有关此电子图书的说明

本人由于一些便利条件，可以帮您提供各种中文电子图书资料，且质量均为清晰的 PDF 图片格式，质量要高于网上大量传播的一些超星 PDG 的图书。方便阅读和携带。只要图书不是太新，文学、法律、计算机、人文、经济、医学、工业、学术等方面的图书，我都可以帮您找到电子版本。所以，当你想要看什么图书时，可以联系我。我的 QQ 是：85013855，大家可以在 QQ 上联系我。

此 PDF 文件为本人亲自制作，请各位爱书之人尊重个人劳动，敬请您不要修改此 PDF 文件。因为这些图书都是有版权的，请各位怜惜电子图书资源，不要随意传播，否则，这些资源更难以得到。

征服 Ajax

Web 2.0

快速入门与项目实践

（Java）

张桂元　贾燕枫　姜波　编著

人民邮电出版社
POSTS & TELECOM PRESS

征服 Ajax——Web 2.0 快速入门与项目实践（Java）

张桂元　贾燕枫　姜　波　编著

人 民 邮 电 出 版 社

图书在版编目（CIP）数据

征服 Ajax．Web 2.0 快速入门与项目实践．java / 张桂元，贾燕枫，姜波编著．
—北京：人民邮电出版社，2006.6

ISBN 7-115-14803-1

Ⅰ．征… Ⅱ．①张… ②贾… ③姜… Ⅲ．①计算机网络－程序设计 ②JAVA 语言－程序设计 Ⅳ．TP393.09②TP312

中国版本图书馆 CIP 数据核字（2006）第 051912 号

内 容 提 要

书中介绍了在 Web 2.0 的标准下结合 Ajax 技术进行各种 Java Web 应用开发的方式和技巧。本书的特色是以案例为主，将相关知识点作为主线贯穿其中。第 1 部分是 Ajax 开发基础，内容涉及搭建运行环境和开发平台、Ajax 核心技术及工作流程、Ajax 应用技巧。第 2 部分提供了一个完整的博客在线网站的设计和开发实例，通过对该网站各个功能模块从需求分析、系统设计及部署实现等全过程的分析，使读者能够快速掌握基于 Eclipse 和 MySQL 开发平台，并结合 Ajax、JSP、Servlet、MVC、Hibernate 等主流设计模式及开发技术的技能。

本书结构清晰、注重实用、深入浅出，非常适合作为 Eclipse 中 Ajax 应用开发的自学或培训教材，同时，也适合作为 Java Web 开发人员的参考用书。

征服 Ajax——Web 2.0 快速入门与项目实践（Java）

◆ 编　　著　张桂元　贾燕枫　姜　波
责任编辑　屈艳莲

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000　1/16
印张：19.5
字数：439 千字　　2006 年 6 月第 1 版
印数：1－5 000 册　　2006 年 6 月北京第 1 次印刷

ISBN 7-115-14803-1/TP・5414

定价：36.00 元（附光盘）

读者服务热线：（010）67132692　印装质量热线：（010）67129223

PREFACE

前　言

关于本书

在前期 Web 技术发展的历程中，软件系统所采用的计算技术由早期的基于 C/S（Client/Server）模式的应用系统向基于 B/S（Browser/Server）模式的应用系统进行了转变。这种转变成功的使开发人员避开了繁琐而没有太多技术内涵的部署工作，即不需要将客户端程序不厌其烦地安装到数以万计的客户端（胖客户）中。用户端只需要提供浏览器（瘦客户）就可以方便地显示服务器端的处理结果。由于这种方式所采用的是同步交互方式，因此带来的不良后果就是客户端的响应速度变慢。

而在 Ajax 的帮助下，传统 Web 应用中的请求/响应模式发生了改变，开发人员可以利用这一技术实现与服务器端进行异步交互，用户的直接感受就是不会因为单击一个按钮就必须等待整个 Web 页面的全部刷新。而对于开发人员来说，也不再为了要实现一个动态效果，就将服务器端数据库中的数据全部在页面装载时发送到客户端，导致页面过度膨胀。总的来说，Ajax 提供了非常方便的客户端处理方式，为实现更加丰富的网页交互效果奠定了基础。

本书通过深入浅出的讲解和丰富的案例，引领着读者学习 Ajax 技术中各种不同 Web 应用程序的开发方式和开发技巧。同时通过大量实际的模块案例，让读者体会在实际项目开发中应用 Ajax 及相关技术的实际过程。

本书主要有以下两个特点。

- 容易上手：本书作者多年从事教学和 Web 应用的开发工作。因此，本书从学生的认知规律入手，引导读者逐步熟悉体验 Ajax 技术，从而让读者容易上手，迅速掌握 Ajax 技术与原有服务器端 Java 应用程序的开发和设计技巧。
- 实战性强：书中的案例是作者对自己教学和开发经验的总结，在实际开发中，读者可以将本书提供的案例稍微改造一下就可以直接使用。

本书读者对象

本书介绍了 Web 2.0 中 Ajax 技术的开发方法和技巧，适合从事 Web 应用开发的初中级

程序员使用。通过对本书的学习，Web 应用的开发人员可以迅速掌握 Ajax 技术的基本应用。本书也可作为 IT 培训机构的培训教材。

本书内容组织方式

本书共分为两个部分，第 1 部分为第 1～5 章。第 2 部分为第 6～8 章。

第 1 部分主要是结合各种实际应用的实例模块，介绍利用 Ajax 技术开发各种不同 Web 应用程序的方式和开发技巧。其中，第 1 章介绍了 Web 2.0 的概念和 Ajax 技术与传统客户端技术的不同之处。第 2 章通过一个实用的验证模块介绍了 Ajax 技术的基本开发方式和应用过程。第 3 章介绍了与 Ajax 相关的一些核心技术，包括 JavaScript、XMLHttpRequest 对象、XML、DOM 等。第 4 章通过一系列 Web 应用中实际效果的实现，介绍了使用 Ajax 技术开发 Java Web 应用的过程和技巧。第 5 章对于与 Ajax 相关的一些技术进行了深入的探讨。

第 2 部分全面系统地介绍了一个完整的博客在线网站（Blog Online）的设计开发过程。其中，第 6 章介绍了整个博客网站的总体设计架构及设计原则。第 7 章详细介绍了博客在线网站前台模块的设计实现，包括博客的申请、日志的发表及管理等。第 8 章介绍了博客在线网站后台维护管理中心涉及到的各个功能模块的具体实现过程。这些功能实际上都是博客网站系统中通用的模块，因此结合这一部分案例的设计和制作过程，读者将体验 Web 应用开发的各个环节，并且能够将各个功能模块方便地移植到实际的 Web 系统中。

本书的约定

本书中提供了大量基于 Ajax 技术的 Web 应用系统完整模块实现的源代码，为了能够使读者在阅读时更快地抓住重点，本书中进行了如下相关的设计。

- 阶段性案例设置：在本书第 1 部分的各个章节中，结合相关的核心知识点，设计完成了大量功能模块性案例，通过这些案例的设计实现，读者能够为第 2 部分，设计实现完整的商务网站项目做好准备。
- 重点代码部分加黑显示：因为网页的前台部分往往由静态设计师完成，为了能够在大量代码中突出对后台程序员最有用的部分，本书对其进行了加黑处理，以方便读者迅速抓住重点。
- 必知必会栏目的设置：考虑到缺乏实际开发经验的读者，在进行案例设计时，缺乏解决问题的经验以及编程的思路，本书在一些关键性位置结合实例讲解了解决问题的思路以及技巧，以帮助读者积累开发经验，寻找程序员的思维习惯。

与作者联系

本书的编写得到了许多优秀 Web 程序开发人员的积极建议和大力支持，在此向他们表示真挚的感谢。

由于时间仓促，加之水平有限，书中的不足之处在所难免，敬请读者批评指正。本书责任编辑的联系方法是 quyanlian2@ptpress.com.cn，欢迎来信交流。

编者

2005 年 5 月

CONTENTS

目　录

CHAPTER 1

第 1 章　Ajax 概述

1.1　Web 2.0 与 Ajax 简介

作为 2005 年互联网世界中非常热门的词汇，Web 2.0 与 Ajax 两个词频频在各种专业媒体的技术文章中出现。最初，许多人总是将他们与最新的观念、最热的技术等形容词关联在一起。事实上，从专业的角度来看，两个热门词汇并不完完全全是一种全新的理念、全新的技术。更符合实际的说法应该是一种观念上的回归，技术上的融合，就像生活中各种流行的趋势一样，它在发展到一定阶段时必将实现自然的回归。Web 2.0 概念的出现，Ajax 技术的日益流行，都是互联网发展到今天的一种必然。

1.1.1　Web 2.0 简介

2005 年 Web 2.0 成了人们关注的焦点。无论是在搜索引擎网站的热门关键字中，还是在众多 Blogger（博客）的日志描述中，Web 2.0 都是一个频繁出现的词汇。伴随着 Web 2.0 的诞生，互联网进入了一个更加开放、交互性更强、由用户决定内容并参与共同建设的可读写网络阶段。早期的 Web 用户大多数是被动接受网络提供的相关内容，Web 2.0 更多提倡的理念是 Web 服务的亲和力，可操作性，即更多地关注用户的体验，用户的感受。

Web 2.0 的出现则意味着网络已经成为了一个开发环境，开发人员可以借助 Web 服务提供的编程接口进行服务上的拓展，从某种意义上说，网站已经变成了开发人员软件设计过程中的一个构件。开发人员在 Web 页面上实现原来桌面程序的那些便利效果，从而使用户可以更加方便地参与。

必知必会：Web 2.0 的定义

Web 2.0 代表的是一个新的网络阶段，它本身并没有特别明确的标准来进行描述，一般我们将促成这个阶段的各种技术和相关的产品服务统称为 Web 2.0，例如：Ajax 就是这一系列技术和产品服务中非常重要的成员．此外还包括博客、数据独立性等各种网络服务方式。

1.1.2　什么是 Ajax

Ajax 是 Web 2.0 阶段系列技术和相关产品服务中非常重要的一种技术。其全称为异步

JavaScript 和 XML（即 Asynchronous JavaScript and XML），从中可以看到与 Ajax 直接相关的几个技术点：一是异步、二是 JavaScript、三是 XML。这几点恰恰集中反映了这项技术关注的两个问题：一是借助异步 JavaScript 实现浏览器和服务器之间的异步交互，如无需重新装载整个页面就可以向服务器发送请求，并接受响应。二是对 XML 文档的解析和处理。

实际上，Ajax 本身并不是一种全新的技术，但是随着 Web 应用中可交互性、可参与性、人性化设计需求的提高，Ajax 在目前的 Web 应用开发过程中已经迅速成为客户端炙手可热的技术，那么 Ajax 的最大优势是什么呢？

必知必会：Ajax 的优势

在前期 Web 技术发展的历程中，软件系统所采用的计算技术由早期的基于 C/S（Client/Server）模式应用系统向基于 B/S（Browser/Server）模式应用系统进行了转变。这种转变成功地使开发人员避开了繁琐但没有太多技术内涵的部署工作，即不需要将客户端程序不厌其烦地安装到数以万计的客户端（胖客户）中去。用户端只需要提供浏览器（瘦客户）就可以方便的显示服务器端的处理结果。但是，由于这种方式所采用的是同步交互方式，因此带来的直接后果就是束缚了用户的手脚，损失了浏览器端和服务器端的交互性。

在 Ajax 的帮助下，传统 Web 应用中的请求/响应模式发生了改变，开发人员可以凭借这一技术自由地与服务器端实现异步交互，用户最直接感受就是不会因为要单击一个按钮就要等待整个 Web 页面的全部刷新。开发人员也不再为了要实现一个动态效果，就将服务器端数据库中的数据在页面装载时全部发送到客户端，导致页面过度膨胀。Ajax 提供了非常方便的客户端处理方式，为在 Web 应用中实现更加丰富的交互效果奠定了基础。

作为一种客户端技术，Ajax 技术的应用实现了浏览器客户端和服务器之间的异步交互，即相当于在 Web 应用中实现了原来 C/S（Client/Server）结构下的交互效果，且这种效果的实现并没有同时带来繁琐的客户端部署工作。

1.2 Ajax 的结构及其意义

Ajax 的出现是因为：随着信息传输量的不断加大，传统的 Web 应用所采用的同步交互方式显现出越来越明显的问题。当服务器端处理请求时，浏览器端的用户就必须要等待，只有到最终的响应结果传输到浏览器客户端时，整个页面才会重新进行刷新，以显示处理的结果。

这种处理方式让用户的体验变得不连贯、不顺畅。Ajax 提倡的异步交互的处理方式则能够很好的解决这个问题。为了让读者更清楚地知道同步交互方式和异步交互方式地根本区别，下面对传统的处理方式以及 Ajax 提供的处理方式进行一个比较。

1.2.1 传统 Web 应用解决方案

传统的 Web 应用是同步交互的方式。这种同步交互方式的处理过程如图 1-1 所示。

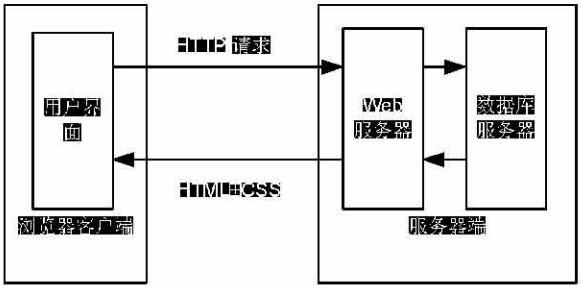


图 1-1 同步交互方式

必知必会：同步交互方式

当用户向 HTTP 服务器提交了一个处理请求时，服务器端将在接受到该请求后，按照预先编写的程序中的业务逻辑进行处理，例如：和数据库服务器之间进行数据信息的交换，然后对请求处理进行响应，即将结果传送回发出请求的浏览器客户端，返回一个 HTML 页面在浏览器端进行显示，在显示该页面时，往往使用 CSS 层叠样式表丰富页面的显示效果。

显然，这样的一种处理方式一定会给用户一种不连贯的体验，因为当服务器在处理请求的时候，用户多数时间只能处于等待状态，页面中显示的内容也只能是一片空白。

1.2.2 Ajax 解决方案的优势

与传统的 Web 应用不同，Ajax 采用的是一种异步交互的处理方式。这种异步交互方式的处理过程如图 1-2 所示。

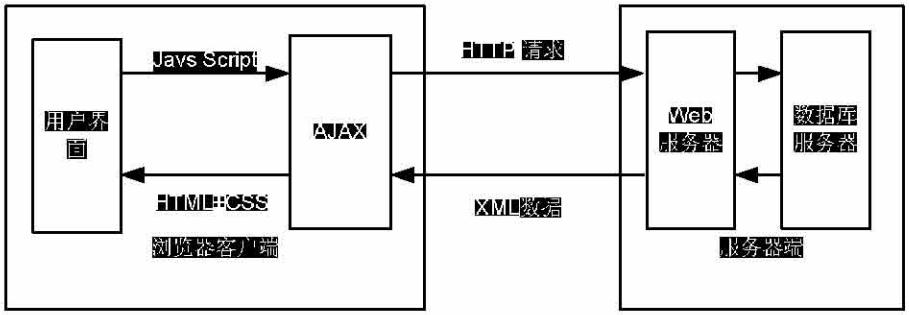


图 1-2 异步交互方式

必知必会：异步交互方式

Ajax 相当于在浏览器客户端与服务器之间架设了一个桥梁、一个媒介，在它的帮助下，可以消除网络交互过程中的处理—等待—处理—等待等缺陷。在处理过程中 Web 服务器响应是标准的且易于解析的 XML 格式的数据传递给 Ajax，然后再转换成 HTML 页面的格式，辅助 CSS 进行显示。

Ajax 相关的一个重要对象是 XMLHttpRequest 对象（XHR），这个对象从 IE 5.0 浏览器起就已经存在了。Ajax 允许用户与服务器端交互异步进行，即独立于用户与服务器之间的直接交流。在这种异步交互的过程中，可以使用 JavaScript 调用 Ajax 来代替传统提交请求的方式，内存中的数据编辑、页面导航、数据校验等处理都不再需要重新载入整个页面的需求，这些工作都可以交给 Ajax 单独来执行。

Ajax 是使用 XMLHttpRequest 对象发送请求并获得服务器端的响应，同时 Ajax 可以在不重新载入整个页面的情况下用 JavaScript 操作 DOM 以实现最终更新页面。因此在读取数据的过程中，用户所面对的不是白屏，而是原来的页面内容。这种更新是瞬间的，用户几乎感觉不到，对用户来讲这是一种连贯的感受。

借助 Ajax，可以把以前一些原本由服务器负担的工作转移到客户端来完成，利用客户端闲置的能力来进行处理，这样也可以有效地减轻服务器和带宽的负担，节约空间和宽带租用成本。

1.2.3 Ajax 相关技术简介

Ajax 实际上是各种技术的一种综合应用，这其中包括 JavaScript 脚本、XHTML 和 CSS、DOM、XML 和 XSTL，以及最重要的一个对象 XMLHttpRequest。

开发人员可以使用 XHTML 和 CSS 实现数据信息的统一化、标准化显示；使用 DOM 实现浏览器端丰富的动态显示效果以及与服务器端的交互；使用 XML 和 XSTL 进行浏览器和服务器两端的数据信息交换与处理；使用 XMLHttpRequest 对象进行浏览器和服务器端的异步数据读取；使用 Javascript 脚本实现对所有数据进一步处理。

为了让读者对 Ajax 有一个概要的了解，下面先概括介绍一下 Ajax 技术中这几个最核心的技术，在本书后续章节会对这些相关的技术进行详细的介绍和举例。

1. JavaScript 脚本

JavaScript 是一种可以与 HTML 标记语言混合使用的脚本语言，其编写的程序可以直接在浏览器中解释执行。因此使用脚本语言编写的程序可以在浏览器的支持下跨平台运行。在 JavaScript 脚本中可以调用浏览器及 Ajax 中提供的相关对象。利用这些对象提供的属性和方法可以实现页面效果的动态控制。

需要注意的是：对于很多初学者而言，往往会将 JavaScript 脚本语言与 Java 编程语言混为一谈。实际上从本质上来说，这二者并没有什么必然的联系。

2. XHTML 和 CSS

XHTML 的全称是可扩展的超文本标记语言（Extensible HyperText Markup Language），是一种为适应 XML 可扩展标记语言（Extensible Markup Language）而重新改造的 HTML 超文本标记语言（HyperText Markup Language）。

从本质上说，XHTML 是一种过渡技术，它结合了 XML 中的部分强大功能及 HTML 中

大部分的简单特性。

相对原来的 HTML 来讲，XHTML 从设计上显得更加严密，其与 XML 关系决定了它的用户可以很自然地实现从 HTML 到 XML 的转换。

CSS 层叠样式表是为了弥补 HTML 超文本标记语言在格式修饰中的不足，同时也为了能够实现页面格式的批量动态更新。为了进一步丰富页面的动态效果，通过使用脚本语言与 CSS 结合，动态控制页面元素的位置、色彩等属性。因此了解 CSS 的常用属性和设置方式是脚本编程中所需要的。

3. DOM

DOM 的全称是文档对象模型（Document Object Model），它在本质上是一种文档平台。借助 DOM，允许结合脚本语言实现动态的数据存储以及动态上传文件的内容、结构以及样式。DOM 是 W3C DOM 工作组提倡的一种接口规范，它是独立于访问、解析或更新 XML 数据机制的一种应用。通俗地说，DOM 是方便开发人员对 XML 文档进行操作的分层模型。有足够清楚的层次模型，才会得到足够灵活的数据操作。

DOM 是以层次结构组织的节点或信息的集合。这个层次结构允许开发人员在结构树中导航寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做其他工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。

DOM 还提供了一个 API，允许开发人员添加、编辑、移动或删除树中任意位置的节点，从而创建一个应用程序。总之，借助 DOM，开发人员可以有层次的处理原本纷乱的数据对象，从而实现清晰化、条理化的数据查询和处理。

4. XML 和 XSTL

XML 可扩展标记语言（即 Extensible Markup Language）。HTML 是一种标记语言，同时，HTML 里面有很多标签，类似<head>、<table>等，都是在 HTML 4.0 里规范和定义的，而 XML 里允许开发人员自行创建这样的标签，所以称为可扩展的。

XML 的优势之一是它允许各个组织、个人建立适合自己需要的标记符号。这一特征使得 XML 可以在电子商务、政府文档、司法、出版、保险机构、厂商和中介组织信息交换等领域中一展身手，针对不同的系统、厂商提供各具特色的独立解决方案。XML 的最大优点就在于它的数据存储格式不受显示格式的制约。

正是基于上述的原因，XML 广泛运用在软件系统的配置文件中，同时也日益成为因特网上的标准数据存储格式和交换格式，即一种通信的标准。通俗的理解，XML 是一种程序之间交换原始数据的简单而标准的方法。它的成功并不在于它容易被人们书写和阅读，更重要的是，它从根本上解决了不同应用系统之间的通用信息交换方式。

XSTL 的全称是可扩展的样式表转换语言（即 Extensible Stylesheet Transformation Language），它是一种用来转换 XML 文档结构的语言。为了使数据便于阅读理解，经常需要将信息显示出来或者打印出来，例如将数据转换成一个 HTML 文件，一个 PDF 文件，甚至

是一段声音。同样，为了使数据适合不同的应用程序，就必须能够将一种数据格式转换为另一种数据格式，比如需求格式可能是一个文本文件，一个 SQL 语句，一个 HTTP 信息，一定顺序的数据调用等。而 XSLT 就是用来实现这种转换功能的语言。XSLT 最主要的功能就是将 XML 转换为 HTML。

由于 CSS 只适合用于输出比较固定的最终文档。CSS 的优点是简洁，消耗系统资源少；而 XSLT 虽然功能强大，但因为要重新索引 XML 结构树，所以消耗内存比较多。因此，在实际使用中常常会将它们结合起来使用，比如在服务器端用 XSLT 处理文档，在客户端用 CSS 来控制显示。

5. XMLHttpRequest

XMLHttpRequest 是 Ajax 技术中最重要的一个对象。一个页面可以在不刷新的情况下通过一个 XMLHttpRequest 发送请求来获取服务器响应。开发人员也可以通过使用该对象在不刷新当前页面的情况下更新页面中的数据。

XMLHttpRequest 是浏览器中已定义好的对象，它是 AJAX 技术的核心组成部分，JavaScript 通过它和服务器之间进行通信，并通过它来解析从服务器传回来的 XML 文件。

XMLHttpRequest 对象最早是由微软作为一个 ActiveX 对象实现的，之后其他流行的浏览器都步其后尘实现了对 XMLHttpRequest 对象的支持。需要注意的是：在定义 XMLHttpRequest 对象时，IE 浏览器和别的浏览器会有所不同。在本书的第 3 章将会对该对象相关的属性和方法进行详细的介绍。

总之，Ajax 实际上是上述这些技术的综合运用，即基于 XHTML 和 CSS，由 DOM 实现动态显示与交互，借助 XML 和 XSLT 进行数据交换及处理，而在这个过程中则是使用 JavaScript 来进行整合。

通过本章的学习，相信读者对 Web 2.0 以及 Ajax 技术有了一个整体了解，在本书后续章节中将会对 Ajax 相关的技术进行深入的介绍和说明。

CHAPTER 2

第 2 章　Ajax 开发初体验

为了能够让读者在短时间内熟悉 Ajax 结合 Java 技术开发 Web 应用的基本流程，本章将通过典型的 Web 应用程序介绍一下基于 Ajax 进行 Web 程序开发的基本步骤。使初学者初步了解基本开发环境的搭建过程，及在浏览器客户端和服务器端进行开发的工作过程。

该程序完成的是商务网站设计中常见的身份验证应用模块，即对于登录电子商务系统的用户进行身份验证。为了能够在开发之初集中初学者的注意力，本章先实现与数据库服务器无关的效果，然后再关联到数据库服务器，完成从浏览器到 Ajax、从 Ajax 到 Web 服务器、最终到数据库服务器的处理。

同时为了让读者能够直观的体会到传统实现方式和 Ajax 实现方式的不同，本例中将先按照传统的方式实现，再采用 Ajax 的方式来实现，使读者更直观地体会其不同的效果。

2.1　开发环境搭建

在第 1 章中曾经提到，Ajax 大多数核心组件都可以在 IE 浏览器环境下直接使用。因此 Ajax 并不需要特别的安装。由于本书服务器端采用的是 J2EE，所以需要首先安装与 Java 开发相关的 API 以及开发工具，本书中使用的数据库管理系统为 MySQL，在本章 2.2.2 小节实现与数据库服务器关联的实例中将进行安装。

必知必会：Ajax 可以与哪些服务器端技术配合使用

与 Ajax 配合的服务器端技术可以有很多种选择，例如：.NET、PHP 及 Ruby 等。作为一种客户端方法，Ajax 并不关心服务器端采用的是哪一种技术。所以对于开发人员而言，这就意味着并不需要学习一种新的语言，也不需要放弃原本熟悉的服务器端技术。要掌握 Ajax，关键是学习一些新的对象、新的属性和方法。

由于本书服务器端选择使用的是 Java 技术，因此开发工具选择使用 Eclipse，同时在进行 Ajax 的应用开发时，需要大量使用 JavaScript，为了调试方便，本书选择使用 MyEclipse 4.1，因为该工具提供了 JavaScript 脚本调试工具。

本书采用的开发环境所对应的软件版本如下所示：

- JDK（Java Development Kits）：1.4；
- Eclipse：3.1；
- MyEclipse：4.1；
- Tomcat：5.0.19；
- MySQL：4.1.14。

下面我们按顺序完成开发环境的搭建。

2.1.1 JDK 的安装

首先需要先安装 JDK（Java Development Kits）。一般要求 JDK 1.2 或以上版本，推荐使用 JDK 1.4 或以上版本，下载地址：http://java.sun.com/j2se。

具体安装步骤如下。

① 运行 J2SDK 安装文件 j2sdk-1.41_02.exe，在弹出的安装向导窗口中，单击图 2-1 所示中的“Next”按钮。

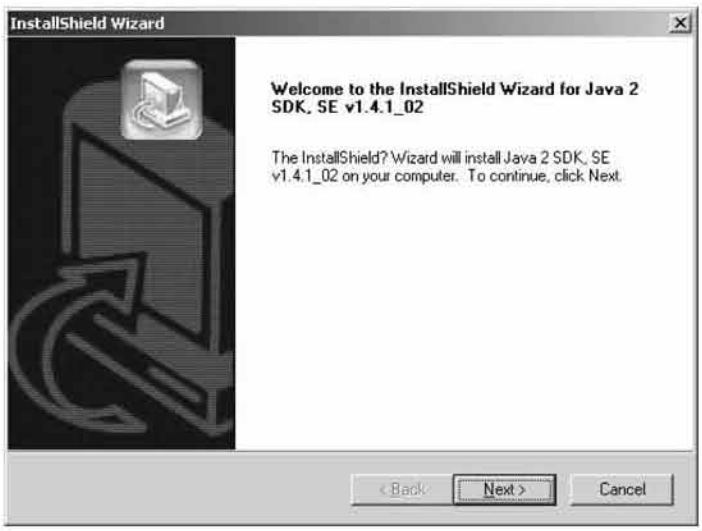


图 2-1 J2SDK 安装向导（Step1）

② 单击如图 2-2 所示的对话框中的“Yes”按钮接受软件许可协议。

③ 在图 2-3 所示的对话框中选择安装的目标目录，此时用户可以不采用默认的安装路径，而通过单击“Browse”按钮指定其他路径，例如 C:\jdk1.4，这样将来配置运行环境时会更简洁一些，但注意必须前后一致。

④ 在接下来的对话框中保持默认设置，单击“Next”按钮，直至出现如图 2-4 所示的完成对话框，单击“Finish”按钮结束安装。

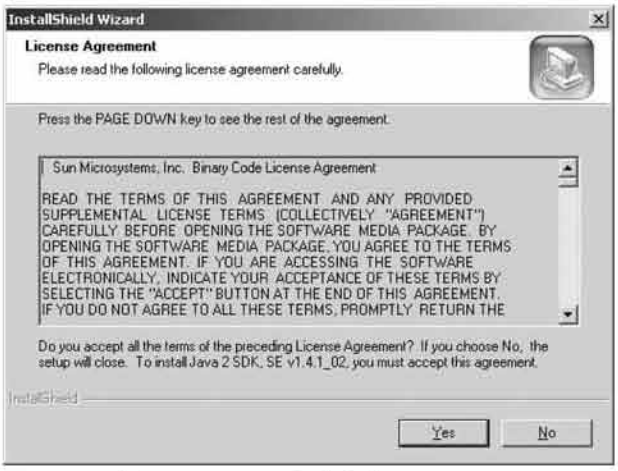


图 2-2 J2SDK 安装向导（Step2）

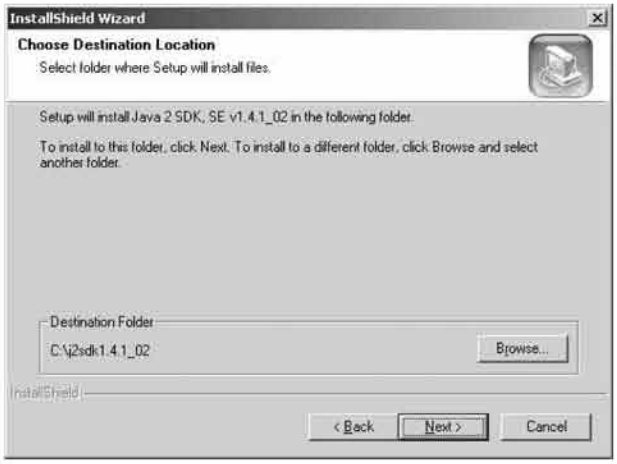


图 2-3 J2SDK 安装向导（Step3）

图 2-4 J2SDK 安装向导（Step4）

2.1.2 JDK 环境变量的设置

在 JDK 安装完成之后，一般需要设置计算机系统的环境变量，以便于其他软件确定 JDK 的安装位置。笔者使用的是 Windows XP 操作系统环境，具体操作步骤如下。

① 选择“开始”菜单中的“控制面板”命令打开控制面板窗口，双击“系统”图标，即可打开如图 2-5 所示的“系统属性”对话框；

② 单击“系统属性”对话框中的“环境变量”按钮，进入如图 2-6 所示的“环境变量”对话框。在该对话框中可以设置只有当前用户登录时才有效的用户变量，也可以设置该系统的所有用户登录时都有效的系统变量。

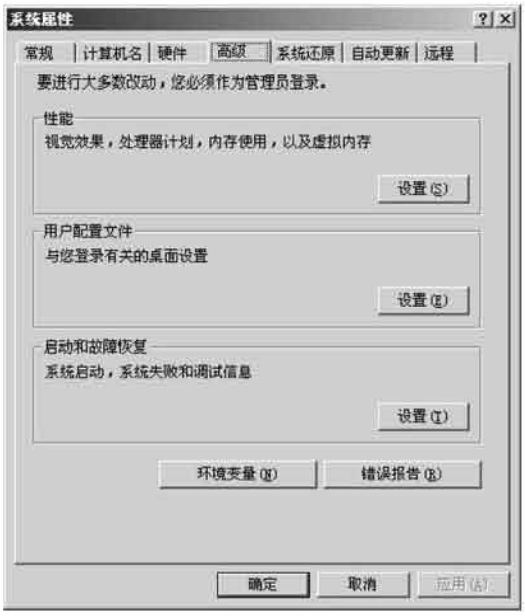


图 2-5 “系统属性”对话框

图 2-6 环境变量对话框

③ 在“环境变量”对话框中单击系统变量框中“新建”按钮，在打开的如图 2-7 所示的新建系统变量对话框中输入变量名为 JAVA_HOME，其环境变量的值应该设置为 JDK 的安装位置，例如 C:\Program Files\Java\jdk1.4。

图 2-7 新建系统变量对话框

④ 然后在如图 2-6 所示的“环境变量”对话框中的用户变量框中的 PATH 环境变量，将 C:\j2sdk1.4.1_02\bin 添加到对应的变量值后面，注意和前面的值用“;”隔开。单击“确定”按钮即可完成 JDK 相关环境变量的设置。

必知必会：设置 CLASSPATH 的意义

最后一个系统变量是 CLASSPATH，Java 虚拟机会根据 CLASSPATH 的设定来搜索 class

文件所在目录，但这不是必需的，开发人员可以在继承开发环境中运行 Java 程序时指定 CLASSPATH，比如在 Eclipse 中运行写好的 Java 程序时，它会自动设定 CLASSPATH，但是为了在控制台环境中能够方便地运行 Java 程序，笔者建议最好还是设置一个 CLASSPATH，把它的值设为“.”，用于代表当前目录。

2.1.3 Eclipse 的安装

Eclipse 是开放源代码的项目，读者可以到 www.eclipse.org 免费下载 Eclipse 的最新版本。其中可以选择下载 Eclipse SDK，这其中包括了 Eclipse 开发环境，Java 开发环境，Plug-in 开发环境，以及所有源代码和文档。

如果只是用 Eclipse 开发 Java 应用，而不是开发 Eclipse 插件或者研究 Eclipse 代码，那么下载一个 Platform Runtime Binary 再加上 JDT Runtime Binary 就可以了。

本书选择介绍的是目前比较新的版本 Eclipse 3.1，与之相关的插件还在陆续的改进和成熟过程之中，但是这并不影响使用该版本进行相关软件的开发工作，所以本书选择下载的是 Eclipse 3.1.1 版本。

必知必会：绿色软件

Eclipse 是绿色软件，在下载完成后，只需要将对应的压缩包文件 eclipse-SDK-3.1.1-win32.zip 解压缩到指定位置，例如：d:\Eclipse 即可完成安装。安装完成 eclipse 文件夹中的目录结构如图 2-8 所示。

直接找到安装目录下的 eclipse.exe 文件双击运行，即可启动 Eclipse，屏幕将显示如图 2-9 所示的启动画面（月蚀，即 Eclipse）。

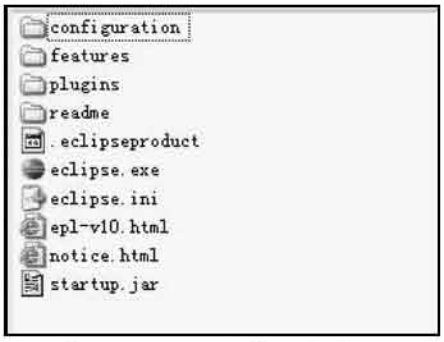


图 2-8 Eclipse 安装后目录结构

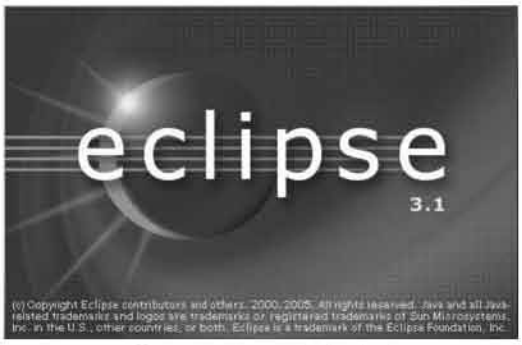


图 2-9 Eclipse 启动画面

在启动画面消失后，屏幕将显示如图 2-10 所示的对话框，提示用户选择工作空间的目录位置，该目录用于存放相关项目文档，后续创建项目对应的目录一般均存放在该工作目录下。

单击图 2-10 所示对话框中的“Browse”按钮，在打开的如图 2-11 所示的对话框中选择

工作区对应的目录，这里设置为 D:\eclipse\workshop。设置完成后单击“确定”按钮。

图 2-10　选择工作区对话框

图 2-11　选择工作区目录对话框

在 Eclipse 启动完成后，将显示如图 2-12 所示的 Eclipse 工作环境。至此 Eclipse 软件的安装及启动过程结束。

图 2-12　Eclipse 欢迎页面

关闭欢迎页面后，即可显示如图 2-13 所示的开发环境界面。

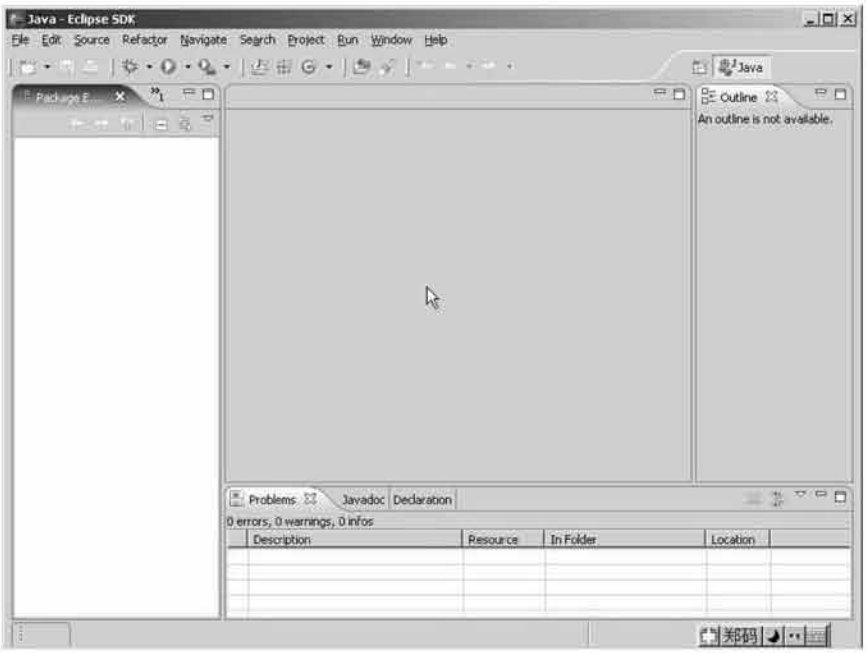


图 2-13 Eclipse 开发环境

必知必会：安装 Eclipse 的多国语言包

Eclipse 为中国用户提供了非常好的支持，通过安装 Eclipse 多国语言包，Eclipse 可以自动实现开发环境的本地化，即自动根据操作系统的语言环境选择对使用语言包进行本地区域化。

读者需要到 www.eclipse.org 下载对应 Eclipse 3.1.1 版本的多国语言包，其名称为 Language Pack。下载完成后，可获得名称为 NLpack1-eclipse-SDK-3.1.1a-win32.zip 的压缩包，下面介绍具体的安装步骤。

① 首先必须关闭 Eclipse。注意在安装 Eclipse 各个插件工具时都需要先关闭 Eclipse;

② 对压缩包进行解压缩。本书将解压缩后获得的文件存放在 Eclipse 安装目录下（如 D:\eclipse）的 Language 子目录中，操作完成后的目录结构如图 2-14 所示;

图 2-14 解压缩多国语言包后的目录结构

③ 在 Eclipse 安装目录下创建子目录 links，并在该目录中新建一个文本文件，名称为 language.start（该文件名可以任意），在文本文件中键入如下所示一行信息；

```
path=d:\\eclipse\\language
```

即指向多国语言包的安装目录，注意路径中反斜杠为双写，如“\\”，此时的目录结构如图 2-15 所示；

图 2-15　安装完多国语言包的目录结构

④ 在安装完语言包后，再重新启动 Eclipse，此时界面将显示为如图 2-16 所示的中文环境。由于在此之前曾经启动过 Eclipse，因此有可能在本地化后，会出现部分单词仍为英文的情况，此时可以先删除 Eclipse 安装目录中的 configuration 子目录下面的 org.eclipse.update 目录，以删除原有记录的英文版信息，然后再重新启动 Eclipse 即可。

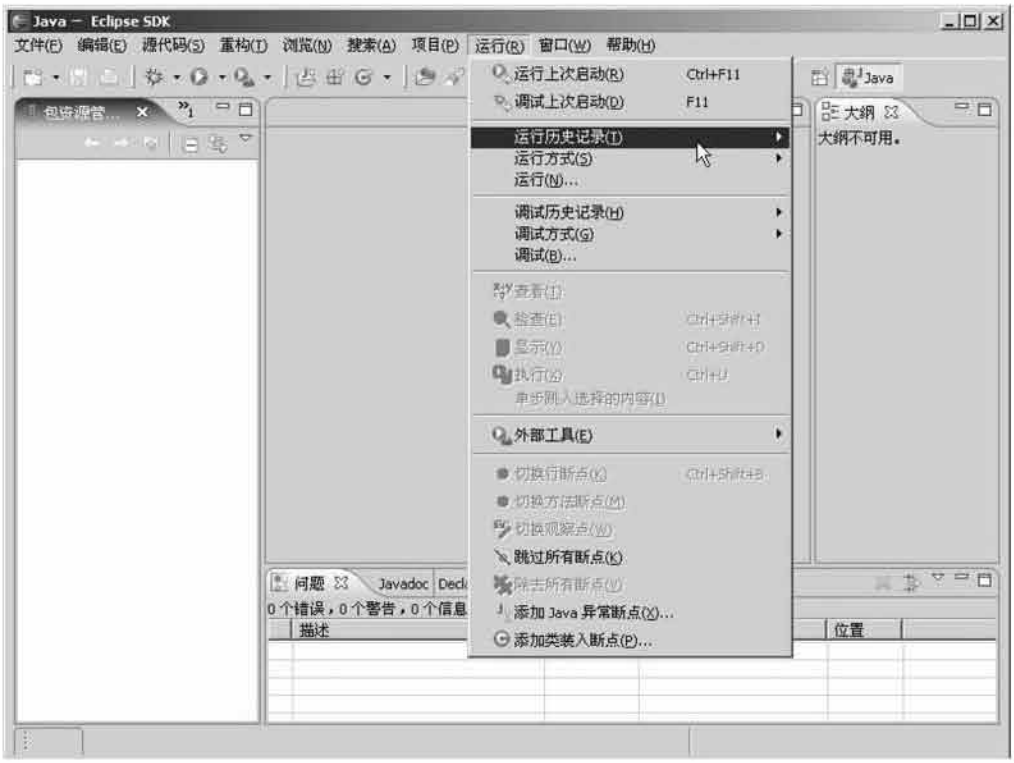


图 2-16　本地化后的 Eclipse 开发环境

如果需要将中文环境再恢复到英文环境，可以删除对应语言包相关的配置文件，即 language.start 即可。为了能够同时删除原有记录的中文版信息，往往同样需要先删除 Eclipse 安装目录中的 configuration 子目录下面的 org.eclipse.update 目录。

上述语言包的安装方式实际上是Eclipse插件中最常见的一种安装方式。具体安装过程如下所示。

① 解压缩对应文件到 Eclipse 安装目录下面的子目录中；

② 在 links 目录下创建对应的连接文件，指定相关的路径；

③ 如果需要强制生效，可以先删除 Eclipse 安装目录中 configuration 子目录下面的 org.eclipse.update 目录，再重新启动。

必知必会：设置 Eclipse 中的文本编辑器的字符集

为了能够在 Eclipse 中采用统一的字符集进行文件的存取，以确保在编写程序源代码的过程中中文信息的正常显示，需要在安装完成 Eclipse 多国语言包后，设置文本编辑器所采用的字符集，由于本书后续案例中所采用的字符集为 UTF-8，因此，下面将 Eclipse 中的文本文件编码类型同样设置为 UTF-8。具体操作步骤如下。

图 2-17 首选项命令

① 选择“窗口”菜单中的“首选项”命令，如图 2-17 所示，打开如图 2-18 所示的对话框，在窗口左侧的树状窗格中选择常规→编辑器选项，在右侧的编辑器窗格中设置文本文件编码为 UTF-8 即可。

图 2-18 设置文本文件编码

必知必会：设置 Eclipse 中与 JDK 相关的环境变量

为了能够在 Eclipse 中进行 Java 应用程序的编写和调试，首先需要检查确认 Eclipse 中与 JDK（Java Development Kits Java 开发工具包）相关的环境变量。选择 Eclipse“窗口”菜单中的“首选项”命令，如图 2-17 所示。

② 在打开的如图 2-19 所示的窗口中选择左边树窗格中的 Java 下面的“已安装的 JRE”选项（即 Java 运行时的环境），然后在右侧窗格中检查对应路径是否正确，单击“确定”按钮确认即可。

图 2-19 “首选项”窗口

2.1.4 安装应用服务器 Tomcat

作为常见的 Web Container，本书选择使用 Tmocat 作为 Web 应用服务器。下面介绍一下应用服务器 Tomcat 的安装过程，本书选择的 Tomcat 版本为 5.0.19。具体操作步骤如下。

① 运行 Apache Tomcat 安装文件 jakarta-tomcat-5.0.19.exe，在弹出的安装向导中根据提示进行选择，单击如图 2-20 所示的“Next”按钮；

② 单击如图 2-21 所示的对话框中的“I Agree”按钮接受软件许可协议；

图 2-20 Apache Tomcat 安装向导（1）

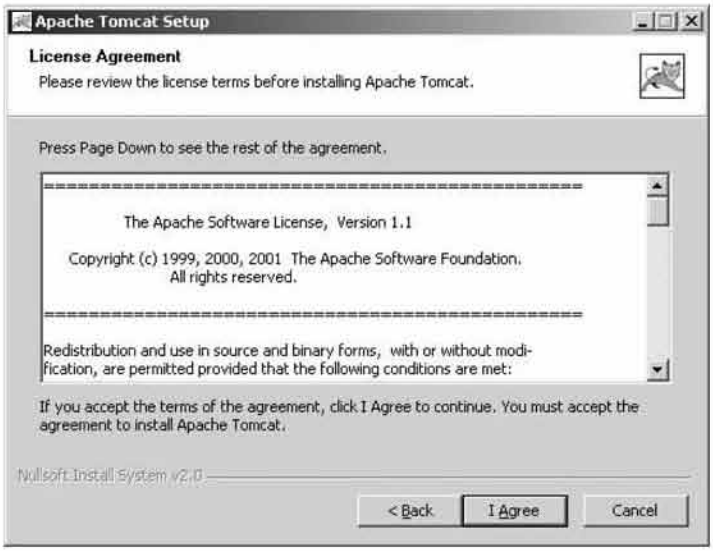


图 2-21 Apache Tomcat 安装向导（2）

③ 在如图 2-22 所示的对话框中选择需要安装的组件，这里建议选择应用服务器的所有相关组件，其中，选择 Service 后可以在计算机启动时自动启动 Tomcat；

④ 接下来在如图 2-23 所示的对话框中选择默认安装的目录位置，单击 Next 按钮。

⑤ 在如图 2-24 所示的对话框中设置 Apache Tomcat 连接端口号，默认为 8080，可以根据需要修改为其他端口号；设置 Apache Tomcat 管理员用户名和密码，这里如果设置，需要记住用户名和密码。完成后单击“Next”按钮。

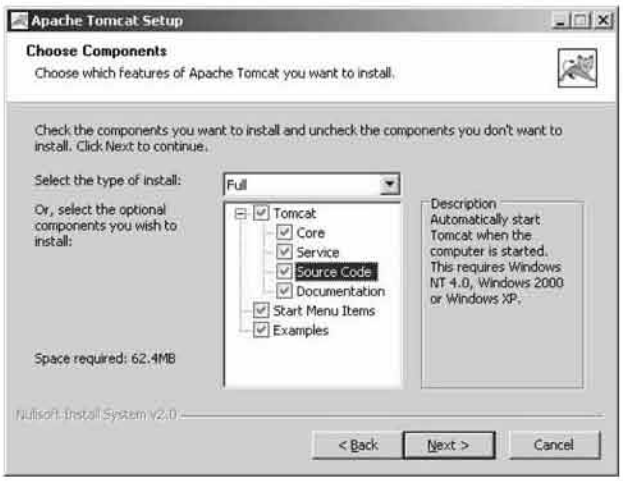


图 2-22　Apache Tomcat 安装向导（3）

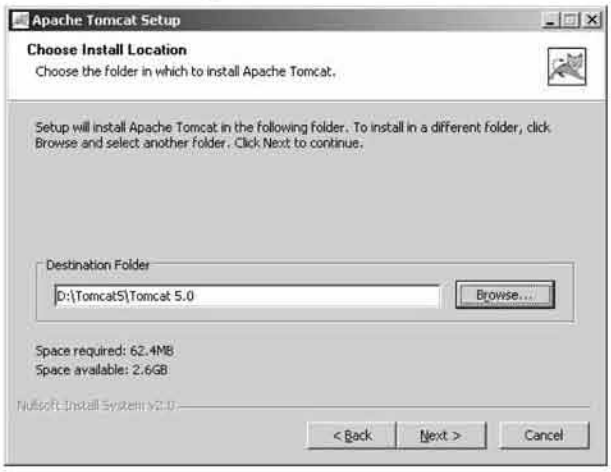


图 2-23　Apache Tomcat 安装向导（4）

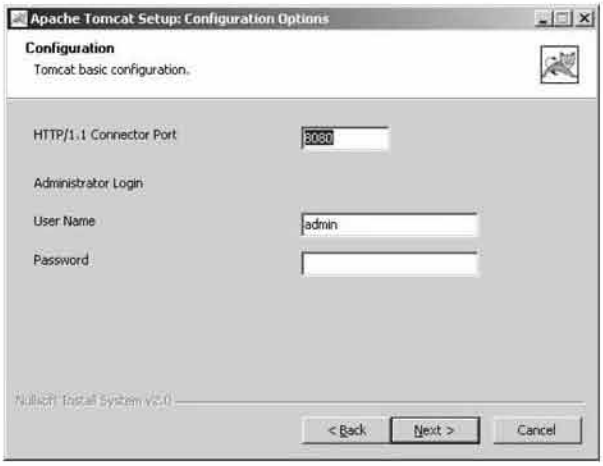


图 2-24　Apache Tomcat 安装向导（5）

⑥ 在如图 2-25 所示的对话框中选择 JDK 的安装目录，注意需要跟前面安装 JDK 时的目录一致。

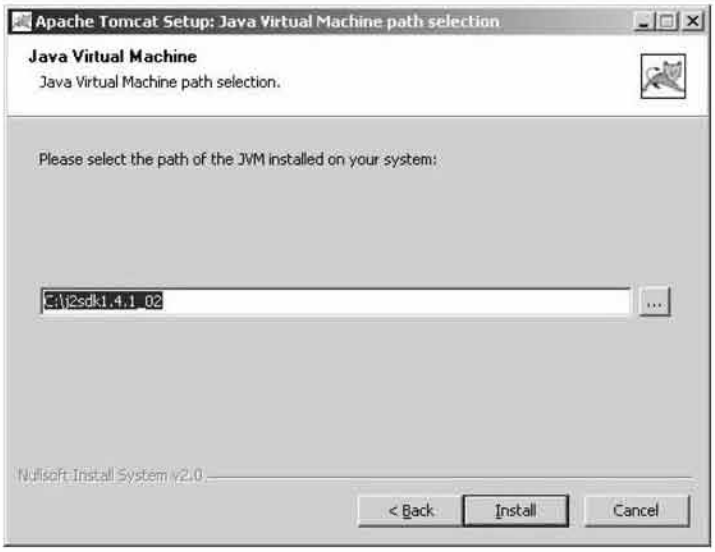


图 2-25 Apache Tomcat 安装向导（6）

⑦ 屏幕上出现如图 2-26 所示的对话框时，单击“Finish”按钮即可结束安装，此时 Tomcat 可以正常启动，在屏幕右下角可以看到 Tomcat 服务器的对应图标。

图 2-26 Apache Tomcat 安装向导（7）

此时，可以打开 IE 浏览器，在地址栏中输入 http://127.0.0.1:8080 或者 http://locahost:8080，

如果显示出如图 2-27 所示的页面效果，则证明 Tomcat 已经正常运行工作。

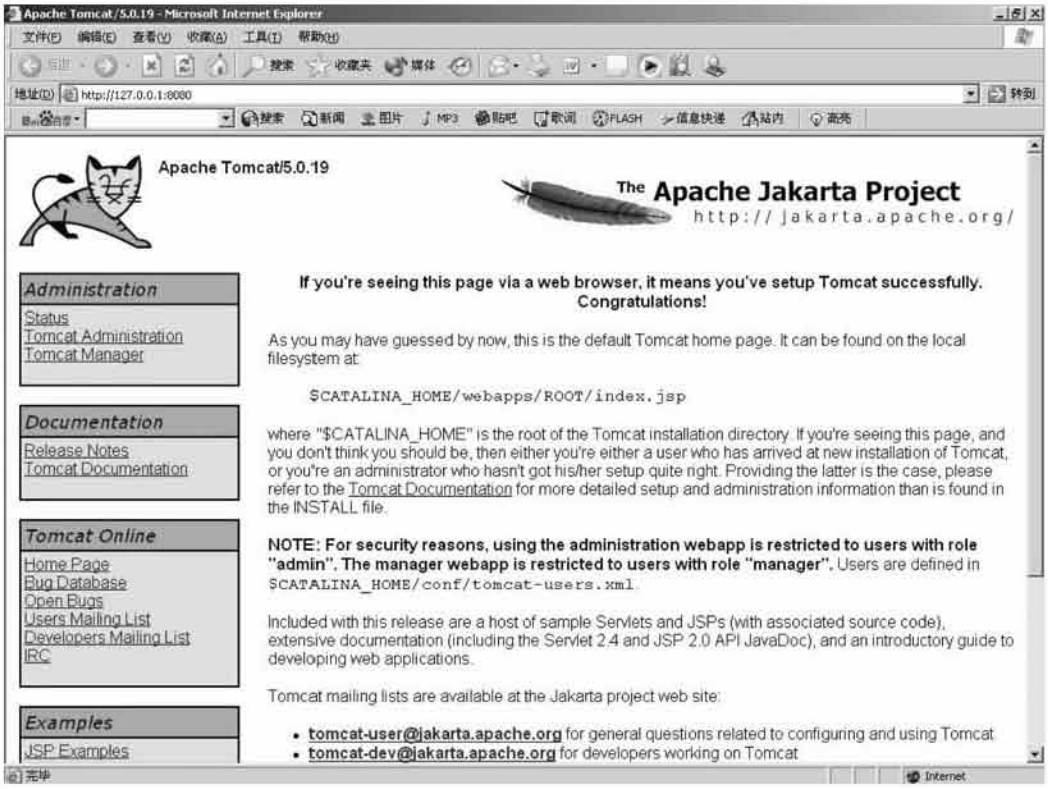


图 2-27　Apache Tomcat 默认首页

必知必会：Tomcat 端口冲突的解决

需要注意的是：在安装 Tomcat 时，如果系统中已经有其他的应用程序占用了 8080 端口，则启动 Tomcat 时会失败，现象就是 Tomcat 的启动窗口一闪而过。

如果遇到这个问题，可以修改 Tomcat 使用的端口号，打开 Tomcat 安装目录下 conf 子目录中的配置文件 server.xml，找到以下这段配置文字：

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="8080" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="8443"
               acceptCount="10" debug="0" connectionTimeout="60000"/>
```

将其中的 port="8080" 对应的端口号 8080 改为其他一个未被使用的端口号，比如 8888，然后重新启动 Tomcat 就可以了，这样以后在访问 Tomcat 时，就应该在浏览器地址栏中输入以下地址，同样会出现图 2-27 所示的页面。

