自己动手写Struts：构建基于MVC的Web开发框架

目录

前言

问题的提出

又一个客户来抱怨了，原来他们企业自己培养的开发人员，在修改了 Struts 的一些配置文件后，整个系统就崩溃掉了，再恢复成原来的配置，仍然不能正常运行，客户却又不知道问题出在哪里。

您是不是也遇到过类似的问题呢？按照书中的示例代码，将程序录入进去，正确的运行结果却没有出来，也没有相应的日志提示，想调试一下，但使用的却只是一个 jar；想看一下相关资料，中文的作者没有讲清楚，英文的却又看不懂。

唉，算了，用别人的东西就是麻烦，自己动手写吧，结果 JSP 中代码满天飞，数据库连接的代码比比皆是，小的项目还行，如果遇到大的项目，改了这个地方，又影响了另外的地方，到最后，再也不敢改现有的代码了，只能从头再来。

我想，现在很多开发人员都会遇到类似的问题：使用国外的项目，不能进行源代码的调试，教科书只是讲解如何使用，并没有讲解类似的调试细节；使用自己编写的框架，视图和逻辑代码纠缠在一起，牵一发而动全身。

《自己动手写 Struts》横空出世

现在，不用再担心上述问题了，本书完整地实现了一个基于 MVC 的 Web 框架，手把手地教读者自己动手来实现一个类似于 Struts 的 Web 框架。通过该框架，把原来开发人员针对一个系统的输入、输出、处理流程编程的方式改为按照模型层、视图层、控制层进行分解，从而使得整个系统的结构责任明确、接口清晰。该框架帮助开发人员加快了设计开发的过程，也使得调试变得轻松起来。

读者也不用担心该框架是否能完成大型项目的开发问题，经过笔者所负责的几个大型制造企业 ERP 项目的实践，该框架完全能满足企业应用的实际需求。

为了帮助众多初学者快速掌握并能设计一个基于 MVC 的 Web 框架，笔者精心编著了本书。本书依照读者的学习规律，首先介绍基本概念和基本操作，在读者掌握了这些基本概念和基本操作的基础上，再对内容进行深入的讲解，严格遵循由浅入深、循序渐进的原则。本书按照应该对 MVC 的知识进行掌握的先后顺序进行编排。

本书包括的内容

第 1 章是 Web 框架介绍。本章主要对 MVC 模式进行一个比较详细的讲解，从 MVC 的设计思想开始，通过对 MVC 的结构和处理过程、Model 规范的讲解，以及对 MVC 模式优劣的比较，介绍如何构建一个基于 MVC 的 Web 框架的大体思路。

第 2 章是快速准备 Web 框架的开发运行环境。本章的主要目的是建立 Web 框架开发运行的环境，然后通过使用 JSP 和 Servlet 实现的示例来回顾一下 JSP、Servlet 通常的开发过程，用来和后面采用 MVC 模式实现的 Web 框架进行比较，以使读者对此有更深入的理解。

第 3 章是快速掌握开发 Web 框架的基础知识。要进行 Web 框架的开发，与其相关的技术是必须要掌握的。在开始构建 Web 框架之前，先介绍一下在构建框架过程中会使用到的一些基础知识，比如：JSP、Servlet 和 JavaScript 等，这里不会对它们进行长篇大论地讲解，而是做一个简单的介绍，可以说是笔者在学习过程中的一些总结和归纳，并且是后面章节中会使用到的技术。

第 4 章是快速实现自己的 Web 框架。在本章中，首先给出所要实现框架的示意图，然后分别从视图层设计、控制层设计和模型层设计分别来进行讲解，最后使用一个简单的示例来验证所实现的框架。

第 5 章是将自己的 Web 框架与 Struts 进行对比。本章首先着重介绍 Struts 是如何实现控制器的，它的 Action 是如何运行的，以及在返回视图层的映射方式、表单的取值方式和多动作的处理等方面是如何实现的，目的是使读者能够清楚了解 Struts 在这些方面的处理方式；然后将 Struts 和本书中的 Web 框架进行对比，看它们的实现方式有哪些不同，并提出改进的方案。

第 6 章是逐步改善自己的 Web 框架。本章首先介绍要改善的要点，然后分别讲解要改善的设计细节，并对每次改善都通过相应的示例来验证。

第 7 章是 Web 框架的持久层封装。本章主要讲解持久层的设计，首先对 JDBC 进行简单讲解，然后概述一下事务处理的概念，最后实现一个持久层并通过示例验证它。

第 8 章是自己动手写数据库连接池。本章首先对数据库的基本工作原理进行讲解，接着实现一个数据库连接池，目的是让读者对数据库连接池的实现方式有一个深入的认识，然后讲解目前的一些开源数据库连接池的使用方法。

第 9 章是自己动手写过滤器。笔者使用 Servlet 来实现控制器的功能，把解析配置文件、转换页面中元素的值、调用对应的方法，以及返回指定页面都放在了 Servlet 中，使得功能划分不清，以后维护起来比较麻烦，而 Servlet 的过滤器提供了在转入 Servlet 之前可以提前进行处理的功能。本章将用过滤器来实现以前全部在 Servlet 中实现的功能。

第 10 章是将自己写的 Web 框架与 Hibernate 整合。本章首先对 Hibernate 进行介绍，并给出一个示例使读者快速了解 Hibernate，接着介绍 Hibernate 的配置方式和映射方法。本章主要讲述如何将 Web 框架与 Hibernate 进行整合。

第 11 章是在自己写的 Web 框架中使用 Log4j。本章主要讲解 Log4j 的使用方法，并将其应用在本书的 Web 框架中。

第 12 章是用 Web 框架实现内容管理系统。本章通过一个内容管理系统实例对本书中的 Web 框架进行一个整体的演示，从而使读者对本书所实现的基于 MVC 的 Web 框架有一个全面的掌握。

适合阅读本书的读者

本书具有通俗易懂、主次分明、指导性强、快速入门、提纲挈领的特点，力求以通俗的语言及丰富的实例来指导读者透彻掌握如何构建一个基于 MVC 的 Web 框架。本书适合如下读者阅读：

— Java Web 开发人员；
— JSP、Servlet 程序员；
— 系统构架师；
— J2EE 开发人员。

本书涉及的开源软件请到 www.broadview.com.cn 上下载

第 4 章 力学笃行：快速实现自己的 Web 框架

“纸上得来终觉浅，绝知此事要躬行”。如果读者只知道理论，而不进行实践，那么对如何实现一个自己地 Web 框架将不会有一个深入的了解。从本章开始，笔者将带领读者来一步一步地实现一个基于 MVC Model 2 规范的 Web 框架，如果读者对 MVC Model 2 规范还不太熟悉，请参看第 1 章的介绍，或者在网上查找相关的资料。

为了使读者对将要实现的 Web 框架有一个整体的认识，在本章中，首先会给出所要实现框架的示意图，然后分别从视图层设计、控制层设计和模型层设计 3 个方面来分别进行讲解，最后会使用一个简单的示例来验证所实现的框架。当然，在本章中所实现的框架是一个最基本的框架，在以后的章节中，将会逐渐对这个框架进行扩充。

4.1 使用 MVC Model 2 规范实现 Web 框架的示意图

在第 1 章中，笔者对 MVC Model 2 规范进行了简单的介绍，但对于开发一个基于 MVC Model 2 规范的 Web 框架来说，这点知识是远远不够的。为了避免抽象的讲解，这里给出一个使用 MVC Model 2 规范实现 Web 框架的示意图，以使读者对此有一个直观的了解。Model 2 规范实现的 Web 框架示意图，如图 4-1 所示。

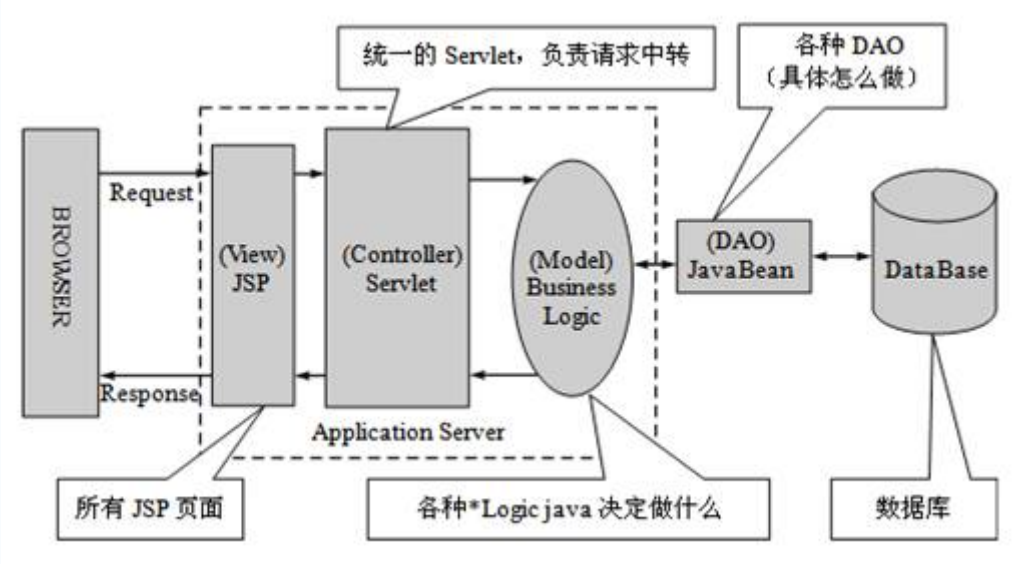


图 4-1 Model 2 规范实现的 Web 框架示意图

对图 4-1 进行简单地说明如下：

— 视图层采用 JSP 实现；
— 控制层采用 Servlet 实现，整个框架采用同一个 Servlet，以实现请求的中转；
— 模型层采用 Java 实现，主要决定用来做什么；
— 在模型层后添加了一个 DAO，目的是将决定做什么和具体怎么做分开。

整个 Web 框架大致的流程是：首先客户端发送请求，提交 JSP 页面给中转器（Servlet）；中转器根据客户的请求，选择相应的模型层，即 Logic，Logic 进行相应的逻辑处理；如果需要使用数据库，则通过 DAO 进行相应的数据库操作。

4.2 视图层设计

视图层用来显示用户所需要的数据，在本书的框架中，是由 JSP、JavaScript、JavaBean 和自定义标签组成的。虽然视图层和控制层在 MVC 的概念上是分离的，但其实它们之间还是需要有很大的联系的，因为控制层需要从视图层获取用户提交的数据，然后把数据进行转换，并传递给模型层；等模型层处理完数据后，控制层还需要把数据返回到视图层，以供用户查看。

4.2.1 使用 JSP

在视图层采用 JSP 技术，是展现用户数据的一种较好的方式。因为在 JSP 中，可以使用 JavaScript、JavaBean 和自定义标签等，这些技术结合在一起，就能获得一个灵活的数据组合方式。在使用 JSP 的过程中，有以下几点需要注意。

（1）为了防止乱码的发生，在本框架中，所有的 JSP 页面都采用 GBK 的编码方式，即在 JSP 页面的顶部添加如下代码：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
```

（2）为了便于统一管理，需要在所有的 JSP 页面的顶部都添加如下代码：

```
<%!
   public static final String appId = 页面名称;
%>
```

（3）为了防止提交时页面的跳转混乱，在 JSP 页面的底端都添加以下代码：

```
<script>
   window.name="<%=页面名称+"_"+session.getId()%>";
```

```
</script>
```

4.2.2 数据的提交和获取方式

在前面第 2 章介绍的用 Servlet 实现 HelloWorld 的示例中，可以看到，通过在 web.xml 中定义 Servlet 的请求方式，可以将所有的 JSP 页面的请求都转入 Servlet 进行处理，这样就很容易通过 request、response 实现 JSP 和 Servlet 之间的数据转换。在 Servlet 中通过 request 获取从 JSP 页面传来的数据，然后将模型层处理过的数据通过 response 传递到 JSP 页面，在 JSP 页面用 request 来获取从控制层传来的数据。下面分别介绍数据提交和获取的方式。

（1）在提交数据时，因为在控制层需要判断客户端提交的动作、要调用的模型层和要返回的页面，所以需要在 JSP 的表单中隐藏 3 个元素：action（动作）、logicName（要调用的模型层）和 forwardJsp（要返回的页面）。示例代码如下：

```
<form name="form1" action="/myApp/do" method="post">
    <input type="hidden" name="action" value="">
    <input type="hidden" name="logicName" value="">
    <input type="hidden" name="forwardJsp" value=" ">
</form>
```

这样，在提交数据时，就可以通过 submit 方法来提交数据。示例代码如下：

```
function submit() {
    form1.forwardJsp.value = "index";
    form1.logicName.value = "Logic";
    form1.target = "<%="index _"+session.getId()%>";
    form1.action.value = "insert";
    form1.submit();
}
```

代码说明：

— 客户端的数据要提交到 Logic 类里的 insert 方法，在 insert 方法处理完毕后，要返回到 index 页面，这个页面返回时仍放在 index 中。

— 通过这种隐藏的方法，还可以把一些不需要客户看到的数据传递到 Servlet。

（2）在获取数据时，因为在 Servlet 中是通过 response 来把数据传递到 JSP 页面中的，所以，在 JSP 页面通过 request 来获取时，如果要获取的数据比较多，则会比较烦琐。因此，要在 Servlet 中把所有要

传递到 JSP 页面中的数据存放在一个 Map 中，这样，在 JSP 页面只需要使用一次 request 把 Map 取出，然后剩余的数据就可以直接从 Map 中获取了。示例代码如下：

```
<%
    HashMap infoOut=(request.getAttribute("infoOut")==null)?new HashMap():
        (HashMap)request.getAttribute("infoOut");
    //定义消息
    String msg=infoOut.get("msg")==null ? "" :(String)infoOut.get("msg");
    String clerkId = null == request.getParameter("clerkId") ? "" :
        request.getParameter("clerkId");
    String clerkName = infoOut.get("clerkName") == null ? "" :
            (String) infoOut.get("clerkName");
%>
```

4.2.3 定义几个通用的 JavaScript 函数

有两个 JavaScript 方法可以通用：一个是表单提交的方法；另一个是打开窗口的方法，下面来分别进行介绍。

（1）从上面数据提交的方法 submit 里可以看出，表单提交时的 target 和 action 属性与 submit 方法其实是通用的，而经常改变的可能是 forwardJsp 和 logicName。所以，可以把通用的抽取出来，则上面提交数据的方法可以改为：

```
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
function insert() {
    form1.forwardJsp.value = "index";
    form1.logicName.value = "Logic";
    submit('<%=index +"_"+session.getId()%>','insert');
}
```

代码说明：

— 这样，所有的提交方法就都可以使用 submit 方法了。

（2）对于窗口的打开方法，为了在整个应用中统一风格，通常在系统中将其抽取出来，成为公用方法。一般来说，会对要打开的 url、窗口名称、窗口宽度和高度进行设定，而其他属性保持一致。示例代码如下：

