翰子昂实训体系前导理论教材丛书序

在政府大力支持职业教育的良好背景下，高等职业教育已成为促进经济发展、社会进步和提高劳动者就业率的一条重要途径。通过几十年的教育积累，职业院校已经形成了一整套完整的教育理论及实施体系，并拥有强大的专业化教师队伍，为我国教育长期发展奠定了坚实的人才基础。随着社会经济的飞速发展，职业教育作为国家发展的持续推动力，其重要程度已为社会各界所认知。

不可否认，职业教育体制在不断发展和完善的同时，其自身的一些局限性随着社会发展对人才需求格局的变化也逐渐显现出来。高等职业院校要办出特色，必须避免走“纯粹学历教育”的误区，要在教育思想上突破传统学历教育所形成的禁锢，从人才的培养目标出发，以先进的职业教育思想为指导，改革传统的教材教学方法体系，回归高等职业教育的本义。

职业教育中的软件职业教育更是面临新的挑战：软件产业是当今世界上发展最快的朝阳产业，并将成为二十一世纪世界经济增长和社会发展的重要推动力，软件人才是产生这种推动力的主要源泉，软件职业教育人才输出的质量与数量将对软件产业的发展影响重大。

为提高软件职业教育的人才输出的质量和数量，北京翰子昂教育科技发展有限公司把多年积累的 IT 从业经验、丰富的软件开发经验、系统的实训人才培养经验相结合，同时调研走访了上百家国内软件企业的相关岗位和人力资源负责人，分析了多家国内外职业教育机构的课程体系，最后在几十位软件领域专家和资深软件工程师、项目经理的协同配合下潜心研发出“翰子昂实训体系”，目的是让更多、更广的职业院校学生接受“软件实训”，从而提高学生整体职业素养和技术水平，在校即达到企业对软件人才的需求，缩小学生与企业之间的距离，从而实现学生从学校到企业的平稳过渡。

翰子昂实训体系前导理论教材丛书的作用对于翰子昂实训体系是不可或缺的。在进行项目实训之前学生的知识结构水平参差不齐，理论知识不够牢固，直接进行项目的开发会存在一定的知识真空区，所以理论知识的学习是非常有必要的。这套前导理论教材是在分析了大量的项目，并且总结了当前流行的开发技术特点之后归纳出的一系列基础知识理论，这些理论知识能够在将来的项目实训中得到很广泛的应用。可以说前导理论教材就是为了实训项目而生。实训过程中，牢固的基础知识能够提高项目开发的效率，所谓“工欲善其事，必先利其器”就是这个道理。

人事部中国高级公务员培训中心、教育部全国高等学校计算机教育研究会、全国信息化计算机应用技术资格认证管理中心从翰子昂实训体系前导理论教材的策划开始就表示了关注与支持，并且在丛书开发过程中给予了很多富有针对性的意见与建议。产品能够得到国家有关部门的认可，我们深感荣幸的同时也更加认识到责任的重大。

为了研发这套前导理论教材，北京翰子昂教育科技发展有限公司汇聚了多名来自知名 IT 公司的软件开发人员、系统分析设计人员、项目经理、人力资源部负责人以及全国重点职业院校的学术带头人，在此特别感谢他们给予的意见与建议，同时也感谢付出辛勤劳动的研发小组的成员，更要感谢国家有关部门的支持。

北京翰子昂教育科技发展有限公司　产品开发部

前言

Struts，Hibernet，Spring 简称 SSH，是目前 JavaEE 开发当中最为流行的框架整合方式，Ajax 异步的 JavaScript And XML，是 web2.0 跨时代的技术产物。WebService 通过面向服务的方式为我们提供了功能更为强大的技术平台，本书以 Struts，Hibernate，Spring，Ajax，Web Service 为核心详细的讲解了这五个组件的基础知识和基本使用方法，并通过一系列集成开发实例讲解了这五个组件在 JavaEE 项目中的应用。本书对每一个组件的讲解都是以入门级实例开始的，然后对组件的架构以及各方面的功能进行了深入的探讨。这样安排的好处是使学员在具备了初步知识的基础上对组件进行更深入的理解，并通过一系列贴近实际的实例来提高学员对组件的把握能力进而迅速丰富开发经验。

本书共分 6 个专题，14 个案例，通过实现不同的功能来讲解以上技术：

专题一为 Struts 篇，通过四个案例讲解了 Struts 的工作机制，以及 Struts 当中各个组件的功能和协同工作方式，利用 Struts 提供的 Validator 框架编写实现国际化的程序，并对数据库连接池进行了深入讨论。

专题二为 Hibernate 篇，通过四个案例，介绍了 Hibernate 原理，Hibernate 核心 API，以及关联映射关系，并对 Hibernate 的高级查询进行讲解。

专题三为 Spring 篇，通过两个案例深入讨论了 IoC 和 AOP 两个概念，培养学生面向接口的编程习惯。

专题四为 SSH 整合篇，介绍了 Spring，Struts 和 Hibernate 的多种整合方式，以及三个框架在项目中的地位和功能。还有一些常用的整合技巧，并提出了声明式事务管理的概念，为企业级应用提供了更好的解决方法。

专题五为 Ajax 篇，本专题通过两个案例，介绍 Ajax 的基本原理以及对 DWR 框架进行初步使用。

专题六为 Web Service 篇，本专题通过一个案例，简单介绍 Web Service 的原理和简单应用。

本书由北京翰子昂教育科技发展有限公司软件工程师实训产品开发部完成。

北京翰子昂教育科技发展有限公司　产品开发部

2008 年 6 月

目录

专题一 Struts

教学目标

掌握 Struts 基本原理

深入理解 Struts 当中的控制器组件

Struts 当中使用数据库连接池

Struts 标签及 Validator 框架

案例一 初识 Struts

1. 教学目标

1.1 使用工具怎样建立一个 Struts 工程

1.2 Struts 是怎样实现 MVC 三层架构的

1.3 理解 Struts 当中的 ActionServlet，FormBean 和 Action 的用法

1.4 Struts 相关配置文件的写法

2. 工作任务

2.1 使用 Struts 框架实现论坛登录示例

3. 相关实践知识

3.1 本书相关实例介绍

在本书中，将给出一个完整的论坛实例，该论坛的功能如下：普通网民可以访问论坛注册成为会员，该论坛会员可以登录论坛发表帖子和回复他人帖子，论坛管理员可以登录论坛后台管理论坛用户、帖子以及帖子类别。

本实例使用 MyEclipse6.0 开发，数据库采用 MySql，并最终将程序部署到 Tomcat6.0 服务器当中，实例数据库 mybbs 当中共涉及六张表，分别是 admin，answer，item，subitem，topic，userinfo，我们会在适当的章节一一向学员介绍每张表的结构。

3.2 使用 MyEclipse6.0 建立 Struts 工程

打开 MyEclipse6.0，创建步骤具体如下：

1. 首先建立一个 Web 项目，点击菜单栏【File】➔【New】➔【Web Project】，在 Project Name 当中输入项目名称“MyBBS”，然后点击【Finish】。

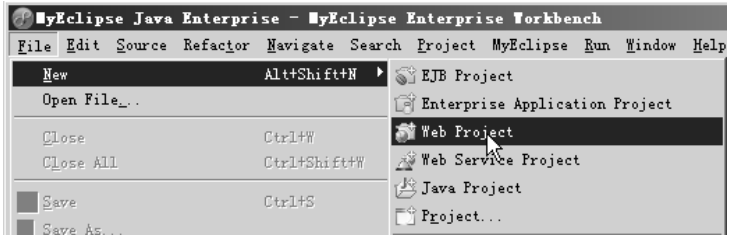


图 1-1 建立 Web 工程

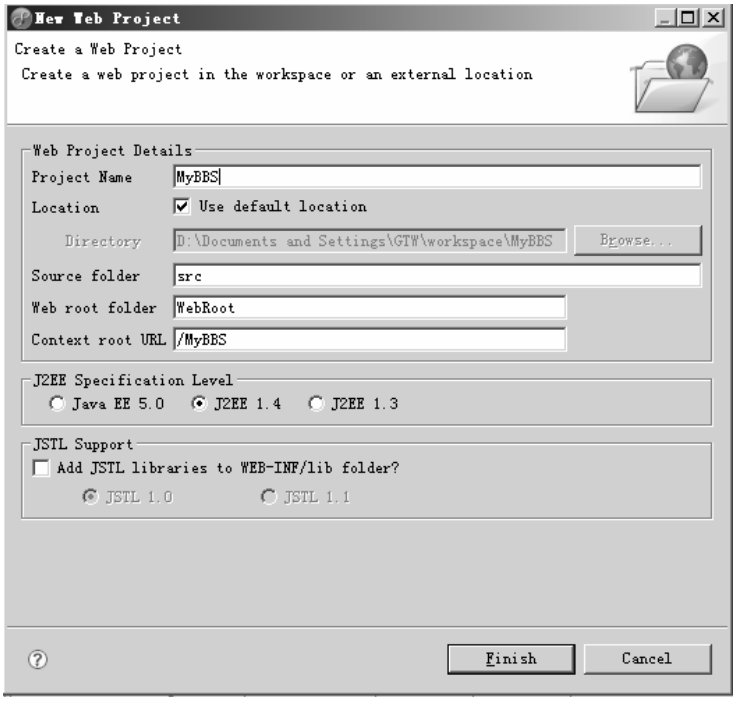


图 1-2 命名 Web 工程

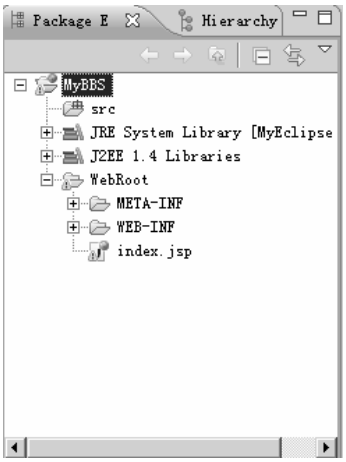


图 1-3 Web 工程结构

2. 然后在项目中加入 Struts 支持,右键点击项目根节点【MyBBS】➔【MyEclipse】➔【Add Struts Capabilities】，然后点击【Finish】。

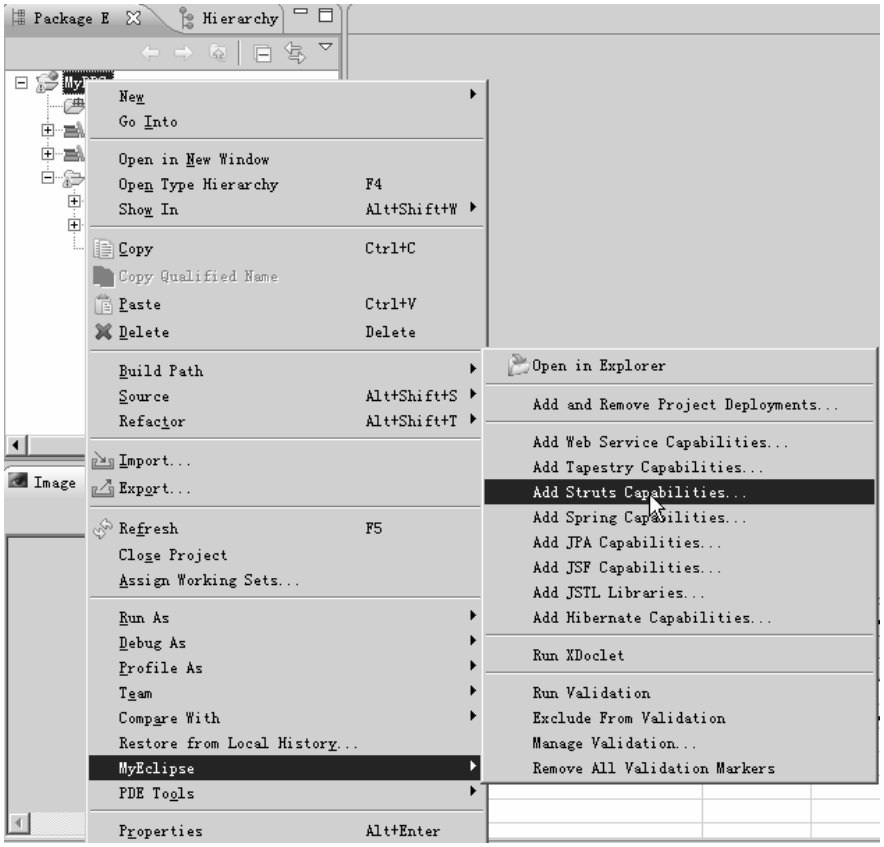


图 1-4　添加 Struts 支持

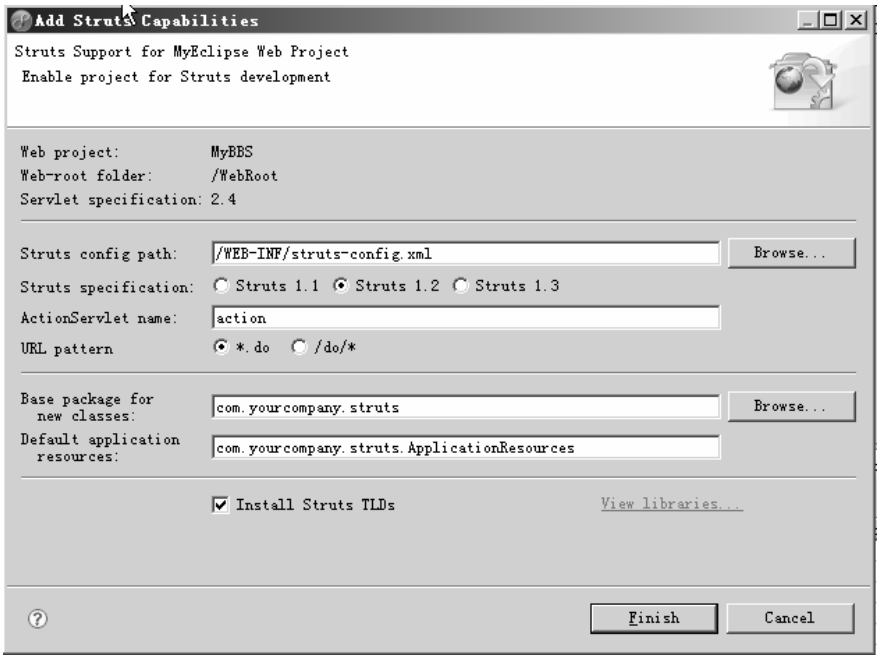


图 1-5 添加 Struts 支持

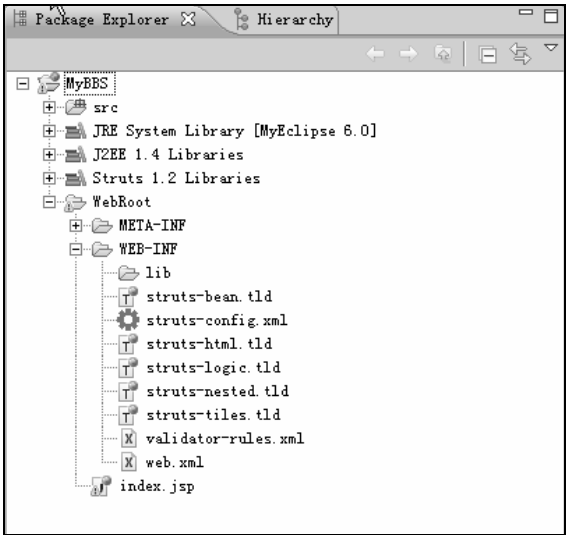


图 1-6　添加了 Struts 的 Web 工程

3. 这样，一个可支持 Struts 的工程就建立完成了我们可以看见项目中加入了 Struts 1.2 Libraries 类库，打开 web.xml 可以看到在 web-app 节点下增加了如下信息：

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
```

```
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</servlet>
```

4. 下面我们用 Struts 来实现 MyBBS 当中的论坛登录功能，此功能设计到 userinfo 表，结构如下：

字段名称	类型	大小	是否为空	备注
userid	int	4	否	主键，自增，代表人员编号
username	varchar	20	否	代表用户名
userpwd	varchar	20	否	代表用户密码
userques	varchar	50	否	代表密码找回问题
userans	varchar	50	否	代表密码找回答案
integral	int	4	否	代表用户积分
grade	varchar	20	否	代表用户等级
useremail	varchar	20	否	代表用户 Email
sex	bit	2	否	代表用户性别 1 男 0 女

5. 在 MySql 数据库中创建 mybbs 数据库和 userinfo 表的脚本如下：

```
create database mybbs;
use mybbs;
create table `mybbs`.`userinfo`(
       `userid` int not null auto_increment,
      `username` varchar(20) default '' not null,
      `userpwd` varchar(20) default '' not null,
      `userques` varchar(50) default '' not null,
```

```
      `userans` varchar(50) default '' not null,
      `integral` int not null,
      `grade` varchar(20) default '' not null,
      `useremail` varchar(20) default '' not null,
      `sex` bit,
       primary key (`userid`)
   )TYPE=INNODB,
    default character set gbk;
```

6. 实现这样的功能，我们首先需要一个 Web 页面，用以用户登录时填写登录信息，在 WebRoot 下建立 jsp 文件夹，并在其中建立 login.jsp 代码如下：

```
<form action="<%=request.getContextPath()%>/LoginAction.do">
        用户名:
        <input type="text" name="name" size="35" maxlength="35" />
        密  码:
        <input type="password" name="password" size="35"
        maxlength="35" />
        验证码:
        <input type="text" name="rand" size="20" maxlength="20">
        <IMG SRC="<%=request.getContextPath()%>/jsp/image.jsp" />
        <input type="submit" value="提交" />
        <input type="reset" value="重置" />
</form>
```

7. 建立 success.jsp 代码如下：

```
<font face="宋体" size="4">登录成功</font>
```

8. 建立 error.jsp 代码如下：

```
 <font face="宋体" size="4" color="red" >您的用户名或密码错误</font>
```

9. 建立用以生成验证码的 image.jsp 代码如下：

```
<%@ page contentType="image/jpeg" import="java.awt.*,
java.awt.image.*,java.util.*,javax.imageio.*"  pageEncoding="GB2312" %>
<%!
Color getRandColor(int fc,int bc){//给定范围获得随机颜色
Random random = new Random();
if(fc>255) fc=255;
if(bc>255) bc=255;
int r=fc+random.nextInt(bc-fc);
int g=fc+random.nextInt(bc-fc);
int b=fc+random.nextInt(bc-fc);
```

```
return new Color(r,g,b);
}
%>
<%
//设置页面不缓存
response.setHeader("Pragma","No-cache");
response.setHeader("Cache-Control","no-cache");
response.setDateHeader("Expires", 0);
//在内存中创建图象
int width=60, height=20;
BufferedImage image =
            new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
//获取图形上下文
Graphics g = image.getGraphics();
//生成随机类
Random random = new Random();
//设定背景色
g.setColor(getRandColor(200,250));
g.fillRect(0, 0, width, height);
//设定字体
g.setFont(new Font("Times New Roman",Font.PLAIN,18));
//随机产生 155 条干扰线，使图象中的认证码不易被其它程序探测到
g.setColor(getRandColor(160,200));
for (int i=0;i<155;i++)
{
int x = random.nextInt(width);
int y = random.nextInt(height);
int xl = random.nextInt(12);
int yl = random.nextInt(12);
g.drawLine(x,y,x+xl,y+yl);
}
//取随机产生的认证码(4 位数字)
String sRand="";
for (int i=0;i<4;i++){
String rand=String.valueOf(random.nextInt(10));
sRand+=rand;
//将认证码显示到图象中
g.setColor(new
Color(20+random.nextInt(110),20+random.nextInt(110),20+random.nextInt(110)
));
//调用函数出来的颜色相同，可能是因为种子太接近，所以只能直接生成
g.drawString(rand,13*i+6,16);
```

```
}
//将认证码存入 SESSION
session.setAttribute("rand",sRand);
//图象生效
g.dispose();
//输出图象到页面
ImageIO.write(image, "JPEG", response.getOutputStream());
out.clear();
out = pageContext.pushBody();
%>
```

10. 新建 com.mybbs.pojo 包并在包下建立叫做 Userinfo 的 pojo 类，其中包括如下字段：

```
  //用户 ID
    private int userid;
    //用户名
    private String username;
    //密码
    private String userpwd;
    //密码找回问题
    private String userques;
    //密码找回答案
    private String userans;
    //用户积分
    private int ntegral;
    //用户等级
    private String grade;
    //用户 Email
    private String useremail;
    //性别
    private byte sex;
```

并用 MyEclipse 生成访问器。

11. 新建 com.mybbs.dao 包并在包下建立 JdbcDao.java 和 UserinfoDao.java 类用于访问数据库，分别在类中，键入如下代码：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
public class JdbcDao {
    //数据库连接对象
    protected Connection  conn=null;
    //定义 Statement 变量
    protected Statement  stmt=null;
    //定义 PreparedStatement 变量
    protected PreparedStatement pstmt=null;
    //结果集
    protected ResultSet  rs=null;
    //MySql 数据库驱动
    Private  String  driverClass_Name="com.mysql.jdbc.Driver";
    //MySql 数据库 url
    Private String
         url="jdbc:mysql://localhost:3306/mybbs?user=root&password=root";
    //默认的构造函数
    public JdbcDao(){}
    //得到数据库连接
    public void getConnection()
    {
        Connection conn=null;
        try
        {
            //JDBC 连接
            Class.forName(driverClass_Name);
            conn=DriverManager.getConnection(url);
            this.conn=conn;
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }
    }
    //关闭数据库连接
    public void closeConnction()
    {
        if(rs!=null)
        {
            try
```

```
            {
                rs.close();
            }
            catch(SQLException  ex)
            {
                ex.printStackTrace();
            }
        }
        if(stmt!=null)
        {
            try
            {
                stmt.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
        }
        if(pstmt!=null)
        {
            try
            {
                pstmt.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
        }
        if(conn!=null)
        {
            try
            {
                conn.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
        }
    }
```

```
}
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.ArrayList;
import com.mybbs.pojo.Userinfo;
public class UserinfoDao extends JdbcDao {
    //定义自身的一个变量
    private static UserinfoDao userinfoDao = null;
    //定义自身私有的构造函数
    private UserinfoDao() {
    };
    //通过一个简单的单利模式将 UserinfoDao 返回到客户端
    public synchronized static UserinfoDao getUserinfoDao() {
        if (userinfoDao != null) {
            return userinfoDao;
        }
        userinfoDao = new UserinfoDao();
        return userinfoDao;
    }
    //得到用户信息，以 list 的形式返回到客户端
    public List<Userinfo> getPerson(Userinfo user) {
        //声明用以存放用户信息的集合
        List<Userinfo> list = new ArrayList();
        //得到数据库连接
        super.getConnection();
        //定义要执行的 sql 语句
    String sqlString = "select * from userinfo where username=? and
userpwd=?";
        try {
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            //传入参数
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, user.getUserpwd());
            //执行 sql 并返回结果集
            rs = pstmt.executeQuery();
            //循环遍历结果集
            if (rs.next()) {
                Userinfo myuser = new Userinfo();
```

```
                myuser.setUsername(rs.getString(2));
                myuser.setUserpwd(rs.getString(3));
                list.add(myuser);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
        return list;
    }
}
```