```
http://localhost:8888
```

至此 Tomcat 应用服务器安装完毕。

2.1.5 安装 MyEclipse

MyEclipse 是在 Eclipse 集成开发环境中提供 Web 应用程序开发的插件，在目前软件开发的 Java 领域中，MyEclipse 是一个使用非常广泛的工具，本书选择安装的是 MyEclipse 4.1，在该版本中，不仅提供了更好的对 Java Web 项目开发的支持功能，更重要的是在该版本中增加了对 JavaScript 脚本的调试功能，实际上就是实现了 MyEclipse 对 Ajax 的支持功能。需要注意的是 MyEclipse 并不是一个完全免费的产品，读者可以选择购买对应的产品。

下面介绍一下 MyEclipse 插件的安装过程。

双击运行名称为 EnterpriseWorkbenchInstaller_4.1GA_E3.1.exe 的安装文件，启动 MyEclipse 的安装程序，屏幕效果如图 2-28 所示。

图 2-28 启动 MyEclipse 安装程序

在解压缩工作完成之后，屏幕上将会显示如图 2-29 所示的安装程序对话框。单击“Next”按钮即可。

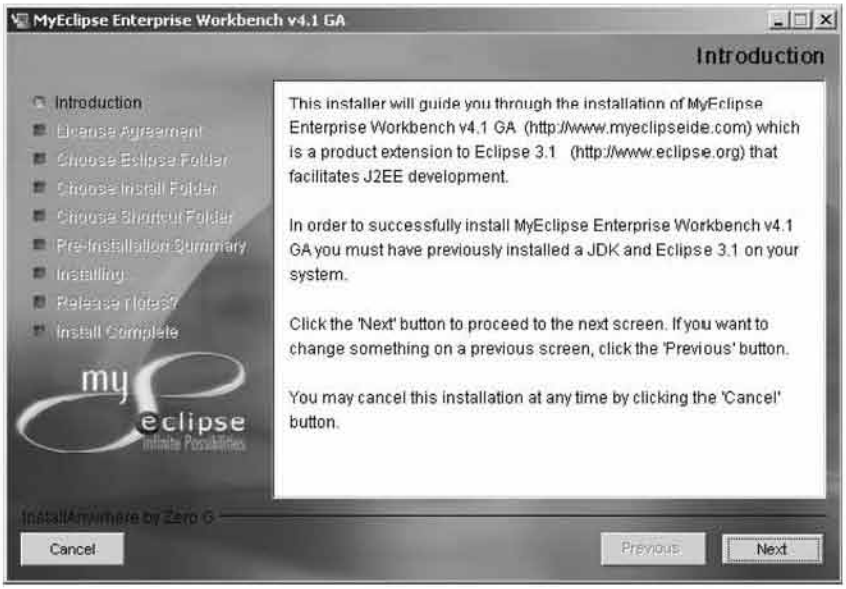


图 2-29 MyEclipse 安装程序介绍对话框

在如图 2-30 所示的对话框中，选择接受许可协议，单击“Next”按钮。

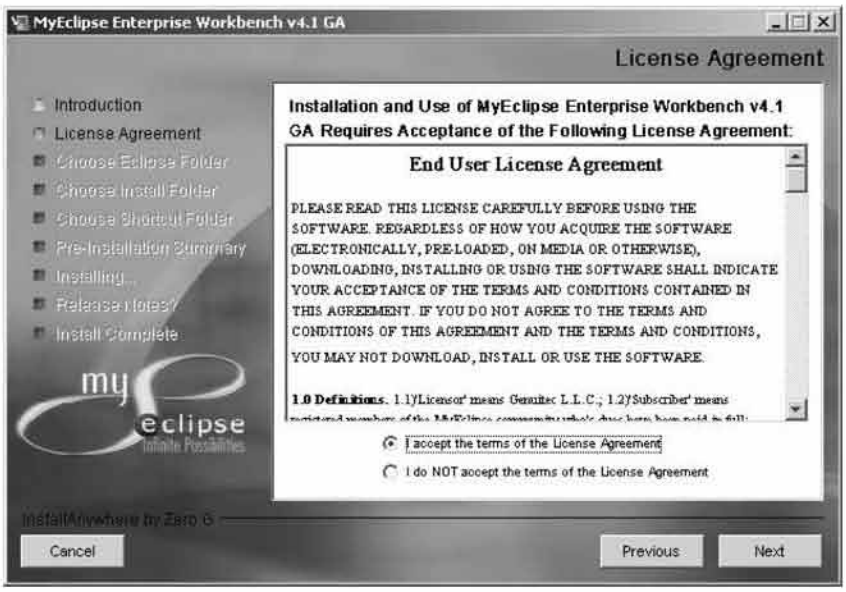


图 2-30　选择接受 MyEclipse 安装协议

由于 MyEclipse 是作为 Eclipse 的插件进行安装的，所以需要在如图 2-31 所示的对话框中选择 Eclipse 的安装目录。单击“Choose”按钮按照提示选择对应的 Eclipse 安装目录即可。

图 2-31　选择 Eclipse 安装目录

设置完成之后，单击“Next”按钮，在如图 2-32 所示的对话框中选择 MyEclipse 的安装目录即可。

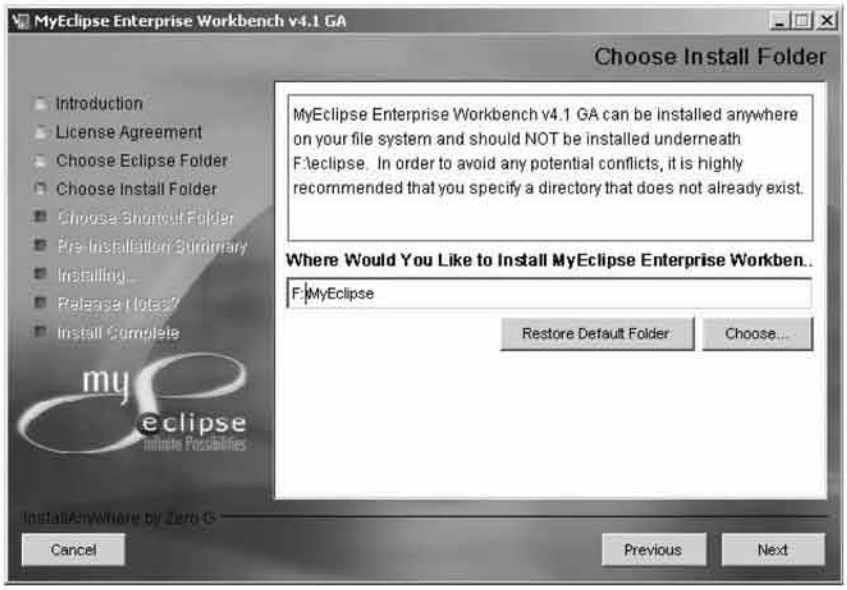


图 2-32 选择 MyEclipse 安装目录

继续单击 Next 按钮，屏幕将显示如图 2-33～2-35 所示的对话框，读者可以了解到有关 MyEclipse 4.1 的一些新的特性。

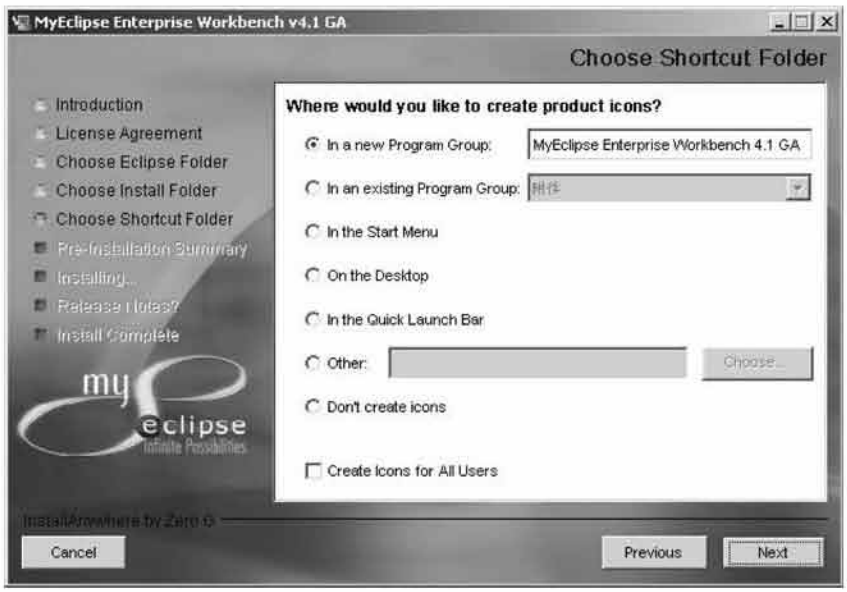


图 2-33 MyEclipse 安装提示对话框（1）

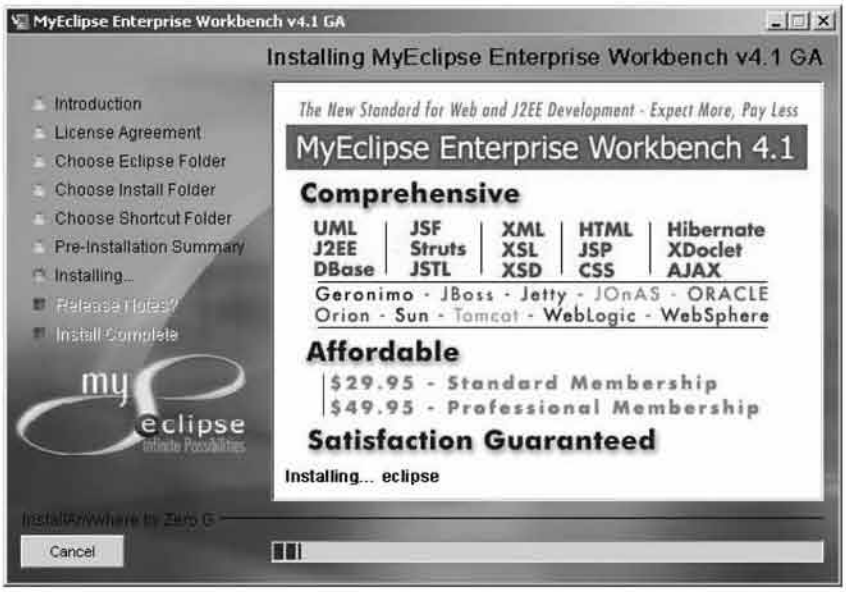


图 2-34　MyEclipse 安装提示对话框（2）

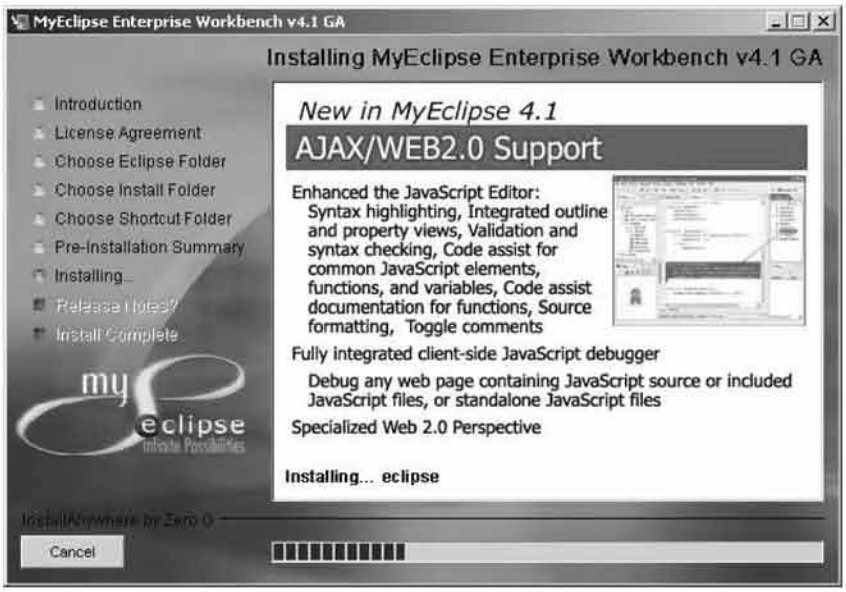


图 2-35　MyEclipse 安装提示对话框（3）

直到屏幕显示如图 2-36 所示的对话框，单击“Done”按钮即可完成 MyEclipse 的安装。

接下来需要在 Eclipse 中设置与 MyEclipse 相关的一些选项，以实现 MyEclipse 与 Web 服务器 Tomcat 之间的关联。

必知必会：将 MyEclipse 与对应 Web 服务器关联在一起

重新启动 Eclipse，此时可以看到如图 2-37 所示的开发环境中增加了与 MyEclipse 有关的

菜单，并且在工具栏中新增了3个与MyEclipse相关的按钮。

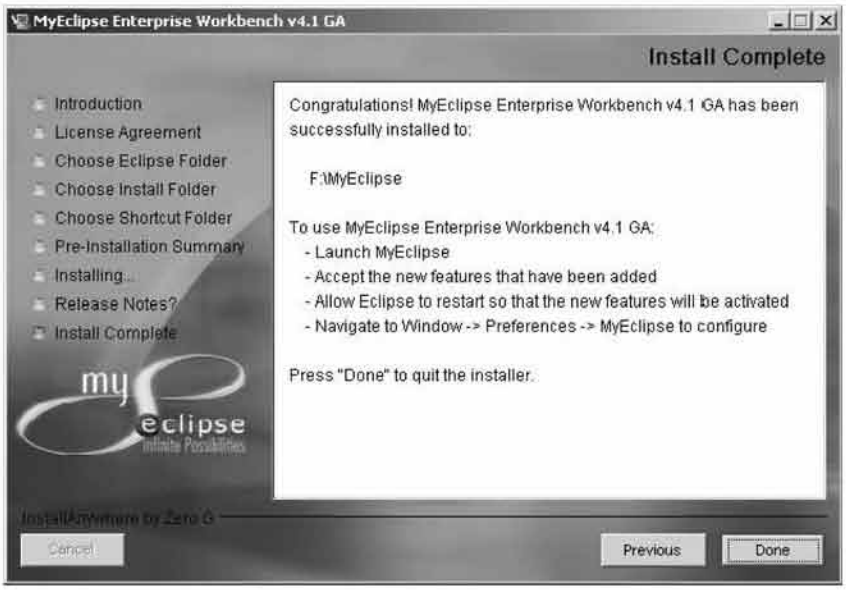


图 2-36 MyEclipse安装提示对话框（4）

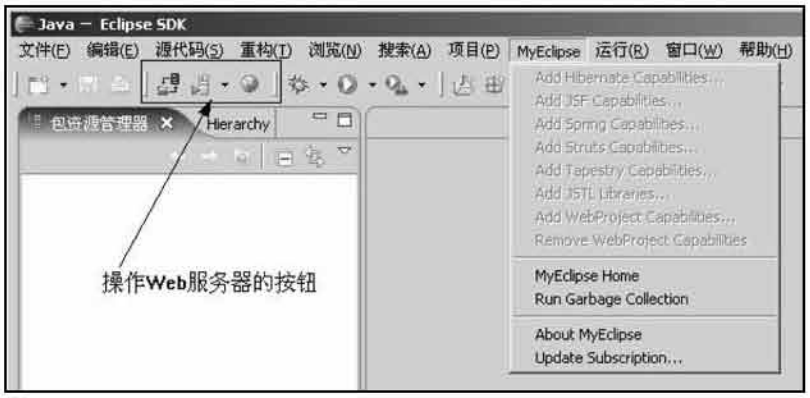


图 2-37 在Eclipse中安装MyEclipse之后的效果

从左到右这3个按钮的功能分别为：部署MyEclipse J2EE的项目到服务器、运行/停止/重启MyEclipse应用服务器以及打开MyEclipse Web浏览器以预览程序执行的效果。

下面通过选项的设置来实现MyEclipse与Web服务器Tomcat之间的关联。选择“窗口”菜单中的“首选项”命令，在如图2-38所示的对话框中进行相关的设置。首先在左侧窗格中选择“MyEclipse”下面的“Application Servers”，然后选择所使用的应用服务器类型和对应版本，这里选择“Tomcat 5”，在右侧的窗格中选择“Enable”选项，单击“浏览”按钮指定本机中安装Tomcat的主目录。

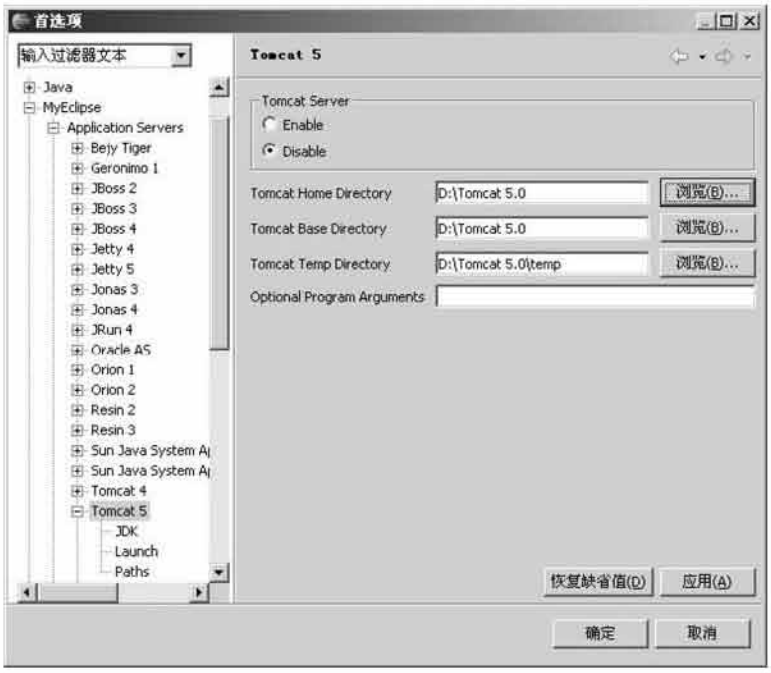


图 2-38　设置与 MyEclipse 相关的首选项

然后选择左侧窗格中“Tomcat 5”下面的“JDK”，在如图 2-39 所示的右侧窗格中设置 Tomcat JDK name，单击“Add”按钮添加对应 Java 运行环境的主目录，单击“确定”按钮完成对应的设置即可。

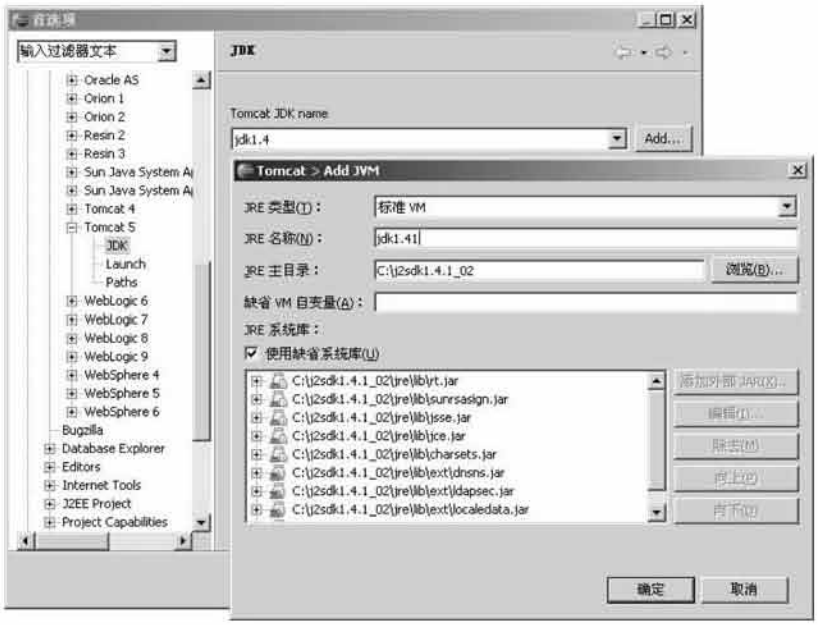


图 2-39　设置与 MyEclipse 相关的首选项

单击如图 2-40 所示的 Start 命令即可启动 Tomcat。

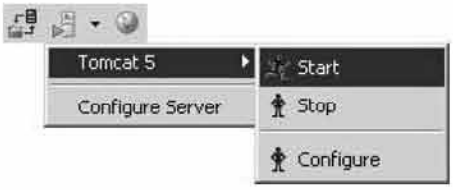


图 2-40 启动 Tomcat

此时，在 Eclipse 的控制台提示窗口中将显示对应 Tomcat 应用服务器的启动信息，如图 2-41 所示。

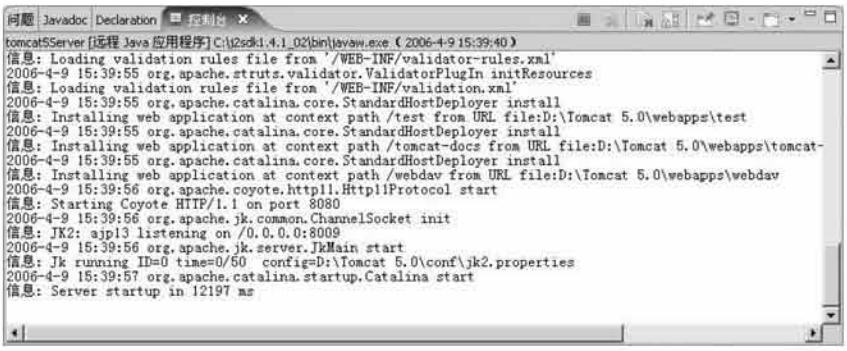


图 2-41 Tomcat 启动信息

至此，与 MyEclipse 相关的设置结束，除了数据库管理系统之外的基本开发环境准备完毕，有关数据库的安装将在本章后续的内容中结合案例进行详细介绍。

2.2 开发商务网站身份验证模块

为了让读者能够尽快体会到 Ajax 技术的魅力，本节将通过一个常见的身份验证模块介绍传统实现方式和 Ajax 实现方式之间的不同。

身份验证应用模块主要用于对商务网站用户的身份进行确认检查，其主要目的是为了确保数据信息的安全。例如不同的用户所看到的与之相关的信息是不一样的，彼此之间的数据也是受保护的。

本节案例中只实现了身份验证程序所要求的最基本功能，同时为了能够让读者体会到 Ajax 实现方式与传统实现方式之间的区别，本例中并没有按照惯例在进行验证之后跳转到新的页面，而是直接在原来页面的基础之上显示提示信息窗口，以便于让读者直观地看到 Ajax 未进行页面刷新的效果。

用户在登录时必须进行身份验证，系统只允许已经注册，拥有合法用户名及密码的用户

进入，登录成功将显示登录成功的信息。如果登录失败将显示登录失败的错误提示信息。

2.2.1 传统实现方式

首先我们使用传统的同步交互方式实现对应的身份验证效果。依据上述的应用需求，具体的用户界面（UI）设计如下。

案例 2-1 传统方式实现 Java Web 页面的登录验证

图 2-42 所示页面为提示用户输入用户名及密码的登录页面，在该页面中提供了填写用户名及密码的文本框、登录信息提交按钮。

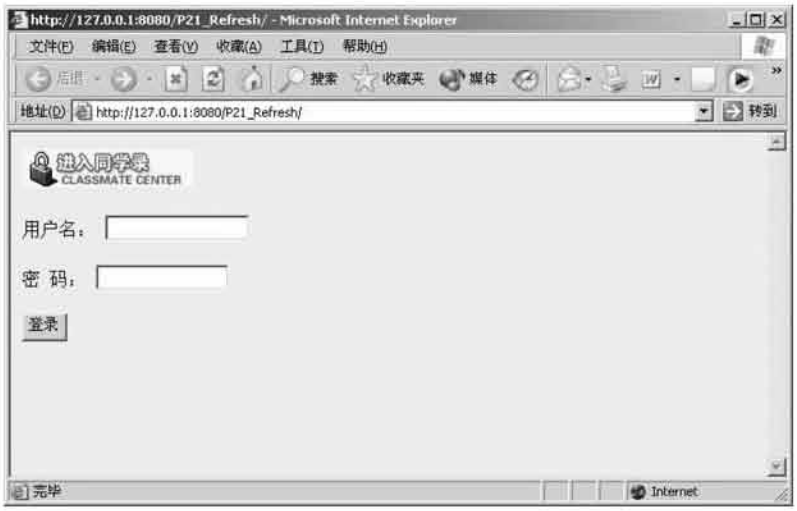


图 2-42 登录页面

图 2-43 所示效果为登录成功之后的欢迎信息，可以看到提示信息显示的窗口后面页面是空白的，这是因为在进行页面刷新，读者也可以直观地看到页面状态栏中的页面刷新显示的进度条。当提示信息窗口未关闭之前，刷新过程是不继续向下进行的。在用户单击“确定”按钮之后，页面将恢复图 2-42 所显示的效果。原来填写的信息将因为进行了刷新而消失。

图 2-44 所示页面为登录失败后的提示信息窗口的效果。在该页面效果中同样可以看到页面在进行刷新的效果。

在本例中并没有实现按照数据库中注册的用户信息进行身份验证的功能。在本章后续学习了数据库的相关操作后，将对本例进行完善，以最终实现用户身份验证的功能。

下面结合 Eclipse 开发环境介绍一下如何实现能够满足上述需求的 Web 应用程序。首先在 Eclipse 中新建一个 Web 项目。选择“文件”菜单下“新建”子菜单中的“项目”命令，在如图 2-45 所示的对话框中选择 MyEclipse 下面的 J2EE Projects 下面的 Web Project。单击“下一步”按钮。

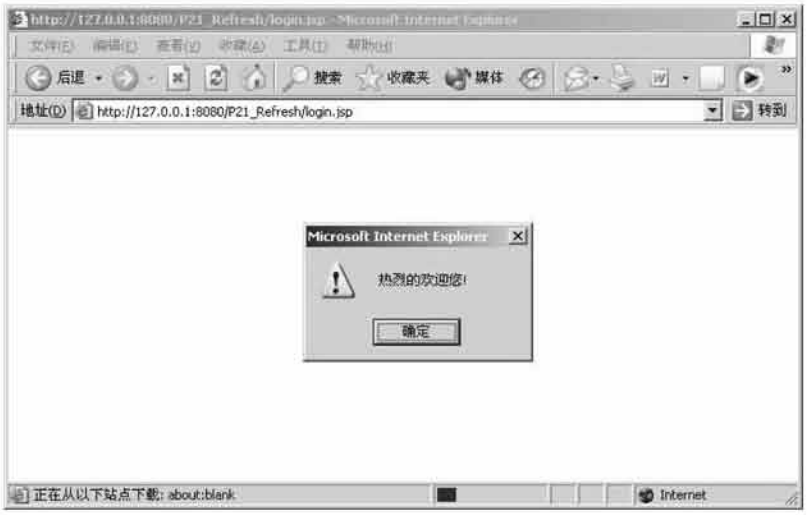


图 2-43 欢迎提示信息

图 2-44 登录失败信息

在如图 2-46 所示的对话框中输入项目的名称“P21_Refresh”，并设置与该项目相关的选项，这里保持默认值即可，单击“完成”按钮结束项目的创建。

此时，屏幕将显示如图 2-47 所示的视图切换提示窗口。单击“是”即可进入 J2EE 开发视图环境。

所创建项目的目录结构如图 2-48 所示。可以看到与 Java Web 项目相关的库文件，以及 Web 应用的配置文件都已经自动生成。

本例采用的是 Web 应用开发中经典的 MVC 设计模式。

图 2-45　新建项目对话框

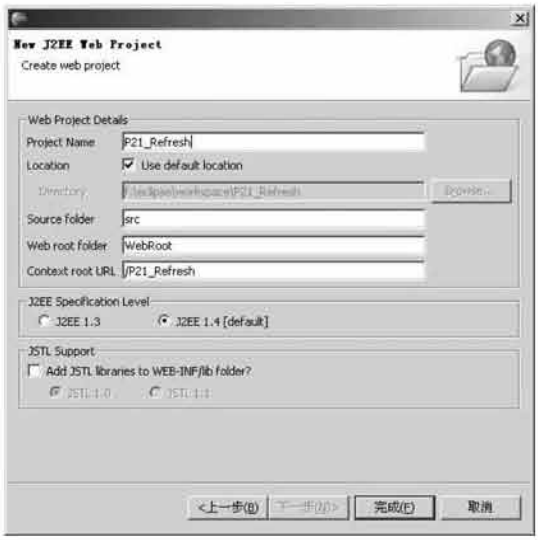


图 2-46　设置项目选项

图 2-47　视图切换提示窗口

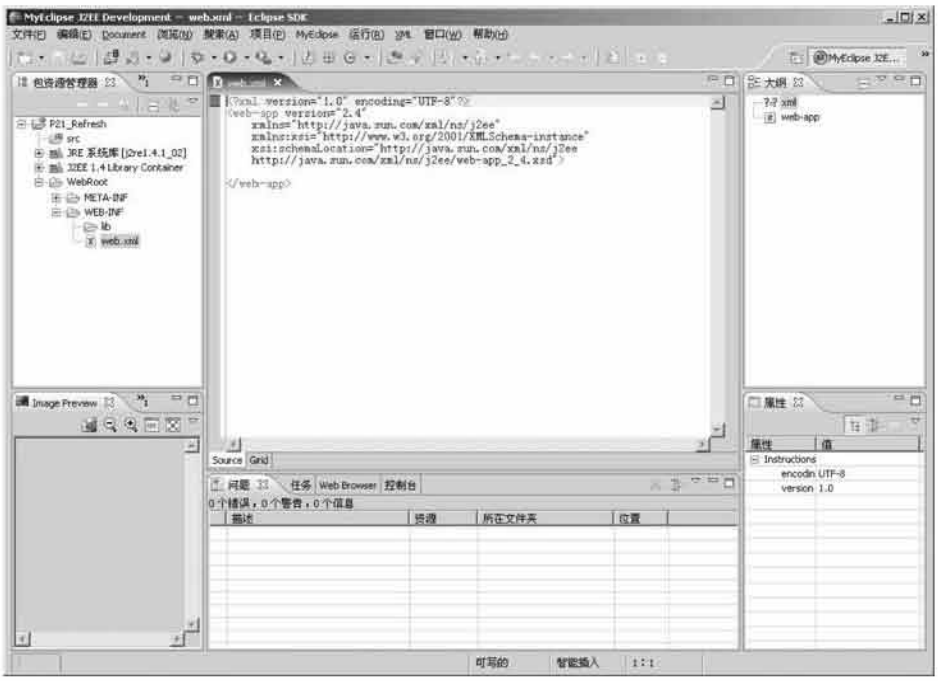


图 2-48　J2EE Development 视图效果

必知必会：MVC 设计模式

MVC（Model–View–Controller）模式，即模型-视图-控制器模式，其核心思想是将整个程序代码分成相对独立而又能协同工作的 3 个组成部分，具体功能如下所示。

- 模型（Model）：业务逻辑层。实现具体的业务逻辑、状态管理的功能；
- 视图（View）：表示层。即与用户实现交互的界面，通常实现数据的输入和输出功能；
- 控制器（Controller）：控制层。起到控制整个业务流程（Flow Control）的作用，实现 View 和 Model 部分的协同工作。

MVC 设计模式可以针对需要为同样的数据提供多个不同视图的应用程序，例如，公司产品数据库中同样的产品信息数据，但需要根据用户的不同需求在页面中显示其所需的不同产品信息。

MVC 设计模式中，事件一般是指客户端 Web 浏览器提交的各种不同请求，这些请求由控制器进行处理，控制器根据事件的类型来改变模型或各个视图，视图也可以接收模型发出的数据更新的通知，依据数据更新的结果调整视图效果，呈现在用户面前。而模型也可以通过视图所获得的用户提交的数据进行具体业务逻辑的处理。

实际上，这样的工作方式在现实生活的各个机构中随处可见，例如，去医院某科室（视图）就诊时，分诊台（控制器）将会根据病人的不同病情安排不同的专家（模型）进行诊治，而专家将会根据病人的不同病情开具不同的处方、化验单等（视图）。

显然这样的运行机制可以起到分工明确、职责清晰、各尽所长的效果。而在软件开发的过程中，这样的方式无疑可以有效地区分不同的开发者，尽可能减少彼此间的互相影响。充分发挥每个开发者的特长。这在开发大型复杂的 Web 项目时体现得尤为突出。

必知必会：JSP 中 Model 2 的工作原理

在 JSP 的 Model 2 中采用 Servlet 作为控制器（Controller），负责接收客户端 Web 浏览器发送来的所有请求，并依据处理的不同结果，转发到对应的 JSP 页面（Viewer）实现在浏览器客户端的显示。

必知必会：Servlet 定义

所谓 Servlet 是指运行在服务器端的 Java 小程序。用于响应客户机的请求。在默认情况下，Servlet 采用一种无状态的请求-响应处理方式。Servlet 代码的主要作用是为了增强 Java 服务器端的功能，它运行在服务器端，用于接收并且处理浏览器客户端发出的请求，该请求是通过配置文件 web.xml 中的相关配置进行转发的。也就是说 Servlet 是一个标准的 Java 类，它符合 Java 类的一般规则。和一般 Java 类不同之处只在于 Servlet 可以处理 HTTP 请求。

通常在 Servlet 中只包含了控制逻辑和一些简单的处理逻辑，更加复杂的业务处理逻辑则借助特定的 JavaBean 具体实现，例如：利用 JavaBean 实现与数据库的连接，对数据库中的数据信息进行维护修改等。

其具体实现方式如图 2-49 所示。

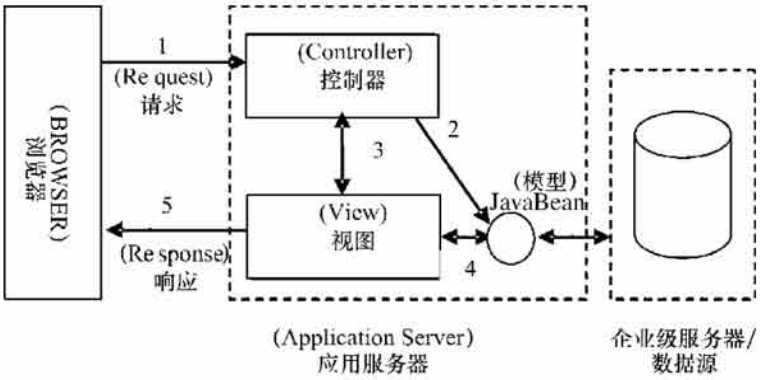


图 2-49 JSP 设计模式 2

由于引入了 MVC 设计模式，Model 2 实际上实现了基于组件的开发，在整个软件开发过程中实现了具体清晰的逻辑划分，能够有效地区分不同的角色，这就更适合于大规模系统的开发和管理。

例如，在规模较大的饭店中，服务员不会直接跟后厨打交道，一般都会在服务员和厨师之间设置统一的资源调配中心，统一协调各个服务员提交的菜单，安排后厨有计划地完成各个菜单。比如：凉菜菜单会转交给冷菜间，再进一步转交给冷素间或者冷荤间；热菜菜单则会按照一定的顺序转交给热炒间，再分派各个厨师。当菜制作完成时，会再次由资源调配中心调度服务员送到对应客人面前。同时在许多大的饭店中，服务员一般也进行了职责的划分，例如，每个人负责几桌等。

这样的运作方式显然能够实现管理的高效率，并且有利于饭店规模的进一步扩大，即能够确保软件系统的强扩展性。当然与此同时，也增加了应用开发的复杂程度，就如同现代化的大酒店要建立一套成熟的酒店管理体制需要一段时间一样，原来建立一个简单的 JSP 页面就能实现的应用，现在被分解成多个协同工作的成分，开发者需要花些时间才能真正掌握其设计和实现过程。

在本例中，采用 JSP 页面承担表示层的功能，Servlet 程序充当控制层，由于对应的业务逻辑比较简单，所以没有单独编写 JavaBean，后续在增加了与数据库相关的操作之后，将通过 JavaBean 来封装对应的业务逻辑。

首先创建一个 JSP 页面，用于显示身份验证的表单，以供用户填写待验证的信息。该页面对应的源代码及相关解释如下。

在用户填写了对应信息后，单击“登录”按钮后，将向服务器端提交登录 login 请求，以便于控制层进行处理。

该程序源代码如下：

源文件：login.jsp

- 设置显示用的字符集

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
```

- 依据不同的验证效果进行不同提示信息的显示

```
<%
     String result = (String)session.getAttribute("result");
          if(result != null && result != ""){
          if(result.equals("ok")){
               out.println("<script>window.alert('热烈的欢迎您！')</script>");
          }else{
               out.println("<script>window.alert('对不起，登录失败！')</script>");
          }
          session.invalidate();
     }
%>
<body vLink="#006666" link="#003366" bgColor="#E0F0F8">
<img height="33" src="enter.gif" width="148">
```

- 填写用户待验证信息的表单

```
<form action="login" method="post">
用户名： <input size="15" name="uname"><p>
密  码： <input type="password" size="15" name="psw"><p>
<input type="submit" value="登录">
</form>
……
```

接下来需要创建对应控制层相关的控制类，即作为控制中心的 Servlet 程序。在写 Servlet 程序时，一定要注意的是：作为一个 Servlet，它必须是 HttpServlet 的子类，在使用时需要重写对应的方法，一般可以重写下面的方法。

- doGet()方法：响应客户端的 Get 请求；
- doPost()：响应客户端的 Post 请求。

也可以直接重写 service 方法以声明在接收到对应请求后具体的处理过程。

本案例中定义了一个 Servlet，用于处理浏览器客户端提交的 login 请求，在该程序中将获取浏览器端提交的信息，然后进行身份验证。

具体程序代码如下：

源文件：LoginAction.java

```
package classmate;
import java.io.IOException;
……
public class LoginAction extends HttpServlet {
     protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
//获取浏览器客户端提交的用户待验证信息
      String uname = req.getParameter("uname");
      String psw = req.getParameter("psw");
      String target = "";
//准备 session 对象以进行验证结果的传递
      HttpSession session = req.getSession();
//依据验证结果跳转到不同的页面
      if(uname.equals("jenny") && psw.equals("hi")){
            session.setAttribute("result","ok");
            target = "/P21_Refresh/login.jsp";
      }else{
            session.setAttribute("result","wrong");
            target = "/P21_Refresh/login.jsp";
      }
      resp.sendRedirect(target);
   }
}
```

对应 web.xml 配置文件的源代码如下所示，在该配置文件中可以看到，当浏览器客户端发出不同的请求时，依据配置文件中的配置，将会将请求转发给指定的 Servlet 对象进行处理。

```
<?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">
<!-设置对应 Servlet 对应的类型-->
     <servlet>
          <servlet-name>ms1</servlet-name>
          <servlet-class>classmate.LoginAction</servlet-class>
     </servlet>
<!-设置请求对应相应的 Servlet 名称-->
     <servlet-mapping>
          <servlet-name>ms1</servlet-name>
          <url-pattern>/login</url-pattern>
     </servlet-mapping>
<!-设置默认欢迎页面，这里为请求名称 -->
  <welcome-file-list>
     <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

在该项目相关文件建立完成之后，对应项目在资源管理器中显示的层次结构如图 2-50 所示。

图 2-50　层次结构图

单击工具栏中的部署按钮将对应项目部署到 Web 服务器中，单击启动 Tomcat 按钮启动服务器，单击打开浏览器按钮，在浏览器地址栏中录入地址 http://localhost:8080/ P21_Refresh 查看页面效果。

2.2.2 Ajax 的实现方式

本节将采用 Ajax 的方式实现异步交互的效果，本例中页面的初始效果与前面的案例相同，不同的是由于采用异步交互方式，所以在显示验证结果信息的过程中没有页面刷新的效果。

案例 2-2 采用 Ajax 技术实现 Java Web 页面的登录验证

本例中显示验证信息的页面如图 2-51、图 2-52 所示。

图 2-51 所示效果为登录成功之后的欢迎信息，可以看到提示信息显示的窗口后面的页面并不是空白的，读者在页面状态栏中也看不到页面刷新显示的进度条。

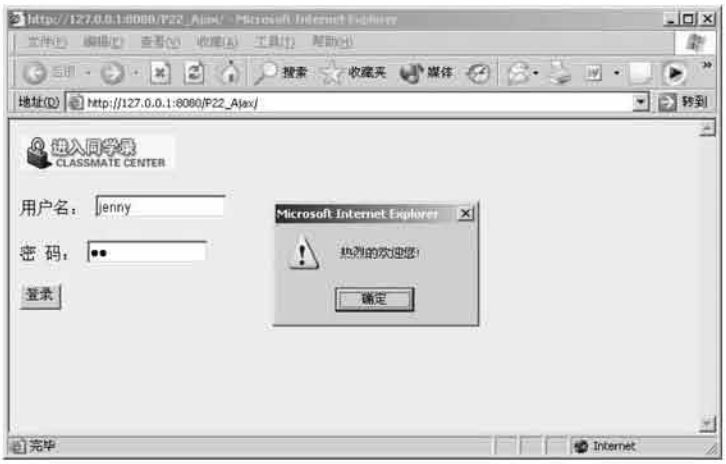


图 2-51 欢迎提示信息

图 2-52 所示页面为登录失败后的提示信息窗口的效果。在该页面效果中同样也看不到页面在进行刷新的效果。

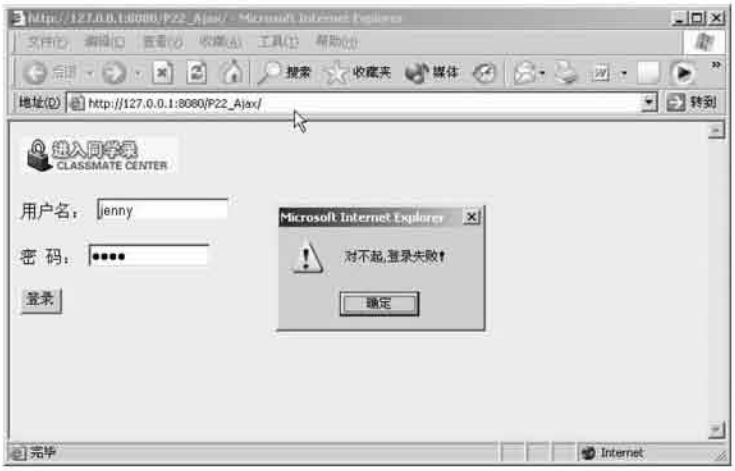


图 2-52 登录失败信息

在本例中并没有实现按照数据库中注册的用户信息进行身份验证的功能。在本章后续学习了数据库的相关操作后，将对本例进行完善，以最终实现用户身份验证的功能。

首先在 Eclipse 中新建一个 Web 项目，项目名称为“P22_Ajax”。该项目的目录结构与上面的案例基本相同，其中配置文件“web.xml”也没有进行修改，不同的是视图页面中增加了与 Ajax 技术相关的程序代码，Servlet 中传递回来的数据信息为 XML 格式的数据。

本例中先简单介绍一下程序代码及对应的含义，有关 Ajax 技术的核心问题将在后续章节中详细阐述。

首先依然要创建一个 JSP 页面，用于显示身份验证的表单，以供用户填写待验证的信息。同时在该页面中使用 Ajax 技术用来提交请求，并接受服务器端请求处理之后的结果数据，以进行显示。

该程序源代码如下：

源文件：login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq = false;
     //创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
          if(window.XMLHttpRequest) { //Mozilla 浏览器
               XMLHttpReq = new XMLHttpRequest();
          }
          else if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               } catch (e) {
                    try {
                         XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                    } catch (e) {}
               }
          }
     }
     //发送请求函数
     function sendRequest(url) {
          createXMLHttpRequest();
          XMLHttpReq.open("GET", url, true);
          XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
          XMLHttpReq.send(null);  // 发送请求
     }
     // 处理返回信息函数
     function processResponse() {
     if(XMLHttpReq.readyState == 4) { // 判断对象状态
```

```
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].first
Child.data;
                window.alert(res);
            }else { //页面不正常
                window.alert("您所请求的页面有异常。");
            }
        }
    }
    // 身份验证函数，验证通过提交对应的请求
    function userCheck() {
        var uname = document.myform.uname.value;
        var psw = document.myform.psw.value;
        if(uname=="") {
            window.alert("用户名不能为空。");
            document.myform.uname.focus();
            return false;
        }
        else {
            sendRequest('login?uname='+ uname + '&psw=' + psw);
        }
}

</script>
<body vLink="#006666" link="#003366" bgColor="#E0F0F8">
<img height="33" src="enter.gif" width="148">
<form action="" method="post" name="myform">
用户名： <input size="15" name="uname"><p>
密  码： <input type="password" size="15" name="psw"><p>
<input type="button" value="登录" onclick="userCheck()" >
</form>
……
```

接下来需要创建对应的控制层相关的控制类，作为控制中心的 Servlet 程序。注意本例中返回数据的格式为 XML 格式。

具体程序代码如下：

源文件：LoginAction.java

```
package classmate;

import java.io.IOException;
……
public class LoginAction extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
    }
    /*
     *  处理<GET> 请求方法
     */
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //接收浏览器端提交的信息
         String uname = request.getParameter("uname");
         String psw = request.getParameter("psw");
         //设置输出信息的格式及字符集
       response.setContentType("text/xml; charset=UTF-8");
       response.setHeader("Cache-Control", "no-cache");
       //创建输出流对象
       PrintWriter out = response.getWriter();
       //依据验证结果输出不同的数据信息
       out.println("<response>");
        if(uname.equals("jenny") && psw.equals("hi")){
              out.println("<res>" + "热烈的欢迎您!" + "</res>");
        }else{
              out.println("<res>" + "对不起,登录失败! " + "</res>");
        }
        out.println("</response>");
        out.close();
    }
    /*
     *  处理<POST> 请求方法
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        doGet(request, response);
    }
}
```

必知必会：解决中文信息显示的乱码问题

注意在本例中为了解决中文信息显示的乱码问题所采取的几个措施，首先考虑到程序的国际化问题，本书中采用的编码统一为“UTF-8”，其次开发人员需要从以下几个方面引起关注。

1. Eclipse 中文本文件编辑器所采用的字符集设置为 UTF-8，如图 2-53 所示。
2. MyEclipse 中相关类型文件的字符集设置为 UTF-8，如图 2-54 所示。
3. JSP 页面用于显示的字符集设置为 UTF-8。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
```

4. Servlet 中接受请求和发送响应时对应的字符集设置为 UTF-8。

```
request.setCharacterEncoding("UTF-8");
response.setContentType("text/xml; charset=UTF-8");
```

图 2-53 Eclipse 中文本文件编码设置

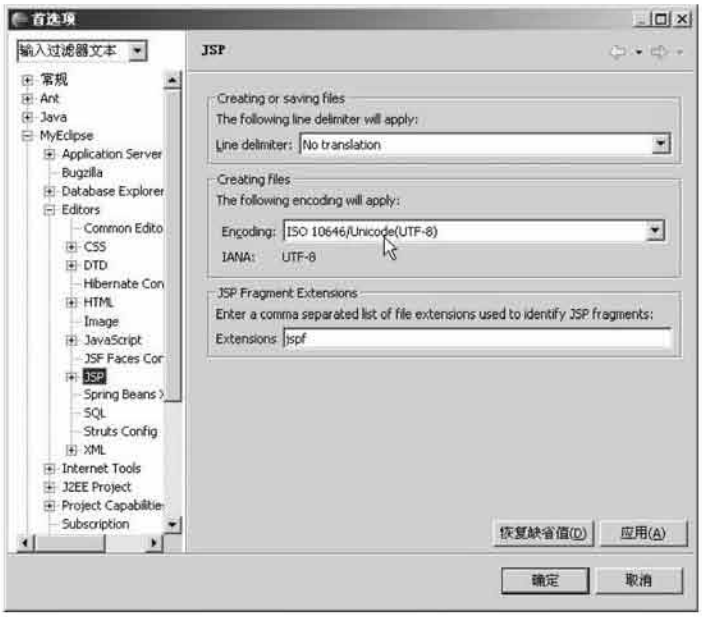


图 2-54 MyEclipse 中相关类型文件的字符集设置

在上面的代码中可以看到，无论是视图层的页面，还是控制层中的 Servlet 程序都发生了改变，这主要是因为本例中采用的是 Ajax 技术提供的一个核心对象 XMLHttpRequest 来负责向服务器端提交请求以及接受从服务器发回的响应数据。读者可以先通过本例中的注

释对该对象所实现的功能有一个初步的理解，有关该对象属性及方法的详细内容将在后续进行介绍。

在上面的例子中，并没有针对数据库中的数据进行验证，而是按照固定的信息在进行验证，下面为了能够实现依据数据库中已注册用户的信息进行身份验证的功能，我们对上面的案例进行一定的改进。

首先介绍一下有关数据库管理系统的安装和准备。

作为一个支持并发连接、多用户、多线程的关系数据库管理系统，MySQL 在 Java 应用的开发中得到了越来越广泛的应用。MySQL 数据库管理系统虽然不像 Eclipse、Tomcat、Struts 一样是完全开源的产品，但是从某种意义上说，MySQL 可以方便地从网上进行下载，可以自由使用，再加上它不像 Oracle 数据库管理系统那样复杂、庞大，因此几乎被业界认为是 Eclipse 开发环境中数据库应用开发的最佳搭档。

在本书的后续章节中将采用 MySQL 作为数据库应用开发的支持环境，因此本节将详细介绍 MySQL 数据库管理系统、服务器端管理工具以及客户端浏览工具的安装，同时通过一个完整例子的实现过程介绍如何使用 JDBC 在 Java 程序中连接 MySQL 数据库，并处理数据库中存储的数据。

读者可以在 MySQL 的官方网站上下载到对应 MySQL 的安装程序，下载 MySQL 安装程序的网址如下所示。

```
http://dev.mysql.com/downloads/mysql/4.1.html
```

本书所使用的版本为 MySQL 4.1.14，对应安装程序压缩包的文件名称为 mysql-4.1.14- win32.zip。

安装 MySQL 数据库的具体操作步骤如下所示。

① 解压缩下载的压缩文件，即可获得对应的安装程序，名称为 setup.exe。双击对应的安装程序即可开始 MySQL 数据库系统的安装，屏幕将显示如图 2-55 所示的安装向导对话框。

② 在如图 2-55 所示的对话框中单击 Next 按钮，在如图 2-56 所示的安装类型选择对话

图 2-55　安装向导-步骤 1

图 2-56　安装向导-步骤 2

框中选择软件安装的方式，这里选择 Custom 自定义安装方式。

③ 在如图 2-56 所示的对话框中单击 Next 按钮，在如图 2-57 所示的对话框中用户可以选择要安装的组件，同时也可以单击 Change 按钮依据提示设置待安装数据库系统的位置。此时，屏幕将显示如图 2-58 所示的确认信息对话框，确认该对话框中所设置的安装方式之后，单击 Install 按钮即可开始安装。

图 2-57 安装向导-步骤 3

图 2-58 安装向导-步骤 4

④ 在相关安装过程结束后，屏幕将显示 2-59 所示的新建账户以进行注册的对话框，这里选择 Skip Sign Up 跳过注册过程即可，到此为止数据库系统相关的安装结束，接下来选中如图 2-60 所示对话框中的 Configure the MySQL Server now 以开始配置对应的数据库服务器，此时屏幕将显示如图 2-61 所示的界面，即可根据提示进行服务器的相关配置。

图 2-59 安装向导-步骤 5

图 2-60 安装向导-步骤 6

⑤ 在如图 2-62 所示的服务器实例配置对话框中选择 Detailed Configuration 选项，以进行详细的服务器配置。

图 2-61　安装向导-步骤 7

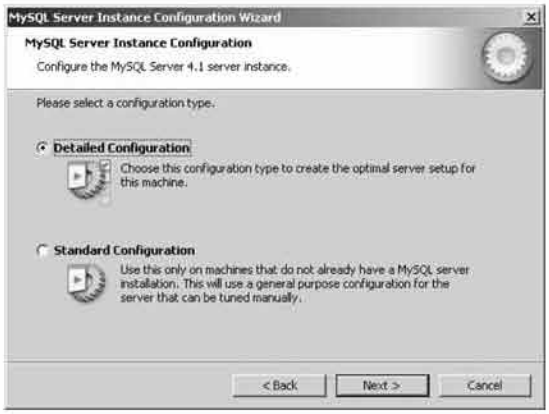


图 2-62　安装向导-步骤 8

⑥ 在如图 2-63 所示的对话框中，选择配置方式为 Developer Machine，以满足开发人员的基本需求。在如图 2-64 所示的对话框中选择第一个选项以配置支持多重功能的 MySQL 服务器。

图 2-63　安装向导-步骤 9

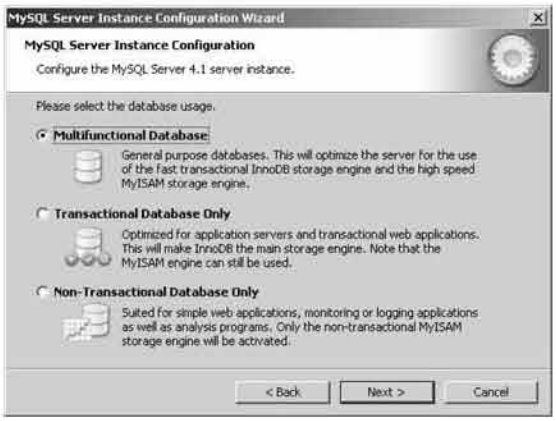


图 2-64　安装向导-步骤 10

⑦ 在如图 2-65 所示的对话框中确认数据库对应文件的安装位置，然后在接下来打开的如图 2-66 所示的对话框中选择第一个选项以支持 OLAP。

⑧ 在如图 2-67 所示的对话框中设置对应的网络传输协议，同时可以设置对应 MySQL 数据库服务器的服务端口，其默认端口为 3306。

⑨ 在如图 2-68 所示的对话框中选择对应的字符集，这里设置为 utf8，在接下来显示的图 2-69 所示的对话框中选中对应选项以便于将 MySQL 服务添加到系统服务中，同时可以设置对应服务的服务名，以便于在程序中按照名称连接对应数据库服务器。

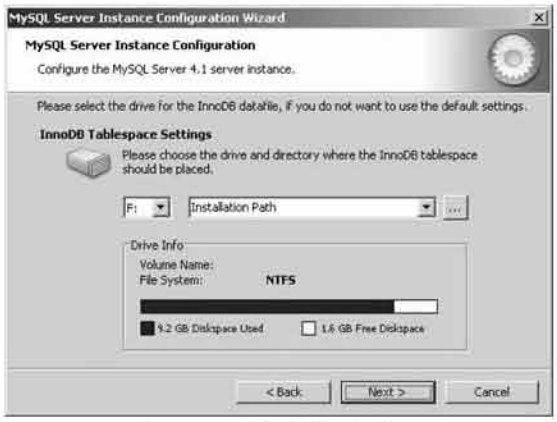


图 2-65 安装向导-步骤 11

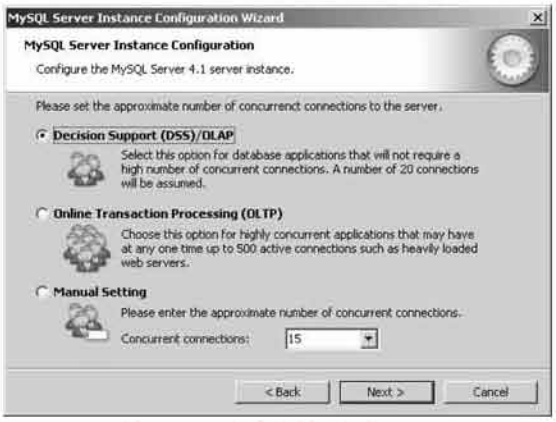


图 2-66 安装向导-步骤 12

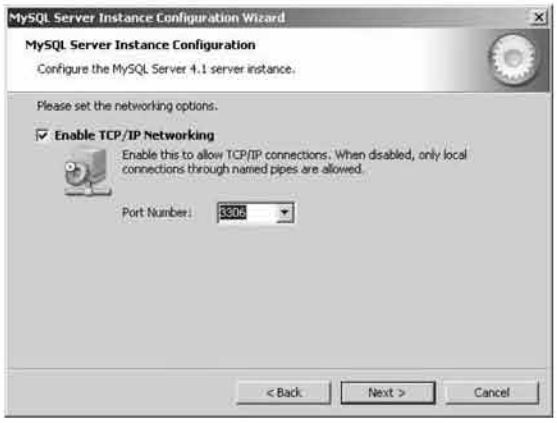


图 2-67 安装向导-步骤 13

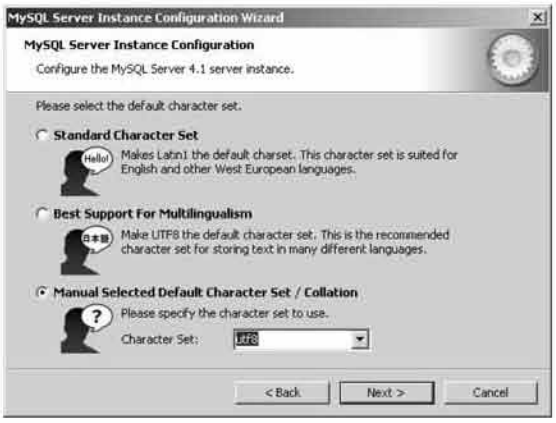


图 2-68 安装向导-步骤 14

图 2-69 安装向导-步骤 15

注意：图 2-68 所示界面中，应该将 MySQL 的字符集设置为 utf8，以解决数据存取时的中文乱码问题。

⑩ 在如图 2-70 所示的对话框中，用户可以直接修改相关的安全性设置，主要是为 MySQL

服务器的超级用户 root 设置对应密码，这里我们设置为 zgy01，后续可以登录服务器时使用。为了最后确认数据库管理系统的安装情况，用户可以在如图 2-71 所示的对话框中进行测试，支持 MySQL 数据库管理系统安装完毕。

图 2-70　安装向导-步骤 16

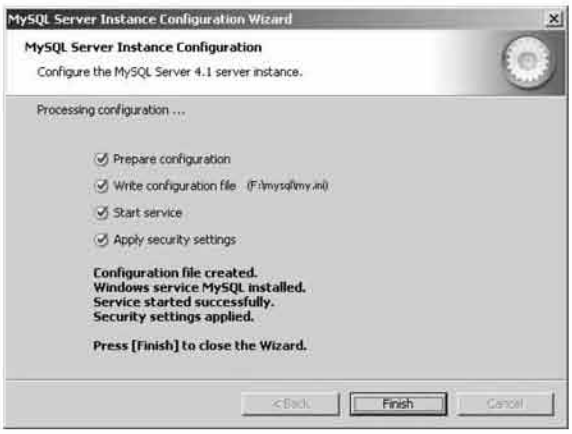


图 2-71　安装向导-步骤 17

由于在安装测试过程中已经启动了对应数据库的服务，因此打开如图 2-72 所示的“控制面板”→“管理工具”→“服务”窗口，即可在系统服务列表中查看到对应 MySQL 数据管理系统的服务已经存在并正常启动起来。

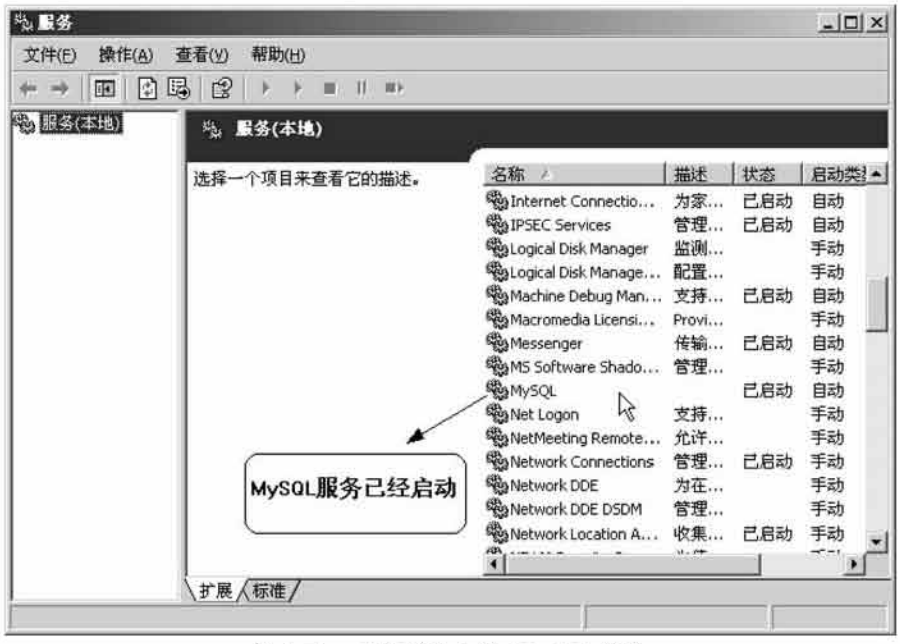


图 2-72　系统服务中的 MySQL 服务

该服务后续将随着操作系统的启动自动启动。

1. 下载并安装 MySQL 服务器端管理工具

为了方便地实现数据库服务器端的管理工作，在 MySQL 的官方网站上还提供了服务器端管理工具 MySQL Administrator 的下载，利用该工具，可以控制服务的启动和停止，也可以以图形界面的方式实现如备份及还原数据库、用户管理等服务器管理的工作。

下载的服务器端管理工具对应的安装文件名为 mysql-administrator-1.1.3-win.msi，双击该文件即可在向导的提示下方便地完成对应软件的安装。

图 2-73 连接数据库服务器对话框

在“开始”→“所有程序”对应的 MySQL 菜单中选择 MySQL Administrator 命令即可启动对应管理工具。在如图 2-73 所示的连接服务器对话框中正确输入 root 用户的密码以及待连接的数据库服务器的相关信息，单击 OK 按钮即可。

此时屏幕将显示如图 2-74 所示的 MySQL Administrator 管理主界面，用户可以在该界面左侧的窗格中选择不同的管理工具，从而方便地实现数据库服务器的日常管理工作。

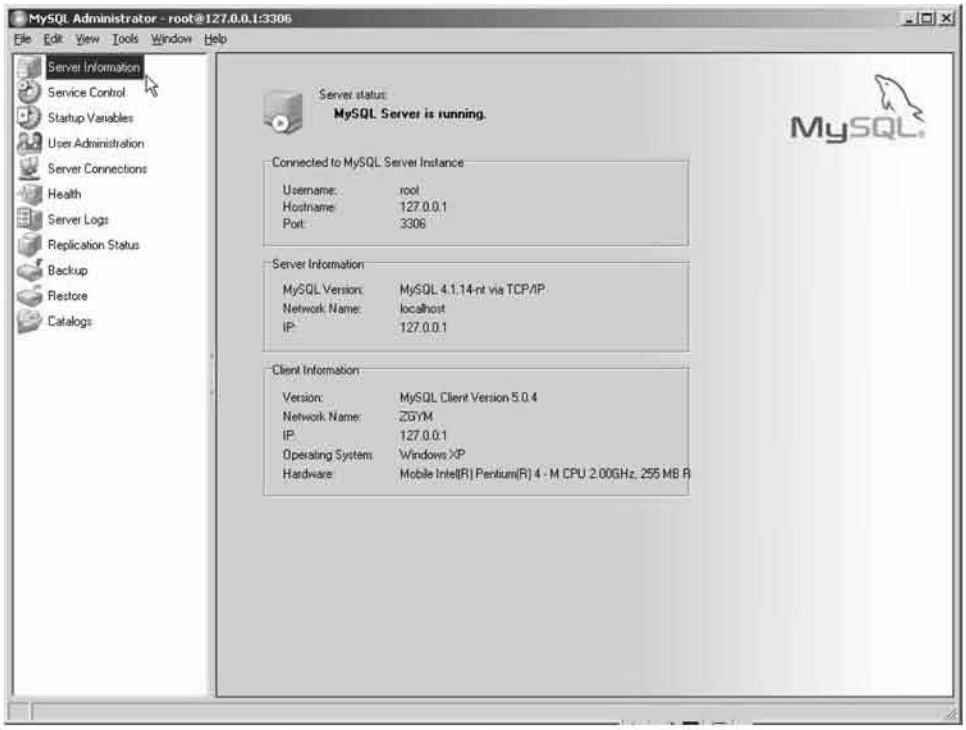


图 2-74 MySQL 数据库管理主界面

2. 下载并安装 MySQL 客户端查询浏览工具

除了提供了管理数据库服务器的辅助工具之外，在 MySQL 的官方网站上还提供了客户端查询浏览的工具，借助该工具可以方便地执行相关的查询语句，浏览对应方案下数据表的相关信息。

下载的客户端查询浏览工具对应的安装文件名为 mysql-query-browser-1.1.15-win.msi，双击该文件即可在向导的提示下方便地完成对应软件的安装。

在“开始”→“所有程序”对应的 MySQL 菜单中选择 MySQL Query Browser 命令即可启动对应查询浏览工具。在如图 2-75 所示的连接服务器对话框中正确输入 root 用户的密码，以及待连接的数据库服务器的相关信息，单击 OK 按钮即可。

图 2-75 连接数据库客户端浏览工具对话框

3. 下载并安装 MySQL 客户端查询浏览工具

此时屏幕将显示如图 2-76 所示的 MySQL Query Browser 查询浏览器工具主窗口。

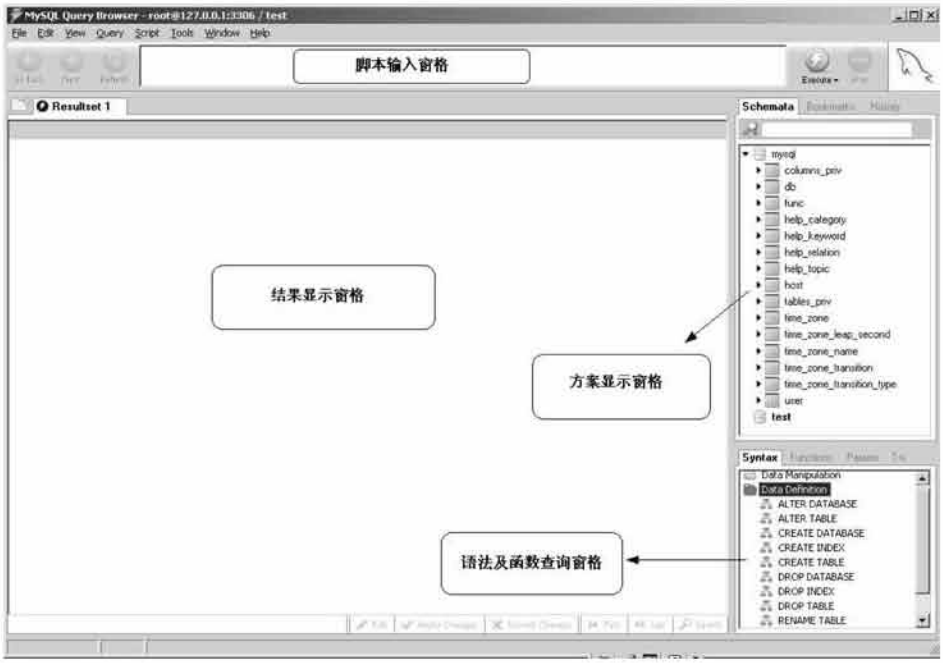


图 2-76 MySQL Query Browser 查询浏览器工具主窗口

需要注意的是，如果待连接的数据库服务器的服务没有正常启动，屏幕将显示如图 2-77 所示的提示信息，表明没有建立与对应数据库服务器的正常连接，单击该对话框中的 Ping 按钮可以测试与服务器的连接情况。

图 2-77 连接数据库服务器对话框

必知必会：MySQL 数据库管理系统中的方案（Schema）

图 2-78 所示窗口主要包含 4 个窗格，具体功能如下所示。

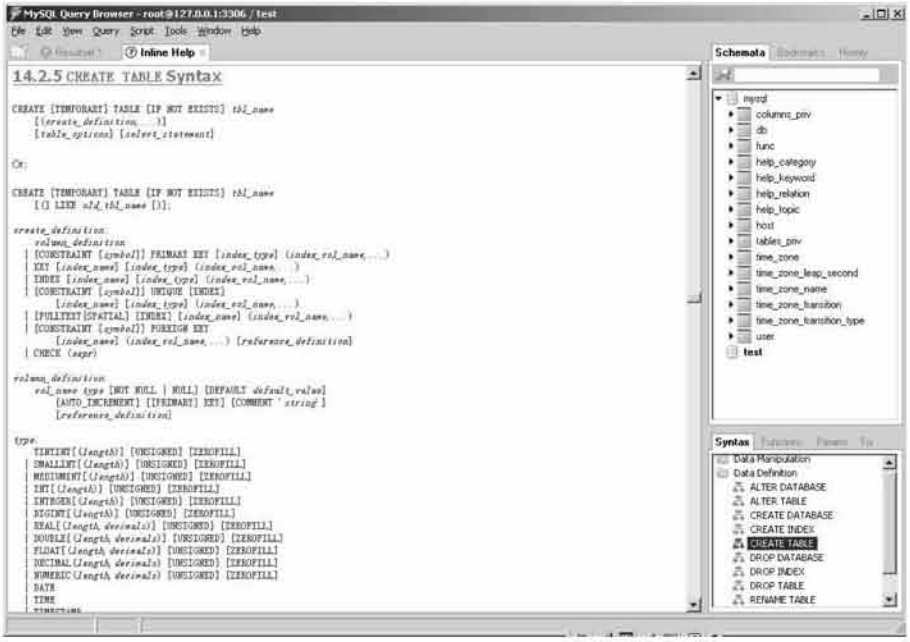


图 2-78 语法查询结果显示

- 脚本输入窗格：该窗格用于输入待执行的 SQL 脚本，例如建表、插入数据及数据查询等语句；
- 结果显示窗格：该窗格用于显示对应脚本执行的结果；
- 方案显示窗格：该窗格中将以树状结构的方式显示所连接数据库服务器中所提供的各个方案，类似于 SQL Server 数据库管理系统中一个一个数据库的概念，用于分别存放针对不同应用的数据；
- 语法及函数查询窗格：该窗格中的各个选项卡给用户提供了相应的帮助，用户可以在该窗格中双击待查询的语句或者函数，屏幕将显示如图 2-78 所示的详细语法介绍。

至此，有关 MySQL 数据库管理系统以及相关辅助工具即安装完毕，接下来通过一个简单的身份验证的例子，介绍如何在 Eclipse 中借助 JDBC 实现 Java 程序和 MySQL 数据库之间的连接。

首先需要从 MySQL 的官方网站中下载对应的 JDBC 连接包，下载的压缩包文件名称为 mysql-connector-java-3.0.17-ga.tar.gz，将其解压缩，即可获得名称为 mysql-connector-java-3.0.17-ga-bin.jar 的包。

必知必会：Web 项目能够使用 JDBC 对应的 jar 包

将解压缩之后的名称为 mysql-connector-java-3.0.17-ga-bin.jar 的包放置在 Web 应用服务器 Tomcat 对应目录下面的 common 子目录中的 lib 目录中，即可使对应的 Web 应用正常使用 JDBC 驱动程序。

案例 2-3　采用 Ajax 技术结合数据库实现 Web 页面的登录验证

下面对案例 2-2 进行改进，以实现针对 MySQL 数据库中已注册的用户信息进行身份的验证。首先准备案例中所需要的数据库数据。按照前面第三章中的介绍在如图 2-79 所示的 MySQL 浏览器工具中新建一个 users 表，以存储用户的相关信息。

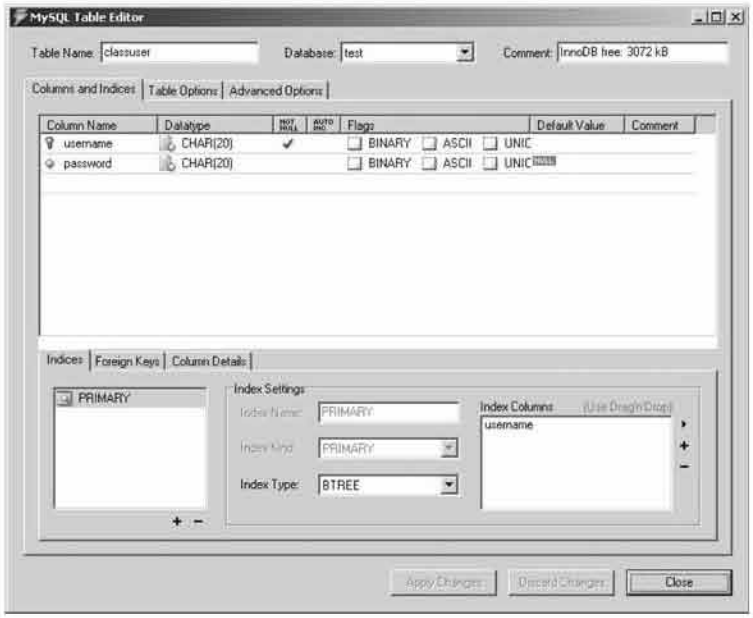


图 2-79　新建数据表

该表中包含了两个字段，分别用于存放对应的用户名以及密码。

创建 MySQL 中对应原始数据的相关脚本如下：

```
create table classuser (
```

```
        username char(20) not null primary key,
        password char(20)
    )

    insert into classuser values('jenny','hi')
    insert into classuser values(mike,'ok')
    insert into classuser values('rose','bye')
```

在 MySQL 中察看数据准备情况的效果如图 2-80 所示。

username	password
jenny	hi
mike	ok
rose	bye

图 2-80 准备测试数据

下面开始创建对应案例的项目。首先新建一个项目，项目名称为 P23_AjaxDb。该项目在案例 2-3 的基础之上，添加了一个封装了对数据库进行操作的相关业务逻辑的 JavaBean。同时对 Servlet 也进行了修改，完成查询数据库中的数据信息完成验证的功能。本例中的页面实现效果以及对应的 JSP 页面的代码与案例 2-3 相同，这里不再赘述。

首先创建一个 JavaBean，以封装对数据库进行操作的业务逻辑，该程序源代码如下：

源文件：DB.java

```
package classmate;

import java.sql.Connection;
……

public class DB {

      Connection connect = null;
      ResultSet rs = null;
      public DB() {
            try {
                  Class.forName("org.gjt.mm.mysql.Driver"); //设置驱动程序类型
            }
            catch(java.lang.ClassNotFoundException e) {
                  e.printStackTrace();
            }
      }
      //执行查询语句的方法
      public ResultSet executeQuery(String sql) {
            try {
                  connect = DriverManager.getConnection("jdbc:mysql://localhost/test",
"root", "zgy01");  //建立与数据库服务器的连接
                  Statement stmt = connect.createStatement(ResultSet.TYPE_SCROLL_
INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
```

```
                rs = stmt.executeQuery(sql);//执行指定的数据查询语句
            }
            catch(SQLException ex) {
                ex.printStackTrace();
            }
            return rs;
        }
        //执行增、删、改语句的方法
        public int executeUpdate(String sql) {
            int result = 0;
            try {
                connect = DriverManager.getConnection("jdbc:mysql://localhost/test",
"root", "zgy01");
                Statement stmt = connect.createStatement();
                result = stmt.executeUpdate(sql);//执行指定的数据操作语句
            }
            catch(SQLException ex) {
                System.err.println(ex.getMessage());
            }
            return result;
        }
        //关闭数据库连接的方法
        public void close(){
            if(connect!=null){
                try{
                    connect.close();
                    connect = null;
                }catch(SQLException ex) {
                    System.err.println(ex.getMessage());
                }
            }
        }
    }
    ……
```

下面介绍一下对 Servlet 所做的修改，与案例 2-3 相同的部分这里就不再重复了。

源文件：LoginAction.java

```
package classmate;

import java.io.IOException;
……
public class LoginAction extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
    }

    /*
     * 处理<GET> 请求方法
     */
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
……
     out.println("<response>");

     //连接数据库进行身份验证
     DB db = new DB();
      ResultSet rs;
     String strSql=null;

     StrSql="select * from classuser where username='"
           + uname + "' and password='" + psw + "'";
      rs = db.executeQuery(strSql);

      try{
           if ( rs.next()) {
                 out.println("<res>" + "热烈的欢迎您!" + "</res>");
           }else{
                 out.println("<res>" + "对不起,登录失败! " + "</res>");
           }
     } catch(SQLException e) {
           e.printStackTrace();
     }
  out.println("</response>");
  out.close();
}
}
……
```

在上面的程序中依据获取到的浏览器客户端的信息组合成查询语句，然后依据查询的结果返回不同的响应数据。

至此本章中有关商务系统的身份验证模块案例即全部完成。下一节将结合该案例介绍基本的 Ajax 技术的工作流程。

2.2.3 Ajax 工作流程总结

在上面的案例中，许多最初开始接触 Ajax 的读者普遍会遇到的问题是搞不清它的具体运行流程、或者说 Ajax 的具体工作原理。下面通过上述的案例中所涉及到的整个工作流程结合 Ajax 作一个总体的介绍。

图 2-81 所示为采用 Ajax 技术之后的 MVC 设计模式的工作示意图。

在采用传统的 MVC 设计模式实现的 WEB 应用中，由于浏览器和服务器所使用的请求/响应的断开式网络通信模式，决定了在 Model 层中对数据进行了更新之后，无法主动向 View 层发出数据更新事件，所以一般所采用的实现方式是由 Model 层执行完具体的业务逻辑之后通知 Control 层，然后由 Control 层承担向 View 发送数据更新的功能。

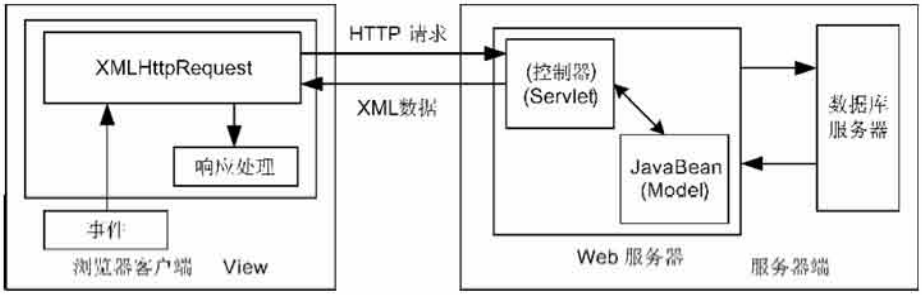


图 2-81 采用 Ajax 技术的 MVC 设计模式

而在加入了 Ajax 技术之后，可以借助 Ajax 实现异步响应的 OnReadyStateChange 事件在客户端程序中实现事件监听功能。这样一来，整个 MVC 设计模式的工作过程发生了改变。下面结合前面的案例，具体解释一下在使用 Ajax 技术之后的具体工作流程。

首先在浏览器客户端创建对应的 XMLHttpRequest 对象，在上述案例的 login.jsp 页面中可以找到对应的代码如下：

```
var XMLHttpReq = false;
    //创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
        if(window.XMLHttpRequest) { //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
        }
        else if (window.ActiveXObject) { // IE 浏览器
            try {
                XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try {
                    XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
    }
```

在上面的代码中，依据不同的浏览器类型创建对应的 XMLHttpReques 对象。

接下来，当用户单击登录按钮提交对应的请求之后，即可通过内置的 Ajax 核心对象 XMLHttpRequest 以异步的方式发送请求，在上例中发送的是 login 请求，同时提供对应的表单数据信息。login.jsp 页面中对应的程序代码如下：

```
// 身份验证函数，验证通过提交对应的请求
    function userCheck() {
        var uname = document.myform.uname.value;
        var psw = document.myform.psw.value;
        if(uname=="") {
            window.alert("用户名不能为空。");
            document.myform.uname.focus();
            return false;
```