```
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
```

```
    var screenHeight = screen.height;
    var w;
    <!--定义窗口属性-->
        w=window.open(url,name, "width="+width+",height="+height+",
            menubar= no,resizable=yes,toolbar=no,directories=no,
            location=no,scrollbars =yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
```

代码说明：

— 传入的参数有 4 个，分别是：窗口名称、要打开的 url、窗口宽度和高度。

— w.moveTo((screenWidth-width)/2, (screenHeight-height)/2)的意思是：使打开的窗口在屏幕的中间。

— w.focus()的意思是：使焦点在打开的窗口上。

4.2.4 一个较为完整的 JSP 示例

下面给出一个比较完整的 JSP 示例代码。

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
    javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "index"; %>
<%
    //获取从逻辑传来的值
HashMap infoOut = (request.getAttribute("infoOut") == null) ? new HashMap() :
        (HashMap)request.getAttribute("infoOut");
    //定义从逻辑传来的消息
    String msg = infoOut.get("msg") == null ? "" :
        (String) infoOut.get("msg");
    String clerkId = null == request.getParameter("clerkId") ? "" :
        request.getParameter("clerkId");
    //获取从逻辑传来的用户名
    String clerkName = infoOut.get("clerkName") == null ? "" :
        (String)infoOut.get("clerkName");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<title>示例代码</title>
<style type="text/css">
<!--定义 body 的 css 属性
body {
    background-color: #FFFFFD;
}
窗体的 css 属性
td {
    vertical-align:bottom;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
    color: #FFFFFF;
}
边框的 css 属性
.hborder {
    border-top: solid #7C7C7A 1px;
    border-bottom: solid #F1F1EE 1px;
}
-->
</style>
<script language=Javascript>
<!--提交动作-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
    <!--设定窗口的相关属性-->
    w=window.open(url,name, "width="+width+",height="+height+",menubar=
                no,resizable=yes,toolbar=no,directories=no,location=no,scrollbars
                =yes,status=yes,copyhistory=0");
    <!--移动到窗口中心-->
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
```

```
<!--退出窗口-->
function goExit() {
    window.close();
}
<!--提交新增动作-->
function insert() {
    form1.forwardJsp.value = "index";
    form1.logicName.value = "Logic";
    submit('<%="index _"+session.getId()%>','insert');
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myApp/do" method="post">
<!--表单示例-->
<table width="100%" border="0" cellpadding="0" cellspacing="0" style=
          "border-collapse: collapse">
  <tr bgcolor="#DFE4EB">
    <tdclass="hborder"colspan="10"style="filter: progid:DXImageTransform.
              Microsoft.gradient(GradientType=0,startColorstr=#F7F8FA,
              endColorstr=#C9D0DD)"> </td>
  </tr>
</table>
<!--隐藏页面的相关属性-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
```

```
<script language=Javascript>
    window.name = "<%="index _"+session.getId()%>";
</script>
</body>
</html>
```

4.3 控制层设计

前面讲过，控制层主要用来转发从视图层传来的数据和请求到相对应的模型层，因此，实现它最好的方式莫过于使用 Servlet 了。当从视图层获取请求后，首先通过对 web.xml 文件的配置，使其转入 Servlet，在 Servlet 中完成对页面中数据的封装和对相应模型的选择，然后再到相应的模型层进行数据处理；当在模型层数据处理完毕后，通过 RequestDispatcher 将处理后的数据返回相应的视图页面。

4.3.1 使用 Servlet

在第 3 章中，讲解了 get 和 post 的区别，因此在 Servlet 中，将使用 doPost()来处理相应的中转请求，如果开发人员使用 get 提交方式，则使用如下方式进行处理。示例代码如下：

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
    doPost(req, res);
}
//使用 post 提交方式
public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
    do_Dispatcher (req, res);
}
```

代码说明：

— 不论采用 get 还是 post 提交方式，都将执行 do_Dispatcher(req, res)方法。

— do_Dispatcher(req, res)是用来处理视图层发送来的请求的方法。

4.3.2 获取从视图层传来的值

直接使用 request 方式来获取从页面提交的数据，在要获取的数据比较多的情况下，会比较烦琐，而且直接将 request 传递给模型层不符合 Model 2 规范。所以，这里将对从页面传来的值进行封装，将其放在一个 Map 中，然后再传递给模型层，这样在模型层就可以直接使用 Map 中的值。示例代码如下：

```
private HashMap getRequestToMap(HttpServletRequest req) throws Exception {
    req.setCharacterEncoding("GBK");
        HashMap infoIn = new HashMap();
        for (Enumeration e = req.getParameterNames(); e.hasMoreElements ();)
                {//获取页面中所有元素名
            String strName = (String)e.nextElement();
            String[] values = (String[]) req.getParameterValues (strName);
                //根据名称获取对应的值
            if (values == null) {//假如没有值
                infoIn.put(strName, "");
            } else if (values.length == 1) {//假如只有一个值
                infoIn.put(strName, values[0]);
            } else {//假如有多个值
                infoIn.put(strName, values);
            }
        }
    return infoIn;
}
```

代码说明：

— req.setCharacterEncoding("GBK")，这里首先将从视图层传来的数据设定编码为 GBK。

— HashMap infoIn = new HashMap()，定义一个 HashMap，用来存放从 request 中获取的数据。

— req.getParameterNames()，用来获取从页面中传来的所有元素。

— req.getParameterValues()，用来根据元素名称来获取元素对应的值，并将元素名称和值的对应关系存入 HashMap 中。如果元素的值为空，则在 HashMap 中将元素名称对应的值置为空；如果只有一个值，则将该值存入；如果有多个值，则存入数组。

4.3.3 处理请求到模型层

一个视图对应一个模型，也可能一个视图对应多个模型，但只有一个控制器，所以，为了实现一个控制器可以转发到多个模型中去，就需要使用接口，让所有模型都实现这个接口，然后在控制器里，仅仅是面对接口编程即可。这里定义一个接口 Action.java，Action.java 的示例代码如下：

```
//******* Action.java**************
```

```
package com.gd.;
import java.util.*;
public interface Action{
    public HashMap doAction(HashMap infoIn);
}
```

在控制器中只针对这个接口处理即可。示例代码如下：

```
Actionaction = (Action) Class.forName(getActionName(systemName,
            logicName)).newInstance();
HashMap infoOut = action.doAction(infoIn);
```

代码说明：

— getActionName()方法是获取实现接口 Action 的类的名称和所在的包。示例代码如下：

```
private String getActionName(String systemName ,String actionName)
            throws IOException, Exception {
return "com. " + systemName + ".action." + actionName;
}
```

4.3.4 返回视图层

这里使用 RequestDispatcher 返回视图层。示例代码如下：

```
req.setAttribute("infoOut", infoOut);
RequestDispatcher rd = req.getRequestDispatcher("/"+ systemName +
        "/jsp/" + forwardJsp+ ".jsp");
rd.forward(req, res);
```

代码说明：

— 这里表示 JSP 文件放在项目中系统名下的 jsp 文件夹下。

4.3.5 定义 web.xml 文件

定义 web.xml 文件，是将所有以 do 为后缀的请求，都转入控制器进行分派。web.xml 文件的示例代码如下：

```
//******* web.xml**************
<?xml version="1.0" encoding="GBK"?>
```

```
<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">
    <display-name>web.xml 的示例</display-name>
    <description>
    控制器示例
    </description>
<!--定义控制器-->
<servlet>
        <servlet-name>gdServlet</servlet-name>
        <servlet-class>com.gd.action.GdServlet</servlet-class>
</servlet>
<!--拦截所有以 do 为后缀的请求-->
    <servlet-mapping>
         <servlet-name>gdServlet</servlet-name>
         <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

代码说明：

- com.gd.web.GdServlet，表示控制器为 com.gd.web.GdServlet。
- *.do，表示拦截所有以 do 为后缀的请求。

4.3.6 一个完整的控制层示例 GdServlet.java

一个完整的控制层示例 GdServlet.java 的代码如下：

```
//******* GdServlet.java**************
package com.gd.action;
import java.io.IOException;
```

```
import java.util.Enumeration;
import java.util.HashMap;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class GdServlet extends HttpServlet{
    /**
     * 进行初始化工作
     */
    public void init() throws ServletException {
    }
    /**
     * 页面中使用 get 方式提交，执行 doPost 方法
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException {
       doPost(req, res);
    }
    /**
     * 页面中使用 post 方式提交，执行 do_Dispatcher 方法
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException {
       do_Dispatcher (req, res);
    }
    private void do_Dispatcher (HttpServletRequest req,
                    HttpServlet Response res) {
```

```
    try {
      //获取页面设定元素值
      String forwardJsp=(String) req.getParameter("forwardJsp");
      String logicName = (String) req.getParameter("logicName");
      //定义系统名变量
      String systemName = "";
      //获取访问路径
      String ss = req.getServletPath();
      systemName = ss.split("/")[1];
      //进行页面值的转换
      HashMap infoIn = getRequestToMap(req);
      //调用相应的业务逻辑
      Action action  = (Action) Class.forName(getActionName
                    (systemName,logicName)).newInstance();
        HashMap infoOut = action.doAction(infoIn);
      //将处理后的结果返回页面
      req.setAttribute("infoOut", infoOut);
      RequestDispatcher rd = req.getRequestDispatcher("/"+
                    systemName +"/jsp/" + forwardJsp+ ".jsp");
      rd.forward(req, res);
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
    }
}
/**
 * 将页面中所有元素放在 HashMap 中
 * @param req
 * @return HashMap
```

```
     * @throws Exception
     */
    private HashMap getRequestToMap(HttpServletRequest req) throws
                    Exception {
        req.setCharacterEncoding("GBK");
        HashMap infoIn = new HashMap();
                for (Enumeration e = req.getParameterNames(); e.hasMoreElements();)
                        {//获取页面中所有元素名
            String strName = (String)e.nextElement();
            String[] values = (String[]) req.getParameterValues (strName);
                        //根据名称获取对应的值
            if (values == null) {//假如没有值
                infoIn.put(strName, "");
            } else if (values.length == 1) {//假如只有一个值
                infoIn.put(strName, values[0]);
            } else {//假如有多个值
                infoIn.put(strName, values);
            }
        }
        return infoIn;
    }
    //获取指定的逻辑名
    private String getActionName(String systemName ,String actionName)
                    throws IOException, Exception {
          return "com." + systemName + ".action." + actionName;
    }
    public void destroy() {
    }
}
```

4.4 模型层设计

在上面的示例中，并没有具体地给出从视图层传来的请求如何经过控制器到达模型层的，只是给出了一个接口，然后让所有模型层类都实现这个接口，最后通过这个接口中的方法 doAction 来转入模型层进行处理。本节就来讲述如何在模型层通过 doAction 方法来实现视图层的请求处理。

4.4.1 实现一个公用的接口 Action.java

这里再把前面定义的接口 Action. java 列出，Action. java 的示例代码如下：

```
//******* Action.java**************
package com.gd.;
import java.util.*;
public interface Action{
    public HashMap doAction(HashMap infoIn);
}
```

代码说明：

— 这个接口的作用就是通过 doAction 方法来实现动作的转换，因此，所有的模型层类都要实现这个接口。

4.4.2 所有的模型层类都实现这个接口

假定有一个模型层类为 WebExamAction.java，主要用来负责在线考试系统的业务处理，则这个类要实现 Action 接口。示例代码如下：

```
//******* WebExamAction.java**************
package com.gc.action;
import java.util.HashMap;
public class WebExamAction implements Action{
    //根据页面的请求，进行动作的转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" :
                        (String)infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))          infoOut = this.doInit (infoIn);
        else if (action.equals("insert")) infoOut = this.doInsert (infoIn);
        return infoOut;
    }
```

```
/**该方法设置用户登录时页面的初始信息
* @param infoIn
* @return HashMap
*/
private HashMap doInit(HashMap infoIn) {
   HashMap infoOut = infoIn;
   int clerkId = (infoIn.get("clerkId") == null || "".equals
               (infoIn.get("clerkId"))) ? -1 : Integer.parseInt((String)
               infoIn.get ("clerkId"));
   try {
      //取得考生姓名
      Users user = new Users();
      String clerkName = user.getName(clerkId);
      infoOut.put("clerkName", clerkName);
   } catch(Exception e) {
      e.printStackTrace();
   } finally {
      return infoOut;
   }
}


/**该方法用来进行新增
* @param infoIn
* @return HashMap
*/
private HashMap doInsert(HashMap infoIn) {
   HashMap infoOut = infoIn;
   int clerkId = (infoIn.get("clerkId") == null || "".equals
            (infoIn.get("clerkId"))) ? -1 : Integer.parseInt((String)
            infoIn.get ("clerkId"));
   try {
      //取得考生姓名
      Users user = new Users();
      String clerkName = user.getName(clerkId);
      infoOut.put("clerkName", clerkName);
```

```
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

代码说明：

— 这里，在 doAction 中根据从页面传来的 action 进行动作请求的转换。

— 通过一个名为 infoIn 的 HashMap，来负责传入页面中元素的值。

— 通过一个名为 infoOut 的 HashMap，来负责将处理后的数据传出。

4.4.3 一个完整的模型层示例

从上面的示例可以看出，如果模型层类都实现接口 Action，实作 doAction 方法，即可实现动作请求的转换。一个完整的模型层示例××Action.java 的代码如下：

```
//******* ××Action.java**************
package com.gc.action;
import java.util.HashMap;
public class ××Action implements Action{
    //根据页面的请求，进行动作的转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" : (String)
                        infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))          infoOut = this.doInit (infoIn);
        else if (action.equals("insert")) infoOut = this.doInsert (infoIn);
        else if (action.equals("update")) infoOut = this.do Update(infoIn);
        else if (action.equals("delete")) infoOut = this.do Delete(infoIn);
        else if (action.equals("query")) infoOut = this. doQuery(infoIn);
        return infoOut;
    }

    /**该方法设置初始信息
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
```

```
        try {
            ....
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }

    /**该方法用来进行新增
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInsert(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            ....
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
/**该方法用来进行修改
    * @param infoIn
    * @return HashMap
    */
    private HashMap doUpdate(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            ....
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }

    }
/**该方法用来进行删除

    * @param infoIn

    * @return HashMap

    */

    private HashMap doDelete(HashMap infoIn) {
```

```
        HashMap infoOut = infoIn;
        try {
            ....
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
/**该方法用来进行查询
    * @param infoIn
    * @return HashMap
    */
    private HashMap doQuery(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            ....
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

经过上面的设计，一个初步的 Web 框架就快速地编写完成了。下面通过一个示例来验证这个快速实现的 Web 框架是否可用。

4.5 通过实现 HelloWorld 示例来验证框架

下面通过实现 HelloWorld 示例来验证这个框架。实现思路是：首先编写实现输出的页面 index.jsp，然后编写业务逻辑 HelloWorldAction.java，最后配置 web.xml 文件，并启动 Tomcat 运行示例。

4.5.1 编写实现输出的页面 index.jsp

在 myApp 下新建一个 gc 文件夹，在 gc 文件夹下再建一个 jsp 文件夹，在 jsp 文件夹中新建一个文件 index.jsp，并编写如下代码。

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
      javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "index"; %>
<%
   //获取从逻辑传来的值
   HashMapinfoOut=(request.getAttribute("infoOut")==null)? new HashMap() :
                (HashMap)request.getAttribute("infoOut");
   //获取从逻辑传来的消息
   String msg = infoOut.get("msg") == null ? "" : (String)infoOut. get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现 HelloWorld 输出</title>
<style type="text/css">
<!--
<!--设定 body 的 css 属性-->
body {
   background-color: #FFFFFD;
   font-family: Verdana, "宋体";
   font-size: 12px;
   font-style: normal;
```