12. 新建 com.mybbs.service 包并在包下建立 UserinfoService.java 类，判断用户登陆，键入如下代码：

```
import com.mybbs.pojo.Userinfo;
import com.mybbs.dao.UserinfoDao;
import java.util.List;
import java.util.Iterator;
public class UserinfoService {
    //得到操作用户信息的 DAO
    private UserinfoDao userinfoDao=UserinfoDao.getUserinfoDao();
    //验证用户登录信息，用户名密码正确返回 true
    public boolean userLogin(Userinfo user)
    {
         boolean isLogin=false;
         //根据传入的 user 对象从数据库中查询结果集
         List list=userinfoDao.getPerson(user);
         Iterator it=list.iterator();
         //如果集合中有数据说明用户可以登录
         if(it.hasNext())
         {
             isLogin=true;
         }
         return isLogin;
    }
}
```

13. 新建 com.mybbs.actionform 包并在包下建立 LoginActionform.java 类，键入如下代码：

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
```

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

public class LoginActionform extends ActionForm {
    //用户密码
    private String password;
    //用户名
    private String name;
    //验证码
    private String rand;
    public String getRand() {
        return rand;
    }
    public void setRand(String rand) {
        this.rand = rand;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //actioform 特有的验证方法
    public ActionErrors validate(ActionMapping mapping,HttpServletRequest
request)
    {
        return null;
    }
    //actionform 特有的重置方法
    public void reset(ActionMapping mapping,HttpServletRequest request)
    {
    }
}
```

14. 新建 com.mybbs.action 包并在包下建立 LoginAction.java 类，键入如下代码：

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.LoginActionform;
import com.mybbs.pojo.Userinfo;
import com.mybbs.service.UserinfoService;
import javax.servlet.http.HttpSession;
public class LoginAction extends Action {
    String scriptStr=null;
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        //得到 LoginActionForm 对象
        LoginActionform LoginActionForm = (LoginActionform) form;
        //得到 session 对象
        HttpSession session=request.getSession();
        //判断验证码是否输入正确

    if(!session.getAttribute("rand").toString().equals(LoginActionForm.ge
tRand()))
        {
            //页面跳转
            return mapping.findForward("self");
        }
        //实例化用户 bean
        Userinfo user=new Userinfo();
        //得到用户输入的用户名
        user.setUsername(LoginActionForm.getName());
        //得到用户输入密码
        user.setUserpwd(LoginActionForm.getPassword());
        //声明业务逻辑对象
        UserinfoService userinfoService=new UserinfoService();
        //如果用户名和密码输入正确登录成功
        if(userinfoService.userLogin(user))
        {
            return mapping.findForward("success");
        }
        //如果输入错误提示错误
        else
        {
            return mapping.findForward("error");
```

```
        }
    }
}
```

15. 打开 WEB-INF 当中的，struts-config.xml 文件，在其中键入：

```
<struts-config>
  <form-beans >
     <form-bean name="LoginActionForm"
     type="com.mybbs.actionform.LoginActionform" />
     </form-beans>
  <action-mappings >
   <action
     input="/jsp/login.jsp" attribute="LoginActionForm"
     name="LoginActionForm" path="/LoginAction"
     scope="request" type="com.mybbs.action.LoginAction" >
           <forward name="self" path="/jsp/login.jsp"></forward>
           <forward name="success" path="/jsp/success.jsp"></forward>
           <forward name="error" path="/jsp/error.jsp"></forward>
    </action>
   </action-mappings>
</struts-config>
```

16. 部署运行程序，在地址栏里输入 http://localhost:8080/mybbs/jsp/login.jsp 打开登录页面。

图 1-7 登录页面

17. 输入正确的用户名，密码和验证码后点击提交，提示登录成功。

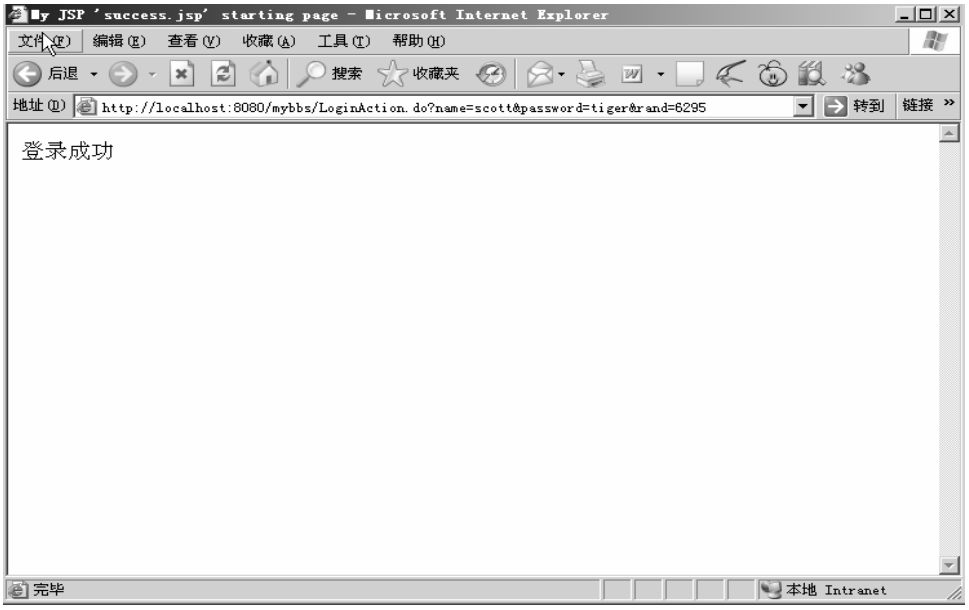


图 1-8 提示登录成功

18. 验证码、用户名或密码输入错误将在登录页面提示用户。

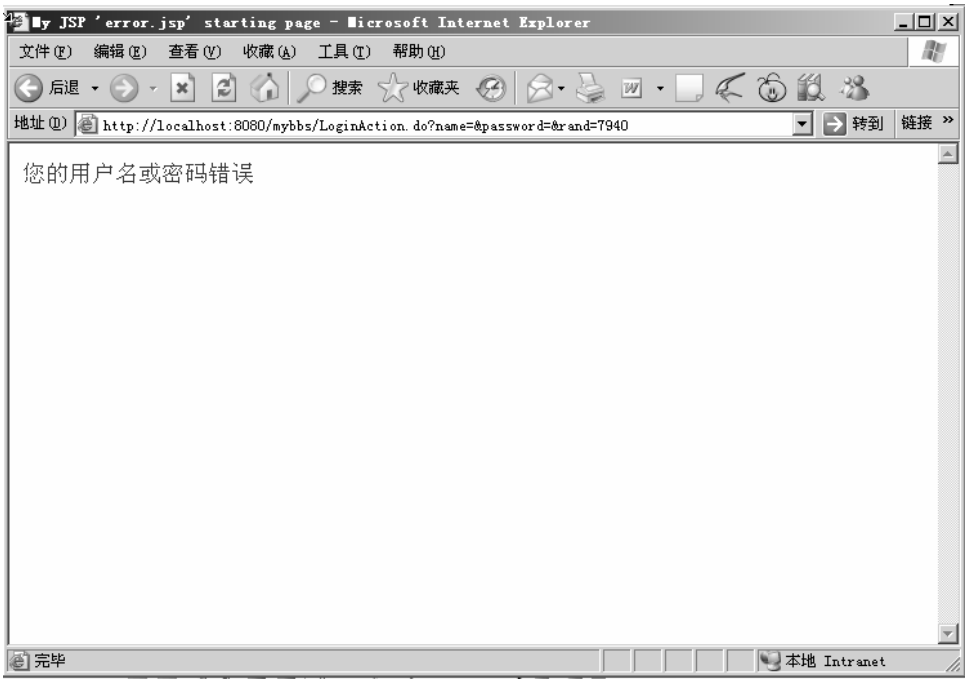


图 1-9 提示用户名密码错误

3.3 使用 MyEclipse6.0 加速 Struts 开发

我们可以利用 MyEclipse 提供的工具来简化 Struts 的开发。我们现在用工具来建立 LoginActionform.java，LoginAction.java 和相关的 struts-config.xml。

1. 在建立完 MyBBS 工程，并在工程中添加了 Struts 支持后，点击菜单栏【File】➔【New】➔【Other】。

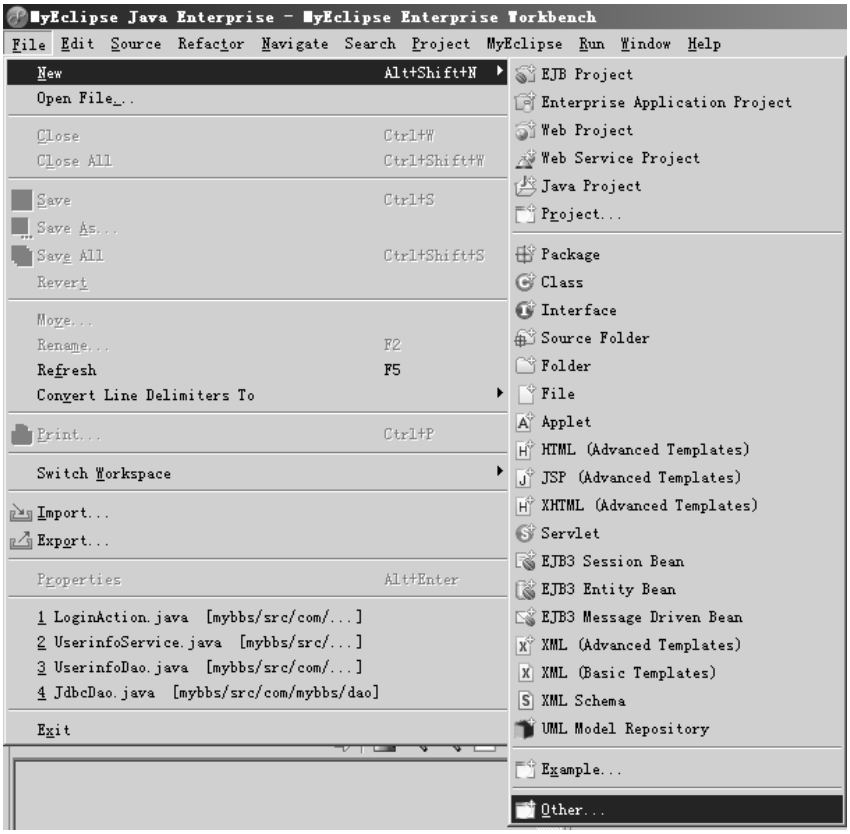


图 1-10 利用工具建立 struts

2. 选中【Web-Struts】➔【Struts1.2】 ➔【Struts1.2 Form,Action & JSP】。

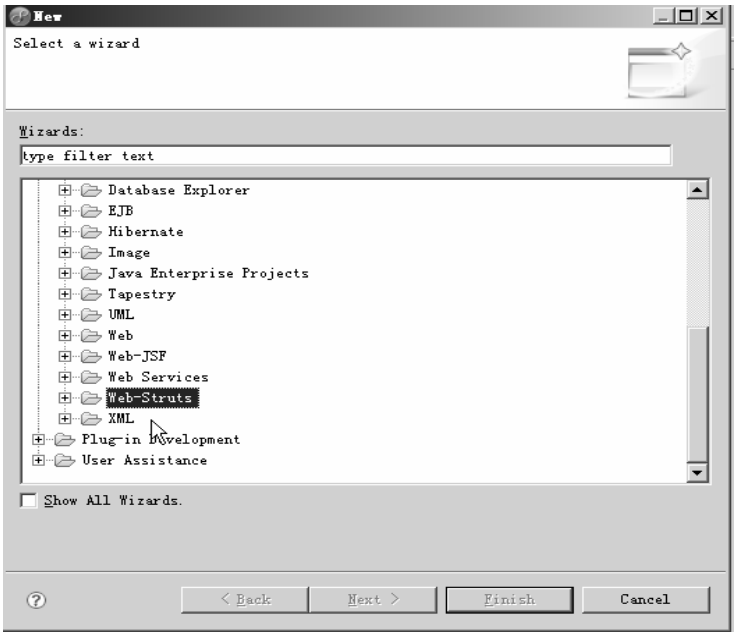


图 1-11 工具建立 Struts

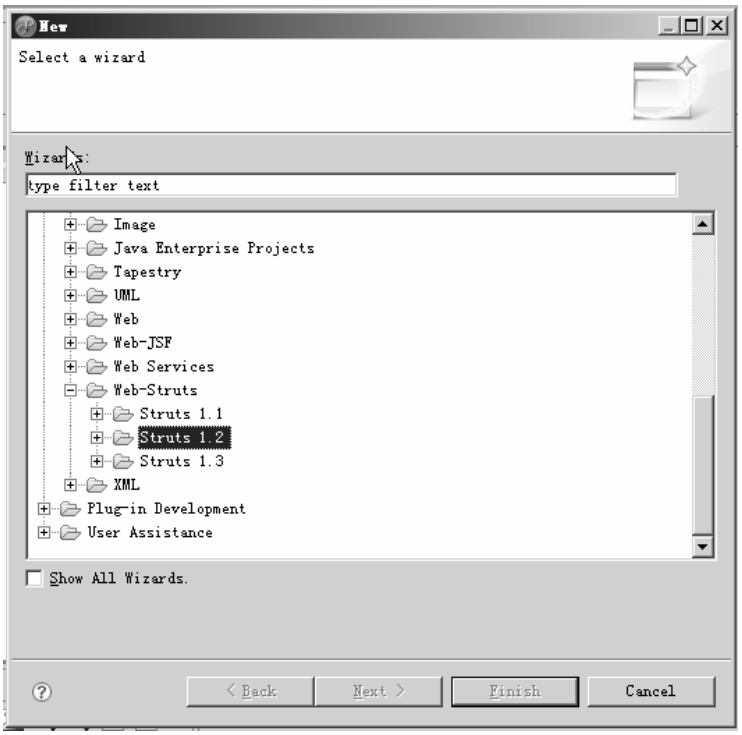


图 1-12 工具建立 Struts

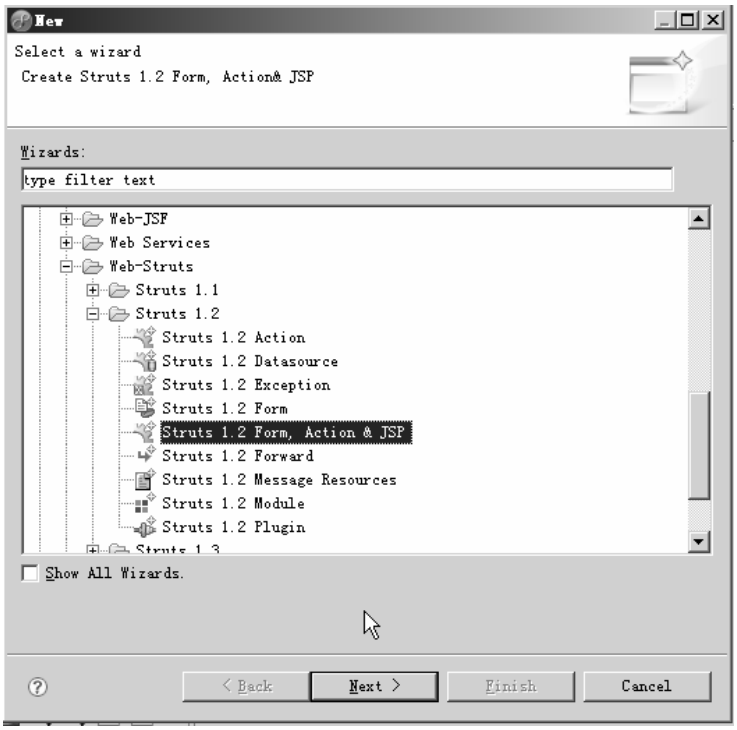


图 1-13 工具建立 Struts

3. 点击打开后填写 FormBean 的信息，Name 中填写“LoginActionForm”，SuperClass 选中“org.apache.struts.action.ActionForm”，Form Type 中填写“com.mybbs.struts.actionfrom .LoginActionForm”。点击【add】为 LoginActionForm 添加“name”和“password”属性。

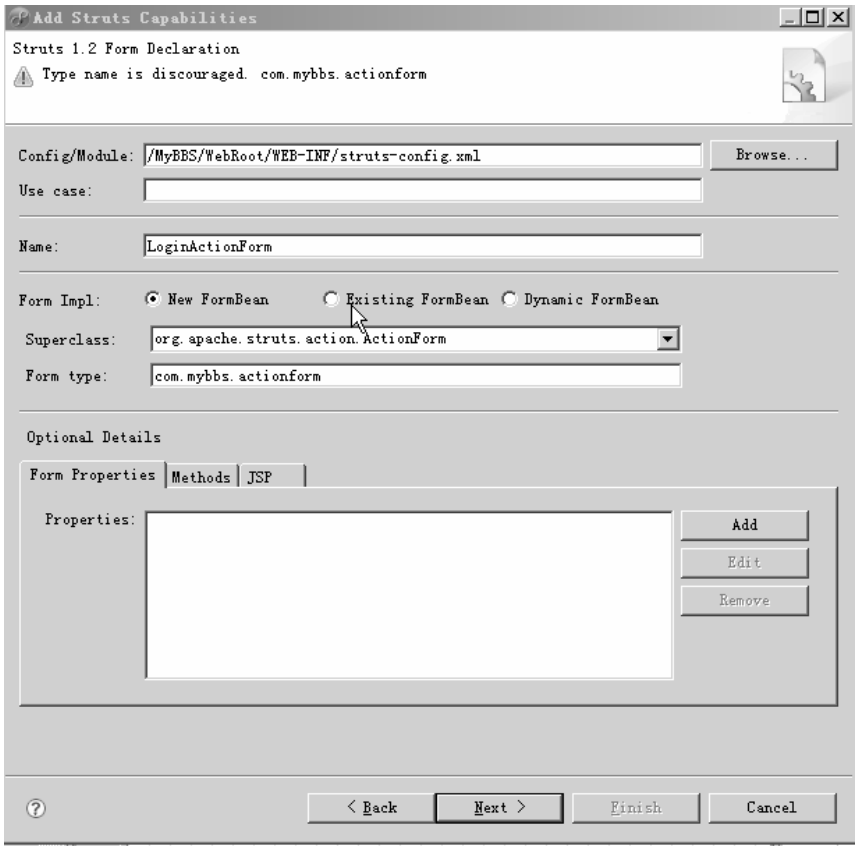


图 1-14 工具建立 Struts

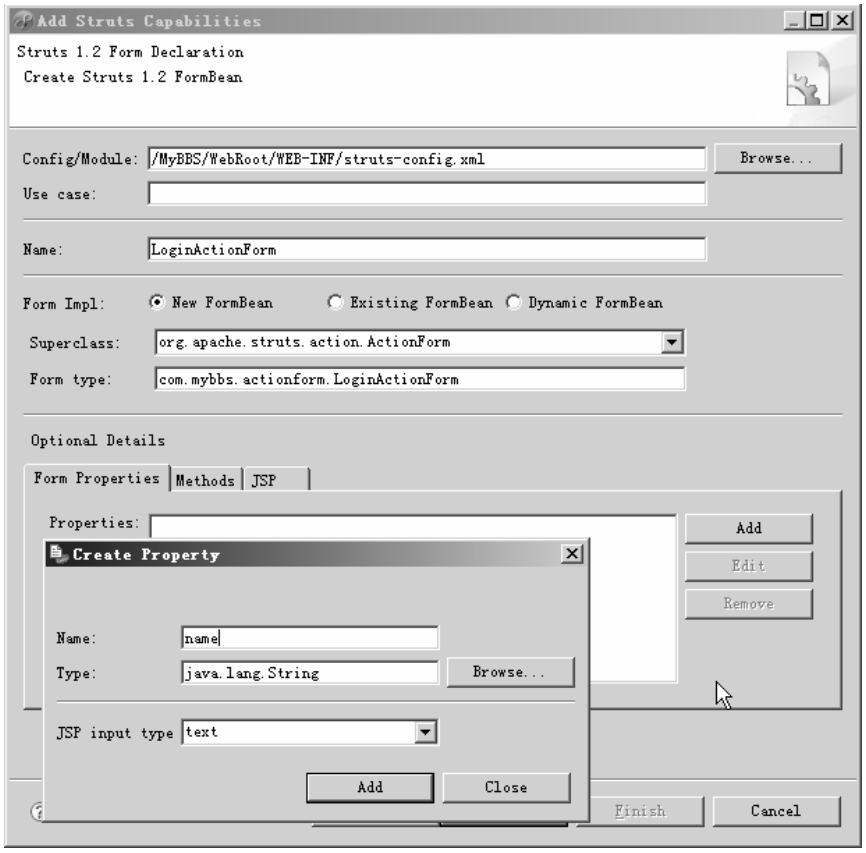


图 1-15 工具建立 Struts

4. 点击【Next】填写 action 的信息，path 填写“/LoginAction,superclass”，Superclass 选中”org.apahce.struts.action.Action”，Type 填写“com.mybbs.action.LoginAction”。Name 中填写”LoginActionForm”。Attribute 和 scope 使用工具生成的默认值，点击【Finish】，这样 LoginAtion,LoginActionForm 以及相关的 struts-config.xml 就算生成了。

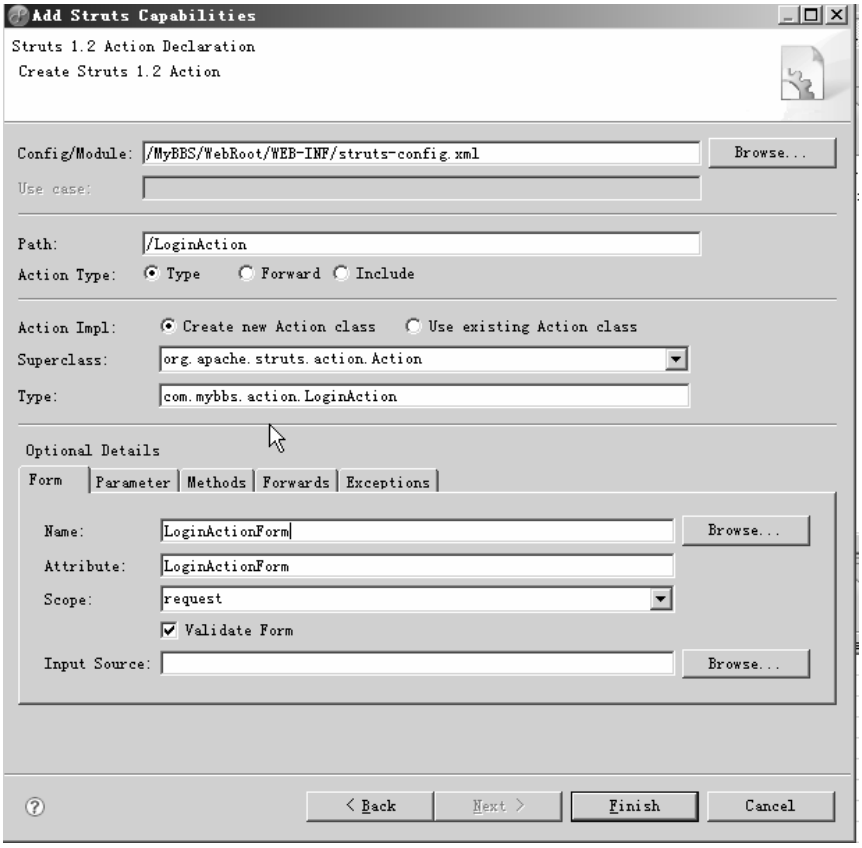


图 1-16 工具建立 Struts

4. 相关理论知识

4.1 Struts 概念

Struts 的含义是“支柱，枝干”，来源于在建筑和旧式飞机中使用的金属支架，它的目的是为了减少程序开发的时间，项目的创建者认为 JSP，Servlet 的存在虽然可以帮助用户解决大部分的问题，但是由于它们的编码对项目的开发带来了很多的不便，可重用性也差，所以提出 Struts 这个框架，帮助用户在最短的时间内解决这些问题。Struts 框架提供如下的服务：

作为控制器层的 Servlet。

提供大量的标签库。

提供了用于国际化的框架，利用不同的配置文件，可以帮助用户选择合适自己的语言。

提供了 JDBC 的实现，来定义数据源和数据库连接池。

XML 语法分析的工具。

文件下载机制。

开始的时候，Struts 仅被作为开发包发布，经过不断的扩充，现在的 Struts 已经是内容相当完整的框架了。

4.2 Struts 与 Model2 的关系

在 Java Web 发展过程中的应用架构 Model1 和 Model2，而 Struts 就是基于 Model2 的架构产生的。Struts 的 3 层模型如下：

在视图层，除了可以使用 JSP 及其标签库以外，还提供了一个强大的 Struts 标签库，来帮助用户解决显示逻辑，并且利用 ActionForm 组件将信息递交到控制器层。

在控制器层，Struts 提供了一个控制器组件 ActionServlet，它继承自 HttpServlet，并重载了 HttpServlet 的 doGet()，doPost()方法，可以接受 HTTP 的响应，并进行转发，同时还提供了使用 XML 进行转发 Mapping（映射）的功能。

在模型层，Struts 提供 Action 对象，来管理业务逻辑的调用，帮助用户分离业务逻辑，也就是说 Struts 本身不实现业务逻辑，但可以调用已完成的业务逻辑。

4.3 Web 应用中 Struts 所处 MVC 的位置

在 MVC 模型中，可以将 Struts 大致分割成如图

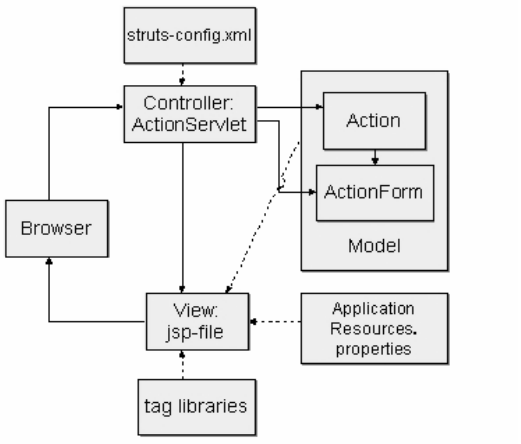


图 1-17 Struts MVC 图

Struts 标签库：处于视图层，用来辅助 JSP 展现页面，类似 JSP 自带标签库。

ActionForm 对象：视图层和控制器层之间的接口，可以看成是一个值对象（Value Object，VO）提供 get()，set()方法，在 Struts 中这个对象还提供了校验数据的方法。

ActionServlet 控制器组件：在控制层中控制页面流转，并调用模型层，继承自 HttpServlet。

Action 对象：在控制层和模型层进行交互，该对象被 ActionServlet 组件调用。可以在 Action 中调用业务逻辑，并将页面回复给 ActionServlet 组件，它是控制层和模型层之间的桥梁。

Struts 通过配置文件将上面这些组件和对象关联起来一起工作。在 Web.xml 中，应该将 Struts 提供的 ActionServlet 配置成一个 Servlet，接着指定 Struts 自身的配置文件 struts-config.xml，这个文件在 Servlet 中被调用解析。Struts 是靠 XML 的配置来完成各层之间的运行。

批注 [w1]: 删除一句话

4.4 Struts 的中心控制类 ActionServlet

ActionServlet 来自于 org.apache.struts.action 包，它继承自 HttpServlet，作为 Struts 的 Servlet 控制器，是 Struts 框架控制器的核心。Web 程序初始化时，会读取 struts-config.xml 中的内容作为它的环境上下文，所有客户端的请求都会通过 ActionServlet 开始整个流程,请求到来时无论 doGet 还是 doPost 方法都会调用 process 方法，ActionServlet 的 process 方法十分重要，它调用 getModuleConfig 得到配置上下文，将其传入 getRequestProcessor 方法，再由 getRequestProcessor 来返回 RequestProcessor 对象，并调用 RequestProcessor 的 process 方法来完成整个工作流程。对于 ActionServlet 来说，程序员完全可以继承它，并覆盖其中的某些方法来完成自己需要的全新的 ActionServlet。

下面是一段 ActionServlet 的代码：