```
            }
            else {
                  sendRequest('login?uname='+ uname + '&psw=' + psw);
            }
          }
//发送请求函数
    function sendRequest(url) {
          createXMLHttpRequest();
          XMLHttpReq.open("GET", url, true);
          XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
          XMLHttpReq.send(null);  // 发送请求
    }
```

在请求提交之后为 Ajax 核心对象的 onreadystatechange 指定好响应的函数之后，该监听器就开始监听工作。

接下来浏览器客户端提交的请求将被发送到服务器端，按照配置文件 web.xml 中的配置将请求提交给对应的 Servlet 进行处理，对应配置文件中的代码如下：

```
<!—设置对应 Servlet 对应的类型-->
    <servlet>
          <servlet-name>ms1</servlet-name>
          <servlet-class>classmate.LoginAction</servlet-class>
    </servlet>
<!—设置请求对应相应的 Servlet 名称-->
    <servlet-mapping>
          <servlet-name>ms1</servlet-name>
          <url-pattern>/login</url-pattern>
    </servlet-mapping>
```

当请求提交给控制层对应的 Servlet 之后，负责处理请求的 LoginAction 首先获取随请求一起提交过来的表单数据，然后调用 JavaBean 中的有关数据库操作的业务逻辑进行身份验证的处理工作，依据验证的结果，在 Servlet 中返回对应格式的响应消息，对应 LoginAction.jsp 中的代码如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String uname = request.getParameter("uname");
          String psw = request.getParameter("psw");
          //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");
          if(uname.equals("jenny") && psw.equals("hi")){
                out.println("<res>" + "热烈的欢迎您!" + "</res>");
          }else{
```

```
                out.println("<res>" + "对不起,登录失败! " + "</res>");
            }
            out.println("</response>");
            out.close();
        }
```

所调用的对应 JavaBean 中的代码如下：

```
//执行查询语句的方法
    public ResultSet executeQuery(String sql) {
        try {
            connect = DriverManager.getConnection("jdbc:mysql://localhost/test",
"root", "zgy01");  //建立与数据库服务器的连接
            Statement stmt = connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
            rs = stmt.executeQuery(sql);//执行指定的数据查询语句
        }
        catch(SQLException ex) {
            ex.printStackTrace();
        }
        return rs;
    }
```

为了能够让读者在第一时间体会到 Ajax 技术中对 XML 文档的处理，该案例中返回的消息格式采用的 XML 格式进行封装，具体生成的格式效果如下：

如果验证通过，则返回的数据格式为：

```
<response>
    <res 热烈的欢迎您</res>
</response>
```

如果验证未通过，则返回的数据格式为：

```
<response>
    <res>对不起,登录失败! </res>
</response>
```

当响应的数据返回浏览器客户端时，在客户端一直处于监听状态的 Ajax 核心对象的 onreadystatechange 将会监听到对应的响应数据，此时就会按照事先设置好的业务逻辑进行处理，例如采用 DOM 进行页面的更新处理，本例中使用的 window 对象提供的 alert()方法进行提示信息的输出。login.jsp 页面中对应的程序代码如下：

```
// 处理返回信息函数
    function processResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
            var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].firstChild.data;
            window.alert(res);
            } else { //页面不正常
                window.alert("您所请求的页面有异常。");
            }
        }
    }
```

需要注意的是从服务器端发送回的消息可以是采用 XML 封装的数据，也可以是一段文本信息，甚至可以是一段 HTML 代码，一般不推荐发送 HTML 格式的数据，推荐还是用 XML 封装业务数据即可。

从上面的例子中可以看出：在 Ajax 技术中最核心的对象是 XMLHttpRequest，该对象提供了对应的方法和属性来完成向服务器端提交请求，以及接收来自服务器端处理的结果等功能。在后续章节中，将会对 Ajax 技术中所涉及到的技术进行详细的介绍。

CHAPTER 3

第 3 章　Ajax 核心技术

上一章中对有关商务网站设计中常见的身份验证应用模块实现过程进行了详细介绍，相信读者对采用 Ajax 技术实现 Web 应用中异步交互方式的工作过程及工作原理应该有了一个基本的了解。本章将继续从 Ajax 的各个相关技术出发详细介绍在 Ajax 技术中各个核心技术的工作原理，以及 Ajax 应用运行机制。

本章仍然会通过实用的案例介绍 Ajax 技术中相关核心技术的具体应用，使初学者能够真正体会到这些技术在实际 Web 应用开发过程中的应用场合及实现方式。

3.1　JavaScript 脚本

JavaScript 是一种可以与 HTML 标记语言混合使用的脚本语言，其编写的程序可以直接在浏览器中解释执行。因此使用脚本语言编写的程序可以在浏览器的支持下跨平台运行。

3.1.1　如何插入 JavaScript 脚本

在动态网页编程中，脚本语言承担着非常重要的作用，尽管由于开发环境、调试工具的原因曾经一度使许多开发人员放弃对脚本语言的使用，但是无论是实现客户端的动态效果，还是服务器端与数据库相关的动态页面，往往都会需要以脚本语言实现浏览器对象的操作和使用。特别是在 Ajax 技术中，正是通过 JavaSrcipt 脚本语言实现对 Ajax 核心对象的控制。

本书不对 JavaScript 脚本语言的基本语法作详细介绍。而是从掌握 Ajax 核心技术的角度，通过几个实现实际效果的案例介绍 JavaScript 脚本语言的应用场合和实现方式。在 Eclipse 中新建一个项目，项目的名称为“P31_JavaScriptTest”。

首先介绍一下如何在页面中插入 JavaScript 脚本语言。在该项目中新建一个 HTML 的页面文件，页面名称为“3-1.htm”。读者可以使用 MyEclipse 中提供的针对 HTML 页面设计的工具，单击 Eclipse 编辑窗格下方的“Design”设计选项卡即可查看到如图 3-1 所示的设计界面，在该界面中可以选择 HTML 页面中出现的各种表单元素，生成对应的 HTML 中的标记。

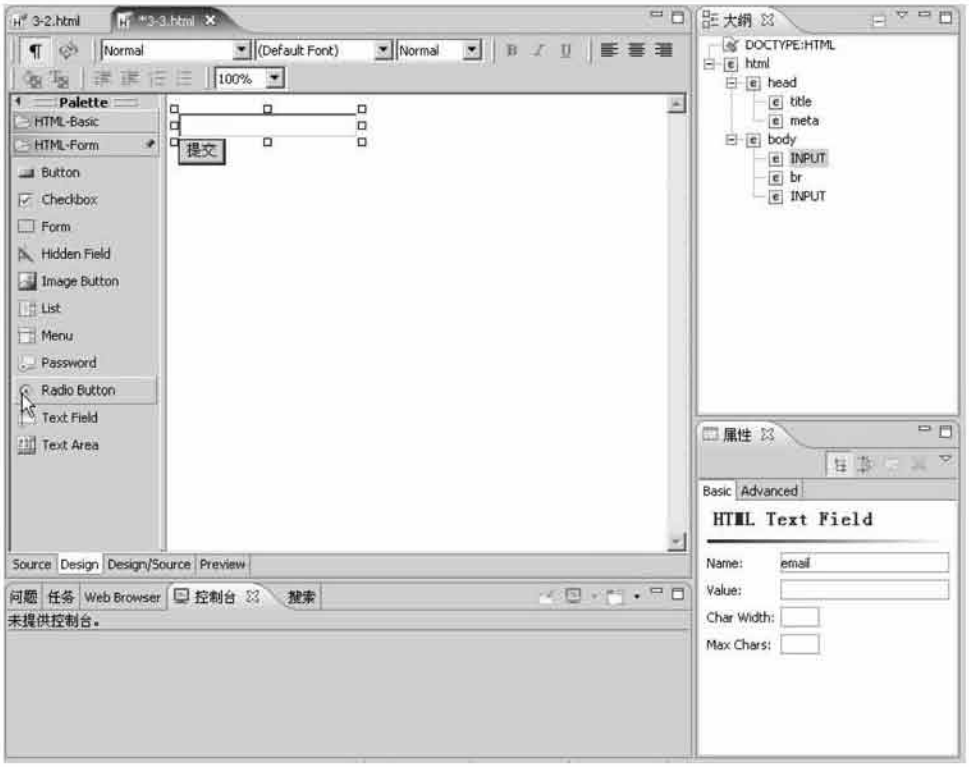


图 3-1 MyEclipse 提供的 HTML 的设计工具

在 HTML 文档中插入脚本语言可以使用<script>标记。例如，在下面的程序中，我们在对应的网页文件中插入了一段 JavaScript 的脚本，代码如下：

源程序名称：3-1.html

```
<html>
    <head>
        <script language ="JavaScript">
            // 下面添加 JavaScript 代码.
            window.alert("欢迎使用脚本语言!");
        </script>
     </head>
    <body>
    </body>
</html>
```

在该程序<script>标记中使用 language 属性声明了所使用的是哪一种脚本语言。本例中脚本语言一共使用了两行代码，其中“// 下面添加 JavaScript 代码.”为注释语句。“window.alert("欢迎使用脚本语言!");”为警告窗口输出语句，该语句的功能是实现提示信息的输出，在上一章的案例中就是采用该语句实现了验证结果的输出。

由于脚本语言添加在了<head>部分，因此在装载页面的主体内容之前，浏览器中首先弹

出一个警告窗口，在该窗口中显示的警告信息有 alert（）方法的字符串类型的参数决定。同时在该窗口中将自动出现一个“确定”按钮，在用户单击该按钮后，警告提示窗口消失，页面继续后面的装载过程。单击 Eclipse 编辑窗格下方的“Preview”预览选项卡即可查看到如图 3-2 所示的页面效果。

图 3-2　脚本语言实现的效果

在实际应用中考虑到脚本语言书写的代码可能会比较长，开发人员也可以将脚本语言单独写在扩展名为.js 的脚本文件中，然后在<script>标记中设置 src 属性的值为脚本文件的位置和名称即可。例如，将上例中的脚本语句存放在 init.js 文件中，然后使用下面的语句。这种书写方式在实际的页面设计中同样比较常见。

```
<script language="JavaScript" src = "init.js">
</script>
```

3.1.2　事件驱动的处理机制

在动态网页的编程中，一般都是通过事件驱动的方式调用脚本语言中定义的函数。下面介绍一下有关事件的概念及工作模式。

必知必会：事件驱动的处理机制

为了能够实现具有交互功能的动态页面，需要借助事件驱动的处理机制，即在特定的事件发生时进行特定的操作。事件实际上是浏览器与用户之间实现交互的一种机制。例如，针对页面中按钮元素的“onclick”事件，可以让浏览器知道用户单击了该按钮，即发生了针对该按钮的单击事件。一旦事件发生，浏览器具体会进行什么样的处理，则可以由脚本语言通过相应的程序代码，例如定义对应的事件处理函数来进行处理。

在实际的页面设计中，正是由于引入了事件驱动的处理机制，才使得页面具有了真正的交互的功能，同时也可以进一步丰富页面的动态效果。

案例 3-1 借助 JavaScript 实现丰富的页面交互效果

在下面的程序中，读者可以初步体会到 JavaScript 脚本语言中事件处理机制的工作方式和工作过程，我们继续在项目“P31_JavaScriptTest”中新建 HTML 文件。

源程序名称：3-2.html

```
<html>
      <head>
      </head>
      </body>
           <marquee onmouseover="this.stop()" onmouseout="this.start()" scrollDelay=110
><a href="" target=_blank><font color=#ff0000>热点新闻:Ajax 成为互联网技术中的新宠
</font></a></marquee><br>;
      </body>
</html>
```

该页面效果如图 3-3 所示。

图 3-3 事件驱动机制

在该程序中实现了网页中常见的滚动新闻的效果，其中新闻字幕的滚动采用的是滚动字幕<marquee>组件，使用超级链接标记实现了新闻页面的跳转，同时在<marquee>标记中添加了两个针对事件的处理。其中“onmouseover”事件将在鼠标指针悬停在该滚动字幕对象上方时发生，在本例中一旦发生这个事件，滚动字幕对象调用该类对象的 stop()方法，使得滚动字幕停止滚动。与之相对应，当鼠标指针离开滚动字幕上方时，即一旦发生“onmouseout”事件，此时将调用该对象的 start()方法使得滚动字幕继续滚动。

通过上面这个简单的案例，相信读者对事件驱动的处理机制，包括其基本工作原理已经有所了解，下面结合 JavaScript 脚本语言介绍几个比较常见的事件。

onclick 事件即鼠标单击事件，该事件可以让页面中设置的按钮真正发挥作用。例如下面的程序：

源程序名称：3-3.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
  <head>
     <title>3-3.html</title>
     <meta http-equiv="content-type" content="text/html; charset=UTF-8">
     <script language="JavaScript">
                  function checkValid()
                  {
                  var tmp1,temp2,str1;
                  //获得 text1 的值
                  str1 = document.myform.email.value;
                  //获取第一个@的位置
                  tmp1 = str1.indexOf('@');
                  //获取最后一个@的位置
                  tmp2 = str1.lastIndexOf('@');
                  if(str1.length == 0)
                         //不合法，字符串为空串
                         window.alert("请输入内容后再提交");
                  else if(tmp1 < 0)
                         //不合法，字符串中不包含@字符
                         window.alert("字符串中不包含@");
                  else if(tmp1 == 0 || tmp1 == str1.length-1)
                         //不合法，@字符出现的位置不对
                         window.alert("@字符出现的位置不对");
                  else if(tmp1 != tmp2)
                         //不合法，@字符出现了不止一个
                         window.alert("@字符出现了不止一个");
                  else
                         //合法的 E-mail 地址
                         alert("E-mail 地址合法有效");
               }
     </Script>

  </head>
  <body>
       <form name="myform">
          请输入您的 E-mail 地址：<br>
          <input type=text value="" name="email">
          <p><input type=button value="提交"name="ok"onclick="checkValid()">
       </form>
  </body>
</html>
```

在该程序中当用户单击提交按钮时，将调用在脚本中定义的函数“checkValid()”。依据用户输入的不同内容，进行 E-mail 地址有效性的检查，如果用户输入的地址无效，将针对具体情况对用户进行提示，否则输出为一个合法有效的 E-mail 地址。本例中使用的 document.myform.email.value 是在浏览器环境中使用 DOM（有关 DOM 将在本章后续的内容中进行介绍）完成对象的定位，并对其属性进行操作的用法，也可以简写成 myform.email.value。

图 3-4 所示为用户输入的地址中包含了多个@符号之后的提示效果。

图 3-4 E-mail 地址的有效性检查

onchange 事件即当对应元素的值发生改变时触发的事件。该事件经常用在针对列表框数据处理的场合中。下面给出一个实例，以让读者体会到该事件的应用场合。

源程序名称：3-4.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>3-4.html</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <script language="JavaScript">
        //当列表框中的被选中选项发生改变时调用的函数
        function change()
        {
            //依据选择的不同选项，显示不同的作者信息
            switch(myform.option1.value)
            {
                case "s1":
                    myform.txt1.value="张桂元";
                    break;
                case "s2":
                    myform.txt1.value="贾燕枫";
                    break;
                case "s3":
                    myform.txt1.value="张宇翔";
                    break;
            }
```

```
        }
    </Script>
  </head>
  <body>
    <p>书籍作者信息查询：
     <form name="myform">
       <select type="select" name="option1" onchange="change()">
         <option selected value="s0"><-----请选择-----></ option >
         < option value="s1">Web 2.0 开发入门与实践（Java）</ option >
         < option value="s2">Web 2.0 开发入门与实践（.NET）</ option >
         < option value="s3">Ajax 核心技术</ option >
       </select>
       <input type="text" name="txt1" value="作者信息">
     </form>
  </body>
</html>
```

在该程序实现的页面中包含了两个表单元素，一个为列表框，另一个为文本框。当用户在列表框中选择不同的选项，即不同书籍的名称时，在文本框中将显示该书作者的相应信息，页面效果如图 3-5 所示。

图 3-5　书籍作者信息查询

在上面的程序中“myform.option1.value”可以获取对应列表框的值，而 onchange 事件将在列表框的值改变时发生。

onload 事件及 onunload 事件分别在 HTML 页面载入和卸载时发生。因此一般有关页面的初始化工作往往放在 onload 事件发生时完成。示例如下：

源程序名称：3-5.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
```

```
    <title>3-5.html</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
     <Script Language="JavaScript">
          function init()
          {
                alert("欢迎光临！");
          }
          function finiish()
          {
                alert("下次再来！");
          }
    </Script>
  </HEAD>
  <BODY OnLoad="init()" OnUnLoad="finiish()">
  </BODY>
</HTML>
```

该页面的效果为，在页面刚刚打开时，将弹出“欢迎光临!”窗口，当用户关闭该页面时，将弹出“下次再来!”窗口。

onfoucs 事件及 onblur 事件分别在对应元素获得焦点或者失去焦点时发生。例如，当用户将插入点定位在某个文本框中时，该文本框即获得焦点。当插入点离开该文本框时，即失去焦点。示例如下：

源程序名称：3-6.html

```
<html>
<script language="JavaScript">
     function begin()
     {
          usrId.value="";
     }
     function end()
     {
          alert("请确认输入的信息，谢谢！");
     }
</script>
<body >
<input type="text" value="请输入用户名" name="usrId" onfocus="begin()" onblur="end()">
</body>
</html>
```

该页面的效果为，首先显示文本框，并在其中显示相应的提示信息“请输入用户名”，一旦用户将插入点定位到文本框中，提示信息将自动消失。当用户结束输入，离开文本框时，将显示告别窗口。

结合 JavaScript 脚本可以使用的事件还有许多，本书就不再一一举例了，下表中给出了可以结合脚本使用的相关事件，读者可以在需要时进行查阅。

表 3-1　　JavaScript 事件汇总

事　件		触 发 时 机
一般事件	onclick	鼠标单击时触发此事件
	ondblclick	鼠标双击时触发此事件
	onmousedown	按下鼠标时触发此事件
	onmouseup	鼠标按下后松开鼠标时触发此事件
	onmouseover	当鼠标指针移动到某对象范围的上方时触发此事件
	onmousemove	鼠标移动时触发此事件
	onmouseout	当鼠标指针离开某对象范围时触发此事件
	onkeypress	当键盘上的某个键被按下并且释放时触发此事件
	onkeydown	当键盘上某个按键被按下时触发此事件
	onkeyup	当键盘上某个按键被按放开时触发此事件
页面相关事件	onabort	图片在下载时被用户中断
	onbeforeunload	当前页面的内容将要被改变时触发此事件
	onerror	出现错误时触发此事件
	onload	页面内容完成时触发此事件
	onmove	浏览器的窗口被移动时触发此事件
	onresize	当浏览器的窗口大小被改变时触发此事件
	onscroll	浏览器的滚动条位置发生变化时触发此事件
	onstop	浏览器的停止按钮被按下时触发此事件或者正在下载的文件被中断
	onunload	当前页面将被改变时触发此事件
表单相关事件	onblur	当前元素失去焦点时触发此事件
	onchange	当前元素失去焦点并且元素的内容发生改变而触发此事件
	onfocus	当某个元素获得焦点时触发此事件
	onreset	当表单中 RESET 的属性被激发时触发此事件
	onsubmit	一个表单被递交时触发此事件
滚动字幕事件	onbounce	在 Marquee 内的内容移动至 Marquee 显示范围之外时触发此事件
	onfinish	当 Marquee 元素完成需要显示的内容后触发此事件
	onstart	当 Marquee 元素开始显示内容时触发此事件
编辑事件	onbeforecopy	当页面当前被选择内容将要复制到浏览者系统的剪贴板前触发此事件
	onbeforecut	当页面中的一部分或者全部的内容将被移离当前页面[剪贴]并移动到浏览者的系统剪贴板时触发此事件

续表

事件		触发时机
编辑事件	onbeforeeditfocus	当前元素将要进入编辑状态
	onbeforepaste	内容将要从浏览者的系统剪贴板传送[粘贴]到页面中时触发此事件
	onbeforeupdate	当浏览者粘贴系统剪贴板中的内容时通知目标对象
	oncontextmenu	当浏览者按下鼠标右键出现菜单时或者通过键盘的按键触发页面菜单时触发的事件
	oncopy	当页面当前的被选择内容被复制后触发此事件
	oncut	当页面当前的被选择内容被剪切时触发此事件
	ondrag	当某个对象被拖动时触发此事件 [活动事件]
	ondragdrop	一个外部对象被鼠标拖进当前窗口或者帧触发此事件
	ondragend	当鼠标拖动结束时触发此事件，即鼠标的按钮被释放了
	ondragenter	当被鼠标拖动的对象进入其容器范围内时触发此事件
	ondragleave	当被鼠标拖动的对象离开其容器范围内时触发此事件
	ondragover	当某被拖动的对象在另一对象容器范围内拖动时触发此事件
	ondragstart	当某对象将被拖动时触发此事件
	ondrop	在一个拖动过程中，释放鼠标键时触发此事件
	onlosecapture	当元素失去鼠标移动所形成的选择焦点时触发此事件
	onpaste	当内容被粘贴时触发此事件
	onselect	当文本内容被选择时触发此事件
	onselectstart	当文本内容选择将开始发生时触发的事件
外部事件	onafterprint	当文档被打印后触发此事件
	onbeforeprint	当文档即将打印时触发此事件
	onfilterchange	当某个对象的滤镜效果发生变化时触发的事件
	onhelp	当浏览者按下F1键或者浏览器的帮助选择时触发此事件
	onpropertychange	当对象的属性之一发生变化时触发此事件
	onreadystatechange	当对象的初始化属性值发生变化时触发此事件

通过前面的实例，可以了解到脚本语言的一些基本用法，同时也可以了解到：一个利用JavaScript实现交互功能的Web页面总是包含下面三个部分的内容。

- 在<Head>部分定义一些JavaScript函数，其中的一些可能是事件处理函数，另外一些可能是为了配合这些事件处理函数而编写的普通函数；
- HTML本身的各种控制标记；
- 拥有事件处理属性的HTML标记，主要涉及到一些与用户实现交互的界面元素。这

些元素可以把 HTML 同 JavaScript 代码相连。

在 Ajax 技术中，JavaScript 实现的是沟通作用。通过 JavaScript 脚本将相关的各种对象串接在一起使用。

3.2 XHTML 和 CSS

XHTML 的全称是可扩展的超文本标记语言（Extensible HyperText Markup Language），该语言是一种为适应 XML 可扩展标记语言（Extensible Markup Language）而重新改造的 HTML 超文本标记语言（HyperText Markup Language），它是 HTML 的下一代语言。

3.2.1 XHTML 与 HTML 之间的区别

XHTML 设计的目的是为了实现从 HTML 向 XML 过渡，它结合了 XML 中的部分强大功能及 HTML 中大部分的简单特性。为了适应新的网络应用阶段提出的更多需求，XHTML 增加了许多可扩展性和灵活性的设计。

相对原来的 HTML 来讲，XHTML 从设计上显得更加严密，更加清晰。其与 XML 的关系决定了它的用户可以很自然地实现从 HTML 到 XML 的转换。XHTML 本身与 HTML4.01 是兼容的。通过 XML 与 HTML 的结合，发挥它们各自的长处，就获得了现在并且在将来都可以使用的标记语言——XHTML。

必知必会：HTML 与 XHTML 之间的主要区别

首先我们关注一下 XHTML 和 HTML 之间的主要区别。

- XHTML 元素必须合理嵌套；
- XHTML 文档必须格式正确；
- XHTML 文档中对应的标签名称必须是小写；
- 所有 XHTML 元素必须关闭。

下面通过一些代码片段作一些对比。在 HTML 中，下面的代码虽然不够严格，但是允许使用。

```
<ul>
  <li>.NET 类图书</li>
  <li> Java 类图书
     <ul>
        <li>Eclipse 开发入门与项目实践</li>
        <li>Web2.0 开发入门与项目实践</li>
     </ul>
  <li>Ajax 类图书</li>
</ul>
```

这是在列表嵌套的时经常会出现的一个问题，就是忘记了在列表中插入的新列表必须

在一个<li>标记中，并应正常结束。但是在 HTML 页面中，依然可以显示出如图 3-6 所示的效果。

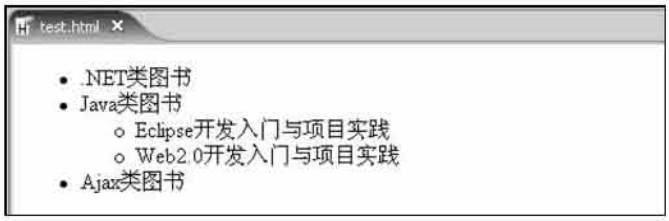


图 3-6 列表显示效果

按照 XHTML 的要求，上面的代码应该调整为下面的书写格式。

```
<ul>
  <li>.NET 类图书</li>
  <li> Java 类图书
     <ul>
        <li>Eclipse 开发入门与项目实践</li>
        <li>Web2.0 开发入门与项目实践</li>
        </Li>
     </ul>
  <li>Ajax 类图书</li>
</ul>
```

在传统的HTML页面中，允许没有<html>标记，但是按照XHTML的要求，所有的XHTML标记必须被嵌套使用在<html> 根标签之中。所有其他的标签可以有自己的子标签。位于父标签之内的子标签也必须成对并且正确的嵌套使用。因此，在 XHTML 页面中，正确规范的文档基本结构如下所示。

```
<html>
     <head> ... </head>
     <body> ... </body>
</html>
```

在 XHTML 文档中，所有标记的名称都必须小写，这是因为 XHTML 文档是 XML 应用程序的格式，在 XML 中是区分大小写的，所以在 XHTML 中如果大小写不同将被认为是两种不同的标签。

在 XHTML 文档中，所有非空的标记都必须进行关闭，例如，在 HTML 中大多数设计者习惯于单独使用段落标记<p>，示范代码如下。

```
<p>第一章  Web2.0 简介
<p>1.1  Web2.0 与 Ajax 简介
```

但是按照 XHTML 的要求，对应的代码应该规范成下面的格式。

```
<p>第一章  Web2.0 简介</p>
<p>1.1  Web2.0 与 Ajax 简介</p>
```

上述代码的执行效果如图 3-7 所示。

同时，对于原来 HTML 中空的标记，也应该进行关闭，例如，下面的代码在 HTML 中是可以接受的。

```
Ajax 成为行业新宠<br>
<hr>
```

而在 XHTML 中，对应的标记要求必须关闭，注意在标记名称和结束符之间有一个空格，这样确保能够与目前的浏览器兼容。规范的代码如下所示。

```
Ajax 成为行业新宠<br />
<hr />
```

上述代码执行效果如图 3-8 所示。

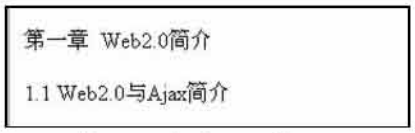


图 3-7 段落显示效果

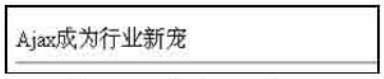


图 3-8 页面显示效果

3.2.2 XHTML 的语法规则

XHTML 与 HTML 相比，在语法规则上发生的变化之外，在 XHTML 中还有一些特殊的语法规则需要强调，主要包括下面几点。

- XHTML DTD 定义强制元素；
- 属性名称必须为小写；
- 属性值使用双引号；
- 属性简写是不允许的；
- 用 id 属性来替代原来的 name 属性。

根据上面的语法规则，所有的 XHTML 文档都必须有一个 DOCTYPE 声名。html、head 和 body 元素必须出现，并且 title 必须在 head 元素里，同时属性声明必须要完整。此外，在 HTML 中，name 属性可以用来标识 identify <a>, <applet>, <form>, <frame>, <iframe>, <img> 和 <map>标记。XHTML 1.0 Strict 和 XHTML 1.1 standards 已经删除了对 name 属性的支持。这里应该使用 id 属性来唯一标识一个页面上的元素

这样一来，更加规范的 XHTML 文档需要由三个主要方面构成：文档类型、头部信息及主干部分。完整的文档结构为：

```
<!DOCTYPE ...>
<html>
<head>
<title>... </title>
</head>
<body> ... </body>
</html>
```

其中，文档类型声明定义了文档的类型。目前一共有三种 XHTML 文档类型的声明方式，分别对应严格类、过渡类及框架集类的声明。三种的声明格式如下。

```
严格类型
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

过渡类型
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

框架集类型
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

通过上面的介绍，我们了解到 XHTML 是一种非常严格的标记语言。它的规则很简单，但是这些规则实际上决定了它的可扩展性很小 ，也就是说，设计人员不能随意编写自己的定义来表示语言如何动作，XHTML 要求设计人员必须遵循规则。

XHTML 的优势主要在于两点。

- 可扩展性

XML 文档要求格式良好（元素嵌套正确）。使用 HTML，添加新的元素组需要更改整个 DTD。在基于 XML 的 DTD 中，新的元素组只需要内部一致并且格式良好，就可以添加到现有的 DTD 中。这实际上极大地简化了新元素集合的开发和集成。

- 可移植性

在现在的互联网时代，用户已经越来越频繁地使用非台式设备来访问因特网。在大多数情况下，这些设备不可能具备台式计算机的计算处理能力，并且也不能够像标准桌面浏览器那样适用于格式差的 HTML。实际上，如果这些非桌面浏览器没有接收到格式良好的标记（HTML 或 XHTML），它们可能根本无法显示文档。

因此，XHTML 同样是为了适用互联网发展的新的需求而产生的。下面给出一个按照 XHTML 要求书写的页面文档的示例，注意页面中的严格声明方式。

案例 3-2　使用 XHTML 语法规则实现 Web 页面

在 Eclipse 中新建一个项目，项目的名称为“P32_XHTML”。在该项目中新建一个 HTML 文档，页面的源代码如下：

```
源程序名称：regist.html
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html>
    <head>
        <title>regist.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
        <script language="JavaScript" type="text/javascript" src="check.js"></script>
    </head>
    <body>
        <center>
            <form name="myform" action="">
                <p>
                    请输入您的姓名：
                    <input type="text" name="txt1" value="" />
                </p>
                <p>
                    请选择您的性别：
                </p>
                <p>
                    男
                    <input type="radio" value="男" name="sex" checked="checked" />
                </p>
                <p>
                    女
                    <input type="radio" value="女" name="sex" />
                </p>
                <p>
                    请选择您的爱好：
                </p>
                <p>
                    游泳
                    <input type="checkbox" value="游泳" name="inst" checked=
"checked" />
                </p>
                <p>
                    滑冰
                    <input type="checkbox" value="滑冰" name="inst" />
                </p>
                <p>
                    散步
                    <input type="checkbox" value="散步" name="inst" />
                </p>
                <p>
                    <input type="button" value="注册" onclick="checkValid()" />
                </p>
            </form>
        </center>
    </body>
</html>
```

注意在上面的程序代码中按照 XHTML 语法规则书写的方式，页面首行为文档类型的声明，对应的标记都进行了关闭，属性声明采用的是完整规范的声明方式。文档类型声明中所

使用的这些规则存储在一个称为“文档类型声明（Document Type Declaration，DTD）”的单独文档中，并且使用这些规则验证 XHTML 文档结构的准确性。准确地说，DTD 的目的是描述 XHTML 中允许的语言和语法。

<html>标记声明中必须包括带有“xmlns=http://www.w3.org/1999/xhtml ”属性的 XML 名称空间的标识。XML 名称空间标识 XHTML 文档使用的标记范围。它用来确保一个 DTD 使用的名称不与用户定义的标记或其他 DTD 中定义的标记冲突。

由于在过渡时期 XHTML 对 JavaScript 脚本的使用有限制，所以在本例中对于 JavaScript 脚本的处理采用单独放在一个文件中的方式，对应的 JavaScript 脚本文件如下：

源程序名称：check.js

```
function checkValid()
{
      if(document.myform.txt1.value=="")
      {
            alert("请输入姓名！");
      }
      else
      {
            alert("姓名："+document.myform.txt1.value );
            for(i=0;i<document.myform.sex.length;i++)
            {
                  if(document.myform.sex(i).checked == true)
                        alert("性别："+document.myform.sex(i).value);
            }
            for(i=0;i<document.myform.inst.length;i++)
            {
                  if(document.myform.inst(i).checked == true)
                        alert("爱好："+document.myform.inst(i).value);
            }
      }
}
```

该页面的效果如图 3-9 所示。

这是一个简单的注册登记页面，并进行了数据有效性的基本检查。在 XHTML 中事件驱动的处理机制与 HTML 相同，这里不再赘述。对应的项目目录结构如图 3-10 所示。

3.2.3 CSS 的基本功能

XHTML 可以与层叠样式表（CSS）一起使用来最终实现显示的效果。本节介绍一下有关 CSS 层叠样式表的基本使用方式。

必知必会：CSS 层叠样式表的优势

CSS 层叠样式是为了实现在 Web 页面中将数据与格式分离，从而实现更加丰富的页面效果，同时也能够实现页面格式的批量动态更新。

图 3-9　注册登记页面效果

图 3-10　项目对应的结构

目前的网页排版软件中大量使用了 CSS 的语法。为了进一步丰富页面的动态效果，往往使用脚本语言与 CSS 结合，动态控制页面元素的位置、色彩等属性。因此了解 CSS 的常用属性和设置方式也是脚本编程中非常需要的。

借助 CSS 层叠样式表可以进一步美化修饰页面，例如，可以使用样式表为页面中的元素添加更多的效果，如颜色、背景、边界、边框等等。在 HTML 中许多有关格式修饰的属性只有在特定的一些标记中才能使用。而通过样式表可以使原本没有这些属性的标记实现对应的效果。例如：不能直接在 HTML 的一级标题文字标记<h1>中使用 color、face 等属性。而借助样式表则可以轻松扩充该标记的功能。

样式表可以使网页设计者的工作更加轻松。如果在一个页面中多处用到同一种字体效果，当然可以使用<font face="Arial" color="blue" size="6">，但是一旦需要对这种格式进行修改，问题随之产生：是否真的需要一个一个修改，或者改一个，然后复制粘贴若干次。答案是否定的。事实上设计人员往往选择使用样式表，设计者只需要指定参数一次，样式就会被整个网站所应用。

样式既可以定义在 HTML 文档的标记中，也可以单独放在外部的 CSS 层叠样式表文件中，这样就实现了页面的内容和页面的格式更加彻底地分离，即可以将样式文件作为外部独立的文件来使用。设计者可以将所有的样式信息放在该文件中，所有页面共享这个样式文件，而不需要在每个页面中书写一堆相同的样式代码，这样就可以使页面体积更小，从而减少页面下载的时间。

3.2.4 CSS的语法规则

在CSS的一个样式表中往往包含了若干条样式规则，将样式表添加到HTML文档中的方法很多，方法之一是可以将针对样式表的<style>标记添加到页面的<head>部分，称为嵌入式样式表。每个样式规则都有一个选择符，通常是一个HTML的标记元素，例如body、h1和该选择符所需要设置的样式属性。

样式规则的基本语法为：

```
选择符 {属性：值}
```

注意属性与其值之间用冒号隔开，而非空格。如果针对一个选择符需要定义多个样式属性，一般可以在属性之间用分号隔开。

```
选择符 {属性1：值1；属性2：值2 }
```

案例3-3 使用CSS层叠样式表丰富Web页面显示效果

下面给出一些使用CSS层叠样式表的页面文档的示例。在Eclipse中新建一个项目，项目的名称为“P33_CSS”。在该项目中新建一个HTML文档，对应的页面名称为css1.html。页面的源代码如下：

例如需要针对h1标记添加颜色和字体大小的属性：

```
源程序名称：css1.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css1.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
        <!--
        H1 { font-size: x-large; color: red}
        H2 { font-size: large; color: blue; font-family: Arial;font-style: italic}
        -->
    </style>
    </head>
    <body>
        <h1>
            第一章 Web2.0 与 Ajax
        </h1>
        <h2>
            第一节 什么是 Web2.0
        </h2>
    </body>
</html>
```

在该程序中，通过样式表对HTML文档中的一级标题和二级标题标记进行了完善，其中除修改了对应文字的颜色和大小之外，对二级标题还设置了文字的字体为Arial，文字的样式

为 italic。

需要提示的是：该程序中的<!--和-->符号，主要是为了使一些不支持 CSS 的浏览器忽略 CSS 的代码，以避免代码原样显示在浏览器中。

该页面的效果如图 3-11 所示。

图 3-11 层叠样式表基本效果

除了按照上述方法在<head>中定义层叠样式表之外，也可以在<body>中针对对应的标记直接设置 style 属性的值来实现修饰的效果，称为内联式样式表。新建一个 HTML 页面，对应的页面名称为 css2.html。

对应程序的源代码如下：

```
源程序名称：css2.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
     <head>
          <title>css2.html</title>
          <meta http-equiv="content-type" content="text/html; charset=UTF-8">
     </head>
     <body>
          <p style="text-indent:2em;background:yellow;color:red;">
               Ajax 带来网络异步交互新天地！Ajax 带来网络异步交互新天地！欢迎使用 Ajax 技术！欢迎使用 Ajax 技术！欢迎使用 Ajax 技术！
          </p>
     </body>
</html>
```

在该程序中针对<p>标记添加了 style 属性。设置段落的首行缩进 2 个字符。背景色为黄色，文字颜色为红色。

该程序实现的页面效果如图 3-12 所示。

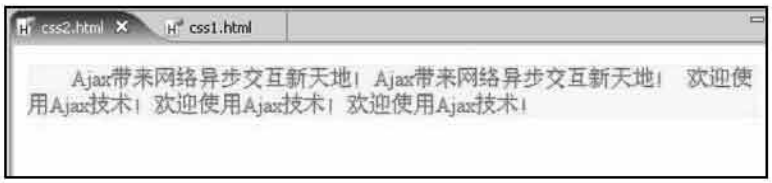


图 3-12 CSS 在 body 中的实现效果

除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。需要用到该样式表的页面可以在本页面的head中使用 link 标记引用样式表文件，称为外挂式样式表。

需要强调的是在创建层叠样式表文件时，文件内容只能包含有关的样式规则，而不允许有其他的内容，示范文件如下所示：

源程序名称：mystyle.css

```
h1 {
    color: green;
    font-family: 黑体;
    text-align: center
}

p {
    text-indent: 2em;
    background: yellow;
    font-family: 宋体
}
```

其中，针对一级标题设置了文字颜色、字体及对齐方式。针对段落设置了首行缩进、背景色及字体。在准备好层叠样式表文件后，就可以利用 link 标记引用外部样式表文件。对应的程序代码如下：

源程序名称：css3.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css3.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <link rel="stylesheet" type="text/css" href="mystyle.css">
    </head>

    <body>
        <h1>
            引入 CSS 外部文件的方式
        </h1>
        <p>
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，
            为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往
            会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件
            中定义。
            需要用到该样式表的页面可以在本页面的 head 中使用 link 标记
            引用样式表文件。
        </p>
    </body>
</html>
```

在该程序中的head部分，使用了 link 标记，该标记用于引用外部的的文件。其中 rel 属

性设定引用文件类型为样式表“stylesheet”，href 属性规定了外部文件的位置和名称，使用的是相对路径，样式表类型为“text/css”。

对应的页面效果如图 3-13 所示。

图 3-13　使用 link 标记引用外部样式表文件

此外，也可以在 head 部分的 style 标记中使用 import 导入外部的样式表文件，例如：

源程序名称：css4.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css4.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
         <!--
             @import url(mystyle.css);
         -->
         </style>
    </head>
    <body>
         <h1>
             引入 CSS 外部文件的方式
         </h1>
         <p>
             除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
          </p>
      </body>
</html>
```

该程序所实现的页面效果与图 3-13 所示相同。

必知必会：CSS 层叠样式表的优先级问题

在掌握了使用 CSS 层叠样式表的基本方法之后，需要再关注一下有关优先级的问题。如果在页面中采用了多种样式表的定义方式，并且使用不同方式针对同一个 HTML 标记进行了

重复定义，且定义的属性中有冲突，例如，在外部样式表文件中对 h1 标记进行了颜色的定义，而在本页面的 style 样式中既导入了外部的样式文件，又对该标记进行了颜色的重新定义，还有可能在页面某处的 h1 标记中，使用 style 属性进行了不同颜色的定义。那么到底谁的设置会优先生效呢？

```
源程序名称：css5.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css5.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <link rel="stylesheet" href="mystyle.css" type="text/css">
        <style type="text/css">
            h1 {color:red; font-size:30}
            p  {background:green;color:gold}
        </style>
    </head>
    <body>
        <h1 style="color:brown">
            引入 CSS 外部文件的方式
        </h1>
        <p>
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
        </p>
    </body>
</html>
```

在该程序中一级标题文字的颜色分别在外部样式表文件中定义为绿色、head 的 style 标记中定义为红色，<h1>标记中定义为棕色，最终页面呈现出来的为棕色。而段落标记<p>标记在外部样式表文件中定义背景色为黄色，head 的 style 标记中定义为绿色，最终呈现出来的是绿色。

对应的页面效果如图 3-14 所示。

图 3-14 样式表的优先级

由此可见，当样式表定义有冲突时，采用就近原则，即与被修饰元素最近的优先生效。如果彼此之间不冲突，将共同作用于对应元素。

在 CSS 层叠样式表中，除了可以将 HTML 现有的标记作为 CSS 的选择符，实现格式设置扩充之外，也可以使用类选择符或者 ID 选择符进行样式规则的设定。

考虑到同一个标记有可能需要在页面的不同位置呈现出不一样的效果，CSS 中允许针对同一个标记设置不同的类，从而实现不一样的样式。基本语法是，在 HTML 标记后添加.类选择符名，例如：

```
p.basic { text-indent:2em; background:yellow; font-family:宋体}
p.main { text-indent:2em; background:blue; font-family:宋体; font-style:italic; color:white}
```

在上面的定义中，对于段落标记根据需要可以有两种不同的格式。例如，针对基本段落设置的格式是首行缩进两个字符，背景色为黄色，字体为宋体。而针对需要重点强调的主要段落，则将背景色设置为蓝色，文字颜色为白色，并且以斜体显示。需要提醒的是，在给类命名时尽量能够反映出该类的功能。

在页面中需要具体使用时，可以在对应的标记中使用 class（类）属性规定采用的到底是哪一个类，示例程序如下所示：

```
源程序名称：css6.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css6.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
            p.basic { text-indent:2em; background:yellow; font-family:宋体}
            p.main { text-indent:2em; background:blue; font-family:宋体; font-style:italic; color:white}
        </style>
    </head>
    <body>
        <p class="basic">
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。 需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
        </p>
        <p class="main">
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
        </p>
    </body>
</html>
```

页面的效果如图 3-15 所示。

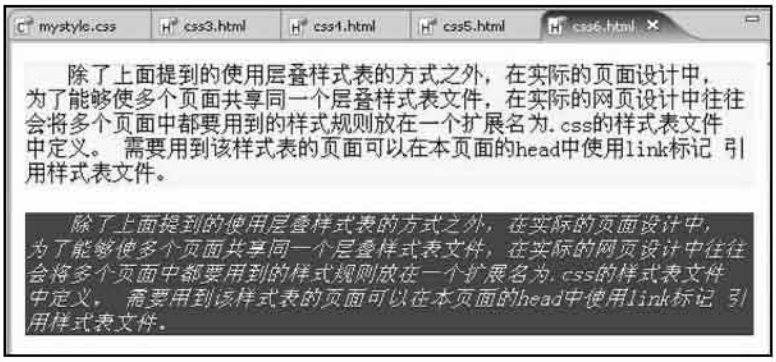


图 3-15 针对同一标记的类选择符

此外，也可以定义与任何 HTML 标记无关的独立的类选择符。这样，如果有多个标记采用同样的格式，就可以借助 class 属性使用同一个类选择符中定义的属性。

源程序名称：css7.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css7.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
            .basic { text-indent:2em; background:yellow; font-family:宋体;color:yellow}
        </style>
    </head>
    <body>
        <h1 class="basic" style="font-family:黑体">
            引入 CSS 外部文件的方式
        </h1>
        <p class="basic" style="font-size:20">
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义.需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
        </p>
    </body>
</html>
```

在该页面中<h1>标记和<p>标记都使用了 basic 类选择符，并在此基础之上利用标记本身的 style 属性，对格式进行了补充或修改。页面效果如图 3-16 所示。

除了可以使用类选择符之外，也可以使用 ID 选择符，在页面中使用频率并不太高，其基本格式是：

```
<style type="text/css">
    #jenny { text-indent: 2em }
</style>
```

如果在 HTML 标记中需要使用该 ID 选择符，可以使用 id 属性，例如：

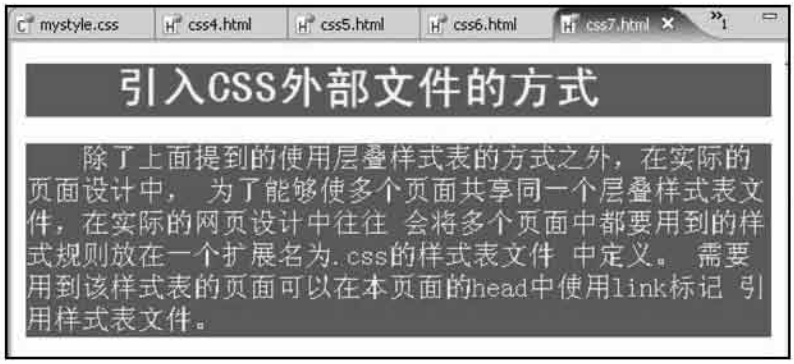


图 3-16　不同标记使用同一类选择符

源程序名称：css8.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css8.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
            #jenny { text-indent:2em; background:yellow; font-family:宋体}
        </style>

    </head>

    <body>
        <p id="jenny">
            除了上面提到的使用层叠样式表的方式之外，在实际的页面设计中，为了能够使多个页面共享同一个层叠样式表文件，在实际的网页设计中往往会将多个页面中都要用到的样式规则放在一个扩展名为.css 的样式表文件中定义。需要用到该样式表的页面可以在本页面的 head 中使用 link 标记引用样式表文件。
        </p>
    </body>
</html>
```

对应的页面效果如图 3-17 所示。

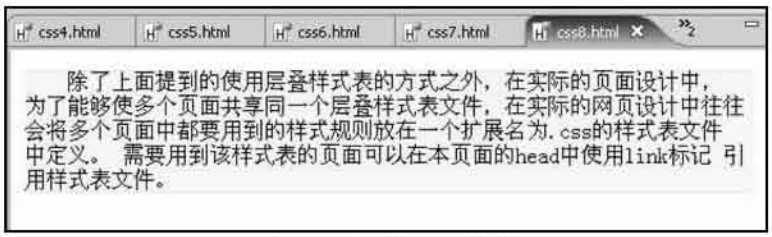


图 3-17　使用 ID 选择符

在 CSS 层叠样式表中还提供了一些特殊的类及与之相关的元素，其所应用的场合和所起

的作用都已经事先进行了约定。

例如，在页面中经常看到的文字超级链接效果。在超级链接点击前和点击后，往往颜色不同，很多页面还可以在鼠标悬停在超链接文字上方时，改变文字的显示效果。这些效果实现的方法之一就是利用针对超链接<a>标记的几个特殊的伪类，即锚伪类，分别设定在不同状态时的效果。

需要注意的是，在样式规则中使用伪类时，不能使用圆点，而应该在伪元素和伪类之间使用冒号。

源程序名称：css9.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css9.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
            a:link {font-size: 10pt; color:red; text-decoration: none}
            a:visited{font-size: 9pt; color:green; text-decoration: none}
            a:hover{font-size: 15pt; color:brown; text-decoration: underline}
        </style>
    </head>
    <body>
        <a href="#">有关 Ajax 的最新动态</a>
    </body>
</html>
```

在该程序中，对锚元素，即<a>标记的几个特殊的类进行了重新定义，其中 link 是未访问的超链接文字效果，默认如果对文字添加了超级链接，文字颜色往往会自动设置为蓝色加下划线，而在该页面中改变设置超级链接的文字颜色为红色，并利用 CSS 的 text-decoration 属性去掉了下划线。

此外 visited 是访问过的超链接文字效果，文字尺寸减小，文字颜色为绿色。hover 设置的是当鼠标悬停在超链接文字上方时显示的效果：加大文字尺寸、文字颜色为棕色，文字添加了下划线。

在预览页面效果时，鼠标指针悬停在文字上方时的效果如图 3-18 所示。

图 3-18 锚伪类的使用

在传统的 HTML 页面中，如果需要精确定位元素的位置，往往以表格搭建页面的布局，然后将元素放在表格的单元格中以实现目标的定位。但有时需要结合脚本语言动态的控制元

素在页面中出现的位置，此时往往利用 CSS 控制该元素的位置，脚本语言也可以通过改变 CSS 中与位置相关的属性值来达到目的，下面的示例介绍具体的实现方式。代码如下：

源程序名称：css10.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>css10.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
        <!--
          div{
                position:absolute;
                top:70;
                left:200;
            }
        -->
        </style>
    </head>
    <body>
        <div>
            <img src="smile.gif" alt="">
        </div>
    </body>
</html>
```

在该程序中，用到了块标记 ，该标记可以将其内部的图片及文字作为一个整体，即分层进行控制，往往可以实现页面效果的局部改变。本例中利用样式表定义了该标记的位置，具体页面顶端 70 像素、页面左端 200 像素。因此在块标记内部的图片显示在了页面中指定的位置，效果如图 3-19 所示。

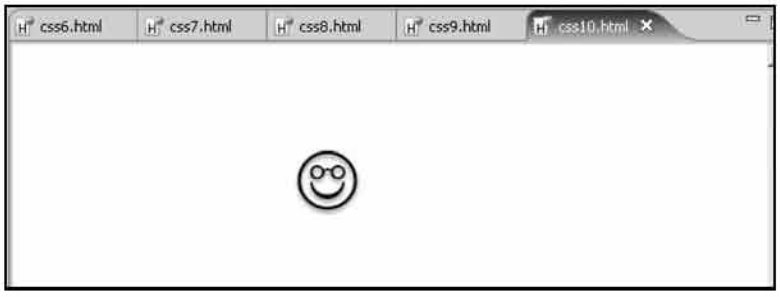


图 3-19　CSS 精确控制元素的位置

在 CSS 中还可以依据需要控制元素的显示和隐藏，与脚本语言相结合就能够动态控制目标元素的显示和隐藏，示例程序如下：

源程序名称：css11.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```
<head>
     <title>css11.html</title>
     <meta http-equiv="content-type" content="text/html; charset=UTF-8">
     <style type="text/css">
     <!--
       div.show{
               position:absolute;
               top:10;
               left:200;
               visibility:visible;
          }
       div.hide{
               position:absolute;
               top:120;
               left:200;
               visibility:hidden;
          }
     -->
     </style>

</head>

<body>
     <div class="hide">
          <img src="smile.gif" alt="">
     </div>
     <div class="show">
          <img src="cry.gif" alt="">
     </div>
</body>
</html>
```

在该程序中页面中应该有两张图片，一张是笑脸，一张是哭脸。但是利用样式表的visibility属性将页面下方的一张图片设置为隐藏，因此在页面中只能看到上方的一张，页面效果如图3-20所示。

图3-20 CSS控制元素的显示和隐藏

在页面中还可以使用CSS中的cursor属性改变鼠标指针的形状。例如：

源程序名称：css12.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```
<head>
    <title>css12.html</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <style type="text/css">
    <!--
        .hand{cursor:hand;}                    /*手形*/
        .crosshair{cursor:crosshair;}          /*精确定位“+”字*/
        .move{cursor:move;}                    /*移动*/
        .e-resize{cursor:e-resize;}            /*箭头朝右方*/
        .ne-resize{cursor:ne-resize;}          /*箭头朝右上方*/
        .nw-resize{cursor:nw-resize;}          /*箭头朝左上方*/
        .n-resize{cursor:n-resize;}            /*箭头朝上方*/
        .se-resize{cursor:se-resize;}          /*箭头朝左下方*/
        .s-resize{cursor:s-resize;}            /*箭头朝下方*/
        .w-resize{cursor:w-resize;}            /*箭头朝左方*/
        .text{cursor:text;}                    /*“I”字型*/
        .wait{cursor:wait;}                    /*等待*/
        .help{cursor:help;}                    /*帮助*/
    -->
    </style>

</head>

<body>
    <p class="hand">
        Give me a hand!(手形)
    </p>
    <p class="crosshair">
        Give me a crosshair(精确定位)!
    </p>
    <p class="move">
        Give me a Move(移动)!
    </p>
    <p class="e-resize">
        Give me a e-resize(箭头方向朝右)!
    </p>
    <p class="ne-resize">
        Give me a ne-resize(箭头方向朝右上)!
    </p>
    <p class="nw-resize">
        Give me a nw-resize(箭头方向朝左上)!
    </p>
    <p class="n-resize">
        Give me a n-resize(上)!
    </p>
    <p class="se-resize">
        Give me a se-resize(左下)!
    </p>
    <p class="s-resize">
```

```
        Give me a s-resize(下)!
    </p>
    <p class="w-resize">
        Give me a w-resize(左)!
    </p>
    <p class="text">
        Give me a text(文本)!
    </p>
    <p class="wait">
        Give me a wait(等待)!
    </p>
    <p class="help">
        Give me a help(帮助)!
    </p>

</body>
</html>
```

在该程序中列出了常见的鼠标指针的形状，当鼠标悬停在不同位置时，其指针形状将发生相应的改变，页面效果如图 3-21 所示。

图 3-21 CSS 控制鼠标指针的形状

除了本书中所涉及到的 CSS 样式表的基本功能之外，还可以借助 CSS 实现对图片、文字的修饰，由于对程序员来讲，不是非常常用，本书中就不再涉及。

下面给出几个结合 CSS 及 JavaScript 脚本实现的动态交互的效果。

源程序名称：setHome.html

```
<html>
<script language="JavaScript">
    function set()
    {
        sethome.style.behavior="url(#default#homepage)";
```

```
            sethome.setHomePage("http://www.getjob.com.cn");
        }
    </script>
    <body >
        <span id=sethome onclick="set()" style="CURSOR: hand"><br>将本站设置为首页
</span>
    </body>
    </html>
```

在上面的程序中，当用户单击将本页设置为首页时，将弹出询问是否将本页设置为浏览器默认首页的对话框，页面效果如图 3-22 所示。注意在该程序中使用层叠样式表修改了鼠标指针的形状，同时也使用了浏览器对象调用了默认主页设置的方法。

图 3-22　设置默认主页

这里还可以进一步丰富鼠标的显示效果，例如，onmousemove 事件是鼠标在指定元素范围之内移动时发生的，我们可以利用这一点实现图片随着鼠标指针而移动。

源程序名称：movPic.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>movPic.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <style type="text/css">
            <!--
            #topdown{
                position:absolute;
                left:12;
                top:222;
                }
            -->
        </style>
        <script language="JavaScript" type="text/javascript">
            function move(x, y)
            {
                topdown.style.left = x;
```

```
                topdown.style.top = y;
            }
        </script>

    </head>

    <body onmousemove="move(event.x, event.y)">
        <div id="topdown">
            <img src="smile.gif" alt="">
        </div>
    </body>
</html>
```

在该程序中首先使用层标记 将图片单独放在一层中，然后使用 CSS 层叠样式表定义了该图片的位置，再与 onmousemove 事件结合，追踪鼠标的位置（event.x 和 event.y），并随时用鼠标的位置修改对应图片的位置，因此该程序实现的效果是页面中显示出一张始终跟着鼠标指针移动的笑脸。页面效果如图 3-23 所示。

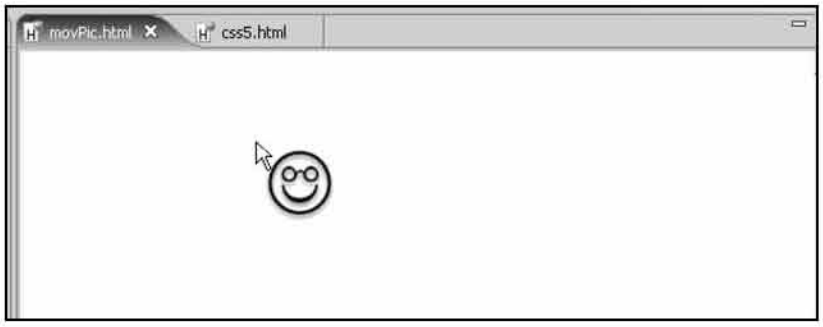


图 3-23 随着鼠标指针移动的图片

3.3 DOM 简介

DOM 的全称是文档对象模型（即 Document Object Model），它在本质上是一种文档平台。文档对象模型（DOM）是表示文档（比如 HTML 和 XML）和访问、操作构成文档的各种元素的应用程序接口（API）。支持 Javascript 的所有浏览器都支持 DOM。DOM 实际上是一个能够让程序和脚本动态访问和更新文档内容、结构和样式的一种语言平台。

本书中所涉及的 DOM，是指 W3C 定义的标准的文档对象模型，它以树形结构表示 HTML 和 XML 文档，定义了遍历这个树和检查、修改树的节点的方法和属性。在实际应用中，DOM 一般被分为不同的部分（核心，XML 和 HTML），它们分别对应着不同的版本（DOM 1/2/3）。

- Core DOM：定义了任意结构文档的标准对象集合；
- XML DOM：定义了针对 XML 文档的标准对象集合；
- HTML DOM：定义了针对 HTML 文档的标准对象集合。

为了让读者逐步接受有关 DOM 与 HTML 以及 XML 之间相关的操作，本节先简单介绍一下 DOM 的基本功能和意义，在本书第五章中将详细介绍结合 HTML 文档以及 XML 文档相关的具体操作方法。

HTML DOM 是针对 HTML 的文档对象模型，它定义了针对 HTML 的标准对象集合以及访问和操作 HTML 文档的方法。在 HTML DOM 中，HTML 跟 XML 一样是一种树形结构的文档，可以作为一个对象的集合来使用，这些对象有属性和方法，同时也可以对事件做出反应，调用对应的函数进行处理。 <html>是根（root）节点，<head>、<body>是<html>的子（children）节点，互相之间是兄弟（sibling）节点；<body>下面还包含有子节点<table>、、<p>等。

在 HTML 文档的树形结构中主要包含表示元素、标记的节点和表示文本串的节点。对应的 HTML 的树形结构图 3-24 如下所示。

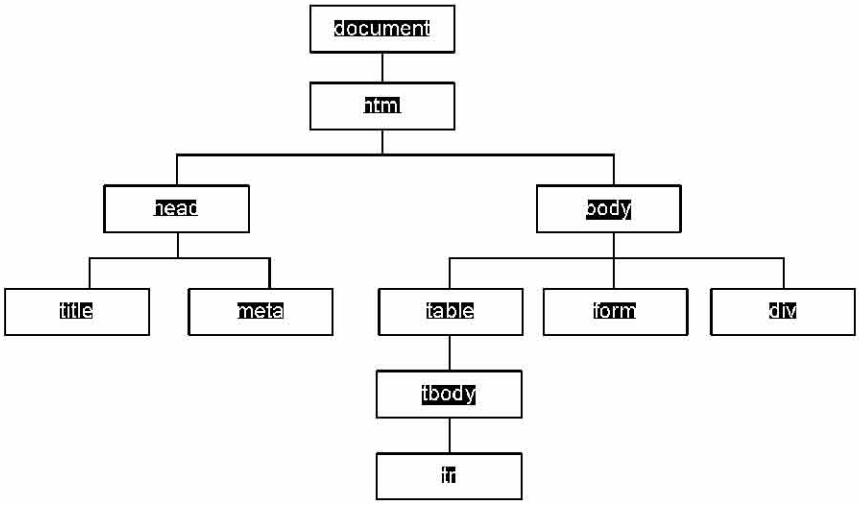


图 3-24　HTML 的树形结构图

常用的 HTML DOM 中对应对象的功能如下所示。

- window 对象：代表一个浏览器窗口的对象；
- navigator：可提供当前所使用浏览器的版本号，运行的平台以及浏览器使用的语言等信息；
- document 对象：含有当前网页的各种特性，例如标题、背景以及使用的语言等等；
- location 对象：含有当前网页的 URL 地址；
- history 对象：含有以前访问过的网页的 URL 地址；
- form 对象：表单对象代表一个 HTML 表单。HTML 中一有<form>标签，一个表单对象就被建立了；

- anchors 对象：锚对象代表一个 HTML 超级链接。HTML 文档中有<a>标签，一个锚对象就随之建立了；
- links 对象：是从属于浏览器对象 document 的一个数组，为处理超级连接提供属性和方法，每一个超级链接就是这个数组中的一个元素。

在浏览器中定位 HTML 文档中的某一个对象就是依据 DOM 提供的方式，即一种树型遍历、逐级定位的方式。例如，在前面所举的例子中，我们注意到如果要取出页面中某一个文本框的值，其基本格式为：

documen.表单名.文本框名.value

其中 document 一般可以省略不写，这种表示形式就像我们在资源管理器中描述一个文件的位置和名称时，会从对应盘符逐步定位到一级子文件夹、二级子文件夹直至定位到文件所在的文件夹，最终依据文件的文件名获得文件的内容。浏览器对象定位的层次结构与文件系统的资源管理是类似的。下面再给出一些示例。

案例 3-4　使用 DOM 提供的方式访问 HTML 文档中的对象

在 Eclipse 中新建一个项目，项目的名称为“P34_DOM”。在该项目中新建一个 HTML 文档，对应的页面名称为 dom1.html。页面的源代码如下：

源程序名称：dom1.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>dom1.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <script language="JavaScript" type="text/javascript">
            function check()
            {
                if(document.myform.txt1.value=="")
                    alert("请输入姓名！");
                else
                    alert("您的姓名是："+document.myform.txt1.value);
            }
        </script>

    </head>

    <body>
        <form name="myform" action="">
            <p>
                请输入您的姓名：
                <input type="text" name="txt1" value="">
            </p>
            <p>
```

```
                    <input type="button" value="确定" name="ok" onclick="check()">
                </p>
            </form>

        </body>
    </html>
```

注意该程序中定位文本框的方式。该程序可以在用户单击确定按钮后，检查用户是否在文本框中输入了姓名，如果用户未输入对应的内容，则提示用户输入。

页面效果如图 3-25 所示。

图 3-25　使用 DOM 定位

另外一个非常常用的对象是 window 对象，该对象提供的方法如下所示。

- open(URL,windowName,parameterList)：open 方法创建一个新的浏览器窗口，并在新窗口中载入一个指定的 URL 地址；
- close()：close 方法关闭一个浏览器窗口；
- alert()：弹出一个消息框；
- confirm()：弹出一个确认框；
- promt()：弹出一个提示框；
- setTimeout(expression,time)：定时设置，在一定的时间后自动执行 expression 的代码，使用 time 设置时间，单位是毫秒；
- clearTimeout(timer)：取消利用 setTimeout 的定时设置；
- setIntervel(expression,time)：设定一个时间间隔，可以定时反复的自动执行 expression 描述的代码，使用 time 设置时间，单位是毫秒。

例如：我们在页面中经常看到的自动弹出的广告窗口，往往是使用 window 对象的 open() 方法，打开窗口，并显示指定的页面。示例程序如下：

源程序名称：dom11.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
    <head>
        <title>dom1.html</title>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <script language="JavaScript" type="text/javascript">
            function advertis()
            {
              window.open("ad.htm","my","toolbar=no, left=150, top=200, menubar=no,
width=250,height=200");
            }
        </script>

    </head>

    <body onload = "advertis()">
        <form name="myform" action="">
            <p>
                请输入您的姓名：
                <input type="text" name="txt1" value="">
            </p>
            <p>
                <input type="button" value="确定" name="ok" onclick="check()">
            </p>
        </form>

    </body>
</html>
```

在上面的例子中 window.open("ad.htm","my","toolbar=no, left=150, top=200, menubar=no, width=250,height=200")；语句中指定了在该窗口中需要显示的页面，也可以是浏览器能够显示的其他类型的文件，例如：图片文件。上例中的页面在打开时将自动弹出小窗口，并在该窗口中显示事先制作完成的广告页面。页面效果如图 3-26 所示。

图 3-26 广告窗口的自动弹出

在下面的程序中，我们将完成页面中常见的飘浮广告的功能，读者同样可以关注一下在使用文档中对象时的访问方式。同时，在该程序中也综合运用了 CSS 层叠样式表，脚本编程的知识。代码如下：

源程序名称：dom2.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
     <head>
          <title>dom2.html</title>
          <meta http-equiv="content-type" content="text/html; charset=UTF-8">
          <script language="javascript" type="text/javascript">
               var x1=10;
               var y1=-10;
               //控制图片移动的函数
               function fly(){
                      flypic.style.top = parseInt(flypic.style.top) - y1;
                      flypic.style.left = parseInt(flypic.style.left) - x1;
                      if (parseInt(flypic.style.top) < 0){
                                    y1=-y1;
                      }
                      if (parseInt(flypic.style.top)>300){
                                    y1=-y1;
                      }
                      if (parseInt(flypic.style.left) < 0){
                                  x1=-x1;
                      }
                        if (parseInt(flypic.style.left) >630){
                                  x1=-x1;
                      }
                     winow.setTimeout("fly()", 300);
               }
          </script>
     </head>
     <body onload="fly()">
          <div id="flypic" style="position:absolute; top:20; left:40">
               <img src="smile.gif" alt="">
          </div>
     </body>
</html>
```

在实现了上述的程序功能后，我们也同样可以实现页面中另一种随滚动条滚动的常见的飘浮广告的功能。代码如下：

源程序名称：dom3.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>dom3.html</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <script language="javascript">
```

```
            //控制图片移动的函数
            function moveIt() {

                    truck.style.top=  document.body.scrollTop+parseInt(document.body.
clientHeight)- 90;

                    window.setTimeout('moveIt()', 1000);
            }
        </script>

    </head>

    <body BGCOLOR="#ffffff" TEXT="#000000" link="#0000ff" Vlink="#660099" onload="moveIt()">
  <div id="truck" style="position:relative; top:30px; left:500px; width:40px; height:16px;">
  <img  src="smile.gif">
  </div>
  </body>
  </html>
```

上面的程序中利用 document 对象获取到了垂直滚动条的位置，从而动态控制广告图片的位置，为了查看效果，读者可以在该页面中添加更多的内容，使页面长度增加，从而出现滚动条。

在后续章节中介绍 XML DOM 的使用方法，同时介绍更加深入的结合 DOM 操作 HTML 文档的方法。

3.4 XML 和 XSTL

本节将介绍在未来互联网中的数据传递及交换的标准格式 XML，以及与之相关的 XSTL。

3.4.1 XML 可扩展标记语言

XML 的全称为可扩展标记语言（即 Extensible Markup Language），它是 W3C 定义的一种标准。那么为什么会出现 XML？其优势到底在哪里呢？

必知必会：XML 与 HTML 相比的优势

相信读者对 HTML 都非常熟悉，HTML 是一种标记语言，同时，HTML 里面有很多标签，类似<head>、<table>等，都是在 HTML 4.0 里规定的，但是随着互联网技术的日益发展，HTML 本身存在的很多缺陷也日益显现出来，例如，数据内容和页面的排版、显示控制等混在一起，导致页面过度膨胀。数据库存储的信息，经过动态编程语言的解释执行被转换为 HTML 文档格式后，信息原有的格式已经变得面目全非，这给前台页面的设计者及后台系统的程序开发人员带来许多麻烦，这些都使得原有的 HTML 阻碍了互联网的快速发展。

XML 可以将数据的内容与控制数据显示效果的格式清晰的分开，同时 XML 之所以称为可扩展的标记语言，是因为在 XML 中允许开发人员自行创建所需要的标签，正是因为 XML

的这一特点，我们可以借助自定义标记的方式分层次清晰的表达出数据之间的关系，所以有人也将 XML 文档称为“流动的数据岛”文件。

在本书第一个有关 Ajax 的例子中，我们就是采用了 XML 格式的数据进行响应，然后在浏览器客户端方便地进行解析，获取指定位置指定层次的数据，这种方式便于数据的传递，也便于在传递过程中很好地保存数据与数据之间的层次关系。例如：在本书第二章的案例中对应 XML 格式的数据如下所示。

如果身份验证通过，则返回的数据格式为：

