性指定的 JSP 页面, 但是在浏览器的地址栏中并不显示 forward 指令所转向的 JSP 页面的 URL 表示, 浏览器的地址栏仍然显示 forward 转向前的 JSP 页面的 URL 表示。

需要注意的是, 当 forward 动作标记不需要 param 子标记时, 必须使用上述第一种形式。forward 标记可以使用 param 动作标记作为子标记, 以便向所转向的页面传送信息。forward 动作标记指定的所转向的 JSP 文件可以使用 Tomcat 服务器提供的 request 内置对象获取 param 子标记中 name 属性的值。

例 2-13 中的 example2_13.jsp 页面使用 forward 动作标记转向 come.jsp 页面, 并向 come.jsp 页面传递一个数值。example2_13.jsp 和 come.jsp 页面保存在 Web 服务目录 ch2 中。在浏览器的地址栏输入: http://127.0.0.1:8080/ch2/example2_13.jsp, 页面的效果如图 2-12 所示。

地址 ①  http://127.0.0.1:8080/ch2/example2_13.jsp

您传过来的数值是:
0.026984719363912002

图 2-12 向转向的页面传递数据

例 2-13

example2_13.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY>
<% double i = Math.random();
%>
<jsp: forward page = "come.jsp" >
    <jsp: param name = "number" value = "<% = i %>" />
</jsp: forward>
</BODY></HTML>
```

come.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY bgcolor = cyan><FONT Size = 5>
<% String str = request.getParameter("number");
double n = Double.parseDouble(str);
%>
<P>你传过来的数值是: <BR>
<% = n %>
</FONT></BODY></HTML>
```

2.7.4 plugin 动作标记

在页面中使用普通的 HTML 标记<applet></applet>可以让用户下载运行一个 Java applet 小程序,但并不是所有用户的浏览器都支持 Java applet 小程序。如果 Java applet 小程序使用了 JDK1.2 以后的类,那么,有些浏览器并不支持这个 Java 小应用程序,而使用 plugin 动作标记可以保证用户能执行小应用程序。

该动作标记指示 JSP 页面加载 Java plugin 插件。该插件由用户负责下载,并使用该插件来运行 Java applet 小程序。

plugin 动作标记:

```
<jsp: plugin type = "applet" code = "小程序的字节码文件"
jreversion = "Java 虚拟机版本号" width = "小程序宽度值" height = "小程序高度值" >
<jsp: fallback>
    提示信息: 用来提示用户的浏览器是否支持插件下载
</jsp: fallback>
</jsp: plugin>
```

假设有一个 Java applet 小程序,主类字节码文件是 B.class,该文件存放在 Web 服务目录 ch2 中,含有 plugin 动作标记的 JSP 文件 example2_14.jsp 也存放在 ch2 中。