```
public void doGet(HttpServletRequest request,HttpServletResponse
response)throws IOException,ServletException
{
    process(request,response);
}
public void doPost (HttpServletRquest request,HttpServletResponse
response)throws IOException,ServletException
{
    process(request,response);
}
protected void process(HttpServletRequest request,HttpServletResponse
response)throws IOException,ServletException
{
    RequestUtils.selectModule(request,getServletContext());
getRequestProcessor(getModuleConfig(request)).process(request,response);
}
```

在 web.xml 当中我们将 ActionServlet 配置成一个普通的 Servlet，通过 url-pattern 的设置后，如果有“*.do”的请求，就会调用该 ActionServlet。

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet
   </servlet-class>
   <init-param>
     <param-name>config</param-name>
     <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <init-param>
     <param-name>debug</param-name>
     <param-value>3</param-value>
   </init-param>
```

```
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</servlet>
```

4.5 Struts 当中的 FormBean

ActionForm 是 Struts 框架中的一个重要组成部分，它保存 request 参数。ActionForm 从本质上讲，就是一个 JavaBean，这些 Bean 有与 HTTP request 参数相对应的属性名称，控制器根据 struts-config.xml 文件创建 ActionForm Bean 的实例。并根据配置，把实例传给 Action 类，ActionForm Bean 的实例也被称为表单。ActionForm Bean，必须继承 org.apache.struts.action.ActionForm 类。当 ActionServlet 调用 Action 类时，它创建生成相应的 ActionForm Bean 实例，然后把它传递个 Action 类。ActionForm Bean 中会包含许多 get/set 方法，set 方法的调用是由 Struts 框架自动完成的，请求开始处理时，Struts 框架从 request 对象中提取 url 参数，并自动使用这些参数为 FormBean 中对应的属性赋值。

validate 方法可以对 ActionFrom Bean 中的内容进行验证，原形如下：

```
public ActionErrors validate(ActionMapping mapping,ServletRequest request)
public ActionErrors validate(ActionMapping mapping,HttpServletRequest
request)
```

在 ActionFrom Bean 中重载 validate 方法后，并不会自动调用，要想 Struts 框架得到 request 参数后，自动调用 validate 方法，必须在 struts-config.xml 中，把使用该 ActionForm Bean 的 <action>元素的 validate 属性设置为 true。

在 validate 方法验证出错误后，需要产生一个 ActionError 的实例，ActionErrors 中会有 0 个或多个 ActionError 实例。validate 方法返回 ActionErrors，如果 ActionErrors 中 ActionError 的实例多于 0，那么就说明 validate 方法验证到了一个错误，程序将不在进入 Action，而会根据<action>元素的 input 属性值，返回指定的页面。如下例我们可以用 validate 方法验证 name 属性是否为空。

```
public ActionErrors validate(ActionMapping mapping,ServletRequest request)
{
    ActionErrors errors=new ActionErrors();
     if(name.equals(“”)
    {
        ActionError error=new ActionError(“error.errorname”);
        errors.add(“noname”,error);
```

```
    }
}
```

本例当中的 error.errorname 可在资源文件当中找到用于读取错误信息，并通过 Struts 的自带标签<html:error property=“noname”/>显示给客户端。有关资源文件和 Struts 标签的用法本书会在后续章节为学员介绍。

Reset 方法可以对 ActionFrom Bean 中的内容进行复位，原形如下：

```
public void reset(ActionMapping mapping,ServletRequest request)
public void reset(ActionMapping mapping,HttpServletRequest request)
```

使用 reset 方法，主要考虑 ActionFrom bean 的作用范围是什么，<action>元素的 scope 属性指定了 ActionForm 的作用范围，两种作用范围分别为“request”和“session”

当 scope 属性被指定为“request”，也即被写成“scope=request”时，每次调用 ActionForm Bean 都会生成一个新的实例。因此 ActionServlet 调用的 ActionForm Bean 不会受到其他的影响，每次提交或刷新页面都会得到一个新的 ActionForm Bean。

当 scope 属性指定了为“session”，也即被写成“scope=session”的时候，由于在一次生命周期中本 ActionForm Bean 只创建一次，其他的调用都会使用第一次创建的 Action Form Bean。也就是说，当 ActionForm Bean 中的变量存在后，如果不被改变，任何时候调用该 ActionForm Bean 都会得到第一次创建的值，这样的特性在跨页面收集数据时十分有效。但这样有可能对该 ActionFrom Bean 中的部分数据值进行修改，在“scope=session”的时候，那些没有被修改的值就会沿用之前的值，这样 ActionForm Bean 将成为一个新旧变量的混合体。

这时，就可以使用 reset 方法来解决问题。再次为 ActionForm Bean 的变量赋值前，调用 reset 方法使其复原，这个工作是 Struts 框架自动进行的。如下例我们可以用 reset 方法复位 name 的值。

```
public void reset(ActionMapping mapping,HttpServletRequest request)
{
    Name="";
}
```

4.6 Struts 当中的 Action 类

Struts 中的 Action 类，必须继承 Struts 提供的类 org.apache.action.Action.Actoin 类的目标是处理一个请求，Struts 框架为每一个 Action 类只创建一个实例，因为所有的用户都使用这一个实例，所以 Action 类被运行在一个多线程的环境中。

ActionServlet 调用 Action 类，并调用 exectue 方法，

Action 类的 execute 方法声明如下：

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm    actionForm,
                             ServletRequest request,
                             ServletResponse response
                               )
```

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm    actionForm,
                             HttpServletRequest request,
                             HttpServletResponse response
                               )
```

exectue 方法有四个参数：

1． ServletRequest 参数与 ServletResponse 参数， Action 类可以用它读取和设置 request 和 response 属性。

2． ActionForm 参数，用于获得请求中的数据。

3．ActionMapping 参数，这个参数在 execute 方法中有很大的作。其具体类型为 org.apache.struts.action.ActionMapping。给对象封装了<action>元素当中的所有信息，通过 ActionMapping 的各个方法，<action>元素当中的所有属性都可以动态修改。

Action 类的 execute 方法的处理流程是：从 actionForm 中得到请求的数据信息。如果有特殊的需求，也可以利用 request 对象。调用业务逻辑对象。分析业务逻辑对象的调用结果，按 ActionMapping 是的映射信息生成响应。

exectue 方法的返回值是一个 ActionForward 对象。ActionForward 对象可以生成影响信息，ActionMapping 对象的 findForward()方法，可以得到 ActionForward 对象：

```
public ActionForward findForward(String name)
```

ActionMapping 类还提供了一个方法，该方法查找所有与该操作相关的本地转发：

```
public ActionForward[] findForward()
```

ActionMapping 类也提供了一个给映射动态增加 ActionForward 的方法：

```
public void addForward(ActionForward forward)
```

execute 的返回值为 ActionForward 是用来导航的组件，它由 org.apache.struts.ActionForward 来实现。它的功能是当 execute 方法运行完后，ActionServlet 根据 ActionMapping 将信息转发到适当的地方：

```
return  mapping.findForward(“success”)
```

4.7 Struts 的核心配置文件 struts-config.xml

作为 Struts 框架的核心描述符，struts-config.xml 可以说是“一切尽在掌握”。它不但描述了 MVC 模型，定义所有视图层和控制层之间的接口（ActionForm），与控制层和模型层的接口（Action）进行结合，而且可以定义一些附加组件，如国际化信息的资源配置（Resources）文件，标签库信息以及验证框架等。

简单的说，struts-config.xml 就是将所有动作的映射用一个 XML 反应出来。要真正理解 S

truts 框架，就必须理解 struts-config.xml。

struts-config.xml 由许多标签组成，由于 XML 的标签通常被称作为元素，所以我们将 XML 的标签成为元素，而这些元素都会在 ActionServlet 组件中被解析。

配置视图层和控制层的接口：form-beans 元素

```
<form-beans>
          <form-bean name=”LoginForm” type=”com.mybbs.actionform.
            LoginActionForm”>
          </form-bean>
</form-beans>
```

在 form-beans 元素当中可以定义 0 个或 1 个以上的 form-bean 元素，每个 form-bean 被认为是一个 ActionForm 对象，name 属性指定 form-bean 元素的名称，type 属性指定其类名和路径。本例定义了 1 个 ActionForm 对象 LoginForm。这个对象的具体类都在 com.mybbs.actionform 路径下。

配置全局处理：global-exceptions 元素

该元素配置全局异常处理，其子元素 exception 定义了异常的详细信息。

```
<global-exceptions>
    <exception key=”exceptionid”
               type=”java.io.IOException”
               scope=”session”
               path=”/error.jsp” />
</global-exceptions>
```

在 global-exceptions 元素中可以定义 0 个或者 1 个以上的 exception 元素，exception 元素的 key 属性标识了一个类似“exceptionid=’出错了’”这样的报错信息，这样的信息会被定义在程序的资源配置文件中（资源配置文件也会被定义在 struts-config.xml 中，将在后面的章节进行介绍），type 属性代表的是怎样的一个异常。

本实例的含义是，当发生了 java.io.IOException 的异常后，在 error.jsp 中可以显示从资源配置文件查找到的出错信息。

声明全局转发关系：global-forwards 元素

global-forwards 元素用来声明全局的转发关系，其子元素 forward 负责将请求映射到具体的映射。

```
<global-forwards>
    <forward name=”success” path=”/Sampleaction1” />
</global-forwars>
```

在 global-forwards 元素中可以定义 0 个或 1 个以上的 forward 元素，每个 forward 元素接收 name 属性定义的请求，并转发到 path 属性定义的另一个请求中去。

本实例定义的 forward 元素，将请求 success 转发到“/ Sampleaction1”。

设置映射:action-mappings 元素

action-mappings 元素用来包含零到多个 action，其子元素 action 负责具体映射的详细信息。

```
<action-mappings>
    <action path="/LoginAction"
            type="com.mybbs.action.LoginAction"
            name="LoginForm"
            parameter="method"
            scope="request"
            validate="false">
            <forward name="success" path="/success.jsp" />
            </action>
</action-mappings>
```

在 action-mapping 元素中可以定义 0 个或 1 个以上的 action 元素。每个 action 元素接受 path 属性定义的请求，并映射到 type 属性所定义的具体 action 对象。在映射过程中，将 name 属性定义的 actionform 一并传过去，它有如下的属性：

Parameter，scope 两个属性指定了传送方式和范围，scope 常用的值有两个“session”和“request”。

validate 属性指定了是否需要 actionform 的验证。

forward 元素与 global-forwards 元素中的 forward 元素的功能一样。

本例中将请求“/LoginAction”映射到是 com.mybbs.action.LoginAction 类实例的 action 对象，将“LoginForm”作为 actionform 对象，并将其传递到自身的 action 对象。这个 actionform 不需要验证，action 对象以 forward 元素定义的“success” 作为标始，转发到“/success.jsp 页面”。

配置控制器：controller 元素

controller 元素用于配置 struts 框架的控制器。

```
<controller contentType="text/html;charset=GB2312" local="true"
processorClass="MyRequestProcessor" />
```

controller 元素有许多的属性，这些属性描述如下：

className 属性指定了和<controller>元素对应的配置类，默认为 org.apache.struts.config.ControllerConfig。

contentType 指定响应结果的类型和字符编码，该属性为可选项，默认值为 text/html。如果在 action 和 jsp 网页也设置了内容类型和字符编码，将会覆盖该设置。

local 属性决定是否把 local 对象保存到当前用户的 session 中，默认为 false。

processorClass 属性指定负责处理请求的 Java 类的完整类名，默认为 org.apache.struts.action.RequestProcessor。如果要使用自定义的类，那么自定义的类应该继承 org.apache.struts.action.RequestProcessor 类。

buffersize 属性指定上传文件的输入缓冲的大小，该属性可选，默认为 4096。

tempDir 属性指定文件上传时的临时目录，该属性可选，默认采用 Servlet 容器为 Web 应用分配的工作目录。

nocache 属性的功能是，是否为响应结果加入参数，默认情况为 false，不加入头参数。若为 true，在响应的结果中加入特定的头参数：Pragma，Cache-Control，Expires 可以防止页面被储存在客户浏览器的缓存中。

本例配置了一个 ActionServlet 的辅助功能，该响应结果和编码被指定为“text/html;charset=GB2312”，并且 Locale 被保存到 session 中，这点在国际化中时常被用到。响应请求的类已经被定义为“MyRequestProcessor”，这个类是一个自定义类，很明显该类继承自 org.apache.struts.action.RequestProcessor。

设置资源配置文件的路径：message-resources 元素

message-resources 元素被用来设定资源配置文件的路径，其 parameter 属性指定了路径和名称，其 key 属性可以帮助程序指定那个文件被调用，当只有一个配置文件时，可以不必指定 key 属性。

```
<message-resources                                          key="resources1"
parameter="com.mybbs.struts.ApplicationResources" />
```

在 message-resources 元素中定义了 key 为“resources1”的文件，该文件的具体路径在“com.mybbs.struts”目录下，名为 ApplicationResources。在这里，因为只有一个资源配置文件，所以 key 可以不必指定。

配置插件：plug-in 元素

plug-in 元素用来配置插件，在不需要插件的项目中，该元素可以不必指定。

```
<plug-in className="org.apache.struts.titles.TilesPlugin">
    <set-property property="definitions-config"
    value="/WEB-INF/titles-defs.xml" />
    <set-property property="definitions-parser-details"
    value="2" />
    <set-property property="definitions-parser-validate"
   value="true" />
</plug-in>
```

className 属性提供了实现 org.apache.struts.action.PlugIn 接口的 TilesPlugin 类，表示插件的主类。插件的参数配置使用 set-property 子元素。

本例是对于 Tiles 插件的配置。className 指定了插件的主类为“org.apache.struts.tiles.TilesPlugin”，很明显这个类实现了 org.apache.struts.action.Plugin interface 接口。该类需要 4 个配置参数，都通过 set-property 子元素设定。

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 mybbs 数据库及 userinfo 表和 MyBBS 工程。（10 分钟）
2. 编写 login.jsp，success.jsp，error.jsp 页面。（15 分钟）
3. 编写用以生成验证码的 image.jsp。（15 分钟）
4. 编写用于数据访问的 JdbcDao.java，UserinfoDao.java 类。（10 分钟）
5. 编写业务逻辑类 UserinfoService。（10 分钟）
6. 编写 LoginAction 和 LoginActionForm 类。（20 分钟）
7. 配置 struts-config.xml 文件，并调试运行程序。（10 分钟）

6. 作业

利用 Struts 框架实现登录功能，并将登录者信息显示在登录成功页面当中。

案例二 深入 Struts

1. 教学目标

1.1 使用 Struts 内置的控制器组件 DispatchAction

1.2 了解其他相关控制器组件的应用

1.3 Struts 当中的多模块编程

2. 工作任务

2.1 使用 Struts 框架实现论坛版块的增删查改

3. 相关实践知识

3.1 本书相关实例介绍

1. 本章我们利用 Struts 内置的控制器组件，实现 MyBBS 论坛后台程序当中对论坛版块的增删查改，本实例用到了 mybbs 数据库当中的 item 表，其结构如下：

字段名称	类型	大小	是否为空	备注
itemid	int	4	否	主键，自增，代表论坛版块 id
itemname	varchar	50	否	代表版块名称
itemcode	varchar	20	否	代表版块编码

其创建脚本如下：

```
use mybbs;
create table `mybbs`.`item`(
       `itemid` int not null auto_increment,
      `itemname` varchar(50) default '' not null,
      `itemcode` varchar(50) default '' not null,
       primary key (`itemid`)
   )TYPE=INNODB,
    default character set gbk;
```