```
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",

                menubar=no,resizable=yes,toolbar=no,directories=no,location=

                no,scrollbars=yes,status=yes,copyhistory=0");

    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);

    w.focus();

}

<!--退出页面-->

function goExit() {

    window.close();

}

<!--表单的提交动作-->

function insert() {

    form1.forwardJsp.value = "index";

    form1.logicName.value = "HelloWorldAction";

    submit('<%="index_"+session.getId()%>','insert');

}

</script>

</head>

<body leftmargin="0" topmargin="0">
```

```
<form name="form1" action="/myApp/do" method="post">
<H1><%=msg%><H1>
<!--页面中隐藏的一些元素-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--定义页面的名称-->
<script language=Javascript>
    window.name = "<%="index_"+session.getId()%>";
</script>
</body>
</html>
```

4.5.2 编写业务逻辑 HelloWorldAction.java

在 com.gc.action 下，新建一个类 HelloWorldAction.java，实现 Action 接口。HelloWorldAction.java 的示例代码如下：

```
//******* HelloWorldAction.java**************
package com.gc.action;
import java.util.HashMap;
import com.gd.action.Action;
public class HelloWorldAction implements Action {
/**该方法用来实现分动作处理
    * @param infoIn
    * @return HashMap
    */
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" :
                        (String) infoIn.get("action");
```

```
        HashMap infoOut = new HashMap();
        if (action.equals(""))            infoOut = this.doInit (infoIn);
        else if (action.equals("show")) infoOut = this.doShow (infoIn);
        return infoOut;
    }
    /**该方法用来实现没有传入动作时要处理的内容
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("msg", "HelloWorld");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
    /**该方法用来实现输出 HelloWorld
    * @param infoIn
    * @return HashMap
    */
    private HashMap doShow(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("msg", "HelloWorld");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

4.5.3 配置 web.xml 文件

将 web.xml 文件放在 myApp 的 WEB-INF 文件夹下。web.xml 文件的示例代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<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">
    <display-name>web.xml 的示例</display-name>
    <description>
    使用 Servlet 实现 HelloWorld 的例子
    </description>
<!--过滤器-->
<servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>com.gd.action.GdServlet</servlet-class>
</servlet>
<!--拦截/HelloWorld 的请求-->
    <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

4.5.4 运行并验证示例

重新启动 Tomcat，然后在浏览器地址栏中输入 http://localhost:8080/myApp/ gc/index.do?logicName=HelloWorldAction&forwardJsp=index,即可看到页面中输出了 HelloWorld，如图 4-2 所示。

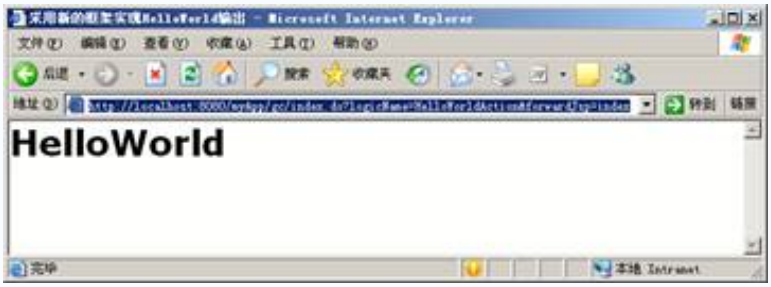


图 4-2　采用 Web 框架页面中输出了 HelloWorld

4.6 通过实现用户登录示例来验证框架

如果仅仅通过实现 HelloWorld 还不能体现新框架是否可行的话，这里笔者再通过一个用户登录示例来验证新的框架。实现思路如下：

步骤1 编写用户登录页面 login.jsp。

步骤2 用户提交登录信息后，如果登录成功，则转向登录成功的页面success.jsp；如果不成功，则仍然返回原来的登录页面login.jsp。

步骤3 编写业务逻辑 LoginAction.java。

步骤4 配置 web.xml 文件并启动 Tomcat 验证示例。

4.6.1 编写实现登录的页面 login.jsp

在 myApp 下 gc 文件夹的 jsp 文件夹中新建一个文件 login.jsp，用来实现用户登录，并编写如下代码。

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
   javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "login"; %>
<%
   //获取逻辑传来的值
   HashMap infoOut = (request.getAttribute("infoOut") == null) ? newHashMap
           ():(HashMap)request.getAttribute("infoOut");
   String msg = infoOut.get("msg") == null ? "" : (String)infoOut. get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户登录验证</title>
<style type="text/css">
```

```
<!--
<!--设定 body 的 css 属性-->
body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",menubar=
                no,resizable=yes,toolbar=no,directories=no,location=no,
                scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->

function goExit() {

    window.close();

}

<!--表单的登录动作-->

function check() {

    form1.forwardJsp.value = "login";

    form1.logicName.value = "LoginAction";

    submit('<%="login_"+session.getId()%>','login');

}

</script>
```

```
</head>
<body leftmargin="0" topmargin="0">
<!--设定页面布局-->
<form name="form1" action="/myApp/gc/login.do" method="post">
<H3><font color='red'><%=msg%></font><H3>
用户名：<input type="text" name="username" value=""><br>
密码：  <input type="text" name="password" value=""><br>
<input type="button" name="button" value="提交" onClick="return check()">
<input type="reset" name="button" value="重置">
<!--表单中隐藏的元素-->
   <input type="hidden" name="action" value="">
   <input type="hidden" name="forwardJsp" value="">
   <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
   window.name = "<%="login_"+session.getId()%>";
</script>
</body>
</html>
```

4.6.2 编写登录成功的页面 success.jsp

在\myApp\gc\jsp 文件夹中，新建一个文件 success.jsp，用来实现用户登录验证成功后显示的页面，并编写如下代码。

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
          "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
    javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "success"; %>
<%
   //获取从逻辑传来的值
```

```
    HashMap infoOut = (request.getAttribute("infoOut") == null) ? new HashMap
            () : (HashMap)request.getAttribute("infoOut");
    String msg = infoOut.get("msg") == null ? "" : (String)infoOut. get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户登录验证</title>
<style type="text/css">
<!--
<!--设定 body 的 css 属性-->
body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name,"width="+width+",height="+height+",menubar=no,
                resizable=yes,toolbar=no,directories=no,location=no,
                scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的登录动作-->
function check() {
    form1.forwardJsp.value = "login";
```

```
    form1.logicName.value = "LoginAction";
    submit('<%="login_"+session.getId()%>','login');
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myApp/do" method="post">
<H3><font color='red'><%=msg%></font><H3>
    <!--表单中隐藏的元素-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
    window.name = "<%="login_"+session.getId()%>";
</script>
</body>
</html>
```

4.6.3 编写业务逻辑 LoginAction.java

在 com.gc.action 下，新建一个类 LoginAction.java，实现 Action 接口。LoginAction.java 的示例代码如下：

```
//******* LoginAction.java**************
package com.gc.action;
import java.util.HashMap;
import com.gd.action.Action;
public class LoginAction implements Action {
    //根据页面的请求，进行动作的转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" :
                    (String) infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals("")) infoOut = this.doInit (infoIn);
        else if (action.equals("login")) infoOut = this.doLogin (infoIn);
```

```
        return infoOut;
    }
    /**该方法用来实现没有传入动作时要处理的内容
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("msg", "请输入用户名和密码");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }

    /**该方法用来实现对用户填写的登录信息进行业务逻辑处理
    * @param infoIn
    * @return HashMap
    */
    private HashMap doLogin(HashMap infoIn) {
        HashMap infoOut = infoIn;
        String username = (infoIn.get("username") == null) ? "" :
                    (String)infoIn.get("username");
        String password = (infoIn.get("password") == null) ? "" :
                    (String)infoIn.get("password");
        try {
            if ("gd".equals(username) && "123456".equals(password)){
                infoOut.put("msg", "登录成功");
            }else if ("gd".equals(username) && !"123456".equals(password)){
                infoOut.put("msg", "密码错误");
            }else if (!"gd".equals(username) && "123456".equals(password)){
                infoOut.put("msg", "用户名错误");
            }else if (!"gd".equals(username) && !"123456".equals(password)){
                infoOut.put("msg", "用户名和密码都输入错误");
            }else if ("".equals(username) && "".equals(password)){
                infoOut.put("msg", "请输入用户名和密码");
            }
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

4.6.4 配置 web.xml 文件

将 web.xml 文件放在 myApp 的 WEB-INF 文件夹下。web.xml 文件的示例代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<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">
    <display-name>web.xml 的示例</display-name>
    <description>
    使用 Servlet 实现登录的例子
    </description>
<!--过滤器-->
<servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>com.gd.action.GdServlet</servlet-class>
</servlet>
<!--拦截.do 的请求-->
    <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

4.6.5 运行并验证示例

步骤 1 重新启动Tomcat，然后在浏览器地址栏中输入http://localhost:8080/myApp/ gc/login.do?forwardJap=Login& logicName=LoginAction，即可看到用户登录页面，如图 4-3 所示。

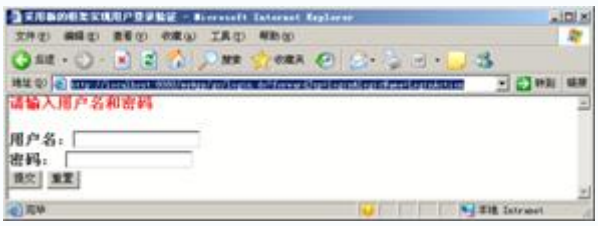


图 4-3 用户登录页面

步骤 2 也可以在地址栏中输入http://localhost:8080/myApp/gc/jsp/login.jsp进入该页面，但因为没有进入Servlet，所以提示消息不会出现，如图 4-4 所示。

图 4-4　没有提示消息的用户登录页面

为了安全考虑，一般情况下，不允许客户端直接访问 JSP 页面，解决方法就是把 JSP 页面放在 WEB-INF 文件夹下。

步骤 3 在页面中输入用户名“gf”，密码“123456”，然后单击“提交”按钮，即可看到页面报出“用户名错误”的信息提示，如图 4-5 所示。

图 4-5　“用户名错误”的信息提示

步骤 4 在页面中输入用户名“gd”，密码“12345”，然后单击“提交”按钮，即可看到页面报出“密码错误”的信息提示，如图 4-6 所示。

图 4-6 “密码错误”的信息提示

步骤 5 在页面中输入用户名“gd”，密码“123456”，然后单击“提交”按钮，即可看到页面报出“登录成功”的信息提示，如图 4-7 所示。

图 4-7　“登录成功”的信息提示

步骤 6 这里，笔者的本意是想让登录成功后，返回到登录成功的页面，而不是仍然返回到登录页面，这时候，就需要修改一下LoginAction.java。

4.6.6 修改 LoginAction.java 自定义返回的页面

根据前面对控制器的修改，在 LoginAction.java 里可以增加自定义返回页面的功能。修改后的 LoginAction.java 示例代码如下：

```
//******* LoginAction.java**************
package com.gc.action;
import java.util.HashMap;
import com.gd.action.Action;
public class LoginAction implements Action {
    //根据页面的请求，进行动作的转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" : (String)
                        infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))          infoOut = this.doInit(infoIn);
        else if (action.equals("login")) infoOut = this.doLogin(infoIn);
        return infoOut;
    }
    /**该方法用来实现没有传入动作时要处理的内容
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("msg", "请输入用户名和密码");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }

    /**该方法用来实现处理登录的业务逻辑
    * @param infoIn
    * @return HashMap
    */
    private HashMap doLogin(HashMap infoIn) {
        HashMap infoOut = infoIn;
        String username = (infoIn.get("username") == null) ? "" :
                        (String)infoIn.get("username");
        String password = (infoIn.get("password") == null) ? "" :
```

```
                (String)infoIn.get("password");
        try {
            if ("gd".equals(username) && "123456".equals(password)){
                infoOut.put("forwardJsp", "success");
                infoOut.put("msg", "登录成功");
            } else if ("gd".equals(username) && !"123456".equals(password)){
                infoOut.put("msg", "密码错误");
            } else if (!"gd".equals(username) && "123456".equals(password)){
                infoOut.put("msg", "用户名错误");
            } else if (!"gd".equals(username) && !"123456".equals(password)){
                infoOut.put("msg", "用户名和密码都输入错误");
            } else if ("".equals(username) && "".equals(password)){
                infoOut.put("msg", "请输入用户名和密码");
            }
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

4.6.7 重新验证示例

在页面中输入用户名“gd”，密码“123456”，然后单击“提交”按钮，即可看到自动返回到登录成功的页面，如图 4-8 所示。

图 4-8 自动返回到登录成功的页面

4.7 让新的 Web 框架支持 sendRedirect

在前面的示例中，每次演示的时候，都是在浏览器的地址栏中输入一长串的地址，如果只是简单地输入 http://localhost:8080/myApp/gc/login.do 会如何呢？下面试试看。

4.7.1 为什么要支持 sendRedirect

在浏览器的地址栏中直接输入 http://localhost:8080/myApp/gc/login.do，然后按回车键，即可看到提示找不到 JSP 的页面，如图 4-9 所示。

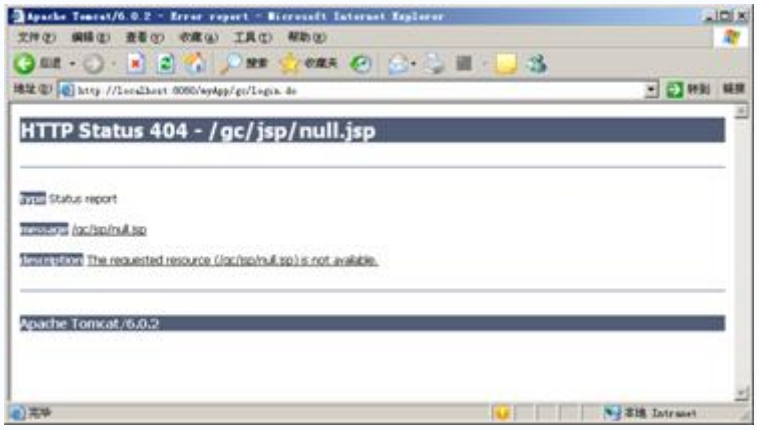


图 4-9　提示找不到 JSP 的页面

这是因为在控制器中，必须根据 forwardJsp 和 logicName 才能找到返回的页面和对应的逻辑；如果没有 forwardJsp 和 logicName，框架就不知道该如何处理了。因为本书中的 Web 框架使用的是 gd 文件夹，所以如果有应用使用别的文件夹，比如上例中的 gc 文件夹，这个时候根据前面对 forward 和 sendRedirect 的介绍，就需要使用 sendRedirect，使其转向 gd 文件夹下面默认的一个显示画面 welcome.jsp。

4.7.2　修改控制器中返回视图层的设计

原来返回视图层的代码如下：

```
req.setAttribute("infoOut", infoOut);
        RequestDispatcher rd = req.getRequestDispatcher("/"+ systemName +"/jsp/" + forwardJsp
            + ".jsp");
rd.forward(req, res);
```