```
<response>
     <res 热烈的欢迎您</res>
</response>
```

如果身份验证未通过，则返回的数据格式为：

```
<response>
     <res>对不起,登录失败！</res>
</response>
```

注意，在对 XML 文件进行数据解析时，对应的结点名称，即对应标记的名称就显得至关重要。例如：第二章案例 2-2 中最终在 Ajax 中获取返回的响应数据的语句如下：

```
// 处理返回信息函数
    function processResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
          if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
               var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].first
Child.data;
               window.alert(res);
               } else { //页面不正常
                    window.alert("您所请求的页面有异常。");
               }
          }
    }
```

下面通过一个简单的案例让读者对标准 XML 文件的格式有一个初步的了解。

案例 3-5　使用 XML 及 XSTL 文档存储及显示数据

新建一个项目，项目名称为“P35_XML”，在该项目中新建一个 XML 文件，可以选择如图 3-27 所示的命令选择依据模版来创建一个新文件。

此时系统自动生成的代码如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-body>

</xml-body>
```

可以看到最基本的 XML 文档的格式，注意在页面中的第一行声明了 XML 对应版本，以及所使用的编码对应的字符集。

图 3-27 依据模版来新建一个 XML 文件

我们在对应的文档中录入所需要存储的数据信息，可以看到在该页面中方便地反映了数据之间的对应关系以及层次结构。对应代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<classmates>
     <student>
          <sid>1</sid>
          <sname>张桂元</sname>
          <gre>1700</gre>
          <toefl>630</toefl>
          <tse>120</tse>
     </student>
     <student>
          <sid>2</sid>
          <sname>贾燕枫</sname>
          <gre>1800</gre>
          <toefl>599</toefl>
          <tse>110</tse>
     </student>
     <student>
          <sid>3</sid>
          <sname>张宇翔</sname>
          <gre>1880</gre>
          <toefl>700</toefl>
          <tse>130</tse>
     </student>
     <student>
          <sid>4</sid>
          <sname>张大牛</sname>
```

```
            <gre>1600</gre>
            <toefl>660</toefl>
            <tse>110</tse>
        </student>
    </classmates>
```

单击 Eclipse 中编辑窗格下方的 Grid 选项卡，也可以进入网格状态进行 XML 文档的编辑和修改。对应的效果如图 3-28 所示。

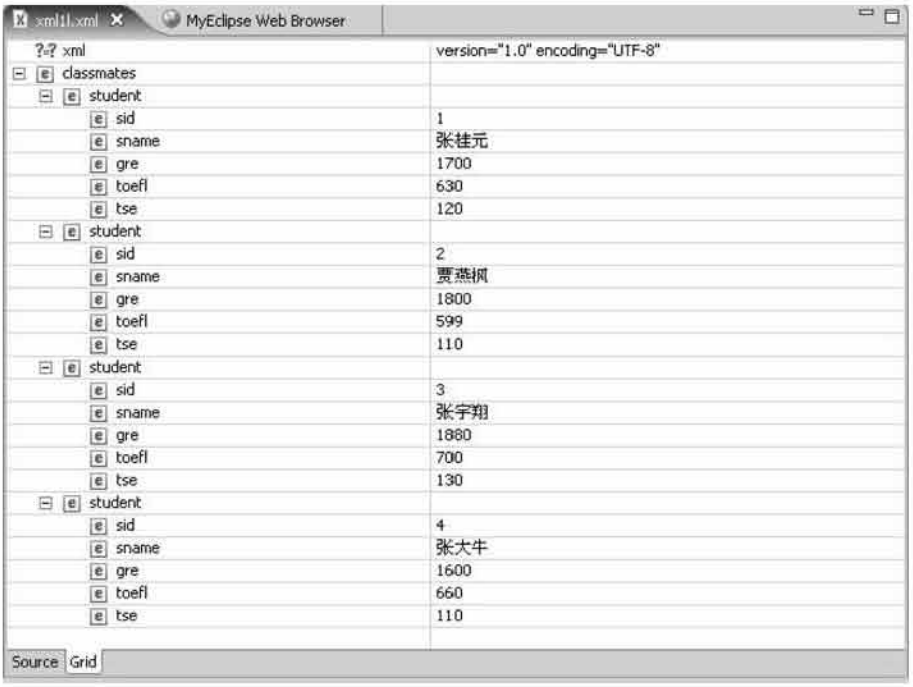


图 3-28　网格编辑状态

如果在浏览器窗口中直接打开该页面，对应的显示效果如图 3-29 所示。

从图 3-29 中可以看到，在浏览器中清晰地反映出了对应数据的树型结构。因此可以将 XML 文档中所存储的数据看作一个可以操作的树，开发人员可以根据需要对这棵树进行遍历、修改以达到对数据访问的目的。

从上面的例子中可以了解到：由于在 XML 中可以进行自定义标记，因此 XML 的优势之一是它允许各个组织、个人建立适合自己需要的标记符号。这一特征使得 XML 可以在电子商务、政府文档、司法、出版、保险机构、厂商和中介组织信息交换等领域中提供各具特色的独立解决方案。XML 的最大优点就在于它的数据存储格式不受显示格式的制约。

XML 是一种程序之间交换原始数据的简单而标准的方法。它的成功并不在于它容易被人们书写和阅读，更重要的是，它从根本上解决了不同应用系统之间的通用信息交换方式。下面所示即为在本书第 2 章的案例中所使用的一个典型 Web 应用配置文件的格式。

图 3-29 浏览器中显示 XML 文档的效果

```
<?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">
<!—设置对应 Servlet 对应的类型-->
   <servlet>
        <servlet-name>ms1</servlet-name>
        <servlet-class>classmate.LoginAction</servlet-class>
   </servlet>
<!—设置请求对应相应的 Servlet 名称-->
  <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/login</url-pattern>
  </servlet-mapping>
<!—设置默认欢迎页面，这里为请求名称 -->
 <welcome-file-liat>
   <welcome-file>login.jsp</welcome-file>
 </welcome-file-list>
```

```
</web-app>
```

可以看到在对应的配置文件中，根结点为<web-app>，其中包含了三个子结点，分别是<servlet>、<servlet-mapping>及<welcome-file-list>，其中每个子结点又包括了各自的子节点。显然该文件占用的是纯文本文件的存储空间，但是可以清晰地表达出数据的层次结构。

3.4.2 XSTL 可扩展的样式表转换语言

已经了解到 XML 文件存储数据时的优势和特点，那么如何对这些数据进行显示等操作呢？有关 XML 详细的解析方式，将在本书的后续章节进行介绍，本章首先关注如何方便地将 XML 文档中存储的数据以 HTML 的格式在页面中进行显示？这里既可以使用 CSS 来实现输出效果比较固定的显示控制，也可以使用 XSTL 完成更加灵活的显示控制。

下面先介绍一下如何使用 CSS 层叠样式表实现 XML 文档数据的显示。例如，我们针对 xml.xml 文件中的数据，配合 CSS 层叠样式表实现对应数据信息的显示，在项目“P35_XML”中新建一个 CSS 文件，对应的样式文件中样式的设置为：

```
sid
{
color:blue;
font-size:36;
text-indent:2em;
display: block;
}
sname
{
color:red;
font-size:24;
text-indent:2em;
display: block;
}
gre
{
color:red;
font-size:24;
text-indent:2em;
display: block;
}
toefl
{
color:red;
font-size:24;
text-indent:2em;
display: block;
}
tse
{
```

```
color:red;
font-size:24;
text-indent:2em;
display: block;
}
```

然后在原有的 xml1.xml 文件中添加新的一行如下所示，该行用于指定所需使用的样式表文件的类型、位置以及名称。

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="xml1.css"?>
<classmates>
     <student>
          <sid>1</sid>
          <sname>张桂元</sname>
          <gre>1700</gre>
          <toefl>630</toefl>
          ……
```

此时，在浏览器中再次直接打开 xml1.xml 文件的效果如图 3-30 所示。可以看到树型结构中的数据已经按照 CSS 层叠样式表中的数据进行了相应的显示。

图 3-30 浏览器中配合 CSS 显示 XML 文档的效果

虽然 CSS 能够很好的控制输出的样式，比如色彩、字体、大小等，但是它也存在着严重的局限性，例如：不能重新排序文档中的元素；不能判断和控制哪个元素被显示，哪个不被显示；不能统计计算元素中的数据；因此 CSS 只适合用于输出比较固定的最终文档，对于输出要求比较灵活的需求往往不能够做到。

必知必会：CSS 与 XSTL 各自的优势

CSS 的优点是相对简洁，消耗系统资源少；但是如果输出需求比较灵活，就应该采用 XSTL 来进行显示控制。需要提醒读者的是 XSLT 虽然功能强大，但因为要重新索引 XML 结构树，所以消耗内存比较多。因此，在实际使用中常常会将它们结合起来使用，比如在服务器端用 XSLT 处理文档，在客户端用 CSS 来控制显示。

必知必会：XSL 与 XSTL 的关系

下面介绍一下如何使用 XSTL 实现对 XML 文档中的数据进行显示控制。首先先明确一下 XSL 的概念。XSL(eXtensible Stylesheet Language)和我们书中介绍的 XSLT 从狭义上理解是一样的,XSL 实际上包括了 XSLT(结构转换)和 XSL-FO(formatting objects)(格式化输出)两种分支语言，其中 XSL-FO 的作用就类似前面介绍的 CSS 在 HTML 中的作用。而本书中重点讨论的是第一步的转换过程，也就是 XSLT。按照 W3C 的标准，XSLT 相对 XSL 的说法更严格些，因此在本书中我们统一使用 XSLT。

XSTL 的全称是可扩展的样式表转换语言（即 Extensible Stylesheet Transformation Language）。它是一种用来转换 XML 文档结构的语言。为了使 XML 文档中存储的数据便于阅读理解，往往需要将信息显示出来或者打印出来，例如将存储的数据转换成一个 HTML 文件，一个 PDF 文件，甚至是一段声音。同样，为了使数据适合不同的应用程序，就必须提供能够将一种数据格式转换为另一种数据格式的方法，比如需求格式可能是一个文本文件，一个 SQL 语句，一个 HTTP 信息，一定顺序的数据调用等。而 XSLT 就是用来实现这种转换功能的语言。XSLT 最主要的功能就是将 XML 转换为 HTML。

必知必会：Xpath 的定义

通过前面的学习，已经了解到 XML 是一个完整的树型结构的文档。在转换 XML 文档时往往需要处理其中的一部分(节点)数据，那么如何查找和定位 XML 文档中的信息呢？XPath 就是一种专门用来在 XML 文档中查找信息的语言。XPath 隶属 XSTL，因此我们通常会将 XSTL 语法和 XPath 语法结合在一起说。

用一种比较好理解的解释：如果将 XML 文档看作一个数据库，XPath 就是 SQL 查询语言；如果将 XML 文档看成 DOS 目录结构，XPath 就是 cd,dir 等目录操作命令的集合。

如果要从 XML 里根据需要提取相关的数据进行显示，就需要用到 XSTL 提供的模式化查询语言。所谓模式化查询语言，就是通过相关的模式匹配规则表达式从 XML 里提取数据的特定语句。常用的模式化查询语言一般可以分为三种情况，包括选择模式、测试模式以及匹配模式，下面分别进行介绍。

1. 选择模式:

在该模式中包括<xsl:for-each>、<xsl:value-of>和 <xsl:apply-templates>等语句。选择模式语句可以实现将数据从 XML 中提取出来，它提供的是一种简单获得数据的方法，这几个标记都有一个 select 属性，用于获取 XML 中指定结点名中对应的数据。下面分别介绍一下对应的几个模式语句。

<xsl:for-each>类似于程序代码中的循环语句，例如在 XML 中有这样的数据:

```
<authors>
   <name>张桂元</name>
   <name>贾燕枫</name>
   <name>张宇翔</name>
   <name>张大牛</name>
</authors>
```

如果需要读取对应的作者名字，就可以使用<xsl:for-each>，对应的代码片段如下：

```
<xsl:for-each select="author/name">
……
</xsl:for-each>
```

其中的 select 属性可以依据名称选定 XML 中特定唯一的标记，也可以选择某一类相同的标记，一般称之为结点集。该语句的标准语法如下：

```
<xsl:for-each select="pattern" order-by="sort-criteria-list">
```

如果需要对文档中的某一种标记的内容显示方式进行格式化，就可以使用 select 指定对应的标记名称以获取对应的数据。order-by 中使用的是以分号（;）分隔作为排序标准的列表。在列表元素前添加加号（+）表示按此标记的内容以升序排序，添加减号（-）表示逆序排序。作为一种简化的表示就是，排序标准列表就是由 select 规定的标记的子标记的序列，每个标记之间以（;）分隔。

<xsl:for-each>模式语句只是选取结点，并没有取出结点的值，开发人员需要使用<xsl:value-of>来真正获取对应结点的值。该语句的语法如下：

```
<xsl:value-of select="pattern">
```

其中 select 属性用来指定待获取值的结点名。

2. 匹配模式:

在该模式中包括<xsl:template>以及<xsl:apply-templates>语句。

该语句使用 match 属性从 XML 文档中选取满足条件的结点，针对这些特定的结点形成一个特定输出形式的模版。

该语句的完整语法如下：

```
<xsl:template match="node-context" language="language-name">
```

其中，match 属性用于确定在什么情况下执行此模版。作为一种简化的说明，在此处一般使用标记的名字来进行指定；其中最上层模版必须将 match 设置为“/”。

language 属性用于确定在此模版中执行什么脚本语言，其取值与 HTML 中的 script 标记

中的 language 属性的取值相同，缺省值是 JavaScript。

<xsl:apply-templates>用于调用对应的模块。其基本语法为：

```
<xsl:apply-templates select="pattern" order-by="sort-criteria-list">
```

其中，select 属性用于确定在此上下文环境中应执行什么模版，即选取用< xsl:template >标记建立的模版（块）。order-by 属性用于指定以分号（;）分隔的排序标准，通常是子标记的序列。下面结合案例来进行说明。在项目“P35_XML”中新建一个 XML 文件，名称为 template.xml，对应文档的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="template.xsl"?>
<classmates>
    <student>
        <sid>1</sid>
        <sname>张桂元</sname>
        <toefl>630</toefl>
    </student>
    <student>
        <sid>2</sid>
        <sname>贾燕枫</sname>
        <toefl>599</toefl>
    </student>
    <student>
        <sid>3</sid>
        <sname>张宇翔</sname>
        <toefl>700</toefl>
    </student>
    <student>
        <sid>4</sid>
        <sname>张大牛</sname>
        <toefl>660</toefl>
    </student>
</classmates>
```

在该文档中以树型结构的方式存储了若干个学员的成绩信息，下面我们结合 XSTL 的模版实现对应的显示。显示的效果如图 3-31 所示。

为实现图 3-31 中所显示的效果，在项目“P35_XML”中新建一个 XSL 文件，名称为 template.xsl，对应文档的内容如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <!--根模版-->
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩</title>
            </head>
            <body>
                <xsl:apply-templates select="classmates/student" />
```

图 3-31 应用模版之后的显示效果

```
            </body>
      </html>
</xsl:template>
<!--简历模版-->
<xsl:template match="student">
      <table border="1" cellspacing="0">
            <caption>
                  个人成绩单
                  <xsl:eval>formatIndex(childNumber(this),"1")</xsl:eval>
            </caption>
            <xsl:apply-templates select="sid" />
            <xsl:apply-templates select="sname" />
            <tr />
            <th>托福成绩</th>
            <td COLSPAN="5">
                  <table cellspacing="0">
                        <xsl:apply-templates select="toefl" />
                  </table>
            </td>
      </table>
      <br />
</xsl:template>
<!--学号模版-->
<xsl:template match="sid">
      <th>学号</th>
```

```
            <td>
                  <xsl:value-of />
            </td>
      </xsl:template>
      <!--姓名模版-->
      <xsl:template match="sname">
            <th>姓名</th>
            <td>
                  <xsl:value-of />
            </td>
      </xsl:template>
      <!--托福成绩模版-->
      <xsl:template match="toefl">
            <tr>
                  <td>
                        <xsl:value-of />
                  </td>
            </tr>
      </xsl:template>
</xsl:stylesheet>
```

在上面的样式表文档中，如下所示的代码即为 sid 结点定义的模版样式，例如：学号作为标题加粗显示，学号的值单独显示在一个单元格中。

```
<!--学号模版-->
      <xsl:template match="sid">
            <th>学号</th>
            <td>
                  <xsl:value-of />
            </td>
      </xsl:template>
```

对应的指定应用该模版的语句如下：

```
<xsl:apply-templates select="sid" />
```

注意在该例中，下面的语句是为了生成对应的索引编号进行显示。

```
<xsl:eval>formatIndex(childNumber(this),"1")</xsl:eval>
```

3. 测试模式:

在该模式中包括<xsl:if> 和<xsl:when>等语句。

前面曾经介绍过：XML 技术的优势之一就在于数据输出时的可选择性，即选择需要的数据进行显示输出。前面我们所讲到的选择模式语句:<xsl:for-each>、<xsl:value-of>及匹配模式语句<xsl: template>只是简单的选取通过“/”符号层层到达的节点，那么如果我们对 XML 数据不需要全部输出，而只需要满足某条件的部分数据，就需要使用条件判断<xsl:if>与多条件判断<xsl:choose>及<xsl:when>等语句，其使用方式与传统的程序设计语言中的流程控制同样是类似的。

<xsl:if>的语法结构如下：

```
<xsl:if expr="script-expression" language="language-name" test="pattern">
```

其中，expr 属性用于设置脚本语言表达式，其计算结果为“真”或者“假”；如果计算结果为“真”，且通过 test 属性所设置的对原始数据的测试条件，则在输出中显示其中的内容（该属性可以省略）。

language 属性用于设置 expr 属性中表达式的脚本语言类型，其取值与 HTML 标记 script 的 language 属性的取值相同，缺省为“JavaScript”。

test 属性用于指定对原始数据进行测试的条件。

下面通过一个实例进行说明，依然是针对前面案例中的 template.xml 文档，针对该文档采用 XSTL 中的模版以及<xsl:if>语句对其进行显示控制，对应的样式文件名称为“if.xsl”。注意修改 XML 文档中与设置样式文件位置和名称相关的语句。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
     <!--根模版-->
     <xsl:template match="/">
          <html>
               <head>
                    <title>学生成绩</title>
               </head>
               <body>
                    <xsl:apply-templates select="classmates/student" />
               </body>
          </html>
     </xsl:template>
     <!--主体模版-->
     <xsl:template match="student">
          <table border="1" cellspacing="0">
               <th>学号</th>
               <th>姓名</th>
               <th>成绩</th>
               <tr>
                    <td>
                         <xsl:value-of select="sid" />
                    </td>
                    <td>
                         <xsl:value-of select="sname" />
                    </td>
                    <td>
                         <xsl:apply-templates select="toefl" />
                    </td>
               </tr>
          </table>
     </xsl:template>

     <xsl:template match="toefl">
          <xsl:if test=".[value()$le$650]">
```

```
                <xsl:attribute name="style">color:red</xsl:attribute>
            </xsl:if>
            <xsl:value-of />
        </xsl:template>
</xsl:stylesheet>
```

对应实现的效果如图 3-32 所示，可以看到托福成绩低于 650 的成绩以红色进行了显示。

图 3-32　进行显示控制之后的效果

在上面的例子中，le是关系运算符中的“小于等于”，对应的其他关系有小于（lt）、大于（gt）、大于等于（ge）、等于（eq）、不等于（ne）等。“.”表示引用当前标记，本例中为“toefl”，[ ]表示筛选，只有满足筛选条件的标记才能适用于对应的显示控制。value()是 XSTL 中的函数，其他常用的函数还有 text()、end()、index()等等。

如果我们希望对同一数据同时测试多个条件，根据不同条件输出相应结果。那么除了使用<xsl:if>语句之外，也可以使用<xsl:choose>语句，该语句的语法如下：

```
<xsl:choose>
<xsl:when expr="script-expression" language="language-name" test="pattern">
<xsl:otherwise>
```

其中<xsl:choose>表示一个多路选择分支的开始。<xsl:when>语句中相关属性的含义与<xsl:if>类似，这里不在赘述。<xsl:otherwise>表示如果有不满足<xsl:when>规定的条件，则按照此标记中的内容进行输出。

下面依然通过一个实例进行说明，针对前面案例中的 template.xml 文档，针对该文档采用 XSTL 中的模版以及<xsl:choose>语句对其进行显示控制，对应的样式文件名称为“choose.xsl”，

注意修改XML文档中与设置样式文件位置和名称相关的语句。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <!--根模版-->
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩</title>
            </head>
            <body>
                <xsl:apply-templates select="classmates/student" />
            </body>
        </html>
    </xsl:template>
    <!--主体模版-->
    <xsl:template match="student">
        <table border="1" cellspacing="0">
            <th>学号</th>
            <th>姓名</th>
            <th>托福成绩</th>
            <tr>
                <td>
                    <xsl:value-of select="sid" />
                </td>
                <td>
                    <xsl:value-of select="sname" />
                </td>
                <td>
                    <xsl:apply-templates select="toefl" />
                </td>
            </tr>
        </table>
    </xsl:template>

    <xsl:template match="toefl">
        <xsl:choose>
            <xsl:when test=".[value()$ge$700]">优秀</xsl:when>
            <xsl:when test=".[value()$gt$600]">一般</xsl:when>
            <xsl:when test=".[value()$gt$500]">及格</xsl:when>
            <xsl:otherwise>不及格</xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

对应实现的效果如图3-33所示，可以看到托福成绩根据实际的分数依据不同的评价等级进行了显示。

最后再通过两个实例介绍，如何综合使用上面介绍的相关语句实现对前面XML文件的显示。例如，我们针对xml1.xml文件中的数据，配合对应的XSTL实现对应数据信息的显示，

图 3-33　进行显示控制之后的效果

将对应的文件进行复制，得到 xml2.xml。在项目“P35_XML”中新建一个 XSTL 文件，名称为 xml2.xsl，对应的样式文件中样式的设置为：

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <html>
            <body>
                <center>
                    <table border="1">
                        <tr>
                            <td>学号</td>
                            <td>姓名</td>
                            <td>GRE 成绩</td>
                        </tr>
                        <xsl:for-each select="classmates/student">
                            <tr>
                                <td>
                                    <xsl:value-of select="sid" />
                                </td>
                                <td>
                                    <xsl:value-of select="sname" />
                                </td>
                                <td>
                                    <xsl:value-of select="gre" />
```

```
                                    </td>
                                </tr>
                            </xsl:for-each>
                        </table>
                    </center>
                </body>
            </html>
        </xsl:template>
    </xsl:stylesheet>
```

然后在原有的 xml2.xml 文件中添加新的一行如下所示，该行用于指定所需使用的样式表文件的类型、位置以及名称。

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xml2.xsl"?>
<classmates>
    <student>
        <sid>1</sid>
        <sname>张桂元</sname>
        <gre>1700</gre>
        <toefl>630</toefl>
        <tse>120</tse>
    </student>
        ……
```

此时，在浏览器中再次直接打开 xml1.xml 文件的效果如图 3-34 所示。可以看到树型结构中的数据已经以表格的形式进行了相应的显示。

图 3-34 浏览器中配合 XSLT 显示 XML 文档的效果

下面再给出一个实现效果更加复杂的例子，在项目中新建一个 XML 的文档，名称为 xml3.xml，在该文档中存储的是相关下载软件的列表信息，对应的文档内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xml3.xsl" ?>
<搞笑中心软件店>
    <说明>此测试页使用 XML 语言制作，必须使用 IE5.0 或以上版本才能正常浏览.</说明>
    <软件列表>
        <软件>
            <序号>1</序号>
            <名称>Goodnight 歌</名称>
            <作恶度>--</作恶度>
            <搞笑度>* * * *</搞笑度>
            <大小>633k</大小>
            <介绍>这不是恶作剧软件，但却是一首很好听的歌，动画也很有趣，建议送给 baby。
        </介绍>
            <网址>../main/soft/011goodnight.exe</网址>
        </软件>

        <软件>
            <序号>2</序号>
            <名称>搞怪鼠标（3）</名称>
            <作恶度>* *</作恶度>
            <搞笑度>*</搞笑度>
            <大小 />
            <介绍>使鼠标突然变大。</介绍>
            <网址>../main/soft/024vigor.zip</网址>
        </软件>

        <软件>
            <序号>3</序号>
            <名称>模拟电话拨号程序</名称>
            <作恶度>* * * *</作恶度>
            <搞笑度>* * * *</搞笑度>
            <大小>70k</大小>
            <介绍>铃，铃.....电话响了，听听是谁来的电话！</介绍>
            <网址>../main/soft/051ringing.zip</网址>
        </软件>
        <软件>
            <序号>4</序号>
            <名称>电脑升级</名称>
            <作恶度>* * *</作恶度>
            <搞笑度>* * *</搞笑度>
            <大小>23k</大小>
            <介绍>经过这次软升级，可使你的内存效率提高 16 倍，硬盘效率提高 10 倍！(运行前 c 盘根
目录下要有一个 320x400，256 色的 bmp 图像，文件名 logo.sys，不然会出错)</介绍>
            <网址>../main/soft/052level.zip</网址>
        </软件>

    </软件列表>
    <版权说明>搞笑中心软件店版权所有</版权说明>
</搞笑中心软件店>
```

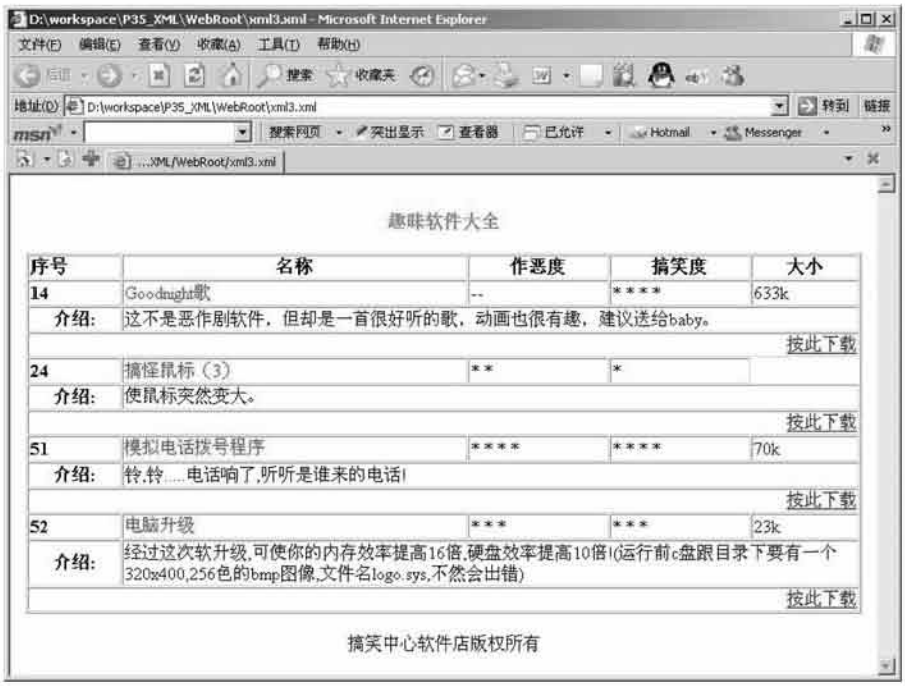
可以看到，这是一个标准的 XML 文档，特殊之处在于在该文档中使用了中文的标记名称。

下面我们配合使用 XSTL 样式文件控制对应内容的显示。新建一个文件，名称为 xml3.xsl，对应的样式内容如下：

```
<?xml version="1.0" encoding="GB2312" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
      <xsl:template match="/">
            <br>
                <b>
                    <div align="center">
                        <font color="#00aaaa">趣味软件大全</font>
                    </div>
                </b>
            </br>
            <table border="1" cellpadding="1" cellspacing="1" width="760" align=
"center">
                <tr>
                    <th width="80" align="left">序号</th>
                    <th>名称</th>
                    <th>作恶度</th>
                    <th>搞笑度</th>
                    <th>大小</th>
                </tr>
                <xsl:for-each select="搞笑中心软件店/软件列表/软件">
                    <tr>
                        <td>
                            <b>
                                <xsl:value-of select="序号" />
                            </b>
                        </td>
                        <td>
                            <font color="#ff0000">
                                <xsl:value-of select="名称" />
                            </font>
                        </td>
                        <td>
                            <xsl:value-of select="作恶度" />
                        </td>
                        <td>
                            <xsl:value-of select="搞笑度" />
                        </td>
                        <td>
                            <xsl:value-of select="大小" />
                        </td>
                    </tr>
                    <tr>
                        <th>介绍:</th>
                        <td colspan="4">
                            <xsl:value-of select="介绍" />
                        </td>
                    </tr>
```

```
                    <tr>
                        <td colspan="5" align="right">
                            <a target="_blank">
                                <xsl:attribute name="href">
                                    <xsl:value-of select="网址" />
                                </xsl:attribute>
                                按此下载
                            </a>
                        </td>
                    </tr>
                </xsl:for-each>
            </table>
            <p align="center">
                <xsl:value-of select="搞笑中心软件店/版权说明" />
            </p>
        </xsl:template>
    </xsl:stylesheet>
```

注意在该例中实现超级链接的方式时，所使用的<xsl:attribute>标记。对应 XML 文档在页面中的显示效果如图 3-35 所示。



序号	名称	作恶度	搞笑度	大小
14	Goodnight歌	--	****	633k
介绍:	这不是恶作剧软件，但却是一首很好听的歌，动画也很有趣，建议送给baby。			
				按此下载
24	搞怪鼠标（3）	**	*	
介绍:	使鼠标突然变大。			
				按此下载
51	模拟电话拨号程序	****	****	70k
介绍:	铃,铃.....电话响了,听听是谁来的电话!			
				按此下载
52	电脑升级	***	***	23k
介绍:	经过这次软升级,可使你的内存效率提高16倍,硬盘效率提高10倍!(运行前c盘跟目录下要有一个320x400,256色的bmp图像,文件名logo.sys,不然会出错)			
				按此下载

图 3-35　浏览器中配合 XSLT 显示 XML 文档的效果

3.5　XMLHttpRequest 对象

XMLHttpRequest 是 XMLHTTP 组件的对象，通过这个对象，Ajax 可以像桌面应用程序

一样只同服务器进行数据层面的信息交换，而不用每次都刷新页面，也不用每次将数据处理的工作都交给服务器来做；这样既减轻了服务器的负担又加快了响应速度、缩短了用户等待的时间。

XMLHttpRequest 是 Ajax 技术中最重要的一个对象。通过本书第 2 章中的实例，读者已经初步了解到一个页面在不刷新的情况下可以通过 XMLHttpRequest 对象发送请求来获取服务器响应。在实际的开发中，设计人员也可以通过使用该对象在不刷新当前页面的情况下更新页面局部的数据。

XMLHttpRequest 是浏览器中已定义好的对象，它是 Ajax 技术的核心组成部分，JavaScript 通过它和服务器之间进行通信，并借助它来解析从服务器传回来的 XML 文件。

我们已经通过前面的实例了解了有关该对象的一些属性和方法。本节将系统的介绍有关 XMLHttpRequest 对象更多的属性和方法。

需要注意的是，该对象的创建方法是与浏览器相关的。从 IE 5.0 浏览器开始，开发人员可以在 Web 页面内部使用 XMLHTTP ActiveX 组件扩展自身的功能。Mozilla 1.0 浏览器以及 NetScape 7 浏览器则是创建继承 XML 的代理类 XMLHttpRequest。在大多数情况下，XMLHttpRequest 对象和 XMLHTTP 组件都是非常接近，它们的方法和属性基本类似，只是在部分属性上有所不同。下面给出的是在本书第 2 章的案例 2-2 中所使用的相关创建对象的语句。

```
<script language="javascript">
     var XMLHttpReq = false;
     //创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
          if(window.XMLHttpRequest) { //Mozilla 浏览器
               XMLHttpReq = new XMLHttpRequest();
          }
          else if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               } catch (e) {
                    try {
                         XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                    } catch (e) {}
               }
          }
     }
</script>
```

虽然在不同的浏览器中创建该对象的方法不同，由于 JavaScript 脚本语言是动态的，解释型的语言，并且 XMLHttpRequest 对象在不同的浏览器中是兼容的，因此在实际编程中开发人员可以采用同样的方式使用该对象的属性和方法。

表 3-2 中列出了 XMLHttpRequest 对象相关的属性和方法。

表 3-2 XMLHttpRequest 对象的方法

方　法	功　能
abort()	停止当前请求
getAllResponseHeaders()	把 HTTP 请求的所有响应首部作为键/值对返回
getResponseHeader("headerLabel")	返回指定首部的字符串的值
open("method","URL""[,asyncFlag[,"userName"[, "password"]]])	建立对服务器的访问。其中 method 参数可以使用 GET、POST 及 PUT，URL 参数既可以使用绝对地址，也可以使用相对地址，此外还包括与建立连接相关的三个参数
send(content)	向服务器发送请求
setRequestHeader("label", "value")	设置 header 并和请求一起发送，即指定首部设置为所提供的值，注意在设置首部前必须先使用 open()方法

在上表中列出的方法里面，open("method","URL""[,asyncFlag[,"userName"[, "password"]]]) 方法的对应参数一共有 5 个，其中前面两个是必须的，后面三个是可选的。调用该方法将建立 Ajax 对服务器的调用，即完成请求初始化的方法。在调用该方法时，需要指定调用的方法，包括 GET、POST 及 PUT，还需要提供所调用资源的 URL，此外，boolean 类型的 asyncFlag[用于指定在进行调用时是采用同步方式还是异步方式，默认为 true，即采用异步方式。如果参数值为 false，则采用同步方式实现和服务器的交互，即进行等待直到服务器返回响应结果为止。最后两个参数用于在进行连接时指定特定的用户名和密码。

send(content)方法用于具体向服务器发出请求。可以使用的参数包括 DOM 对象、字符串等等。

setRequestHeader("label", "value")方法是为 HTTP 请求中给定的一个首部设置值，其中第一个参数表示要设置的首部名称，第二个参数用于指定放在首部中的值。需要注意该方法在使用前必须先调用 open()方法。

除了上面的相关方法之外，XMLHttpRequest 对象还提供了许多属性，相关属性名称及功能如表 3-3 所示。

表 3-3 XMLHttpRequest 对象的属性

方　法	功　能
onreadystatechange	状态改变的事件触发器
readyState	对象状态(integer): 0 = 未初始化 1 = 读取中 2 = 已读取 3 = 交互中 4 = 完成

续表

方　法	功　能
ResponseText	服务器进程返回数据的文本，表示为一个字符串
responseXML	服务器进程返回数据的兼容 DOM 的 XML 文档对象
status	服务器返回的状态码，如：404 = "文件未找到" 、200 ="成功"
statusText	服务器返回的状态文本信息（OK 或者 Not Found 等）

其中 onreadystatechange 属性是状态改变的事件触发器，每个状态改变时都会触发这个事件触发器，通常在触发后会调用一个 JavaScript 脚本语言编写的函数。

3.6 Ajax 核心工作机制

在了解了 XMLHttpRequest 对象相关的属性和方法之后，我们再明确一下发送和接收 XMLHttpRequest 请求的基本程序框架。

通过前面的案例分析可以清楚地看到：Ajax 实质上是遵循 Request/Server 模式来进行工作的，所以这个框架基本的流程包括下面几个具体的步骤。

（1）对象初始化

（2）发送请求

（3）服务器接收请求并进行处理

（4）服务器返回响应数据

（5）客户端接收

（6）依据响应数据修改客户端页面内容

注意整个过程中的通信交互方式是异步的。下面依据具体的处理过程再次强调 Ajax 的核心工作机制。

1. *初始化对象并发出 XMLHttpRequest 请求*

为了让 Javascript 可以向服务器发送 HTTP 请求，必须使用 XMLHttpRequest 对象。使用之前，要先将 XMLHttpRequest 对象实例化。之前说过，各个浏览器对这个实例化过程实现不同。IE 浏览器以 ActiveX 控件的形式提供，而 Mozilla 浏览器等浏览器则直接以 XMLHttpRequest 类的形式提供。为了让编写的程序能够跨浏览器运行，可以这样写：

```
if(window.XMLHttpRequest) { //Mozilla 浏览器
        XMLHttpReq = new XMLHttpRequest();
    }
    else if (window.ActiveXObject) { // IE 浏览器
        try {
            XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
```

```
                try {
                    XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
```

有些版本的 Mozilla 浏览器处理服务器未包含 XML mime-type 头部信息的返回内容时会出错。因此，要确保返回的内容包含 text/xml 信息。

```
XMLHttpReq = new XMLHttpRequest();
XMLHttpReq.overrideMimeType("text/xml");
```

2. 指定响应处理函数

接下来要指定当服务器返回信息时客户端的处理方式。只要将相应的处理函数名称赋给 XMLHttpRequest 对象的 onreadystatechange 属性就可以了。比如：

```
XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
```

需要注意的是，这个函数名称不加括号，不指定参数。也可以用 Javascript 即时定义函数的方式定义响应函数。比如：

```
XMLHttpReq.onreadystatechange = function() { };
```

3. 发出 HTTP 请求

指定响应处理函数之后，就可以向服务器发出 HTTP 请求了。这一步调用 XMLHttpRequest 对象的 open 和 send 方法。

```
XMLHttpReq.open("GET", url, true);
XMLHttpReq.send(null);  // 发送请求
```

open 的第一个参数是 HTTP 请求的方法，为 GET、POST 或者 Head。open 的第二个参数是目标 URL。基于安全考虑，这个 URL 只能是同网域的，否则会提示“没有权限”的错误。这个 URL 可以是任何的 URL，包括需要服务器解释执行的页面，不仅仅是静态页面。目标 URL 处理请求 XMLHttpRequest 请求则跟处理普通的 HTTP 请求一样，比如 JSP 可以用 request.getParameter(“”)或者 request.getAttribute(“”)来取得 URL 参数值。

open 的第三个参数只是指定在等待服务器返回信息的时间内是否继续执行下面的代码。如果为 true，则不会继续执行，直到服务器返回信息。默认为 true。

按照顺序，open 调用完毕之后要调用 send 方法。send 的参数如果是以 POST 方式发出的话，可以是任何想传给服务器的内容。

4. 处理服务器返回的信息

在第二步我们已经指定了响应处理函数，这一步是用来描述响应处理函数具体应该做的事情。

首先，它要检查 XMLHttpRequest 对象的 readyState 值，判断请求目前的状态。参照前面的属性表可以知道，readyState 值为 4 的时候，代表服务器已经传回所有的信息，可以开始处理信息并更新页面内容了。例如：

```
if (XMLHttpReq.readyState == 4) {
```

```
    // 信息已经返回，可以开始处理
} else {
    // 信息还没有返回，等待
}
```

服务器返回信息后，还需要判断返回的 HTTP 状态码，确定返回的页面没有错误。所有的状态码都可以在 W3C 的官方网站上查到。其中，200 代表页面正常，基本程序结构如下：

```
if (XMLHttpReq.status == 200) {
    // 页面正常，可以开始处理信息
} else {
    // 页面有问题
}
```

XMLHttpRequest 对成功返回的信息有两种处理方式：一种为 responseText：即将传回的信息当字符串使用；另一种为 responseXML：即将传回的信息当 XML 文档使用，可以用 DOM 处理。本书前面的实例采用的是第二种方式。

总结上面的步骤，我们就可以整理出一个初步的 Ajax 开发框架，供以后调用。在本程序框架中，将服务器返回的信息用 window.alert 以字符串的形式显示出来：

```
<script language="javascript">
      var XMLHttpReq = false;
      //创建 XMLHttpRequest 对象
      function createXMLHttpRequest() {
            if(window.XMLHttpRequest) { //Mozilla 浏览器
                  XMLHttpReq = new XMLHttpRequest();
            }
            else if (window.ActiveXObject) { // IE 浏览器
                  try {
                        XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                  } catch (e) {
                        try {
                              XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                        } catch (e) {}
                  }
            }
      }
      //发送请求函数
      function sendRequest(url) {
            createXMLHttpRequest();
            XMLHttpReq.open("GET", url, true);
            XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
            XMLHttpReq.send(null);  // 发送请求
      }
      // 处理返回信息函数
      function processResponse() {
      if (XMLHttpReq.readyState == 4) { // 判断对象状态
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].first
                        Child.data;
```

```
                window.alert(res);
            } else { //页面不正常
                window.alert("您所请求的页面有异常。");
            }
        }
    }
</script>
```

在本章的最后，我们再通过一个实例进一步帮助读者体会 Ajax 技术完整的工作过程及设计方式。该实例实现的是网页中常见的提示信息显示的效果。在传统的实现方式中，往往需要提前从数据库中获取大量的信息，或者需要页面全部刷新才能够实现。

案例 3-6　利用 Ajax 技术实现页面提示效果

这里我们使用 Ajax 技术实现只有在需要的时候才发送所需的信息，实现页面局部刷新的效果。在 Eclipse 中新建一个项目，项目的名称为“P36_Tips”。该程序实现的效果如图 3-36 所示。当鼠标指针悬停在对应的商品图片上方时，将从服务器端数据库中获取对应的商品描述信息，传递回 HTML 页面以分层表格的形式进行显示。

图 3-36　工具提示显示效果

对应的 HTML 页面程序的代码如下所示。在该程序中除了与 Ajax 相关的程序基本框架之外，读者需要特别关注一下使用 DOM 操作 HTML 文档中对应元素的方法。

源程序名称：main.htm

```
<html>
    <head>
        <title>shopping online</title>
        <meta http-equiv="content-type" content="text/html; charset=gb2312">
        <link href="images/css.css" type="text/css" rel="stylesheet">
    </head>
    <script type="text/javascript">
  var datadiv;
  var datatablebody;
  var curelement;

    var xmlhttpreq = false;
    //创建 xmlhttprequest 对象
  function createxmlhttprequest() {
        if(window.xmlhttprequest) { //mozilla 浏览器
            xmlhttpreq = new xmlhttprequest();
        }
        else if (window.activexobject) { // IE 浏览器
            try {
                xmlhttpreq = new activexobject("msxml2.xmlhttp");
            } catch (e) {
                try {
                    xmlhttpreq = new activexobject("microsoft.xmlhttp");
                } catch (e) {}
            }
        }
    }
    //发送请求函数
    function getdetail(element) {
        datatablebody = document.getelementbyid("databody");
        datadiv = document.getelementbyid("popup");
         createxmlhttprequest();
        curelement = element;
        var url = "tipservlet?key=" + escape(element.id);
         xmlhttpreq.open("get", url, true);
         xmlhttpreq.onreadystatechange = processresponse;//指定响应函数
         xmlhttpreq.send(null);  // 发送请求
    }
    // 处理返回信息函数
    function processresponse() {
    if (xmlhttpreq.readystate == 4) { // 判断对象状态
          if (xmlhttpreq.status == 200) { // 信息已经成功返回，开始处理信息
                    setdata(xmlhttpreq.responsexml);
            } else { //页面不正常
                window.alert("您所请求的页面有异常。");
            }
        }
  }
```

```
    // 显示提示框
    function setdata(data) {
        cleardata();
        setoffsets();
        var content = data.getelementsbytagname("content")[0].firstchild.data;
        var row = createrow(content);
        datatablebody.appendchild(row);
    }
    //生成表格内容行
    function createrow(data) {
      var row, cell, txtnode;
      row = document.createelement("tr");
      cell = document.createelement("td");
      cell.setattribute("bgcolor", "#fffafa");
      cell.setattribute("border", "0");
      txtnode = document.createtextnode(data);
      cell.appendchild(txtnode);
      row.appendchild(cell);
      return row;
    }
    //设置显示位置
    function setoffsets() {
      datadiv.style.border = "black 1px solid";
      var top = 0;
      while(curelement) {
        top += curelement["offsettop"];
        curelement = curelement.offsetparent;
      }
      datadiv.style.left = 50 + "px";
      datadiv.style.top = top + "px";
     }

    // 清除提示框
    function cleardata() {
      var ind = datatablebody.childnodes.length;
      for (var i = ind - 1; i >= 0 ; i--) {
           datatablebody.removechild(datatablebody.childnodes[i]);
      }
      datadiv.style.border = "none";
     }
</script>

    <body leftmargin="0" topmargin="0">
        <table cellspacing="0" cellpadding="0" width="778" align="center" border="0">
            <tbody>
                <tr>
                    <td height="10"></td>
                </tr>
              </tbody>
```

```
            </table>
            <table height="148" cellspacing="0" cellpadding="0" width="778" align="center"
border="0">
                  <tbody>
                        <tr valign="top">
                              <td width="236">
                                    <table width="375" height="340">
                                          <!--dwlayouttable-->
                                          <tbody>
                                                <tr>
                                                      <td width="348" height="1">
                                                      </td>
                                                      <td height="1" width="29"></td>
                                                </tr>
                                                <tr>
                                                      <td height="13" width="348">
                                                      <table id="autonumber1" style="border-
collapse: collapse" bordercolor="#111111" height="20" cellspacing="0" cellpadding="0"
width="151" background="images/fu.gif" border="0">
                                                            <tbody>
                                                                  <tr>
                                                                        <td align="middle"
width="82">
                                                                              <b>精品推荐</b>
                                                                        </td>
                                                                        <td align="middle">
                                                                        </td>
                                                                  </tr>
                                                            </tbody>
                                                      </table>
                                                </td>
                                                <td height="13" width="29">
                                                </td>
                                          </tr>
                                          <tr>
                                                <td valign="top" height="328" width="348">
                                                      <table cellspacing="0" cellpadding="0"
width="103%" border="0">
                                                            <tbody>
                                                                  <tr>
                                                                        <td width="50%">
                                                                              <img id="1" onm-
ouseover="getdetail(this);" onmouseout="cleardata();" src="images/ibmt43bb4.jpg" vspace="6"
border="0" alt="">
                                                                              <a><b>
                                                                              <br>
                                                                        </td>
                                                                        <td width="50%">
                                                                              <br>
```

```
                                                                    商品名称：
                                                                    ibm-t43bb
                                                                    <br>
                                                                    会员价:2500元
                                                                    <br>
                                                               </td>
                                                          </tr>
                                                          <tr>
                                                               <td width="50%">
                                                                    <img id="2" onm-
ouseover="getdetail(this);" onmouseout="cleardata();" src="images/hpnc4200.jpg" vspace="6"
border="0" alt="">
                                                               </td>
                                                               <td width="50%">
                                                                    <br>
                                                                    商品名称：
                                                                    hp-nc4200
                                                                    <br>
                                                                    <br>
                                                                    会员价：1150元
                                                                    <br>
                                                               </td>
                                                          </tr>
                                                          ……
                         <tr>
                              <td valign="top" colspan="4" height="16" width="776">
                              </td>
                         </tr>
                    </tbody>
               </table>
               <div style="position:absolute;" id="popup">
                    <table bgcolor="#fffafa" border="0" cellspacing="2" cellpadding="2" />
                         <tbody id="databody">
                         </tbody>
                    </table>
               </div>
          </body>
     </html>
```

在上面的程序中，id 为 popup 的 div 层中放置了一个动态的表格，该表格主体的 id 为 databody，当 Ajax 接收到返回的响应数据，即查询到对应商品的描述信息后，将会将其作为表格的一行添加到该动态表格中，并依据所选商品图片的位置计算出对应 div 层的位置，然后显示在页面中。

在下一次进行操作时，将首先清除动态表格中原有的内容，然后再进行同样的处理以进行显示。本案例中使用到了借助 DOM 操作 HTML 文档的相关方法。例如，下面的代码段就是完成创建表格中的一行，并将获取的数据信息作为单元格的内容添加进去。

```
var row, cell, txtnode;
row = document.createelement("tr");
```

```
cell = document.createelement("td");
cell.setattribute("bgcolor", "#fffafa");
cell.setattribute("border", "0");
txtnode = document.createtextnode(data);
cell.appendchild(txtnode);
row.appendchild(cell);
```

而将对应的行添加到表格中的语句如下：

```
datatablebody.appendchild(row);
```

将原有表格中的内容清除的语句为：

```
datatablebody.removechild(datatablebody.childnodes[i]
```

计算位置并进行美化输出的语句如下：

```
datadiv.style.border = "black 1px solid";
          var top = 0;
          while(curelement) {
             top += curelement["offsettop"];
             curelement = curelement.offsetparent;
          }
          datadiv.style.left = 50 + "px";
          datadiv.style.top = top + "px";
```

有关借助 DOM 操作 HTML 文档及 XML 文档的详细方式将在本书后续章节中进行具体介绍。

服务器端对应的程序文件分别如下，其中配置文件 web.xml 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>TipServlet</servlet-name>
           <servlet-class>noteBookStore.TipServlet</servlet-class>
      </servlet>
      <servlet-mapping>
           <servlet-name>TipServlet</servlet-name>
           <url-pattern>/TipServlet</url-pattern>
      </servlet-mapping>
      <session-config>
           <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
           <welcome-file>main.htm</welcome-file>
      </welcome-file-list>
</web-app>
```

创建 MySQL 中对应原始数据的相关脚本如下所示。

```
CREATE TABLE sort(
   id INTEGER PRIMARY KEY,
   Name VARCHAR (40) NOT NULL)
```

```
CREATE TABLE product(
    id INTEGER PRIMARY KEY,
    sortid INTEGER NOT NULL
           REFERENCES sort(id) ON DELETE CASCADE,
    name VARCHAR (50) NOT NULL,
    price DOUBLE NOT NULL,
    salepriceDOUBLE NOT NULL,
    descript TEXT(500) NOT NULL,
    contents TEXT(2000) NULL,
    saledate DATE NOT NULL,
    salecountINTEGER NULL,
    image VARCHEAR (50) NULL
)

insert into sort  values(1,'IBM')
insert into sort  values(2,'SONY')
insert into sort  values(3,'HP')
insert into sort  values(4,'TOSHIBA')
insert into sort  values(5,'联想')
insert into sort  values(6,'同方')
```

```
insert into product values(1,1,'IBM-T43',14500,13500,'CPU 处理器:Intel Pentium-M(Dothan 760)2.0GHz','芯片组:Intel 915PM 系统主频:533 MHz 二级缓存:2M 图形显示卡:ATI Mobility Radeon X300,PCI-Express 显示内存: 64MB DDR2802.11B+G 无线网卡,内置蓝牙技术。','2005-10-22',100,'images/IBMT43CH8.jpg')

insert into product values(2,1,'HP-NC4200',13500,13000,'CPU 处理器:Intel Pentium-M(Dothan 760)
2.0GHz','芯片组:Intel 915PM 系统主频:533 MHz 二级缓存:2M 图形显示卡:ATI Mobility Radeon X300,PCI-Express 显示内存: 64MB DDR2802.11B+G 无线网卡,内置蓝牙技术。','2005-10-22',100,'images/HPNC4200.jpg')

insert into product values(3,1,'SONY-A21C',16500,15000,'CPU 处理器:Intel Pentium-M(Dothan 760)2.0GHz','芯片组:Intel 915PM 系统主频:533 MHz 二级缓存:2M 图形显示卡:ATI Mobility Radeon X300,PCI-Express 显示内存:64MB DDR2802.11B+G 无线网卡,内置蓝牙技术。','2005-10-22',100,'images/SONYA21C.jpg')
```

在 MySQL 中查看数据准备情况的效果如图 3-37 所示。

id	name
1	IBM
2	SONY
3	HP

id	so...	name	price	saleprice	descript	contents	saledate
1	1	IBM-T43	14500	13500	CPU处理器:Intel...	芯片组：Intel 915PM系统主频：533 ...	0000-00
2	1	HP-NC4200	13500	13000	CPU处理器:Intel...	芯片组：Intel 915PM系统主频：533 ...	0000-00
3	1	SONY-A21C	16500	15000	CPU处理器:Intel...	芯片组：Intel 915PM系统主频：533 ...	0000-00

图 3-37　准备测试数据

接下来创建一个 JavaBean，以封装对数据库进行操作的业务逻辑，该程序源代码如下：

源文件：DB.java

```
package classmate;

import java.sql.Connection;
……

public class DB {

      Connection connect = null;
      ResultSet rs = null;
      public DB() {
            try {
                  Class.forName("org.gjt.mm.mysql.Driver");   //设置驱动程序类型
            }
            catch(java.lang.ClassNotFoundException e) {
                  e.printStackTrace();
            }
      }
      //执行查询语句的方法
      public ResultSet executeQuery(String sql) {
            try {
                  connect = DriverManager.getConnection("jdbc:mysql://localhost/test",
"root", "zgy01");  //建立与数据库服务器的连接
                  Statement stmt = connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
                  rs = stmt.executeQuery(sql);//执行指定的数据查询语句
            }
            catch(SQLException ex) {
                  ex.printStackTrace();
            }
            return rs;
      }
     //执行增、删改语句的方法
      public int executeUpdate(String sql) {
            int result = 0;
            try{
                  connect = DriverManager.getConnection("jdbc:mysql://localhost/test",
"root", "zgy01");
                  Statement stmt = connect.createStatement();
                  result = stmt.executeUpdate(sql);//执行指定的数据操作语句
            }
            catch(SQLException ex) {
                  System.err.println(ex.getMessage());
            }
            return result;
      }
      //关闭数据库连接的方法
      public void close(){
```

```
            if(connect!=null){
                try{
                    connect.close();
                    connect = null;
                }catch(SQLException ex) {
                    System.err.println(ex.getMessage());
                }
            }
        }
    }
    ……
```

再创建一个 JavaBean，以封装对商品信息进行操作的业务逻辑，该程序源代码如下所示。其中 GetDetail()方法中封装了按照编号获取对应商品描述信息的操作步骤。

源文件：Product.java

```
package noteBookStore;

import java.sql.ResultSet;

public class Product {
    private int id;
    private String name;
    private String contents;
    public String getContents() {
        return contents;
    }
    public void setContents(String contents) {
        this.contents = contents;
    }
    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 static Product GetDetail(DB db,Integer Id) throws Exception{
        ResultSet rs;
       String strSql=null;
       String rplContent=null;

       strSql = "select * from product where id = " + Id ;
        rs = db.executeQuery(strSql);
```

```
            Product product = new Product();
            if (rs.next()){

                  product.setName(rs.getString("name")) ;

                  rplContent = rs.getString("contents");
                  rplContent = rplContent.replaceAll("\n","<br>");
                  product.setContents(rplContent) ;
            }
            return product;
      }
}
```

最后介绍一下对应的 Servlet 控制器程序。

源文件：TipServlet.java

```
package noteBookStore;

import java.io.*;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.*;
import javax.servlet.http.*;

public class TipServlet extends HttpServlet {

      public void init(ServletConfig config) throws ServletException {
      }

      /**
        *处理 Get 方法
        */
      protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      request.setCharacterEncoding("UTF-8");
          Integer key = Integer.valueOf(request.getParameter("key"));
//从数据库获取内容详细数据
        DB db = new DB();
        Product product=null;
        try {
              product = Product.GetDetail(db,key);
         } catch (Exception e) {
              e.printStackTrace();
         }
//传回响应数据
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        out.println("<response>");
```

```
            out.println("<content>" + product.getContents() + "</content>");
            out.println("</response>");
            out.close();
        }

    }
```

在上面的程序中，依据获取到的浏览器客户端的信息组合成查询语句，然后依据查询的结果返回不同的响应数据。响应数据的格式采用 XML 文档的形式，具体内容如下：

```
<response>
<content>描述信息</content>
</response>
```

本章中介绍了 Ajax 中的相关核心技术，在下一章中将更深入地讨论与 Ajax 关联的常用设计方法和技巧。

CHAPTER 4

第 4 章　Ajax 相关技术深入

前面，我们结合实际案例介绍了 Ajax 的核心技术。对 Ajax 框架的核心工作机制已经有了一定的了解。为了掌握更加灵活的 Ajax 应用开发方式，本章将对 Ajax 相关的技术进行深入介绍。

本章讲解中，将通过几个实际案例介绍如何结合 DOM 对 HTML 文档以及 XML 文档进行操作和控制，以及请求提交方式。介绍如何结合服务器端技术和 Ajax 实现动态 Java 应用程序的构建。

在本章的案例中，读者将掌握操作 HTML 文档及 HTML 文档的方式，以及在 Ajax 技术中提交请求的几种不同方式，讲解 Ajax 中常见的拖拽效果以及褪色技术的实现过程。

4.1　使用 DOM 操作 HTML 文档

在本书第 3 章的案例 3-6 中，为了能够实现 HTML 页面中工具提示的效果，我们采用 DOM 对 HTML 文档进行了操作，前面的代码如下：

```
var row, cell, txtnode;
row = document.createelement("tr");
cell = document.createelement("td");
cell.setattribute("bgcolor", "#fffafa");
cell.setattribute("border", "0");
txtnode = document.createtextnode(data);
cell.appendchild(txtnode);
row.appendchild(cell);
```

在对应代码中使用的就是 DOM 操作 HTML 的方式。例如，创建一个表格行元素、单元格元素、设置单元格对应的属性，包括背景色、边框线等等。此外，还包括向单元格中添加文本信息的内容，这里是以文本结点的方式创建，并作为单元格元素的子结点添加进去然后将单元格元素作为表格行元素的子结点添加到表格对应的行中。

由于当前的浏览器大多使用 W3C DOM 来表示 Web 页面的内容，因此 Web 页面在不同的浏览器版本中呈现的方式是一样的。用于修改页面内容的脚本在使用方式上也没有区别，开发人员可以方便地使用 DOM 结合 JavaScript 脚本实现对 HTML 文档的操作。

下面具体介绍一下对应的操作方法。已经知道脚本语言通过 DOM 才可以实现和页面进行交互，Web 开发人员总是将页面中的元素当作对象进行操作，通过获取对应的属性及调用相应的方法控制修改对象的状态，例如，document 就代表页面对象本身，可以通过修改该对象的某些属性对 HTML 文档进行修改。下面，借助该案例中若干个页面效果来体会 DOM 操作 HTML 文档的方式。

案例 4-1　使用 DOM 操作 HTML 文档

在 Eclipse 中新建一个项目，项目的名称为“P41_HTMLDOM”。首先，新建一个 HTML 文档，在页面首部添加 JavaScript 脚本，当我们在脚本区域中输入 document.时，页面中将显示如图 4-1 所示的提示列表框，可以看到有关该对象的属性和方法均显示出来。

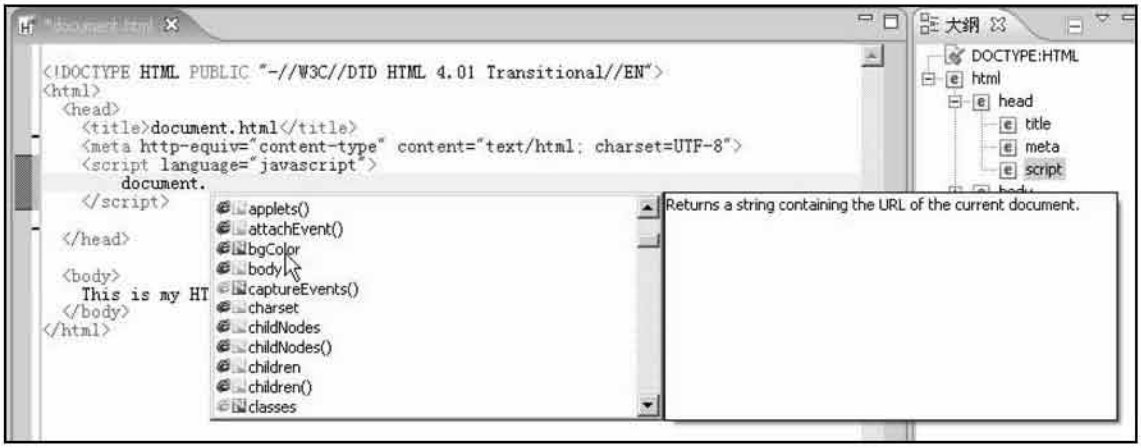


图 4-1　有关 DOM 对象属性及方法的提示列表框

如果我们选择“bgColor”属性，并对其值进行设置，就是修改 document 对象的背景色属性，达到控制页面背景色的目的。对应的代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>document.html</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <script language="javascript">
     document.bgColor = "#00eeaa";
    </script>
  </head>
  <body>
    有关对象的属性 <br>
  </body>
</html>
```

预览该页面可以看到对应的背景色已经发生了改变。通过这个实例，了解到 DOM 的

基本操作原理，是以对象的形式对 HTML 文档中的各个组成部分进行表示，HTML 文档的树型结构中主要包含表示元素、标记的元素结点（elementNode）和表示文本串的结点（textNode）。

那么，如何方便地通过 DOM 遍历这棵树并检查、修改树上结点的方法和属性呢？就需要了解 DOM 中针对结点操作的相关属性和方法。

必知必会：HTML DOM 中常用的对象操作属性及方法

按照 DOM 的操作规则，HTML 文档中的各个标记、元素被视为各种类型的 Node 对象，即结点对象。每个 Node 对象都有自己的属性和方法，利用这些属性和方法可以遍历整个文档树。考虑到 HTML 文档的复杂性，DOM 定义了 nodeType 来表示结点的类型，以便于分类进行不同的操作和处理。

4.1.1 DOM 中相关属性和方法

在表 4-1 中列出了 DOM 中常用的几种结点类型。

表 4-1 常用的几种结点类型

接　口	nodeType 常量	nodeType 值	备　注
element	Node.ELEMENT_NODE	1	元素结点
text	Node.TEXT_NODE	3	文本结点
document	Node.DOCUMENT_NODE	9	document
comment	Node.COMMENT_NODE	8	注释的文本
documentFragment	Node.DOCUMENT_FRAGMENT_NODE	11	document 片断
attr	Node.ATTRIBUTE_NODE	2	结点属性

在 DOM 的树型结构中，其根结点是 document 对象，该对象的 documentElement 属性引用表示文档根元素的 element 对象（对于 HTML 文档，实际上就是<html>标记）。当使用 JavaScript 脚本语言操作 HTML 文档的时，document 即指向整个文档，<body>、<table>等结点类型即为 element，comment 类型的结点则是指文档的注释。

在使用 DOM 操作 HTML 文档时，经常使用 document 对象的相关方法，有关该对象的常用方法如表 4-2 所示。

表 4-2 document 对象的常用方法

方　法	描　述
createAttribute()	用指定的名字创建新的 attr 结点，即属性结点
createComment()	用指定的字符串创建新的 comment 结点
createElement()	用指定的标记名创建新的 element 结点

续表

方　法	描　述
createTextNode()	用指定的文本创建新的 textNode 结点
getElementById()	返回文档中具有指定 id 属性的 element 结点
getElementsByTagName()	返回文档中具有指定标记名的所有 element 结点

除了 document 对象对应的属性及方法比较常用之外，对于 element 结点，可以通过调用 getAttribute()、setAttribute()、removeAttribute()方法来查询、设置或者删除一个 element 结点的相关属性，例如<table>标记的 border 属性。

element 对象常用的属性是 tagName 元素，其含义为该对象对应的标记名称，比如<p>元素为 P，注意 HTML 文档返回的 tagName 均为大写。

表 4-3 中列出了 element 对象的常用方法。

表 4-3　　element 对象的常用方法

方　法	描　述
getAttribute()	以字符串形式返回指定属性的值
getAttributeNode()	以 attr 结点的形式返回指定属性的值
getElementsByTabName()	返回一个 Node 数组，包含具有指定标记名的所有 element 结点的子孙结点，其顺序为在文档中出现的顺序
hasAttribute()	如果该元素具有指定名字的属性，则返回 true
removeAttribute()	从元素中删除指定的属性
removeAttributeNode()	从元素的属性列表中删除指定的 attr 结点
setAttribute()	把指定的属性设置为指定的字符串值，如果该属性不存在则添加一个新属性
setAttributeNode()	把指定的 attr 结点添加到该元素的属性列表中

attr 对象代表文档元素的属性，有 name、value 等属性，可以通过 Node 接口的 attributes 属性或者调用 element 接口的 getAttributeNode()方法来获取。不过，在大多数情况下，使用 element 元素有关属性操作的最简单方法是 getAttribute()和 setAttribute()两个方法，而不是 attr 对象。

node 对象定义了一系列属性和方法，以便于遍历整个文档。用 parentNode 属性和 childNodes[]数组可以在文档树中上下移动；通过遍历 childNodes[]数组或者使用 firstChild 和 nextSibling 属性进行循环操作，也可以使用 lastChild 和 previousSibling 进行逆向循环操作，还可以根据需要枚举指定结点的子结点。而调用 appendChild()、insertBefore()、removeChild()、replaceChild()方法可以改变一个结点的子结点从而改变文档树。

需要注意的是，childNodes[]的值实际上是一个 NodeList 对象。因此，可以通过遍历

childNodes[]数组的每个元素，来枚举一个给定结点的所有子结点；通过递归，可以枚举树中所有结点。

表 4-4 中列出了 node 对象的一些常用属性。

表 4-4 node 对象的常用属性

属　性	描　述
attributes	如果该结点是一个 element，则以 NamedNodeMap 形式返回该元素的属性
childNodes	以 node[]的形式存放当前结点的子结点。如果没有子结点，则返回空数组
firstChild	以 node 的形式返回当前结点的第一个子结点。如果没有子结点，则为 null
lastChild	以 node 的形式返回当前结点的最后一个子结点。如果没有子结点，则为 null
nextSibling	以 node 的形式返回当前结点的兄弟下一个结点。如果没有这样的结点，则返回 null
nodeName	结点的名字，element 结点则代表 element 的标记名称
nodeType	代表结点的类型
setAttributeNode()	把指定的 attr 结点添加到该元素的属性列表中
parentNode	以 node 的形式返回当前结点的父结点。如果没有父结点，则为 null
previousSibling	以 node 的形式返回紧挨当前结点、位于它之前的兄弟结点。如果没有这样的结点，则返回 null

在使用 DOM 操作 HTML 文档时，会使用 node 对象的相关方法，通过这些方法实现对指定结点的具体操作。

有关 node 对象中常用的方法如表 4-5 所示。

表 4-5 node 对象的常用方法

方　法	描　述
appendChild()	通过把一个结点增加到当前结点的 childNodes[]组，给文档树增加结点
cloneNode()	复制当前结点，或者复制当前结点以及它的所有子孙结点
hasChildNodes()	如果当前结点拥有子结点，则将返回 true
insertBefore()	给文档树插入一个结点，位置在当前结点的指定子结点之前。如果该结点已经存在，则删除之再插入到它的位置
removeChild()	从文档树中删除并返回指定的子结点
replaceChild()	从文档树中删除并返回指定的子结点，用另一个结点替换它