2. MyBBS 后台程序页面，本书采用的是当下比较流行的 CSS+DIV 布局，以便学员课下学习，但由于 CSS 不是本书所研究的重点，所以本实例所用到的.css 文件和静态页面，我们会以课件的形式随本书发放到学员手中。

3.2 使用 MyEclipse6.0 实现实例功能

1. 打开 MyEclipse6.0，如上章所述建立 Web 工程，并添加 Struts 支持。

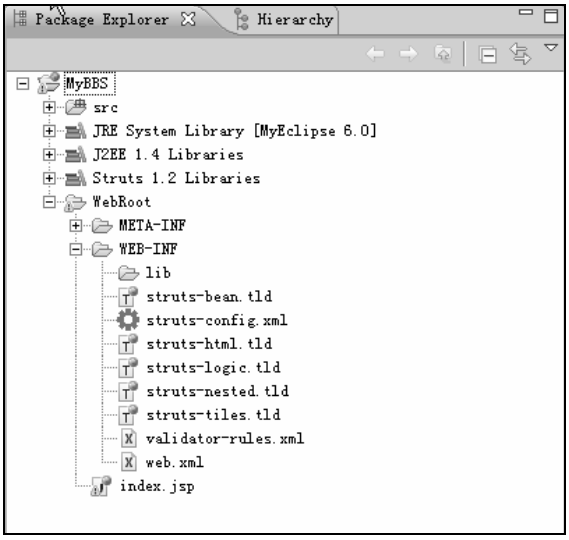


图 1-18 添加了 Struts 的 Web 工程

2. 实现这样的功能，我们首先需要一个 Web 页面，用以将论坛版块信息以表格的形式展现到客户端，在 WebRoot 下建立 jsp 文件夹，并在其中建立 leibie.jsp：

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<%@ page import="com.mybbs.pojo.Item"%>
<%@ page import="java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
        <title>MyBBS 后台管理页面 </title>
        <link href="<%=request.getContextPath()%>/css/main.css"
        rel="stylesheet" type="text/css" />
        <link href="<%=request.getContextPath()%>/css/lb.css"
     rel="stylesheet" type="text/css" />
    </head>
    <body>
        <div id="bgmain">
            <div id="top">
                <div class="l">
                    <a href="#" class="a">MyBBS 我的论坛</a>
                </div>
                <div class="r">
                    <a href="#" class="a">登录论坛 </a>
                </div>
```

```
            </div>
            <div class="header">
            </div>
            <div id="content_tb">
                <h3 class="boxhd">
                    帖子类别管理
                </h3>
                <div class="rtj">
                    <a class="al"
    href="<%=request.getContextPath()%>/jsp/addleibie.jsp"> 添加帖子类别</a>
                </div>
                <div class="rows">
                    <ul>
                        <li class="t_leftli">
                            编号
                        </li>
                        <li class="t_leftli">
                            类别名称
                        </li>
                        <li class="t_leftli">
                            类别编码
                        </li>
                        <li class="t_leftli">
                            修改
                        </li>
                        <li class="t_leftli">
                            删除
                        </li>
                    </ul>
                    <%
                        java.util.List         list         =         (List)
request.getAttribute("list");
                    java.util.Iterator it = list.iterator();
                        while (it.hasNext()) {
                            Item item = (Item) it.next();
                    %>
                    <ul>
                        <li class="leftli">
                            <%=item.getSubid()%>
                        </li>
                        <li class="leftli">
                            <%=item.getSubname()%>
```

```
                        </li>
                        <li class="leftli">
                            <%=item.getSubcode()%>
                        </li>
                        <li class="leftli">
                            <a
href="<%=request.getContextPath()%>/ItemAction.do?method=beforeUpdateItem&i
d=<%=item.getSubid()%>&name=<%=item.getSubname()%>&code=<%=item.getSubcode(
)%>" class="al">修改</a>
                        </li>
                        <li class="leftli">
                            <a
href="<%=request.getContextPath()%>/ItemAction.do?method=deleteItem&id=<%=i
tem.getSubid()%>" class="al">删除</a>
                        </li>
                    </ul>
                    <%
                        }
                    %>
                </div>
            </div>
            <div id="menu">
                <ul>
                    <li>
                        <a href="leibie.html" class="a2">类别管理</a>
                    </li>
                </ul>
                <div class="bt">
                        论坛内部信息
                </div>
                <div class="text_area">
                    <textarea name="textarea" cols="17" rows="8"></textarea>
                </div>
            </div>
            <div id="footer">
                <p>
                    版权所有：MyBBS 我的论坛
                </p>
            </div>
        </div>
    </body>
</html>
```

3. 建立用以添加论坛版块的的页面 addleibie.jsp。

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html;
       charset=gb2312" />
        <title>商业资讯后台管理主页面</title>
        <link href="<%=request.getContextPath()%>/css/main.css"
        rel="stylesheet" type="text/css" />
        <link href="<%=request.getContextPath()%>/css/tjgl.css"
        rel="stylesheet" type="text/css" />
    </head>

    <body>
        <div id="bgmain">
            <div id="top">
                <div class="l">
                    <a href="#" class="a">MyBBS 我的论坛</a>
                </div>
                <div class="r">
                    <a href="#" class="a">登录论坛 </a>
                </div>
            </div>
            <div class="header">
            </div>
            <div id="content_tb">
                <h3 class="boxhd">
                    帖子类别管理
                </h3>
                <div class="xbt">
                    添加类别
                </div>
        <form action="<%=request.getContextPath() %>
                        /ItemAction.do?method=addItem" method="post"  >
                <div class="row02">
                    <div class="row01_l">
                        类别名称:
                    </div>
```

```
                    <div class="row01_r">
                    <input name="subname" class="box" id=xname size="20" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row01_l">
                        类别编码:
                    </div>
                    <div class="row01_r">
                    <input name="subcode" class="box" id=xname size="20" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row03_c">
                        <input type="submit"  value="保 存" />
                         
                        <input type="reset"  value="取 消" />
                    </div>
                </div>
                </form>
            </div>
            <div id="menu">
                <ul>
                    <li>
                        <a href="leibie.html" class="a2">类别管理</a>
                    </li>
                </ul>
                <div class="bt">
                        论坛内部信息
                </div>
                <div class="text_area">
                    <textarea name="textarea" cols="17" rows="8">
                    </textarea>
                </div>
            </div>
            <div id="footer">
                <p>
                    版权所有：MyBBS 我的论坛
                </p>
            </div>
        </div>
```

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

4. 用以更新论坛版块的页面 updateleibie.jsp。

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<%@ page import="com.mybbs.pojo.Item"%>
<!DOCTYPE   html   PUBLIC   "-//W3C//DTD   XHTML   1.0   Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html;
                                                charset=gb2312" />
        <title>商业资讯后台管理主页面</title>
        <link href="<%=request.getContextPath()%>/css/main.css"
            rel="stylesheet" type="text/css" />
        <link href="<%=request.getContextPath()%>/css/tjgl.css"
            rel="stylesheet" type="text/css" />
    </head>

    <body>
        <div id="bgmain">
            <div id="top">
                <div class="l">
                    <a href="#" class="a">MyBBS 我的论坛</a>
                </div>
                <div class="r">
                    <a href="#" class="a">登录论坛 </a>
                </div>
            </div>
            <div class="header">
            </div>
            <div id="content_tb">
                <h3 class="boxhd">
                    帖子类别管理
                </h3>
                <div class="xbt">
                    添加类别
                </div>
    <form action="<%=request.getContextPath() %>
                   /ItemAction.do?method=updateItem" method="post"  >
                <div class="row02">
                    <div class="row01_l">
```

```
                        类别名称:
                    </div>
                    <%
                        Item item=(Item)request.getAttribute("item");
                    %>
                    <div class="row01_r">
                        <input name="subname" class="box" id=xname
                               size="20" value="<%=item.getSubname()%>" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row01_l">
                        类别编码:
                    </div>
                    <div class="row01_r">
                        <input   name="subcode"   class="box"   id=xname
size="20" value="<%=item.getSubcode() %>" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row03_c">
                        <input type="hidden" name="subid"
                                 value="<%=item.getSubid() %>" />
                        <input type="submit"  value="保 存" />
                         
                        <input type="reset"  value="取 消" />
                    </div>
                </div>
                </form>
            </div>
            <div id="menu">
                <ul>
                    <li>
                        <a href="leibie.html" class="a2">类别管理</a>
                    </li>
                </ul>
                <div class="bt">
                        论坛内部信息
                </div>
                <div class="text_area">
                    <textarea name="textarea" cols="17" rows="8">
                    </textarea>
```

```
                </div>
            </div>
            <div id="footer">
                <p>
                    版权所有：MyBBS 我的论坛
                </p>
            </div>
        </div>

    </body>
</html>
```

5. 新建 com.mybbs.pojo 包并在包下建立叫作 Item 的 pojo 类，这个类就代表帖子的版块描述信息，其中包括如下字段：

```
  //帖子类别 id
    private int subid;
    //帖子类别名称
    private String subname;
    //帖子编号
   private int subcode;
```

6. 新建 com.mybbs.dao 包并在包下建立 JdbcDao.java 和 ItemDao.java 类，JdbcDao.java 用于打开和关闭数据库连接，ItemDao.java 用于对 item 表的持久化操作，分别在类中，键入如下代码：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.ArrayList;
import com.mybbs.pojo.Item;

public class ItemDao extends JdbcDao {
    //定义自身的一个变量
    private static ItemDao itemDao = null;
    //定义自身私有的构造函数
    private ItemDao() {
    };

    //通过一个简单的单利模式将 ItemDao 返回到客户端
    public synchronized static ItemDao getItemDao() {
```

```
        if (itemDao != null) {
            return itemDao;
        }
        itemDao = new ItemDao();
        return itemDao;
    }
    //得到帖子类别信息，以 list 的形式返回到客户端
    public List<Item> getItem() {
        //声明用以存放帖子类别信息的集合
        List<Item> list = new ArrayList();
        //得到数据库连接
        super.getConnection();
        //定义要执行的 sql 语句
        String sqlString = "select * from item";
        try {
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            rs = pstmt.executeQuery();
            //循环遍历结果集
            while (rs.next()) {
                Item item = new Item();
                item.setSubid(rs.getInt(1));
                item.setSubname(rs.getString(2));
                item.setSubcode(rs.getInt(3));
                list.add(item);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
        return list;
    }

    //插入帖子类别信息
    public void insertItem(Item item)
    {
        //得到数据库链接
        super.getConnection();
        //定义要执行的 sql 语句
        String sqlString="insert into Item values(?,?,?)";
        try {
```

```
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            pstmt.setInt(1,item.getSubid());
            pstmt.setString(2, item.getSubname());
            pstmt.setInt(3,item.getSubcode());
            //执行 sql 动作
            pstmt.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
    }

    //插入帖子类别信息
    public int selectMaxItemId()
    {
        //得到数据库链接
        super.getConnection();
        //定义要执行的 sql 语句
        String sqlString="select max(itemid) from Item";
        try {
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            //执行 sql 动作
            ResultSet rs=pstmt.executeQuery();
            //如果结果集当中有数据
            if(rs.next())
            {
                int itemid=rs.getInt(1);
                return itemid;
            }
            return 0;
        } catch (Exception ex) {
            ex.printStackTrace();
            return 0;
        } finally {
            super.closeConnction();
        }
    }

    //更新帖子类别
```

```
    public void updateItem(Item item)
    {
        //得到数据库链接
        super.getConnection();
        //定义要执行的 sql 语句
    String  sqlString="update  Item  set  itemname=?  ,  itemcode=?  where
itemid=?";
        try {
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            pstmt.setString(1, item.getSubname());
            pstmt.setInt(2,item.getSubcode());
            pstmt.setInt(3, item.getSubid());
            //执行 sql 动作
            pstmt.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
    }
    // 删除帖子类别
    public void deleteItem(Item item)
    {
        //得到数据库链接
        super.getConnection();
        //定义要执行的 sql 语句
        String sqlString="delete from Item where itemid=?";
        try {
            //预编译 sql 返回一个 PrePareStatement 对象
            pstmt = conn.prepareStatement(sqlString);
            pstmt.setInt(1,item.getSubid());
            //执行 sql 动作
            pstmt.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
    }
}
```

7. 新建 com.mybbs.service 包，并在包下建立 ItemService 类，用于封装持久化代码，进而完成对论坛版块信息的增删查改，其中键入如下代码：

```
import com.mybbs.pojo.Item;
import com.mybbs.dao.ItemDao;
import java.util.List;
import java.util.Iterator;

public class ItemService {
    //得到操作帖子类别信息的 DAO
    private ItemDao itemDao=ItemDao.getItemDao();
    //插入新的帖子类别
    public void insertItem(Item item)
    {
        //得到新的帖子类别 id
        item.setSubid(itemDao.selectMaxItemId()+1);
        itemDao.insertItem(item);
    }

    //更新帖子类别
    public void updateItem(Item item)
    {
        itemDao.updateItem(item);
    }

    //删除帖子类别
    public void deleteItem(Item item)
    {
        itemDao.deleteItem(item);
    }

    //得到所有帖子类别信息
    public java.util.List getItem()
    {
        java.util.List<Item> list=itemDao.getItem();
        return list;
    }
}
```

8. 新建 com.mybbs.actionform 包，并在包下建立 Itemform 类，键入如下代码：

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
```

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

public class Itemform extends ActionForm {
    //帖子类别 id
    private int   subid;
    //帖子类别名称
    private String subname;
    //帖子编号
    private int   subcode;

    public int getSubid() {
        return subid;
    }
    public void setSubid(int subid) {
        this.subid = subid;
    }
    public String getSubname() {
        return subname;
    }
    public void setSubname(String subname) {
        this.subname = subname;
    }
    public int getSubcode() {
        return subcode;
    }
    public void setSubcode(int subcode) {
        this.subcode = subcode;
    }
    //actioform 特有的验证方法
    public ActionErrors validate(ActionMapping mapping,HttpServletRequest
request)
    {
        return null;
    }
    //actionform 特有的重置方法
    public void reset(ActionMapping mapping,HttpServletRequest request)
    {
    }
}
```

9. 新建 com.mybbs.action 包，并在包下建立 ItemAction 类，注意此类继承自 org.apache.

struts.actions.DispatchAction，在其中键入如下代码：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.Itemform;
import com.mybbs.pojo.Item;
import com.mybbs.service.ItemService;
import javax.servlet.http.HttpSession;
import org.apache.struts.actions.DispatchAction;
import org.apache.struts.action.Action;

public class ItemAction extends DispatchAction {
    //添加帖子类别
    public ActionForward addItem(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
        //得到有关的 actionform 对象
        Itemform itemform=(Itemform)form;
        //声明有关帖子类别的 pojo 对象
        Item  item=new Item();
        item.setSubname(itemform.getSubname());
        item.setSubcode(itemform.getSubcode());
        //声明业务逻辑对象
        ItemService itemSerivce=new ItemService();
        itemSerivce.insertItem(item);
        return mapping.findForward("updatesuccess");
    }
    public ActionForward deleteItem(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
        String itemid=request.getParameter("id");
        //声明有关帖子类别的 pojo 对象
        Item  item=new Item();
        item.setSubid(Integer.parseInt(itemid));
        //声明业务逻辑对象
        ItemService itemSerivce=new ItemService();
        itemSerivce.deleteItem(item);
        return mapping.findForward("deletesuccess");
    }
    public ActionForward updateItem(ActionMapping mapping, ActionForm form,
```

```
        HttpServletRequest request, HttpServletResponse response) {
        //得到有关的 actionform 对象
        Itemform itemform=(Itemform)form;
        //声明有关帖子类别的 pojo 对象
        Item  item=new Item();
        item.setSubid(itemform.getSubid());
        item.setSubcode(itemform.getSubcode());
        item.setSubname(itemform.getSubname());
        //声明业务逻辑对象
        ItemService itemSerivce=new ItemService();
        itemSerivce.updateItem(item);
        return mapping.findForward("updatesuccess");
    }
    public ActionForward beforeUpdateItem(ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
 HttpServletResponse response) {
        //声明有关帖子类别的 pojo 对象
        Item  item=new Item();
        item.setSubid(Integer.parseInt(request.getParameter("id")));
        item.setSubcode(Integer.parseInt(request.getParameter("code")));
        item.setSubname(request.getParameter("name"));
        request.setAttribute("item",item);
        return mapping.findForward("update");
    }
    public ActionForward getItem(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
        //声明业务逻辑对象
        ItemService itemSerivce=new ItemService();
        java.util.List<Item> list=itemSerivce.getItem();
        request.setAttribute("list",list);
        return mapping.findForward("success");
    }
}
```

10. 打开 WEB-INF 当中的，struts-config.xml 文件，在其中键入：

```
<struts-config>
  <form-beans >
    <form-bean name="ItemForm" type="com.mybbs.actionform.Itemform" />
  </form-beans>
  <action-mappings >
   <action
```

```
     attribute="ItemForm"
     name="ItemForm"
     path="/ItemAction"
     scope="request"
     parameter="method"
     type="com.mybbs.action.ItemAction" >
        <forward name="success" path="/jsp/leibie.jsp"></forward>
        <forward name="update" path="/jsp/updateleibie.jsp"></forward>
        <forward name="updatesuccess"
path="/ItemAction.do?method=getItem" ></forward>
        <forward name="deletesuccess"
                          path="/ItemAction.do?method=getItem"></forward>
   </action>
  </action-mappings>
  <message-resources   parameter="com.mybbs.struts.ApplicationResources"
/>
</struts-config>
```

11. 部署运行程序，在地址栏里输入 http://localhost:8080/mybbs/ItemAction.do?method=getItem 打开查询页面，点击红色连接，为论坛添加版块。

图 1-19 查询论坛版块信息

12. 输入版块名称和编码后，点击保存。

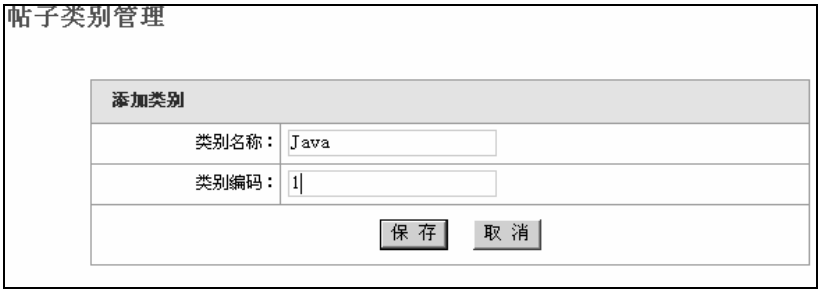


图 1-20 添加论坛版块

13. 如果要修改刚才填写的信息，点击红色“修改”链接。

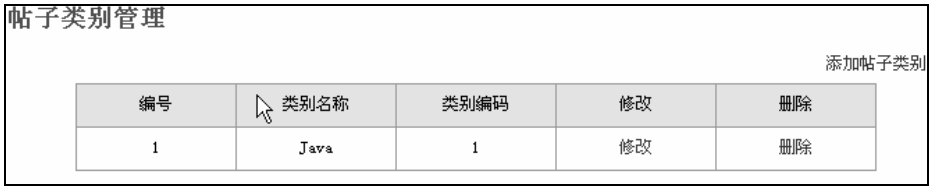


图 1-21 添加论坛版块

14. 填写要修改的信息。

图 1-22 修改论坛版块

15. 点击保存。

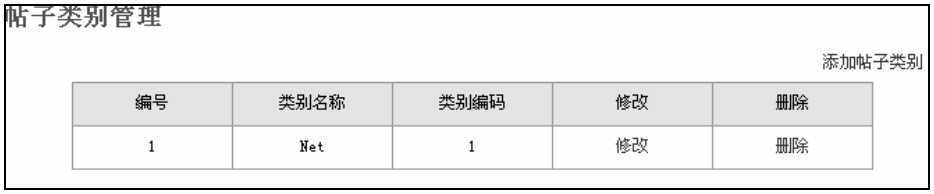


图 1-23 修改论坛版块

16. 要删除论坛版块点击红色“删除”链接便可。

图 1-24 删除论坛版块

4. 相关理论知识

4.1 Struts 当中的 DispatchAction

通常，在一个 Action 中只能完成一种业务操作，如果希望在同一个 Action 类中完成一组相关的业务操作，可以使用 DispatchAction 类。比如对一个用户对象来说，存在增加、删除、修改的操作，一种设计方案是为每种业务操作创建独立的 Action 类，如 addUserAction，del

UserAction，updateUserAction。尽管这种设计方案是可行的，但是这三个 Action 在执行各自的任务中，可能会执行一些相同的操作，比如 addUserAction 和 updateUserAction 都要进行相同的数据验证。

为了减少重复编程，使应用更加便于维护，可以由同一个 Action 类来完成一组相关的业务操作，DispatchAction 就提供了这种功能。

DispatchAction 是一个抽象类，它继承自 org.apache.struts.action.Action，如果要创建一个扩展 DispatchAction 类的子类，不必覆盖 execute()方法，而是创建一些实现实际业务操作的方法，这些业务方法都应该和 execute()方法具有同样的签名，即他们的参数和返回类型都应该相同，此外也应该申明抛出 Execption，如下例是一个扩展 DispatchAction 的例子。

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;
public class UserAction extends DispatchAction
{
    public ActionForward addUser (ActionMapping mapping,ActionForm form,
              HttpServletRequest   request,HttpServletResponse   response)
throws Exception
    {
             //增加用户业务的逻辑
            return mapping.findForward(“addsuccess”);
    }

    public ActionForward delUser(ActionMapping mapping,ActionForm form,
            HttpServletRequest request,HttpServletResponse response) throws
Exception
    {
             //删除用户业务的逻辑
            return mapping.findForward(“delsuccess”);
    }

    public ActionForward updateUser(ActionMapping mapping,ActionForm form,
            HttpServletRequest request,HttpServletResponse response) throws
Exception
    {
             //更新用户业务的逻辑
            return mapping.findForward(“updatesuccess”);
    }
}
```

如何实现这些不同方法的调用呢？那就是要在 struts-config.xml 文件中更改 action-mapping 的配置，如下：

```
< action-mappings >
    < action
      attribute = "addUserForm"
      input = "/addUser.jsp"
      name = "addUserForm"
      parameter="method"
      path = "/addUser"
      scope = "request"
      type="com.mybbs.action.UserAction" >
    </ action >
    < action
      attribute = "delUserForm"
      input = "/delUser.jsp"
      name = "delUserForm"
      parameter="method"
      path = "/delUser"
      scope = "request"
      type=" com.mybbs.action.UserAction" />
< action
      attribute = "updateUserForm"
      input = "/updateUser.jsp"
      name = "updateUserForm"
      parameter="method"
      path = "/updateUser"
      scope = "request"
      type=" com.mybbs.action.UserAction" />
  </ action-mappings >
```

可以看到每个<action />中都增加了 parameter=“”项，这个值可以随便命名，如上面命名为 method，当用户请求访问 DispatchAction 时，应该提供 method 请求参数，http://localhost:8080/mybbs/addUser.do?method= addUser。