这里将它抽取出来，单独写一个方法 forwardOrSendRedirect。forwardOrSendRedirect 的示例代码如下：

```
        private void forward(HttpServletRequest req, HttpServletResponse res) throws Exception {
      HashMap infoOut = (req.getAttribute("infoOut") == null) ? newHashMap
          (): (HashMap)req.getAttribute("infoOut");
      String forwardJsp = (String)infoOut.get("forwardJsp");
      String sendRedirectJsp = (String)infoOut.get("sendRedirectJsp");
      //判断是否设定使用 sendRedirect
```

```
        if (null != sendRedirectJsp && !"".equals(sendRedirectJsp)) {
            res.sendRedirect(sendRedirectJsp);
        } else {//不使用 sendRedirect
            RequestDispatcher rd = req.getRequestDispatcher("/"+
                        (String)infoOut.get("systemName") +"/jsp/" + forwardJsp+
                        ".jsp");
            rd.forward(req, res);
        }
}
```

另外，需要在 do_Dispatcher 方法里，将 systemName 存放在 infoOut 中。do_Dispatcher 方法的示例代码如下：

```
private void do_Dispatcher (HttpServletRequest req, HttpServletResponseres) {
        try {
            //获取页面设定元素值
            String forwardJsp = (String) req.getParameter("forwardJsp");
            String logicName = (String) req.getParameter("logicName");
            //定义系统名变量
            String systemName = "";
            //获取访问路径
            String ss = req.getServletPath();
            systemName = ss.split("/")[1];
            //进行页面值的转换
            HashMap infoIn = getRequestToMap(req);
            //调用相应的业务逻辑
            Action action = (Action) Class.forName(getActionName
                        (systemName, logicName)).newInstance();
              HashMap infoOut = action.doAction(infoIn);
            //将 systemName 存放在 infoOut 里
            infoOut.put("systemName", systemName);
              //将处理后的结果返回页面
            req.setAttribute("infoOut", infoOut);
            //进行页面返回
            forward(req, res);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
}
```

4.7.3 增加实现接口 Action 的类 GdAction.java

当从客户端提交的页面元素中没有 logicName 时，还需要增加一个默认的处理逻辑 GdAction.java。GdAction.java 的示例代码如下：

```
//******* GdAction.java**************
package com.gd.action;
import java.util.HashMap;
public class GdAction implements Action{
    //进行动作处理
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" : (String)
                        infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))        infoOut = this.doInit(infoIn);
        return infoOut;
    }
    /**该方法设置用户登录时页面的初始信息
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("sendRedirectJsp", "../jsp/welcome.jsp");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

4.7.4 设计默认的欢迎页面 welcome.jsp

默认的欢迎页面，放在 myApp 文件夹下 gd 中的 jsp 文件夹下。welcome.jsp 的示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java"
        import="java.sql.*" errorPage="" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>欢迎您使用 gf 的 Web 框架</title>
</head>
```

```
<body center>
<H1><font color='red'>
欢迎您使用 gf 的 Web 框架
</font>
</H1>
</body>
</html>
```

4.7.5 验证是否支持 sendRedirect

在浏览器的地址栏中输入 http://localhost:8080/myApp/gc/login.do，然后按回车键，即可看到默认的欢迎页面，如图 4-10 所示。

图 4-10 默认的欢迎页面

4.8 使用 MVC Model 2 规范实现 Web 框架的完整代码

经过上面的验证和修改，下面给出目前完整的基于 MVC Model 2 规范的 Web 框架的代码。

4.8.1 视图层代码

位于 gd 文件夹下的 jsp 文件夹中的 welcome.jsp 的代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>欢迎您使用 gf 的 Web 框架</title>
```

```
</head>
<body center>
<H1><font color='red'>
欢迎您使用 gf 的 Web 框架
</font>
</H1>
</body>
</html>
```

采用这个 Web 框架所使用的页面 login.jsp 的示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
          "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
       javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "login"; %>
<%
   HashMap infoOut=(request.getAttribute("infoOut")==null)?new HashMap() :
               (HashMap)request.getAttribute("infoOut");
   String msg = infoOut.get("msg") == null ? "" : (String) infoOut.get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户登录验证</title>
<style type="text/css">
<!—
<!--设定 body 的 css 属性-->
body {
```

```
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",
                menubar=no,resizable=yes,toolbar=no,directories=no,location=
                no,scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的登录动作-->
function check() {
```

```
    form1.forwardJsp.value = "login";
    form1.logicName.value = "LoginAction";
    submit('<%="login_"+session.getId()%>','login');
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myApp/gc/login.do" method="post">
<H3><font color='red'><%=msg%></font><H3>
用户名：<input type="text" name="username" value=""><br>
密码：  <input type="text" name="password" value=""><br>
<input type="button" name="button" value="提交" onClick="return check()">
<input type="reset" name="button" value="重置">
      <!--表单中隐藏的元素-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
    window.name = "<%="login_"+session.getId()%>";
</script>
</body>
</html>
```

4.8.2 控制器代码

位于包 com.gd.action 下的控制器 GdServlet.java 的示例代码如下：

```
//******* GdServlet.java**************
package com.gd.action;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class GdServlet extends HttpServlet{
    /**
     * 进行初始化工作
     */
    public void init() throws ServletException {
    }
    /**
     * 页面中使用 get 方式提交，执行 doPost 方法
```

```
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
        doPost(req, res);
    }
    /**
     * 页面中使用 post 方式提交，执行 do_Dispatcher 方法
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
        do_Dispatcher (req, res);
    }
    private void do_Dispatcher (HttpServletRequest req,
                HttpServletResponse res) {
        try {
            //获取页面设定元素值
            String forwardJsp = (String) req.getParameter("forwardJsp");
            String logicName = (String) req.getParameter("logicName");
            //定义系统名变量
            String systemName = "";
            //获取访问路径
            String ss = req.getServletPath();
            systemName = ss.split("/")[1];
            //进行页面值的转换
            HashMap infoIn = getRequestToMap(req);
            //调用相应的业务逻辑
            Action action = (Action) Class.forName(getActionName
                        (systemName, logicName)).newInstance();
            HashMap infoOut = action.doAction(infoIn);
            //将处理后的结果返回页面
            infoOut.put("systemName", systemName);
            req.setAttribute("infoOut", infoOut);
            //进行返回页面处理
            forward(req, res);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
    }
/**
     * 将页面中所有元素放在 HashMap 中
     * @param req
     * @return HashMap
     * @throws Exception
```

```
 */
private HashMap getRequestToMap(HttpServletRequest req)throwsException {
   req.setCharacterEncoding("GBK");
   HashMap infoIn = new HashMap();
   for (Enumeration e = req.getParameterNames(); e.hasMore Elements();)
      {//获取页面中所有元素名
      String strName = (String)e.nextElement();
      String[] values = (String[]) req.getParameterValues(str Name);
           //根据名称获取对应的值
      if (values == null) {//假如没有值
          infoIn.put(strName, "");
      } else if (values.length == 1) {//假如只有一个值
          infoIn.put(strName, values[0]);
      } else {//假如有多个值
          infoIn.put(strName, values);
      }
   }
   return infoIn;
}
//获取业务逻辑
private String getActionName(String systemName ,String actionName)
      throws IOException, Exception {
    if (actionName == null) {//如果为空，则默认
    return "com.gd.action.GdAction";
    } else {
    return "com." + systemName + ".action." + actionName;
    }
}
//进行返回页面处理
private void forward(HttpServletRequest req, HttpServletResponse
      res) throws Exception {
   HashMap infoOut = (req.getAttribute("infoOut") == null) ? new
         HashMap() : (HashMap)req.getAttribute("infoOut");
   String forwardJsp = (String)infoOut.get("forwardJsp");
   String sendRedirectJsp = (String)infoOut.get("sendRedirectJsp");
   //判断是否使用 sendRedirect
   if (null != sendRedirectJsp && !"".equals(sendRedirectJsp)) {
      res.sendRedirect(sendRedirectJsp);
   } else {//不使用
      RequestDispatcher rd = req.getRequestDispatcher("/"+
           (String)infoOut.get("systemName") +"/jsp/" + forwardJsp+
           ".jsp");
      rd.forward(req, res);

   }
```

```
    }
        /**
     * 进行销毁处理工作
     */
    public void destroy() {
    }
}
```

4.8.3 模型层代码

位于包 com.gd.action 下的模型层接口 Action.java 的示例代码如下：

```
//******* Action.java**************
package com.gd.action;
import java.util.HashMap;
public interface Action {
    public HashMap doAction(HashMap infoIn);
}
```

默认的处理逻辑 GdAction.java 的示例代码如下：

```
//******* GdAction.java**************
package com.gd.action;
import java.util.HashMap;
public class GdAction implements Action{
    //进行动作转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" : (String)
                infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))          infoOut = this.doInit(infoIn);
        return infoOut;
    }
```

```
    /**该方法设置用户登录时页面的初始信息
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("sendRedirectJsp", "../jsp/welcome.jsp");
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            return infoOut;
        }
    }
}
```

4.8.4 将自己的 Web 框架打包成 jar

前面的示例都是直接使用 Web 框架的源代码，如果读者需要在自己的项目中使用，就需要将其打包成 jar，就像使用 struts.jar 一样。具体打包成 jar 的方法如下。

步骤 1 单击Eclipse中的“File”菜单，弹出“File”菜单项，如图 4-11 所示。

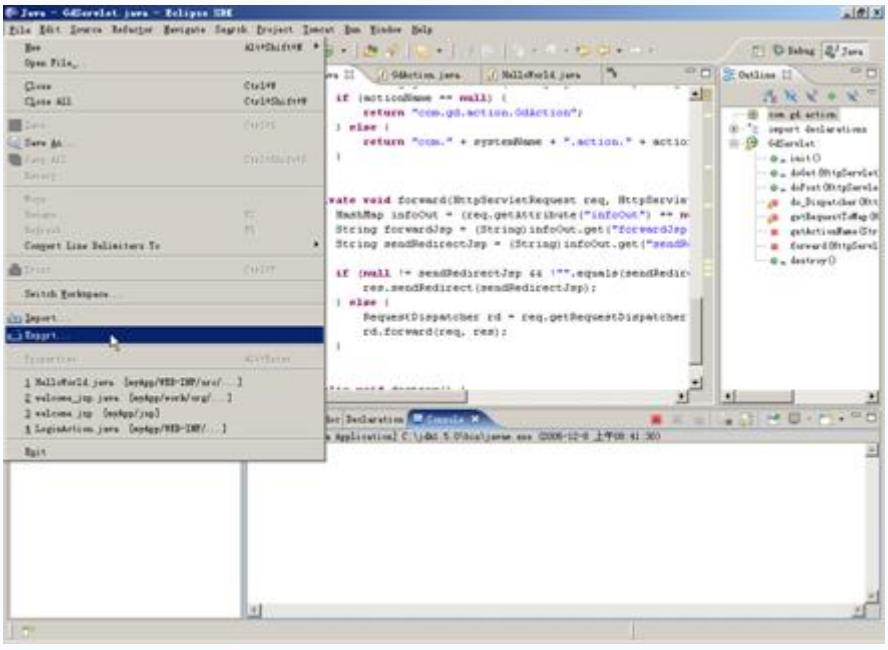


图 4-11 “File”菜单项

步骤 2 在“File”菜单项中，单击“Export”项，弹出“Export”对话框，如图 4-12 所示。

步骤 3 在“Export”对话框中，单击“Java”前面的“＋”将其展开，选择“JAR file”，然后单击“Next”按钮，将出现“JAR Export”对话框，如图 4-13 所示。

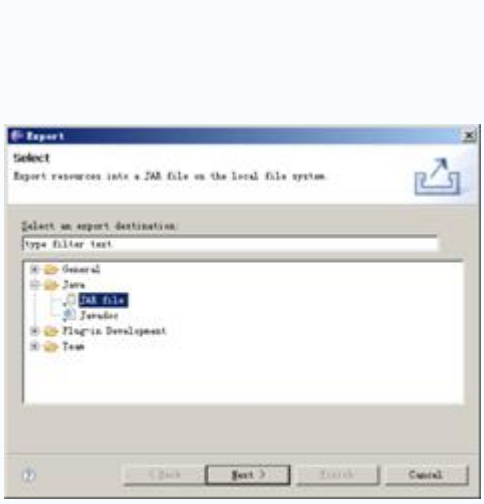


图 4-12 “Export”对话框　　图 4-13 “JAR Export”对话框

步骤 4 在“JAR Export”对话框中，选中“myApp”，其他保持默认选项，然后单击“Browse”按钮，将弹出“另存为”对话框，如图 4-14 所示。

步骤 5 在“另存为”对话框中，选择保存在D盘根目录下，本书中把自己编写的Web框架叫做Nancy，所以文件名填写为“nancy.jar”，然后单击“保存”按钮，返回“JAR Export”对话框，如图 4-15 所示。

图 4-14 “另存为”对话框　　图 4-15 设定完 jar 保存路径和名称的“JAR Export”对话框

步骤6 在“JAR Export”对话框中，单击“Finish”按钮，即可开始打包jar。打包完成后的nancy.jar将位于D盘根目录下。

4.9 使用打包好的 jar 开发一个实现用户注册的示例

下面将通过实现用户注册的示例来演示在一个新的项目中，如何使用打包好的 jar 来开发 MVC 应用程序。实现思路是：首先建立新的项目开发环境，然后实现用户注册、注册成功和修改用户密码的页面，以及相关的业务逻辑，最后验证程序。

4.9.1 在 Eclipse 中建立 Tomcat 工程项目 myMVC 并配置开发环境

在 Eclipse 中建立 Tomcat 工程项目 myMVC 并配置开发环境的步骤如下。

步骤1 运行Eclipse，单击菜单栏中的“File”菜单，Eclipse将显示“File”菜单项。

步骤2 将鼠标移动到“New”上，在出现的子菜单中单击“Project”，Eclipse将弹出“New Project”对话框，如图 4-16 所示。

步骤3 用鼠标选择列表框中“Java”下的“Tomcat Project”，然后单击“Next”按钮，将弹出“New Tomcat Project”对话框，如图 4-17 所示。

步骤4 在“New Tomcat Project”对话框中，在“Project name”文本框中输入“myMVC”，然后单击“Finish”按钮，项目即建立成功，myMVC的目录结构如图 4-18 所示。

步骤5 在myMVC上单击鼠标右键，在弹出的右键菜单中选择“New”下子菜单中的“Package”，将弹出“New Java Package”对话框，如图 4-19 所示。