了解了 DOM 中针对 HTML 文档操作提供的相关对象及对应的属性和方法后，通过两个实际的案例帮助读者体会有关 DOM 操作 HTML 文档的方法和技巧。

必知必会：innerHTML 属性

使用 HTML DOM 对象的 innerHTML 属性实现信息的动态显示，也可以使用 DOM 提供

的操作 HTML 文档相关方法和属性实现更加丰富的显示效果。

4.1.2 innerHTML 属性

该属性对应的是一个简单的串，表示一组开始标记和结束标记之间的内容。借助该属性可以实现响应的结果在对应元素位置的显示。下面通过一个实例进行说明。

在本节对应的项目“P41_HTMLDOM”中新建一个 HTML 文件，名称为“menu.html”。该程序实现的动态效果如图 4-2 所示。该页面实现的是简单级联菜单的效果，当用户单击主菜单中对应选项时，页面中将显示出下一级子菜单。

图 4-2 动态级联菜单的效果

对应页面的源代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <link href="images/css.css" type="text/css" rel="stylesheet">
</head>
<script language="javascript" type="text/javascript">
     // 创建级联菜单函数
     function showSubMenu(obj) {
          var currentSort =document.getElementById(obj);
          currentSort.parentNode.style.display = "";
          var subMenu;
          if (currentSort.id=="IBM"){
                 subMenu = "  IBM-T43";
          }
          else if (currentSort.id=="SONY"){
                 subMenu = "  SONY-888";
          }
          currentSort.innerHTML = subMenu;
     }

</script>
```

```
    <table style="BORDER-COLLAPSE: collapse" bordercolor="#111111" cellspacing="0"
cellpadding="0" width="200" bgcolor="#f5efe7" border="0">
          <tr>
               <td align="middle" height="4">
                    <img height="4" src="images/promo_list_top.gif" width="100%" border=
"0" alt="">
               </td>
          </tr>
          <tr>
               <td align="middle" bgcolor="#dbc2b0" height="19">
                    <b>笔记本品牌</b>
               </td>
          </tr>
          <tr>
               <td height="20">
                    <a onclick="showSubMenu('IBM')">IBM</a>
               </td>
          </tr>
          <tr style="display:none">
               <td height="20" id="IBM">
               </td>
          </tr>
          <tr>
               <td height="20">
                    <a onclick="showSubMenu('SONY')">SONY</a>
               </td>
          </tr>
          <tr style="display:none ">
               <td id="SONY" height="20">
               </td>
          </tr>
    </table>
```

在该页面中，当用户单击主菜单中的选项时，将调用“**showSubMenu('XXX')**”函数，在该函数中将首先获取所选择选项的标志信息，然后依据获取到的信息进行逻辑判断，构造子菜单对应的信息，最终使用 innerHTML 属性显示在对应的位置。

在本书第 5 章的案例中将会对本例进一步进行修改，实现依据服务器端数据库中的数据信息动态生成子菜单并显示的效果。

考虑到 innerHTML 属性并不是 HTML 元素的标准属性，因此在使用时有可能会受到一定的限制。下一节将介绍更加灵活的实现方式。

4.1.3 操作 HTML 文档

除了使用 innerHTML 属性方便地控制信息在对应位置动态显示外，为了更加灵活地控制 Web 页面的动态显示效果，往往需要借助 DOM 提供的方法和属性对 HTML 文档进行操作。

下面结合具体的案例进行介绍，在项目“P41_HTMLDOM”中新建一个 HTML 文件，名称为“dyUpdate.html”。该程序实现的动态效果如图 4-3 所示。该页面实现的是动态表格的效果，当用户在文本框中输入新增品牌名称，单击“确认”按钮，在下方的表格中直接显示出新增的信息，同时用户在动态表格中选择了对应的品牌，单击“删除”按钮则将从表格中删除对应的品牌信息。

图 4-3　动态级联菜单的效果

对应页面的源代码如下：

```
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <link href="images/css.css" type="text/css" rel="stylesheet">
</head>
<script language="javascript" type="text/javascript">

     // 增加品牌函数
     function addSort() {
        var name = document.getElementById("name").value;
        if(name == "" ) {
             return;
        }

        var row = document.createElement("tr");
        row.setAttribute("id", name);
        var cell = document.createElement("td");
        cell.appendChild(document.createTextNode(name));
        row.appendChild(cell);

        var deleteButton = document.createElement("input");
        deleteButton.setAttribute("type", "button");
```

```
            deleteButton.setAttribute("value", "删除");
            deleteButton.onclick = function () { deleteSort(name); };
            cell = document.createElement("td");
            cell.appendChild(deleteButton);
            row.appendChild(cell);

            document.getElementById("sortList").appendChild(row);
        //清空输入框
            document.getElementById("name").value = "";
        }

        // 删除品牌函数
        function deleteSort(id) {
            if (id!=null){
                var rowToDelete = document.getElementById(id);
                var sortList = document.getElementById("sortList");
                sortList.removeChild(rowToDelete);
            }
        }

    </script>

    <table style="BORDER-COLLAPSE: collapse" bordercolor="#111111" cellspacing="0"
cellpadding="2" width="400" bgcolor="#f5efe7" border="0">
        <tr>
            <td align="middle" height="4" colspan="3">
                <img height="4" src="images/promo_list_top.gif" width="100%" border=
"0" alt="">
            </td>
        </tr>
        <tr>
            <td align="middle" bgcolor="#dbc2b0" height="19" colspan="3">
                <b>品牌信息管理</b>
            </td>
        </tr>
        <tr>
            <td height="20">
                增加新品牌:
            </td>
            <td height="20">
                <input id="name" type="text" size="15">
            </td>
            <td height="20">
                <img src="images/ok.gif" onclick="addSort();" alt="">
            </td>

        </tr>
        <tr>
            <td height="20">
```

```
                品牌信息管理：
            </td>

        </tr>
        <table border="1" width="400">
            <tr>
                <td height="20" valign="top" align="center">
                    品牌名称：
                </td>
                <td id="pos_1" height="20">
                    操作
                </td>
            </tr>
            <tbody id="sortList"></tbody>
        </table>
    </table>
```

在该页面中当用户填写了新增品牌的名称，单击确认按钮之后，将调用“**addSort()**”函数，在该函数中将首先获取用户填写的新增品牌信息，然后依据获取到的信息，借助 DOM 操作 HTML 的方式生成动态的表格，对应的程序代码段如下：

```
var row = document.createElement("tr");
row.setAttribute("id", name);
var cell = document.createElement("td");
cell.appendChild(document.createTextNode(name));
row.appendChild(cell);

var deleteButton = document.createElement("input");
deleteButton.setAttribute("type", "button");
deleteButton.setAttribute("value", "删除");
deleteButton.onclick = function () { deleteSort(name); };
cell = document.createElement("td");
cell.appendChild(deleteButton);
row.appendChild(cell);

document.getElementById("sortList").appendChild(row);
```

可以看到，在对应代码中建立了表格中的行以及对应的单元格，同时装入了对应的文本内容及按钮。这里使用了元素、文本结点。借助“**setAttribute()**”方法设置元素相应的属性。

在本书第 5 章的案例中将进一步修改该页面，实现依据服务器端数据库中的数据信息动态生成表格的效果，同时，浏览器客户端的插入或者删除操作也将直接影响数据库中存储的数据信息。

4.2 解析 XML 文档

相对 HTML 文档而言，XML 文档的树型结构更加直观，在 DOM 对应的版本中，除了 HTML DOM 之外，XML DOM 也是很常用的。

使用 XML 文档时，需要解决的最关键问题是，如何方便地对 XML 树型结构中的结点进行遍历，XML DOM 提供了相关方法和属性来解决此问题。

必知必会：XML DOM 如何进行解析

由于 XML 将数据组织为一颗树，所以 DOM 就是对这颗树的一个对象描叙。通俗的说，就是通过解析 XML 文档，为 XML 文档在逻辑上建立一个树型结构，树的结点是一个个对象。我们通过存取这些对象就能够存取 XML 文档的内容。如图 4-4 所示 XML 文档在浏览器中显示出的树型结构，下面我们针对该 XML 文档进行解析。

下面例子介绍使用 JavaScript 脚本结合 DOM 对 XML 进行解析。

案例 4-2　解析 XML 文档

在 Eclipse 中新建一个项目，项目的名称为“P42_XMLDOM”。首先，新建一个 HTML 文档，页面名称为“readXML.jsp”。 该页面实现的效果如图 4-4 所示。用户单击“测试”按钮，页面中显示从对应 XML 文档中解析获得的数据信息。

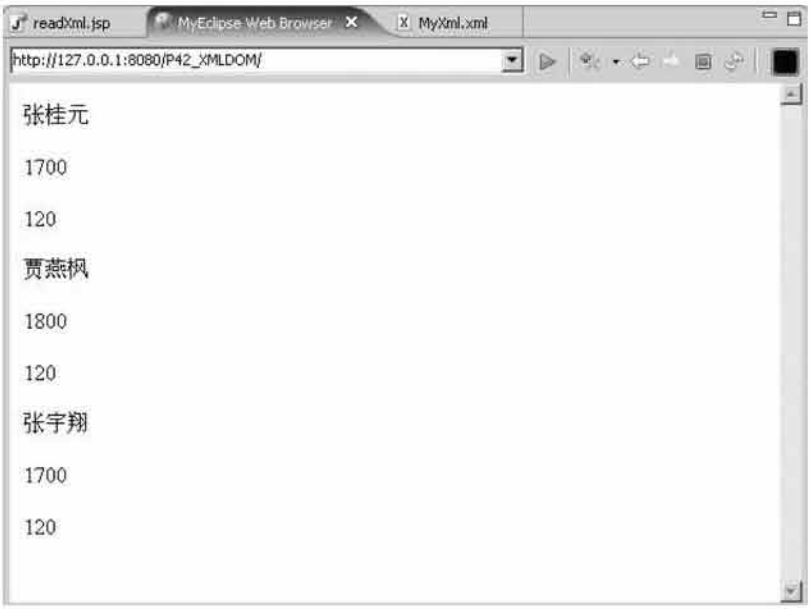


图 4-4　在 JavaScirpt 脚本中解析 XML 文档的效果

在该例中实现的是从 XML 文档中读取指定位置的数据信息，然后在 HTML 页面中进行显示，本例中所使用的 XML 文档的源代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<classmates>
     <student >
          <sid>1</sid>
```

```
            <sname>张桂元</sname>
            <gre>1700</gre>
            <tse>120</tse>
        </student>
        <student >
            <sid>1</sid>
            <sname>贾燕枫</sname>
            <gre>1800</gre>
            <tse>120</tse>
        </student>
        <student >
            <sid>1</sid>
            <sname>张宇翔</sname>
            <gre>1800</gre>
            <tse>120</tse>
        </student>
</classmates>
```

对该 XML 文档进行解析的 HTML 页面的源代码如下所示。在该页面中当用户单击“测试”按钮后，将调用“sendRequest()”函数。在该函数中将直接开始读取 MyXml.xml 文档中的信息。

```
<head>
    <META http-equiv=Content-Type content="text/html; charset=UTF-8">
    <LINK href="images/css.css" type=text/css rel=stylesheet>
</head>
<script language="javascript">
    var XMLHttpReq;
    //创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
        if(window.XMLHttpRequest) { //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
        }
        else if (window.ActiveXObject) { // IE 浏览器
            try {
                XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try {
                    XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
    }
    // 发送请求的函数
    function sendRequest() {
        var url = "MyXml.xml";
        createXMLHttpRequest();
        XMLHttpReq.onreadystatechange = processResponse;
        XMLHttpReq.open("GET", url, true);
        XMLHttpReq.send(null);
```

```
        }
        // 处理响应的函数
        function processResponse() {
        if (XMLHttpReq.readyState == 4) { // 判断对象状态
              if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                    readXml();
                } else { //页面不正常
                      window.alert("您所请求的页面有异常。");
                }
              }
        }
        // 读取 XML 文档中数据信息的函数，即解析函数
        function readXml() {
              var students = XMLHttpReq.responseXML.getElementsByTagName("student");
              for(var i=0;i<students.length;i++) {
                    var stud = students[i];
                    var name = stud.getElementsByTagName("sname")[0].firstChild.data;
                    var gre = stud.getElementsByTagName("gre")[0].firstChild.data;
                    var tse = stud.getElementsByTagName("tse")[0].firstChild.data;
                    document.write(name + "<p>");
                    document.write(gre + "<p>");
                    document.write(tse);
                    document.write("");
              }
        }

</script>

<table>
      <input type="button" value="测试" onclick="sendRequest();"/>
</table>
```

需要注意的是：在 Ajax 提交请求时，如果 URL 地址为文件名称，将开始直接操作对应的 XML 文档。在进行解析时，首先按照标记名称获取对应的结点数组，然后依据每一个数组元素继续获取子结点的相关信息，层层定位、层层解析。

4.3 Ajax 的响应处理

本书前面的案例中，已经初步介绍了 XML 文档的格式以及功能。已经了解到从服务器发回的响应数据既可以是文本串的格式，也可以采用 XML 文档的格式。当数据比较复杂时，往往需要使用 XML 结构化方式来表示数据。需要注意的是，比较适合选择 XML 的场合包括这样几种情况：不用考虑带宽和处理效率；或者需要实现与系统其他 API 或其他系统进行交互，即需要作为一种数据中转的中介；或者需要特定格式的输出效果。

但是如果本身需要处理的数据格式并不复杂，同时还需要考虑处理效率及速度等方面的

因素时，就不建议采用 XML 文档的格式。此时如果需要从服务器端返回响应数据就可以采用纯文本的形式。

4.3.1 responseText 属性

针对纯文本格式以及 XML 格式的响应数据，XMLHttpRequest 对象提供了对应的两个属性，一个是属性 responseText，即将响应提供为一个串。另一个属性为 responseXML，即将响应提供为一个 XML 对象。本节先介绍 responseText 属性。

案例 4-3　Ajax 的响应处理

在 Eclipse 中新建一个项目，项目的名称为“P43_Response”。首先，新建一个 HTML 文档，页面名称为“login.jsp”。 该页面实现的效果如图 4-5 所示。用户输入对应的登录信息，单击“登录”按钮，页面中将显示登录是否成功的提示信息。

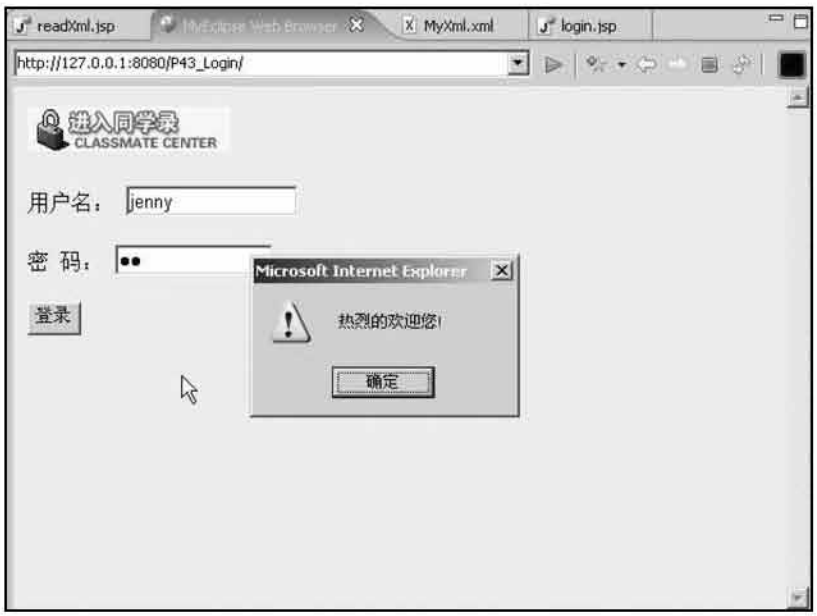


图 4-5　登录验证的效果

对应的 Web 页面的代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq = false;
     //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
```

```
        if(window.XMLHttpRequest) { //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
        }
        else if (window.ActiveXObject) { // IE 浏览器
            try {
                XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try {
                    XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
    }
    //发送请求函数
    function sendRequest(url) {
        createXMLHttpRequest();
        XMLHttpReq.open("GET", url, true);
        XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
        XMLHttpReq.send(null);  // 发送请求
    }
    // 处理返回信息函数
    function processResponse() {
    if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
            var res=XMLHttpReq.responseText;
              window.alert(res);
        } else { //页面不正常
            window.alert("您所请求的页面有异常。");
        }
      }
  }
  // 身份验证函数
  function userCheck() {
      var uname = document.myform.uname.value;
      var psw = document.myform.psw.value;
      if(uname=="") {
          window.alert("用户名不能为空。");
          document.myform.uname.focus();
          return false;
      }
      else {
          sendRequest('login?uname='+ uname + '&psw=' + psw);
       }
}

</script>

<body vLink="#006666" link="#003366" bgColor="#E0F0F8">
<img height="33" src="enter.gif" width="148">
```

```
<form action="" method="post" name="myform">
用户名： <input size="15" name="uname"><p>
密  码： <input type="password" size="15" name="psw"><p>
<input type="button" value="登录" onclick="userCheck()" >
</form>
```

在上面的页面中，当用户输入用户名及密码信息，单击“登录”按钮后，将调用“**userCheck()**”函数，在该函数中将首先获取用户填写的信息，进行最基本的数据有效性检查，如果检查通过，将借助 Ajax 发送请求，并等待服务器端的响应，一旦接收到服务器端的响应数据，则通过 **LHttpReq.responseText** 返回对应的数据信息，然后显示在提示窗口中。

该 Web 应用的配置文件 web.xml 对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“**login**”请求，服务器端类名为“**classmate.LoginAction**”的 Servlet 程序进行处理。

```
<?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>ms1</servlet-name>
        <servlet-class>classmate.LoginAction</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面关注一下服务器端 Servlet 程序 LoginAction.java 中对应的程序代码。当接收到浏览器端提交的请求后，Servlet 程序将首先获取浏览器端提交的用户名及密码信息，然后进行身份验证，本例中为了关注 Ajax，没有引入与数据库相关的操作。

```
package classmate;

import java.io.IOException;
……
public class LoginAction extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
```

```
    }

    /*
     *  处理<GET> 请求方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //接收浏览器端提交的信息
          String uname = request.getParameter("uname");
          String psw = request.getParameter("psw");
          //设置输出信息的格式及字符集
       response.setContentType("text/xml; charset=UTF-8");
       response.setHeader("Cache-Control", "no-cache");
       //创建输出流对象
       PrintWriter out = response.getWriter();
       //依据验证结果输出不同的数据信息
       if(uname.equals("jenny") && psw.equals("hi")){
            out.println("热烈的欢迎您!");
       }else{
            out.println("对不起,登录失败! ");
       }
       out.close();
    }
}
```

如果验证通过，将返回“热烈的欢迎您！”的提示，如果验证失败，将显示“对不起，登录失败！”的信息，信息是以 responseText 的格式返回客户端的。

4.3.2 responseXML 属性

使用属性 responseText 将响应提供为一个串。对于结构比较复杂的响应数据，可以借助另一个属性 responseXML，将响应提供为一个 XML 对象。在本书第 2 章的案例中使用的就是 responseXML 属性，将响应解析为 XML 文档。

这里回顾一下在案例 2-2 中对应的代码，在 login.jsp 页面中，解析返回的 XML 格式响应数据的函数代码如下：

```
// 处理返回信息函数
    function processResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
          if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
               var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].firstChild.data;
                    window.alert(res);
               } else { //页面不正常
                    window.alert("您所请求的页面有异常。");
               }
```

```
        }
    }
```

其中的 **XMLHttpReq.responseXML.getElementsByTagName("res")[0].firstChild.data** 就是在借助 responseXML 属性对返回的 XML 格式的数据进行解析。

4.4 Ajax 不同请求提交方式的处理

本书第 3 章中已经介绍过，借助 Ajax 提交请求，常用的有两种提交请求的方式，一种为“GET”，另外一种为“POST”。需要注意的是：如果采用 GET 方式，对于提交给服务器的信息大小是有限制的，采用 POST 方式，就没有大小容量上的限制。

4.4.1 采用 POST 方式提交请求

在使用 POST 方式提交请求时，对应的参数将作为请求体的一部分进行发送，这一点与 GET 方式不同。GET 方式是将参数作为 URL 的一部分追加在其后面。

前面的案例中，我们一直采用的是以 GET 方式提交请求，下面是采用 POST 方式提交请求的例子。

案例 4-4 Ajax 的请求处理

在 Eclipse 中新建一个项目，项目的名称为“P44_Request”。首先，新建一个 HTML 文档，页面名称为“login.jsp”。 本例中实现的效果与案例 4-3 相同，这里不再赘述，下面看一下具体的实现页面。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq = false;
     var uname;
     var psw;
     //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
          if(window.XMLHttpRequest) { //Mozilla 浏览器
               XMLHttpReq = new XMLHttpRequest();
          }
          else if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               } catch (e) {
                    try {
                         XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
```

```
                        } catch (e) {}
                    }
                }
            }
            //发送请求函数
            function sendRequest() {
                createXMLHttpRequest();
                XMLHttpReq.open("POST", "login", true);
                XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
            XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
                XMLHttpReq.send("uname=" + uname + "&psw=" + psw);  // 发送请求
            }
            // 处理返回信息函数
            function processResponse() {
            if (XMLHttpReq.readyState == 4) { // 判断对象状态
                if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                    var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].first
Child.data;
                        window.alert(res);
                } else { //页面不正常
                        window.alert("您所请求的页面有异常。");
                }
            }
            }
            // 身份验证函数
            function userCheck() {
                uname = document.myform.uname.value;
                psw = document.myform.psw.value;
                if(uname=="") {
                    window.alert("用户名不能为空。");
                    document.myform.uname.focus();
                    return false;
                }
                else {
                    sendRequest();
                }
            }

    </script>

    <body vLink="#006666" link="#003366" bgColor="#E0F0F8">
    <img height="33" src="enter.gif" width="148">
    <form action="" method="post" name="myform">
    用户名： <input size="15" name="uname"><p>
    密  码： <input type="password" size="15" name="psw"><p>
    <input type="button" value="登录" onclick="userCheck()" >
    </form>
```

可以看到，如果以 POST 方式提交请求，需要配合使用下面的代码完成正确的请求及参数的提交。

```
XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
XMLHttpReq.send("uname=" + uname + "&psw=" + psw);
```

其中第一条语句是为了确保服务器知道请求体中有请求参数，而最终是通过调用 send() 方法将串作为参数进行传递。服务器端的 Servlet 程序处理过程与前面的案例相同，不再赘述。

4.4.2 提交 XML 格式的请求参数

除了可以处理已经有的 XML 文档之外，在 Ajax 技术中还可以在向服务器端提交请求时，将请求的参数作为 XML 的格式进行发送。即将 XML 文档作为请求体的一部分发送到服务器，在服务器端可以从请求体中读取 XML，并加以处理。

下面我们继续对案例 4-4 中的 login.jsp 页面进行调整。调整后的页面代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq = false;
     var uname;
     var psw;
     //创建 XMLHttpRequest 对象
    function createXMLHttpRequest() {
          if(window.XMLHttpRequest) { //Mozilla 浏览器
               XMLHttpReq = new XMLHttpRequest();
          }
          else if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               } catch (e) {
                    try {
                         XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                    } catch (e) {}
               }
          }
     }
     //发送请求函数
     function sendRequest() {
          createXMLHttpRequest();
          var xml = createXML();
          window.alert(xml);
          XMLHttpReq.open("POST", "login", true);
          XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
     XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form- urlencoded");
          XMLHttpReq.send(xml);  // 发送请求
     }
     // 处理返回信息函数
     function processResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
```

```
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                var res=XMLHttpReq.responseText;
                  window.alert(res);
                } else { //页面不正常
                    window.alert("您所请求的页面有异常。");
                }
            }
        }
        // 身份验证函数
        function userCheck() {
              uname = document.myform.uname.value;
              psw = document.myform.psw.value;
              if(uname=="") {
                    window.alert("用户名不能为空。");
                    document.myform.uname.focus();
                    return false;
              }
              else {
                    sendRequest();
              }
        }
        //创建 XML
        function createXML() {
            var xml = "<user>";
            xml = xml + "<name>" +uname + "<\/name>";
            xml = xml + "<psw>" + psw + "<\/psw>";
            xml = xml + "<\/user>";
            return xml;
        }

    </script>

    <body vLink="#006666" link="#003366" bgColor="#E0F0F8">
    <img height="33" src="enter.gif" width="148">
    <form action="" method="post" name="myform">
    用户名： <input size="15" name="uname"><p>
    密  码： <input type="password" size="15" name="psw"><p>
    <input type="button" value="登录" onclick="userCheck()" >
    </form>
```

在上面的页面中，一旦开始调用函数进行请求的发送处理时，将会首先借助“createXML()”函数将获取到的用户名及密码信息封装成 XML 文档的格式。然后在采用 POST 方式将其作为参数提交过去。

下面介绍一下 XML 格式的数据提交到服务器端之后，如何解析的？对应的服务器端 Servlet 程序代码如下：

```
package classmate;
import java.io.BufferedReader;
……
```

```
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;


public class LoginXmlAction extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
    }

    /*
     * 处理<POST>请求方法
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //
        StringBuffer bxml = new StringBuffer();
        String line = null;
        try {
            BufferedReader reader = request.getReader();
            while((line = reader.readLine()) != null) {
                bxml.append(line);
            }
        }
        catch(Exception e) {
            System.out.println(e.toString());
        }

        String xml =bxml.toString();
        Document xmlDoc = null;
        try {
             xmlDoc =
                    DocumentBuilderFactory.newInstance().newDocumentBuilder()
                    .parse(new ByteArrayInputStream(xml.getBytes()));
        }
        catch(ParserConfigurationException e) {
            System.out.println(e);
        }
        catch(SAXException e) {
            System.out.println( e);
        }

String uname = xmlDoc.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
String psw = xmlDoc.getElementsByTagName("psw").item(0).getFirstChild().getNodeValue();
        String responseText;
        //依据验证结果输出不同的数据信息
```

```
            if(uname.equals("jenny") && psw.equals("hi")){
                responseText = "热烈的欢迎您!" ;
            }else{
                responseText = "对不起,登录失败! ";
            }
            //设置输出信息的格式及字符集
            response.setContentType("text/xml; charset=UTF-8");
            response.setHeader("Cache-Control", "no-cache");
            response.getWriter().print(responseText);

        }
    }
```

注意在 Java 程序中，为了实现对 XML 格式数据的解析，需要首先导入与其相关的类所在的包。然后借助 **DocumentBuilderFactory** 提供的相关方法进行解析。

首先，我们需要建立一个解析器工厂，利用这个工厂来获得一个具体的解析器对象：

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

当获得一个工厂对象后，使用它的静态方法 newDocumentBuilder()方法可以获得一个 DocumentBuilder 对象，这个对象代表了具体的 DOM 解析器。

然后，我们就可以利用这个解析器来对 XML 文档进行解析了：

```
Document doc = db.parse("c:/xml/message.xml");
```

DocumentBuilder 的 parse()方法接受一个 XML 文档名作为输入参数，返回一个 Document 对象，这个 Document 对象就代表了一个 XML 文档的树模型。以后所有的对 XML 文档的操作，都与解析器无关，直接在这个 Document 对象上进行操作就可以了。而具体对 Document 操作的方法，就是由 DOM 所定义的。

从上面得到的 Document 对象开始，就可以开始按照 DOM 的方式进行解析。使用 Document 对象的 getElementsByTagName()方法，可以得到一个 NodeList 对象，一个 Node 对象代表一个 XML 文档中一个标签元素，而 NodeList 对象，它所代表的是一个 Node 对象的列表。然后，可以使用 NodeList 对象的 item()方法来得到列表中每一个 Node 对象。当一个 Node 对象被建立之后，保存在 XML 文档中的数据就被提取出来并封装在这个 Node 中。然后再使用 getFirstChild()方法来获得对应标记下面的第一个子 Node 对象。依此类推就可以获取到对应的数据，开始使用了。

4.5 JavaScript 脚本技术深入

从目前的应用状况来看，JavaScript 脚本已经不再是很多程序员眼中简单的语言，充分使用脚本语言可以让 Web 页面的效果更加丰富。

在 Ajax 所推荐的系列效果中，拖拽效果和褪色技术一直是其中的热点。在本章的最后，我们再通过两个实例介绍一下如何使用 JavaScript 脚本实现这两个效果。

案例 4-5　页面中拖拽效果的实现

可拖放 DOM 模式（Draggable DOM pattern）可以让用户在 Web 页面中对各个部分进行编辑，即只需要选中要移动的部分，将其拖拽到新的位置上，就可以重新安排整个页面的布局效果，下面介绍一下具体的实现方式。

在 Eclipse 中新建一个项目，项目的名称为“P45_DropDrag”。首先，新建一个 HTML 文档，页面名称为“main.html”。 该页面实现的效果如图 4-6 所示。当用户使用鼠标选中对应的方框时，就可以通过拖拽调整其在页面中的位置。

图 4-6　拖拽效果

本例中所使用的 HTML 文档的源代码如下所示。注意在本页面中调用了一个开源框架中有关定位的 JavaScript 脚本文件，读者可以参考本书配套光盘中的源代码。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Main</title>
        <script language="javascript" src="dom-drag.js"></script>
        <link rel="stylesheet" href="style.css">
    </HEAD>
    <body>
        <form id="Form1" method="post" runat="server">
            <div id="news_root" style="LEFT:20px; TOP:20px" class="root">
                <div id="news_handle" class="handle">定制窗口<span style="TEXT-ALIGN:
right"></span></div>
                <div id="news" class="text">欢迎使用拖拽功能</div>
            </div>
        </form>
        <script language="javascript">
        // 初始化新闻面板的拖动效果
```

```
        var news_handle = document.getElementById("news_handle");
        var news_root   = document.getElementById("news_root");
        Drag.init(news_handle, news_root);
        news.style.backgroundColor = "#ffff00";
        news.style.cursor = "hand";
        </script>
    </body>
</HTML>
```

层叠样式表文件对应的代码如下：

```
.root
 {
     position:absolute;
     height:150px; width:200px;
     border:1px solid #333;
     BACKGROUND-COLOR: #eee
}
.handle
{
     margin:2px;
     padding:2px;
     width:194px;
     color:white;
     background-color:navy;
     font-family:verdana, sans-serif;
     font-size:12px;

}
.text
{
     color:navy;
     font-family:verdana, sans-serif;
     font-size:12px;

}
```

案例 4-6　褪色技术的实现

采用 Ajax 技术所实现的页面中，为了能够更加清楚地表现出页面中局部数据信息的改变，往往采用褪色技术对发生改变的部分进行特殊的标注，但这种标注会随着内容变旧逐步消失。下面介绍一下具体的实现方式。

在 Eclipse 中新建一个项目，项目的名称为“P46_Color”。首先，新建一个 HTML 文档，页面名称为“main.html”。该页面实现的效果如图 4-7 所示。对于页面中黄色部分显示出的数据会随着时间的推移慢慢消失。

在该例中实现的是从 XML 文档中读取指定位置的数据信息，然后在 HTML 页面中进行显示，本例中所使用的 XML 文档的源代码如下所示。在页面中主要使用了定时器动态控制

颜色的变化。

```
<HTML>
    <HEAD>
        <title>Main</title>
```

图 4-7 褪色效果

```
        <link rel="stylesheet" href="style.css">
    </HEAD>
    <body>
        <form id="Form1" method="post">
            <div id="news_root" style="LEFT:20px; TOP:20px" class="root">
                <div id="news_handle" class="handle">定制窗口 <span style=
"TEXT-ALIGN: right"></span></div>
                <div id="news" class="text">欢迎使用褪色技术</div>
            </div>
        </form>

        <script language="JavaScript">
        // 颜色渐变的速度
        var speed = 3;
        // 颜色渐变时需要使用的定时器对象数组
        var timers;

        news.style.backgroundColor = "#ffff00";
        news.style.cursor = "hand";
        timers = setTimeout("changeColor()", 100);


        function changeColor()
        {
            var color = news.style.backgroundColor;
            // 当前背景色(rgb)的 rg 部分:如#123456 的 1234
            var color_rg = color.slice(1, 5);
```

```
            // 当前背景色(rgb)的b部分:如#123456的56
            // 并将b部分增加一个数值,转换为10进制整数(向白色靠近)
            var color_b = parseInt(color.slice(5), 16) + speed;

            // 如果b部分不超过255
            if (color_b < 256)
            {
                // 求b的16进制形式
                var b1 = Math.floor((color_b / 16)).toString(16);
                var b2 = (color_b % 16).toString(16);

                // 设置新的背景色
                news.style.backgroundColor = "#" + color_rg + b1 + b2;

                timers = setTimeout("changeColor()", 100);
            }
            else //背景色已经是白色
            {
                // 停止计时器
                clearTimeout(timers);
            }
        }
        </script>
    </body>
</HTML>
```

本例中所使用的层叠样式表文件与前面相同，不再赘述。本例中并没有实现与服务器端数据库进行关联的效果，读者可以尝试在本例基础上进行调整，以动态获取数据库中的数据信息，采用褪色技术进行显示。

CHAPTER 5

第 5 章　Ajax 常用实现技巧

通过前面四章的介绍，相信读者已经掌握了 Ajax 核心技术的相关设计方法，如：核心对象 XMLHttpRequest，与 Ajax 相关的 JavaScript 脚本技术、DOM 文档对象化模型、以及用于存储、传送数据的 XML 文档，和控制显示输出的 CSS 及 XHTL。

为了能够让读者熟悉更加灵活的 Ajax 应用方案，本章将通过若干实际案例的实现过程帮助读者积累更多 Ajax 实战的经验，体会更多 Ajax 应用的魅力。本章将实现常见的表单数据验证、Web 页面中级联菜单的效果、动态加载列表框、自动刷新页面、Web 页面的局部动态更新以及自动完成功能等。相信丰富的实例一定可以帮助读者在短时间内体验 Ajax 的方便、灵活、人性化的交互方式。

5.1　实现表单数据验证

为保证数据的有效性，杜绝错误、无效的信息存储到数据库中，对于接收到的浏览器客户端提交的数据，需要进行基本的有效性、合理性的检查。在传统的 Web 应用中，客户端一般采用 JavaScript 脚本所声明的函数，对提交的表单数据进行验证，但是这种方式并不能解决所有的验证问题，例如，在进行用户信息注册时，客户端只能实现类似是否填写了必要的信息、长度是否满足需求等基本的有效性检查，但是对于所填写的用户信息是否已经被占用等诸如此类的检查，在客户端脚本中是没有办法实现的。

同时，如果按照传统的方式将此类验证逻辑放在服务器端实现，也会因为面临需要整个页面刷新的问题而使得问题不能以直观的方式迅速显示在用户面前。借助 Ajax 技术，在异步交互的前提下，调用服务器端事先编写好的验证逻辑可以很好地解决此问题。

下面，我们实现一个对注册用户信息进行验证的实例，该实例所实现的效果如图 5-1 所示。用户首先在表单中填写用户注册信息，当用户单击“注册”按钮后，将对用户所填写的信息进行验证，如果用户没有填写用户名，或者密码输入的不一致，都会在页面中显示对应的提示信息，此外，当用户所填写的用户名已经被占用时，Ajax 所调用的服务器端处理程序同样会返回提示信息。

图 5-1 表单数据验证的效果

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P51_SignUp，对应的浏览器端页面代码如下所示。在该页面中提供了对应的表单以供用户填写注册信息，当用户填写信息，单击“注册”按钮后，将调用 signUp()函数，在该函数中首先借助 DOM 获取对应表单元素中用户填写的数据信息，进行基本的有效性验证，如果用户没有填写用户名，将显示“用户名不能为空”的提示信息，如果用户输入的两遍密码不一致，将显示“两次输入密码不同”的提示信息。如果浏览器端的数据有效性检查通过，此后，再借助 Ajax 提交请求，并同时提交用户填写的信息，到服务器端，等待服务器端的处理。当服务器端后续处理完成后，将返回对应的响应信息，在浏览器客户端进行显示。

源文件：login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq;
     //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
          if (window.XMLHttpRequest) { //Mozilla 浏览器
               XMLHttpReq = new XMLHttpRequest();
          }
          else if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               } catch (e) {
                    try {
                         XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                    } catch (e) {}
               }
          }
     }
     //发送请求函数
     function sendRequest(url) {
          createXMLHttpRequest();
          XMLHttpReq.open("GET", url, true);
          XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
          XMLHttpReq.send(null);  // 发送请求
     }
     // 处理返回信息函数
     function processResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
          if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
              var res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].
firstChild.data;
                  window.alert(res);
```

```
                } else { //页面不正常
                    window.alert("您所请求的页面有异常。");
                }
            }
        }
        // 注册函数
        function signUp() {
            var uname = document.myform.uname.value;
            var psw = document.myform.psw.value;
            var psw2 = document.myform.psw2.value;
            if (uname=="") {
                window.alert("用户名不能为空。");
                document.myform.uname.focus();
                return false;
            }
            else if(psw!=psw2) {
                window.alert("两次输入密码不同。");
                document.myform.psw.focus();
                return false;
            }
            else {
                sendRequest('signUp?uname='+ uname + '&psw=' + psw);
            }
        }
    </script>

    <body vLink="#006666" link="#003366" bgColor="#E0F0F8">
    <img height="33" src="enter.gif" width="148">
    <form action="" method="post" name="myform">
    用户名： <input size="15" name="uname"><p>
    密  码： <input type="password" size="15" name="psw"><p>
    重输密码： <input type="password" size="20" name="psw2"><p>
    <input type="button" value="注册" onclick="signUp()" >
    </form>
```

该Web应用的配置文件web.xml对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“**signUp**”请求时，将由服务器端的类名为“**classmate.SignUpAction**”的Servlet程序进行处理。

```
<?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>ms1</servlet-name>
```

```
        <servlet-class>classmate.SignUpAction</servlet-class>
    </servlet>

    <servlet-mapping>
          <servlet-name>ms1</servlet-name>
          <url-pattern>/signUp</url-pattern>
    </servlet-mapping>

<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>signUp.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端 Servlet 程序 SignUpAction.java 中对应的程序代码。可以看到在接收到浏览器端提交的请求后，Servlet 程序将首先获取浏览器端提交的用户名及密码信息，然后借助封装了数据库操作的 JavaBean 完成依据数据库中存储的注册用户信息进行验证的目的。

```
package classmate;
……

public class SignUpAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
     }

    /*
     *  处理<GET> 请求方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String uname = request.getParameter("uname");
          String psw = request.getParameter("psw");
          //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");

        //数据库操作
        DB db = new DB();
         ResultSet rs;
         int insRes = 0;
```

```
            String strSql=null;
            //判断用户名是否重复
            strSql = "select * from classuser where username='"
                   + uname + "'";
             rs = db.executeQuery(strSql);
             boolean bnoRepeat = false;
             try {
                   if ( !rs.next()) {
                          bnoRepeat = true;
                   }
             } catch (SQLException e) {
                   e.printStackTrace();
             }
           //用户名不重复，插入记录
             if (bnoRepeat){
                 strSql = "Insert Into classuser values('"+ uname + "','" + psw + "')";
                   insRes = db. executeUpdate(strSql);
           }
           System.out.println(uname);
             if(!bnoRepeat){
                   out.println("<res>" + "注册失败！用户名已存在，重新输入用户名" +
"</res>");
             }else if(insRes>0){
                   out.println("<res>" + "注册成功！" + "</res>");
             }else{
                   out.println("<res>" + "注册失败！" + "</res>");
             }

             out.println("</response>");
             out.close();
       }
   }
```

在进行服务器端验证时，如果用户名没有被占用，会将对应的用户信息插入到数据库中去，并返回“注册成功”的信息，如果用户名已经被占用，将返回“注册失败！用户名已占用，请重新输入用户信息”的提示，其他原因的注册不成功，将显示“注册失败”的信息，信息都是以 XML 文档的格式返回客户端的。

采用 Ajax 技术，可以看到页面并没有进行完全的刷新，这无疑会给注册信息的用户带来更好的感受，同时数据信息的有效性问题也得到了很好的解决。

5.2 实现 Web 页面中级联菜单的设计

在 Web 页面中经常需要实现如图 5-2 所示的级联菜单的效果。即根据用户的选择，动态展开并显示出对应选项子菜单中的内容。本例中是根据用户所选择的商品类别信息，在子菜单中显示对应的商品名称。

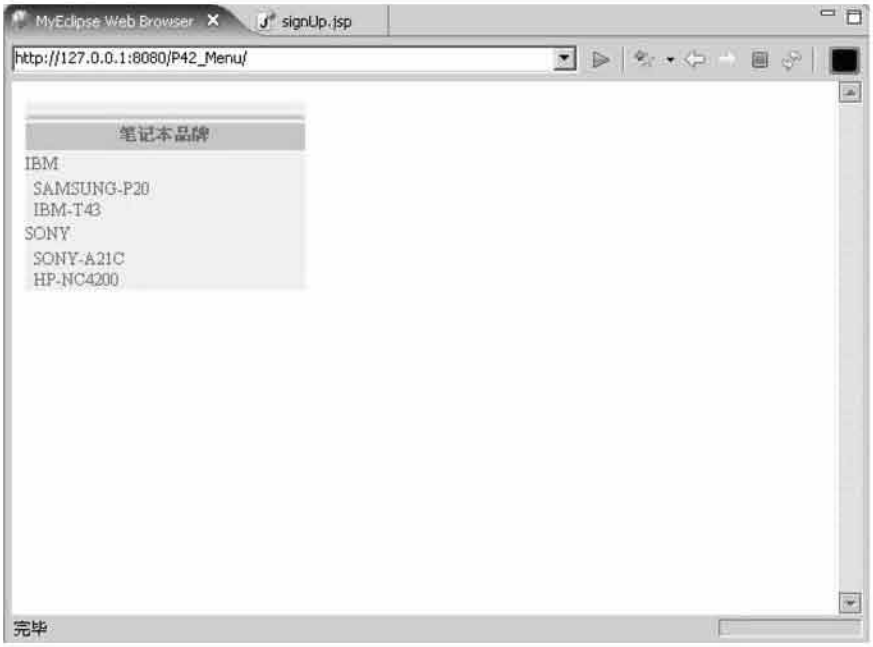


图 5-2　级联菜单的效果

在传统的 Web 实现方式中，一般是在页面初始化时动态获取到服务器端数据库中对应所有子菜单中的信息，放置页面中对应的位置，然后在结合 CSS 层叠样式表动态控制对应子菜单的显示或者隐藏。

采用这种方法的弊端在于会造成浏览器端下载的页面非常臃肿，许多根本用不到的信息也必须提前装载到页面中。

而如果借助 Ajax 技术，我们可以实现当用户选择对应的菜单项后，将对应的请求以异步方式提交到服务器端，然后有选择地从服务器端获取到对应的子菜单信息，再返回浏览器端进行响应显示。

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P52_Menu，对应的浏览器端页面代码如下：

源文件：menu.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
     <LINK href="images/css.css" type=text/css rel=stylesheet>
</head>
<script language="javascript">
     var XMLHttpReq;
     var currentSort;
     //创建 XMLHttpRequest 对象
```

```
function createXMLHttpRequest() {
      if(window.XMLHttpRequest) { //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
      }
      else if (window.ActiveXObject) { // IE浏览器
            try {
                  XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                  try {
                        XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                  } catch (e) {}
            }
      }
}
//发送请求函数
function sendRequest(url) {
      createXMLHttpRequest();
      XMLHttpReq.open("GET", url, true);
      XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
      XMLHttpReq.send(null);  // 发送请求
}
// 处理返回信息函数
function processResponse() {
if (XMLHttpReq.readyState == 4) { // 判断对象状态
      if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  updateMenu();
          } else{ //页面不正常
                  alert("您所请求的页面有异常。");
          }
      }
}
//更新菜单函数
function updateMenu() {
     var res=XMLHttpReq.responseXML.getElementsByTagName("res")
     var subMenu = "";
     for(var i = 0; i < res.length; i++) {
          subMenu = subMenu + "  " + res[i].firstChild.data + "<br>";
     }
     currentSort.innerHTML = subMenu;
}
// 创建级联菜单函数
function showSubMenu(obj) {
     currentSort =document.getElementById(obj);
     currentSort.parentNode.style.display = "";
     sendRequest("menu?sort=" + obj);
}

</script>
```

```
<table  style="BORDER-COLLAPSE: collapse" borderColor=#111111
              cellSpacing=0 cellPadding=0 width=200 bgColor=#f5efe7 border=0>
    <TR>
        <TD align=middle height=4><IMG height=4
        src="images/promo_list_top.gif" width="100%"
        border=0>
         </TD>
    </TR>
    <TR>
      <TD align=middle bgColor=#dbc2b0
                  height=19><B>笔记本品牌</B>
      </TD>
    </TR>
    <tr>
       <td height="20">
              <a onClick="showSubMenu('IBM')">IBM</a>
        </td>
   </tr>
   <tr style="display:none">
       <td height="20" id="IBM"> </td>
   </tr>
   <tr>
       <td height="20">
              <a onClick="showSubMenu('SONY')">SONY</a>
        </td>
   </tr>
   <tr style="display:none ">
      <td id="SONY" height="20"> </td>
   </tr>
</table>
```

在该页面中提供了对应的菜单以供用户进行选择，用户选择菜单后，将调用“**showSubMenu('XXX')**”函数，该函数带参数，其中参数用于传递所选菜单项的标志信息，以决定获取服务器端的哪个子菜单信息。在“**showSubMenu('XXX')**”函数中首先获取到所选菜单项的识别信息，然后再借助 Ajax 提交请求，并同时提交用户所选菜单的标识信息到服务器端，等待服务器端的处理。当服务器端处理完成后，将返回对应的子菜单信息，在 Ajax 提供的处理函数中对应菜单项的位置进行显示，这里用 innerHTML 属性实现定位显示。

该 Web 应用的配置文件 web.xml 对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“menu”请求时，将由服务器端的类名为“classmate.MenuAction”的 Servlet 程序进行处理。

```
<?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>ms1</servlet-name>
       <servlet-class>classmate.MenuAction</servlet-class>
   </servlet>

  <servlet-mapping>
       <servlet-name>ms1</servlet-name>
       <url-pattern>/menu</url-pattern>
   </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>menu.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端 Servlet 程序 MenuAction.java 中对应的程序代码。当接收到浏览器端提交的请求后，Servlet 程序将首先获取浏览器端提交的所选菜单项的标志信息，即所选择的商品类别信息，然后借助封装了数据库操作的 JavaBean 完成数据库的查询工作，依据类别获取到对应商品名称的信息。

```
package classmate;

import java.io.IOException;
……
public class MenuAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
     }

     /*
      *  处理<GET> 请求方法
      */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //接收浏览器端提交的信息
        String sort = request.getParameter("sort");
        //设置输出信息的格式及字符集
      response.setContentType("text/xml; charset=UTF-8");
      response.setHeader("Cache-Control", "no-cache");
      //创建输出流对象
      PrintWriter out = response.getWriter();
      //依据验证结果输出不同的数据信息
      out.println("<response>");
```

```
        //数据库操作
        DB db = new DB();
         ResultSet rs;
        String strSql=null;
        //查找该品牌具体型号
        strSql = "select product.name from sort,product where sort.id=product.sortid
                and sort.name='"+ sort + "'";
         rs = db.executeQuery(strSql);
         try {
              while (rs.next()) {
                    out.println("<res>" + rs.getString("name") + "</res>");
              }
         } catch (SQLException e) {
              e.printStackTrace();
         }

         out.println("</response>");
         out.close();
    }
}
```

注意在本例中，返回的 XML 文档的格式如下所示：

```
<response>
     <res>商品名称 1</ res>
     <res>商品名称 2</ res>
</response>
```

提示：读者可以进一步完善该程序，实现从服务器端动态获取菜单项信息，然后再依据用户选择动态显示子菜单信息的效果。这里所说的刷新，指的是通过重发 URL 请求来从服务器获得更新的数据以更新视图的显示。

5.3 实现动态加载列表框

本例中所实现的效果如图 5-3 所示。当用户在品牌选择的下拉列表框中选择了对应品牌后，该品牌下所有的商品信息将会动态出现在商品选择列表框中以供用户进一步选择。这种表单元素之间的联动效果在网页中非常常用。例如，随着所选专业，显示班级选择列表。随着所选部门，显示职务选择列表等。

传统的 Web 实现方式，页面初始化时准备好所有列表框中的信息，并通过逻辑控制建立列表框之间对应选项的联系。再依据用户的选择，通过逻辑判断将事先准备好的列表框信息装入对应的联动列表框中。

这种方法的弊端是造成浏览器端下载的页面非常臃肿，许多在本次交互中根本用不到的信息也必须提前装载到页面中。

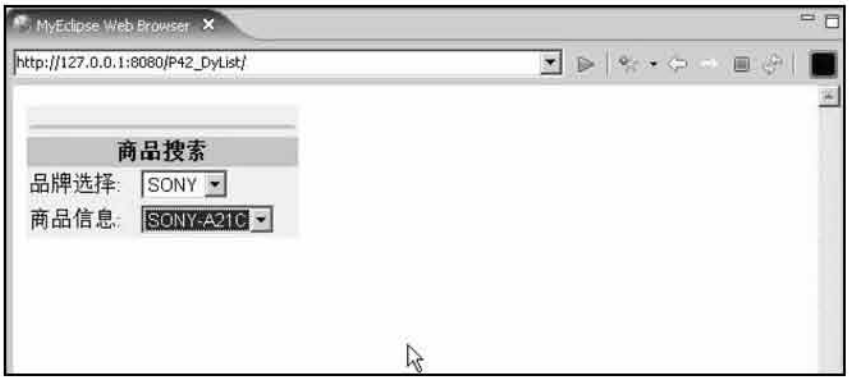


图 5-3 动态加载列表框的效果

借助 Ajax 技术，可以实现当用户选择第一个列表框中对应的选项后，将对应的请求以异步方式提交到服务器端，然后有选择地从服务器端获取到下一个列表框中的列表信息，再返回浏览器端进行响应显示，下面介绍一下具体的实现过程。

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P53_DyList，对应的浏览器端页面代码如下：

源文件：dyList.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
    <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
     var XMLHttpReq;
     //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
           if(window.XMLHttpRequest) { //Mozilla 浏览器
                 XMLHttpReq = new XMLHttpRequest();
           }
           else if (window.ActiveXObject) { // IE 浏览器
                 try {
                       XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                 } catch (e) {
                       try {
                             XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                       } catch (e) {}
                 }
           }
     }
     //发送请求函数
     function sendRequest(url) {
           createXMLHttpRequest();
           XMLHttpReq.open("GET", url, true);
```

```
                XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
                XMLHttpReq.send(null);  // 发送请求
            }
            // 处理返回信息函数
            function processResponse() {
            if (XMLHttpReq.readyState == 4) { // 判断对象状态
                  if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                        updateList();
                  } else { //页面不正常
                        window.alert("您所请求的页面有异常。");
                  }
               }
        }
        // 刷新列表框函数
        function refreshList() {
            var sort = document.getElementById("sort").value;
            if(sort == "" ) {
                  clearList();
                  return;
            }
            var url = "dyList?sort=" + sort;
            sendRequest(url)
        }
        // 更新列表框中列表项函数
        function updateList() {
            clearList();
            var product = document.getElementById("product");
            var results = XMLHttpReq.responseXML.getElementsByTagName("name");
            var option = null;

            for(var i = 0; i < results.length; i++) {
                  option = document.createElement("option");
                  option.appendChild(document.createTextNode(results[i].firstChild.
nodeValue));
                  product.appendChild(option);
            }
        }
        // 清除列表框中原有选项的函数
        function clearList() {
            var product = document.getElementById("product");
            while(product.childNodes.length > 0) {
                product.removeChild(product.childNodes[0]);
            }
        }

    </script>

    <table  style="BORDER-COLLAPSE: collapse" borderColor=#111111
              cellSpacing=0 cellPadding=2 width=200 bgColor=#f5efe7 border=0>
```

```
<TR>
    <TD align=middle height=4 colspan="2"><IMG height=4
    src="images/promo_list_top.gif" width="100%"
    border=0>
     </TD>
</TR>
<TR>
  <TD align=middle bgColor=#dbc2b0
              height=19 colspan="2"><B>商品搜索</B>
  </TD>
</TR>
<tr>
   <td height="20">
          品牌选择:
    </td>
    <td height="20">
          <select id="sort" onchange="refreshList();">
               <option value="default">请选择</option>
               <option value="IBM">IBM</option>
               <option value="SONY">SONY</option>
               <option value="联想">联想</option>
          </select>
     </td>

</tr>
 <tr>
     <td height="20">
          商品信息:
     </td>
     <td height="20">
     <select id="product" >
     </select>
     </td>
 </tr>

</table>
```

在该页面中，第一个列表框中提供了对应的品牌选项以供用户进行选择，当用户选择了所需的品牌信息后，将调用“**refreshList()**”函数，在该函数中首先获取到所选列表项的标志信息，如果用户选择的是“请选择”，则调用“**clearList()**”函数清除第二个列表框中的内容，如果用户选择的是有效的品牌选项，则借助 Ajax 提交请求，并同时提交用户所选选项的标识信息到服务器端，等待服务器端的处理。当服务器端后续处理完成后，将返回第二个列表框中的信息，然后在 Ajax 提供的处理函数中在对应的列表框中显示，在显示时使用了本书第 3 章中介绍的使用 DOM 操作 HTML 的方式。

该 Web 应用的配置文件 web.xml 对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“dyList”请求时，将由服务器端的类名为“classmate.DyListAction”的 Servlet

程序进行处理。

```
<?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>ms1</servlet-name>
        <servlet-class>classmate.DyListAction</servlet-class>
    </servlet>

    <servlet-mapping>
         <servlet-name>ms1</servlet-name>
         <url-pattern>/dyList</url-pattern>
     </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>dyList.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端 Servlet 程序 DyListAction.java 中对应的程序代码。当接收到浏览器端提交的请求后，Servlet 程序首先获取浏览器端提交的所选列表项的标志信息，即所选择的商品类别信息，然后借助封装了数据库操作的 JavaBean 完成数据库的查询工作，依据类别获取到对应的商品名称信息。

```
package classmate;

import java.io.IOException;
……

public class DyListAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
     }

    /*
     *  处理<GET> 请求方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String sort = request.getParameter("sort");
```

```
        //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");

        //数据库操作
        DB db = new DB();
         ResultSet rs;
        String strSql=null;
        //查找该品牌具体型号
        strSql ="select product.name,product.id from sort,product where sort.id=
                product.sortid and sort.name='"+ sort + "'";
         rs = db.executeQuery(strSql);
         try {
              while (rs.next()) {
                    out.println("<name>" + rs.getString("name") + "</name>");
              }
         } catch (SQLException e) {
              e.printStackTrace();
         }

         out.println("</response>");
         out.close();
    }
}
```

注意在本例中，返回的 XML 文档的格式如下所示：

```
<response>
      <name>商品名称 1</ name>
      <name>商品名称 2</name>
</response>
```

读者可以进一步完善该程序，实现从服务器端动态获取第一个列表框中的信息，然后再依据用户选择动态显示第二个列表框中的信息。

5.4 实现自动刷新页面

本例实现页面自动刷新的效果，该页面的内容如图 5-4 所示，在该页面中将根据数据库中存储的最新数据信息更新页面中热卖商品的信息，但是对这些信息的修改并不会导致整个页面的刷新。

实际的 Web 应用中，诸如：天气预报、直播比赛以及股市行情等页面往往都需要定期进行自动刷新。

传统的 Web 实现方式中，要想实现类似的效果必须进行整个页面的刷新，在网络速度受

到一定限制的情况下，这种因为一个局部变动而牵动整个页面的处理方式显得有些得不偿失。

图 5-4 自动刷新页面的效果

借助 Ajax 技术，可以实现对页面中局部区域的动态刷新，使得用户能够以更好的方式获得最新的数据信息，下面介绍一下自动刷新页面的具体实现过程。

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P54_AutoRefresh，对应的浏览器端页面代码如下：

源文件：aotuRefresh.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
     <META http-equiv=Content-Type content="text/html; charset=UTF-8">
</head>
<script language="javascript">
      var XMLHttpReq;
      //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
           if(window.XMLHttpRequest) { //Mozilla 浏览器
                 XMLHttpReq = new XMLHttpRequest();
           }
           else if (window.ActiveXObject) { // IE 浏览器
                 try {
                       XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                 }catch(e){
                       try {
                          XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                       } catch (e) {}
                 }
           }
      }
      //发送请求函数
      function sendRequest() {
           createXMLHttpRequest();
          var url = "refresh";
           XMLHttpReq.open("GET", url, true);
           XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
           XMLHttpReq.send(null);  // 发送请求
      }
```

```
        // 处理返回信息函数
        function processResponse() {
        if (XMLHttpReq.readyState == 4) { // 判断对象状态
             if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                      DisplayHot();
                      setTimeout("sendRequest()", 1000);
              } else { //页面不正常
                   window.alert("您所请求的页面有异常。");
              }
          }
        }
        // 显示更新数据信息的函数
        function DisplayHot() {
            var name = XMLHttpReq.responseXML.getElementsByTagName("name")[0].
firstChild.nodeValue;
            var count = XMLHttpReq.responseXML.getElementsByTagName("count")[0].
firstChild.nodeValue;
            document.getElementById("product").innerHTML = name;
            document.getElementById("count").innerHTML = count;
        }

    </script>

    </SCRIPT>
    <body onload =sendRequest()>
    <table  style="BORDER-COLLAPSE: collapse" borderColor=#111111
                cellSpacing=0 cellPadding=0 width=200 bgColor=#f5efe7 border=0>
       <TR>
           <TD align=middle height=4 colspan="2"><IMG height=4
           src="images/promo_list_top.gif" width="100%"
           border=0>
            </TD>
        </TR>
        <TR>
         <TD align=middle bgColor=#dbc2b0
                     height=19 colspan="2"><B>正在热卖的笔记本</B>
         </TD>
       </TR>
       <tr>
          <td height="20">
                  型号：
           </td>
          <td height="20" id="product">
           </td>
       </tr>
       <tr>
          <td height="20">
                  销售数量：
           </td>
```

```
            <td height="20" id="count">
            </td>
        </tr>
    </body>
    </table>
```

在该页面中在第一次进行页面装载时，将调用“**sendRequest()**”函数，在该函数中将借助 Ajax 提交请求，等待服务器端的处理。当服务器端后续处理完成后，将返回最新热卖商品的信息，然后在 Ajax 提供的处理函数中调用“**DisplayHot()**”函数在页面的对应位置进行显示，在进行显示时使用了本书第 3 章中介绍的使用 DOM 操作 HTML 的方式，同时使用了对应元素的 innerHTML 属性。此外，为了能够实现页面的自动刷新，在“**sendRequest()**”函数的最后设置了定时器，即在 1000ms 之后再次调用本函数，执行上面所有的处理工作，从而真正实现页面的自动刷新。注意本例中所使用的 **setTimeout ("sendRequest()", 1000)**函数是 HTML DOM 中 window 对象提供的设置定时器的方法。

该 Web 应用的配置文件 web.xml 对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“**refresh**”请求时，将由服务器端的类名为“**classmate.AutoRefreshAction**”的 Servlet 程序进行处理。

```
<?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>ms1</servlet-name>
        <servlet-class>classmate.AutoRefreshAction</servlet-class>
    </servlet>

   <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/refresh</url-pattern>
    </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>autoRefresh.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端 Servlet 程序 AutoRefreshAction.java 中对应的程序代码。当接收到浏览器端提交的请求后，借助封装了数据库操作的 JavaBean 完成数据库的查询工作，获取最新热卖的商品信息进行响应。

```
package classmate;

import java.io.IOException;
……

public class AutoRefreshAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
    }

    /*
     *  处理<GET> 请求方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
          //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");

       //数据库操作
       DB db = new DB();
        ResultSet rs;
       String strSql=null;
       //查询热卖产品
       strSql = "select name,salecount from product order by salecount desc";
        rs = db.executeQuery(strSql);
        try {
             if ( rs.next()) {
                    out.println("<name>" + rs.getString("name") + "</name>");
                    out.println("<count>" + rs.getString("salecount") + "</count>");
             }
        } catch (SQLException e) {
             e.printStackTrace();
        }

        out.println("</response>");
        out.close();
    }
}
```

注意在本例中，返回的 XML 文档的格式如下所示：

```
<response>
     <name>热卖商品名称</ name>
     <count>热卖数量</count>
</response>
```

由于数据库中商品的销售数量一直在发生改变，因此，本例中实现的原则是通过页面自动刷新定期获取数据库中销售数量最高的商品信息，在浏览器端的页面中进行显示。

5.5 实现 Web 页面的局部动态更新

本例中实现的是页面中局部动态更新的效果，对应页面的显示效果如图 5-5 所示，在该页面中实现的相当于商品信息的后台维护页面，在该页面中用户可以动态增加新的品牌信息到数据库中，增加后的品牌信息将会直接在动态表格中进行显示。同时也可以借助动态表格中的删除按钮进行品牌信息的删除操作。同样在本例中对这些信息的修改并不会导致整个页面的刷新。

图 5-5 Web 页面的局部刷新效果

传统的 Web 实现方式中，实现类似的效果必须进行整个页面的刷新。

借助 Ajax 技术，可以实现对页面中局部区域的动态刷新，使得用户能够以更好的方式获得最新的数据信息，下面介绍一下具体的实现过程。

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P55_DyUpdate，对应的浏览器端页面代码如下：

源文件：dyUpdate.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<head>
    <META http-equiv=Content-Type content="text/html; charset=UTF-8">
    <LINK href="images/css.css" type=text/css rel=stylesheet>
</head>
<script language="javascript">
     var XMLHttpReq;
```

```
var PREFIX = "Sort";
//创建 XMLHttpRequest 对象
function createXMLHttpRequest() {
      if(window.XMLHttpRequest) { //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
      }
      else if (window.ActiveXObject) { // IE 浏览器
            try {
                  XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e){
                  try{
                        XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                  } catch (e) {}
            }
      }
}
// 处理增加品牌响应函数
function AddStateChange() {
 if (XMLHttpReq.readyState == 4) { // 判断对象状态
       if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
             AddSortList();
         }else { //页面不正常
             window.alert("您所请求的页面有异常。");
         }
       }
}
// 增加品牌函数
function addSort() {
    name = document.getElementById("name").value;
    if(name == "" ) {
         return;
    }
    var url = "dyUpdate?action=add" + "&name="+ name;
    createXMLHttpRequest();
    XMLHttpReq.onreadystatechange = AddStateChange;
    XMLHttpReq.open("GET", url, true);
    XMLHttpReq.send(null);
}

function AddSortList() {
    var AddID = XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.
nodeValue;
    if (AddID!=null){
          var row = document.createElement("tr");
          row.setAttribute("id", PREFIX + AddID);
          var cell = document.createElement("td");
          cell.appendChild(document.createTextNode(name));
          row.appendChild(cell);
```

```
                var deleteButton = document.createElement("input");
                deleteButton.setAttribute("type", "button");
                deleteButton.setAttribute("value", "删除");
                deleteButton.onclick = function () { deleteSort(AddID); };
                cell = document.createElement("td");
                cell.appendChild(deleteButton);
                row.appendChild(cell);

                document.getElementById("sortList").appendChild(row);
            //清空输入框
                document.getElementById("name").value = "";
            }
        }

        // 删除品牌函数
        function deleteSort(id) {
            var url = "dyUpdate?action=delete" + "&id=" + id;
            createXMLHttpRequest();
            XMLHttpReq.onreadystatechange = DeleteStateChange;
            XMLHttpReq.open("GET", url, true);
            XMLHttpReq.send(null);
        }
        // 处理删除品牌响应函数
        function DeleteStateChange() {
        if(XMLHttpReq.readyState == 4) { // 判断对象状态
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  deleteSortList();
               }else { //页面不正常
                    window.alert("您所请求的页面有异常。");
               }
            }
        }

        function deleteSortList() {
        deleteID = XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.
nodeValue;
             if (deleteID!=null){
                  var rowToDelete = document.getElementById(PREFIX + deleteID);
                  var sortList = document.getElementById("sortList");
                  sortList.removeChild(rowToDelete);
             }
        }


    </script>

    <table  style="BORDER-COLLAPSE: collapse" borderColor=#111111
                cellSpacing=0 cellPadding=2 width=400 bgColor=#f5efe7 border=0>
```

```
<TR>
    <TD align=middle height=4 colspan="3"><IMG height=4
    src="images/promo_list_top.gif" width="100%"
    border=0>
     </TD>
</TR>
<TR>
   <TD align=middle bgColor=#dbc2b0
               height=19 colspan="3"><B>品牌信息管理</B>
   </TD>
</TR>
<tr>
    <td height="20">
          增加新品牌:
    </td>
    <td height="20">
          <input id="name" type="text"    size="15">
    </td>
    <td height="20">
          <img src="images/ok.gif" onclick="addSort();">
    </td>

</tr>
 <tr>
     <td height="20">
          品牌信息管理:
     </td>

 </tr>
<table border="1" width="400">
 <tr>
     <td height="20" valign="top" align="center">
          品牌名称:
     </td>
    <td id="pos_1" height="20">
          操作
     </td>
 </tr>
 <tbody id="sortList"></tbody>
</table>
</table>
```

在该页面中当用户在文本框中输入新品牌的名称，单击“提交”按钮后，将调用“**addSort()**”函数，在该函数中首先获取到用户输入的新品牌信息，借助 Ajax 提交请求，同时提交文本框中用户填写的新的品牌信息，请求的格式为**" dyUpdate?action=add" + "&name="+ name**，可以看到，为了表明请求的类型，使用了参数 action，其值为“add”，等待服务器端的处理。当服务器端后续处理完成后，将返回最新的商品品牌信息列表，然后在 Ajax 提供的处理函数中调用“**AddSortList()**”函数以动态表格的方式在页面的

对应位置进行显示，在进行显示时使用了本书第3章中介绍的使用DOM操作HTML的方式。

此外，当用户在动态表格中选择了对应品牌，单击“操作”按钮后，将调用“**deleteSort()**”，该函数带参数，参数用于传递待操作的品牌标志信息，在“**deleteSort()**”函数中将提交请求，同时提交品牌标志信息到服务器端，请求的格式为**"dyUpdate?action=delete" + "&id=" + id**，可以看到，为了表明请求的类型，使用了参数action，其值为“delete”。等待服务器端的处理。当服务器端后续处理完成后，将调用“**deleteSortList()**”函数借助DOM操作HTML方式从动态表格中去除对应的品牌信息。

该Web应用的配置文件web.xml对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“**dyUpdate**”请求时，将由服务器端的类名为“**classmate.DyUpdateAction**”的Servlet程序进行处理。

```
<?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>ms1</servlet-name>
        <servlet-class>classmate.DyUpdateAction</servlet-class>
    </servlet>

   <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/dyUpdate</url-pattern>
    </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>dyUpdate.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端Servlet程序DyUpdateAction.java中对应的程序代码。当接收到浏览器端提交的请求后，首先获取请求的类型及相关的数据信息，然后借助封装了数据库操作的JavaBean完成数据库的操作，如果是添加新品牌信息的请求，则向数据库中进行记录插入，否则进行记录删除操作。