以上 method 请求参数值为“addUser”，它指定了需要调用的业务方法，因此 DispatchAction 将调用相应的 addUser()。

4.2 Struts 当中其他控制器组件

4.2.1 org.apache.struts.actions.ForwardAction 类

在 JSP 网页中，尽管可以直接通过<jsp:forward>标签把请求转发给其他 Web 组件，但是 Struts 框架提倡先把请求转发给控制器，再由控制器来负责请求转发。

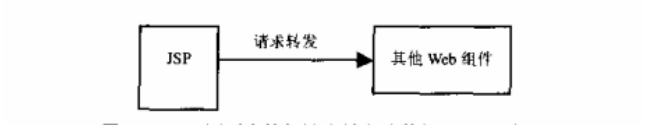


图 1-25 网页直接把请求发送到 Web 页面

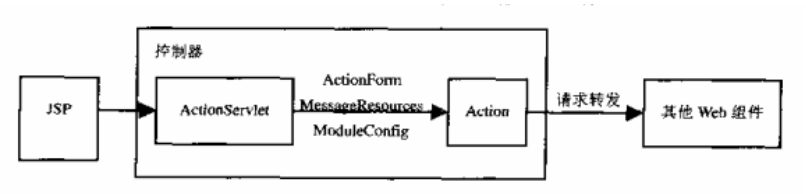


图 1-26 Jsp 网页经过控制器把请求转发给其他 Web 组件

用控制器来负责请求转发有以下一些优点：

控制器具有预处理请求功能，能够选择正确的子模块来处理请求，并且把子模块的 ModuleConfig 和 MessageResources 对象存放在 request 范围内。这样，请求转发的目标 Web 组件就可以正常地访问 ModuleConfig 和 MessageResources 对象，其中 ModuleConfig 和 MessageResources 为 Struts 内部组件，用于存放配置信息和资源信息。

如果 JSP 页面中包含 HTML 表单，那么控制器能够创建和这个表单对应的 ActionForm 对象，把用户输入表单数据组装到 ActionForm 中。如果<action>元素的 validate 属性为 true，那么还会调用 ActionForm 的表单验证方法。控制器把 ActionForm 对象存放在 request 或 session 范围内，这样请求转发的目标 Web 组件也可以访问 ActionForm。

JSP 网页之间直接相互转发违背了 MVC 的分层原则，按照 MVC 设计思想，控制器负责处理所有请求，然后选择适当的视图组件返回给用户，如果直接让 JSP 相互调用，控制器就失去了流程控制作用。

对于用户自定义的 Action 类，既可以负责请求转发，还可以充当客户端的业务代理，如果仅仅需要 Action 类提供请求转发功能，则可以使用 org.apache.struts.actions.ForwardAction 类。ForwardAction 类专门用于转发请求，不执行任何其他业务操作。

如下例我们可以通过“login.do”进入 login.jsp 页面：

```
<action path="/login" parameter="/login.jsp"
       type="org.apache.struts.actions.ForwardAction" scope="request" />
```

ActionServlet 把请求转发给 ForwardAction，ForwardAction 再把请求转发给<action>元素中 parameter 属性指定的 Web 组件。总之，在 Web 组件之间通过 ForwardAction 类来进行请求转发，可以充分利用 Struts 控制器的预处理请求功能。此外，也可以通过<action>元素的 forward 属性来实现请求转发。

此外，也可以通过<action>元素的 forward 属性来实现请求转发，以下代码完成同样的功能：

```
<action path="/login" forward="/login.jsp" scope="request" />
```

4.2.2 org.apache.struts.actions.IncludeAction 类

JSP 网页中，尽管可以直接通过<include>指令包含另一个 Web 组件，但是 Struts 框架提倡先把请求转发给控制器，再由控制器来负责包含其他 Web 组件。IncludeAction 类提供了包含其他 Web 组件的功能。与 ForwardAction 一样，Web 组件通过 IncludeAction 类来包含另一个 Web 组件，可以充分利用 Struts 控制器的预处理功能。

```
<action path=”/login” parameter=”/login.jsp”
       type=”org.apache.struts.actions.IncludeAction” scope=”request” />
```

<action>的 paramter 属性指定需要包含的 Web 组件。此外，也可以通过<action>元素的 include 属性来包含 Web 组件。

```
<action path=”/login” include=”/login.jsp” scope=”request” />
```

4.2.3 org.apache.struts.actions.LookupDispatchAction 类

LookupDispatchAction 类是 DispatchAction 的子类，在 LookupDispatchAction 类中也可以定义多个业务方法。通常 LookupDispatchAction 主要应用于在一个表单中有多个提交按钮，而这些按钮又都有一个共同的名字，这些按钮的名字和具体的 ActionMapping 的 parameter 属性值相对应。

直接以实例来说明，在继承 LookupDispatchAction 之后，您要重新定义 getKeyMethodMap()方法，并定义好自己的相关处理方法，例如：

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.apache.struts.actions.*;

public class EditAction extends LookupDispatchAction {
   protected Map getKeyMethodMap() {
      Map map = new HashMap();
      map.put("button.save", "save");
      map.put("button.preview", "preview");
      map.put("button.reset", "reset");
      return map;
   }

   public ActionForward save(ActionMapping mapping,
                        ActionForm form,
                        HttpServletRequest request,
                        HttpServletResponse response)
                           throws Exception {
      // ......
   }
```

```
   public ActionForward preview(ActionMapping mapping,
                        ActionForm form,
                        HttpServletRequest request,
                        HttpServletResponse response)
                          throws Exception {
      // ......
   }

   public ActionForward reset(ActionMapping mapping,
                        ActionForm form,
                        HttpServletRequest request,
                        HttpServletResponse response)
                          throws Exception {
      // ......
   }
}
```

在资源文件中包括以下的信息：

```
button.save=Save
button.preview=Preview
button.reset=Reset
```

为了要使用 LookupDispatchAction，在 struts-config.xml 中定义请求参数中该有的名称：

```
<action path="/edit"
         type="com.mybbs.action.EditAction"
         parameter="method"
         name="editForm"/>
```

现在假设您的表单页面包括以下的内容：

```
<form name="editForm" method="post"
       action="/edit.do">
    .....
    <input type="submit" name="method" value="Save"/>
    <input type="submit" name="method" value="Preview"/>
    <input type="submit" name="method" value="Reset"/>
 </form>
```

当您按下任一个按钮时，请求参数中会包括 method=Save 或是 method=Preview 或是 method= Reset，假设是 method=Save，LookupDispatchAction 会根据它作为 value，在资源文件当中找到对应的 key，然后根据 key 与 getKeyMethodMap()得知要执行的方法为 save()方法。

4.2.4 org.apache.struts.actions.SwitchAction 类

SwitchAction 类用于子应用模块之间的切换。如果使用 SwitchAction 类来进行子应用模块之间的切换，只需在 Struts 配置文件中做如下配置：

```
<action path=”/toModule” type=”org.apache.struts.actions.SwitchAction” />
```

对于请求访问 SwitchAction 的 URL，需要提供两个参数：

prefix，指定子应用模块的前缀，以“/”开头，默认子应用模块的前缀为空字符串“”

page，指定被请求 Web 组件的 URI，只需指定相对于被切换后的子模块的相对路径。

例如，如果要从默认模块切换到 ModuleB，并将请求转发給 ModuleB 的“/index.do”，可以采用与以下类似的 URL：

http://localhost:8080/action/toModule?prefix=/moduleB&page=/index.do

4.3 Struts 当中的多模块编程

在 Struts 框架中，还允许使用多个 struts-config.xml 文件，一个命名为“struts-config.xml”，一个命名为“struts-config-dao.xml”，配置时就应该用“,”他们隔开：

```
<init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml,
                /WEB-INF/struts-config-dao.xml
    </param-value>
</init-param>
```

这样两个配置文件的信息都会被 Web 应用环境所识别而一起工作。不过这样的配置有一个问题，因为每个<action>都是全局的，若有相同路径的<action>就会发生冲突。

为了解决这个问题，Struts 框架引入了模块的概念。将每个配置文件都视为一个模块就可以了，在 web.xml 中配置如下：

```
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
<param-name>config/dao</param-name>
<param-value>/WEB-INF/struts-config-dao.xml</param-value>
</init-param>
```

“config/xxx”中“/xxx”就是所指的模块，每个模块都通过<init-param>进行配置。在这里指定了一个缺省模块“config”，一个普通模块“config/dao”，这样就可以根据模块名转发到各个模块，进行各自不同的工作。

而在各个模块间的转发则是通过<forward>元素和 ContextRelative 的属性来实现。

```
<action input=”/Login.do”
        name=”LoginActionForm”
```

```
        path=”/LoginAction”
        scope=”request”
       type=”com.mybbs.action.LoginAction”
       validate=”true”>
       <forward name=”success” contextRelative=”true”  redirect="true"
        path=”dao/LoginAction.do”
</action>
```

注意<forward>元素部分的配置。

contextRelative 用来指定是否允许在模块间转发。

path 属性指定模块名，说明配置在 struts-config-dao.xml 中的 LoginAction.do，在原来的 path 前加上模块名，就可以在不同的模块前进行转发。对于缺省模块，不带模块名的 path 属性就可以了。

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 在 mybbs 数据库中建立 item 表并建立 MyBBS 工程。（10 分钟）
2. 编写 leibie.jsp，addleibie.jsp，updateleibie.jsp 页面。（30 分钟）
3. 编写用于数据访问的 JdbcDao.java，ItemDao.java 类。（10 分钟）
4. 编写业务逻辑类 ItemService。（10 分钟）
5. 编写 ItemAction 和 Itemform 类。（20 分钟）
6. 配置 struts-config.xml 文件，并调试运行程序。（10 分钟）

6. 作业

利用 DispatchAction 实现论坛帖子的增删查改。

案例三 Struts 当中使用数据连接池

1. 教学目标

1.1 Struts 数据库连接池的配置

1.2 Tomcat 数据库连接池配置

1.3 两种连接池的性能简要比较

1.4 Struts 国际化

2. 工作任务

2.1 使用 Struts 数据库连接池实现论坛登录示例

2.2 使用 Tomcat 数据库连接池实现库论坛登录示例

3. 相关实践知识

3.1 本章相关实例介绍

本章我们用 Struts 和 Tomcat 的数据库连接池分别去实现第一章所作的论坛登录示例，本例依然采用数据库当中的 userinfo 表，和第一章用到过的 login.jsp，image.jsp，error.jsp，success.jsp 四个页面，由于前面已经给出创建脚本和代码，本章不再给出，数据库连接池服务采用 APACHE 提供的 DBCP，此服务涉及到三个工具包，commons-pool-1.2.jar，commons-collections-3.1.jar，commons-dbcp-1.2.1.jar。

1. 打开 MyEclipse6.0，导入案例一的工程。

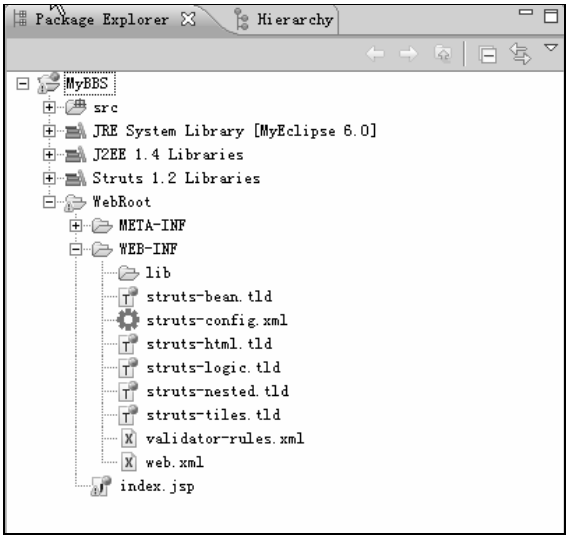


图 1-27 添加了 Struts 的 Web 工程

2. 将 DBCP 服务所需的三个工具包，和 MySql 数据库驱动加入工程。

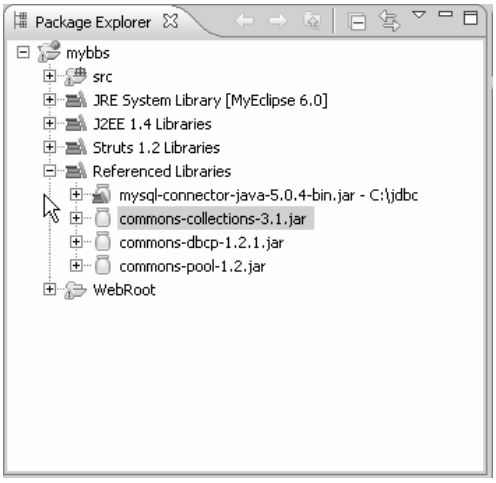


图 1-28 添加了驱动程序

3.2 使用 Sturts 数据库连接池实现论坛登录实例

1. 修改 com.mybbs.dao 中的 UserInfoDao 代码，修改成如下代码：

```
package com.mybbs.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.ArrayList;
import javax.sql.DataSource;
import javax.servlet.http.HttpServletRequest;
import com.mybbs.pojo.Userinfo;
public class UserinfoDao extends JdbcDao {
    //定义自身的一个变量
    private static UserinfoDao userinfoDao = null;
    //定义自身私有的构造函数
    private UserinfoDao(DataSource dataSource) {
        super(dataSource);
        try {
            conn = super.dataSource.getConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    //通过一个简单的单利模式将UserinfoDao返回到客户端
    public synchronized static UserinfoDao
getUserinfoDao(DataSource dataSource) {
        if (userinfoDao != null) {
            return userinfoDao;
        }
        userinfoDao = new UserinfoDao(dataSource);
        return userinfoDao;
    }
    //得到用户信息，以list的形式返回到客户端
    public List<Userinfo> getPerson(Userinfo user) {
        //声明用以存放用户信息的集合
        List<Userinfo> list = new ArrayList();
        //得到数据库连接
        super.getConnection();
        //定义要执行的sql语句
        String sqlString = "select * from userinfo where username=?
and userpwd=?";
        try {
            //预编译sql返回一个PrePareStatement对象
            pstmt = conn.prepareStatement(sqlString);
            //传入参数
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, user.getUserpwd());
            //执行sql并返回结果集
            rs = pstmt.executeQuery();
            //循环遍历结果集
            if (rs.next()) {
                Userinfo myuser = new Userinfo();
                myuser.setUsername(rs.getString(2));
                myuser.setUserpwd(rs.getString(3));
                list.add(myuser);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();

        }
        return list;
    }
}
```

2.修改 com.mybbs.service 包下的 UserinfoService 类，键入如下代码：

```
package com.mybbs.service;
import com.mybbs.pojo.Userinfo;
import com.mybbs.dao.UserinfoDao;
import java.util.List;
import java.util.Iterator;
import javax.sql.DataSource;
public class UserinfoService {
    private DataSource dataSource=null;
    //得到操作用户信息的 DAO
    private UserinfoDao userinfoDao=null;
    //验证用户登录信息，用户名密码正确返回 true
    public boolean userLogin(Userinfo user)
    {
         boolean isLogin=false;
         //根据传入的 user 对象从数据库中查询结果集
         List list=userinfoDao.getPerson(user);
         Iterator it=list.iterator();
         //如果集合中有数据说明用户可以登录
         if(it.hasNext())
         {
             isLogin=true;
         }
         return isLogin;
    }
    public UserinfoService(DataSource dataSource)
    {
        this.dataSource=dataSource;
        userinfoDao=UserinfoDao.getUserinfoDao(dataSource);
    }
}
```

3. 修改 com.mybbs.action 包下的 LoginAction 类，键入如下代码：

```
package com.mybbs.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.LoginActionform;
```

```
import com.mybbs.pojo.Userinfo;
import com.mybbs.service.UserinfoService;
import javax.servlet.http.HttpSession;
import javax.sql.DataSource;
import javax.naming.Context;
import javax.naming.InitialContext;

public class LoginAction extends Action {
    String scriptStr=null;
    DataSource dataSource=null;
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {

        //得到 LoginActionForm 对象
        LoginActionform LoginActionForm = (LoginActionform) form;
        //得到 session 对象
        HttpSession session=request.getSession();
        //判断验证码是否输入正确

    if(!session.getAttribute("rand").toString().equals(LoginActionForm.getR
and()))
        {
            scriptStr="<script type='text/javascript'>alert('请输入正确的验证
码!!!!! ');</script>";
            request.setAttribute("scriptStr", scriptStr);
            //页面跳转
            return mapping.findForward("self");
        }
        //实例化用户 bean
        Userinfo user=new Userinfo();
        //得到用户输入的用户名
        user.setUsername(LoginActionForm.getName());
        //得到用户输入密码
        user.setUserpwd(LoginActionForm.getPassword());
        //得到 struts 数据源
        dataSource=this.getDataSource(request);
        //声明业务逻辑对象
        UserinfoService userinfoService=new UserinfoService(dataSource);
        //如果用户名和密码输入正确登录成功
        if(userinfoService.userLogin(user))
        {
            return mapping.findForward("success");
```

```
        }
        //如果输入错误提示错误
        else
        {
            return mapping.findForward("error");
        }
    }
}
```

4. 打开 WEB-INF 当中的，struts-config.xml 文件，在其中键入如下代码：

```
<struts-config>
    <data-sources>
        <data-source key="org.apache.struts.action.DATA_SOURCE"
            type="org.apache.commons.dbcp.BasicDataSource">
            <set-property property="driverClassName"
                value="com.mysql.jdbc.Driver" />
            <set-property property="url"
                value="jdbc:mysql://127.0.0.1:3306/mybbs" />
            <set-property property="username" value="root" />
            <set-property property="password" value="sa" />
            <set-property property="maxActive" value="200" />
            <set-property property="maxIdel" value="30" />
            <set-property property="maxWait" value="10000" />
            <set-property property="autoReconnect" value="true" />
            <set-property property="max-connections" value="10" />
            <set-property property="min-connections" value="2" />
            <set-property property="inactivity-timeout" value="30" />
            <set-property property="wait-timeout" value="30" />
            <set-property property="eroDateTimeBehavior"
                                                value="convertToNull" />
        </data-source>
    </data-sources>
    <form-beans>
        <form-bean name="LoginActionForm"
            type="com.mybbs.actionform.LoginActionform" />
    </form-beans>
    <action-mappings>
        <action input="/jsp/login.jsp" attribute="LoginActionForm"
            name="LoginActionForm" path="/LoginAction" scope="request"
            type="com.mybbs.action.LoginAction">
            <forward name="self" path="/jsp/login.jsp"></forward>
            <forward name="success" path="/jsp/success.jsp"></forward>
```

```
            <forward name="error" path="/jsp/error.jsp"></forward>
        </action>
    </action-mappings>
    <message-resources
        parameter="com.mybbs.struts.ApplicationResources" />
</struts-config>
```

3.3 使用 tomcate 数据库连接池实现论坛登录实例

1. 修改 com.mybbs.action 包下的 LoginAction 类，键入如下代码：

```
package com.mybbs.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.LoginActionform;
import com.mybbs.pojo.Userinfo;
import com.mybbs.service.UserinfoService;
import javax.servlet.http.HttpSession;
import javax.sql.DataSource;
import javax.naming.Context;
import javax.naming.InitialContext;

public class LoginAction extends Action {
    String scriptStr=null;
    DataSource dataSource=null;
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {

        //得到 LoginActionForm 对象
        LoginActionform LoginActionForm = (LoginActionform) form;
        //得到 session 对象
        HttpSession session=request.getSession();
        //判断验证码是否输入正确
    if(!session.getAttribute("rand").toString().equals(LoginActionForm.get
Rand()))
        {
            scriptStr="<script type='text/javascript'>alert('请输入正确的验证
码!!!!! ');</script>";
            request.setAttribute("scriptStr", scriptStr);
```

```
            //页面跳转
            return mapping.findForward("self");
        }
        //实例化用户 bean
        Userinfo user=new Userinfo();
        //得到用户输入的用户名
        user.setUsername(LoginActionForm.getName());
        //得到用户输入密码
        user.setUserpwd(LoginActionForm.getPassword());
        //得到 tomcat 数据源
        try {
            Context ctx = new InitialContext();
            dataSource                    =                    (DataSource)
ctx.lookup("java:comp/env/jdbc/TestDB");
        } catch (Exception e) {
            e.printStackTrace();
        }
        //声明业务逻辑对象
        UserinfoService userinfoService=new UserinfoService(dataSource);
        //如果用户名和密码输入正确登录成功
        if(userinfoService.userLogin(user))
        {
            return mapping.findForward("success");
        }
        //如果输入错误提示错误
        else
        {
            return mapping.findForward("error");
        }
    }
}
```

2. 打开 WEB-INF 当中的 struts-config.xml 文件，在其中键入如下代码：

```
<struts-config>
  <form-beans >
     <form-bean name="LoginActionForm"
           type="com.mybbs.actionform.LoginActionform" />
     </form-beans>
  <action-mappings >
   <action
     input="/jsp/login.jsp" attribute="LoginActionForm"
     name="LoginActionForm" path="/LoginAction"
```

```
     scope="request" type="com.mybbs.action.LoginAction" >
           <forward name="self" path="/jsp/login.jsp"></forward>
           <forward name="success" path="/jsp/success.jsp"></forward>
           <forward name="error" path="/jsp/error.jsp"></forward>
    </action>
  </action-mappings>
</struts-config>
```

3. 在 tomcat6.0 主目录的 conf 文件夹下新建 context.xml 文件，在其中键入如下代码：

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Resource name="jdbc/TestDB"
              auth="Container" type="javax.sql.DataSource"
              maxActive="100" maxIdle="30" maxWait="10000"
              username="root" password="sa"
               driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/mybbs"/>
</Context>
```