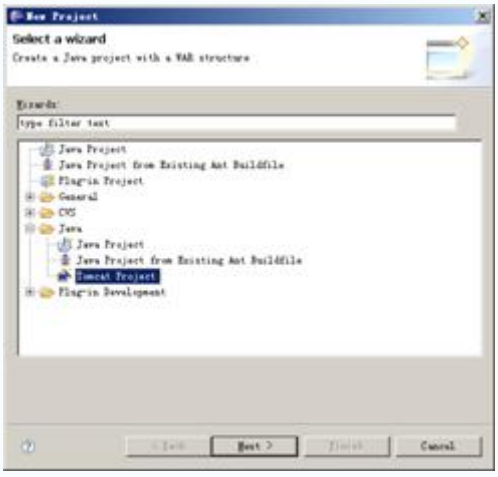


图 4-16 “New Project”对话框

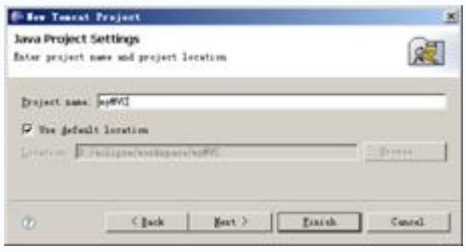


图 4-17 “New Tomcat Project”对话框

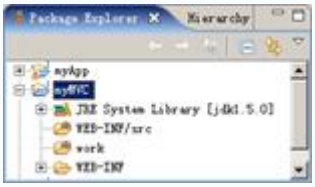


图 4-18 myApp 的目录结构

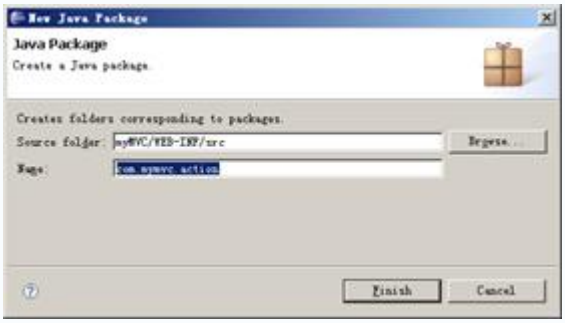


图 4-19 “New Java Package”对话框

步骤 6 在“New Java Package”对话框中，在“Name”文本框中输入“com.mymvc. action”，然后单击“Finish”按钮，即可建立com.mymvc.action包。

步骤 7 在myMVC上单击鼠标右键，在弹出的右键菜单中选择“New”下子菜单中的“Folder”，将弹出“New Folder”对话框，如图 4-20 所示。

图 4-20 “New Folder”对话框

步骤 8 在“New Folder”对话框中在“Folder name”文本框中输入“mymvc/jsp”，然后单击“Finish”按钮，即可建立mymvc/jsp文件夹。

步骤 9 将上节生成的nancy.jar复制到myMVC下WEB-INF的lib文件夹下。

步骤 10 在myMVC上单击鼠标右键，Eclipse将弹出右键菜单。

步骤 11 在弹出的右键菜单中，单击“Properties”，将弹出“Properties for myMVC”对话框，如图 4-21 所示。

步骤 12 在“Properties for myMVC”对话框中，选择对话框左边列表框中的“Java Build Path”。

步骤 13 选择“Properties for myMVC”对话框中间的“Libraries”选项卡。

步骤 14 在“Libraries”选项卡中，单击“Add JARs…”按钮，弹出“JAR Selection”对话框，如图 4-22 所示。

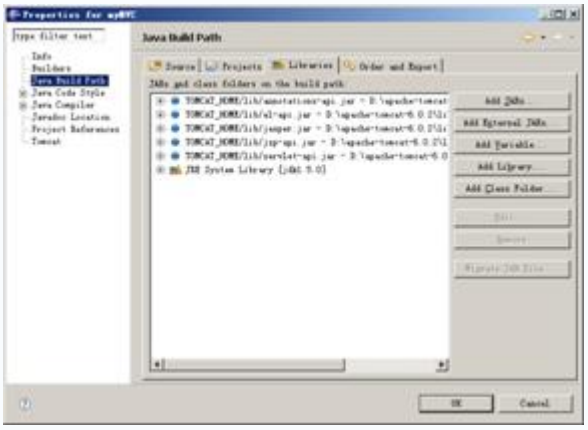


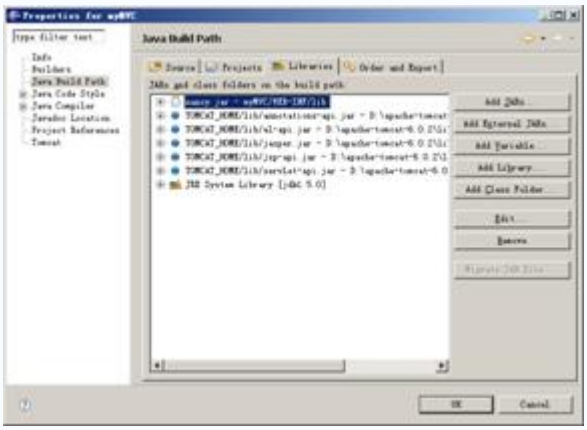
图 4-21 “Properties for myMVC”对话框　　图 4-22 “JAR Selection”对话框

步骤 15 在“JAR Selection”对话框中，展开“myMVC”一直到lib目录下出现nancy.jar。

步骤 16 选中这个jar，然后单击“OK”按钮，返回“Properties for myMVC”对话框，如图 4-23 所示。

步骤 17 在“Properties for myMVC”对话框中，单击“OK”按钮，即可完成对本书中Web框架的配置。

步骤 18 最终配置好的myMVC项目的目录结构如图 4-24 所示。



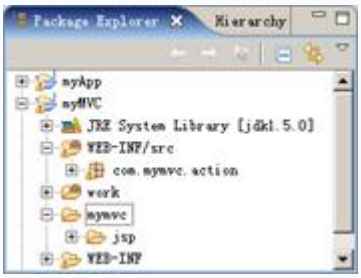


图 4-23 返回“Properties for myMVC”对话框　图 4-24 配置好的 myMVC 项目的目录结构

4.9.2 编写实现用户注册的页面 regedit.jsp

用户注册页面 regedit.jsp 放在\mymvc\jsp 文件夹下。regedit.jsp 的示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
   "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
   javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "regedit"; %>
<%
   HashMap infoOut = (request.getAttribute("infoOut") == null) ? new
       HashMap() : (HashMap)request.getAttribute("infoOut");
```

```
    String msg = infoOut.get("msg") == null ? "" : (String)infoOut.get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户登录验证</title>
<style type="text/css">
<!—
<!--设定 body 的 css 属性-->
body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",
                menubar= no,resizable=yes,toolbar=no,directories=no,
                location=no,scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的登录动作-->
function check() {
    form1.forwardJsp.value = "regedit";
    form1.logicName.value = "RegeditAction";
    submit('<%="regedit_"+session.getId()%>','regedit');
```

```
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myMVC/mymvc/regedit.do" method="post">
<H3><font color='red'><%=msg%></font><H3>
用户名：<input type="text" name="username" value=""><br>
密码：  <input type="password" name="password" value=""><br>
<input type="button" name="button" value="注册" onClick="return check()">
<input type="reset" name="button" value="重置">
   <!--表单中隐藏的元素-->
   <input type="hidden" name="action" value="">
   <input type="hidden" name="forwardJsp" value="">
   <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
   window.name = "<%="regedit_"+session.getId()%>";
</script>
</body>
</html>
```

4.9.3 编写注册成功的页面 success.jsp

注册成功页面 success.jsp 放在\mymvc\jsp 文件夹下。success.jsp 的示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
          "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
       javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "success"; %>
<%
   HashMap infoOut = (request.getAttribute("infoOut") == null) ? new
             HashMap() : (HashMap)request.getAttribute("infoOut");
   String msg = infoOut.get("msg") == null ? "" : (String)infoOut.get("msg");
%>

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

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<title>采用新的框架实现用户注册验证</title>

<style type="text/css">
```

```
<!--

<!--设定 body 的 css 属性-->

body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",
                menubar=no,resizable=yes,toolbar=no,directories=no,location=
                no,scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的登录动作-->
function check() {
    form1.forwardJsp.value = "login";
    form1.logicName.value = "LoginAction";
    submit('<%="success_"+session.getId()%>','login');
}

</script>

</head>

<body leftmargin="0" topmargin="0">
```

```
<form name="form1" action="/myMVC/do" method="post">
<H3><font color='red'><%=msg%></font><H3>
<a href='/myMVC/mymvc/updatePaswor.do?logicName=RegeditAction&forwardJsp
        =updatePassword'>进入修改画面</a>
    <!--表单中隐藏的元素-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
    window.name = "<%="success_"+session.getId()%>";
</script>
</body>
</html>
```

4.9.4 编写修改用户密码的页面 updatePassword.jsp

修改用户密码的页面 updatePassword.jsp 放在\mymvc\jsp 文件夹下。updatePassword. jsp 的示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
        "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
      javax.servlet.http.*,java.text.*,java.math.*"
%>
<%! public static final String _AppId = "updatePassword"; %>
<%
    HashMap infoOut = (request.getAttribute("infoOut") == null) ? new
              HashMap() : (HashMap)request.getAttribute("infoOut");
    String msg = infoOut.get("msg") == null ? "" : (String)infoOut. get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户注册验证</title>
<style type="text/css">
```

```
<!--
<!--设定 body 的 css 属性-->
body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",
                menubar= no,resizable=yes,toolbar=no,directories=no,location=no,
                scrollbars=yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
```

```
}
<!--表单的登录动作-->
function check(action) {
    form1.forwardJsp.value = "updatePassword";
    form1.logicName.value = "RegeditAction";
    submit('<%="updatePassword_"+session.getId()%>',action);
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myMVC/mymvc/regedit.do" method="post">
<H3><font color='red'><%=msg%></font><H3>
用户名：<input type="text" name="username" value=""><br>
密码：  <input type="password" name="password" value=""><br>
<input type="button" name="button" value="修改" onClick="return
            check('update')">
<input type="button" name="button" value="删除" onClick="return
            check('delete')">
    <!--表单中隐藏的元素-->
    <input type="hidden" name="action" value="">
    <input type="hidden" name="forwardJsp" value="">
    <input type="hidden" name="logicName" value="">
</form>
<!--设定页面的名称-->
<script language=Javascript>
    window.name = "<%="updatePassword_"+session.getId()%>";
</script>
</body>
</html>
```

4.9.5 编写业务逻辑 RegeditAction.java

在包 com.mymvc.action 上建立业务逻辑 RegeditAction.java。RegeditAction.java 的示例代码如下：

```
//******* RegeditAction.java**************
package com.mymvc.action;
import java.util.HashMap;
import com.gd.action.Action;
public class RegeditAction implements Action {
    //根据请求，进行页面动作转换
    public HashMap doAction(HashMap infoIn) {
        String action = (infoIn.get("action") == null) ? "" : (String)
                        infoIn.get("action");
        HashMap infoOut = new HashMap();
        if (action.equals(""))              infoOut = this.doInit (infoIn);
        else if (action.equals("regedit")) infoOut = this.doRegedit (infoIn);
        else if (action.equals("query")) infoOut = this.doQuery (infoIn);
        else if (action.equals("update")) infoOut = this.doUpdate (infoIn);
        else if (action.equals("delete")) infoOut = this.doDelete (infoIn);
        return infoOut;
    }
    /**该方法用来实现没有传入动作时要处理的内容
    * @param infoIn
    * @return HashMap
    */
    private HashMap doInit(HashMap infoIn) {
        HashMap infoOut = infoIn;
        try {
            infoOut.put("msg", "请输入用户名和密码");
        } catch(Exception e) {
```

```
        e.printStackTrace();
    } finally {
        return infoOut;
    }
}

/**该方法用来实现注册
* @param infoIn
* @return HashMap
*/
private HashMap doRegedit(HashMap infoIn) {
    HashMap infoOut = infoIn;
    String username = (infoIn.get("username") == null) ? "" :
                (String)infoIn.get("username");
    String password = (infoIn.get("password") == null) ? "" :
                (String)infoIn.get("password");
    try {
        if ("gd".equals(username) && "123456".equals(password)) {
            infoOut.put("forwardJsp", "success");
            infoOut.put("msg", "注册成功");
        } else if ("".equals(username) || "".equals(password)) {
            infoOut.put("msg", "请输入用户名或密码");
        }
    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        return infoOut;
    }
}
```

```
/**该方法用来实现查询
* @param infoIn
* @return HashMap
*/
private HashMap doQuery(HashMap infoIn) {
    HashMap infoOut = infoIn;
    String username = (infoIn.get("username") == null) ? "" :
                    (String)infoIn.get("username");
    String password = (infoIn.get("password") == null) ? "" :
                    (String)infoIn.get("password");
    try {
        if ("gd".equals(username) && "123456".equals(password)){
            infoOut.put("msg", "登录成功");
        } else if("gd".equals(username)&& !"123456".equals (password)){
            infoOut.put("msg", "密码错误");
        } else if(!"gd".equals(username)&& "123456".equals (password)){
            infoOut.put("msg", "用户名错误");
        } else if(!"gd".equals(username)&& !"123456".equals (password)){
            infoOut.put("msg", "用户名和密码都输入错误");
        } else if ("".equals(username) && "".equals(password)){
            infoOut.put("msg", "请输入用户名和密码");
        }
    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        return infoOut;
    }
}
/**该方法用来实现修改
```

```
* @param infoIn
* @return HashMap
*/
private HashMap doUpdate(HashMap infoIn) {
   HashMap infoOut = infoIn;
   String username = (infoIn.get("username") == null) ? "" :
                (String)infoIn.get("username");
   String password = (infoIn.get("password") == null) ? "" :
                (String)infoIn.get("password");
   try {
      if ("gd".equals(username) && "123456".equals(password)){
         infoOut.put("msg", "修改成功");
      } else if ("gd".equals(username)&& !"123456".equals (password)){
         infoO ut.put("msg", "密码错误");
      } else if (!"gd".equals(username)&& "123456".equals (password)){
         infoOut.put("msg", "用户名错误");
      } else if(!"gd".equals(username)&& !"123456".equals (password)){
         infoOut.put("msg", "用户名和密码都输入错误");
      } else if ("".equals(username) && "".equals(password)){
         infoOut.put("msg", "请输入用户名和密码");
      }
   } catch(Exception e) {
      e.printStackTrace();
   } finally {
      return infoOut;
   }
}
/**该方法用来实现删除
* @param infoIn
* @return HashMap
*/
private HashMap doDelete(HashMap infoIn) {
   HashMap infoOut = infoIn;
   String username = (infoIn.get("username") == null) ? "" :
                (String)infoIn.get("username");
   String password = (infoIn.get("password") == null) ? "" :
                (String)infoIn.get("password");
   try {
      if ("gd".equals(username) && "123456".equals(password)) {
         infoOut.put("msg", "删除成功");
      } else if (!"gd".equals(username)) {
         infoOut.put("msg", "删除失败，请填写正确的用户名");
      }
   } catch(Exception e) {
      e.printStackTrace();
   } finally {
```