```
package classmate;

import java.io.IOException;
```

```
……

public class DyUpdateAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
    }

    /*
     *  处理<GET> 请求方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //接收浏览器端提交的信息
         String action = request.getParameter("action");
         String name = request.getParameter("name");
         String id = request.getParameter("id");
         //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");

        //数据库操作
        DB db = new DB();
         ResultSet rs;
   String strSql=null;
         int insRes = 0;
        if ("add".equals(action)){
              //判断用户名是否重复
              strSql = "select max(id) from sort";
               rs = db.executeQuery(strSql);
               int iMaxId=0;
               try {
                    if ( rs.next()) {
                            iMaxId=rs.getInt(1)+1;
                    }
                    else {
                            iMaxId=1;
                    }
               } catch (SQLException e) {
                    e.printStackTrace();
               }
               if ( iMaxId>0 ){
                    strSql = "insert into sort values('"
                     + iMaxId      +"','"
```

```
                        + name    +"')";
                    insRes = db. executeUpdate(strSql);
                }
                if(insRes>0){
                      out.println("<id>" + iMaxId + "</id>");
                      out.println("<name>" + name + "</name>");
                }
            }
            else if ("delete".equals(action)){
                  strSql = "delete from sort where id= " + id;
                   insRes = db. executeUpdate(strSql);
                   if(insRes>0){
                         out.println("<id>" + id + "</id>");
                   }
            }
             out.println("</response>");
             out.close();
        }
}
```

在本例中，完成添加新品牌的请求处理之后，返回的 XML 文档的格式如下：

```
<response>
     <id>新品牌 id</ id>
     <name>新品牌名称</name>
</response>
```

完成品牌删除的请求处理之后，返回的 XML 文档的格式如下：

```
<response>
     <id>删除品牌 id</ id>
</response>
```

5.6 实现自动完成功能

本章实现的最后一个案例效果如图 5-6 所示。该页面实现的效果对读者并不陌生，我们在著名的搜索网站 Google 中已经非常熟悉自动完成功能的效果。本例中当用户在文本框中输入待查询的商品名称时，页面中将根据用户输入的文字信息，及时给出提示列表，以帮助用户快速进行选择，该提示信息来源于服务器端数据库中的数据。当用户选择了对应商品名称，单击“搜索”按钮后，将会在文本区域中显示商品相关的描述信息。

首先在 Eclipse 中新建一个 Web 项目，项目名称为 P56_AutoComplete，对应的浏览器端页面代码如下：

源文件：autoComplete.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<HTML><HEAD><TITLE>Shopping Online</TITLE>
    <META http-equiv=Content-Type content="text/html; charset=UTF-8">
    <LINK href="images/css.css" type=text/css rel=stylesheet>
```

图 5-6　自动完成的效果

```
</HEAD>
<script language="javascript">
     var XMLHttpReq;
     var completeDiv;
     var inputField;
     var completeTable;
     var completeBody;


      //创建 XMLHttpRequest 对象
     function createXMLHttpRequest() {
           if(window.XMLHttpRequest) { //Mozilla 浏览器
                 XMLHttpReq = new XMLHttpRequest();
           }
           else if (window.ActiveXObject) { // IE 浏览器
                 try {
                       XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                 } catch(e){
                       try{
                           XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                       } catch (e) {}
                 }
           }
     }
     //发送匹配请求函数
     function findNames() {
         inputField = document.getElementById("names");
         completeTable = document.getElementById("complete_table");
         completeDiv = document.getElementById("popup");
         completeBody = document.getElementById("complete_body");
```

```
    if (inputField.value.length > 0) {
          createXMLHttpRequest();
          var url = "autoComplete?action=match&names=" + escape(inputField.value);
          XMLHttpReq.open("GET", url, true);
          XMLHttpReq.onreadystatechange = processMatchResponse;//指定响应函数
          XMLHttpReq.send(null); // 发送请求
    } else {
          clearNames();
    }
}
 // 处理返回匹配信息函数
function processMatchResponse() {
 if (XMLHttpReq.readyState == 4) { // 判断对象状态
      if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  setNames(XMLHttpReq.responseXML.getElementsByTagName("res"));
          }else { //页面不正常
               window.alert("您所请求的页面有异常。");
          }
      }
}
 //生成与输入内容匹配行
      function setNames(names) {
         clearNames();
         var size = names.length;
         setOffsets();

         var row, cell, txtNode;
         for (var i = 0; i < size; i++) {
               var nextNode = names[i].firstChild.data;
               row = document.createElement("tr");
               cell = document.createElement("td");

               cell.onmouseout = function() {this.className='mouseOver';};
               cell.onmouseover = function() {this.className='mouseOut';};
               cell.setAttribute("bgcolor", "#FFFAFA");
               cell.setAttribute("border", "0");
               cell.onclick = function() { completeField(this); } ;

               txtNode = document.createTextNode(nextNode);
               cell.appendChild(txtNode);
               row.appendChild(cell);
               completeBody.appendChild(row);
         }
      }
  //设置显示位置
      function setOffsets() {
          completeTable.style.width = inputField.offsetWidth; + "px";
          var left = calculateOffset(inputField, "offsetLeft");
          var top = calculateOffset(inputField, "offsetTop") + inputField.
```

```
offsetHeight;
                completeDiv.style.border = "black 1px solid";
                completeDiv.style.left = left + "px";
                completeDiv.style.top = top + "px";
            }
        //计算显示位置
        function calculateOffset(field, attr) {
          var offset = 0;
          while(field) {
            offset += field[attr];
            field = field.offsetParent;
          }
          return offset;
        }
        //填写输入框
        function completeField(cell) {
            inputField.value = cell.firstChild.nodeValue;
            clearNames();
        }
        //清除自动完成行
        function clearNames() {
            var ind = completeBody.childNodes.length;
            for (var i = ind - 1; i >= 0 ; i--) {
                  completeBody.removeChild(completeBody.childNodes[i]);
            }
            completeDiv.style.border = "none";
        }
        //搜索请求函数
        function search() {
            var sortName = document.getElementById("names");
            createXMLHttpRequest();
            var url = "autoComplete?action=search&names=" + escape(inputField.value);
            XMLHttpReq.open("GET", url, true);
            XMLHttpReq.onreadystatechange = processSearchResponse;//指定响应函数
            XMLHttpReq.send(null); // 发送请求
        }
        // 处理返回匹配信息函数
        function processSearchResponse() {
        if (XMLHttpReq.readyState == 4) { // 判断对象状态
              if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                    var res=XMLHttpReq.responseXML.getElementsByTagName("res");
                    if (res.length>0){
                         document.getElementById("content").value=res[0].firstChild.data;
                    }
                }else { //页面不正常
                    window.alert("您所请求的页面有异常。");
                }
            }
        }
```

```
</script>


<table  style="BORDER-COLLAPSE: collapse" borderColor=#111111
              cellSpacing=0 cellPadding=2 width=400 bgColor=#f5efe7 border=0>
    <TR>
        <TD align=middle height=4 colspan="3"><IMG height=4
        src="images/promo_list_top.gif" width="100%"
        border=0>
         </TD>
    </TR>
    <TR>
      <TD align=middle bgColor=#dbc2b0
                  height=19 colspan="3"><B>商品信息搜索</B>
      </TD>
    </TR>
    <tr>
        <td height="20">
               输入品牌关键字:
         </td>
         <td height="20">
               <input type="text"  size="15" id="names" onkeyup="findNames();" style=
                "height:20;">
         <div style="position:absolute;" id="popup">
              <table id="complete_table" bgcolor="#FFFAFA" border="0" cellspacing=
              "0" cellpadding="0"/>
                  <tbody id="complete_body"></tbody>
              </table>
         </div>
         </td>


         <td height="20">
               <img src="images/search.gif" onclick="search();">
         </td>

    </tr>
    <tr>
         <td height="20" valign="top" align="center">
               产品描述:
         </td>
        <td id="pos_1" height="80">
               <textarea id="content">

               </textarea>
         </td>
    </tr>

</table>
```

在该页面中一旦用户开始在文本框中输入待查询商品的名称，即触发“onkeyup”事件，调用“**findNames()**”函数，在该函数中首先获取到用户已经在文本框中输入的信息，然后借助 Ajax 提交请求，同时提交文本框中用户已经填写的信息，请求的格式为" **autoComplete?action=match&names=" + escape(inputField.value)**，可以看到，为了表明请求的类型，使用了参数 action，其值为“match”，等待服务器端的处理。当服务器端后续处理完成后，将返回获取到的与用户已输入信息相匹配的信息列表，在 Ajax 提供的处理函数中调用“**setNames(names)**”函数以动态方式在页面的对应位置进行显示，在进行显示时还调用了“**setOffsets()**”函数以及“**calculateOffsets()**”函数。

此外，当用户录入完待查询的商品名称，单击“搜索”按钮后，将调用“**search()**”，该函数将首先获取待查询的商品名称信息，然后提交请求，同时提交待查询商品信息到服务器端，请求的格式为" **autoComplete?action=search&names=" + escape(inputField.value)**，可以看到，为了表明请求的类型，使用了参数 action，其值为“search”。等待服务器端的处理。当服务器端后续处理完成后，将调用“**processSearchResponse()**”函数在对应的文本区域中显示查询的商品描述信息。

该 Web 应用的配置文件 web.xml 对应的代码如下所示。从该配置文件中可以了解到，当浏览器端提交“autoComplete”请求时，将由服务器端的类名为“classmate.AutoCompleteAction”的 Servlet 程序进行处理。

```
<?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>ms1</servlet-name>
        <servlet-class>classmate.AutoCompleteAction</servlet-class>
    </servlet>

   <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/autoComplete</url-pattern>
    </servlet-mapping>


<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>autoComplete.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

下面我们关注一下服务器端 Servlet 程序 AutoCompleteAction.java 中对应的程序代码。当

接收到浏览器端提交的请求后，首先获取请求的类型及相关的数据信息，然后借助封装了数据库操作的 JavaBean 完成数据库的操作，例如：获取候选商品名称信息，或者查询指定商品的描述信息等。

```
package classmate;

import java.io.IOException;
……
public class AutoCompleteAction extends HttpServlet {

     public void init(ServletConfig config) throws ServletException {
     }

……
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String action = request.getParameter("action");
          String name = request.getParameter("names");
          //设置输出信息的格式及字符集
       response.setContentType("text/xml; charset=UTF-8");
       response.setHeader("Cache-Control", "no-cache");
       //创建输出流对象
       PrintWriter out = response.getWriter();
       //依据验证结果输出不同的数据信息
       out.println("<response>");

       //数据库操作
       DB db = new DB();
        ResultSet rs;
       String strSql=null;
       //匹配
       if ("match".equals(action)){
            strSql = "select * from product where name like'" + name + "%'";
        rs = db.executeQuery(strSql);
        try {
             while(rs.next()) {
                  out.println("<res>" + rs.getString("name") + "</res>");
             }
        }catch (SQLException e) {
             e.printStackTrace();
        }
       }
       else if ("search".equals(action)){
            strSql = "select contents from product where name ='" + name + "'";
        rs = db.executeQuery(strSql);
        try {
```

```
                if ( rs.next()) {
                        out.println("<res>" + rs.getString("contents") + "</res>");
                }
            }catch (SQLException e) {
                e.printStackTrace();
            }

           }
            out.println("</response>");
            out.close();
        }
    }
```

在本例中，完成 match 类型的请求处理之后，返回的 XML 文档的格式如下所示：

```
<response>
      <res>候选商品名称 1</ res>
      <res>候选商品名称 1</ res>
<res>候选商品名称 1</ res>
</response>
```

完成 search 类型的请求处理之后，返回的 XML 文档的格式如下所示：

```
<response>
      <res>商品描述信息</ res>
</response>
```

至此，本章相关的案例介绍完毕，通过本章的学习，读者对 Ajax 的实际应用方式和技巧都应有了一定程度的掌握和理解。

CHAPTER 6

第 6 章　Blog Online 网站概述及设计方案

通过本书第一部分的介绍，读者已经对 Ajax 技术在 Web 应用设计的开发有了全面的了解。为了更好的实际运用 Ajax 技术，第二部分将讲解一个电子商务网站开发的全部过程，从而进一步强化 Ajax 技术在 Java Web 应用开发的技术应用及使用技巧，在该项目的实现过程中，读者可以进一步体会到目前几个主流的设计模式、开发架构及数据库系统，如：MVC 设计模式、工厂设计模式、Hibernate 及 MySQL 数据库系统等。

该商务系统网站是一个充分体现 Web2.0 理念的博客网站，与论坛不同的是，博客给了网络用户更多的自主权，类似于以前的个人网站，但比个人网站管理维护起来更加方便，也更容易在网络上实现信息的共享和交流。作为一个博客网站，首先需要提供的基本功能包括新博客的注册、登录验证，以及热门博客及热门文章的推荐，还应该包括按照博客名称或者文章标题关键字的搜索功能。

同时为了能够更好地体现出在博客网站中由博客自主决定页面内容、页面显示效果以及方便地对相关内容实现维护管理等特点，博客网站还应该给每个博客提供管理自己博客页面的功能，包括文章信息（即日志）的维护，文章分类的维护，友情链接的维护及个人基本资料等信息的维护。

这些功能都是在博客网站中通用的模块，因此结合本书第二部分案例的设计和实现过程，读者将完整体验博客网站平台开发的各个环节，并且能够将各个功能模块方便地移植到实际的博客系统中。通过这个案例的实战，读者对于 Web2.0，包括博客概念本身，以及 Ajax 技术的应用方式等都会有更深入的理解。

本章按照电子商务平台开发的实际流程，对整个网站的设计实现进行总体描述，包括整体方案设计、运行架构的选择、功能模块的划分及用户界面设计等。有关数据模型的确立将在具体功能模块实现章节中进行介绍。

6.1　Blog Online 博客网站总体描述

越来越多的网络用户希望能够在网络平台上更多地展现自己的个性，更方便地与他人互动交流，在传统的 Web1.0 时代，无论是论坛、社区还是个人网站，都试图在这些方面进行

努力，随着 Web2.0 时代的到来，一个新的概念出现了——博客。

6.1.1 项目背景

商业网站设计的主要目的一般是通过网站的推广，实现更多网络用户的关注，从而吸引更多的企业用户投入更多的宣传经费到网站上。

长期的战略目标，商业网站不仅是公共产品信息和服务的推广，它还是将买家与卖家、厂商和合作伙伴紧密结合的平台，借助这些商业网站可以消除企业与客户之间时间与空间带来的障碍。

Blog Online 博客网站致力于为广大博客提供优质博客页面服务的商业网站。每个博客都希望借助自己的博客页面宣传自己，而博客数量越多，网站的点击率越高就越能够吸引广大的企业客户选择该商业网站作为媒介，将自己的产品展现给客户。可以说，对这些博客网站而言：为博客提供良好的服务就意味着为网站带来更多的商业客户。因此，在具体设计实现该博客网站时，主要考虑了主流博客网站的几个主要功能。

- 博客的注册、登录验证功能
- 网络用户通过关键字搜索博客以及文章功能
- 最热门博客页面推荐浏览
- 最新文章推荐浏览
- 文章详细内容及相关评论显示
- 博客页面访问量统计
- 博客个人文章管理维护功能
- 博客个人文章分类管理维护功能
- 博客个人友情链接维护功能
- 博客个人基本信息管理维护功能

Blog Online 博客网站的核心是通过为博客提供优质的互动交流平台，提高网站的知名度和访问量，从而获得为企业提供更多产品介绍及展示的机会，提升自己网站的价值。

6.1.2 网站开发基本流程

必知必会：网站开发的基本流程

一个商务网站项目的开发过程一般可分为站点策划、设计制作、动态编程及测试发布 4 个阶段，各阶段主要工作任务如下。

1. 站点策划

根据站点的具体形象和功能需求，在收集各方面信息的基础之上，确定整个站点的层次结构。对所有需表达的信息内容进行合理的编排，使用户可以方便地找到需要的信息，尽量使网站访问者可以在较短的时间内查找到最需要的信息，这一过程类似于软件项目开发过程

中的需求分析和系统设计阶段。

2. 设计制作

这一阶段主要是指静态网页的设计制作，一般包括构思草图、图形图像的制作、 网页效果图制作及页面制作等。在网站内容与结构都已经确定的前提下，由专业网页制作人员使用专业的网页设计软件（例如 FrontPage、DreamWeaver 等）进行主页、表格等内容的设计和制作。图 6-1 所示为某房地产公司网站的首页在设计制作阶段的草图。

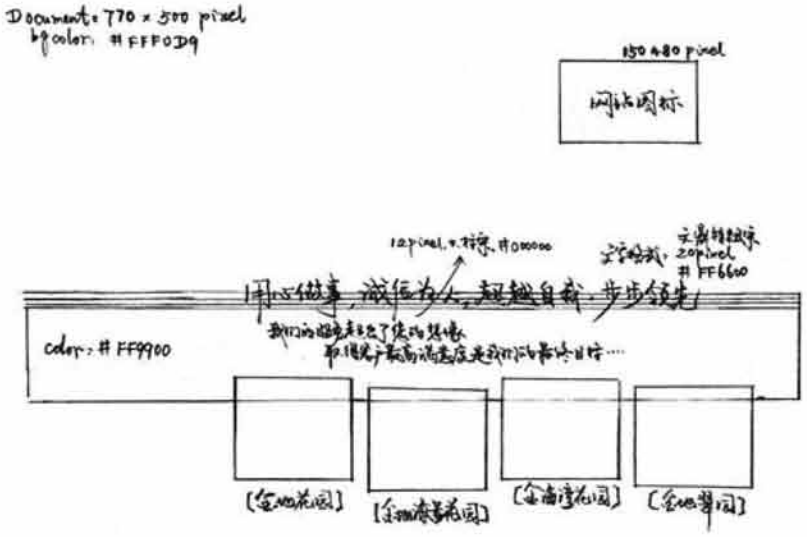


图 6-1 网页设计草图

3. 动态编程

根据网站动态功能的需求，确定各个动态部分的功能模块、设计数据库、编写程序代码和测试运行。该阶段主要由网站程序员负责一般使用 JSP、Servlet 等进行程序的设计与开发。

4. 测试发布

在网站系统工程师的管理与维护下，对外发布整个网站。在测试运行的过程中，及时发现相关的技术问题和错误，并接受相关信息反馈。对整个站点的页面进行改进和完善。

本书后面章节主要介绍 Blog Online 购物在线网站后台动态网站编程部分的具体设计实现过程。

6.2 方案设计

根据软件开发过程中客户实际的需求，除了进行代码编写工作之外，一般会有专业的软件系统分析师和设计师确定出整个软件开发项目的整体架构，包括系统的选型、运行环境的确定、功能模块的划分及用户界面的设计。

6.2.1 设计原则

必知必会：系统的选型

在进行软件系统开发的最初环节，一般都需要进行系统的选型，即根据系统功能的实际需求，选择合适的开发工具及软件架构。

Blog Online 博客网站对系统的可靠性、稳定性有比较高的要求。本系统设计时，比较流行的 B/S 设计有基于 JSP、ASP、PHP、CGI 及 J2EE 等模式。相比较而言 PHP 的功能相对简单，不适合做大程序；而 CGI 效率相对较低，所以也不考虑。

因为该系统并没有原有的基础平台需要扩展，也不需要与其他系统进行太多的交互，所以使用 J2EE 的模式并不能够体现出 J2EE 本身的优势。而 JSP 又是 J2EE 的核心技术之一，可以随时升级为 J2EE 程序。这里暂时不考虑用 J2EE。只需要在 ASP 和 JSP 中进行选择。

必知必会：ASP 与 JSP 的各自优势

项目组对 ASP 和 JSP 进行了全面的比较，具体比较数据如表 6-1 所示。

表 6-1 ASP 与 JSP 的对比

	ASP	JSP
Web 服务器	微软的 IIS（Internet Information Server）或 PWS（Personal Web Server）	任何 WEB 服务器包括 Apache、Netscape 和 IIS
操作系统	微软的视窗系统	绝大多数流行平台，包括 Solaris 操作系统、微软的视窗系统、MAC OS、Linux，及其他 UNIX 系列平台产品
跨平台访问	需要第三方 ASP 的引入产品	支持 Web 信息机构环境中不同系列的计算机群，即保证用户在当前软硬件及人力资源上的投资完全兼容，JSP 技术提供灵活、开放选择可以使用提供商提供的各种各样的工具，高度体现工业化标准输入与配置

在进行了诸多因素的比较之后，项目组最终认为，采用 JSP 是目前这个阶段比较合适的选择，而选择 MVC 设计模式作为整个 Web 应用的开发模式，选用 Hibernate 作为数据持续性处理层，则是考虑到了其高速的开发效率，及代码重用性高、易于维护等各种优势。其最终目的是希望提高系统底层业务逻辑的可重用性，增加系统的可扩展性，降低系统维护成本。当然，为了能够给博客提供更加人性化、更加周到的交互体验，在本系统的适当功能选择使用了 Ajax 技术与 MVC 设计模式中其他组件的结合。

6.2.2 运行架构

为了增加系统的吞吐量，提高并发处理客户请求数量，系统采用了 IBM 服务器作为主机。在数据库处理方面，不需要在数据层借助存储过程及数据库服务器端函数封装过多的业务逻辑，因此数据库系统采用相对精巧的 MySQL。

该在线购物系统服务器端如果需要布置到其他主机上，则该主机必备条件如下。

- 服务器端操作系统：独立于操作系统的跨平台系统，客户端 MicroSoft Windows 2000 及以上；
- 数据库：MySQL 4.1 版本；
- Web 服务器：Tomcat 5.0.19 及以上版本，配合 MVC 设计模式及 Hibernate 开发架构；
- 客户端运行环境：能运行 IE 5 以上或 Netscape 5 以上浏览器的操作系统，配合使用 Ajax 技术；
- 客户端运行工具：目前的系统采用浏览器作为客户端，为了支持 Ajax 开发框架，应该选择使用 IE 5 以上版本浏览器。

本商务网站系统以 NT 为 Web 平台，JSP+Ajax+Servlet+JavaBean+Hibernate 为网站实现技术，建立基于 MySQL 数据库系统的核心动态网页，实现博客网站前台及博客个人维护管理等功能模块。

本系统表示层均采用完全符合 HTML 4.0 、XHTML 及 XML 的页面实现技术，也采用了对应优化页面的代码和图片技术。

6.3 模块划分及界面设计

在完成了系统选型，确定系统开发运行环境之后，就需要进一步明确该系统的功能模块及基本的界面设计需求。

6.3.1 模块划分

博客网站一般可以分为面向网络用户的前台及面向博客个人维护管理的后台，依据博客网站中所需要实现的基本功能的设定来进行功能模块的划分。表 6-2 给出的是从客户需求中确定出的本博客网站各栏目的功能设置。

表 6-2　　Blog Online 博客网站栏目设置

栏　目	说　明
首页	首页中需要提供以下几个操作界面：热门博客页面推荐显示、最新博客文章推荐显示、博客的注册登录界面、按照关键字搜索对应文章的界面
博客注册登录管理	实现博客的注册及登录管理的功能
博客及文章搜索	提供热门博客推荐、最新文章推荐及按照文章标题关键字进行搜索等功能
博客个人主页	提供博客文章分类显示、相关评论显示及友情链接推荐、页面访问量统计显示等功能
博客文章管理	提供博客对自己的文章及文章分类进行增加、删除等操作的功能
博客友情链接管理	提供博客对自己的友情链接进行增加、删除等操作的功能
博客个人资料管理	提供博客对自己的个人信息进行维护修改等操作的功能

根据上表中列出的各个栏目的相关设置，可以确定出 Blog Online 博客网站如下的各个功能模块。

1. 博客注册登录管理模块

博客注册登录管理模块用于建立博客网站固定的客户群体，通过记录对应的博客档案，实现对博客信息的后台维护及管理，同时也便于通过博客档案库将网站最新动态及相关企业的信息方便地传达给每一位潜在的客户。

该功能模块能够提供以下几个子功能。

- 新博客在线注册；
- 博客登录管理；
- 跳转到博客主页。

只有进行登录并通过身份验证的博客，才可以在个人的博客页面发表文章，并借助个人设置实现对个人博客相关信息的管理维护。对于没有经过身份验证的网络用户不允许在博客页面中发表文章，更不允许对博客页面信息进行管理维护。

2. 博客及文章检索查询模块

博客及文章检索查询模块为网络用户提供便捷的搜索，以及文章阅读浏览等功能，同时对文章的评论信息博客推荐也能够及时反馈给网络用户。

该模块应该能够在网页中随时提供在线的最新文章信息。该信息需要定期更新，网络用户可以随时获得最新的文章以及最热门的博客推荐。在客户选择了某个博客或者某个感兴趣的文章后，可以方便地跳转到对应博客页面进行文章的阅读，并和博客进行交流互动。

为了使网络用户尽快定位到所需的博客资料以及文章信息，本模块提供了搜索功能，用户可以对所关注的文章信息按照标题进行关键字搜索，以避免用户浏览多个页面来寻找所需的文章信息。

该功能模块应该能够提供以下几个子功能。

- 热门博客页面推荐；
- 最新博客文章推荐；
- 文章信息关键字搜索。

3. 博客页面显示模块

当网络用户或者博客进入到某一个博客页面后，在该页面中将提供博客文章的显示，同时为了能够方便查看，在该模块中提供了依据分类进行文章内容的显示，提供评论查看，此外还应该允许网络用户在博客页面中发表评论、查看友情链接，允许博客在自己的页面中发表新文章等功能。

在该模块中还提供了博客页面访问量统计，文章及评论信息分页显示等方便用户的显示

效果。

该功能模块提供如下几个子功能。

- 用户可以分页查看对应的文章内容及评论信息；
- 用户可以针对文章内容发表评论；
- 用户可以分类查看文章内容；
- 用户可以借助提供的友情链接进行跳转。

4. 博客页面个人维护管理模块

博客页面个人维护管理模块用于实现博客对自己个人页面及相关信息的动态管理。

借助该模块，博客自己可以随时对自己页面中的内容进行增加修改，包括文章分类信息的更新、友情链接栏目中内容的增加等功能，也允许博客对自己的个人信息进行维护及管理。

该功能模块提供如下几个子功能：

- 文章及文章分类管理；
- 友情链接管理；
- 个人基本信息维护管理。

上面只是对各个模块的主要功能概要性的描述，具体各功能模块的划分及实现过程，将在本书后续的章节中逐一详细介绍。由于篇幅所限，本书只选择了博客网站中最核心的几个功能模块，相关扩展功能模块的工作都是类似的，有兴趣的读者可以在本书案例的基础上继续进行完善和修改。

6.3.2 界面设计

软件系统开发的关键在于系统设计，用户界面设计的好坏直接决定用户对系统的评价，因此良好的用户界面设计是系统成功的必要条件。

图 6-2 为 Blog Online 博客网站页面层次结构图。

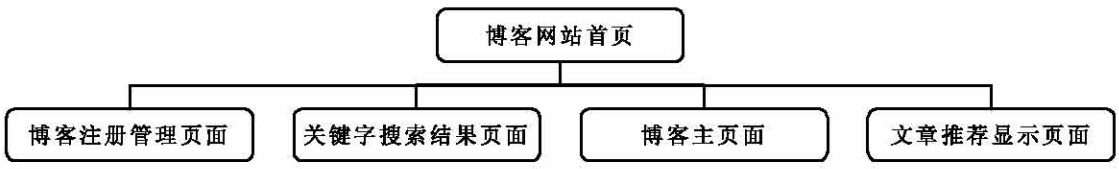


图 6-2 页面层次结构图

该层次结构图表现出了首页和各一级页面之间的链接关系，一级页面与相关的各二级页面之间的链接关系将在本书后续的章节中详细介绍。

Blog Online 博客网站面向网络用户的首页以及主要链接页面的效果如图 6-3～6-8 所示，

该网站中博客主页面采用的是框架页面的结构，对应页面的效果及其功能将在后面的章节中详细介绍。

图 6-3 所示为博客网站首页，本页面主要功能是提供后续各页面的相关跳转。包括用户注册、关键字查询、热门博客推荐及最新文章推荐等相关链接。

表 6-3 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-3　　首页页面设计说明

界 面 标 识	A1	模 板 文 件	index.htm
界面尺寸	1024*768		
界面功能	系统的首页，提供向各个子栏目页面的跳转		
备注	同时显示最热门博客推荐及最新文章推荐，用户可以在单击超链接后跳转到对应博客页面，同时还提供关键字搜索界面		
界面图题	图 6-3 所示		

图 6-3　Blog Online 博客网站首页

图 6-4 为按照文章标题关键字搜索文章的显示页面。该页面是在首页搜索文本框中输入待搜索文章标题关键字之后跳转到搜索结果页面，通过本页面提供的超级链接可以继续跳转到对应的博客页面。

表 6-4 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-4　　按照关键字搜索博客或者文章的显示页面说明

界 面 标 识	A2	模 板 文 件	searchResult.htm
界面尺寸	1024*768		
界面功能	首页搜索文本框中输入待搜索文章标题关键字之后跳转到搜索结果页面		
备注	通过本页面提供的超级链接可以继续跳转到对应的博客页面		
界面图题	图 6-4 所示		

图 6-4　搜索博客或者文章的显示页面

图 6-5 为博客个人主页的显示效果，本页面主要功能是显示博客个性化的信息，该页面为框架页面。

表 6-5 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-5 博客个人首页显示页面说明

界 面 标 识	A3	模 板 文 件	blogMain.htm
界面尺寸	1024*768		
界面功能	博客个人首页的信息显示效果，本页面主要功能是显示该博客个性化的信息		
备注	该页面同时作为在网站首页中，单击热门博客推荐的超链接以及最新文章推荐中的博客超链接页面，该页面为框架页面		
界面图题	图 6-5 所示		

图 6-5 博客个人首页显示页面

图 6-6 为文章详细信息及相关评论显示页面，是在如图 6-5 所示的博客个人首页中，单击对应文章查看评论超链接跳转页面。本页面主要功能是提供对应文章信息，包括相关评论显示的功能。在该页面中，用户还可以针对该文章发表评论。同时，当用户在首页最新文章推荐中，单击文章标题超链接，或者在搜索结果页面中，单击文章标题也会跳转到该页面，该页面同样是框架页面。

表 6-6 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-6　　文章详细信息及相关评论显示页面说明

界 面 标 识	A4	模 板 文 件	otherBlogMain.htm
界面尺寸	1024*768		
界面功能	文章详细信息及相关评论显示页面，是在博客首页中，单击对应文章，查看评论超链接跳转到的页面。本页面主要功能是，提供对应文章信息，包括相关评论显示的功能。在该页面中，用户还可以针对该文章发表评论		
备注	当用户在首页最新文章推荐中，单击文章标题的超链接，或者在搜索结果页面中，单击文章标题也会跳转到该页面，该页面同样是框架页面		
界面图题	图 6-6 所示		

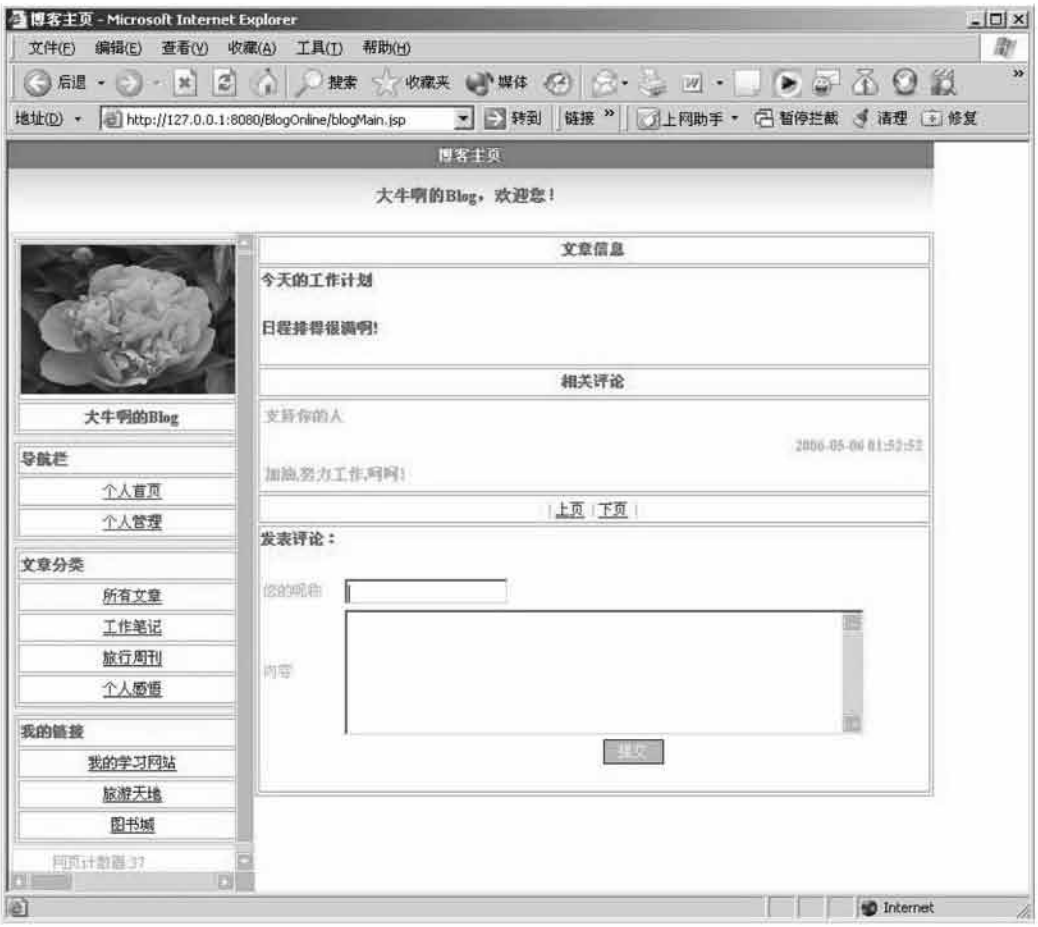


图 6-6　文章详细信息及相关评论显示页面

图 6-7 为在如图 6-5 所示的博客个人首页中，单击发表文章超链接跳转到的页面。本页面主要功能是提供发表文章的界面，本页面是作为框架页面的一部分显示在博客首页右下方

框架中的。

表 6-7 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-7　　发表文章页面说明

界 面 标 识	A5	模 板 文 件	newArticle.htm
界面尺寸	1024*768		
界面功能	在博客首页中，单击发表文章超链接跳转到的页面。本页面主要功能是发表文章的功能界面		
备注	本页面是作为框架页面的一部分显示在博客首页右下方框架中的		
界面图颜	图 6-7 所示		

图 6-7　发表文章页面

图 6-8 为注册新博客的页面，该页面是在首页中，单击新博客注册超链接跳转过来的页面。在该页面中，用户可以填写个人注册信息成为网站中的一名博客。

表 6-8 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-8　　新博客注册页面说明

界 面 标 识	A6	模 板 文 件	regist.htm
界面尺寸	1024*768		
界面功能	注册新博客的页面，是在首页中，单击新博客注册超链接跳转过来的页面		
备注	在该页面中，用户可以填写个人注册信息成为网站的一名博客		
界面图题	图 6-8 所示		

图 6-8　新博客注册页面

图 6-9 为注册成功之后跳转的页面，通过该页面中提供的超链接，可以进入对应博客的个人首页，页面效果如图 6-5 所示。

表 6-9 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-9　　新博客注册成功页面说明

界 面 标 识	A6	模 板 文 件	registOK.htm
界面尺寸	1024*768		
界面功能	注册成功之后跳转的页面		

续表

界 面 标 识	A6	模 板 文 件	registOK.htm
备注	通过该页面中提供的超链接可以进入对应博客的个人首页		
界面图题	图 6-9 所示		

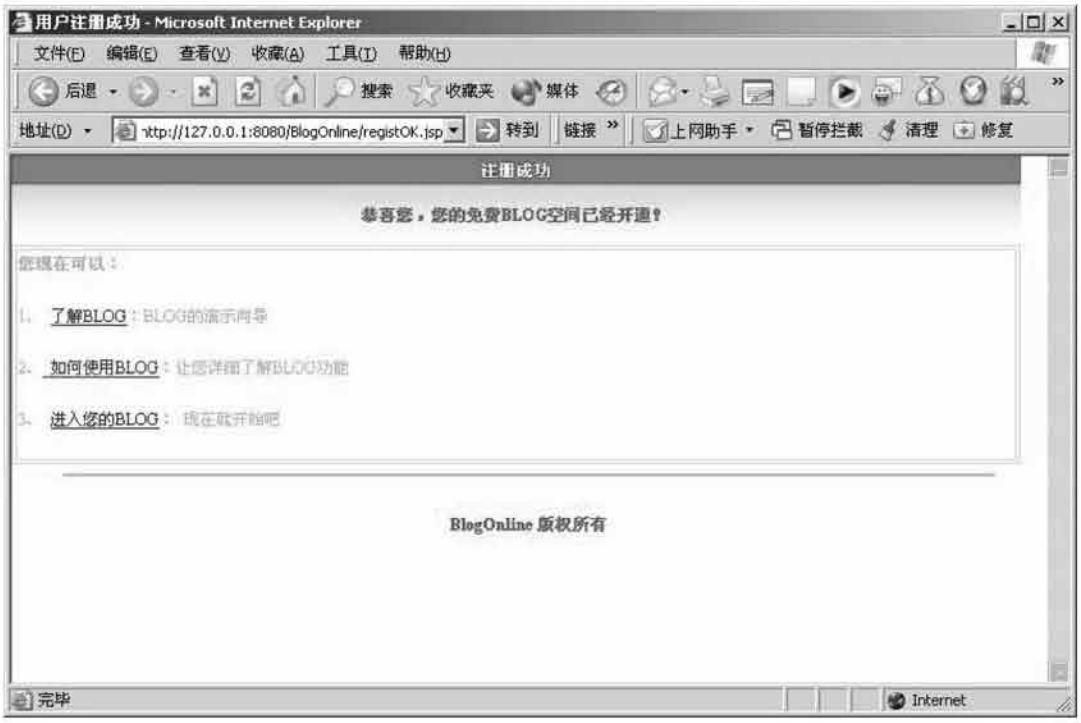


图 6-9 注册成功之后跳转的页面

图 6-10 为 Blog Online 博客网站博客个人维护管理的页面层次结构图。

图 6-10 页面层次结构图

该层次结构图表现出了博客个人维护主页和各一级页面之间的链接关系，一级页面与相关的各二级页面之间的链接关系将在本书后续章节中详细介绍。

Blog Online 博客网站后台主页以及主要链接页面的效果如图 6-11～6-15 所示，对应跳

转页面页面的详细效果及其功能将在后面的章节中进行介绍。

图 6-10 为该博客网站个人维护管理主页，本页面主要功能是，提供后续各页面的相关跳转。包括文章信息、文章类别信息、链接信息及博客个人信息维护管理的相关链接。该页面为框架页面。

表 6-10 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-10　　首页页面设计说明

界 面 标 识	A7	模 板 文 件	blogAdminMain.htm
界面尺寸	1024*768		
界面功能	博客网站个人维护管理主页，本页面主要功能是提供后续各页面的相关跳转		
备注	包括文章信息、文章类别信息、链接信息及博客个人信息维护管理的相关链接。该页面为框架页面		
界面图题	图 6-11 所示		

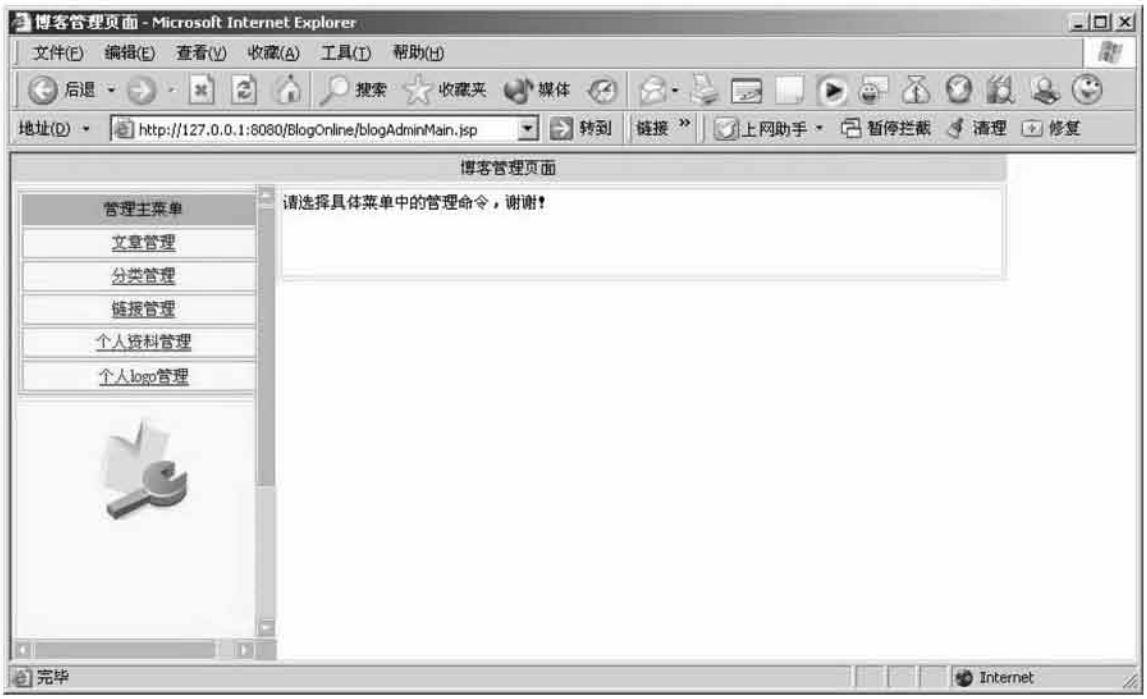


图 6-11　Blog Online 博客网站个人维护管理首页

图 6-12 为博客文章管理主页面，在该页面中博客可以对自己页面中发表的文章进行查找及删除等操作。

表 6-11 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-11 博客文章管理主页面说明

界 面 标 识	A8	模 板 文 件	adminArticle.htm
界面尺寸	1024*768		
界面功能	博客文章管理主页面		
备注	在该页面中，博客可以对自己页面中发表的文章进行编辑、查找及删除等操作		
界面图题	图 6-12 所示		

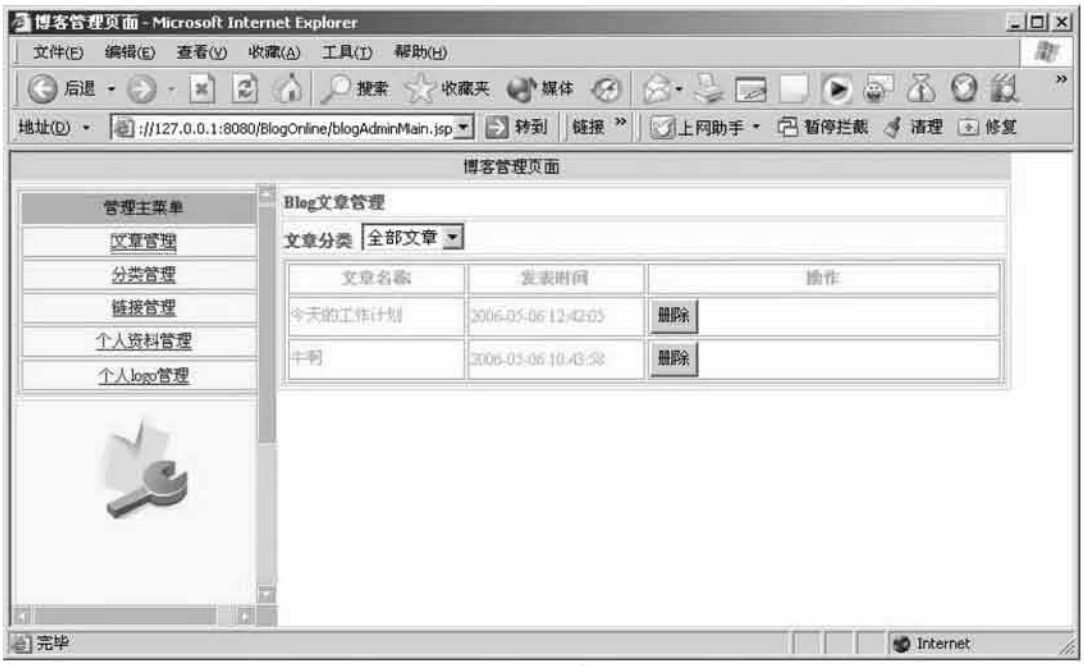


图 6-12 博客文章管理主页面

图 6-13 为博客文章分类管理主页面，在该页面中博客可以对自己页面中文章类别进行添加及删除等操作。

表 6-12 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-12 博客文章分类管理主页面说明

界 面 标 识	A9	模 板 文 件	adminSort..htm
界面尺寸	1024*768		
界面功能	博客文章分类管理主页面		
备注	在该页面中，博客可以对自己页面中文章类别进行添加及删除等操作		
界面图题	图 6-13 所示		

图 6-13　博客文章分类管理主页面

图 6-14 为博客友情链接管理主页面，在该页面中，博客可以对自己页面中的友情链接项进行添加及删除等操作。

表 6-13 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-13　　博客友情链接管理主页面说明

界面标识	A10	模板文件	adminLink..htm
界面尺寸	1024*768		
界面功能	博客友情链接管理主页面		
备注	在该页面中，博客可以对自己页面中的友情链接项进行添加及删除等操作		
界面图题	图 6-14 所示		

图 6-14　博客友情链接管理主页面

图 6-15 为博客个人信息修改页面，在该页面中，博客可以对自己的个人信息，包括博客标题、密码、电子邮件地址以及标志图片进行编辑修改等操作。

表 6-14 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-14　　用户信息维护管理页面说明

界 面 标 识	A11	模 板 文 件	adminSelf.htm
界面尺寸	1024*768		
界面功能	博客个人信息修改页面		
备注	在该页面中，博客可以对自己的个人信息，包括博客标题、密码、电子邮件地址以及标志图片进行编辑、修改等操作		
界面图题	图 6-15 所示		

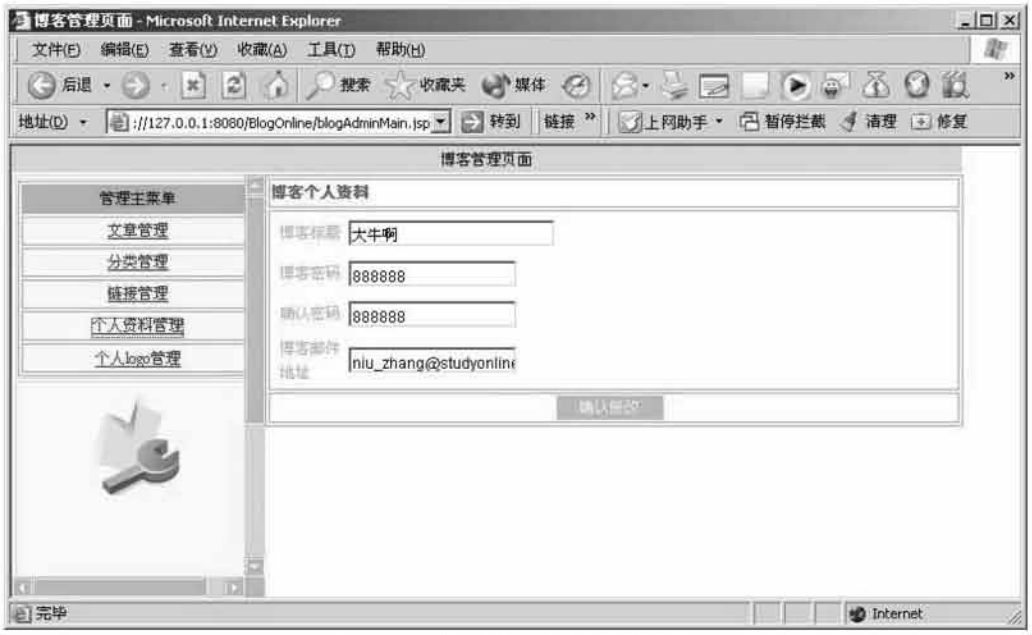


图 6-15　博客个人信息修改页面

图 6-16 为博客 logo 图片修改页面，在该页面中，博客可以对自己的个人 logo 图片进行修改，即实现图片的上传，以更换个人博客页面中的 logo 图片。

表 6-15 为网页设计师提交给后台程序设计人员的页面设计说明。

表 6-15　　博客 logo 图片维护管理页面说明

界 面 标 识	A11	模 板 文 件	adminLogo.htm
界面尺寸	1024*768		

界面功能	博客 logo 图片修改页面

续表

界 面 标 识	A11	模 板 文 件	adminLogo.htm
备注	在该页面中，博客可以对自己的个人 logo 图片进行修改，即实现图片的上传，以更换个人博客页面中的 logo 图片		
界面图题	图 6-16 所示		

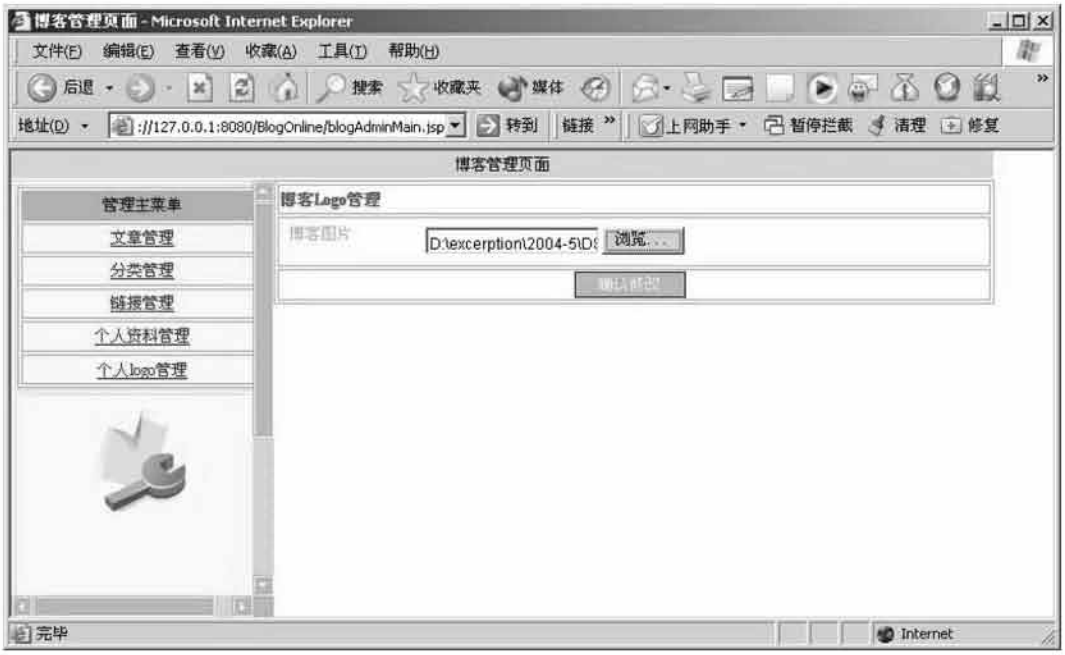


图 6-16　博客 logo 图片维护管理页面

通过本章的介绍，对于 Blog Online 网站的整体设计思路及功能模块都有了一定的了解，后续章节中会具体介绍实现的页面效果。

CHAPTER 7

第 7 章　Blog Online 博客页面实现

Blog Online 博客网站为网络用户以及博客都提供了一系列方便操作的功能，例如：为了能够使普通网络用户高效、方便地查看到自己所关注的博客文章，在 Blog Online 博客网站的首页模块中，提供了包括热门博客推荐、最新文章推荐、博客名称及文章标题关键字查询、博客页面中的文章分类检索等多种方式，使用户可以在最短的时间内定位到目标页面，同时在对应的文章页面中，还提供了显示对应文章评论、允许网络用户发表评论等功能。该系统中还提供了新博客注册及博客登录管理等功能，网络用户方便快捷地在博客网站中拥有自己个性化的博客页面。

本章以需求分析、总体设计、功能设计、数据库设计及详细设计的顺序，介绍 Blog Online 博客网站前台浏览系统的实现过程，在介绍详细设计时，按功能模块分别介绍了具体的实现过程。

7.1　系统设计

7.1.1　需求分析

博客页面系统主要是实现博客信息注册及登录管理、博客文章及相关评论、友情链接的查询、显示等功能的系统。一个典型的博客系统都需要有博客信息的动态显示、博客及文章的查询搜索、博客的注册登录管理等模块。

必知必会：用例图

用例图是指使用简单直观的方式，描述软件系统的功能和需求的图形表示。用例图中一般包括行为者和用例两个方面的描述。其中，用例用以说明用户的具体需求，而行为者则是用来描述要和用例之间进行交互的软件系统外部的人或者系统。

博客系统的用例图效果如图 7-1 所示。

本系统需要完成的具体任务如下。

- 博客注册登录：为了能够使普通网络用户获得博客身份，需要提供博客注册管理功能，为注册成功的博客提供个人博客页面，同时从使用权限的角度考虑，如果要在

自己的博客页面中发表文章或者进行个人博客页面的维护管理，必须先进行登录验证，只有验证通过才能够进行相关的操作。

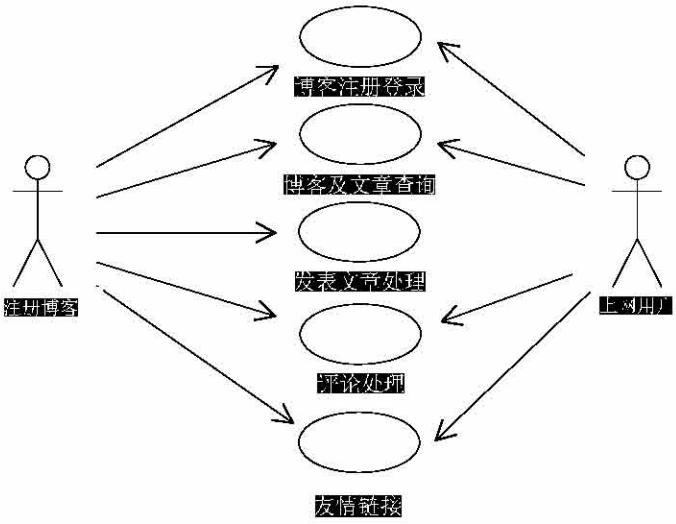


图 7-1　博客系统用例图

- 博客及文章查询：当普通网络用户进入博客网站的首页时，可以通过主页面提供的热门博客推荐以及最新文章推荐，尽快找到自己感兴趣的文章信息进行浏览。同时，在页面中，还应该提供按照文章标题等关键字快速查询所需文章信息的功能。
- 发表文章处理：身份验证通过的博客可以在自己的博客页面中发表文章。
- 发表及显示评论处理：当普通网络用户进入某一博客页面查看相关文章信息时，可以随时对自己所感兴趣的文章发表评论，同时也可以查看到其他人针对该文章的评论。
- 友情链接及网页访问量统计显示：在博客的个人页面中还提供了推荐给普通网络用户的相关友情链接，此外，对个人页面的访问量也在随时进行统计，并在个人页面中进行直观的显示。

7.1.2　总体设计

表 7-1 所示为面向网络用户的博客系统组成，通过本表，读者可以从宏观上了解各个组件的功能，由于在本系统中采用了 Ajax 技术，为了能够让读者清晰地看到应用的位置，在本表中进行了单独的标注。

注意：由于本例中采用了 Hibernate 作为数据持续层技术，因此，在建立项目后，需要通过构建路径的设置导入对应的 Hibernate 包。

表 7-1 系统组成表

用户表示层（视图）		控制处理层（控制器）	数据持续层	业务逻辑层（模型）
JSP 页面	Ajax 技术层			
index.jsp regist.jsp registOK.jsp searchResult.jsp blogMain.jsp top.jsp left.jsp main.jsp feedback.jsp newArticle.jsp otherBlogMain.jsp image.jsp	index.js newArticle.js regist.js	web.xml ArticleSaveAction.java ArticleSortAction.java LoginAction.java OpenArticleAction.java OpenBlogAction.java BlogOperateAction.java CheckUnameAction.java FirstAction.java RegistAction.java SearchAction.java ShowFeedbackAction.java WriteFeedbackAction.java	HibernateUtil.java DbOperate.java hibernate.cfg.xml model.hbm.xml	ImageTool.java Sort.java Blog.java Article.java FeedBack.java Links.java Constants.java

7.1.3 功能设计

1. 博客注册登录管理

该模块实现新博客的注册及登录验证功能。其中，注册新博客时会对用户输入的注册信息进行有效性验证，包括基本数据格式的有效性以及逻辑有效性，例如，用户名被占用时将及时给出提示。注册成功的博客登录时，会随时根据博客输入的注册信息进行提示，如用户名错误或者密码错误。同时在注册页面中还加入了验证码技术，以避免无谓的交互。

2. 博客及文章查询管理

该模块用于帮助网络用户，快速定位到自己关注的博客页面以及对应的文章信息中。在该模块中，分别提供了按照访问量统计结果推荐的十大热门博客排行榜，根据发表时间推荐的十大最新文章推荐列表。此外，还提供了按照文章标题关键字进行搜索的功能。当进入博客个人主页时，还实现了该博客文章的分类显示功能。

3. 发表文章管理

实现博客在个人主页中发表新文章的功能。在该模块中，博客可以在发表文章时，选择文章所属的类别，此外，只有登录验证通过的博客才可以在自己的博客页面中发表文章，没有验证通过的网络用户以及其他博客是不能够发表文章的。

4. 发表及显示评论管理

该模块实现了与文章相关的评论管理功能。网络用户可以方便地查看文章对应地评论信息，同时也可以对自己感兴趣的文章发表评论。

5. 友情链接及网页访问量统计

对博客的访问量的统计并直接显示在对应的博客页面中，此外，博客主页中还提供了推

荐的友情链接功能。

7.2 数据库设计与实现

7.2.1 数据库的需求分析

依据网上商城的处理需求，对应数据表的设计及功能如下。

- 博客基本信息表，简称博客表：存放与网站博客相关的基本信息；
- 文章分类基本信息表，简称类别表：存放每个博客对应的文章分类信息；
- 文章基本信息表简称文章表：存放每个博客的文章基本信息；
- 评论基本信息表简称评论表：存放与文章对应的评论基本信息；
- 友情链接基本信息表简称链接表：存放每个博客友情链接的基本信息。

7.2.2 数据库的逻辑设计

根据以上需求分析，在确定了各个表主键字段的基础上，依据表与表相关字段之间的联系建立了各表之间的关系，对应的关系图如图 7-2 所示。注，PK 意为主键。

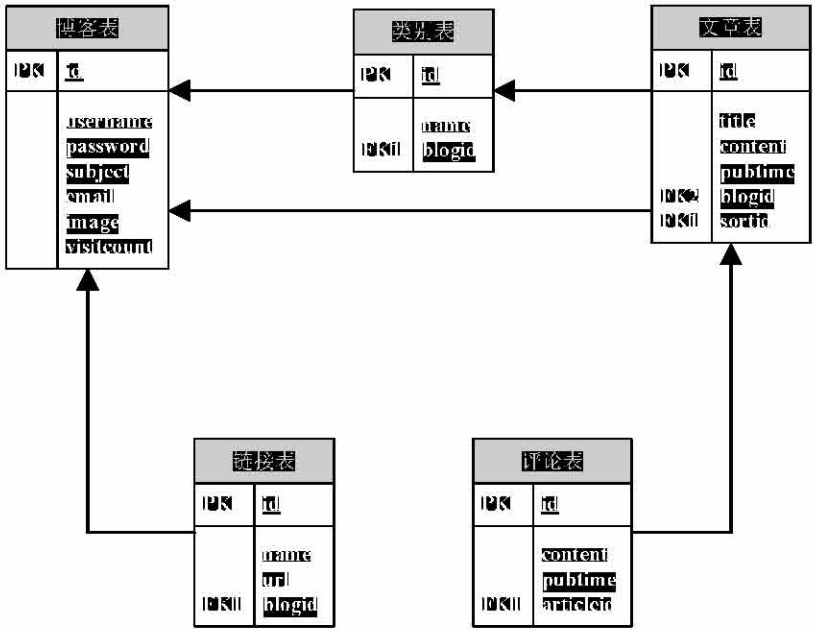


图 7-2 数据库关系图

其中文章表中的博客编号“blogid”与博客表中的博客编号“id”为相关字段，文章表中的分类编号“sortid”与类别表中的分类编号“id”是相关字段，而类别表中的博客编号

“blogid”与博客表中的博客编号“id”相关。评论表中的文章编号“articleid”与文章表中的文章编号“id”为相关字段。链接表中的博客编号“ blogid”与博客表中的博客编号“id”为相关字段。

博客在线系统中各数据表的基本结构如下。

1. 博客基本信息表

博客基本信息表（blog）是网站博客相关的基本信息的数据表，是显示、维护及管理博客数据的依据，表 7-2 所示为该表中所包含的字段及描述信息。

表 7-2　　博客基本信息表

字 段 名	描　述	类　型	长　度	是否允许为空	是 否 主 键
id	博客编号	INTEGER		否	是
username	博客登录用户名	VARCHAR	20	否	否
password	博客登录密码名	VARCHAR	20	否	否
subject	博客名称	VARCHAR	100	否	否
email	博客电子邮件地址	VARCHAR	50	否	否
image	博客 logo 图片	VARCHAR	100	是	否
visitcount	博客页面访问次数	INTEGER	2000	是	否

为了在页面中显示对应博客的 logo 图片，在该数据表中设计了“image”字段，用于保存对应博客 logo 图片的路径，以便按照该路径获取对应图片并显示。示例数据如图 7-3 所示。

id	username	password	subject	email	image	visitcount
1	jenny_jia	888888	累并快乐着	jenny_jia@blogonline.org	default_icon.jpg	3
2	robert_zhang	888888	老古董	robert_zhang@blogonline.org	default_icon.jpg	1
3	niu_zhang	888888	大牛啊	niu_zhang@studyonline.com	default_icon.jpg	41
4	mimi_jia	888888	妞妞	mimi_jia@sina.com.cn	default_icon.jpg	1

图 7-3　博客基本信息表示例数据

2. 文章分类基本信息表

文章分类基本信息表（sort）记录了博客文章各个分类的相关信息，以便对每个博客的文章信息进行分类显示，其字段设置如表 7-3 所示。

表 7-3　　文章分类基本信息表

字 段 名	描　述	类　型	长　度	是否允许为空	是 否 主 键
id	文章分类编号	INTEGER		否	是
name	文章分类名称	VARCHAR	40	否	否
blogid	分类所属博客编号	INTEGER		否	否

为了便于网络用户快速找到所需的文章信息，可以对文章信息按照不同博客进行分类管理。示例数据如图 7-4 所示。

id	name	blogid
1	工作笔记	3
3	旅行周刊	3
4	个人感悟	3
5	海边鱼村	7
6	工作周记	7

图 7-4　文章类别表示例数据

3. 文章基本信息表

文章基本信息表（article）用于存放与文章相关的信息，包括文章的标题、内容、发表时间、所属类别、所属博客等信息。其字段设置如表 7-4 所示。

表 7-4　　文章基本信息表

字 段 名	描　述	类　型	长　度	是否允许为空	是 否 主 键
id	文章编号	INTEGER		否	是
title	文章标题	VARCHAR	20	否	否
content	文章内容	TEXT	2000	是	否
pubtime	发表时间	DATETIME		否	否
blogid	所属博客编号	INTEGER		否	否
sortid	所属类别编号	INTEGER		是	否

图 7-5 所示文章基本信息表的示例数据。

id	title	content	pubtime	blogid	sortid
1	蓝蓝的天啊	我喜欢蓝蓝的天,碧碧的水,我要出去...	2006-05-0...	6	0
2	在发一篇	我在青海湖边休息,呵呵!	2006-05-0...	6	0
3	工作的时候不想别的	今天是工作日，认真努力的工作一定...	2006-05-0...	5	0
4	真是奇怪	原来的文件都丢失了,郁闷啊!	2006-05-0...	6	0
5	牛啊	今天攻破了一个技术难点,自己都说...	2006-05-0...	3	0
6	今天的工作计划	日程排得很满啊!	2006-05-0...	3	1
8	这里的天空	忘记了所有的一切,呵呵!	2006-05-0...	7	5

图 7-5　网上文章基本信息表示例数据

4. 评论基本信息表

评论基本信息表（feedback）用于存放与文章相关的评论信息，其字段设置如表 7-5 所示。

表 7-5 评论基本信息表

字 段 名	描 述	类 型	长 度	是否允许为空	是 否 主 键
id	评论编号	INTEGER		否	是
uname	发表评论用户昵称	VARCHAR	20	否	否
content	评论内容	TEXT	2000	是	否
pubtime	发表时间	DATETIME		否	否
articleid	所属文章编号	INTEGER		否	否

图 7-6 所示是评论信息表的示例数据。

id	uname	content	pubtime	articleid
1	小猪	什么时候带我一起去就好了,骑马放...	2006-05-06 10:23:10	2
2	大熊	我也想去,可是得加班,苦啊!	2006-05-06 10:23:44	2
3	老古董	还是在家里面睡觉舒服.	2006-05-06 10:24:16	2
4	小熊	青海,我梦中的湖!	2006-05-06 10:24:41	2
5	大猪	空气清新啊! 真好!	2006-05-06 10:38:42	1

图 7-6 评论信息表示例数据

5. 友情链接基本信息表

友情链接基本信息表（links）的结构如表 7-6 所示，该表用于存放与博客相关的友情链接的推荐信息。

表 7-6 友情链接基本信息表

字 段 名	描 述	类 型	长 度	是否允许为空	是 否 主 键
id	友情链接编号	INTEGER		否	是
name	链接名称	VARCHAR	40	否	否
url	链接地址	VARCHAR	100	否	否
blogid	所属博客编号	INTEGER		否	否

图 7-7 所示是友情链接信息表的示例数据。

id	name	url	blogid
1	我的学习网站	http://studyonline.com.cn	3
3	旅游天地	http://www.tourist.com	3
4	图书城	http://www.ptpress.com	3
5	蓝天碧海	http://www.sky.com	7

图 7-7 友情链接信息表示例数据

7.2.3 创建数据表的脚本

数据表的结构确定后，就可以在 MySQL 中完成数据表的创建工作，下面给出建立对应

数据表的 SQL 脚本。

1. 博客基本信息表

```
CREATE TABLE blog(
    id            INTEGER          PRIMARY KEY,
    username      VARCHAR (20)     NOT NULL,
    password      VARCHAR (20)     NOT NULL,
    subject       VARCHAR (100)    NOT NULL,
    email         VARCHAR (50)     NOT NULL,
    image         VARCHAR (100)    NULL,
    visitcount    INTEGER          NULL
)
```

2. 文章分类基本信息表

```
CREATE TABLE sort(
    id        INTEGER      PRIMARY KEY,
    name      VARCHAR (40) NOT NULL,
    blogid    INTEGER      NOT NULL
              REFERENCES blog(id) ON DELETE CASCADE
)
```

3. 文章基本信息表

```
CREATE TABLE article(
    id            INTEGER      PRIMARY KEY,
    title         VARCHAR (20)     NOT NULL ,
    content           TEXT(2000)   NULL,
    pubtime           DATETIME NOT NULL,
    blogid        INTEGER      NOT NULL
                  REFERENCES blog(id) ON DELETE CASCADE,
    sortid        INTEGER      NULL
                  REFERENCES sort(id) ON DELETE CASCADE
)
```

4. 评论基本信息表