8. 在地址栏里输入 http://localhost:8080/mybbs/jsp/login.jsp 打开登录页面。

图 1-29 登录页面

9. 输入正确的用户名，密码和验证码后点击提交，提示登录成功。

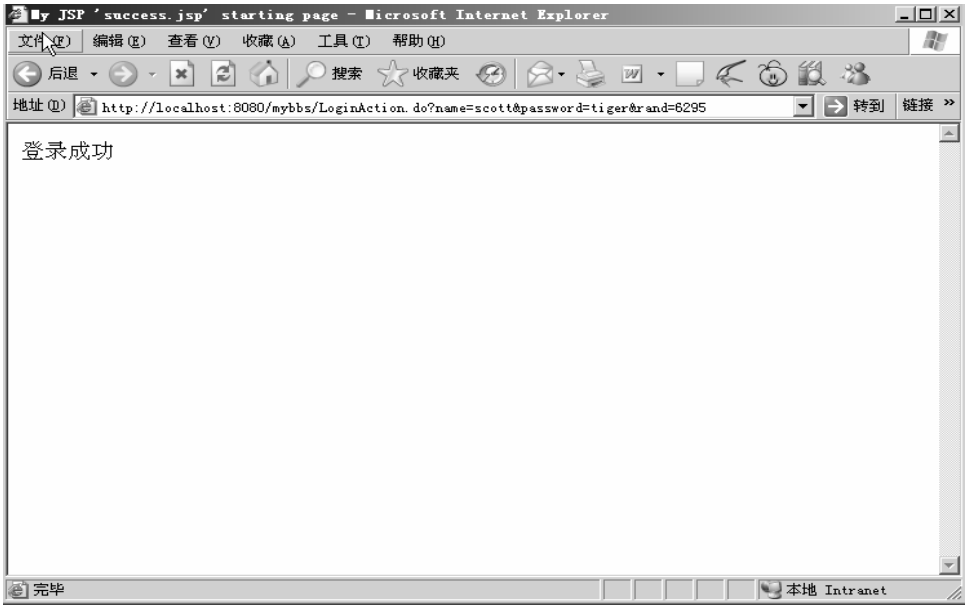


图 1-30 提示登录成功

10. 验证码、用户名或密码输入错误将在登录页面提示用户。

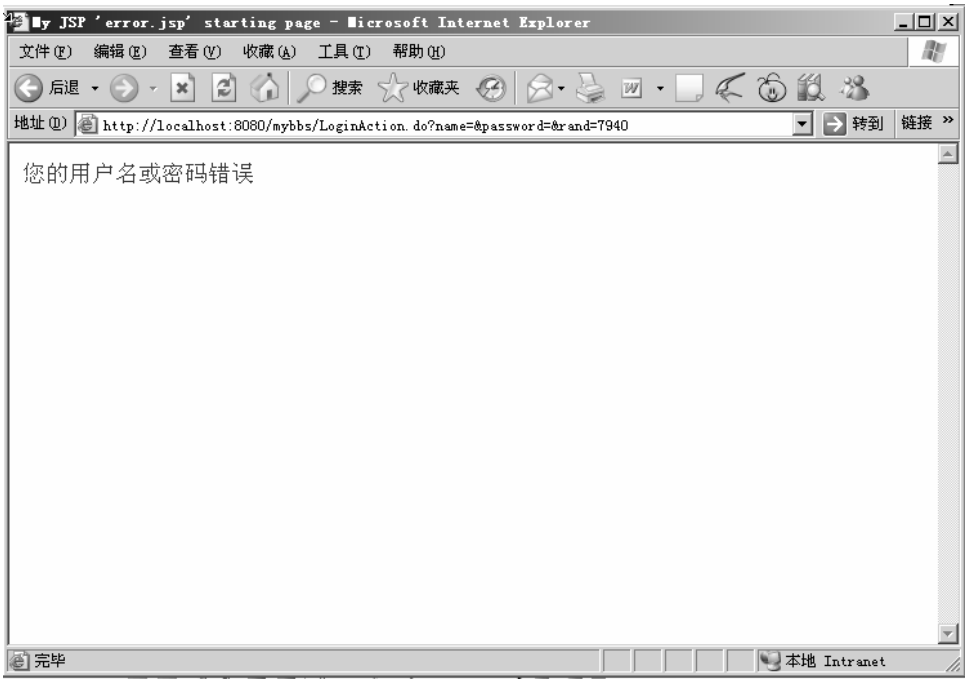


图 1-31 提示用户名密码错误

4. 相关理论知识

4.1 数据源介

传统的数据库连接方式（指通过 DriverManager 和基本实现 DataSource 进行连接）中，一个数据库连接对象均对应一个物理数据库连接，数据库连接的建立以及关闭对系统而言是耗费系统资源的操作，在多层结构的应用程序环境中这种耗费资源的动作对系统的性能影响尤为明显。

在多层结构的应用程序中通过连接池（Connection Pooling）技术可以使系统的性能明显得到提高，连接池意味着当应用程序需要调用一个数据库连接的时，数据库相关的接口通过返回一个通过重用数据库连接来代替重新创建一个数据库连接。通过这种方式，应用程序可以减少对数据库连接操作，尤其在多层环境中多个客户端可以通过共享少量的物理数据库连接来满足系统需求。通过连接池技术 Java 应用程序不仅可以提高系统性能同时也为系统提供了可测量性。

数据库连接池是运行在后台的而且应用程序的编码没有任何的影响。此中状况存在的前提是应用程序必须通过 DataSource 对象（一个实现 javax.sql.DataSource 接口的实例）的方式代替原有通过 DriverManager 类来获得数据库连接的方式。一个实现 javax.sql.DataSource 接口的类可以支持或不支持数据库连接池，但是两者获得数据库连接的代码基本是相同的。

JDBC2.0 提供了 javax.sql.DataSource 的接口，负责与数据库建立连接，实际应用时不需要编写连接数据库代码，直接从数据源获得数据库的连接。DataSource 中事先建立了多个数据库连接，这些数据库连接保持在数据库连接池中，当程序访问数据库时，只需要从连接池读取出空闲的连接，访问数据库结束，在将这些连接归还给连接池。DataSource 对象由容器提供，不能使用创建实例的方法来生成 DataSource 对象，要采用 Java 的 JNDI（Java Naming and Directory Interface，Java 命名和目录接口）来获得 DataSource 对象的引用。JNDI 是一种将对象和名字绑定的技术，对象工厂负责生产出对象，这些对象都和唯一的名字相绑定。程序中可以通过这个名字来获得对象的引用。容器把 DataSource 作为一种可配置的 JNDI 资源来处理，生成 DataSource 对象的工厂为 org.apache.commons.dbcp.BasicDataSourceFactory。

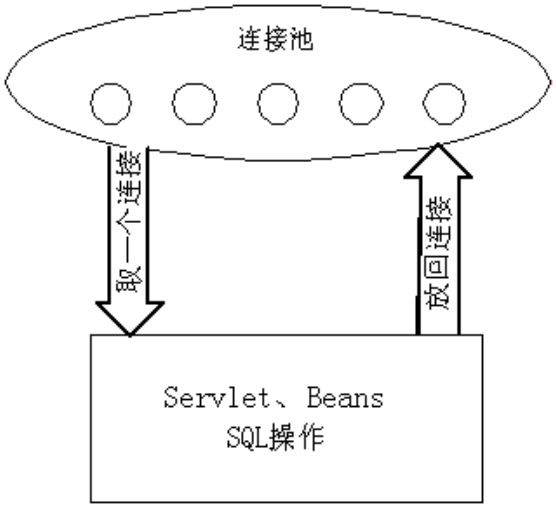


图 1-32 数据库连接池

4.2 Struts 配置数据源

Struts DataSource 管理器在 Struts 配置文件（struts-config.xml）里定义。这个管理器可以用来分发和配置任何实现了 javax.sql.DataSource 接口的数据库连接池（connection pool）。如果你的 DBMS 或者容器内置了符合这些要求的连接池，你可以优先选用它。

下面是一段 Struts-config.xml 配置文件中的数据源配置，你可以更改相应的设置以适合你自己的系统。

```
<data-sources>
        <data-source key="org.apache.struts.action.DATA_SOURCE"
            type="org.apache.commons.dbcp.BasicDataSource">
            <set-property property="driverClassName"
                value="com.mysql.jdbc.Driver" />
            <set-property property="url"
                value="jdbc:mysql://127.0.0.1:3306/mybbs" />
            <set-property property="username" value="root" />
            <set-property property="password" value="sa" />
            <set-property property="maxActive" value="200" />
            <set-property property="maxIdel" value="30" />
            <set-property property="maxWait" value="10000" />
            <set-property property="autoReconnect" value="true" />
            <set-property property="max-connections" value="10" />
            <set-property property="min-connections" value="2" />
            <set-property property="inactivity-timeout" value="30" />
            <set-property property="wait-timeout" value="30" />
            <set-property property="eroDateTimeBehavior"
                            value="convertToNull" />
        </data-source>
</data-sources>
```

该标签指定 JDBC 资源所需的参数。

属性名	说明
driverClassName	JDBC 驱动类的完整名称。通常驱动类的包放在 WEB-INF/lib 下
url	连到 JDBC 的 URL，不同的数据库不同
username	登录数据库的用户名
password	登录数据库的密码
maxActive	最大激活连接数
maxIdle	最大等待连接中的数量
maxWait	最长等待时间，代为 ms
autoReconnect	当数据源断开的时候是否自动重新连接
max-connections	最大连接数

min-connections	最小连接数
inactivity-timeout	数据库连接超时
wait-timeout	数据库连接闲置最大时间值
eroDataTimeBehavior	指定数据库中的 DateTime 字段默认值查询时的处理方式

配置好 DataSource 以后，你就可以在你的应用系统中使用这些数据源了。下面这段代码演示了怎样在 Action 类的 execute 方法中通过这些数据源来生成数据库连接。

```
public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
{
        DataSource dataSource;
        Connection cnn;

        try
        {
                dataSource = getDataSource(request);
                cnn = dataSource.getConnection();
                //数据连接已经建立了，你可以做你想做的事情了
        }
        catch (SQLException e)
        {
                getServlet().log("处理数据库连接", e);
        }
        finally
        {
                //在 finally 块里包含这些代码
                //用以保证连接最后会被关闭
                try
                {
                        cnn.close();
                }
                catch (SQLException e)
                {
                        getServlet().log("关闭数据库连接", e);
                }
        }
}
```

如果你需要在模块中使用多于一个的数据源，你可以在配置文件的<data-source>元素里包含一个 key 属性。

```
<data-source>
      <data-source key="A" type="org.apache.commons.dbcp.BasicDataSource">
      </data-source>
      <data-source key="B" type="org.apache.commons.dbcp.BasicDataSource">
      </data-source>
</data-source>
```

你在代码里，可以通过这些 key 获得不同的数据源。代码如下：

```
dataSourceA = getDataSource(request, "A");
dataSourceB = getDataSource(request, "B");
```

4.3 Tomcat6.0 配置数据源

Tomcat 的灵活配置，允许你可以通过多种方式配置连接池。先定义 Tomcat6 的安装根目录为${tomcat6}，以方便后面的描述。

Tomcat6 的服务器配置文件放在${tomcat6}/conf 目录底下。我们可以在这里找到 server.xml 和 context.xml。当然，还有其他一些资源文件。但我们只用得上这两个，其他的就不介绍了。

首先，需要为数据源配置一个 JNDI 资源，在 Tomcat6 版本中，Context 元素已经从 server.xml 文件中独立出来了，放在一个 context.xml 文件中。因为 server.xml 是不可动态重加载的资源，服务器一旦启动了以后，要修改这个文件，就得重启服务器才能重新加载。而 context.xml 文件则不然，Tomcat 服务器会定时去扫描这个文件。一旦发现文件被修改，就会自动重新加载这个文件，而不需要重启服务器。

由于 Context 元素的可用范围是可以控制的，我们可以根据需要为 Context 元素定义不同级别的可用范围。

4.3.1 全局可用

全局可用的范围意味着 Tomcat 服务器下面的所有应用都可以使用这个 Context 元素定义的资源。全局可用范围的 Context 元素在文件 ${tomcat6}/conf/context.xml 文件中描述。这个文件在 Tomcat 刚刚被安装的时候，是没有定义任何资源的。我们可以看到，这个文件的内容：

```
<Context>
 <WatchedResource>WEB-INF/web.xml<WatchedResource>
<Context>
```

其中的 WEB-INF/web.xml 表示服务器会监视应用的 WEB-INF/web.xml 文件来知道那个应用会引用在此处定义的资源。

4.3.2 指定的虚拟主机可用

指定的虚拟主机内可用就是说，在 Tomcat 服务器配置的虚拟主机中，只有指定的那个虚

拟主机上跑的应用才能使用。什么是虚拟主机和如何配置虚拟主机在这里就不描述了，有兴趣的同学自己去查 Tomcat 的官方资料。

要配置一个虚拟主机可用的 Context 资源，可以在${tomcat6}/conf/目录下的文件 ${enginename}/${hostname}/context.xml.default 中表述。

比如，一般一个 Tomcat 服务器安装好了以后，都有一个默认的叫做 Catalina 的引擎，在这个引擎下有一个叫做 localhost 的虚拟主机。我们的应用一般都放在这个虚拟主机下。那么，如果我们想要配置一个在 Catalina/localhost 虚拟主机下都可以使用的资源，我们需要在目录 ${tomcat6}/conf 下建立路径 Catalina/localhost，在这个路径下创建文件 context.xml.default。全路径是 ${tomcat6}/conf/Catalina/localhost/context.xml.default。

4.3.3 指定的应用可用

一个指定的应用可用的 Context 元素，意味着这是一个只有指定的引擎，指定的虚拟主机，指定的应用才可以使用的 Context 元素。

如果我们用 appname 来代表这个指定的应用的名字，那么元素的定义应该被放置在 ${tomcat}/conf/${enginename}/${hostname}/${appname}.xml 文件中。

例如，假设在 localhost 下我们有一个 Web 应用叫做 MyBBS，那么我们应该创建文件 ${tomcat6}/conf/Catalina/localhost/MyBBS.xml。

下面是一段 context.xml 配置文件中的数据源配置，你可以更改相应的设置以适合你自己的系统，我们将其配置为全局可用的资源。

```
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Resource name="jdbc/TestDB"
                auth="Container" type="javax.sql.DataSource"
                maxActive="100" maxIdle="30" maxWait="10000"
                username="root" password="sa"
                driverClassName="com.mysql.jdbc.Driver"
                  url="jdbc:mysql://localhost:3306/mybbs"/>
</Context>
```

Resource 标签指定了 JDBC 资源所需的参数。

属性名	说明
name	数据源 JNDI 名称
auth	指定管理 Resource 的 Manager。Container 表示容器内资源。
type	指定 Resource 所属的 Java 类名
driverClassName	JDBC 驱动类的完整名称。通常我驱动类的包放在 WEB-INF/lib 下
url	连到 JDBC 的 URL，不同的数据库不同
username	登录数据库的用户名
password	登录数据库的密码
maxActive	最大激活连接数

maxIdle	最大等待连接中的数量
maxWait	最长等待时间，代为 ms

配置好 Resource 以后，你就可以在你的应用系统中使用这些数据源了。下面这段代码演示了怎样在你的应用中生成数据库连接。

```
try
        {
            //JDBC 连接
            Context ctx = new InitialContext();
          DataSource ds
=(DataSource)ctx.lookup("java:comp/env/jdbc/TestDB");
            this.conn=ds.getConnection();
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
        catch(NamingException ex)
        {
            ex.printStackTrace();
        }
```

我们可以在一个 context 元素中定义多个资源。

```
<Context
         docBase="FiberScheduler"
         debug="5"
         reloadable="true"
         crossContext="true">

    <WatchedResource>WEB-INF/web.xmlWatchedResource>

    <Resource name=" jdbc/TestDB "
              auth="Container"
              type="javax.sql.DataSource"
              maxActive="4"
              maxIdle="30"
              maxWait="5000"
              username="netgeo"
              password="netgeo"
              driverClassName="oracle.jdbc.OracleDriver"
              url="jdbc:oracle:thin:@localhost:1521:fred"/>
```

```
    <Resource name=" jdbc/OtherTestDB "
              auth="Container"
              type="javax.sql.DataSource"
              maxActive="4"
              maxIdle="30"
              maxWait="5000"
              username="FiberSchedulerJBPM"
              password="FiberSchedulerJBPM"
              driverClassName="oracle.jdbc.OracleDriver"
              url="jdbc:oracle:thin:@localhost:1521:fred"/>
Context>
```

你代码里，可以通过这些 name 获得不同的数据源。代码如下：

```
DataSource ds =(DataSource)ctx.lookup("java:comp/env/jdbc/TestDB");
DataSource ds =(DataSource)ctx.lookup("java:comp/env/jdbc/OtherTestDB ");
```

4.4 Struts 和 I18N

人们常用 I18N 作为“国际化”的简称，其来源是英文单词 internationalization 的首字符 i 和 n 之间的字符数为 18。随着全球经济的一体化成为一种主流趋势，软件开发者应该支持多国语言的国际化的 Web 应用。对于 Web 应用来说，用样的页面在不同的语言环境下需要显示不同的效果。也就是说，一个 Web 应用程序在运行时能根据用户客户端请求来自的国家和语言的不同显示不同的用户界面。这样，当需要在应用程序添加对一种新的语言开发环境的支持时，不需要对已有的软件返工，无需修改应用程序的程序代码。

在 Struts 框架中进行程序的国际化，支持重点在于应用程序的文本和图片表示。最重要的工作就是准备 Resource Bundle 资源包。事实上，准备资源包的过程，就是把对应不同语言环境的用户涉及的文本和图片保存在多个配置文件中，客户端根据不同的环境需要进行更换。这些配置文件被称为“属性文件”，所有属性文件和在一起被成为资源包（Resource Bundle）。

我们要开发一个国际化的应用程序，使它同时支持中文和英文。

当建立完工程，并添加了 Struts 支持以后，我们编辑 ApplicationResources.properties 文件，选择文件代码如下：

```
Welcome=Welcome!!
```

创建英文资源文件 ApplicationResources_en.properties。

```
Welcome=Welcome!!
```

创建临时中文文件 ApplicationResources_temp.properties。

```
Welcom=欢迎您!!!!
```

对临时中文文件进行编码转换。在 DOS 下执行命令：

```
native2ascii –encoding gb2312 ApplicationResources_temp.properties
            ApplicationResources_zh_CN.properties
```

在 JSP 文件当中我们使用<bean:message />读取信息。

```
<%@ page language="java" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

<html>
    <head>
        <title>JSP for LoginActionForm form</title>
    </head>
    <body>
        <bean:message key=”Welcom”/>
    </body>
</html>
```

设置 IE 浏览器的语言选项。依次选择【工具】➔【Internet 选项】➔【常规】➔【语言】，打开 JSP 页面显示中文“欢迎您!!!!”。

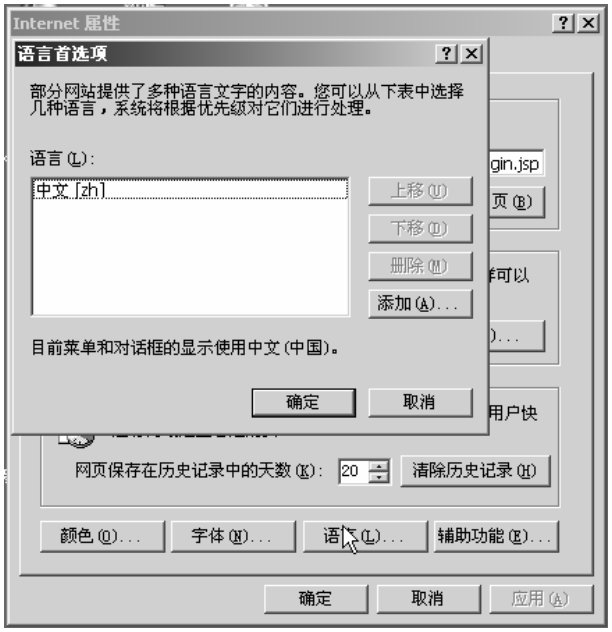


图 1-33 IE 浏览器的语言环境

设置 IE 浏览器的语言选项。依次选择【工具】➔【Internet 选项】➔【常规】➔【语言】➔【添加】，打开 JSP 页面显示英文：Welcome!!。

图 1-34 设置 IE 浏览器的语言环境为英文

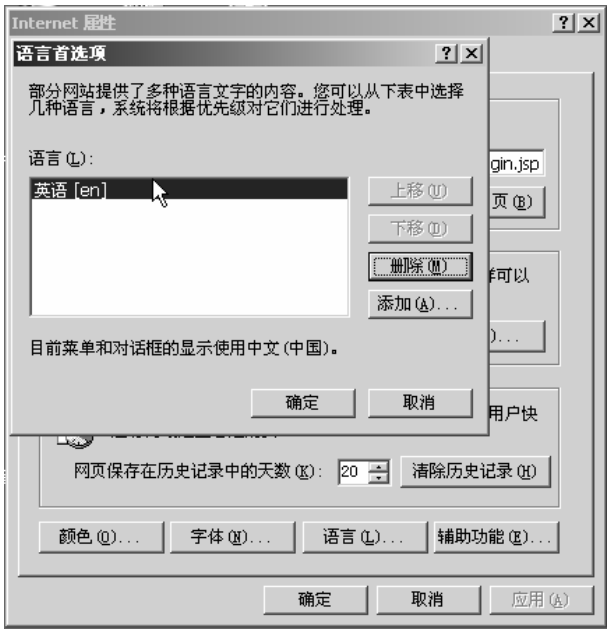


图 1-35 设置 IE 浏览器的语言环境为英文

资源包是一种文件，他包含应用程序默认语言的健/值对。此文件的命名格式为 ResourceBunleName_language_COUNTRY.properties。例如：

英文（美国） ApplicationResources _en_US.properties

中文（简体）ApplicationResources _zh_CN.properties

中文（繁体）ApplicationResources _tw.properties

通过使用<message-resources>元素可以在 Struts 配置文件定义多个资源包。而每个附加资源包可以使用 key 属性指定包的名称。

```
<message-resources parameter=”ApplicationResources” />
<message-resources key=”res” parameter=”MyResources” />
```

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 mybbs 数据库及 userinfo 表和 MyBBS 工程。（5 分钟）
2. 在 Struts 或 Tomcat 当中配置数据库连接池。（15 分钟）
3. 编写 login.jsp，success.jsp，error.jsp 页面。（10 分钟）
4. 编写用以生成验证码的 image.jsp。（15 分钟）
5. 编写用于数据访问的 JdbcDao.java，UserinfoDao.java 类。（10 分钟）
6. 编写业务逻辑类 UserinfoService。（10 分钟）
7. 编写 LoginAction 和 LoginActionForm 类。（20 分钟）
8. 配置 struts-config.xml 文件，并调试运行程序。（10 分钟）