```
            return infoOut;
        }
    }
}
```

4.9.6 配置 web.xml 文件

将 web.xml 文件放在 myMVC 的 WEB-INF 文件夹下。web.xml 文件的示例代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<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">
    <display-name>web.xml 的示例</display-name>
    <description>
    使用 Servlet 实现注册的例子
    </description>
<!--过滤器-->
<servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>com.gd.action.GdServlet</servlet-class>
</servlet>
<!--拦截.do 的请求-->
    <servlet-mapping>
         <servlet-name>Servlet</servlet-name>
         <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

4.9.7 运行并验证示例

启动 Tomcat，然后在浏览器的地址栏中输入 http://localhost:8080/myMVC/mymvc/ regedit.do?logicName=RegeditAction&forwardJsp=regedit，然后按回车键，即可看到注册页面，如图 4-25 所示。

— 如果用户只输入用户名“gf”，然后单击“注册”按钮，即可看到页面显示“请输入用户名或密码”的提示信息，如图 4-26 所示。

— 输入用户名“gd”，密码“123456”，单击“注册”按钮，即可看到注册成功的页面，如图 4-27 所示。

图 4-25　注册页面

图 4-26　显示“请输入用户名或密码”的提示信息

图 4-27　注册成功的页面

单击“进入修改画面”的链接，进入修改页面，如图 4-28 所示。

图 4-28　修改页面

— 输入用户名“gf”，密码“12345”，单击“修改”按钮，即可看到显示“用户名和密码都输入错误”的提示页面，如图 4-29 所示。

图 4-29　显示“用户名和密码都输入错误”的提示页面

— 输入用户名“gd”，密码“12345”，单击“修改“按钮，即可看到修改成功的页面，如图 4-30 所示。

图 4-30　修改成功的页面

— 输入用户名“gf”，密码“12345”，单击“删除”按钮，即可看到删除失败的页面，如图 4-31 所示。

— 输入用户名“gd”，密码“12345”，单击“删除”按钮，即可看到删除成功的页面，如图 4-32 所示。

图 4-31　删除失败的页面

图 4-32　删除成功的页面

4.10　小　结

本章首先给出了基于 MVC Model 2 规范的 Web 框架的示意图，然后分别针对视图层、控制层和模型层是如何设计的进行了讲解；接着通过两个示例验证了这个框架，并进行了部分修改；最后又将该 Web 框架打包成 jar，并进行了演示如何使用 jar 来开发项目。至此，一个简单的 Web 框架被快速编写完成。

当然，“智者千虑，必有一失；愚者千虑，必有一得”。这仅仅是一个实现了基本功能的 Web 框架，如果读者学习过 Struts，可能会觉得本书的框架太简单了吧，能不能把本书的 Web 框架改得像 Struts 那样强大呢？当然可以，不过我们总得先知道本书的 Web 框架和 Str

uts 到底有哪些区别吧？所以，接下来笔者将会对此框架和 Struts 进行对比，总结出它们的不同之处。

第 10 章 集思广益：将 Web 框架与 Hibernate 整合

在前面几章中，笔者自己实现了一个持久层框架，但所谓“闻道有先后，术业有专攻”，本书的持久层框架还不算成熟，而目前市场上已经有很多成熟的开源项目，比如 Hibernate、JPA 等，它们经过市场的洗礼和检验，已经能够胜任持久层的工作。因此，本书中所实现的 Web 框架也可以和目前开源的持久层框架进行整合，下面将主要讲述如何将 Web 框架与 Hibernate 进行整合。

10.1 Hibernate 概述

Hibernate 在对象和关系型之间建立起一座桥梁，使得开发人员可以方便、快捷地进行持久化开发。Hibernate 的第一个正式版本发布于 2001 年末；2003 年 6 月 Hibernate 2 发布，这一版本提供了对大多数数据库的支持；2003 年末 Hibernate 被 JBoss 吸纳；2005 年 3 月 Hibernate 3 正式发布，至此，Hibernate 获得了巨大的成功。Hibernate 使用起来非常简单，这也是 Hibernate 作者 Gavin King 的一贯思想。

10.1.1 下载 Hibernate

Hibernate 的首页地址是 http://hibernate.org/，其首页页面如图 10-1 所示。

图 10-1 Hibernate 的首页页面

本书使用的版本是 3.2.1，单击首页页面中的“Download”链接，即可转入下载页面，除了需要下载 Hibernate 本身外，还需要下载 Hibernate 的两个支持工具：Middlegen-Hibernate 和 hibernate-extensions。

10.1.2 配置 Hibernate

要使用 Hibernate，还需要 Hibernate 相关的一些 jar：antlr.jar、asm.jar、asm-attrs.jars、cglib.jar、commons-collections.jar、commons-logging.jar、log4j.jar、dom4j.jar、ehcache.jar、jta.jar。这些 jar 在 hibernate-3.2.1 的解压缩的 lib 目录里，最重要的是解压缩目录下的 hibernate3.jar，将这些 jar 放在项目工程的 WEB-INF\lib 目录下。

10.2 使用 Hibernate 自动生成代码的工具

与 Hibernate 相关的配置文件主要有 3 种：数据库定义文件、 POJO 和映射文件。Hibernate 提供的工具可以在这 3 种文件之间进行转换：从数据库定义文件通过 MiddleGen 到映射文件，从映射文件通过 hbm2java 到 POJO。

10.2.1 使用 MiddleGen 从数据库定义文件生成映射文件

这里要新建一个包 com.gc.vo，用来存放 Hibernate 生成的 VO 和配置文件。

把前面下载的 Middlegen-Hibernate-r5.zip 解压缩到 D 盘的根目录下。使用 Middlegen 的具体配置步骤如下：

步骤 1 进入Middlegen-Hibernate-r5\config\database目录，可以看到目录下有不同数据库的XML配置文件，本书使用的是MySQL，所以这里对mysql.xml进行修改。mysql.xml示例代码如下：

```
<property name="database.script.file" value="${src.dir}/ sql/${name}
  -mysql.sql"/>
  <property name="database.driver.file" value="${lib.dir}/ mysql-
    connector- java-5.0.0-beta-bin.jar"/>
  <property name="database.driver.classpath"value="${database. driver.
    file}"/>
  <property name="database.driver" value="org.gjt.mm. mysql.Driver"/>
  <property name="database.url" value="jdbc:mysql: //localhost/myApp"/>
  <property name="database.userid" value="root"/>
  <property name="database.password" value="root"/>
  <property name="database.schema" value=""/>
  <property name="database.catalog" value=""/>
```

```
  <property name="jboss.datasource.mapping" value="mySQL"/>
```

代码说明：

— 修改 database.url 为 jdbc:mysql://localhost/myApp。

— 修改 database.userid 为 root。

— 修改 database.password 为 root。

步骤 2 把前面下载的MySQL的驱动mysql-connector-java-5.0.0-beta-bin.jar放在Middlegen-Hibernate-r5\lib目录下。

步骤 3 进入Middlegen-Hibernate-r5 的根目录，打开build.xml，修改其中的代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE project [
   <!ENTITY database SYSTEM "file:./config/database/hsqldb.xml">
]>
<!--定义一个 project ， basedir="."代表根目录是 build.xml 所在的目录-->
<project name="Middlegen Hibernate" default="all" basedir=".">
<!-- project name="Middlegen Hibernate" default="all" basedir="." -->
  <property file="${basedir}/build.properties"/>
  <property name="name" value="com.gc.vo"/>
  <!-- This was added because we were several people (in a course)
     deploying to same app server>
  <property environment="env"/>
  <property name="unique.name" value="${name}.${env. COMPUTERNAME}"/-->
  <property name="gui" value="true"/>
  <property name="unique.name" value="${name}"/>
  <property name="appxml.src.file" value="${basedir} /src/application.
     xml"/>
  <property name="lib.dir" value="${basedir} /lib"/>
  <property name="src.dir" value="${basedir} /src"/>
  <property name="java.src.dir" value="${src.dir} /java"/>
  <property name="web.src.dir" value="${src.dir} /web"/>
```

```
  <property name="build.dir" value="${basedir} /build"/>
  <property name="build.java.dir" value="${build.dir} /java"/>
  <property name="build.gen-src.dir" value="${build.dir} /gen-src"/>
  <property name="build.classes.dir" value="${build.dir} /classes"/>
  &database;
  <!-- define the datasource.jndi.name in case the imported ejb file
    doesn't -->
  <property name="datasource.jndi.name" value="${name}/ datasource"/>
  <path id="lib.class.path">
    <pathelement path="${database.driver.classpath}"/>
    <fileset dir="${lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <!-- The middlegen jars -->
    <!--fileset dir="${basedir}/.."-->
    <fileset dir="${basedir}/middlegen-lib">
      <include name="*.jar"/>
    </fileset>
  </path>
  <target name="init">
    <available property="xdoclet1.2+" classname="xdoclet.modules.
      hibernate.HibernateDocletTask" classpathref="lib.class.path"/>
  </target>
  <!-- ================================================== --
>
  <!-- Fails if XDoclet 1.2.x is not on classpath      -->
  <!-- ================================================== -->
  <target name="fail-if-no-xdoclet-1.2" unless="xdoclet1.2+">
    <fail>
    You must download several jar files before you can build Middlegen.
    </fail>
  </target>
  <!-- ================================================== --
>
  <!-- Create tables                                  -->
  <!-- ================================================== --
>
  <target
    name="create-tables"
```

```
    depends="init,fail-if-no-xdoclet-1.2,check-driver-present,
        panic-if-driver-not-present"
    description="Create tables"
  >
    <echo>Creating tables using URL ${database.url}</echo>
    <sql
      classpath="${database.driver.classpath}"
      driver="${database.driver}"
      url="${database.url}"
      userid="${database.userid}"
      password="${database.password}"
      src="${database.script.file}"
      print="true"
      output="result.txt"
    />
  </target>
  <target name="check-driver-present">
    <available file="${database.driver.file}" type="file" property=
      "driver.present"/>
  </target>
  <target name="panic-if-driver-not-present" unless="driver.present">
    <fail>
    The JDBC driver you have specified by including one of the files
      in ${basedir}/config/database
    doesn't exist. You have to download this driver separately and
      put it in ${database.driver.file}
    Please make sure you're using a version that is equal or superior
      to the one we looked for.
    If you name the driver jar file differently, please update the
      database.driver.file property
    in the ${basedir}/config/database/xxx.xml file accordingly.
    </fail>
  </target>
  <!-- ================================================================= --
>
  <!-- Run Middlegen                                                     -->
  <!-- ================================================================= --
>
  <target
    name="middlegen"
    description="Run Middlegen"
    unless="middlegen.skip"
    depends="init,fail-if-no-xdoclet-1.2,check-driver-present,
      panic-if-driver-not-present"
  >
    <mkdir dir="${build.gen-src.dir}"/>
```

```
 <echo message="Class path = ${basedir}"/>
<taskdef
  name="middlegen"
  classname="middlegen.MiddlegenTask"
  classpathref="lib.class.path"
/>
<middlegen
  appname="${name}"
  prefsdir="${src.dir}"
  gui="${gui}"
  databaseurl="${database.url}"
  initialContextFactory="${java.naming.factory.initial}"
  providerURL="${java.naming.provider.url}"
  datasourceJNDIName="${datasource.jndi.name}"
  driver="${database.driver}"
  username="${database.userid}"
  password="${database.password}"
  schema="${database.schema}"
  catalog="${database.catalog}"
>
  <!--
  We can specify what tables we want Data generated for.
  If none are specified, Data will be generated for all tables.
  Comment out the <table> elements if you want to generate for all tables.
  Also note that table names are CASE SENSITIVE for certain databases,
  so on e.g. Oracle you should specify table names in upper case.
  -->
  <!--table generate="true" name="flights" pktable="flights_ pk"/>
  <table name="reservations"/-->
  <!--
  If you want m:n relations, they must be specified like this.
  Note that tables declare in multiple locations must all have
  the same value of the generate attribute.
  -->
  <!--many2many>
    <tablea generate="true" name="persons"/>
    <jointable name="reservations" generate="false"/>
    <tableb generate="true" name="flights"/>
  </many2many-->
  <!-- Plugins - Only Hibernate Plugin has been included with this
     special distribution  -->

  <!--
  If you want to generate XDoclet markup for hbm2java to include
     in the POJOs then
  set genXDocletTags to true. Also, composite keys are generated
```

```
        as external classes which is
            recommended. If you wish to keep them internal then set genIntergratedComposite
                Keys to true.
            Since r4 the ability to customise the selection of JavaTypes is now provided. The is
                a
            recommended type mapper provided as shown. It is optional - if not provided then
                Middlegen
      itself will select the Java mapping (as it did previously).
            These settings are optional thus if they are not define here values default to false.
      -->
      <hibernate
        destination="${build.gen-src.dir}"
        package="${name}.hibernate"
        genXDocletTags="false"
        genIntergratedCompositeKeys="false"
        javaTypeMapper="middlegen.plugins.hibernate.
          HibernateJavaTypeMapper"
      />

    </middlegen>
    <mkdir dir="${build.classes.dir}"/>
  </target>
  <!-- ================================================================== -->
  <!-- Compile business logic (hibernate)              -->
  <!-- ================================================================== -->
  <target name="compile-hibernate" depends="middlegen" description=
    "Compile hibernate Business Domain Model">
    <javac
      srcdir="${build.gen-src.dir}"
      destdir="${build.classes.dir}"
      classpathref="lib.class.path"
    >
      <include name="**/hibernate/**/*"/>
    </javac>
  </target>
  <!-- ================================================================== --
>
  <!-- Run hbm2java    depends="middlegen"              -->
  <!-- ================================================================== --
>
  <target name="hbm2java" description="Generate .java from .hbm files.">
    <taskdef
      name="hbm2java"
      classname="net.sf.hibernate.tool.hbm2java.Hbm2JavaTask"
      classpathref="lib.class.path"
    />
```

```
      <hbm2java output="${build.gen-src.dir}">
        <fileset dir="${build.gen-src.dir}">
          <include name="**/*.hbm.xml"/>
        </fileset>
      </hbm2java>
  </target>
  <!-- ================================================================= --
>
  <!-- Build everything                                                  -->
  <!-- ================================================================= --
>
  <target name="all" description="Build everything" depends="compile-
    hibernate"/>
  <!-- ================================================================= --
>
  <!-- Clean everything                                                  -->
  <!-- ================================================================= --
>
  <target name="clean" description="Clean all generated stuff">
    <delete dir="${build.dir}"/>
  </target>
  <!-- ================================================================= --
>
  <!-- Brings up the hsqldb admin tool                                   -->
  <!-- ================================================================= --
>
  <target name="hsqldb-gui" description="Brings up the hsqldb admin tool">
    <property name="database.urlparams" value="?user=${database.
      userid}&amp;password=${database.password}"/>
    <java
      classname="org.hsqldb.util.DatabaseManager"
      fork="yes"
      classpath="${lib.dir}/hsqldb-1.7.1.jar;${database.driver.
        classpath}"
      failonerror="true"
    >
      <arg value="-url"/>
      <arg value="${database.url}${database.urlparams}"/>
      <arg value="-driver"/>
      <arg value="${database.driver}"/>
    </java>
  </target>
  <!-- ================================================================= --
>
  <!-- Validate the generated xml mapping documents                      -->
```

```
  <!-- ========================================================== -->
  <target name="validate">
    <xmlvalidate failonerror="no" lenient="no" warn="yes">
      <fileset dir="${build.gen-src.dir}/airline/hibernate" includes=
        "*.xml" />
    </xmlvalidate>
  </target>
</project>
```

代码说明：

— 修改<!ENTITY database SYSTEM "file:./config/database/hsqldb.xml">中的 hsqldb. xml 为 mysql.xml。