```
CREATE TABLE feedback(
    id        INTEGER      PRIMARY KEY,
    uname     VARCHAR (20)     NOT NULL ,
    content       TEXT(2000)   NULL,
    pubtime       DATETIME     NOT NULL,
    articleid     INTEGER      NOT NULL
                  REFERENCES article(id) ON DELETE CASCADE
)
```

5. 友情链接基本信息表

```
CREATE TABLE links(
    id        INTEGER      PRIMARY KEY,
    name      VARCHAR (40)     NOT NULL,
    url       VARCHAR (100)    NOT NULL,
    blogid    INTEGER      NOT NULL
              REFERENCES blog(id) ON DELETE CASCADE
)
```

7.3 系统实现

7.3.1 实现效果

在地址栏中输入地址 http://127.0.0.1:8080/BlogOnline 之后，屏幕中将显示如图 7-8 所示的博客在线网站的首页。

图 7-8 博客在线网站的首页

页面上方显示的是博客的登录界面，为新博客提供了注册的超链接。在页面中部，分别提供了依据访问量统计的十大热门博客排行榜，依据发表时间显示的最新十大文章推荐列表，此外，还提供了可以依据文章标题关键字进行搜索的用户界面。

在博客网站首页博客人气排行榜中单击对应的博客名称超级链接，将跳转到图 7-9 所示的博客个人主页中。在博客页面中显示出了文章分类列表、对应的文章信息、友情链接推荐列表以及页面访问量统计结果。在文章分类中选择对应类别后，右侧框架中将分页显示该类别中的文章信息，如果博客在网站首页中正确地进行了登录，也将进入对应的博客页面，此时单击“发表文章”超级链接则允许进行文章发表操作，也允许进入个人管理系统。

在博客网站首页中最新文章推荐的排行榜中，单击对应的文章名称超级链接，将跳转到图 7-10 所示的博客个人主页中，在该页面中已经显示了对应文章内容及相关评论的信息。

图 7-9　博客个人主页

图 7-10　博客文章相关信息显示页面

在博客网站首页的搜索栏中，输入待搜索文章标题关键字，例如：“工作”，单击“搜索”按钮将显示如图 7-11 所示的搜索结果页面，在该页面中单击对应的文章标题，也将跳转到如图 7-10 所示的博客文章信息显示页面。

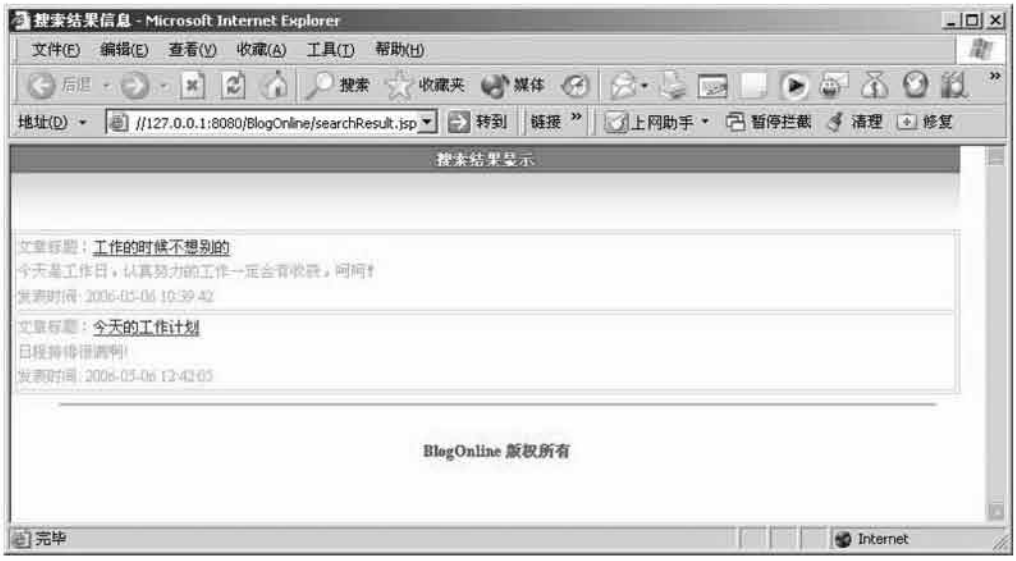


图 7-11 搜索结果页面

如果用户为新用户，则可以单击博客网站首页上方的“新博客注册”超级链接，进入如图 7-12 所示的注册页面进行注册。

图 7-12 新博客注册页面

在该页面中，新博客可以依据表单项目填写相关信息，单击“注册”按钮后，即完成注册工作，并显示注册成功提示信息页面，在该页面中，提供跳转到对应博客主页的超级链接。此外，在新博客填写注册信息的过程中，系统将随时给出相关的提示信息，图 7-13 所示为提示新博客填写的用户名称已经被占用。

图 7-13　用户名已经被占用时的提示信息

注册成功的博客，可以通过博客网站首页提供的登录界面进行身份验证，在进行身份验证时加入了验证码处理，同时在博客输入登录信息的过程中，将及时显示提示信息，如图 7-14 所示为博客输入验证码错误时的提示信息。

图 7-14　验证码错误时的提示信息

普通网络用户进入如图 7-9 所示的博客主页后，可以进行文章内容的查看，单击对应文章的“查看评论”超级链接，可以进入如图 7-15 所示的页面查看评论，并可发表评论。

图 7-15 查看及发表评论页面

已经通过身份验证的博客进入对应博客主页，则允许单击“发表文章”超级链接，进入如图 10-16 所示的页面，在该页面进行新文章的发表操作。

图 7-16 发表文章页面

有关友情链接显示等页面效果在这里就不再单独列出，读者可以参考随书附带光盘中程序源代码的运行效果。

整个博客在线系统在运行过程中，页面的基本流程如图 7-17 所示。

整个博客在线系统在 Eclipse 中的目录结构如图 7-18 所示。

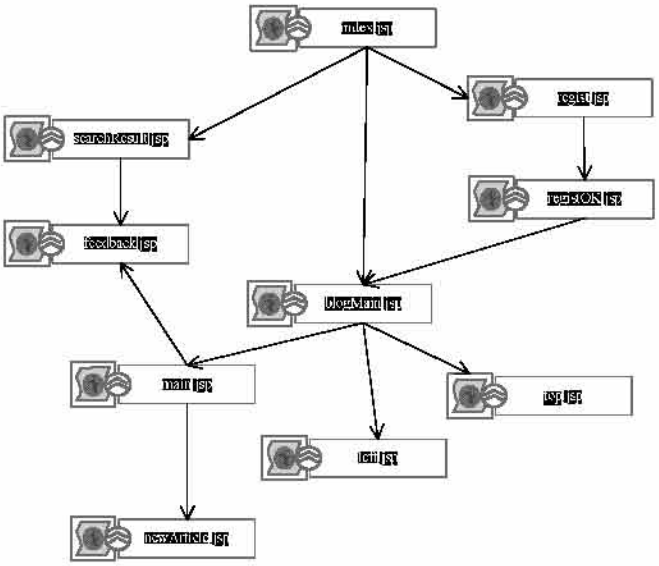


图 7-17　博客在线系统的页面基本流程图

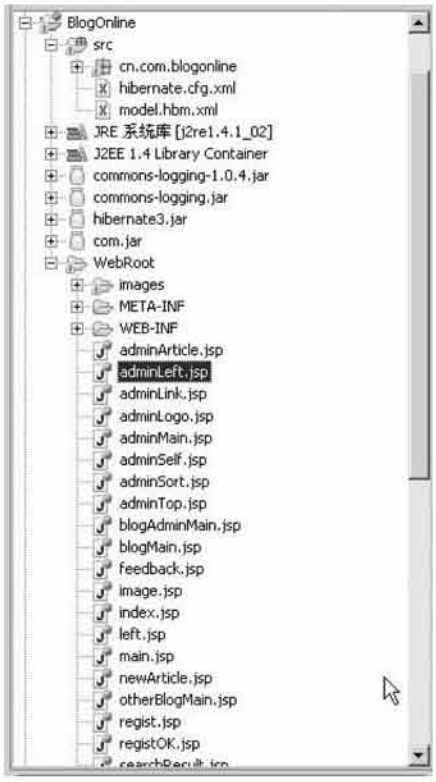


图 7-18　博客在线项目目录结构

7.3.2　JSP 页面

用户表示层主要是一些相关的 JSP 页面，用于实现图 10-8～图 10-17 所示的页面以及其他用户界面。该层中的 JSP 页面应该放在对应项目的 WebRoot 目录下。本书中只给出相应 JSP 页面中核心源代码，详细代码及效果可以参考本书附带源代码光盘。

1. 博客网站首页（index.jsp）

该页面中主要提供了用户注册、登录以及热门博客排行榜、最新文章推荐列表、按照文章标题关键字进行搜索的功能。核心程序代码及相关解释如下：

```
……
<link rel="StyleSheet" type="text/css" href="images/mystyle.css">
<script language="javascript" src="images/index.js" type="text/javascript"></script>
……</head>
```

```
<body onload="loadRequest()">
        ……
                <td align="middle">
                        登录名：
                        <input class="input01" id="uname" tabindex=
                        "1" size="15" name="uname">
                        密码：
                        <input class="input01" id="psw" tabindex="2"
                        type="password" size="15" name="psw">
                        验证码：
                        <input class="input01" id="checkwd" style="WIDTH:
                        60px" tabindex="3" size="15" name= "checkwd">
                        <img height="20" src="image.jsp" width="51"
                        align="absMiddle" alt="">
                        <img src="images/bk_button01.gif" align=
                        "absMiddle" value="提交" name="bt" onclick=
                        "userCheck()" alt="">
                        ……
                                <b><font color="#FF0000">博客
                                人气排行</font></b>
                        ……
                                        <tr>
                                            <td colspan="2"
                                            height="5"></td>
                                        </tr>
                                        <tr>
                                        </tr>
                                </tbody>
                                <tbody id="blogList"></tbody>
                                <tr></tr>

                        ……
</td>
<td width="50%" valign="top">
      <table>
            <td colspan="2">
                  <font color="#FF0000"><b>最新文章推荐
                  </b></font>
            </td>
            <tr>
            </tr>
            <tbody id="articleList"></tbody>
            <tr></tr>
      </table>
</td>
<td width="34%" valign="top">
      <table cellspacing="0" cellpadding="0" width="187"
      border="0">
            <tbody>
```

```
<tr>
    <td align="middle" height="21">
        <font color="#FF0000"><b>BLOG
        搜索</b></font>
    </td>
……
<tr>
    <form name="form1" action="search"
    method="post" target="_blank">

    <td align="middle">
        <input style="BORDER-RIGHT:
#e78d00 1px solid; BORDER-TOP: #e78d00 1px solid; BORDER-LEFT: #e78d00 1px solid; WIDTH:
125px; BORDER-BOTTOM: #e78d00 1px solid; HEIGHT: 22px" name="key">
        <input class="button-submit"
type="submit" value="搜索">
    </td>
    </form>
……
```

在上面的页面进行装载时，将调用“loadRequest()”函数，借助 Ajax 技术实现页面的自动刷新。读者可以在 images 文件夹下找到该页面中所使用的脚本文件，名称为“index.js”。

如果单击“登录”按钮将调用“userCheck()”函数，该函数是采用 Ajax 技术实现的，在该函数中将帮助进行数据基本格式的验证，如果验证通过，则提交“"login?uname=" + uname + "&psw=" + psw + "&checkwd=" + checkwd”请求与服务器端进行交互，从而进行博客登录验证的处理，需要注意的是：在登录信息的录入表单中，验证码是依赖动态生成的图片获得的。具体获得图片的相关文件包括“image.jsp”文件和服务器端封装了具体业务逻辑的 JavaBean，名称为“ImageTool.java”。如果单击“新博客注册”超级链接将跳转到“regist.jsp”页面进行新博客注册信息的录入。

需要注意的是：为了实现该页面中热门博客排行榜和最新文章推荐列表的实时动态更新，本页面中采用了 Ajax 技术，实现了页面的自动局部动态更新，具体实现代码可参考“index.js”脚本文件中定义的函数。

如果在热门博客排行榜中，单击对应博客名称的超级链接，借助 Ajax 提交“"openBlog?blogid=" + id”请求，将按照对应博客的编号获取博客信息，跳转到对应博客的主页中进行显示。

如果在最新文章推荐列表中，单击对应文章名称的超级链接，借助 Ajax 提交“"openArticle?articleid=" + id”请求，将按照对应文章的编号获取详细文章的信息，包括相关评论在内，然后跳转到对应博客的页面中进行显示。

当用户录入搜索文章标题关键字后，单击“搜索”按钮，提交“search”请求，实现按照关键字进行搜索查询。

需要注意的是：在该页面中配合 Ajax 技术的实现，对应 HTML 中的动态部分分别是热门博客排行榜，其名称为“blogList”，以及最新文章推荐列表，其名称为“articleList”。

2. 验证码动态显示页面（image.jsp）

在该页面中，主要提供验证码动态显示的功能。核心程序代码及相关解释如下：

```
……
<jsp:useBean id="image" scope="session" class="cn.com.blogonline.ImageTool" />
<%

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

     ImageIO.write(image.creatImage(), "JPEG", response.getOutputStream());
     session.setAttribute("rand",image.sRand);
%>
```

在上面的页面中，通过调用名称为“cn.com.blogonline.ImageTool”JavaBean 动态生成图片，同时为了便于后续验证码的核对处理，在本页面中还借助 session 的属性，将随机产生的验证码进行存储。

3. 博客个人主页面（blogMain.jsp）

本页面是在响应了网站首页提交的登录或者进入博客页面请求后，即“"login?uname="＋uname＋"&psw="＋psw＋"&checkwd="＋checkwd”或者“"openBlog?blogid="＋id”请求后，跳转到的页面。该页面为博客个人主页面，在该页面中，显示出了博客个人的相关信息，包括 logo 图片、文章列表、分类列表、友情链接列表以及页面访问量的统计结果。核心程序代码及相关解释如下。该页面为框架页面，核心的功能是在左侧框架中的“left.jsp”页面中提交请求，具体结果显示在右侧框架中对应的页面中。有关该页面代码，读者可以参阅本书光盘中对应的源程序。

4. blogMain.jsp 上方框架中的顶端页面（top.jsp）

该页面中一般显示网站的标志性信息，本例具体的效果及源代码，读者可以参阅本书光盘中对应的源程序。

5. blogMain.jsp 左侧框架中的主菜单页面（left.jsp）

在 blogMain.jsp 框架页面的左侧为 left.jsp 主菜单页面。在该页面中，显示出对应博客的个人 logo 图片、文章分类列表、友情链接列表及页面访问量统计等信息。对应的核心程序代码及相关解释如下：

```
……
<%
   Blog blog = (Blog)session.getAttribute(Constants.VISIT_BLOG_KEY);
   List sortList = (List)session.getAttribute(Constants.SORT_LIST_KEY);
   Sort sort = null;
```

```
        List linkList = (List)session.getAttribute(Constants.LINK_LIST_KEY);
        Links link = null;
    %>
    ……
        <td width="100%">
                    <img border="0" src="images/<%=blog.getImage()%>" width="180" height=
"121" alt="">
                                        ……
                    <b><font color="#FF0000"><%=blog.getSubject()%> 的 Blog</font></b>
                                        ……
                    <a href="blogOperate?sortid=0&amp;pageid=0" target= "main">个人首页</a>
                                            ……
                    <a href="blogAdminMain.jsp" target="_parent">个人管理</a>
                                            ……
                    <b><font color="#FF0000">文章分类</font></b>
                                        ……
                                            <p align="center">
                    <a href="blogOperate?sortid=0&amp;pageid=0" target= "main">所有文章</a>
                                            </p>
                                    </td>
                                </tr>

                    <%if(sortList != null ){

            for(int i = 0;i < sortList.size();i++)
            {
                        sort = (Sort)sortList.get(i);
      %>……
             <a   href="blogOperate?sortid=<%=sort.   getId()%>&amp;pageid=0"   target=
"main"><%=sort.getName()%></a>
                                            …….

                                    <%
            }
          }   %>

                                …….
             <b><font color="#FF0000">我的链接</font></b>
                                        ……
             <%if(linkList != null ){

           for(int i = 0;i < linkList.size();i++)
           {
                        link = (Links)linkList.get(i);
      %>
    ……
             <a href="<%=link.getUrl()%>" target="_blank"><%=link. getName()%></a>
```

```
                                                    ……
    <%
          }
       }  %>

……
          <tr>
              <td width="137" valign="top" align="center" height="17">
          <font color="#FF9900">网页计数器:<%=blog.getVisitcount ()%></font>
                    </td>
          </tr>
      ……
```

在本页面中，将首先获取对应博客记录信息、对应博客文章类别列表信息以及对应博客友情链接列表的信息。这些获取到的信息，将在页面对应的位置实现动态的显示，包括图片、访问量、文章分类，列表及友情链接列表等。

此外，在本页面中，当网络用户单击“个人首页”超级链接时，将提交“blogOperate?sortid=0&pageid=0”请求，在右侧框架中借助“blogMain.jsp”页面显示指定页数、指定类别的文章信息。

当单击“个人管理”超级链接时，将跳转到“blogAdminMain.jsp”页面，注意：由于后续页面中会进行身份检查，因此未经过身份验证的用户是不能够进入后续管理页面的。

当单击文章分类下面的对应文章类别的超级链接时，将提交“blogOperate?sortid= <%=sort.getId()%>&pageid=0”请求，在右侧框架中借助“blogMain.jsp”页面显示指定页数、指定类别的文章信息。

如果用户单击对应的链接条目，将依据获取到的链接地址，借助“<a href="<%=link.getUrl()%>"”进行页面的跳转。最终在页面最下方将获取到页面的访问量进行显示。

6. blogMain.jsp 右侧框架中的信息显示页面（blogMain.jsp）

该页面中，将显示对应博客的文章信息，为了便于查看，提供了分页显示的功能。此外，本页面中还提供了发表文章以及查看评论的超级链接。对应的核心程序代码及相关解释如下：

```
……
<%
    List articleList = (List)session.getAttribute(Constants.ARTICLE_LIST_KEY);
    Article article = null;
    Integer tpage=(Integer)session.getAttribute(Constants.CUR_PAGEID_KEY);
    int pageId=tpage.intValue();
    Integer tSortId=(Integer)session.getAttribute(Constants.CUR_SORTID_KEY);
    int sortId=tSortId.intValue();
%>
……
```

```
                                        <b><font color="#FF0000"><a href="newArticle.jsp">
                                        发表文章</a></font></b>
    ……
                    <%if(articleList != null ){
                    System.out.println(articleList.size());
          for(int i = 0;i < articleList.size();i++)
          {
                        article = (Article)articleList.get(i);
         %>
    ……
              <td width="100%">
         <b><font color="#FF0000"><%=article.getTitle()%></font></b>
                                    <p>
         <b><font color="#FF0000"><%=article.getContent()%></font></b>
                                    </p>
                                    <p align="right">
         <b><font color="#FF0000"><a href="showFeedback?articleid=<%=article.getId()%>">
查看评论</a></font></b>
                                    </p>
                               </td>
                    </tr>
         <%
            }
         }   %>
                    <tr>
                          <td width="100%">
                                <p align="center">
                                      | <font color="#999999"> <a href="blogOperate?
pageid=<%=pageId-1%>">上页</a></font> | <a href="blogOperate?pageid=<%=pageId+1%>">下页</a> |
                                </p>
                          </td>
                    </tr>
               </table>
          </body>
    </html>
```

在上面的页面中单击“发表文章”超级链接跳转到“newArticle.jsp”页面，注意：由于后续页面中会进行身份检查，因此没有经过身份验证的用户不能进入后续的页面发表文章。

单击“查看评论”超级链接后，提交“showFeedback?articleid=<%=article.getId()%>”请求，以显示跳转到评论显示页面，显示对应文章的相关评论信息。

单击“上页”或者“下页”超级链接将提交“blogOperate?pageid=<%=pageId-1%>”请求或者“blogOperate?pageid=<%=pageId+1%>”请求，以实现页面的跳转，显示指定页数的文章信息。

7. 发表文章页面（newArticle.jsp）

该页面中将提供对应的表单，以帮助博客完成新文章的发表功能。对应的核心程序代码

及相关解释如下：

```
    ……
    <%
        Blog loginBlog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
        Blog visitBlog = (Blog)session.getAttribute(Constants.VISIT_BLOG_KEY);

        if(loginBlog == null){
               response.sendRedirect("main.jsp");
        }
        else if((loginBlog.getId() != visitBlog.getId())){
               response.sendRedirect("main.jsp");
        }
        else{
    %>
    ……
                                <tr>
                                <td class="redfont" nowrap width="10%" height="30">
                                       日志标题
                                </td>
                                <td>
                                       <input class="text" id="entrytitle" title=
"entrytitle" size="75" name="entrytitle">
                                </td>
                                …….
                                <td class="redfont" height="30">
                                       是否选择文章分类
                                </td>
                                <td width="90%">
                                       <input type="checkbox" name="sortOk" onclick=
"refreshList()">
    ……
                                <td class="redfont" height="30">
                                       日志分类
                                </td>
                                <td width="90%">
                                              <select class="text" id="category" name=
"category"></select>
                                                                  ……
                                                    <td class="redfont" valign="top"
height="30">
                                       日志内容
                                                    </td>
                                                    <td>
                                              <textarea name="entrycontent" rows="10"
cols="50"></textarea>
    ……
                          <td width="100%" align="center">
                                <input class="button-submit" id="save" type="submit"
value="发布日志" name="save" onclick="saveRequest()">
```

```
    ……
    <%}%>
```

在该页面中，借助 Ajax 技术实现了请求的处理，对应的 JavaScript 脚本文件位于应用目录下面的 images 文件夹中，名称为“newArticle.js”。博客填写完新文章的信息，单击“发表日志”按钮后，将调用“saveRequest()”函数，在该函数中将以 POST 方式提交对应的请求，并提交新文章相关的信息，请求名称为“saveArticle”，所提交的参数形式为“"title=" + title + "&sortid=" + sort + "&content=" + content”，注意：采用 POST 方式提交是为了解决提交信息的中文乱码问题。

此外，在该页面中当博客选择了“是否选择文章分类”的复选框后，通过“onclick”事件调用“refreshList()”函数，然后借助 Ajax 技术动态获取对应博客的文章分类信息，装入名称为“category”的动态下拉列表框中，帮助完成文章分类列表的载入。

8. 博客个人主页面（otherBlogMain.jsp）

本页面是在响应了网站首页提交的“"openArticle?articleid=" + id”请求后，跳转到的页面。此外，在搜索结果页面中，单击对应文章的标题也将跳转到该页面。该页面也是博客个人主页面，与 blogMain.jsp 页面不同的是：在该页面的右侧框架中，将显示指定文章及相关评论的信息。有关该页面代码，读者可以参阅本书光盘中的源程序。

9. 显示文章及评论信息页面（feedback.jsp）

在本页面中显示指定文章的信息以及相关的评论信息，网络用户也可以在该页面中发表新的评论。核心程序代码及相关解释如下：

```
    ……
    <%
        Article article = (Article)session.getAttribute(Constants.CUR_ARTICLE_KEY);
        List feedBackList = (List)session.getAttribute(Constants.FEEDBACK_LIST_KEY);
        FeedBack feedBack = null;
        Integer tpage=(Integer)session.getAttribute(Constants.CUR_PAGEID_KEY);
        int pageId=tpage.intValue();
        System.out.println(tpage);
    %>
    ……              <b><font color="#FF0000">文章信息</font></b>
    ……
                            <b><font color="#FF0000"><%=article.getTitle()%></
font></b>
                            <p>
                                <b><font color="#FF0000"><%=article.getContent()
%></font></b>
                            </p>
                            <p>
                            </p>
                        </td>
                    </tr>
                    <tr>
                        <td width="100%">
```

```
                                    <p align="center">
                                        <b><font color="#FF0000">相关评论</font></b>
                                    </p>
                                </td>
                            </tr>

                            <%if(feedBackList != null ){

          for(int i=0;i < feedBackList.size();i++)
          {
                            feedBack = (FeedBack)feedBackList.get(i);
          %>
    ……
              <b><%=feedBack.getUname()%></b>
    ……
              <b><%=feedBack.getPubtime()%></b>
    ……
              <b><%=feedBack.getContent()%></b>
    ……
         <%
            }
          }  %>

    ……
                | <font color="#999999"> <a href="showFeedback?pageid=<%=pageId-1%>">
上页</a></font> | <a href="showFeedback?pageid=<%=pageId+1%>">下页</a> |
                                </p>
          ……
                                <b><font color="#FF0000">发表评论：</font></b>
                        <form name="regForm" onsubmit="" action="writeFeedback" method="post">
                                        <label class="redfont" for="user">
                                                                您的昵称
    ……
                                    <input class="text"id="uname"name="uname"datatype="Username">
                                            </td>
    ……
                                                    <td>
                                    <textarea rows="6" name="content" cols="50"></textarea>
                                                    </td>
    ……
                                    <input class="button-submit" type="submit" value="提交"
name="Submit">
                                                                ……
```

本页面获取要显示的文章记录信息、对应文章评论列表信息以及对应分页显示的当前页数的信息。这些获取到的信息将在页面对应的位置实现动态的显示，包括文章的内容，相关评论信息的分页显示等。单击对应分页显示的超级链接后，提交对应的请求进行页面的跳转。

在本页面中，当网络用户填写了相关评论信息后，单击“提交”按钮将提交“writeFeedback”

请求，Servlet 程序将处理评论信息。

10. 文章搜索结果显示页面（searchResult.jsp）

本页面是在搜索栏中输入对应的关键字信息后，单击“搜索”按钮进行处理后跳转的页面，在该页面中显示出了查询的结果，如果用户没有输入关键字，则将显示所有文章的信息。核心程序代码及相关解释如下：

```
……
<%
     List searchList = (List)session.getAttribute(Constants.ARTICLE_LIST_KEY);
     Article article = null;
%>
……                <%if(searchList != null ){
                  System.out.println(searchList.size());
     for(int i = 0;i < searchList.size();i++)
     {
                  article = (Article)searchList.get(i);
   %>
……
                文章标题：<a href="openArticle?articleid=<%=article.getId()%>" target=
"_blank"><%=article.getTitle()%></a>
                                  </div>
                                  <div class="systemLHeight180">
                                        <%=article.getContent()%>
                                        <div class="author" style="width: 740; height:
19">
                    发表时间：
                                        <%=article.getPubtime()%>
        ……
                  <%
          }
     }  %>
……
```

页面获取查询到的文章列表信息，然后再控制动态显示输出。当用户在该页面中单击对应文章标题的超级链接后，将提交“openArticle?articleid=<%=article.getId()%>”请求，以显示指定的文章的详细信息。

11. 注册页面（regist.jsp）

本页面是单击了“新博客注册”超级链接后跳转到的页面。核心程序代码及相关解释如下：

```
……
        <link rel="StyleSheet" type="text/css" href="images/mystyle.css">
        <script language="javascript" src="images/regist.js" type="text/javascript">
</script>
……
                                  <font color="#FF0000"><b>请填写您的个人信息！</b></font>
      ……
```

```
                    <form name="regForm" onsubmit="return regist()" action=
"regist" method="post">
                        <table cellspacing="2" cellpadding="0" width="100%"
border="0">
                            <tbody>
                                <td width="12%">
                                    <label class="redfont" for="user">
                                        您的用户名
                                    </label>
                                </td>
                                <td>
                <input onblur="checkUser()" class="text" id="uname" name="uname"
msg="用户名不能为空,应以字母开头，只含字母、数字、下划线" >
            ……
                <input class="text" id="psw" type="password" name="psw" msg="密码应
为字母、数字、特殊字符" max="18" min="6">
            ……
                <input class="text" id="psw2" type="password" name="psw2" msg="两次
输入的密码不一致" >
                        <span></span>
            ……
                <input class="text" id="subject" name="subject" msg="博客标题不能为空
" datatype="Require">
                                ……
                <input class="text" id="email" type="text" name="email" msg="需符合
电子邮件地址的格式" max="18" min="6">
                        <span>请按照电子邮件地址的格式填写</span>
            ……
                <input class="button-submit" type="submit" value="注册">
        ……
```

在用户填写注册信息的过程中，结合 Ajax 技术对输入的注册信息进行验证，当用户输入了用户名，准备离开对应文本框时，将借助“onblue”事件调用“checkUser()”函数。对应的 JavaScript 脚本文件位于应用目录下面的 images 文件夹中，名称为“regist.js”。该函数将借助 Ajax 技术提交“"checkUname?uname=" + uname”请求，依据数据库中的数据信息确认对应的用户名是否被占用，然后再对后续的信息进行数据有效性验证。

如果验证通过，当用户单击“注册”按钮后，将提交“regist”请求，以借助 Servlet 完成新博客的注册。

12. 注册成功页面（registOK.jsp）

本页面是新博客注册成功之后跳转到的页面，该页面程序的源代码可以参考本书配套光盘。

7.3.3 Ajax 技术相关脚本文件

本系统中适当选择使用了 Ajax 技术，帮助实现浏览器客户端和服务器之间的交互，以达

到异步通信的目的，从而给用户更好的感受，为配合具体实现，涉及到了相关的 JavaScript 脚本文件，这些文件均统一放置在项目文件夹下的“images”文件夹中，然后在对应的 JSP 页面中使用对应的链接语句将其引入。

下面对相关的脚本文件进行逐一介绍。

1. 网站首页脚本文件（index.js）

该文件是在网站首页 index.jsp 页面中使用的脚本文件，主要用于处理用户登录请求、热门博客排行榜及最新文章推荐列表的自动局部刷新的功能。具体代码及相关解释如下：

```
var XMLHttpReq;
     //创建 XMLHttpRequest 对象
function createXMLHttpRequest() {
    if (window.XMLHttpRequest) { //Mozilla 浏览器
          XMLHttpReq = new XMLHttpRequest();
    } else {
          if (window.ActiveXObject) { // IE 浏览器
               try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
               }
               catch (e){
                    try{
                        XMLHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
                    }
                    catch (e) {
                    }
               }
          }
    }
}
    //发送请求函数
function loadRequest() {
    createXMLHttpRequest();
    var url = "first";
    XMLHttpReq.open("GET", url, true);
    XMLHttpReq.onreadystatechange = processLoadResponse;//指定响应函数
    XMLHttpReq.send(null);  // 发送请求
}
      // 处理返回信息函数
function processLoadResponse() {
       if (XMLHttpReq.readyState == 4) { // 判断对象状态
       if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
              AddBlogList();
              AddArticleList();
                  setTimeout("loadRequest()", 10000);
         } else { //页面不正常
              window.alert("您所请求的页面有异常。");
         }
       }
```

```
    }
        // 增加 Blog 函数
        function AddBlogList() {
            deleteBlogList();
            var xmlDoc =XMLHttpReq.responseXML;
        var blogs = xmlDoc.getElementsByTagName("blog");
        var currentBlog = null;
        for(var i = 0; i < blogs.length; i++) {
            currentBlog = blogs[i];
            var id =currentBlog.getElementsByTagName("id")[0].firstChild.nodeValue;
            var name =currentBlog.getElementsByTagName("name")[0].firstChild.nodeValue;

                var row = document.createElement("tr");
                row.setAttribute("id",id);
                var cell = document.createElement("td");
                cell.appendChild(document.createTextNode("NO " + (i+1) + "."));
                row.appendChild(cell);

                cell = document.createElement("td");
                var href = document.createElement("a");
                 href.setAttribute("href","openBlog?blogid=" + id);

                 href.appendChild(document.createTextNode(name));
                cell.appendChild(href);
                row.appendChild(cell);

                document.getElementById("blogList").appendChild(row);
            }
        }

        // 增加 Article 函数
        function AddArticleList() {
            deleteArticleList();
            var xmlDoc =XMLHttpReq.responseXML;
        var articles = xmlDoc.getElementsByTagName("article");
        var currentArticle = null;
        for(var i = 0; i < articles.length; i++) {
            currentArticle = articles[i];
            var id =currentArticle.getElementsByTagName("id")[0].firstChild.
nodeValue;
            var  name  =currentArticle.getElementsByTagName("title")[0].firstChild.
nodeValue;
            var time =currentArticle.getElementsByTagName("time")[0].firstChild.
nodeValue;

                var row = document.createElement("tr");
                row.setAttribute("id",id);
                var cell = document.createElement("td");
                var href = document.createElement("a");
```

```
                href.setAttribute("href","openArticle?articleid=" + id);
                href.appendChild(document.createTextNode(name));
              cell.appendChild(href);
              row.appendChild(cell);

              cell = document.createElement("td");
              cell.appendChild(document.createTextNode(time));
              cell.setAttribute("width",110);
              row.appendChild(cell);

              document.getElementById("articleList").appendChild(row);
        }
  }
  // 删除 Blog 函数
  function deleteBlogList() {
        var blogList = document.getElementById("blogList");
        while (blogList.hasChildNodes()) {
              blogList.removeChild(blogList.firstChild);
        }
  }

  // 删除 Article 函数
  function deleteArticleList() {
        var articleList = document.getElementById("articleList");
        while (articleList.hasChildNodes()) {
              articleList.removeChild(articleList.firstChild);
        }
  }

  //发送请求函数
  function loginRequest(url) {
      createXMLHttpRequest();
      XMLHttpReq.open("GET",url, true);
  XMLHttpReq.onreadystatechange = processLoginResponse;//指定响应函数
  XMLHttpReq.send(null);  // 发送请求
  }
  // 处理身份验证返回信息函数
 function processLoginResponse() {
  if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
              var
res=XMLHttpReq.responseXML.getElementsByTagName("res")[0].firstChild.nodeValue;
              if (res==1){
              window.alert("用户名错误！");
              }
              else if (res==2){
              window.alert("密码错误！");
              }
              else if (res==3){
```

```
                    window.alert("验证码错误！");
                    }
                    else if (res==0){
                              var
id=XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.nodeValue;
                    window.location = "openBlog?blogid=" + id;
                    }
               } else { //页面不正常
                    window.alert("您所请求的页面有异常。");
               }
          }
     }
      // 博客身份验证函数
     function userCheck() {
               uname = document.loginForm.uname.value;
               psw = document.loginForm.psw.value;
               checkwd = document.loginForm.checkwd.value;
               if(uname=="") {
                     window.alert("用户名不能为空。");
                     document.loginForm.uname.focus();
                     return false;
               }
               else {
                     loginRequest("login?uname=" + uname + "&psw=" + psw + "&checkwd="
+ checkwd);
               }
      }
```

2. 发表新文章页面的脚本文件（newArticle.js）

该文件是在发表文章的页面 newArticle.jsp 使用的脚本，主要用于处理保存新文章信息到数据库中的请求，以及当博客选择指定文章分类后，协助动态获取数据库中的对应博客的文章分类信息以生成动态下拉列表框。具体代码及相关解释如下：

```
……
//发送请求函数
function refreshList() {
     var sort = document.getElementById("sortOk").checked;
     if(sort == false ) {
          clearList();
          return;
     }
     var url = "dyList";
     createXMLHttpRequest();
     XMLHttpReq.open("GET", url, true);
     XMLHttpReq.onreadystatechange = processListResponse;//指定响应函数
     XMLHttpReq.send(null);  // 发送请求
}

// 处理动态下拉列表框返回信息函数
function processListResponse() {
```

```
    if (XMLHttpReq.readyState == 4) { // 判断对象状态
          if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                updateList();
          } else { //页面不正常
                window.alert("您所请求的页面有异常。");
          }
      }
}
    // 生成动态下拉列表框函数
    function updateList() {
         clearList();
         var category = document.getElementById("category");

          var xmlDoc =XMLHttpReq.responseXML;
    var sorts = xmlDoc.getElementsByTagName("sort");
    var currentSort = null;
    for(var i = 0; i < sorts.length; i++) {
          currentSort = sorts[i];
          var id =currentSort.getElementsByTagName("id")[0].firstChild.nodeValue;
          var name =currentSort.getElementsByTagName("name")[0].firstChild.nodeValue;

          var option = null;
             option = document.createElement("option");
             option.appendChild(document.createTextNode(name));
             option.setAttribute("value",id);
             category.appendChild(option);

          }
     }
     // 清除动态下拉列表框函数
    function clearList() {
         var sort = document.getElementById("category");
         while(sort.childNodes.length > 0) {
              sort.removeChild(sort.childNodes[0]);
         }
     }

     //发送保存新文章请求函数
     function saveRequest(url) {
           var title = document.articleForm.entrytitle.value;
           var sort =0;
           if (document.articleForm.sortOk.checked) {
                 sort = document.articleForm.category.value;
           }
           var content = document.articleForm.entrycontent.value;
          var url = "title=" + title + "&sortid=" + sort + "&content=" + content;
           createXMLHttpRequest();
           XMLHttpReq.open("POST","saveArticle", true);
    XMLHttpReq.onreadystatechange = processSaveResponse;//指定响应函数
```

```
    XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded")
    XMLHttpReq.send(url);  // 发送请求
    }
    // 处理保存新文章返回信息函数
function processSaveResponse() {
    if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
            var res=XMLHttpReq.responseText;
            window.alert(res);
        } else { //页面不正常
            window.alert("您所请求的页面有异常。");
        }
    }
}
```

3. 新博客注册页面的脚本文件（regist.js）

该文件是新博客注册页面 regist.jsp 中使用的脚本，主要用于处理验证注册用户名是否已经被占用的功能。具体代码及相关解释如下：

```
……
    //发送请求函数
function sendRequest(url) {
    createXMLHttpRequest();
    XMLHttpReq.open("GET", url, true);
    XMLHttpReq.onreadystatechange = processResponse;//指定响应函数
    XMLHttpReq.send(null);  // 发送请求
}
    // 处理返回信息函数
function processResponse() {
    if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
            var res = XMLHttpReq.responseText;
            if (res == 1) {
                window.alert("用户名已经被占用,请更换!");
                document.regForm.uname.focus();
                userAviable=false;
            }else{
                userAviable=true;
            }
        } else { //页面不正常
            window.alert("\u60a8\u6240\u8bf7\u6c42\u7684\u9875\u9762\u6709\
u5f02\u5e38\u3002");
        }
    }
}
//提交验证用户名是否被占用的函数
function checkUser(){
    var uname= document.regForm.uname.value;
    if (uname== "") {
        window.alert(document.regForm.uname.msg);
```

```
                document.regForm.uname.focus();
                return false;
            } else {
              sendRequest("checkUname?uname=" + uname);
            }
    }
        // 对注册信息进行验证的函数
    function regist() {
        if (!userAviable) {
            return false;
        }
        var uname = document.regForm.uname.value;
        var psw = document.regForm.psw.value;
        var psw2 = document.regForm.psw2.value;
        var subject = document.regForm.subject.value;
        var email = document.regForm.email.value;
        if (uname == "") {
            window.alert(document.regForm.uname.msg);
            document.regForm.uname.focus();
            return false;
        } else {
            if (psw == "") {
                window.alert(document.regForm.psw.msg);
                document.regForm.psw.focus();
                return false;
            } else {
                if (psw != psw2) {
                    window.alert(document.regForm.psw2.msg);
                    document.regForm.psw2.focus();
                    return false;
                } else {
                    if (subject == "") {
                        window.alert(document.regForm.subject.msg);
                        document.regForm.subject.focus();
                        return false;
                    } else {
                        if (email == "") {
                            window.alert(document.regForm.email.msg);
                            document.regForm.email.focus();
                            return false;
                        } else {
                            return true;
                        }
                    }
                }
            }
        }
    }
```

7.3.4 数据持续层

由于本系统采用了 Hibernate 开发架构，实现了数据持续层处理的功能，因此本节中给出与 Hibernate 相关的配置文件及映射文件的具体源代码及其功能解释。对应 Hibernate 的配置文件和映射文件一般放置在项目的 src 目录中。而用于管理 session 以及封装了与数据库操作相关方法的类则与其他类同样，放在了对应的包中。

1. Hibernate 配置文件（hibernate.cfg.xm）

该文件主要是进行 SessionFactory 配置（可以理解成建立与数据库之间连接的配置），在 Hibernate 中，既可以使用 XML 文件，也可以使用属性文件来进行配置，这里介绍使用 XML 配置文件方式，这也是 Hibernate 所建议的配置方式，对应的文件名是“hibernate.cfg.xml”，该文件核心程序代码及相关解释如下：

```
……
   <session-factory>
        <property
name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost/test</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">zgy01</property>
        <property name="hibernate.connection.pool_size">100</property>
           <property name="show_sql">false</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- Mapping files -->
        <mapping resource="model.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

注意在该文件中，读者应该根据自己所使用的数据库系统以及对应的 JDBC 驱动程序的实际情况进行配置。在此之前，应该已经将 JDBC 驱动程序部署在应用服务器 Tomcat 的 common/lib 下面。

2. Hibernate 映射文件（model.hbm.xml）

该映射文件是 Hibernate 架构中最基本也是最重要的一个配置文件，用于声明 Hibernate 中，实现实体类的属性和数据库对应数据表字段之间的映射关系。对应的文件名是“model.hbm.xml”，该文件核心程序代码及相关解释如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="cn.com.blogonline">
```

- 建立与博客信息表之间的映射关系

```
<class name="cn.com.blogonline.Blog"
```

```
            table="blog"
            discriminator-value="C">

           <id name="id">
            <generator class="increment"/>
    </id>

    <property name="username"/>
    <property name="password"/>
    <property name="subject"/>
    <property name="email"/>
    <property name="image"/>
    <property name="visitcount"/>
</class>
```

- 建立与文章分类信息表之间的映射关系

```
<class name="cn.com.blogonline.Sort"
          table="sort"
          discriminator-value="C">

      <id name="id">
            <generator class="increment"/>
     </id>

    <property name="name"/>
    <property name="blogid"/>
</class>
```

- 建立与文章信息表之间的映射关系

```
<class name="cn.com.blogonline.Article"
          table="article"
          discriminator-value="C">

         <id name="id">
           <generator class="increment"/>
    </id>

    <property name="title"/>
    <property name="content"/>
    <property name="pubtime"/>
    <property name="blogid"/>
    <property name="sortid"/>
</class>
```

- 建立与文章评论信息表之间的映射关系

```
<class name="cn.com.blogonline.FeedBack"
          table="feedback"
          discriminator-value="C">

        <id name="id">
          <generator class="increment"/>
```

```
        </id>

        <property name="uname"/>
        <property name="content"/>
        <property name="pubtime"/>
        <property name="articleid"/>
    </class>
```

- 建立与友情链接信息表之间的映射关系

```
    <class name="cn.com.blogonline.Links"
             table="links"
             discriminator-value="C">

         <id name="id">
                <generator class="increment"/>
         </id>

        <property name="name"/>
        <property name="url"/>
        <property name="blogid"/>
    </class>
</hibernate-mapping>
```

注意：本文件中声明的属性名称一定要和数据表中的字段名称对应，也要和业务逻辑层中对应实体类中声明的封装实体信息的属性名称相同。

3. 声明管理 session 的类（HibernateUtil.java）

该类用于管理 session 的生成和关闭。其中声明管理 session 的类，基本上符合在传统 Java Web 开发中的 JDBC 中的 Connection 对象的概念。在 Hibernate 中，该 session 帮助实现和数据库之间的交互。在 Hibernate 的开发过程中，session 类声明的格式基本上是固定的，在项目中新建一个类，该类应该放置在项目文件夹下的 src 对应的包中，名称为 HibernateUtil，程序源代码如下：

```
package cn.com.blogonline;

import org.hibernate.HibernateException;
……
public class HibernateUtil {

     private static Log log = LogFactory.getLog(HibernateUtil.class);
     private static final SessionFactory sessionFactory;

     static{
          try {
               // Create the SessionFactory
               sessionFactory = new Configuration().configure().buildSession
Factory();
          } catch (Throwable ex) {
```

```
                ex.printStackTrace();
                //log.error("Initial SessionFactory creation failed.", ex);
                throw new ExceptionInInitializerError(ex);
            }
        }

        public static final ThreadLocal session = new ThreadLocal();

        public static Session currentSession() throws HibernateException {
            Session s = (Session) session.get();
            // Open a new Session, if this Thread has none yet
            if (s == null||!s.isOpen()) {
                s = sessionFactory.openSession();
                session.set(s);
            }
            return s;
        }
        public static void closeSession() throws HibernateException {
            Session s = (Session) session.get();
            session.set(null);
            if (s != null)
                s.close();
        }
}
```

4. 封装了数据库操作的类（DbOperate.java）

该类中封装了借助 Hibernate 进行数据库相关操作的方法。

```
package cn.com.blogonline;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class DbOperate {

    /**
     * 根据用户名得到 Blog 对象
     */
    public Blog getBlog(String userId) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        Blog blog = null;
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            //创建查询对象
            Query query = session.createQuery("from Blog where username= :userId");
```

```
            query.setParameter("userId", userId);
            List list = query.list();
            if (!list.isEmpty())
                    blog = (Blog) list.get(0);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null)
                    tx.rollback();
            throw e;
        }
        session.close();
        return blog;
}

/**
 * 根据 ID 得到 Blog 对象
 */
public Blog getBlog(int Id) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        Blog blog = null;
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            //创建查询对象
            Query query = session.createQuery("from Blog where id=" + Id);
            List list = query.list();
            if (!list.isEmpty())
                    blog = (Blog) list.get(0);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null)
                    tx.rollback();
            throw e;
        }
        session.close();
        return blog;
}


/**
 * 得到热点 Blog 对象集
 */
public List getBlogs(int showCount) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        List list=null;
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            //创建查询对象
```

```
                        Query query = session.createQuery("from Blog order by visitcount
desc");
                        if (showCount>0){
                                query.setMaxResults(showCount);//记录集最大个数
                        }
                        list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                        tx.commit();
                } catch (HibernateException e) {
                        if (tx!= null)
                                tx.rollback();
                        throw e;
                }
                HibernateUtil.closeSession();
                return list;
        }

        /**
         * 得到匹配 Blog 对象集
         */
        public List getMatchBlogs(String key) throws HibernateException {
                Session session = HibernateUtil.currentSession();
                List list=null;
                Transaction tx = null;
                try {
                        tx = session.beginTransaction();
                        //创建查询对象
                        Query query = session.createQuery("from Blog where subject like
'%" + key + "%'");
                        list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                        tx.commit();
                } catch (HibernateException e) {
                        if (tx!= null)
                                tx.rollback();
                        throw e;
                }
                HibernateUtil.closeSession();
                return list;
        }

        /**
         * 根据 ID 得到 Sort 对象
         */
        public Sort getSort(int Id) throws HibernateException {
                Session session = HibernateUtil.currentSession();
                Sort sort = null;
                Transaction tx = null;
                try {
                        tx = session.beginTransaction();
                        //创建查询对象
```

```
            Query query = session.createQuery("from Sort where id=" + Id);
            List list = query.list();
            if(!list.isEmpty())
                  sort = (Sort) list.get(0);
            tx.commit();
      } catch (HibernateException e) {
            if (tx!= null)
                  tx.rollback();
            throw e;
      }
      session.close();
      return sort;
}

/**
 * 根据 ID 得到 Links 对象
 */
public Links getLink(int Id) throws HibernateException {
      Session session = HibernateUtil.currentSession();
      Links link = null;
      Transaction tx = null;
      try {
            tx = session.beginTransaction();
            //创建查询对象
            Query query = session.createQuery("from Links where id=" + Id);
            List list = query.list();
            if(!list.isEmpty())
                  link = (Links) list.get(0);
            tx.commit();
      } catch (HibernateException e) {
            if (tx!= null)
                  tx.rollback();
            throw e;
      }
      session.close();
      return link;
}

/**
 * 得到指定文章对象
 */
public Article getArticle(int articleid) throws HibernateException {
      Session session = HibernateUtil.currentSession();
      Article article = null;
      Transaction tx = null;
      try {
            tx = session.beginTransaction();
            //创建查询对象
            Query query = session.createQuery("from Article where id=" +
```

```
articleid);
                    List list = query.list();
                    if(!list.isEmpty())
                        article = (Article) list.get(0);
                    tx.commit();
                }catch (HibernateException e) {
                    if (tx!= null)
                        tx.rollback();
                    throw e;
                }
                session.close();
                return article;
        }

        /**
         * 得到匹配 Article 对象集
         */
        public List getMatchArticles(String key) throws HibernateException {
                Session session = HibernateUtil.currentSession();
                List list=null;
                Transaction tx = null;
                try {
                    tx = session.beginTransaction();
                    //创建查询对象
                    Query query = session.createQuery("from Article where title like
'%" + key + "%'");
                    list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                    tx.commit();
                }catch (HibernateException e) {
                    if (tx!= null)
                        tx.rollback();
                    throw e;
                }
                HibernateUtil.closeSession();
                return list;
        }

        /**
         * 得到最新文章对象集
         */
        public List getArticles(int showCount) throws HibernateException {
                Session session = HibernateUtil.currentSession();
                List list=null;
                Transaction tx = null;
                try {
                    tx = session.beginTransaction();
                    //创建查询对象
                    Query query = session.createQuery("from Article order by pubtime
desc");
```

```
                if (showCount>0){
                        query.setMaxResults(showCount);//记录集最大个数
                }
                list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                tx.commit();
            }catch (HibernateException e) {
                if (tx!= null)
                        tx.rollback();
                throw e;
            }
            HibernateUtil.closeSession();
            return list;
        }

        /**
         * 得到指定博客最新文章对象集
         */
        public List getBlogArticles(int blogid,int sortid) throws HibernateException {
            Session session = HibernateUtil.currentSession();
            List list=null;
            Transaction tx = null;
            try {
                tx = session.beginTransaction();
                //创建查询对象
                String strSql = "from Article  where blogid=" + blogid;//创建一
个查询语句,查询指定类别产品;
                if (sortid>0){
                        strSql = strSql + " and sortid=" + sortid ;//记录集最大个数
                }
                strSql = strSql + " order by pubtime desc" ;//记录集最大个数
                Query query = session.createQuery(strSql);
                list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                tx.commit();
            } catch (HibernateException e) {
                if (tx!= null)
                        tx.rollback();
                throw e;
            }
            HibernateUtil.closeSession();
            return list;
        }

        /**
         * 得到指定博客文章分类对象集
         */
        public List getBlogSorts(int blogid) throws HibernateException {
            Session session = HibernateUtil.currentSession();
            List list=null;
            Transaction tx = null;
```

```
            try {
                tx = session.beginTransaction();
                //创建查询对象
                String strSql = "from Sort where blogid=" + blogid;//创建一个查
询语句,查询指定类别产品;
                Query query = session.createQuery(strSql);
                list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                tx.commit();
            }catch (HibernateException e) {
                if (tx!= null)
                    tx.rollback();
                throw e;
            }
            HibernateUtil.closeSession();
            return list;
        }

        /**
         * 得到指定博客超级链接对象集
         */
        public List getBlogLinks(int blogid) throws HibernateException {
            Session session = HibernateUtil.currentSession();
            List list=null;
            Transaction tx = null;
            try {
                tx = session.beginTransaction();
                //创建查询对象
                String strSql = "from Links where blogid=" + blogid;//创建一个
查询语句,查询指定类别产品;
                Query query = session.createQuery(strSql);
                list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                tx.commit();
            } catch (HibernateException e) {
                if (tx!= null)
                    tx.rollback();
                throw e;
            }
            HibernateUtil.closeSession();
            return list;
        }

        /**
         * 得到指定文章回复对象集
         */
        public List getFeedBacks(int articleid) throws HibernateException {
            Session session = HibernateUtil.currentSession();
            List list=null;
            Transaction tx = null;
            try {
```

```
                tx = session.beginTransaction();
                //创建查询对象
                String strSql = "from FeedBack  where articleid=" + articleid
                              + " order by pubtime desc" ;//创建一个查询语句,查
询指定类别产品;
                Query query = session.createQuery(strSql);
                list = query.list(); //从数据库取出数据，并自动封装到 List 集合中
                tx.commit();
            } catch (HibernateException e) {
                if (tx!= null)
                    tx.rollback();
                throw e;
            }
            HibernateUtil.closeSession();
            return list;
        }

        /**
         * 插入实体对象所对应的记录
         */
        public void save(Object obj) throws HibernateException {
            Session session = HibernateUtil.currentSession();
            if(obj!= null) {
                Transaction tx = null;
                try {
                    tx = session.beginTransaction();
                    session.save(obj);
                    tx.commit();
                } catch (HibernateException e) {
                    if(tx!=null)
                        tx.rollback();
                    throw e;
                }
            }
            session.close();
        }

        /**
         * 修改实体对象所对应的记录
         */
        public void update(Object obj) throws HibernateException {
            Session session = HibernateUtil.currentSession();

            if(obj!= null) {
                Transaction tx = null;
                try {
                    tx = session.beginTransaction();
                    session.update(obj);
                    tx.commit();
```

```
                } catch (HibernateException e) {
                        if(tx!=null)
                                tx.rollback();
                        throw e;
                }
        }
        session.close();
    }

    /**
     * 删除对象所对应的记录
     */
    public void delete(Object obj) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        if(obj!= null) {
            Transaction tx = null;
            try {
                tx = session.beginTransaction();
                session.delete(obj);
                tx.commit();
            } catch (HibernateException e) {
                if(tx!=null)
                    tx.rollback();
                throw e;
            }
        }
        session.close();
    }
}
```

在该类声明的与数据库操作相关的方法中，并没有使用传统的 SQL 语句完成与数据库相关的操作，而是借助 Hibernate 提供的 HQL 语句进行数据记录的增加、删除、修改以及查询的功能。这是 Hibernate 开发架构的优势。

7.3.5 业务逻辑层

在本系统的业务逻辑层中，分别完成了对博客信息、文章分类信息、文章信息、文章评论信息以及友情链接信息等类型对象的保存和处理。

注意：业务逻辑层对应的类文件应该放置在项目目录的 src 目录中。

1. 封装了博客基本信息的 bean（Blog.java）

……

- 与博客类型对象相关的基本属性和方法

```
public class Blog {
    private int id;
    private String username;
```

```
        private String password;
        private String subject;
        private String email;
        private String image;
        private int visitcount;
```

- 博客编号属性的 set 及 get 方法

```
public int getId() {
        return id;
}
public void setId(int id) {
        this.id = id;
}
```

- 博客密码属性的 set 及 get 方法

```
public String getPassword() {
        return password;
}
public void setPassword(String password) {
        this.password = password;
}
```

- 博客用户名属性的 set 及 get 方法

```
public String getUsername() {
        return username;
}
public void setUsername(String username) {
        this.username = username;
}
```

- 博客 logo 图片属性的 set 及 get 方法

```
public String getImage() {
        return image;
}
public void setImage(String image) {
        this.image = image;
}
```

- 博客昵称属性的 set 及 get 方法

```
public String getSubject() {
        return subject;
}
public void setSubject(String subject) {
        this.subject = subject;
}
```

- 博客电子邮件地址属性的 set 及 get 方法

```
public String getEmail() {
        return email;
}
public void setEmail(String email) {
        this.email = email;
```

```
}
```

- 博客页面访问次数属性的 set 及 get 方法

```
public int getVisitcount() {
        return visitcount;
}
public void setVisitcount(int visitcount) {
        this.visitcount = visitcount;
}
}
```

在上面有关博客信息封装处理的 bean 中，声明了与博客基本数据相关的基本属性，注意：其名称要与对应 Hibernate 映射文件中的属性名称以及数据表字段名称保持一致。

2. 封装了文章类别信息处理的 bean（Sort.java）

……

- 与文章类别对象相关的基本属性和方法

```
public class Sort {
        private int id;
        private String name;
        private int blogid;
```

- 文章类别编号属性的 set 及 get 方法

```
public int getId() {
        return id;
}
public void setId(int id) {
        this.id = id;
}
```

- 文章类别名称属性的 set 及 get 方法

```
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
```

- 文章类别所属博客编号属性的 set 及 get 方法

```
public int getBlogid() {
        return blogid;
}
public void setBlogid(int blogid) {
        this.blogid = blogid;
}
}
```

在上面有关文章分类信息封装处理的 bean 中，声明了与文章分类数据相关的基本属性和方法。

3. 封装了文章信息处理的 bean（Article.java）

……

- 文章对象的基本属性

```
public class Article {
        private int id;
        private String title;
        private String content;
        private String pubtime;
        private int blogid;
        private int sortid;
```

- 文章编号属性的 set 及 get 方法

```
public int getId() {
        return id;
}
public void setId(int id) {
        this.id = id;
}
```

- 文章所属分类编号属性的 set 及 get 方法

```
public int getSortid() {
        return sortid;
}
public void setSortid(int sortid) {
        this.sortid = sortid;
}
```

- 文章所属博客编号属性的 set 及 get 方法

```
public int getBlogid() {
        return blogid;
}
public void setBlogid(int blogid) {
        this.blogid = blogid;
}
```

- 文章内容属性的 set 及 get 方法

```
public String getContent() {
        return content;
}
public void setContent(String content) {
        this.content = content;
}
```

- 文章标题属性的 set 及 get 方法

```
public String getTitle() {
        return title;
}
public void setTitle(String title) {
        this.title = title;
}
```

- 文章发表时间属性的 set 及 get 方法

```
public String getPubtime() {
```

```
        return pubtime;
}
public void setPubtime(String pubtime) {
        this.pubtime = pubtime;
}

}
```

在上面有关文章信息封装处理的 bean 中，声明了与文章数据相关的基本属性和方法。

4. 封装了文章评论信息的 bean（FeedBack.java）

……

- 与文章评论信息相关的基本属性和方法

```
public class FeedBack {
        private int id;
        private String uname;
        private String content;
        private String pubtime;
        private int articleid;
```

- 文章评论编号属性的 set 及 get 方法

```
public int getId() {
        return id;
}
public void setId(int id) {
        this.id = id;
}
```

- 文章评论内容属性的 set 及 get 方法

```
public String getContent() {
        return content;
}
public void setContent(String content) {
        this.content = content;
}
```

- 文章评论所属文章编号属性的 set 及 get 方法

```
public int getArticleid() {
        return articleid;
}
public void setArticleid(int articleid) {
        this.articleid = articleid;
}
```

- 文章评论发表时间属性的 set 及 get 方法

```
public String getPubtime() {
        return pubtime;
}
public void setPubtime(String pubtime) {
        this.pubtime = pubtime;
}
```

- 文章评论发表用户名称属性的 set 及 get 方法

```
public String getUname() {
        return uname;
}
public void setUname(String uname) {
        this.uname = uname;
}

}
```

在上面有关文章评论信息封装处理的 bean 中，声明了文章评论信息中所需要处理的数据相关的基本属性和方法。

5. 封装了友情链接信息处理的 bean（Links.java）

……

- 与友情链接信息相关的基本属性和方法

```
public class Links {
        private int id;
        private String name;
        private String url;
        private int blogid;
```

- 友情链接编号属性的 set 及 get 方法

```
public int getId() {
        return id;
}
public void setId(int id) {
        this.id = id;
}
```

- 友情链接名称属性的 set 及 get 方法

```
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
```

- 友情链接所属博客编号属性的 set 及 get 方法

```
public int getBlogid() {
        return blogid;
}
public void setBlogid(int blogid) {
        this.blogid = blogid;
}
```

- 友情链接 URL 地址属性的 set 及 get 方法

```
    public String getUrl() {
            return url;
    }
    public void setUrl(String url) {
```

```
            this.url = url;
        }
    }
```

在上面有关友情链接信息封装处理的 bean 中，声明了与友情链接数据相关的基本属性和方法。

6. 常量文件（Constants.java）

在视图组件和 Action 之间经常需要利用 setAttribute()和 getAttribute()方法来传递 request 范围内或者 session 范围内的值，其基本语法是：

```
session.setAttribute(users,usersVector)
```

即将 usersVector 变量中存放的信息装入一个名为 users 的属性 key 中。后续处理中可以按照该 key 获取对应的信息。

在 Web 应用中，提倡将一些属性 key 常量定义在常量文件中，有利于提高请求处理的 Servlet 程序的独立性，例如：如果在程序中，当属性 key 的常量值发生改变，只需要修改常量文件即可，而不需要修改对应的 Servlet 程序，博客网站对应常量文件的代码如下：

```
package cn.com.blogonline;

public final class Constants {
        // Session keys
      public static final String LOGIN_USER_KEY = "LOGIN_USER";
       public static final String SORT_LIST_KEY = "SORT_LIST";
       public static final String LINK_LIST_KEY = "LINK_LIST";
       public static final String BLOG_LIST_KEY = "BLOG_LIST";
       public static final String ARTICLE_LIST_KEY = "ARTICLE_LIST";
       public static final String FEEDBACK_LIST_KEY = "FEEDBACK_LIST";
       public static final String VISITCOUNT_KEY = "VISITCOUNT";
       public static final String VISIT_BLOG_KEY = "OTHER_BLOG";

       public static final String CUR_ARTICLE_KEY = "CUR_ARTICLE";
       public static final String CUR_KEYWORD_KEY = "CUR_KEYWORD";
       public static final String CUR_BLOGID_KEY = "CUR_BLOGID";
       public static final String CUR_SORTID_KEY = "CUR_SORTID";
       public static final String CUR_PAGEID_KEY = "CUR_PAGEID";
       public static final String MODIFY_OK_KEY = "MODIFY_OK";

       public static final int ARTICLE_PAGE_SIZE = 2;
}
```

7. 验证码图片生成的文件（ImageTool.java）

该 JavaBean 是用于动态生成验证码图片的文件，对应代码及相关解释如下：

```
package p1;
……
public class ImageTool {
        public String sRand="";
        public Color getRandColor(int fc,int bc){//给定范围获得随机颜色
```

```
                Random random = new Random();
                if(fc>255) fc=255;
                if(bc>255) bc=255;
                int r=fc+random.nextInt(bc-fc);
                int g=fc+random.nextInt(bc-fc);
                int b=fc+random.nextInt(bc-fc);
                return new Color(r,g,b);
        }
        public BufferedImage creatImage(){
                // 在内存中创建图象
                int width=60, height=20;
                BufferedImage image = new BufferedImage(width, height, BufferedImage.
TYPE_INT_RGB);
                // 获取图形上下文
                Graphics g = image.getGraphics();
                //生成随机类
                Random random = new Random();
                // 设定背景色
                g.setColor(getRandColor(200,250));
                g.fillRect(0, 0, width, height);
                //设定字体
                g.setFont(new Font("Times New Roman",Font.PLAIN,18));
                //画边框
                //g.setColor(new Color());
                //g.drawRect(0,0,width-1,height-1);
                // 随机产生 155 条干扰线，使图象中的认证码不易被其他程序探测到
                for (int i=0;i<155;i++)
                {
                        g.setColor(getRandColor(160,200));
                        int x = random.nextInt(width);
                        int y = random.nextInt(height);
                        int xl = random.nextInt(12);
                        int yl = random.nextInt(12);
                        g.drawLine(x,y,x+xl,y+yl);
                }
                // 取随机产生的认证码(4 位数字)
                sRand = "";
                for (int i=0;i<4;i++){
                        String rand=String.valueOf(random.nextInt(10));
                        sRand+=rand;
                        // 将认证码显示到图象中
                        g.setColor(new  Color(20+random.nextInt(110),20+random.nextInt
(110),20+random.nextInt(110)));
    //调用函数出来的颜色相同，可能是因为种子太接近，所以只能直接生成
                        g.drawString(rand,13*i+6,16);
                }
                // 图象生效
                g.dispose();
                return image;
```

```
        }
    }
```

7.3.6 控制处理层

注意：控制处理层中 Servlet 对应的类文件也同样应该放置在项目目录下的 src 目录中。

1. web.xml 配置文件

该配置文件是 Web 应用程序通用的配置文件。主要完成 ActionServlet 的配置、默认首页的设置等工作。

```
……
   <servlet>
        <servlet-name>ms1</servlet-name>
        <servlet-class>cn.com.blogonline.RegistAction</servlet-class>
    </servlet>

  <servlet-mapping>
        <servlet-name>ms1</servlet-name>
        <url-pattern>/regist</url-pattern>
   </servlet-mapping>

   <servlet>
        <servlet-name>ms2</servlet-name>
        <servlet-class>cn.com.blogonline.CheckUnameAction</servlet-class>
  </servlet>

  <servlet-mapping>
        <servlet-name>ms2</servlet-name>
        <url-pattern>/checkUname</url-pattern>
   </servlet-mapping>

   <servlet>
        <servlet-name>ms3</servlet-name>
        <servlet-class>cn.com.blogonline.FirstAction</servlet-class>
   </servlet>

  <servlet-mapping>
        <servlet-name>ms3</servlet-name>
        <url-pattern>/first</url-pattern>
   </servlet-mapping>

   <servlet>
        <servlet-name>ms4</servlet-name>
        <servlet-class>cn.com.blogonline.LoginAction</servlet-class>
   </servlet>

  <servlet-mapping>
```

```
        <servlet-name>ms4</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ms5</servlet-name>
        <servlet-class>cn.com.blogonline.SearchAction</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ms5</servlet-name>
        <url-pattern>/search</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ms6</servlet-name>
        <servlet-class>cn.com.blogonline.OpenBlogAction</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ms6</servlet-name>
        <url-pattern>/openBlog</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ms7</servlet-name>
        <servlet-class>cn.com.blogonline.BlogOperateAction</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ms7</servlet-name>
        <url-pattern>/blogOperate</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ms8</servlet-name>
        <servlet-class>cn.com.blogonline.ArticleSortAction</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ms8</servlet-name>
        <url-pattern>/dyList</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>ms9</servlet-name>
        <servlet-class>cn.com.blogonline.ArticleSaveAction</servlet-class>
    </servlet>
```

```
  <servlet-mapping>
       <servlet-name>ms9</servlet-name>
       <url-pattern>/saveArticle</url-pattern>
   </servlet-mapping>

   <servlet>
       <servlet-name>ms10</servlet-name>
       <servlet-class>cn.com.blogonline.ShowFeedbackAction</servlet-class>
   </servlet>

  <servlet-mapping>
       <servlet-name>ms10</servlet-name>
       <url-pattern>/showFeedback</url-pattern>
   </servlet-mapping>

   <servlet>
       <servlet-name>ms11</servlet-name>
       <servlet-class>cn.com.blogonline.WriteFeedbackAction</servlet-class>
   </servlet>

  <servlet-mapping>
       <servlet-name>ms11</servlet-name>
       <url-pattern>/writeFeedback</url-pattern>
   </servlet-mapping>

   <servlet>
       <servlet-name>ms12</servlet-name>
       <servlet-class>cn.com.blogonline.OpenArticleAction</servlet-class>
   </servlet>

  <servlet-mapping>
       <servlet-name>ms12</servlet-name>
       <url-pattern>/openArticle</url-pattern>
   </servlet-mapping>
……
<!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