例 2-14

example2_14.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY>
<jsp: plugin type = "applet" code = "B.class" jreversion = "1.2" width = "200" height = "260" >
    <jsp: fallback>
```

Plugin tag OBJECT or EMBED not supported by browser.

</jsp: fallback>

</jsp: plugin>

</BODY></HTML>

当用户访问上述 JSP 页面时,将导致登录 Sun 公司的网站下载 Java plugin 插件,出现用户选择是否下载插件的界面。用户下载插件完毕后,接受许可协议,就可以根据向导一步一步地安装插件。安装完毕后,小程序就开始用 Java 的虚拟机(不再使用浏览器自带的虚拟机)加载执行 Java applet 小程序。以后用户再访问带有 plugin 动作标记的 JSP 页面就能直接执行该页面中包含的 Java applet 小程序。

2.7.5 useBean 动作标记

该标记用来创建并使用一个 JavaBean,是非常重要的一个动作标记,将在第 7 章详细讨论它的用法和作用。Sun 公司倡导:用 HTML 完成 JSP 页面的静态部分,用 JavaBean 完成动态部分,实现真正意义上的静态和动态之分割。

2.8 实验 1: JSP 页面的基本结构

1. 相关知识点

一个 JSP 页面可由普通的 HTML 标记、JSP 标记、成员变量和方法的声明、Java 程序片和 Java 表达式组成。JSP 引擎把 JSP 页面中的 HTML 标记交给用户的浏览器执行显示;JSP 引擎负责处理 JSP 标记、变量和方法声明;JSP 引擎负责运行 Java 程序片、计算 Java 表达式,并将需要显示的结果发送给用户的浏览器。

JSP 页面中的成员变量是被所有用户共享的变量。Java 程序片可以操作成员变量,任何一个用户对 JSP 页面成员变量操作的结果,都会影响到其他用户。如果多个用户访问一个 JSP 页面,那么该页面中的 Java 程序片就会被执行多次,分别运行在不同的线程中,即运行在不同的时间片内。运行在不同线程中的 Java 程序片的局部变量互不干扰,即一个用户改变 Java 程序片中的局部变量的值不会影响其他用户的 Java 程序片中的局部变量。

2. 实验目的

本实验的目的是让学生掌握怎样在 JSP 页面中使用成员变量,怎样使用 Java 程序片、Java 表达式。

3. 实验要求

编写两个 JSP 页面,名字分别为 inputName.jsp 和 people.jsp。

1) inputName.jsp 的具体要求

该页面有一个表单,用户通过该表单输入自己的姓名并提交给 people.jsp 页面, inputName.jsp 的效果如图 2-13(a)所示。

2) people.jsp 的具体要求

(1) JSP 页面有名字为 personList、类型是 StringBuffer 以及名字是 count、类型为 int 的成员变量。

(2) JSP 页面有 public void judge ()方法。该方法负责创建 personList 对象,当 count

的值是 0 时, judge () 方法创建 personList 对象。

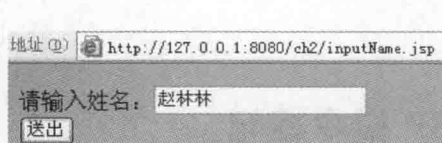
(3) JSP 页面有 public void addPerson(String p) 的方法, 该方法将参数 p 指定的字符串尾加到成员变量 personList, 同时将 count 作自增运算。

(4) JSP 页面在程序片中获取 inputName.jsp 页面提交的姓名, 然后调用 judge() 创建 personList 对象, 调用 addPerson 方法将用户的姓名尾加到成员变量 personList 中。

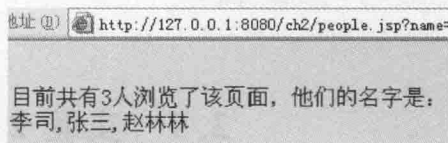
(5) 如果 inputName.jsp 页面没有提交姓名, 或姓名含有的字符个数大于 10, 就使用 <jsp:forward page="要转向的页面" /> 标记将用户转到 inputName.jsp 页面。

(6) 通过 Java 表达式输出 person 和 count 的值。

people.jsp 的效果如图 2-13(b) 所示。



(a) inputName.jsp 页面的效果



(b) people.jsp 页面的效果

图 2-13 两个页面的效果图

4. 参考代码

代码仅供参考, 学生可按着实验要求, 参考本代码编写代码。

JSP 页面参考代码如下:

inputName.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY bgcolor = cyan><FONT Size = 3>
  <FORM action = "people.jsp" method = get name = form>
    请输入姓名: <INPUT type = "text" name = "name">
    <BR><INPUT TYPE = "submit" value = "送出" name = submit>
  </FORM>
</BODY></HTML>
```

people.jsp

```
<% @ page contentType = "text/html; charset = GB2312" %>
<HTML><BODY BGCOLOR = yellow><FONT Size = 3>
  <% ! int count;
    StringBuffer personList;
    public void judge(){
      if(count == 0)
        personList = new StringBuffer();
    }
    public void addPerson(String p){
      if(count == 0)
        personList.append(p);
      else
        personList.append(", " + p);
      count ++ ;
    }
  %>
```

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

备用QQ:2404062482