— 修改<property name="name" value="airline"/>中的 value 为 com.gc.vo。

— 修改 package="${name}.hibernate"为 package="${name} "。

— 修改 genXDocletTags="false"为 genXDocletTags="true"，使其可以产生 XDoclet。

步骤 4 在使用Middlegen之前，要先配置Ant。从http://ant.apache.org/中下载Ant，Ant的下载页面如图 10-2 所示。

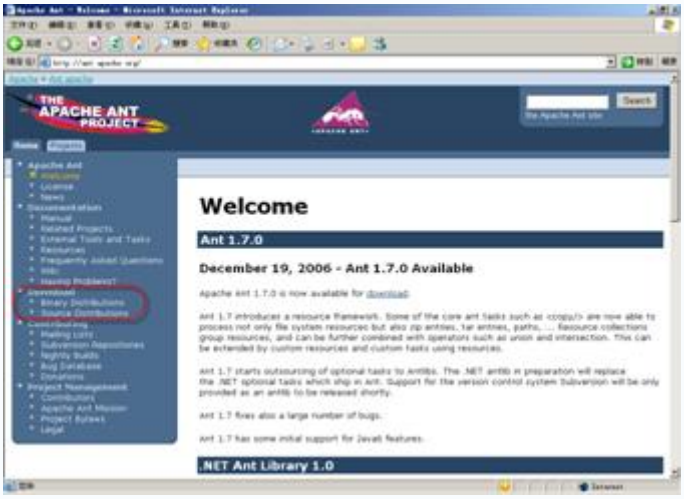


图 10-2 Ant 的下载页面

单击“Binary Distributions”，可以下载 Ant 提供的最新稳定版本；单击“Source Distributions”，可以下载 Ant 提供的最新源代码版本。这里单击“Binary Distributions”，下载 Ant 的 1.7.0 版本 apache-ant-1.7.0-bin.zip

下载 apache-ant-1.7.0-bin.zip 完毕，并解压缩到 D 盘根目录下后，即可开始配置环境变量。具体配置方法是：按照前面设定 JDK 环境变量的方式设定 Ant 的系统变量 Path 和 ANT_HOME，即 Path=D:\apache-ant-1.7.0\bin 和 ANT_HOME=D:\apache-ant-1.7.0，如图 10-3 和图 10-4 所示。

图 10-3　设定系统变量 Path　　图 10-4　设定系统变量 ANT_HOME

在 cmd 命令框中输入 ant 命令，如果出现如图 10-5 所示内容即可证明 Ant 安装成功

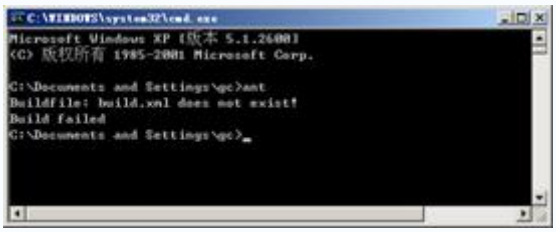


图 10-5　查看 Ant 是否安装成功

步骤 5 通过cmd控制台，进入D:\Middlegen-Hibernate-r5 目录下，然后输入ant，运行Ant，如图 10-6 所示。

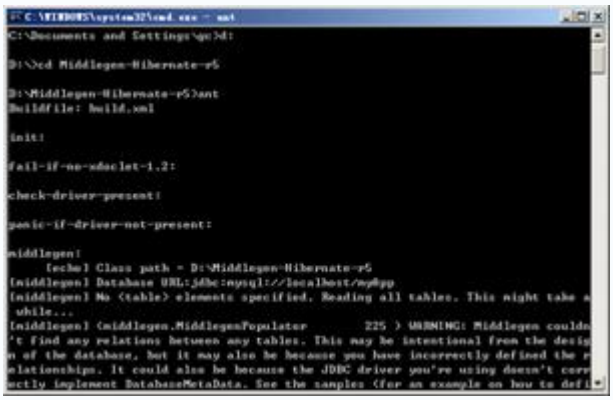


图 10-6　运行 Ant 的页面

步骤 6 运行Ant成功后，即可出现Middlegen的页面，它已经把第 7 章中建立的user表结构导入了，如图 10-7 所示。

图 10-7　Middlegen 的页面

步骤 7 点击表结构中的user，在Middlegen页面的下半部分会出现与user相关的一些属性，这里只修改Key generator（主键生成方式）为increment，其他不变，如图 10-8 所示。

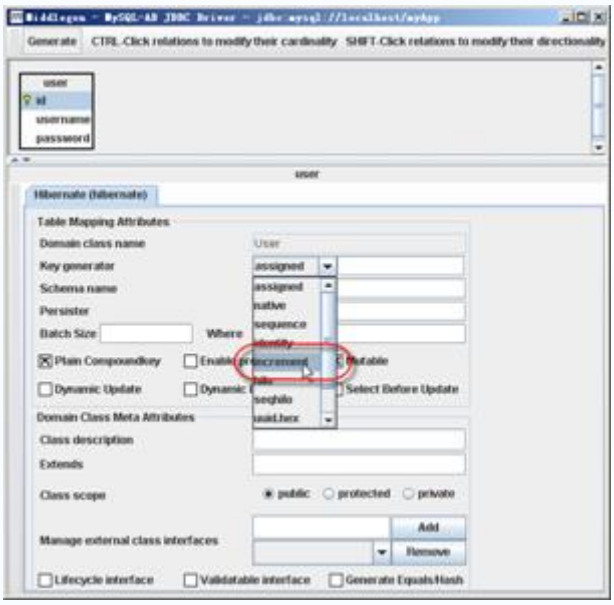


图 10-8　修改 Key generator（主键生成方式）为 increment

步骤 8 点击表结构中的id、username或password，在Middlegen页面的下半部分会出现与id、username或password相关的一些属性，这里不做任何修改，如图 10-9 所示。

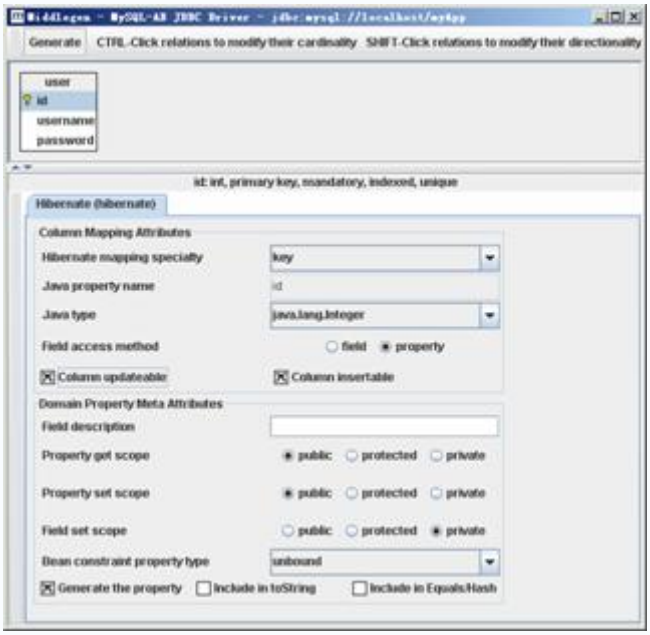


图 10-9　与 id、username 或 password 相关的一些属性

步骤 9 单击Middlegen页面左上方的“Generate”按钮，即可生成与user表相对应的映射文件User.hbm.xml，该文件存放在Middlegen-Hibernate-r5\build\gen-src\com\gc\vo目录下。User.hbm.xml的示例代码如下：

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >
<hibernate-mapping>
<!--
    Created by the Middlegen Hibernate plugin 2.1
    http://boss.bekk.no/boss/middlegen/
    http://www.hibernate.org/
-->
<class
    name="com.gc.vo.User"
    table="user"
>
    <id
        name="id"
        type="java.lang.Integer"
        column="id"
    >
        <generator class="increment" />
    </id>
    <property
        name="username"
        type="java.lang.String"
        column="username"
        not-null="true"
        length="32"
    />
    <property
        name="password"
```

```
        type="java.lang.String"
        column="password"
        not-null="true"
        length="32"
    />
</class>
</hibernate-mapping>
```

10.2.2 使用 hbm2java 从映射文件生成 POJO

从映射文件生成 POJO，要使用 hbm2java，具体步骤如下：

步骤1 检查D:\hibernate-extensions-2.1.3\tools\bin目录下setenv.bat中CP的设定是否准确。用记事本打开setenv.bat，然后查看CP中各种jar的设定是否正确，并增加set HIBERNATE_HOME=D:\hibernate-3.2。setenv.bat的示例代码如下：

```
@echo off
rem ------------------------------------------------------------
rem Setup environment for hibernate tools
rem ------------------------------------------------------------
set JDBC_DRIVER=C:\Progra~1\SQLLIB\java\db2java.zip;C:\mm.mysql-2.0. 14\mm.mysql-2.0.14-bin.j
ar
set HIBERNATE_HOME=D:\hibernate-3.2
set HIBERNATETOOLS_HOME=%~dp0..
echo HIBERNATETOOLS_HOME set to %HIBERNATETOOLS_HOME%
if "%HIBERNATE_HOME%" == "" goto noHIBERNATEHome
set CORELIB=%HIBERNATE_HOME%\lib
set LIB=%HIBERNATETOOLS_HOME%\lib
set CP=%CLASSPATH%;%JDBC_DRIVER%;%HIBERNATE_HOME%\hibernate2.jar;% CORELIB%\com
mons-logging-1.0.4.jar;%CORELIB%\commons-lang-1.0.1.jar;%CORELIB%\cglib-2.1.3.jar;%CORELIB
%\dom4j-1.6.1.jar;%CORELIB%\odmg-3.0.jar;%CORELIB%\xml-apis.jar;%CORELIB%\xerces-2.6.2.jar;
```

```
%CORELIB%\xalan-2.4.0.jar;%LIB%\jdom.jar;%CORELIB%\commons-collections-2.1.1.jar;%LIB%\..\
hibernate-tools.jar
if not "%HIBERNATE_HOME%" == "" goto end
:noHIBERNATEHome
echo HIBERNATE_HOME is not set. Please set HIBERNATE_HOME.
goto end
:end
```

注意：因为 hibernate-extensions-2.1.3 目前还不支持 Hibernate 3，所以这里要麻烦一点，按照下载 Hibernate 3 同样的方法下载 Hibernate 2 并解压缩，把解压缩后 hibernate-2.1 目录下的 hibernate2.jar 放在 hibernate-3.2 目录下，把 hibernate-2.1\lib 目录下的 commons-lang-1.0.1.jar、odmg-3.0.jar 和 xalan-2.4.0.jar 这 3 个 jar 放在 hibernate-3.2\lib 目录下。然后修改 CP 中各个 jar 的版本号，使其与 hibernate-3.2\lib 目录下相应 jar 的版本号一致。

步骤 2 通过cmd控制台，进入D:\hibernate-extensions-2.1.3\tools\bin目录下，然后输入hbm2javaD:\Middlegen-Hibernate-r5\build\gen-src\com\gc\vo\User.hbm.xml--output=D:\ Middlegen-Hibernate-r5\build\gen-src\，即可在D:\Middlegen-Hibernate-r5\build\gen-src\com\ gc\vo目录下生成User.java，如图 10-10 所示。

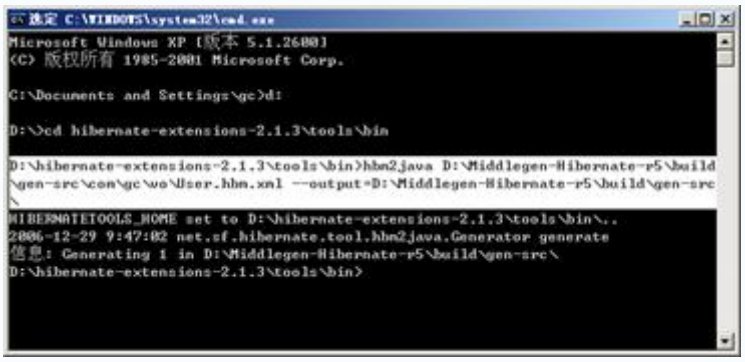


图 10-10　运行 hbm2java

注意：output=D:\Middlegen-Hibernate-r5 中的=两边不能有空格。

步骤 3 生成的 User.java 的示例代码如下：

```
//******* User.java**************
package com.gc.vo;
import java.io.Serializable;
import org.apache.commons.lang.builder.ToStringBuilder;
/** @author Hibernate CodeGenerator */
```

```
public class User implements Serializable {
    /** identifier field */
    private Integer id;
    /** persistent field */
    private String username;
    /** persistent field */
    private String password;
    /** full constructor */
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
    /** default constructor */
    public User() {
    }
    public Integer getId() {
        return this.id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return this.username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return this.password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String toString() {
        return new ToStringBuilder(this)
            .append("id", getId())
            .toString();
    }
}
```

上面通过 MiddleGen 和 hbm2java 这两个工具完成了从数据库定义文件到映射文件，从映射文件到 POJO 的转换。

10.3 Web 框架整合 Hibernate 实现用户注册的示例

前面介绍了 Hibernate 的相关知识，接下来将通过用户注册的示例来演示 Web 框架与 Hibernate 的整合。

10.3.1 整合 Hibernate 环境的配置

将 hibernate3.jar、antlr.jar、asm.jar、asm-attrs.jars、cglib.jar、commons-collections.jar、log4j.jar、dom4j.jar、ehcache.jar、jta.jar 这些 jar 加入到 myApp 的 Classpath 中，如图 10-11 所示。

图 10-11 与 Hibernate 相关的 jar 加入到 Classpath 中

10.3.2 编写 web.xml 文件

建立 web.xml 文件，放在 myApp\WEB-INF 目录下。web.xml 的示例代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<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">
    <display-name>web.xml 的示例</display-name>
<!--日志过滤器-->
<filter>
    <filter-name>GdLogFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdLogFilter
    </filter-class>
```

```
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdLogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--访问执行时间过滤器-->
<filter>
    <filter-name>GdExecuteTimeFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdExecuteTimeFilter
    </filter-class>
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdExecuteTimeFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--编码过滤器-->
<filter>
    <filter-name>SetCharacterEncoding</filter-name>
    <filter-class>
        com.gd.web.filter.GdSetCharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>GBK</param-value>
    </init-param>
</filter>
```

```
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>SetCharacterEncoding</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--文件解析过滤器-->
<filter>
    <filter-name>GdParseXmlFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdParseXmlFilter
    </filter-class>
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdParseXmlFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--数值转换过滤器-->
<filter>
    <filter-name>GdReqToInfoInAndOutFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdReqToInfoInAndOutFilter
    </filter-class>
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdReqToInfoInAndOutFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

```
<!--方法调用过滤器-->
<filter>
    <filter-name>GdExecuteMethodFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdExecuteMethodFilter
    </filter-class>
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdExecuteMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--返回页面过滤器-->
<filter>
    <filter-name>GdForwardFilter</filter-name>
    <filter-class>
        com.gd.web.filter.GdForwardFilter
    </filter-class>
</filter>
<!--过滤所有的访问-->
<filter-mapping>
    <filter-name>GdForwardFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!--控制器-->
<servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>com.gd.mvc.servlet.GdServlet</servlet-class>
</servlet>
```

```
<!--拦截.do 的请求-->
    <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

10.3.3 编写用户注册页面 regedit.jsp

使用前面的用户注册页面 regedit.jsp，包含注册和重置两个按钮，放在 myApp\gc\jsp 目录下。示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
  "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
    javax.servlet.http.*,java.text.*,java.math.*,com.gc.vo.User,com.
        gd.mvc.io.InfoInAndOut,com.gd.mvc.io.impl.GdInfoInAndOut"
%>
<%! public static final String _AppId = "regedit"; %>
<%
    InfoInAndOut infoOut = (request.getAttribute("infoOut") == null) ?
        new GdInfoInAndOut() : (InfoInAndOut)request.getAttribute          ("infoOut");
    String msg = infoOut.get("msg") == null ? "" : (String)infoOut.
        get("msg");
    User user = infoOut.get("user") == null ? new User() : (User)infoOut.
        get("user");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<title>采用新的框架实现用户注册验证</title>
<style type="text/css">
<!--
<!--设定 body 的 css 属性-->
body {
    background-color: #FFFFFD;
    font-family: Verdana, "宋体";
    font-size: 12px;
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",menubar=
        no,resizable=yes,toolbar=no,directories=no,location=no,scrollbars
        =yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
```