2. 负责处理新博客注册请求的 Servlet 程序（RegistAction.java）

该程序负责处理默认新博客注册页面提交的“regist”请求。

```
……
public class RegistAction extends HttpServlet {
……
     protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
```

```
        //设置接收信息的字符集
        request.setCharacterEncoding("UTF-8");
        //接收浏览器端提交的信息
              String uname = request.getParameter("uname");
              String psw = request.getParameter("psw");
              String subject = request.getParameter("subject");
              String email = request.getParameter("email");
        //保存信息
              DbOperate db=new DbOperate();
              Blog blog=new Blog();
              blog.setUsername(uname);
              blog.setPassword(psw);
              blog.setSubject(subject);
              blog.setEmail(email);
              blog.setImage("default_icon.jpg");
              db.save(blog);
             HttpSession session = request.getSession(true);
              session.setAttribute(Constants.LOGIN_USER_KEY,blog);
              response.sendRedirect("/BlogOnline/registOK.jsp");

        }

        protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
throws ServletException, IOException {
              this.doGet(arg0,arg1);
        }
    ……
```

3. 负责处理注册时,检查用户名是否被占用请求的 Servlet 程序(CheckUnameAction.java)

该程序负责处理在进行新博客注册时，检查输入的用户名是否已经被占用时提交的“checkUname”请求。

```
    ……
    public class CheckUnameAction extends HttpServlet {
    ……
        protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
         //设置接收信息的字符集
         request.setCharacterEncoding("UTF-8");
         //接收浏览器端提交的信息
              String uname = request.getParameter("uname");
              //设置输出信息的格式及字符集
           response.setContentType("text/xml; charset=UTF-8");
           response.setHeader("Cache-Control", "no-cache");
           //创建输出流对象
           PrintWriter out = response.getWriter();
           //检查用户名，并输出结果
         DbOperate  db=new DbOperate();
              Blog blog=db.getBlog(uname);
```

```
            if (blog==null) {
                    out.println("0");
            }else {
                    out.println("1");
            }
            out.close();
    ……
```

4. 负责处理首页显示信息并实现自动刷新请求的 Servlet 程序（FirstAction.java）

该程序负责处理网站首页提交或获取相关信息的请求，例如，热门博客信息、最新文章信息提交的“first”请求。

```
    ……
    public class FirstAction extends HttpServlet {
    ……
        protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
           //利用 hibernate 查询热点博客和最新文章
                    DbOperate    db=new DbOperate();
                    List blogList=db.getBlogs(10);
                    List artileList=db.getArticles(10);

          //设置输出信息的格式及字符集
           response.setContentType("text/xml; charset=UTF-8");
           response.setHeader("Cache-Control", "no-cache");
           //创建输出流对象
           PrintWriter out = response.getWriter();
           //依据验证结果输出不同的数据信息
           out.println("<response>");
                 // 输出热点博客
                    Blog curBlog=null;
                    for (int i=0;i<blogList.size();i++){
                            curBlog = (Blog)blogList.get(i);
                           out.println("<blog>");
                            out.println("<id>" + curBlog.getId() + "</id>");
                            out.println("<name>" + curBlog.getSubject() + "</name>");
                           out.println("</blog>");
                    }
                // 输出最新文章
                    Article curArticle=null;
                    for (int i=0;i<artileList.size();i++){
                            curArticle = (Article)artileList.get(i);
                           out.println("<article>");
                            out.println("<id>" + curArticle.getId() + "</id>");
                            out.println("<title>"    +    curArticle.getTitle()    +
"</title>");
                            out.println("<time>"    +    curArticle.getPubtime()    +
"</time>");
                           out.println("</article>");
```

```
            }

            out.println("</response>");
            out.close();
    }
```

……

5. 负责处理博客登录请求的 Servlet 程序（LoginAction.java）

该程序负责处理从网站首页面提交的“login”请求，包括对提交的信息进行验证。

```
……
public class LoginAction extends HttpServlet {
……
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
            //设置接收信息的字符集
            request.setCharacterEncoding("UTF-8");
            //接收浏览器端提交的信息
                String uname = request.getParameter("uname");
                String psw = request.getParameter("psw");
                String checkwd = request.getParameter("checkwd");
                HttpSession session = request.getSession();
       String checkwd2 = (String)session.getAttribute("rand");
                //设置输出信息的格式及字符集
       response.setContentType("text/xml; charset=UTF-8");
       response.setHeader("Cache-Control", "no-cache");
       //创建输出流对象
       PrintWriter out = response.getWriter();
       //依据验证结果输出不同的数据信息
       out.println("<response>");
                Blog blog=new DbOperate().getBlog(uname);
                if (blog==null){          //用户名错误
                out.println("<res>1</res>");
                }
                else{
                      String dbPassword = blog.getPassword();
                      if(!dbPassword.equals(psw)) {  //口令错误
                            out.println("<res>2</res>");
                      }
                      else if(!checkwd.equals(checkwd2)) {  //验证码错误
                            out.println("<res>3</res>");
                      }
                      else{
                            session.setAttribute(Constants.LOGIN_USER_KEY,blog);
                            out.println("<res>0</res>");
                            out.println("<id>" + blog.getId() + "</id>");
                      }
                }
                out.println("</response>");
```

```
                    out.close();
    ……
```

6. 负责处理搜索文章详细信息请求的 Servlet 程序（SearchAction.java）

该程序负责处理按照文章标题关键字进行搜索时提交的“search”请求。

```
    ……
    public class SearchAction extends HttpServlet {
    ……
        protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
         //设置接收信息的字符集
         request.setCharacterEncoding("UTF-8");
         //接收浏览器端提交的信息
              String key = request.getParameter("key");
            HttpSession session = request.getSession(true);
           //利用 hibernate 查询匹配文章
              DbOperate db=new DbOperate();
              List articleList=db.getMatchArticles(key);
              session.setAttribute(Constants.ARTICLE_LIST_KEY,articleList);
            response.sendRedirect("/BlogOnline/searchResult.jsp");
       }
    ……
```

7. 负责处理显示博客个人主页请求的 Servlet 程序（OpenBlogAction.java）

该程序负责处理显示博客个人主页时提交的“openBlog”请求。

```
    ……
    public class OpenBlogAction extends HttpServlet {
    ……
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
         //设置接收信息的字符集
         request.setCharacterEncoding("UTF-8");
         //接收浏览器端提交的信息
              String blogId = request.getParameter("blogid");
              int iBlogId =  Integer.parseInt(blogId);
              /*
               *左侧框架页信息设置
               */
            HttpSession session = request.getSession(true);
              DbOperate db=new DbOperate();

              Blog blog = db.getBlog(iBlogId);
              blog.setVisitcount(blog.getVisitcount() + 1);
              db.update(blog);
              session.setAttribute(Constants.VISIT_BLOG_KEY,blog);
              List sortList=db.getBlogSorts(iBlogId);
              session.setAttribute(Constants.SORT_LIST_KEY,sortList);
              List linkList=db.getBlogLinks(iBlogId);
```

```
            session.setAttribute(Constants.LINK_LIST_KEY,linkList);
            List articleList=db.getBlogArticles(iBlogId,0);
           /*
            *分页显示
            */
        List  dispList=new ArrayList();
        for(int i=0;i<articleList.size();i++){
                    if (i<Constants.ARTICLE_PAGE_SIZE){
                          dispList.add(articleList.get(i));
                    }
        }
            session.setAttribute(Constants.ARTICLE_LIST_KEY,dispList);
            session.setAttribute(Constants.CUR_PAGEID_KEY,new Integer(0));
            session.setAttribute(Constants.CUR_SORTID_KEY,new Integer(0));
            response.sendRedirect("/BlogOnline/blogMain.jsp");
    }
}
```

8. 负责处理博客个人主页分页显示请求的 Servlet 程序（BlogOperateAction.java）

该程序负责处理博客个人主页中，实现分页显示时提交的“blogOperate”请求。

```
……
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String blogId = request.getParameter("blogid");
          String sortId = request.getParameter("sortid");
          String pageId = request.getParameter("pageid");

          HttpSession session = request.getSession(true);
          int iBlogId = 0;
          DbOperate  db=new DbOperate();
          if (blogId!=null) {
                  iBlogId = Integer.parseInt(blogId);
                  Blog blog = db.getBlog(iBlogId);
                  session.setAttribute(Constants.VISIT_BLOG_KEY,blog);
          }
          else {
                  Blog blog = (Blog)session.getAttribute(Constants.VISIT_BLOG_KEY);
                  iBlogId = blog.getId();
          }
          int iSortId = 0;
          if (sortId!=null) {
                  iSortId = Integer.parseInt(sortId);
                  session.setAttribute(Constants.CUR_SORTID_KEY,new
Integer(iSortId));
          }
          else{
```

```
                    Integer temp = (Integer)session.getAttribute(Constants.CUR_SORTID_KEY);
                    iSortId = temp.intValue();
                }
                /*
                 * 页号处理
                 */
                int iPageId = 0;
                int pageCount = 0;
                List articleList=db.getBlogArticles(iBlogId,iSortId);
                if (pageId!=null) {
                        iPageId = Integer.parseInt(pageId);
                        if (iPageId<0) iPageId = 0;
                      /*
                       * 获取总页数
                       */
                      if (articleList.size()%Constants.ARTICLE_PAGE_SIZE ==0){
                                pageCount=articleList.size() / Constants.ARTICLE_PAGE_SIZE;
                       }
                      else{
                       pageCount=articleList.size() / Constants.ARTICLE_PAGE_SIZE+1;
                      }
                       if (iPageId>=pageCount) iPageId = pageCount-1;
                       session.setAttribute(Constants.CUR_PAGEID_KEY,new Integer(iPageId));
                }
                else {
                       Integer temp = (Integer)session.getAttribute(Constants.CUR_PAGEID_KEY);
                       iPageId = temp.intValue();
                }
               /*
                * 分页显示
                */
        List  dispList=new ArrayList();
             if((articleList.size()>iPageId * Constants.ARTICLE_PAGE_SIZE )&&(iPageId>=0)){
               for (int i=iPageId*Constants.ARTICLE_PAGE_SIZE;i<(iPageId+1)*Constants.
ARTICLE_PAGE_SIZE;i++){
                            if (i<articleList.size()){
                                  dispList.add(articleList.get(i));
                            }
             }
          }
            session.setAttribute(Constants.ARTICLE_LIST_KEY,dispList);
          response.sendRedirect("/BlogOnline/main.jsp");
    ……
```

9. 负责处理发表文章时，获取文章分类请求的 Servlet 程序（ArticleSortAction.java）

该程序负责处理发表新文章、选择指定文章分类时，动态获取类别信息时提交的“dyList”请求。

```
    ……
      protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
  throws ServletException, IOException {
          //设置输出信息的格式及字符集
      response.setContentType("text/xml; charset=UTF-8");
      response.setHeader("Cache-Control", "no-cache");
        HttpSession session = request.getSession(true);
        Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
        /*
         *从数据库取出类别信息输出
         */
        DbOperate  db=new DbOperate();
        List sortList=db.getBlogSorts(blog.getId());
      //创建输出流对象
      PrintWriter out = response.getWriter();
      out.println("<response>");
   for(int i=0;i<sortList.size();i++){
         Sort curSort = (Sort)sortList.get(i);
         out.println("<sort>" );
         out.println("<id>" + curSort.getId() + "</id>");
         out.println("<name>" + curSort.getName() + "</name>");
         out.println("</sort>");
         }
         out.println("</response>");
         out.close();
   }
}
```

10. 负责处理发表文章时，存储文章信息请求的 Servlet 程序（ArticleSaveAction.java）

该程序负责处理发表新文章时，提交存储文章信息的“saveArticle”请求。

```
……
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    //设置接收信息的字符集
    request.setCharacterEncoding("UTF-8");
    //接收浏览器端提交的信息
         String title = request.getParameter("title");
         String sortid = request.getParameter("sortid");
         String content = request.getParameter("content");
         int iSortId = 0;
         if (sortid!=null) {
                 iSortId = Integer.parseInt(sortid);
         }
         DbOperate db=new DbOperate();
         /*
          * 获取 Blog 信息
          */
         HttpSession session = request.getSession(true);
         Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
         /*
          * 设置保存文章信息
```

```
            */
            Article article=new Article();
            article.setTitle(title);
            article.setSortid(iSortId);
            article.setBlogid(blog.getId());
            article.setContent(content);
            SimpleDateFormat df  = new SimpleDateFormat("MMddhhmmss");
            df  = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
            article.setPubtime( df.format(new Date()));
            db.save(article);
            /*
             *返回结果信息
             */
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
            out.println("发表成功！");
            out.close();
    }
……
```

11. 负责处理显示文章评论信息请求的 Servlet 程序（ShowFeedbackAction.java）

该程序负责处理显示文章评论信息提交的“showFeedback”请求。

```
……
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String articleId = request.getParameter("articleid");
          String pageId = request.getParameter("pageid");
        /*
         * 文章编号处理
         */
         HttpSession session = request.getSession(true);
         DbOperate  db=new DbOperate();
         int iArticleId = 0;
         Article article = null;
         if (articleId!=null) {
                 iArticleId = Integer.parseInt(articleId);
                 article=db.getArticle(iArticleId);
                 session.setAttribute(Constants.CUR_ARTICLE_KEY,article);
         }
         else{
                 article=(Article)session.getAttribute(Constants.CUR_ARTICLE_KEY);
                 iArticleId = article.getId();
         }
         ……
```

```
            /*
             * 分页显示
             */
    ……
```

12. 负责处理发表评论请求的 Servlet 程序（WriteFeedbackAction.java）

该程序负责处理发表新评论提交的“writeFeedback”请求。

```
……
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      //设置接收信息的字符集
      request.setCharacterEncoding("UTF-8");
      //接收浏览器端提交的信息
            String uname = request.getParameter("uname");
            String content = request.getParameter("content");
          HttpSession session = request.getSession(true);
            Article article = (Article)session.getAttribute(Constants.CUR_ARTICLE_KEY);
        /*
         * 保存至数据库
         */
          DbOperate db=new DbOperate();
          FeedBack feedBack=new FeedBack();
          feedBack.setUname(uname);
          feedBack.setContent(content);
          SimpleDateFormat df  = new SimpleDateFormat("MMddhhmmss");
          df  = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
          feedBack.setPubtime( df.format(new Date()));
          feedBack.setArticleid(article.getId());
          db.save(feedBack);
          response.sendRedirect("showFeedback?articleid="+article.getId() + "&pageid=0");
    }
……
```

13. 负责处理显示文章请求的 Servlet 程序（OpenArticleAction.java）

该程序负责处理单击推荐文章列表超级链接时，提交的“openArticle”请求。

```
……
public class OpenArticleAction extends HttpServlet {
……
    protected  void  doGet(HttpServletRequest  request,  HttpServletResponse
response)
    throws ServletException, IOException {
      //设置接收信息的字符集
      request.setCharacterEncoding("UTF-8");
      //接收浏览器端提交的信息
            String articleId = request.getParameter("articleid");
          HttpSession session = request.getSession(true);
            int iArticleId = Integer.parseInt(articleId);
            /*
             * 保存文章对象及 BlogID
```

```
            */
           DbOperate  db=new DbOperate();
            Article article = db.getArticle(iArticleId);
            int iBlogId = article.getBlogid();
            session.setAttribute(Constants.CUR_ARTICLE_KEY,article);
            session.setAttribute(Constants.CUR_BLOGID_KEY,new Integer(iBlogId));

            /*
             *左侧框架页信息设置
             */
            Blog blog = db.getBlog(iBlogId);
            blog.setVisitcount(blog.getVisitcount() + 1);
            db.update(blog);
            session.setAttribute(Constants.VISIT_BLOG_KEY,blog);

            List sortList=db.getBlogSorts(iBlogId);
            session.setAttribute(Constants.SORT_LIST_KEY,sortList);

            List linkList=db.getBlogLinks(iBlogId);
            session.setAttribute(Constants.LINK_LIST_KEY,linkList);

            List feedBackList=db.getFeedBacks(iArticleId);
           /*
            * 分页显示
            */
      List  dispList=new ArrayList();
      for (int i=0;i<feedBackList.size();i++){
                  if (i< Constants.ARTICLE_PAGE_SIZE){
                        dispList.add(feedBackList.get(i));
                  }
           }
            session.setAttribute(Constants.FEEDBACK_LIST_KEY,dispList);
            session.setAttribute(Constants.CUR_PAGEID_KEY,new Integer(0));
           response.sendRedirect("/BlogOnline/otherBlogMain.jsp");

      }
}
```

7.3.7 技术难点详解

1. 脚本文件的导入

在应用 Ajax 技术时，需要用到大量的脚本，一般将对应的脚本单独封装在扩展名为.js 的脚本文件中，然后在对应的页面中进行导入即可。

```
<script language="javascript" src="images/index.js" type="text/javascript"></script>
```

本例实现的方式是，通过页码计算出所需要传递到页面中显示的记录范围及记录数据，从而进行显示。

2. 数据表之间的数据参照完整性

在设计数据库时，一定要考虑到数据表之间的关系，在本系统中，无论是博客表与文章表之间，还是文章类别表与文章表之间，以及文章表和文章评论表之间都需要建立外键约束，以确保数据的参照完整性，同时也便于后续的关联查询。

因此，无论在保存数据时，还是处理数据时，都应该分工明确，良好的分层处理可以使得程序更利于维护、升级。同时，在借助 Hibernate 进行数据库相关操作时，也要考虑与数据参照完整性相关的信息。

3. 中文信息提交时的乱码问题

当浏览器用户端提交的信息包含中文内容时，往往存入数据库时会显示为乱码，这将直接影响到处理结果在表示层页面中的正常显示，这是由于用户端和服务器数据库采用了不同的字符集所致。本例为解决这个问题，在借助 Ajax 技术提交时，采用 POST 方式进行提交，并按照本书第一部分的介绍，在各个环节均进行了字符集的统一设置。

4. 图片的动态处理

在动态网页中，经常需要动态控制图片的显示，如显示对应博客 logo 图片的显示等。一般采取的方式是，在对应的记录中，设置用于存放图片存储位置路径信息的字段，然后将图片放在对应位置，并按照规律命名，这样在需要动态控制其显示时，只需要依据对应字段中存放的地址，到指定位置获取图片进行显示即可。

5. 分页显示处理

数据记录的分页显示在动态网页中使用频率非常高，代码如下：

```
        List  dispList=new ArrayList();
        if((articleList.size()>iPageId * Constants.ARTICLE_PAGE_SIZE )&&(iPageId>=0)){
              for(int   i=iPageId*Constants.ARTICLE_PAGE_SIZE;i<(iPageId+1)*Constants.
ARTICLE_PAGE_SIZE;i++){
                    if(i<articleList.size()){
                          dispList.add(articleList.get(i));
                    }
              }
         }
    session.setAttribute(Constants.ARTICLE_LIST_KEY,dispList);
```

本例中实现的方式是，通过页码计算出所需要传递到页面中显示的记录范围及记录数据，从而进行显示。

CHAPTER 8

第 8 章　Blog Online 博客个人维护中心

为了让注册博客对文章信息、文章分类信息、链接信息以及个人基本信息等实现动态的维护管理，即通过直观的页面提示就可以直接完成对数据库中数据信息的增加、删除、修改及查询等工作。Blog Online 博客网站提供了专用的个人页面的管理维护中心，博客在身份验证通过后进入维护管理页面，通过动态页面，实现对数据库中博客个人页面相关信息的维护及管理。

本章通过一个完整的案例，详细介绍了 Blog Online 博客在线网站个人维护中心系统，具体为需求分析、总体设计、功能设计、数据库设计等完整过程，在介绍详细设计时，按功能模块分别介绍了具体的实现过程。与博客显示页面实现效果基本类似的模块，本章中将不再重复介绍，读者可以参阅附带光盘中程序的源代码。

8.1　系统设计

8.1.1　需求分析

典型的博客网站个人维护管理系统包括：文章信息管理、文章类别管理、链接信息管理及博客个人信息维护管理等功能。

博客网站个人维护管理系统的用例图如图 8-1 所示。

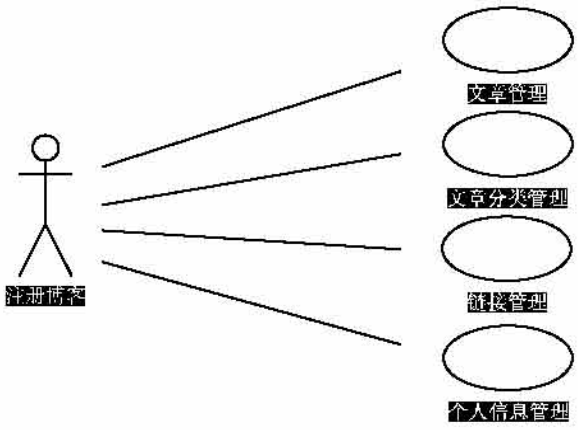


图 8-1　博客网站个人维护管理系统用例图

根据博客网站个人维护管理的基本需求，本系统需要完成的具体任务如下。

- 文章管理：通过该模块，博客可以对个人页面中的文章进行查询、删除等操作；
- 文章分类管理：博客可以借助该模块，实现对个人页面中的文章类别进行添加、查询及删除等操作；

- 链接管理：博客可以借助该模块，实现对个人页面中的友情链接列表进行添加、查询及删除等操作；
- 个人信息管理：博客可以借助该模块，实现对个人页面中的个人基本信息进行编辑修改，包括基本文字信息和 Logo 图片信息的编辑修改。

8.1.2 总体设计

表 8-1 所示为构成博客网站个人维护管理系统的各个组件，通过本表，读者可以从宏观上了解各个组件的功能。

表 8-1 系统组成表

用户表示层（视图）		控制处理层（控制器）	数据持续层	业务逻辑层（模型）
JSP 页面	Ajax 技术层			
blogAdminMain.jsp adminTop.jsp adminLeft.jsp adminMain.jsp adminArticle.jsp adminSort.jsp adminLink.jsp adminSelf.jsp adminLogo.jsp	adminSort.js adminLink.js adminArticle.js	AdminLinksAction.java AdminArticleAction.java AdminSortAction.java SaveLogoAction.java SaveSelfInfoAction.java	HibernateUtil.java DbOperate.java hibernate.cfg.xml model.hbm.xml	ImageTool.java Sort.java Blog.java Article.java FeedBack.java Links.java Constants.java

8.1.3 功能设计

1. 文章管理

该模块实现对博客对文章信息的管理，博客可以随时对现有的文章进行分类、查询、删除等操作。

2. 文章分类管理

该模块实现博客文章分类的管理，博客可以随时增加新的文章类别，也可以根据需要，对已有的文章类别进行删除等操作。

3. 链接管理

该模块实现博客对友情链接列表信息的管理，博客可以随时增加新的链接，也可以根据需要，对已有的链接进行删除等操作。

4. 个人信息管理

该模块实现博客对个人基本信息的管理，博客可以随时修改自己注册登记的相关信息，也可以根据需要，上传图片到服务器完成自己 Logo 图片的更换。

8.2 数据库设计与实现

8.2.1 数据库的需求分析

相关数据表已经在实现博客前台显示系统时进行了介绍，本章就不再赘述。

8.2.2 数据库的逻辑设计

博客个人维护管理中心对应各数据表的基本结构可参阅第 7 章中的相关数据表结构，本章不再赘述。

8.2.3 创建数据表的脚本

相关建立 MySQL 数据表的 SQL 脚本与第 7 章相同，这里不再赘述。

8.3 博客个人维护管理中心

8.3.1 实现效果

登录验证通过的博客，在个人主页中，单击主菜单中的“个人管理”超级链接，即可进入如图 8-2 所示的博客个人维护管理中心主页。注意：没有经过博客身份验证的其他用户和博客是不允许进入管理页面的。

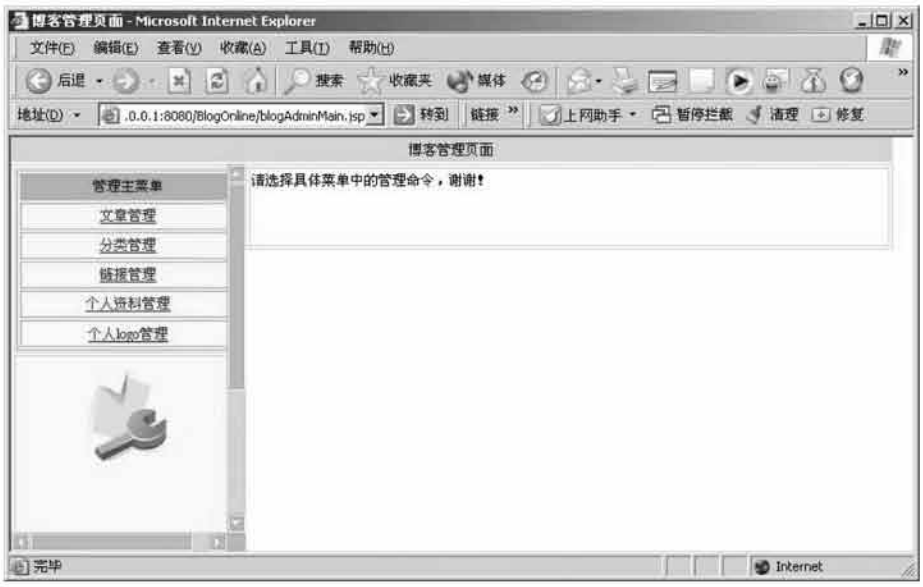


图 8-2 博客个人维护管理中心主页

该页面为框架页面，在左侧框架中选择相应的管理主菜单，右侧将依据选择的不同菜单，显示不同的博客操作界面，以便于博客选择管理的内容。

在管理主页面左侧的管理主菜单中选择“文章管理”，将显示如图 8-3 所示的文章管理页面，在该页面中，博客可以分类查找对应的文章，并对显示出的文章进行删除等操作。

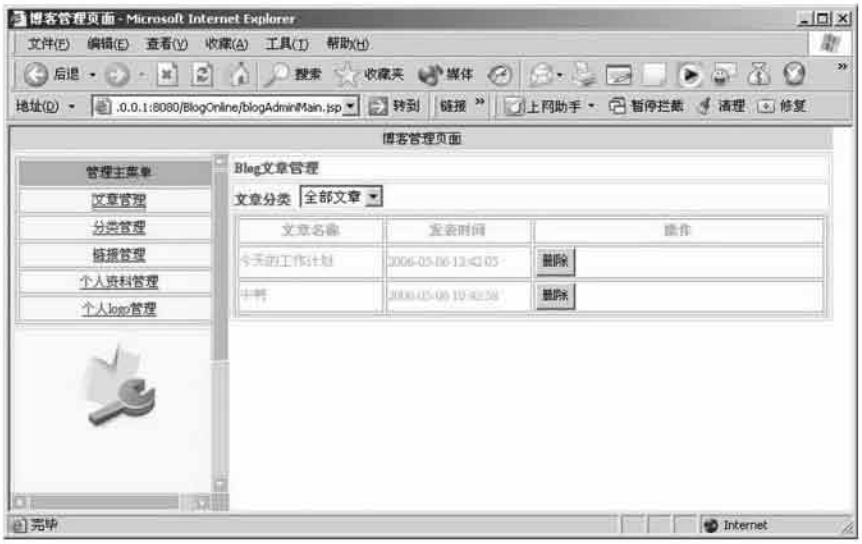


图 8-3 文章管理页面

在管理主页面左侧的管理主菜单中，选择“分类管理”，将显示如图 8-4 所示的文章分类管理页面，在该页面中，博客可以对文章分类信息进行添加、查询、删除等操作。

图 8-4 文章分类管理页面

在管理主页面左侧的管理主菜单中，选择“链接管理”，将显示如图 8-5 所示的链接管理页面，在该页面中，博客可以对链接列表信息进行添加、查询、删除等操作。

图 8-5　链接管理页面

在管理主页面左侧的管理主菜单中，选择“个人资料管理”，将显示如图 8-6 所示的博客个人资料修改管理页面，在该页面中博客可以对个人资料进行编辑修改。

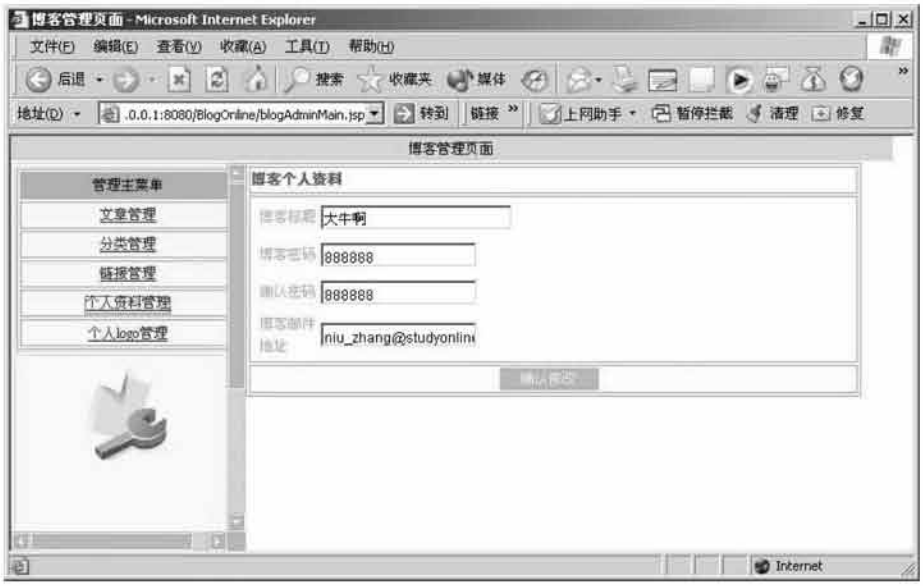


图 8-6　个人资料管理页面

在管理主页面左侧的管理主菜单中，选择“个人 Logo 管理”，将显示如图 8-7 所示的上传图片页面，在该页面中，博客可以通过上传图片更换个人页面中的 Logo 图片。

图 8-7　个人 Logo 管理页面

整个博客个人维护管理中心在运行过程中，页面的基本流程如图 8-8 所示。

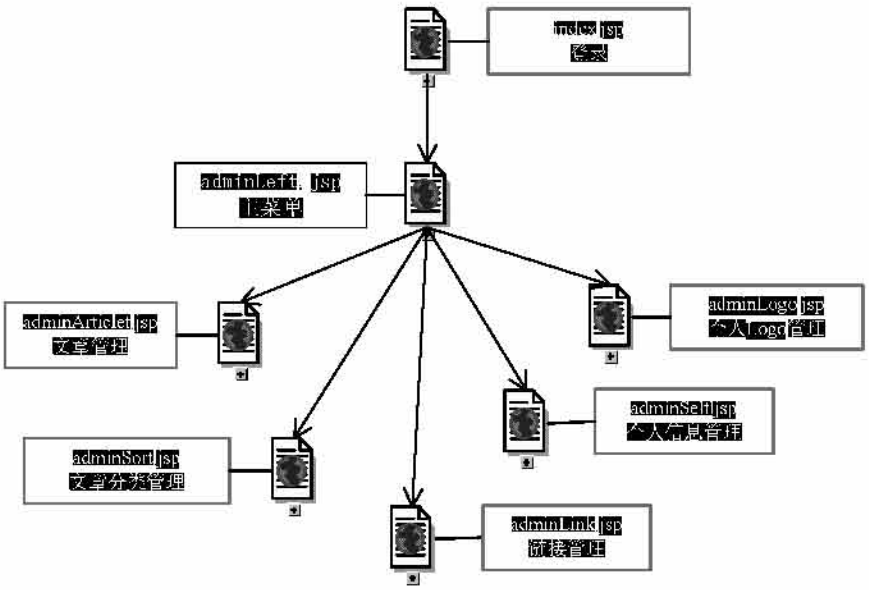


图 8-8　博客个人维护管理中心页面的基本流程图

8.3.2 用户表示层

在用户表示层中，主要是一些相关的 JSP 页面，用于实现图 8-2～8-7 所示的页面。该层中的 JSP 页面应该放在对应项目的 WebRoot 目录下。本章只给出相应 JSP 页面中最核心部分的代码，详细代码及效果可以参考本书附带光盘中的程序源代码。

1. 博客个人维护管理中心首页（blogAdminMain.jsp）

该页面为框架页，整个页面一共分成三个部分，具体对应页面文件及功能如下。

- adminTop.jsp 页面：顶端页面，一般可以放置网站的 logo 或者广告条；
- adminLeft.jsp 页面：管理主菜单，提供博客进入不同维护管理界面的超级链接；
- adminMain.jsp 页面：一个欢迎页面，实际使用时，可以在该页面中加入相关的系统使用说明等信息。

2. 左侧框架页面（adminLeft.jsp）

在该页面中，提供了博客进入不同维护管理界面的超级链接。核心程序代码及相关解释如下：

```
……
                    <font color="#000000">管理主菜单</font>
……
<font color="#000000"><a href="adminArticle.jsp" target="main">文章管理</a></font>
                </td>
            </tr>
            <tr>
                <td width="100%" height="8" align="center">
<font color="#000000"><a href="adminSort.jsp" target="main">分类管理</a></font>
                </td>
            </tr>
            <tr>
                <td width="100%" height="8" align="center">
<font color="#000000"><a href="adminLink.jsp" target="main">链接管理</a></font>
                </td>
            </tr>
            <tr>
                <td width="100%" height="7" align="center">
<font color="#000000"><a href="adminSelf.jsp" target="main">个人资料管理</a></font>
                </td>
            </tr>
            <tr>
                <td width="100%" height="7" align="center">
<font color="#000000"><a href="adminLogo.jsp" target="main">个人 logo 管理
</a></font>
                </td>
……
```

在上面的页面中，如果博客单击“文章管理”超级链接，将跳转到“adminArticle.jsp”

页面，以进行文章信息的维护管理功能。如果博客单击“分类管理”超级链接，将跳转到“adminSort.jsp”页面，以进行文章分类信息的维护管理功能。如果博客单击“链接管理”超级链接，将跳转到“adminLink.jsp”页面，进行链接信息的维护管理功能。如果博客单击“个人资料管理”超级链接，将跳转到“adminSelf.jsp”页面，进行个人基本信息的维护管理功能。如果博客单击“个人 logo 管理”超级链接，将跳转到“adminLogo.jsp”页面，进行图片的上传，实现个人 Logo 的维护管理功能。

3. 文章信息管理页面（adminArticle.jsp）

本页面是在单击“文章管理”超级链接后跳转到的页面，在该页面中，博客可以分类显示文章信息列表，并删除对应的文章记录。核心程序代码如下：

```
……
        <script language="javascript" src="images/adminArticle.js" type="text/
javascript"></script>
……
   <body text="#FCD447" onload="loadRequest()">
……
                              <font color="#FF0000"><b>Blog 文章管理</b></font>
                        </td>.
   ……
                              <font color="#FF0000"><b>文章分类</b></font>
                              <select id="sort" onchange="sortChange()">
                              </select>
……
                                                    <td width="25%" align="center">
                                                          <b>文章名称</b>
                                                    </td>
                                                    <td width="25%" align="center">
                                                          <b>发表时间</b>
                                                    </td>
                                                    <td width="50%" colspan="2" align=
"center">
                                                          <b>操作</b>
                                                    </td>
                                              </tr>
                                              <tbody id="articleList"></tbody>
……
```

注意：在该页面中，使用了 Ajax 技术，在页面装载时通过 onload 事件调用了“loadRequest()”函数，该函数包含在 JavaScript 脚本文件“adminArticle.js”中。该函数借助 Ajax 提交“"adminArticle?action=load" ”请求，以获取服务器端数据库中的文章及分类信息，以进行文章列表信息和文章分类信息的动态装入工作。对应 HTML 中的 DOM 对象的名称分别为“sort”和“articleList”。

当用户在文章分类的下拉列表框中，选择对应的文章类别后，通过 onchange 事件调用“sortChange()”函数，在该函数中借助 Ajax 技术提交“"action=show" + "&sortid="+ sortid”

请求，以获取数据服务器中指定文章分类的文章信息列表，动态装入到名称为“articleList”的动态表格中。

用户单击动态表格中对应文章记录的删除按钮时，根据脚本文件“adminArticle.js”中的设定，将提交“"adminArticle?action=delete" + "&articleid=" + id”请求，以删除指定文章编号的文章信息，然后动态刷新名称为“articleList”的动态表格中的数据。

4. 文章分类信息管理页面（adminSort.jsp）

本页面是在单击“分类管理”超级链接后跳转到的页面，在该页面中，博客可以对文章的分类信息进行添加及删除等操作。核心程序代码如下：

```
……
    <script language="javascript" src="images/adminSort.js" type="text/javascript">
</script>
……
……
……
                              <b><font color="#FF0000">新分类名称</font></b><b><font
color="#FF0000">: </font></b>
                              <input style="BORDER-RIGHT: #e78d00 1px solid; BORDER-TOP:
#e78d00 1px solid; BORDER-LEFT: #e78d00 1px solid; WIDTH: 125px; BORDER-BOTTOM: #e78d00
1px solid; " name="name">
                              <input class="button-submit" type="submit" value="添加"
onclick="addSort()">
……
                                        <td width="25%" align="center">
                                            <b>分类名称</b>
                                        </td>
                                        <td width="50%" colspan="2" align="center">
                                            <b>操作</b>
                                        </td>
                                    </tr>
                                    <tbody id="sortList"></tbody>
……
```

该页面中同样使用了 Ajax 技术，在页面装载时通过 onload 事件调用了“loadRequest()”函数，该函数包含在 JavaScript 脚本文件“adminSort.js”中。该函数借助 Ajax 提交“"adminSort?action=load"”请求，以获取服务器端数据库中的文章分类信息，进行文章分类信息的动态装入工作。对应 HTML 中的 DOM 对象的名称分别为“sortList”。

当用户在新分类名称的文本框中，输入待增加的分类名称之后，单击“添加”按钮，将调用“addSort()”函数，该函数将提交“"action=add" + "&name="+ name”请求，将对应的分类信息保存到数据库对应的分类信息表中。为了解决中文信息的乱码问题，这里采用的是 POST 方式提交数据信息。

当用户单击动态表格中对应分类记录的删除按钮时，根据脚本文件“adminSort.js”中的设定，将提交“"adminSort?action=delete" + "&id=" + id”请求，以删除指定分类编号的分类

信息，然后动态刷新名称为“sortList”的动态表格中的数据。

5. 链接信息管理页面（adminLink.jsp）

本页面是在单击“链接管理”超级链接后跳转到的页面，在该页面中，博客可以对博客页面中的友情链接信息列表进行添加及删除等操作。核心程序代码如下：

```
……
        <script language="javascript" src="images/adminLink.js" type="text/javascript">
</script>
……
   <body text="#FCD447" onload="loadRequest() ">
……
                              <b><font color="#FF0000">新链接名称：</font></b>
                              <input style="BORDER-RIGHT: #e78d00 1px solid; BORDER-TOP:
#e78d00 1px solid; BORDER-LEFT: #e78d00 1px solid; WIDTH: 125px; BORDER-BOTTOM: #e78d00
1px solid; " name="name">
                              <b><font color="#FF0000">地址：</font></b>
                              <input style="BORDER-RIGHT: #e78d00 1px solid; BORDER-TOP:
#e78d00 1px solid; BORDER-LEFT: #e78d00 1px solid; WIDTH: 125px; BORDER-BOTTOM: #e78d00
1px solid; " name="url">
                              <input class="button-submit" type="submit" value="添加"
style="width: 40; height: 20" onclick="addLink() ">
……
                                                  <td width="25%" align="center">
                                                      <b>链接名称</b>
                                                  </td>
                                                  <td width="46%" align="center">
                                                      <b>链接地址</b>
                                                  </td>
                                                  <td width="29%" colspan="2" align="center">
                                                      <b>操作</b>
                                                  </td>
                                              </tr>
                                              <tbody id="linkList"></tbody>
……
```

该页面中同样使用了 Ajax 技术，在页面装载时，通过 onload 事件调用了“loadRequest()”函数，该函数包含在 JavaScript 脚本文件“adminLink.js”中。该函数借助 Ajax 提交“"adminLinks?action=load"”请求，以获取服务器端数据库中的链接信息，进行链接信息的动态装入工作。对应 HTML 中的 DOM 对象的名称分别为“linkList”。

当用户在新链接名称及链接地址文本框中，输入待增加的信息之后，单击“添加”按钮，将调用“addLink()”函数，在该函数中，将提交“"action=add" + "&name="+ name + "&url="+ url1”请求，将对应的链接信息保存到数据库对应的链接信息表中。为了解决中文信息的乱码问题，这里采用的是 POST 方式提交数据信息。

当用户单击动态表格中删除按钮时，根据脚本文件“adminLink.js”中的设定，将提交“"adminLinks?action=delete" + "&id=" + id”请求，以删除指定链接编号的链接信息，然后动

态刷新名称为“linkList”的动态表格中的数据。

6. 个人基本信息管理页面（adminSelf.jsp）

本页面是在单击“个人资料管理”超级链接后跳转到的页面，在该页面中，博客可以对个人基本信息编辑修改。核心程序代码如下：

```
……
<%
    Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
    Integer modiFlag = (Integer)session.getAttribute(Constants.MODIFY_OK_KEY);
    if (modiFlag!=null){
        out.println("修改成功！");
        session.removeAttribute(Constants.MODIFY_OK_KEY);
     }
%>
……
                                    <b><font color="#FF0000">博客个人资料</font></b>
……
……                                  <form method="post" action="saveSelf">
                                            博客标题
                                        </td>
                                        <td>
                                    <input  class="text"  size="26"  name="subject"
value="<%=blog.getSubject()%>">
……
                                            博客密码
                                        </td>
                                        <td>
                                    <input class="text" size="20" name="psw" value="
<%=blog.getPassword()%>">
……
                                        <td class="redfont" nowrap width="10%"
height="30">
                                            确认密码
                                        </td>
                                        <td>
……                                  <input class="text" size="20" name= "psw2"value=
"<%=blog.getPassword()%>">
                                        <td class="redfont" height="30">
                                            博客邮件地址
                                        </td>
                                        <td width="90%">
                                    <input   class="text"   size="20"   name="email"
value="<%=blog.getEmail()%>">
……
                                <input class="button-submit" id="save" type="submit"
 value="确认修改" name="save">
……
```

页面获取登录验证通过后，借助 session 的对应属性保存博客相关信息，然后在对应的表

单中动态装入。

当博客完成对应信息的修改，单击“确认修改”按钮之后，将提交“saveSelf”请求，以借助 Servlet 程序完成数据库中对应数据表信息的更新工作。

7. 个人 Logo 图片管理页面（adminLogo.jsp）

本页面是在单击“个人 Logo 管理”超级链接后跳转到的页面，在该页面中，博客可以实现图片上传，以更换自己的 Logo 图片。核心程序代码如下：

```
……
<%
    Integer modiFlag = (Integer)session.getAttribute(Constants.MODIFY_OK_KEY);
    if (modiFlag!=null){
        out.println("修改成功！");
        session.removeAttribute(Constants.MODIFY_OK_KEY);
    }
%>
……
……
                                    <form enctype="multipart/form-data" method
="post"action="saveLogo">

                                    <tr>
                                        <td class="redfont" valign="top"
height="30">
                                            博客图片
                                        </td>
                                        <td>
                                            <input type="file" name="image">
……
    <input class="button-submit" id="save" type="submit" value="确认修改" name="save">
……
```

当博客借助浏览按钮选择了对应的图片文件，单击“确认修改”按钮之后，将提交“saveLogo”请求以借助 Servlet 实现图片文件的上传。注意该表单中的 enctype="multipart/form-data"设置是为了确保文件信息的正确传递。

8.3.3 Ajax 技术相关脚本文件

由于本系统适当使用了 Ajax 技术，以帮助实现浏览器客户端和服务器之间的交互，达到异步通信的目的，从而给用户更好的感受，为配合具体实现，涉及到了相关的 JavaScript 脚本文件，这些文件均统一放置在项目文件夹下的“images”文件夹中，然后在对应的 JSP 页面中，使用对应的链接语句将其引入。

下面对博客个人维护中心相关的脚本文件进行逐一介绍。

1. 文章管理脚本文件（adminArticle.js）

该文件是在文章管理 adminArticle.jsp 页面中使用的脚本文件，主要用于处理装载文章、

分类信息、删除文章信息等请求。具体代码及相关解释如下：

```
……
    // 处理分类更改响应函数
    function sortChangeResponse() {
     if (XMLHttpReq.readyState == 4) { // 判断对象状态
           if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  updateArticleList();
           } else { //页面不正常
                 window.alert("您所请求的页面有异常。");
           }
       }
    }
    // 选择分类函数
    function sortChange() {
      var sortid = document.getElementById("sort").value;
       var url = "action=show" + "&sortid="+ sortid;
       createXMLHttpRequest();
       XMLHttpReq.onreadystatechange = sortChangeResponse;
       XMLHttpReq.open("POST", "adminArticle", true);
    XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
       XMLHttpReq.send(url);
    }

    function updateArticleList() {
          clearArticleList();
          loadArticleList();
    }

    // 删除链接函数
    function deleteArticle(id) {
         var url = "adminArticle?action=delete" + "&articleid=" + id;
         createXMLHttpRequest();
         XMLHttpReq.onreadystatechange = DeleteStateChange;
         XMLHttpReq.open("GET", url, true);
         XMLHttpReq.send(null);
    }
    // 处理删除链接响应函数
    function DeleteStateChange() {
    if (XMLHttpReq.readyState == 4) { // 判断对象状态
          if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                 deleteArticleList();
           } else { //页面不正常
                 window.alert("您所请求的页面有异常。");
           }
       }
    }

    function deleteArticleList() {
           deleteID = XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.
```

```
nodeValue;
            if (deleteID!=null){
                  var rowToDelete = document.getElementById(deleteID);
                  var articleList = document.getElementById("articleList");
                  articleList.removeChild(rowToDelete);
            }
      }

      // 页面装入请求函数
      function loadRequest() {
           var url = "adminArticle?action=load";
            createXMLHttpRequest();
            XMLHttpReq.onreadystatechange = loadResponse;
            XMLHttpReq.open("GET", url, true);
            XMLHttpReq.send(null);
       }
       // 页面装入处理函数
      function loadResponse() {
       if (XMLHttpReq.readyState == 4) { // 判断对象状态
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                  loadSortList();
                  loadArticleList();
            } else { //页面不正常
                  window.alert("您所请求的页面有异常。");
            }
         }
      }
      // 装入分类列表函数
      function loadSortList() {
         var option = document.createElement("option");
         option.appendChild(document.createTextNode("全部文章"));
          option.setAttribute("value", "0");
         sort.appendChild(option);
              var xmlDoc =XMLHttpReq.responseXML;
      var sorts = xmlDoc.getElementsByTagName("sort");
      var currentSort = null;
      for(var i = 0; i < sorts.length; i++) {
              currentSort = sorts[i];
              var id =currentSort.getElementsByTagName("id")[0].firstChild.nodeValue;
              var name =currentSort.getElementsByTagName("name")[0].firstChild.nodeValue;

                option = document.createElement("option");
                option.appendChild(document.createTextNode(name));
                  option.setAttribute("value", id);
                sort.appendChild(option);
             }
      }
      // 装入文章函数
      function loadArticleList() {
```

```
        var xmlDoc =XMLHttpReq.responseXML;
var articles = xmlDoc.getElementsByTagName("article");
var currentArticle = null;
for(var i = 0; i < articles.length; i++) {
        currentArticle = articles[i];
        var id =currentArticle.getElementsByTagName("id")[0].firstChild.nodeValue;
        var name =currentArticle.getElementsByTagName("name")[0].firstChild.nodeValue;
        var time =currentArticle.getElementsByTagName("time")[0].firstChild.nodeValue;
              addOneArticle(id,name,time);
   }
}
// 插入一行函数
function addOneArticle(id,name,time) {
     var row = document.createElement("tr");
     row.setAttribute("id", id);
     var cell = document.createElement("td");
     cell.appendChild(document.createTextNode(name));
     row.appendChild(cell);

     cell = document.createElement("td");
     cell.appendChild(document.createTextNode(time));
     row.appendChild(cell);

     var deleteButton = document.createElement("input");
     deleteButton.setAttribute("type", "button");
     deleteButton.setAttribute("value", "删除");
     deleteButton.onclick = function () { deleteArticle(id); };
     cell = document.createElement("td");
     cell.appendChild(deleteButton);
     row.appendChild(cell);

     document.getElementById("articleList").appendChild(row);
}

function clearArticleList() {
     var article = document.getElementById("articleList");
     while(article.childNodes.length > 0) {
         article.removeChild(article.childNodes[0]);
     }
}
```

2. 文章分类管理脚本文件（adminSort.js）

该文件是在文章分类管理 adminSort.jsp 页面中使用的脚本文件，主要用于处理装载文章分类信息、添加及删除文章信息等请求。具体代码及相关解释如下：

```
……
  // 处理增加类别响应函数
function AddStateChange() {
  if (XMLHttpReq.readyState == 4) { // 判断对象状态
        if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
```

```
                AddSortList();
        } else { //页面不正常
                window.alert("您所请求的页面有异常。");
        }
    }
}
// 增加类别函数
function addSort() {
   name = document.getElementById("name").value;
      if(name == "" ) {
          return;
      }
      var url = "action=add" + "&name="+ name;
      createXMLHttpRequest();
      XMLHttpReq.onreadystatechange = AddStateChange;
      XMLHttpReq.open("POST", "adminSort", true);
XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      XMLHttpReq.send(url);
}

  function AddSortList() {
       var AddID = XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.nodeValue;
       if (AddID!=null){
                addOneSort(AddID,name);
                //清空输入框
                document.getElementById("name").value = "";
       }
}

// 删除类别函数
function deleteSort(id) {
      var url = "adminSort?action=delete" + "&id=" + id;
      createXMLHttpRequest();
      XMLHttpReq.onreadystatechange = DeleteStateChange;
      XMLHttpReq.open("GET", url, true);
      XMLHttpReq.send(null);
}
// 处理删除响应函数
function DeleteStateChange() {
if (XMLHttpReq.readyState == 4) { // 判断对象状态
       if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
             deleteSortList();
       } else { //页面不正常
            window.alert("您所请求的页面有异常。");
       }
    }
}

function deleteSortList() {
```

```
    deleteID=XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.nodeValue;
    if (deleteID!=null){
          var rowToDelete = document.getElementById(deleteID);
          var sortList = document.getElementById("sortList");
          sortList.removeChild(rowToDelete);
    }
}

// 页面装入请求函数
function loadRequest() {
      var url = "adminSort?action=load";
       createXMLHttpRequest();
       XMLHttpReq.onreadystatechange = loadResponse;
       XMLHttpReq.open("GET", url, true);
       XMLHttpReq.send(null);
}
// 页面装入处理函数
function loadResponse() {
if (XMLHttpReq.readyState == 4) { // 判断对象状态
      if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
            loadSortList();
      } else { //页面不正常
           window.alert("您所请求的页面有异常。");
      }
    }
}
// 装入全部分类函数
function loadSortList() {
       var xmlDoc =XMLHttpReq.responseXML;
 var sorts = xmlDoc.getElementsByTagName("sort");
 var currentSort = null;
 for(var i = 0; i < sorts.length; i++) {
         currentSort = sorts[i];
         var id =currentSort.getElementsByTagName("id")[0].firstChild.nodeValue;
         var name =currentSort.getElementsByTagName("name")[0].firstChild.nodeValue;
              addOneSort(id,name);
         }
}
// 插入一行函数
function addOneSort(id,name) {
           var row = document.createElement("tr");
           row.setAttribute("id", id);
           var cell = document.createElement("td");
           cell.appendChild(document.createTextNode(name));
           row.appendChild(cell);

           var deleteButton = document.createElement("input");
           deleteButton.setAttribute("type", "button");
           deleteButton.setAttribute("value", "删除");
```

```
                deleteButton.onclick = function () { deleteSort(id); };
                cell = document.createElement("td");
                cell.appendChild(deleteButton);
                row.appendChild(cell);

                document.getElementById("sortList").appendChild(row);
    }
```

3. 链接管理脚本文件（adminLink.js）

该文件是在链接管理 adminLink.jsp 页面中使用的脚本文件，主要用于处理装载链接信息、添加及删除链接信息等请求。具体代码及相关解释如下：

```
……
    // 处理增加链接响应函数
    function AddStateChange() {
      if (XMLHttpReq.readyState == 4) { // 判断对象状态
            if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                AddLinkList();
            } else { //页面不正常
                window.alert("您所请求的页面有异常。");
            }
        }
    }
    // 增加链接函数
    function addLink() {
        var name = document.getElementById("name").value;
        var url1 = document.getElementById("url").value;
        if(name == "" ) {
            return;
        }
        var url = "action=add" + "&name="+ name + "&url="+ url1;
        createXMLHttpRequest();
        XMLHttpReq.onreadystatechange = AddStateChange;
        XMLHttpReq.open("POST", "adminLinks", true);
    XMLHttpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
       XMLHttpReq.send(url);
    }

    function AddLinkList() {
            var xmlDoc =XMLHttpReq.responseXML;
          var AddID = xmlDoc.getElementsByTagName("id")[0].firstChild.nodeValue;
          var                                AddName                                =
xmlDoc.getElementsByTagName("name")[0].firstChild.nodeValue;
          var AddUrl = xmlDoc.getElementsByTagName("url")[0].firstChild.nodeValue;
          if (AddID!=null){
                addOneLink(AddID,AddName,AddUrl);
                //清空输入框
              document.getElementById("name").value = "";
              document.getElementById("url").value = "";
          }
```

```
        }

        // 删除链接函数
        function deleteLink(id) {
            var url = "adminLinks?action=delete" + "&id=" + id;
            createXMLHttpRequest();
            XMLHttpReq.onreadystatechange = DeleteStateChange;
            XMLHttpReq.open("GET", url, true);
            XMLHttpReq.send(null);
        }
        // 处理删除链接响应函数
        function DeleteStateChange() {
        if (XMLHttpReq.readyState == 4) { // 判断对象状态
              if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                    deleteLinkList();
              } else { //页面不正常
                  window.alert("您所请求的页面有异常。");
              }
           }
        }

        function deleteLinkList() {
              deleteID = XMLHttpReq.responseXML.getElementsByTagName("id")[0].firstChild.
nodeValue;
              if (deleteID!=null){
                    var rowToDelete = document.getElementById(deleteID);
                    var linkList = document.getElementById("linkList");
                    linkList.removeChild(rowToDelete);
              }
       }

       // 页面装入请求函数
       function loadRequest() {
            var url = "adminLinks?action=load";
             createXMLHttpRequest();
             XMLHttpReq.onreadystatechange = loadResponse;
             XMLHttpReq.open("GET", url, true);
             XMLHttpReq.send(null);
       }
       // 页面装入处理函数
       function loadResponse() {
         if (XMLHttpReq.readyState == 4) { // 判断对象状态
              if (XMLHttpReq.status == 200) { // 信息已经成功返回，开始处理信息
                    loadLinkList();
               } else { //页面不正常
                     window.alert("您所请求的页面有异常。");
               }
           }
       }
```

```
// 装入全部链接函数
function loadLinkList() {
       var xmlDoc =XMLHttpReq.responseXML;
var links = xmlDoc.getElementsByTagName("link");
var currentLink = null;
for(var i = 0; i < links.length; i++) {
        currentLink = links[i];
        var id =currentLink.getElementsByTagName("id")[0].firstChild.nodeValue;
        var name =currentLink.getElementsByTagName("name")[0].firstChild.nodeValue;
        var url =currentLink.getElementsByTagName("url")[0].firstChild.nodeValue;
              addOneLink(id,name,url);
        }
}
// 插入一行函数
function addOneLink(id,name,url) {
            var row = document.createElement("tr");
            row.setAttribute("id", id);
            var cell = document.createElement("td");
            cell.appendChild(document.createTextNode(name));
            row.appendChild(cell);

            cell = document.createElement("td");
            cell.appendChild(document.createTextNode(url));
            row.appendChild(cell);

            var deleteButton = document.createElement("input");
            deleteButton.setAttribute("type", "button");
            deleteButton.setAttribute("value", "删除");
            deleteButton.onclick = function () { deleteLink(id); };
            cell = document.createElement("td");
            cell.appendChild(deleteButton);
            row.appendChild(cell);

            document.getElementById("linkList").appendChild(row);
 }
```

8.3.4 数据持续层

本系统数据持续逻辑层中对应的业务代码，已经在第 7 章对应的数据持续层中进行了展示，这里不再赘述。

8.3.5 业务逻辑层

本系统业务逻辑层中对应的业务代码，已经在第 7 章对应的业务逻辑层中进行了展示，这里不再赘述。

8.3.6 控制处理层

注意：控制处理层中 bean 对应的类文件也同样应该放置在项目目录下的 src 目录中。

1. web.xml 配置文件

该配置文件是 Web 应用程序通用的配置文件。主要完成 ActionServlet 的配置、默认首页的设置等工作。该文件与第 7 章中的类似，这里不再赘述。

2. 负责处理文章管理请求的 Servlet（AdminArticleAction.java）程序

该程序负责处理文章管理页面提交的“adminArticle”请求。以帮助完成文章及文章分类信息的载入、删除等操作。

```
……
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
     //设置接收信息的字符集
     request.setCharacterEncoding("UTF-8");
     //接收浏览器端提交的信息
          String action = request.getParameter("action");
          String articleid = request.getParameter("articleid");
          String sortid = request.getParameter("sortid");
          int iSortId = 0;
          if (sortid!=null) {
                   iSortId = Integer.parseInt(sortid);
          }
         HttpSession session = request.getSession(true);
          Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);

          //设置输出信息的格式及字符集
       response.setContentType("text/xml; charset=UTF-8");
       response.setHeader("Cache-Control", "no-cache");
       //创建输出流对象
       PrintWriter out = response.getWriter();
       //依据验证结果输出不同的数据信息
       out.println("<response>");
       /*
        * 执行数据库操作并返回结果
        */
          DbOperate  db=new DbOperate();
        if ("load".equals(action)){
            List sortList=db.getBlogSorts(blog.getId());
            for (int i=0;i<sortList.size();i++){
                    Sort sort =   (Sort)sortList.get(i);
               out.println("<sort>");
                    out.println("<id>" + sort.getId() + "</id>");
                    out.println("<name>" + sort.getName() + "</name>");
               out.println("</sort>");
            }
             List artList=db.getBlogArticles(blog.getId(),iSortId);
             for(int i=0;i<artList.size();i++){
                    Article article =(Article)artList.get(i);
              out.println("<article>");
                    out.println("<id>" + article.getId() + "</id>");
```

```
                out.println("<name>" + article.getTitle() + "</name>");
                out.println("<time>" + article.getPubtime() + "</time>");
            out.println("</article>");
        }
      }
      if ("show".equals(action)){
          List artList=db.getBlogArticles(blog.getId(),iSortId);
          for (int i=0;i<artList.size();i++){
          Article article =   (Article)artList.get(i);
        out.println("<article>");
            out.println("<id>" + article.getId() + "</id>");
            out.println("<name>" + article.getTitle() + "</name>");
            out.println("<time>" + article.getPubtime() + "</time>");
        out.println("</article>");
       }
    }
    else if ("delete".equals(action)){
         Article article = db.getArticle(Integer.parseInt(articleid));
         db.delete(article);
         out.println("<id>" + articleid + "</id>");
    }
         out.println("</response>");
         out.close();
}
……
```

3. 负责处理文章分类管理请求的 Servlet（AdminSortAction.java）程序

该程序负责处理文章分类管理页面提交的“adminSort”请求。以帮助完成文章分类信息的载入、添加及删除等操作。

```
……
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
  //设置接收信息的字符集
  request.setCharacterEncoding("UTF-8");
  //接收浏览器端提交的信息
       String action = request.getParameter("action");
       String name = request.getParameter("name");
       String id = request.getParameter("id");

      HttpSession session = request.getSession(true);
       Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);

       //设置输出信息的格式及字符集
  response.setContentType("text/xml; charset=UTF-8");
  response.setHeader("Cache-Control", "no-cache");
  //创建输出流对象
  PrintWriter out = response.getWriter();
  //依据验证结果输出不同的数据信息
  out.println("<response>");
```

```
        /*
         * 执行数据库操作并返回结果
         */
           DbOperate   db=new DbOperate();
        if("load".equals(action)){
           List sortList=db.getBlogSorts(blog.getId());
           for (int i=0;i<sortList.size();i++){
                   Sort sort = (Sort)sortList.get(i);
               out.println("<sort>");
                    out.println("<id>" + sort.getId() + "</id>");
                    out.println("<name>" + sort.getName() + "</name>");
               out.println("</sort>");
          }
        }
         else if ("add".equals(action)){
             Sort sort =new Sort();
             sort.setName(name);
             sort.setBlogid(blog.getId());
             db.save(sort);
             out.println("<id>" + sort.getId() + "</id>");
             out.println("<name>" + sort.getName() + "</name>");
         }
         else if ("delete".equals(action)){
             Sort sort = db.getSort(Integer.parseInt(id));
             db.delete(sort);
             out.println("<id>" + sort.getId() + "</id>");
         }
            out.println("</response>");
            out.close();
    }
……
```

4. 负责处理链接管理请求的 Servlet（AdminLinksAction.java）程序

该程序负责处理链接管理页面提交的“adminLinks”请求。以帮助完成链接信息的载入、添加及删除等操作。

```
……
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      //设置接收信息的字符集
      request.setCharacterEncoding("UTF-8");
      //接收浏览器端提交的信息
            String action = request.getParameter("action");
            String name = request.getParameter("name");
            String url = request.getParameter("url");
            String id = request.getParameter("id");

        //利用 hibernate 查询热点博客和最新文章
           DbOperate db=new DbOperate();
          HttpSession session = request.getSession(true);
```

```
        //设置输出信息的格式及字符集
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        //创建输出流对象
        PrintWriter out = response.getWriter();
        //依据验证结果输出不同的数据信息
        out.println("<response>");
          Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);

        if ("load".equals(action)){
             List linkList=db.getBlogLinks(blog.getId());
             for (int i=0;i<linkList.size();i++){
                    Links link = (Links)linkList.get(i);
              out.println("<link>");
                   out.println("<id>" + link.getId() + "</id>");
                   out.println("<name>" + link.getName() + "</name>");
                   out.println("<url>" + link.getUrl() + "</url>");
              out.println("</link>");
             }
        }
        else if ("add".equals(action)){
             Links link =new Links();
             link.setName(name);
             link.setUrl(url);
             link.setBlogid(blog.getId());
             db.save(link);
             out.println("<id>" + link.getId() + "</id>");
             out.println("<name>" + link.getName() + "</name>");
                    out.println("<url>" + link.getUrl() + "</url>");
        }
        else if ("delete".equals(action)){
             Links link = db.getLink(Integer.parseInt(id));
             db.delete(link);
             out.println("<id>" + link.getId() + "</id>");
        }
             out.println("</response>");
             out.close();
    }
```

……

5. 负责处理个人信息编辑修改请求的 Servlet（SaveSelfInfoAction.java）程序

该程序负责处理个人信息编辑修改页面提交的“saveSelf”请求。以帮助完成个人信息的编辑修改工作。

……

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
 //设置接收信息的字符集
 request.setCharacterEncoding("UTF-8");
```

```
        //接收浏览器端提交的信息
              String subject = request.getParameter("subject");
              String psw = request.getParameter("psw");
              String email = request.getParameter("email");

              HttpSession session = request.getSession(true);
              Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
              blog.setPassword(psw);
              blog.setSubject(subject);
              blog.setEmail(email);

            /*
             * 保存信息
             */
                DbOperate  db=new DbOperate();
                db.update(blog);
                session.setAttribute(Constants.MODIFY_OK_KEY,new Integer(1));
                response.sendRedirect("/BlogOnline/adminSelf.jsp");
    }
……
```

6. 负责处理上传图片请求的 Servlet（SaveLogoAction.java）程序

该程序负责处理个人 Logo 图片修改提交的“saveLogo”请求。以帮助完成个人图片的上传及修改工作。

```
public class SaveLogoAction extends HttpServlet {

    private ServletConfig config;

    public void init(ServletConfig config) throws ServletException {
           this.config = config;
    }
……
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        /*
         * 上传文件
         */
          String imageName = null;
          SmartUpload mySmartUpload = new SmartUpload();
          try {
                 // 初始化
                 mySmartUpload.initialize(config,request,response);
                 // 上载
                 mySmartUpload.upload();
                 // 保存上载文件到指定目录
                 mySmartUpload.save("\\images");
                 imageName = mySmartUpload.getFiles().getFile(0).getFileName();

          } catch (Exception e){
```

```
                System.out.println(e);
            }

        /*
         * 保存信息
         */
            HttpSession session = request.getSession(true);
            Blog blog = (Blog)session.getAttribute(Constants.LOGIN_USER_KEY);
            blog.setImage(imageName);

            DbOperate  db=new DbOperate();
            db.update(blog);
            session.setAttribute(Constants.MODIFY_OK_KEY,new Integer(1));
            response.sendRedirect("/BlogOnline/adminLogo.jsp");
    }
……
```

需要注意的是，为了实现文件的上传功能，在该程序中使用了上传组件 SmartUpload，在使用前，需要通过项目构建路径的设置，将对应的包导入进来，读者可在配套的光盘中获取到该组件。

8.3.7 技术难点详解

1. DOM 技术的应用

本例中多个页面均借助 Ajax 技术实现了页面的局部动态更新，解决的方式是借助 DOM 对 HTML 文档进行操作，以控制动态表格中行的增加或者减少，对应的核心代码如下：

```
function addOneArticle(id,name,time) {
            var row = document.createElement("tr");
            row.setAttribute("id", id);
            var cell = document.createElement("td");
            cell.appendChild(document.createTextNode(name));
            row.appendChild(cell);

            cell = document.createElement("td");
            cell.appendChild(document.createTextNode(time));
            row.appendChild(cell);

            var deleteButton = document.createElement("input");
            deleteButton.setAttribute("type", "button");
            deleteButton.setAttribute("value", "删除");
            deleteButton.onclick = function () { deleteArticle(id); };
            cell = document.createElement("td");
            cell.appendChild(deleteButton);
            row.appendChild(cell);

            document.getElementById("articleList").appendChild(row);
    }
```

2. 文件上传功能的实现

本模块需要实现图片文件上传到服务器的功能，在具体实现时，在对应表单的相关标记中要注意以下两点。

```
<form enctype="multipart/form-data" method="post" action="saveLogo">
<input type="file" name="image">
```

在具体获取文件信息时，借助了对应组件来达到目的，具体核心代码如下：

```
String imageName = null;
SmartUpload mySmartUpload = new SmartUpload();
try {
        // 初始化
        mySmartUpload.initialize(config,request,response);
        // 上载
        mySmartUpload.upload();
        // 保存上载文件到指定目录
        mySmartUpload.save("\\images");
        imageName = mySmartUpload.getFiles().getFile(0).getFileName();

} catch (Exception e){
        System.out.println(e);
}
```

有关此电子图书的说明

本人由于一些便利条件，可以帮您提供各种中文电子图书资料，且质量均为清晰的 PDF 图片格式，质量要高于网上大量传播的一些超星 PDG 的图书。方便阅读和携带。只要图书不是太新，文学、法律、计算机、人文、经济、医学、工业、学术等方面的图书，我都可以帮您找到电子版本。所以，当你想要看什么图书时，可以联系我。我的 QQ 是：85013855，大家可以在 QQ 上联系我。

此 PDF 文件为本人亲自制作，请各位爱书之人尊重个人劳动，敬请您不要修改此 PDF 文件。因为这些图书都是有版权的，请各位怜惜电子图书资源，不要随意传播，否则，这些资源更难以得到。