6. 作业

利用 Struts 框架的国际化功能，修改 login.jsp 页面，完成国际化。提示：

1．修改 login.jsp 如果所示：

```
<%@ page language="java" pageEncoding="Gb2312"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean"
   prefix="bean"%>
<HTML>
   <HEAD>
      <TITLE>MyBBS登录页面</TITLE>
      <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=gb2312">
   </HEAD>
   <BODY BGCOLOR=#FFFFFF LEFTMARGIN=0 TOPMARGIN=0 MARGINWIDTH=0
      MARGINHEIGHT=0 VLINK="#3399CC">
      <link type="text/css" rel="stylesheet"
         href="<%=request.getContextPath()%>/css/login.css" />
      <%
         if (request.getAttribute("scriptStr") != null) {
```

```
        %>
        <%=request.getAttribute("scriptStr")%>
        <%
            request.setAttribute("scriptStr", null);
            }
        %>

        <TABLE WIDTH=768 BORDER=0 CELLPADDING=0 CELLSPACING=0
ALIGN="center">
            <TR>
                <TD COLSPAN=3>
                    <IMG
SRC="<%=request.getContextPath()%>/images/1_1.gif" WIDTH=768
                        HEIGHT=142 ALT="">
                </TD>
            </TR>
            <TR>
                <TD HEIGHT="30px">
                </TD>
            </TR>
            <TR>
                <TD>
                    <DIV ID="logindiv">
                        <BR />
                        <BR />
                        <form
action="<%=request.getContextPath()%>/LoginAction.do">
                            <DIV ALIGN="center">
                                <TABLE>
                                    <CAPTION>
                                        <bean:message key="login"/>
                                    </CAPTION>
                                    <TR>
                                        <TD>
                                        <bean:message key="name"/>:
                                        </TD>
                                        <TD>
                                            <input type="text" name="name"
size="35" maxlength="35" />
                                        </TD>
                                    </TR>
                                    <TR>
```

```
                                    <TD HEIGHT="5px"></TD>
                                </TR>
                                <TR>
                                    <TD>
                                        <bean:message key="pass"/>:
                                    </TD>
                                    <TD>
                                        <input type="password"
name="password" size="35"
                                            maxlength="35" />
                                    </TD>
                                </TR>
                                <TR>
                                    <TD HEIGHT="5px"></TD>
                                </TR>
                                <TR>
                                    <TD>
                                        <bean:message key="code"/>:
                                    </TD>
                                    <TD>
                                        <TABLE>
                                            <TR>
                                                <TD>
                                                    <input type="text"
name="rand" size="20" maxlength="20" />
                                                </TD>
                                                <TD>
                                                    <IMG
SRC="<%=request.getContextPath()%>/jsp/image.jsp" />
                                                </TD>
                                            </TR>
                                        </TABLE>
                                    </TD>

                                </TR>
                                <TR>
                                    <TD HEIGHT="20px"></TD>
                                </TR>
                                <TR>
                                    <TD ALIGN="center">

         
```

```
                                        <input type="submit"
value='<bean:message key="submit"/>' />
                                    </TD>
                                    <TD ALIGN="center">

        
                                        <input type="reset"
value='<bean:message key="reset"/>' />
                                    </TD>
                                </TR>
                            </TABLE>
                        </DIV>
                    </form>
                </DIV>
            <TD>
          </TR>
       </TABLE>
   </BODY>
</HTML>
```

2．修改并新建工程的资源文件，ApplicationResources_zh_CN.properties 代码如下：

```
login=\u8bba\u8c08\u767b\u5f55
name=\u7528\u6237\u540d
pass=\u5bc6\u7801
code=\u9a8c\u8bc1\u7801
submit=\u63d0\u4ea4
reset=\u91cd\u7f6e
```

ApplicationResources_en_US.properties 代码如下：

```
login=bbs login
name=user name
pass=password
code=code
submit=submit
reset=reset
```

3．以不同的 IE 语言环境打开登录页面。

案例四 Struts 标签和 Validator 框架

1. 教学目标

1.1 Struts 常用标签的使用

1.2 Validator 验证框架

2. 工作任务

4.1 使用 Validator 验证框架实现论坛注册

3. 相关实践知识

本案例，涉及到 mybbs 数据库当中的 userinfo 表，表结构如下：

字段名称	类型	大小	是否为空	备注
userid	int	4	否	主键，自增，代表人员编号
username	varchar	20	否	代表人员姓名
userpwd	varchar	20	否	代表密码
userques	varchar	50	否	代表密码找回问题
userans	varchar	50	否	代表密码找回答案
integral	int	4	否	代表用户名
grade	varchar	20	否	代表用户等级
useremail	varchar	20	否	代表邮箱
sex	bit	1	否	代表性别

表 4-1 userinfo 表结构

1.新建 userinfo 表，其创建脚本如下:

```
Create database mybbs;
create table `mybbs`.`userinfo`(
       `userid` int not null auto_increment,
      `username` varchar(20) default '' not null,
      `userpwd` varchar(20) default '' not null,
      `userques` varchar(50) default '' not null,
      `userans` varchar(50) default '' not null,
      `integral` int not null,
      `grade` varchar(20) default '' not null,
      `useremail` varchar(20) default '' not null,
      `sex` bit,
       primary key (`userid`)
   )TYPE=INNODB,
```

```
    default character set gbk;
```

2. 打开 MyEclipse6.0，建立 MyBBS 工程，并添加 Struts 支持。
3. 新建 com.mybbs.pojo 包，在包下建立 Userinfo 类：

```
public class Userinfo implements java.io.Serializable{
  //用户编号
  private int userid;
  //用户名
  private String username;
  //密码
  private String userpwd;
  //密码找回问题
  private String userques;
  //密码找回答案
  private String userans;
  //用户积分
  private int ntegral;
  //用户等级
  private String grade;
  //电子邮箱
  private String useremail;
  //性别
  private byte sex;
  public int getUserid() {
      return userid;
  }
  public void setUserid(int userid) {
      this.userid = userid;
  }
  public String getUsername() {
      return username;
  }
  public void setUsername(String username) {
      this.username = username;
  }
  public String getUserpwd() {
      return userpwd;
  }
  public void setUserpwd(String userpwd) {
      this.userpwd = userpwd;
  }
  public String getUserques() {
```

```
        return userques;
    }
    public void setUserques(String userques) {
        this.userques = userques;
    }
    public String getUserans() {
        return userans;
    }
    public void setUserans(String userans) {
        this.userans = userans;
    }
    public int getNtegral() {
        return ntegral;
    }
    public void setNtegral(int ntegral) {
        this.ntegral = ntegral;
    }
    public String getGrade() {
        return grade;
    }
    public void setGrade(String grade) {
        this.grade = grade;
    }
    public String getUseremail() {
        return useremail;
    }
    public void setUseremail(String useremail) {
        this.useremail = useremail;
    }
    public byte getSex() {
        return sex;
    }
    public void setSex(byte sex) {
        this.sex = sex;
    }
}
```

4. 新建 com.mybbs.dao 包，在包下建立 JdbcDao 类和 UserinfoDao 类，用于访问数据库：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcDao {
    //数据库连接对象
    protected Connection conn=null;
    protected Statement stmt=null;
    protected PreparedStatement pstmt=null;
    protected ResultSet rs=null;
    //数据库驱动
    private String driverClass_Name="com.mysql.jdbc.Driver";
    //数据库链接 URL
    private                                                    String
url="jdbc:mysql://localhost:3306/mybbs?user=root&password=sa";

    public JdbcDao(){}

    //用于得到数据库连接
    public void getConnection()
    {
        Connection conn=null;
        try
        {
            //加载驱动
            Class.forName(driverClass_Name);
            //得到连接
            conn=DriverManager.getConnection(url);
            this.conn=conn;
        }
        catch(SQLException ex)
        {
            ex.printStackTrace();
        }
        catch(ClassNotFoundException ex)
        {
            ex.printStackTrace();
        }

    }
    //释放数据库资源
    public void closeConnction()
    {
```

```
        if(rs!=null)
        {
            try
            {
                rs.close();
            }
            catch(SQLException  ex)
            {
                ex.printStackTrace();
            }
        }
        if(stmt!=null)
        {
            try
            {
                stmt.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
        }
        if(pstmt!=null)
        {
            try
            {
                pstmt.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
        }
        if(conn!=null)
        {
            try
            {
                conn.close();
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
```

```
            }
        }
    }
}
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.ArrayList;
import com.mybbs.pojo.Userinfo;

public class UserinfoDao extends JdbcDao {

    private static UserinfoDao userinfoDao = null;

    private UserinfoDao() {
    };

    public synchronized static UserinfoDao getUserinfoDao() {
        if (userinfoDao != null) {
            return userinfoDao;
        }
        userinfoDao = new UserinfoDao();
        return userinfoDao;
    }
    //得到用户的最大编号
    public int getUserMaxId() {
        int maxid = 0;
        //得到数据库连接
        super.getConnection();
        //拼写 SQL 语句
        String sqlString = "select max(userid) from userinfo ";
        try {
            pstmt = conn.prepareStatement(sqlString);
            rs = pstmt.executeQuery();
            //返回最大编号值
            if (rs.next()) {
                maxid = rs.getInt(1);
            }
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
        return maxid;
    }
    //插入用户信息
    public void insertUserInfo(Userinfo user)
    {
        //得到数据库连接
        super.getConnection();
        //拼写 SQL 语句
        String sqlString =
                    "insert into mybbs.userinfo values (?,?,?,?,?,?,?,?,?)";
        try {
            pstmt = conn.prepareStatement(sqlString);
            pstmt.setInt(1,user.getUserid() );
            pstmt.setString(2, user.getUsername());
            pstmt.setString(3, user.getUserpwd());
            pstmt.setString(4, user.getUserques());
            pstmt.setString(5, user.getUserans());
            pstmt.setInt(6, user.getNtegral());
            pstmt.setString(7, user.getGrade());
            pstmt.setString(8, user.getUseremail());
            pstmt.setByte(9, user.getSex());
            //插入信息
            pstmt.executeUpdate();
            } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            super.closeConnction();
        }
    }
}
```

5. 新建 com.mybbs.service 包，在包下建立 UserinfoService 类，用于插入用户信息：

```
import com.mybbs.pojo.Userinfo;
import com.mybbs.dao.UserinfoDao;
import java.util.List;
import java.util.Iterator;
```

```
public class UserinfoService {

    private UserinfoDao userinfoDao=UserinfoDao.getUserinfoDao();

    public void insertUserInfo(Userinfo user)
    {
        user.setUserid(userinfoDao.getUserMaxId()+1);
        userinfoDao.insertUserInfo(user);
    }

}
```

6. 新建 com.mybbs.actionform 包，在包下建立 InserUserinfoActionForm 类：

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.validator.ValidatorForm;

public class InsertUserInfoActionForm extends ValidatorForm {
    //性别
    private Byte sex=1;
    //电子邮箱
    private String useremail;
    //密码找回问题
    private String userques;
    //用户名
    private String username;
    //密码找回答案
    private String userans;
    //密码
    private String userpwd;
    //确认密码
    private String validateuserpwd;
    //等级
    private String grade;
    //分数
    private int integral;

    public Byte getSex() {
        return sex;
    }
```

```
    public void setSex(Byte sex) {
        this.sex = sex;
    }

    public String getUseremail() {
        return useremail;
    }

    public void setUseremail(String useremail) {
        this.useremail = useremail;
    }

    public String getUserques() {
        return userques;
    }

    public void setUserques(String userques) {
        this.userques = userques;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUserans() {
        return userans;
    }

    public void setUserans(String userans) {
        this.userans = userans;
    }

    public String getUserpwd() {
        return userpwd;
    }

    public void setUserpwd(String userpwd) {
```

```
        this.userpwd = userpwd;
    }

    public String getGrade() {
        return grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public int getIntegral() {
        return integral;
    }

    public void setIntegral(int integral) {
        this.integral = integral;
    }

    public String getValidateuserpwd() {
        return validateuserpwd;
    }

    public void setValidateuserpwd(String validateuserpwd) {
        this.validateuserpwd = validateuserpwd;
    }
}
```

7. 新建 com.mybbs.action 包，在包下建立 LoginAction 类：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.actions.DispatchAction;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.InsertUserInfoActionForm;
import com.mybbs.pojo.Userinfo;
import com.mybbs.service.UserinfoService;

public class InsertUserInfoAction extends DispatchAction {
```

```
    public ActionForward add(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        InsertUserInfoActionForm         InsertUserInfoActionForm         =
(InsertUserInfoActionForm) form;
        //将用户信息保存到 UserInfo 的实例当中
        Userinfo user=new Userinfo();
        user.setUsername(InsertUserInfoActionForm.getUsername());
        user.setUserpwd(InsertUserInfoActionForm.getUserpwd());
        user.setUseremail(InsertUserInfoActionForm.getUseremail());
        user.setSex(InsertUserInfoActionForm.getSex());
        user.setUserans(InsertUserInfoActionForm.getUserans());
        user.setUserques(InsertUserInfoActionForm.getUserques());
        user.setNtegral(0);
        user.setGrade("初级用户");
        //利用 UserinfoService 类的实例将用户信息插入数据库
        UserinfoService userivce=new UserinfoService();
        userivce.insertUserInfo(user);
        return null;
    }
}
```

8. 新建 com.mybbs.struts 包，在包下建立 ApplicationResources.properties：

```
username=\u7528\u6237\u540d
userpwd=\u5bc6\u7801
validateuserpwd=\u786e\u8ba4\u5bc6\u7801
useremail=EMAIL
userques=\u95ee\u9898
userans=\u7b54\u6848
name=\u540d\u5b57
password=\u5bc6\u7801
rand=\u9a8c\u8bc1\u7801
email=EMAIL
addr=\u5730\u5740
phonelength=11
errors.required={0} \u4e0d\u80fd\u4e3a\u7a7a.
errors.email={0} \u9519\u8bef\u7684\u683c\u5f0f.
errors.invalid={0}\u683c\u5f0f\u4e0d\u5bf9.
phoneerror=\u76ee\u524d\u8fd8\u6ca1\u6709\u60a8\u7684\u6210\u7ee9
```

9. 新建和编写“register.jsp”页面：

```
<%@ page language="java" pageEncoding="Gb2312"%>
```

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean"
prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html"
    prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic"
    prefix="logic"%>
<HTML>
    <HEAD>
        <TITLE>登录页面</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=gb2312">
</HEAD>
<BODY BGCOLOR=#FFFFFF LEFTMARGIN=0 TOPMARGIN=0 MARGINWIDTH=0
        MARGINHEIGHT=0 VLINK="#3399CC">
<link type="text/css" rel="stylesheet"
href="<%=request.getContextPath()%>/css/login.css" />
<TABLE WIDTH=768 BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN="center">
<TR>
<TD COLSPAN=3>
<IMG SRC="<%=request.getContextPath()%>/images/1_1.gif" WIDTH=768
HEIGHT=142 ALT="">
</TD>
</TR>
<TR>
<TD HEIGHT="30px"><div id="nav">当前位置:>><a href="index.html">首页</a>>><a
href="register.html">注册页面</a></div></TD>
</TR>
<TR>
<TD><DIV ID="logindiv"><BR /><BR />
    <html:javascript formName="InsertUserInfoActionForm"/>
<html:form  action="/InsertUserInfoAction.do?method=add"  onsubmit="return
validateInsertUserInfoActionForm(this)">
    <DIV ALIGN="center">
        <TABLE>
        <CAPTION>用户注册</CAPTION>
        <TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />用户名: </TD>
<TD><html:text property="username" size="35" maxlength="35" /></TD>
        </TR>
        <TR><TD HEIGHT="5px"></TD></TR>
        <TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />密  
码:</TD>
```

```
<TD><html:text property="userpwd" size="35" maxlength="35" /></TD>
      </TR>
<TR><TD HEIGHT="5px"></TD></TR>
<TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />确认密码:</TD>
<TD><html:text property="validateuserpwd" size="35" maxlength="35" /></TD>
     </TR>
     <TR><TD HEIGHT="5px"></TD></TR>
    <TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />性  
别:</TD>
<TD>
<TABLE>
    <TR>
<TD ALIGN="CENTER">  男:<html:radio property="sex" value="1"  />
</TD>
<TD ALIGN="CENTER">      女:
<html:radio property="sex" value="2" />
</TD></TR></TABLE></TD>
</TR><TR><TD HEIGHT="5px"></TD></TR>
<TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />EMAIL:</TD>
<TD><html:text property="useremail" size="35" maxlength="35" /></TD>
</TR>
<TR><TD HEIGHT="5px"></TD></TR>
<TR>
<TD><IMG  SRC="<%=request.getContextPath()%>/images/11.gif"  />密码找回问
题:</TD>
<TD><html:text property="userques" size="35" maxlength="35" /></TD>
</TR>
<TR><TD HEIGHT="5px"></TD></TR>
<TR>
<TD><IMG SRC="<%=request.getContextPath()%>/images/11.gif" />密码找回答:</TD>
<TD><html:text property="userans" size="35" maxlength="35" /></TD>
</TR>
<TR><TD HEIGHT="20px"></TD></TR>
<TR><TD ALIGN="center" COLSPAN="2">     
<html:submit>提交</html:submit></TD></TR></TABLE></DIV></html:form></DIV>
<TD></TR></TABLE></BODY></HTML>
```

10. 配置 struts-config.xml 文件：

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

```
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration                                                    1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  <data-sources />
  <form-beans >
    <form-bean name="InsertUserInfoActionForm"
                type="com.mybbs.actionform.InsertUserInfoActionForm" />
  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action
      attribute="InsertUserInfoActionForm"
      input="/cc.jsp"
      name="InsertUserInfoActionForm"
      path="/InsertUserInfoAction"
      scope="request"
      parameter="method"
      type="com.mybbs.action.InsertUserInfoAction" />
   </action-mappings>
  <message-resources parameter="com.mybbs.struts.ApplicationResources" />
  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml" />
  </plug-in>
</struts-config>
```

11. 在 WEB-INF 下新建 validation.xml，将其配置为：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
        "-//Apache  Software  Foundation//DTD  Commons  Validator  Rules
Configuration 1.0//EN"
        "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
<form-validation>
    <formset>
        <form name="InsertUserInfoActionForm">
        <field property="username" depends="required">
        <arg0 key="username"/>
        </field>
        <field property="userpwd" depends="required">
```

```
        <arg0 key="userpwd"/>
        </field>
        <field property="validateuserpwd" depends="required">
        <arg0 key="validateuserpwd"/>
        </field>
        <field property="useremail" depends="email,required">
        <arg0 key="useremail"/>
        </field>
        <field property="userques" depends="required">
        <arg0 key="userques"/>
        </field>
        <field property="userans" depends="required">
        <arg0 key="userans"/>
        </field>
       </form>
</formset>
</form-validation>
```

12. 运行效果：

图 1-36 注册页面

4. 相关理论知识

4.1 Struts 标签库

为了更容易、更快速地进行开发，Struts 提供了功能同 JSP 类似的标签库，常用的包括：HTML、Bean、Logic。

在运用 Struts 标签库前，我们需要通过 3 个步骤来配置一个 Struts 应用程序。

1. 在部署描述符（web.xml 文件）中注册

```
<taglib>
<taglib-uri>
http://jakarta.apache.org/struts/tags-html
</taglib-uri>
<taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
```

上面的代码告诉 ServletContainer 有关 Struts HTML 标签库的情况，以及在那里可以找到标签库的 TLD 文件。

2. 将 Struts-html.tld 文件复制到 WEB-INF 目录中。你不用担心标签库类文件，因为它们已经包含在 struts.jar 文件中了。

3. 在运用标签的每一个页面中，插入下面的<taglib>指示符：

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html"
prefix="html" %>
```

另外，你也可以运用 struts-html.tld 文件来学习它所支持的标签及每个标签可以带有哪些属性。

4.1.1 Struts HTML 标签库

HTML 标签提供了一组标签，它与 HTML 用于创建表单的标签集类似。使用 HTML 标签库可以将 Form Bean 中的数据放置在表单控制器中。标签库中的部分 HTML 标签如表。

属性名	说明
html	表示一个 HTML<html>元素
base	表示一个 HTML<base>元素
text	表示输入类型为文本的 HTML<input>元素
textarea	表示输入类型为文本区域的 HTML<input>元素
radio	表示一个单选按钮
form	定义一个表单元素
button	定义一个按钮输入字段
cancel	定义一个取消按钮
checkbox	定义一个复选框字段
rewrite	定义一个 URI

select	表示输入类型为选择的 HTML<input>元素

HTML 标签用来生成 HTML 语言中的 HTML 元素。这个标签有 Lang 和 Xhtml 两个属性。

- Lang：表明 HTML 标签生成的 HTML 元素里有没有 lang 属性。如果取值为 true，html 元素将呈现 lang 属性并设置为用户在本次 Web 会话中的区域语言。如果在 HttpSession 对象里没有找到区域语言信息，则使用 HTTP 标头中的 Accept-Language 字段所指定的语言；如果在当前请求里没有找到 Accept-Language 标头字段，则使用服务器的默认语言。
- Xhtml：如果这个属性的值设为 true，其他 HTML 标签将被强制性呈现为 XHTML 元素而不是普通的 HTML 元素。

JSP 页面里的<html:html lang="true">标签将生成如下所示的 HTML 元素：

```
<html lang=”zh-CN” >
```

Xhtml 属性将把与自己在同一个页面里的其他 HTML 标签强制性地呈现为 XHTML 元素而不是普通的 HTML 元素。

form 标签生成 HTML 语言中的 form 元素。这个标签最重要的属性是 action，它指定了将在用户提交表单时调用的 Struts 动作。在默认的情况下，HTML 表单都将按照 post 方法提交。action 属性的值必须指定 Struts 配置文件里定义过的动作的路径。同时，在 Struts 配置文件里，相应的 action 元素的 name 属性必须指向一个动作表单。

- action：指定 HTML 表单将提交给哪个动作。这个属性的值必须是在 Struts 配置文件里定义过的动作的路径。
- acceptCharset：服务器将接收的输入数据所使用的字符集编码清单
- enctype：用 post 方法提交的 HTML 表单的内容将使用的编码方案。在默认的情况下，这个属性的值是 application/x-www-form- urlencoded。对于一个用来上传文件的表单，这个属性必须设置为 multipart/form-data。
- method：用来提交这个请求的 HTTP 方法。在默认的情况下，这个属性的值是 post

一个 form 标签的示例。

```
<html:form action="/InsertUserInfoAction.do?method=add" >
</html>
```

text、password、hidden 和 textarea 标签将分别呈现为以下几种 HTML 元素：<input type= "text">、<input type= "password">、<input type= "hidden">和<textarea>。我们把这几个标签都有的属性列出：