```
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的提交动作-->
function check() {
    submit('<%="regedit_"+session.getId()%>','regedit');
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myApp/gc/regedit.do" method="post">
<H3><font color='red'><%=msg%></font><H3>
用户名：<input type="text" name="username_user" value="<%=user. getUsername()
    %>"><br>
密码：<input type="password" name="password_user" value="<%=user. getPassword()
    %>"><br>
<input type="button" name="button" value="注册" onClick="return check ()">
<input type="reset" name="button" value="重置">
<!--页面中隐藏的一些元素-->
    <input type="hidden" name="action" value="">
</form>
<!--定义页面的名称-->
<script language=Javascript>
    window.name = "<%="regedit_"+session.getId()%>";
</script>
</body>
```

```
</html>
```

10.3.4 编写用户注册成功页面 success.jsp

使用前面的用户注册成功页面 success.jsp，放在 myApp\gc\jsp 目录下。示例代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import=
   "java.sql.*" errorPage="" %>
<%@ page import="java.sql.*,java.util.*,javax.servlet.*,
   javax.servlet.http.*,java.text.*,java.math.*,com.gd.mvc.io.
      InfoInAndOut, com.gd.mvc.io.impl.GdInfoInAndOut"
%>
<%! public static final String _AppId = "login"; %>
<%
   InfoInAndOut infoOut = (request.getAttribute("infoOut") == null) ?
      new GdInfoInAndOut() : (InfoInAndOut)request.getAttribute
      ("infoOut");
   String msg = infoOut.get("msg") == null ? "" : (String)infoOut. get("msg");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>采用新的框架实现用户注册验证</title>
<style type="text/css">
<!--
<!--设定 body 的 css 属性-->
body {
   background-color: #FFFFFD;
   font-family: Verdana, "宋体";
   font-size: 12px;
```

```
    font-style: normal;
}
-->
</style>
<script language=Javascript>
<!--表单提交的方法-->
function submit(target, action) {
    form1.target = target;
    form1.action.value = action;
    form1.submit();
}
<!--打开窗口-->
function openWin(name, url, width, height) {
    var screenWidth = screen.width;
    var screenHeight = screen.height;
    var w;
<!--设定窗口的属性-->
    w=window.open(url,name, "width="+width+",height="+height+",menubar=
        no,resizable=yes,toolbar=no,directories=no,location=no,scrollbars
       =yes,status=yes,copyhistory=0");
    w.moveTo((screenWidth-width)/2, (screenHeight-height)/2);
    w.focus();
}
<!--退出页面-->
function goExit() {
    window.close();
}
<!--表单的提交动作-->
function check() {
```

```
    submit('<%="login_"+session.getId()%>','login');
}
</script>
</head>
<body leftmargin="0" topmargin="0">
<form name="form1" action="/myApp/do" method="post">
<H3><font color='red'><%=msg%></font><H3>
<!--页面中隐藏的一些元素-->
    <input type="hidden" name="action" value="">
</form>
<!--定义页面的名称-->
<script language=Javascript>
    window.name = "<%="login_"+session.getId()%>";
</script>
</body>
</html>
```

10.3.5 建立数据库表结构

使用在 MySQL 中创建的数据库 myApp，用户名和密码都为 root；数据库中表名为 user，包含 3 个字段：id、username 和 password，设定 id 为主键，自动增长。

10.3.6 根据数据库表生成映射文件 User.hbm.xml

使用前面介绍的方法，利用 Middlegen 生成映射文件 User.hbm.xml，放在包 com.gc.vo 下。User.hbm.xml 的示例代码如下：

```
<?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>
<!--
```

```
    Created by the Middlegen Hibernate plugin 2.1

    http://boss.bekk.no/boss/middlegen/

    http://www.hibernate.org/
-->
<class
    name="com.gc.vo.User"
    table="user"
>
    <id
        name="id"
        type="java.lang.Integer"
        column="id"
    >
        <generator class="increment" />
    </id>
    <property
        name="username"
        type="java.lang.String"
        column="username"
        not-null="true"
        length="32"
    />
    <property
        name="password"
        type="java.lang.String"
        column="password"
        not-null="true"
        length="32"
    />
```

```
</class>
</hibernate-mapping>
```

10.3.7 根据映射文件生成 POJO

使用 hbm2java 根据映射文件生成 POJO，放在包 com.gc.vo 下。User.java 的示例代码如下：

```
//******* User.java**************
package com.gc.vo;
import java.io.Serializable;
/** @author Hibernate CodeGenerator */
public class User implements Serializable {
    /** identifier field */
    private Integer id;
    /** persistent field */
    private String username;
    /** persistent field */
    private String password;
    /** full constructor */
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
    /** default constructor */
    public User() {
    }
    public Integer getId() {
        return this.id;
    }
    public void setId(Integer id) {
        this.id = id;
```

```
    }
    public String getUsername() {
        return this.username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return this.password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

10.3.8 编写接口 UserDAOHibernate.java

如果读者使用过 JUnit，就会明白针对接口编程的好处了，因为它会非常方便开发人员的测试，因此，这里首先编写一个接口，在接口里定义操作数据的实现方式。在包 com.gc.dao 中编写一个接口类 UserDAOHibernate.java，UserDAOHibernate.java 的示例代码如下：

```
//******* UserDAOHibernate.java**************
package com.gc.dao;
import java.sql.SQLException;
import java.util.List;
import com.gc.vo.User;
public interface UserDAOHibernate {
    /**
     * 根据 Object 实现新增
     * @param user
     * @throws SQLException
     */
    public abstract void createUser(User user) throws SQLException;
    /**
     * 根据 Object 实现修改
     * @param user
     * @throws SQLException
     */
    public abstract void updateUser(User user) throws SQLException;
    /**
     * 根据 Object 实现删除
     * @param user
     * @throws SQLException
     */
```

```
    public abstract void deleteUser(User user) throws SQLException;
    /**
     * 根据 Object 实现批量新增
     * @param user
     * @throws SQLException
     */
    public abstract int createListUser(List list) throws SQLException;
}
```

10.3.9 编写实现类 UserDAOHibernateImpl.java

前面在接口中定义了操作数据的实现方式，还需要编写具体操作数据的实现代码，因此，这里编写接口 UserDAOHibernate 的实现类 UserDAOHibernateImpl.java。UserDAOHibernateImpl.java 的示例代码如下：

```
//******* UserDAOHibernateImpl.java**************
package com.gc.dao.impl;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import com.gc.vo.User;
import com.gc.dao.UserDAOHibernate;
public class UserDAOHibernateImpl implements UserDAOHibernate {
    private Session session = null;
    /**
     * 构造函数
     * @param conn
     */
    public UserDAOHibernateImpl(Session session) {
        this.session = session;
    }
    /**
     * 根据 Object 实现新增
```

```
 */
public void createUser(User user) throws SQLException{
    session.save(user);
}
/**
 * 根据 Object 实现修改
 */
public void updateUser(User user) throws SQLException {
    session.update(user);
}
/**
 * 根据 Object 实现删除
 */
public void deleteUser(User user)  throws SQLException{
    session.delete(user);
}
/**
 * 根据 Object 实现批量新增
 */
public int createListUser(List list) throws SQLException{
    User user = null;
    int counts = 0;
    for (int i = 0; list != null && list.size() > i; i++) {
        user = (User)list.get(i);
        createUser(user);
        counts ++;
    }

    return counts;
```

```
    }
}
```

10.3.10 编写配置文件 config-servlet.xml

编写配置文件 config-servlet.xml，放在 WEB-INF 目录下。config-servlet.xml 的示例代码如下：

```
<?xml version="1.0" encoding="GBK"?>
<nancy-config>
    <bean id="viewResolver">
        <prefix>/gc/jsp/</prefix>
       <suffix>.jsp</suffix>
    </bean>
    <bean id="actionResolver">
        <extends>com.gd.mvc.action.GdAction</extends>
    </bean>
    <bean id="valueObjects">
        <valueObject id="user" class="com.gc.vo.User" table="user"
          type="single"/>
    </bean>
    <page path="page" actionName="com.gc.action.RegeditActionNew">
        <forward name="page" path="/gc/jsp/page.jsp"/>
    </page>
    <page path="login" actionName="com.gc.action.LoginActionNew">
        <forward name="login" path="/gc/jsp/login.jsp"/>
        <action action="login" method="doLogin" forward="login"/>
    </page>
    <page path="regedit" actionName="com.gc.action.RegeditActionHibernate">
        <forward name="regedit" path="/gc/jsp/regedit.jsp"/>
        <action action="regedit" method="doRegedit" forward="regedit"/>
```

```
    </page>
```

缩不上去，如果缩上去字就太紧了，只有这样了。
其它章也有这样情况就不一一标明
lgog

```
    <page path="updatePassword" actionName="com.gc.action.RegeditActionNew">
        <forward name="updatePassword" path="/gc/jsp/updatePassword. jsp"/>
        <action action="update" method="doUpdate" forward="update
        Password"/>
        <action action="delete" method="doDelete" forward="update
        Password"/>
    </page>
</nancy-config>
```

10.3.11 编写 Hibernate 的配置文件 hibernate.cfg.xml

编写 Hibernate 的配置文件 hibernate.cfg.xml，放在 WEB-INF\src 目录下。hibernate.cfg.xml 的示例代码如下：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
        <!--定义方言-->
        <property name="dialect">org.hibernate.dialect.MySQLDialect
        </property>
        <!--定义驱动-->
        <property name="connection.driver_class">com.mysql.jdbc.Driver
        </property>
        <!--定义 url-->
```

```
        <property name="connection.url">jdbc:mysql://localhost/myApp
        </property>
        <!--定义用户名-->
        <property name="connection.username">root</property>
        <!--定义密码-->
        <property name="connection.password">root</property>
        <!--定义是否显示 sql-->
        <property name="show_sql">true</property>
        <!-- Mapping files -->
        <mapping resource="com/gc/vo/User.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

10.3.12 编写控制器 RegeditActionHibernate.java

编写控制器 RegeditActionHibernate.java，该类放在包 com.gc.action 中。RegeditActionHibernate.java 的示例代码如下：

```
//******* RegeditActionHibernate.java**************
package com.gc.action;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.gd.mvc.action.impl.GdAction;
import com.gd.mvc.exception.VODataValidate;
import com.gc.dao.UserDAOHibernate;
import com.gc.dao.impl.UserDAOHibernateImpl;
import com.gc.vo.User;
public class RegeditActionHibernate extends GdAction{
    private User user = null;
    private String username = null;
    private String password = null;
    public void doBegin() throws VODataValidate, Exception{
        //获取页面中的值
        user = (User)infoIn.getVO();
        username = user.getUsername();
        password = user.getPassword();
        infoOut.put("msg", "请输入用户名和密码");
    }
    public void doEnd() {
        infoOut.put("user", user);
    }
    /**该方法用来实现注册
    */
    public void doRegedit() throws Exception{
```

```
        SessionFactory sessionFactory = null;
        Transaction transaction = null;
        Session session = null;
        try {
            //通过配置文件获取 SessionFactory
            sessionFactory = new Configuration().configure().
                build SessionFactory();
            //一个 session 类似一个连接
            session = sessionFactory.openSession();
            //进行事务处理
            transaction = session.beginTransaction();
            UserDAOHibernate userDao = new UserDAOHibernateImpl(session);
            userDao.createUser(user);
            transaction.commit();
            infoOut.put("msg", "注册成功");
        }catch(Exception e) {
            transaction.rollback();
            e.printStackTrace();
        } finally {
            session.close();
            sessionFactory.close();
        }
    }
}
```

10.3.13 运行并验证用户注册示例

启动 Tomcat，在浏览器地址栏中输入 http://localhost:8080/myApp/gc/regedit.do，按回车键，即可看到用户注册的页面，如图 10-12 所示。

在用户注册页面中，输入用户名“gf”，密码“123456”，然后单击“注册”按钮，即可看到注册成功的页面，如图 10-13 所示。

图 10-12 用户注册的页面

图 10-13 注册成功的页面

此时查看 Eclipse 的控制台，即可看到 Hibernate 的输出信息，如图 10-14 所示。

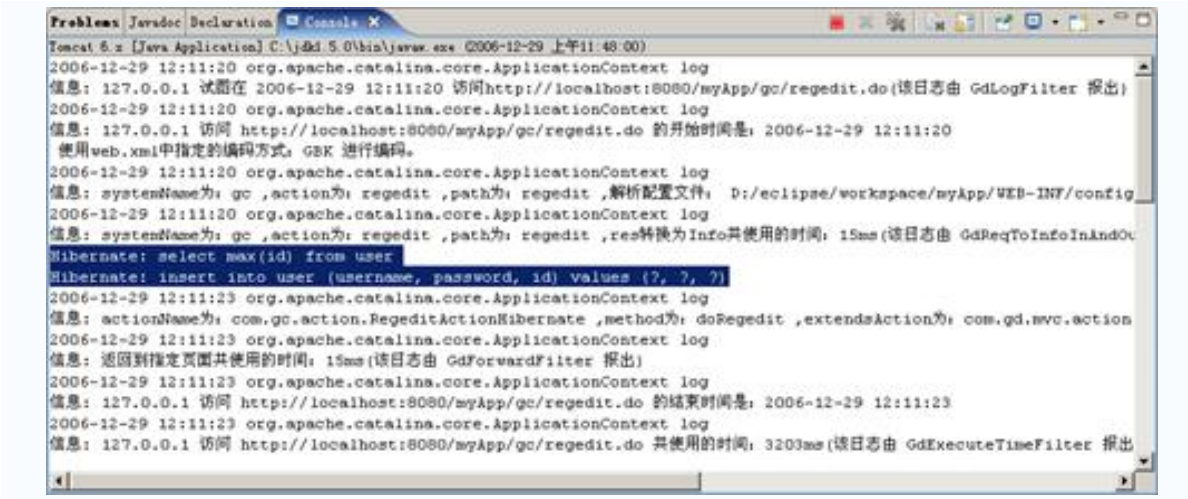


图 10-14　Hibernate 的输出信息

从上面的示例可以看出，Hibernate 的几个主要类是 Configuration、SessionFactory、Session、Transaction 等。

使用 Hibernate 的基本步骤是：首先编写映射文件和 POJO，然后使用它的几个主要类进行数据库的操作。

10.4　小　结

本章主要讲解了 Hibernate 的基本知识和相关自动生成代码工具的使用，最后通过一个用户注册的示例讲解了本书的 Web 框架和 Hibernate 之间的简单整合。

至此，本书关于框架的讲解基本就结束了，但聪明的读者一定可以看到，本书中所有的代码日志和异常的输出都是使用 System 和 e.printStackTrace 来实现的，这样对于日志的输出是非常不方便的，那有没有强大的日志输出工具呢？“山重水复疑无路，柳暗花明又一村”，Log4j 就是其中一个，因此，下一章笔者将讲解 Log4j 的使用方法。Hibernate 默认是使用 Log4j 输出日志，所以在下一章读者可以看到，如果使用 Log4j，则本章的示例将会输出更多的信息，这样也便于程序的调试工作。