- indexed：表明是否要为那些被赋值给 name 属性的值建立索引。
- name：表明由 property 属性指定的属性保存在哪一个作用域变量里。如果 name 属性不存在，则使用其封闭的 form 标签的 name 属性值。
- property：给出其封闭的 form 标签所对应的动作表单里将与呈现的 HTML 输入字段相关联的那个属性。请注意，property 属性做出的设置可以被 value 属性重写。

- value：一个常数，它将成为呈现的 HTML 输入字段的值。

此外，password 标签还有 redisplay 属性，表明是否要把 password 标签的现有值重新显示。hidden 标签还有 write 属性，如果 write 属性的值是 true，hidden 标签的值将发送到 HTTP 响应。

通常这四个标签一起使用，看下面示例：

```
<html:form action="/Login">
        sex : <html:hidden property="sex"/><br/>
        password : <html:text property="password"/><br/>
        comment : <html:textarea property="comment"/><br/>
        name : <html:text property="name"/><br/>
        <html:submit/><html:cancel/>
    </html:form>
```

submit 标签用来生成 HTML 语言中的<input type= "submit">元素，浏览器将把它呈现为 HTML 表单里的一个提交按钮。类似于 text 标签，submit 标签也有 property 属性，但它的 property 属性几乎没人会用到，因为需要把提交按钮与动作表单的某个属性关联起来的情况非常少见。

cancel 标签也将呈现为一个提交按钮，但其结果 HTML 元素的 name 属性永远会赋值为 org.apache.struts.taglib.html.CANCEL，cancel 标签也有 property 属性，但这个属性没有任何实际的用途。

当用户在 HTML 表单里按下取消按钮时，一个名为 org.apache.struts.taglib.html.CANCEL 的请求参数（这个参数的值是什么并不重要）就会发送到相应的动作 servlet。这个参数的作用是告诉 Struts“用户按下了取消按钮”，这样 Struts 将不对相关动作表单的属性做任何输入验证不管在 Struts 配置文件里把相应的 action 元素的 validate 属性设为什么值。接下来，控制权都将转交给相关动作类的 execute 方法，该动作类的实例会让它的 isCanceled 方法返回 true。于是，程序员只需检查这个方法就可以知道用户按下的是提交按钮还是取消按钮。如果用户按下的是取消按钮，把控制权转交给相应的资源就行了。

4.1.2 Struts Bean 标签库

此标签库和 JavaBean 有很强的关联性，设计的本意是要在 JSP 和 Java Bean 之间提供一个接口。Struts 提供了一套小巧的有用的标签库来操纵 Java Bean 和相关的对象。另外这套标签还可以读取资源文件当中的信息。

属性名	说明
cookie	检索 HTTP cookie 的值
define	基于 Bean 属性值定义一个脚本变量
header	从已命名的请求头检索值
include	检索 Web 应用程序资源的结果
message	从已定义的资源包中检索带健的值
page	检索存储在页面上下文中的 JSP 对象的值
parameter	检索由 name 属性确定的请求参数的值

resource	检索 Web 应用程序资源的值
size	检索集合或映射中包含的元素个数
struts	将 Struts 内部组件复制到脚本变量
write	检索并输出已命名 Bean 属性的值

<bean:define>标记用于定义一个变量。它的 id 属性用来指定变量的名称，toScope 属性用于指定变量存放的范围。一般情况下若不指定 toScope 属性，则默认的是存放在页面范围内。

定义变量有以下三种方式：

第一种是通过 value 属性直接设置变量的值，在这种情况下 id 属性所指定的变量是一个 String 类型。以下是示例代码：

```
<bean:define id="testString1" value="This is a test string"/>
<bean:write name="testString1"/>
```

运行效果：This is a test string

第二种方式是通过 name 属性和 property 属性共同指定一个变量来赋给 id 属性所定义的变量，此时 id 属性所定义的变量可以是任何类型。以下是示例代码：

```
<%
Date d = new Date();
pageContext.setAttribute("currDate",d);
%>
<bean:define id="milliseconds" name="currDate" property="time"/>
当前时间距离 1970 年 1 月 1 日的毫秒数为：<bean:write name="milliseconds"/>
```

上面的代码中将当前时间的 Date 型对象放入了 pageContext 对象中，然后使用<bean:define>标记定义了一个名为 milliseconds 的变量，标记的 name 属性指定了 Date 型对象，property 指定了 Date 型对象的 time 属性。因此，通过 id 定义的 milliseconds 属性的类型应为 long 型，它的值代表是 1970 年 1 月 1 日到现在的毫秒数。以下是代码的运行效果：当前时间距离 1970 年 1 月 1 日的毫秒数为：1211530579281。

第三种方式是通过 type 属性和 name 属性联合指定 id 所定义的变量的类型。type 属性是 id 所定义变量的完整类型，name 属性指定了已经存在的某个 JavaBean。以下是一段代码示例：

```
<%
    pageContext.setAttribute("session",request.getSession());
%>
<bean:define id="session_dup" name="session"
type="javax.servlet.http.HttpSession"/>
session 的上次访问时间：<bean:write name="session_dup"
property="creationTime"/>
```

在上面的代码中，首先将 session 对象存入 pageContext 对象中，然后使用<bean:define>标记的 id 属性定义的变量 session_dup 来引用它，type 属性设定为 javax.servlet.http.HttpSession。以下是运行的实际效果，显示的是 session 的上次访问时间。

运行效果：session 的上次访问时间：1211530579281。

<bean:parameter>标记用于读取 HTTP 请求中的参数。先看以下代码：

```
<html:link
page="/bean-parameter.jsp?testString=a+new+string&testInt=10000">
  点击此处添加请求参数
</html:link>
```

在上面的代码中，使用了<html:link>标记来向本页面发送请求，在请求后面跟上了两个参数 testString 和 testInt。

以下是<bean:parameter>标记的代码示例：

```
<bean:parameter id="test1" name="testString" value=""/>
请求参数 testString 的值为：<bean:write name="test1"/>
<bean:parameter id="test2" name="testInt" value=""/>
请求参数 testInt 的值为：<bean:write name="test2"/>
```

上面的代码中用到了<bean:parameter>标记的三个属性。

- name：用于指定页面请求中的参数名，如在上面的示例中，先后指定的请求参数名为 testString 和 testInt。
- id：在<bean:parameter>标记将 name 属性中所指定的请求参数取出后，保存在 id 属性所命名的变量中，这个变量存放在 pageContext 内。
- value：该属性指定请求参数的默认值。当 HTTP 请求中不包含 name 属性所指定的参数时，倘若不为该参数指定一个默认值，则后台将报错。因此，通常情况下都应为请求参数指定一个默认值。在上例中为请求参数 testString 和 testInt 指定的默认值均为一个空串。

<bean:message>用来读取资源文件中的信息。

```
<bean:message key=”propmpt.mailHostName” />
```

<bean:size>标记用于获取集合对象或数组对象的长度。它的 id 属性定义一个整型变量，它的值就是集合对象的长度，name 属性指定已经存在的集合对象或数组对象的名称。常用的集合对象有 HashMap，ArrayList 等。以下是一段示例代码：

```
<%
    ArrayList testlist = new ArrayList();
    testlist.add(new Integer(1));
    testlist.add(new Integer(2));
    testlist.add(new Integer(3));
pageContext.setAttribute("listforcount",testlist);
%>
<bean:size id="size" name="listforcount"/>
长度为：<bean:write name="size"/>
```

在上面的代码中，首先定义了一个 ArrayList 型的对象，然后为其添加三个成员，再将它存入 pageContext 对象中，并命名为 listforcount。接下来使用<bean:size>标记把这个 ArrayList 型的对象的长度取出放在一个叫 size 的 int 型整数中，并将它显示出来。

运行效果：3

<bean:write>标记用于在网页上转出内容。除了 Java 的基本类型外，它可以输出各种类型的对象，有点类似于 System.out.println()的功能。通常情况下，若所要输出的对象实现了 toString()方法，它就会调用对象的 toString()方法并输出相应字符串。若对象未实现 toString()方法，则会调用 Object 类的 toString()方法输出一个相应的字符串。

以下是一些简单的使用<bean:write>标记的例子：

```
<%
    String a = "string for test1";
    pageContext.setAttribute("test1",a);
%>
<bean:write name="test1"/>
```

上面的代码首先将一个字符串变量存于 pageContext 对象中，再用<bean:write>标记将其显示出来。

运行效果：string for test1

4.1.3 Struts Logic 标签库

Struts Logic 标签库中的标签可以根据特定的逻辑条件来控制输出页面内容，或者循环遍历集合中的所有元素。

属性名	说明
empty	检查 name 或 property 属性标识的脚本变量是否为空
notEmpty	检查 name 或 property 属性标识的脚本变量是否为空
eaqual	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否等于 value 属性指定的常量值
notEqual	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否不等于 value 属性指定的常量值
forward	将控制权交给 ActionForward 实体指定的页面
redirect	显示 HTTP redirect
greaterEqual	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否大于或等于 value 属性指定的常量值
greaterThan	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否大于 value 属性指定的常量值
iterate	在指定的集合上重复此标签的嵌套标签体内容
lessEqual	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否小于或等于 value 属性指定的常量值
lessThan	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否小于 value 属性指定的常量值

match	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否包含 value 属性指定的常量值
notMatch	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否不包含 value 属性指定的常量值
messagePresent	如果指定的消息出现在此请求中，则生成此标签的嵌套标签
messageNotPresent	如果指定的消息没有出现在此请求中，则生成此标签的嵌套标签
present	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否在应用的变量内
notPresent	检查 cookie，header，name，parameter 或 property 属性之一指定的变量是否不在应用的变量内

在 Logic 标记库中有以下六个比较标记：<logic:equal>、<logic:notEqual>、<logic:greaterThan>、<logic:greaterEqual>、<logic:lessThan>、<logic:lessEqual>。

从标记的名称中可以看出它们各自的功能。以下是一个简单的程序小示例：

```
<%
    pageContext.setAttribute("test1",new Integer(10000));
%>
<logic:equal name="test1" value="10000">
变量 test1 等于 10000。
</logic:equal>
```

上面的代码首先在 pageContext 内存入一个整型变量，命名为 test1。然后使用<logic:equal>标记把 test1 变量和常量值 10000 进行比较，如果比较值为真，则显示“变量 test1 等于 10000”的字符串。可以看出，在<logic:equal>标记中使用 value 属性来指定常量值，使用 name 属性指定一个已经在某个范围内存在的变量，这个范围可以是 pageContext，request 等。

运行效果：变量 test1 等于 10000

<logic:iterate>标记用于在页面中创建一个循环，以此来遍历如数组 Collection，Map 这样的对象。该标记的功能强大，在 Struts 应用的页面中经常使用到。以下是一段示例代码：

```
<%
    String                []                testArray1                =
{"str0","str1","str2","str3","str4","str5","str6"};
    pageContext.setAttribute("test1",testArray1);
%>
<logic:iterate id="array1" name="test1">
    <bean:write name="array1"/>
</logic:iterate>
```

在上面的代码中，首先定义了一个字符串数组，并为其初始化。接着，将该数组存入 pageContext 对象中，命名为 test1。然后使用 <logic:iterate>标记的 name 属性指定了该数组，并使用 id 来引用它，同时使用<bean;write>标记来将其显示出来。运行效果如下所示：

str0

str1

str2

str3

str4

str5

str6

Match 标记共有两个：<logic:match>，<logic:notMatch>。

Match 标记的功能有些类似于 java.lang.String 类中的 indexOf 方法。以下是一个简单的例子：

```
<%
    pageContext.setAttribute("test","HelloWorld");
%>
<logic:match name="test" value="Hello">
<bean:write name="test"/>
</logic:match>
```

面的代码首先在 pageContext 内存入一个字符串对象“HelloWorld”，并将其命名为 test。接下来使用<logit:match>标记的 value 属性指定一个子串“Hello”，来判断它是否被包含在 test 字符串中。如果被包含，则显示出 test 字符串的内容。很显然，结果为真。运行结果如下所示：

HelloWorld

Presence 标记包括以下四个<logic:present>、<logic:notPresent>、 <logic:messagesPresent>、<logic:messagesNotPresent>。

<logic:present>和<logic:present>两个标记，它们的功能是用于判断所指定的对象是否存在。例如有以下代码示例：

```
<%
    pageContext.setAttribute("ExistingString","teststring");
%>
<logic:present name="ExistingString">
ExistingString 的值为<bean:write name="ExistingString"/>
</logic:present>
```

上面的代码首先向 pageContext 对象中存入一个字符串对象，并命名为 ExistingString。然后使用<logic:present>标记来判断是否存在“ExistingString”这个变量。如果存在，就将 ExistingString 变量的值输出。以下是运行效果：
ExistingString 的值为 teststring。

<logic:present>和<logic:present>标记有以下几个常用属性：

- header：判断是否存在 header 属性所指定的 header 信息。
- parameter：判断是否存在 parameter 属性指定的请求参数。

- cookie：判断 cookie 属性所指定的同名 cookie 对象是否存在。
- name：判断 name 属性所指定的变量是否存在。
- property：和 name 属性同时使用，当 name 属性所指定的变量是一个 JavaBean 时，判断 property 属性所指定的对象属性是否存在。

<messagePresent>标记和<messageNotPresent>标记，这两个标记是来判断是否在 request 内存在特定的 ActionMessages 或 ActionErrors 对象。它们有几个常用的属性：

- name：指定了 ActionMessages 在 request 对象内存储时的 key 值。
- message：message 属性有两种取值。当其为 true 时，表示使用 Globals.MESSAGE_KEY 做为从 request 对象中获取 ActionMessages 的 key 值，此时无论 name 指定什么都无效；当其为 false 时，则表示需要根据 name 属性所指定的值做为从 request 对象中获取 ActionMessages 的 key 值，倘若此时未设置 name 属性的值，则使用默认的 Globals.ERROR_KEY。
- property：指定 ActionMessages 对象中某条特定消息的 key 值。

以下是一段代码示例：

```
<%
    ActionMessages messages = new ActionMessages();
    messages.add("message1",new ActionMessage("html.errors.error1"));
    request.setAttribute(Globals.MESSAGE_KEY,messages);
%>
<logic:messagesPresent message="true" property="message1">
所查找的 ActionMessage 存在。
</logic:messagesPresent>
```

在上面的代码中，首先创建了一个 ActionMessages 对象，然后向其中添加了一个 ActionMessage，将其命名为 message1。再把这个 ActionMessages 对象存入 requeset 对象中，命名为 Globals.MESSAGE_KEY。接下来，将<messagePresent>标记的 message 属性设为 true（即默认在 request 中查找 key 值为 Globals.MESSAGE_KEY 的对象），property 属性的值设为 message1，判断是否存在这样一个 ActionMessage 对象，倘若存在，则输出相应文本。运行效果如下所示：

所查找的 ActionMessage 存在。

Empty 标记共有两个：<logic:empty>，<logic:notEmpty>。

这两个标记的使用比较简单，以下是代码示例：

```
<%
    pageContext.setAttribute("test1","");
%>
<logic:empty name="test1">
test1 变量为空!
</logic:empty>
```

上面的代码首先向 pageContext 对象中存入一个空字符串对象，命名为 test1。然后使用<logic:empty>标记判断它是否为空。以下是实际运行效果：

test1 变量为空！

<logic:forward>标记用于进行全局转发，使用到该标记的页面一般不再编写其它内容，因为随着转发，页面将跳转，原页面中的内容也就无意义了。<logic:forward>标记和 struts-config.xml 文件中的<global-forward>内的子项相对应，以下是示例代码：

```
<logic:forward name="index"/>
```

当页面中包含有上面这样一句代码时，若用户请求该页面，则会自动跳转到主页。因此，用户在浏览器内看到的会是 index.jsp。

<logic:redirect>标记用于进行重定向请求。在 Java Web 的基本常识中用户应能首先了解过转发和重定向的区别，在此就不再赘述。因此，该标记与<logic:forward>在效果上基本一样，但它有另外几个属性分别是：

- href：将页面重定向到 href 指定的完整外部链接。
- forward：该属性与 struts-config.xml 中的<global-forward>内的子项相对应。即将页面重定向到 forward 所指定的资源。
- page：该属性指定一个本应用内的一个网页，标记将页面重定向到这个新的网页。

4.2 Validator 验证框架

Validator 框架以声明的方式为应用程序配置已存在的校验规则，并允许程序员添加更多的校验规则。Validator 框架属于 Jakarta 的子项目，也被当作 Struts 发布包的一部分。这说明，Validator 框架能被 Struts 框架很好的支持，也可以被独立出来使用。因此 Validator 框架和 Struts 框架是松散耦合的。Validator 框架为程序员带来了许多好处。

- 将校验统一在一个地方。
- 以声明而非编程的方式来实现校验。
- 校验规则与应用程序是松散耦合的，因此是可重用的。
- 添加新的校验规则和修改已存在的校验规则十分方便。
- 支持国际化。
- 支持正则表达式。

在任何 Web 应用当中，数据校验仿佛是无法避免的一种行为。然而，在 Validator 框架出现之前，校验的行为是有缺点的。

- 重用性的缺乏：开发时每次都要重新编写一遍校验逻辑，造成资源浪费。
- 统一控管的缺失：对后期维护来说，要改变校验就要对每个写过这种校验逻辑的代码进行修改，查找到校验的位置也是一个大麻烦。

在 Validator 框架出现后，校验的行为就得到了一定的控制，也解决了前面的问题。该框架使校验规则可以重复使用，同时，允许在整个 Web 应用中，将校验统一在某个特定的组件中。

Validator 框架由 Validator 校验类、Validator 配置文件、资源文件和 JSP 标签组成。

4.2.1 Validator 校验类

Validator 校验类就是在进行校验时所需要的一个 Java 类。每一次校验的发生，都会调用该校验类当中的方法。虽然校验类的方法都只针对一种校验规则，但是通过配置文件的声明，可以使校验规则产生合作，组成更复杂的校验规则。

Validator 框架的 Validator 校验类本身提供 14 种默认的校验规则，这些校验规则是最常见的校验。

声明名称	Validator 校验类方法名	作用
byte	isByte(String)	校验该校验值是否可以安全转换成为 byte 类型
short	isShort(String)	校验该校验值是否可以安全转换成为 short 类型
integer	isInteger(String)	校验该校验值是否可以安全转换成为 integer 类型
long	isLong(String)	校验该校验值是否可以安全转换成为 long 类型
float	isFloat(String)	校验该校验值是否可以安全转换成为 float 类型
double	isDouble(String)	校验该校验值是否可以安全转换成为 double 类型
creditCard	isCreditCard(String)	校验该校验值是否是合法的信用卡号
date	isDate(String)	校验该校验值是否是合法的日期
email	isEmail(String)	校验该校验值是否是属于合法的 E-mail
mask	matchRegexp(String,String)	校验该校验值是否能匹配一个正则表达式
maxLength	maxLength(String,int)	校验该校验值长度是否在给定的最大长度范围内
minLength	minLength(String,int)	校验该校验值长度是否大于等于在给定的长度
range	isInRange(int,int,int)	校验该校验值是否在最大值和最小值之间
required	isBlankOrNull(String)	校验该校验值是否不为空或不为 null

“声明名称”一栏是指定在配置文件中声明的实现所有的名称，“Validator 校验类方法名”一栏是指在 Validator 框架的校验类中根据声明调用的方法。

4.2.2 Validator 配置文件

通常说，Validator 配置文件是指如下两个配置文件：

- validator-rules.xml 是校验规则的配置文件，定义了应用程序中可以使用的 Validator 校验规则，换句话说，就是将所有可用的校验规则声明在 validator-rules.xml 中等待被调用。
- validation.xml 是具体校验配置文件，将 validator-rules.xml 中的校验和应用程序结合起来的配置文件，它的名字并不一定是“validation.xml”，可以由程序员自己指定，甚至可以不用该文件而直接在 validator-rules.xml 中指定具体校验，但为了降低耦合，还是应该分出配置文件。就 Struts 框架和 Validator 框架协同工作来看，validation.xml 将和 AcionForm 建立映射来校验从页面传入的数据。

我们需要以插件的形式将这两个文件配置在 struts-config.xml 中。

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property                                    property="pathnames"
value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml" />
```

```
</plug-in>
```

4.2.3 资源文件

在使用 validator 框架时，说得通俗一点，资源配置文件就是报错信息文件，是为了将校验出错信息显示给客户端而使用的。就 Struts 框架和 Validator 框架协同工作来看，可以认为，它就是 Struts 的 Resource 绑定所指的资源配置文件。

这是一个未经编码处理的资源文件，在项目中为了很好的支持中文我们需要使用 native2 ascii 编码来进行处理。下面是没处理的文件：

```
userName=姓名
passWord=密码
age=年龄
email=电子邮箱
errors.email={0}不是正确的电子邮箱
errors.maxlength={0}必须小于 12 位的字符
errors.minlength={0}必须大于 6 位的字符
errors.range={0}必须大于{1}小于{2}
errors.required={0}必须输入
```

4.2.4 Jsp 标签库

JSP 标签库用于读取资源配置文件中的抱错信息，并将其显示出来。Struts 框架中许多与资源配置文件相关的标签都实现了这种功能。

要在 JSP 页面生成校验的 JavaScript，必须在 JSP 中使用<html:javascript>标签，该标签有一个 formName 属性，该属性是定义在 struts-config.xml 以及 validation.xml 中所要校验的 ActionForm 名。

```
<html:javascript formName="CheckForm" />
```

如果需要触发自动生成的 JavaScript 校验规则，就要把校验的代码写在 form 的 onsubmit 方法中。

```
<html:form action="saveAction.do" onsubmit="return validateCheckForm(this);"
>
```

4.2.5 validation.xml

在 validator-rules.xml 中将校验类所实现的校验规则和声明衔接起来，通过声明的实现。通常情况下，validator-rules.xml 文件一旦被创建后就不应该被修改。它应该被当作一个校验的模板来看待，只有在新增加了其他校验之后，才会在 validator-rules.xml 中添加声明。通常使用 Sturts 框架给我们提供的 validator-rules.xml 文件就可以满足我们的需求。

validation.xml 是将 validator-rules.xml 中的校验和应用程序结合起来的配置文件。有了 validator-rules.xml 后，只要通过 validation.xml 将它与具体校验逻辑结合，就能完成真正的具体校验。

在 Struts 框架当中校验会放在 ActionForm 中进行。

```
import org.apache.struts.validator.ValidatorForm;
public class CheckForm extends ValidatorForm {
    private String userName;
    private int age;
    private String email;
    private String passWord;
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassWord() {
        return passWord;
    }
    public void setPassWord(String passWord) {
        this.passWord = passWord;
    }
}
```

要使用 Struts 的 Validator 框架进行自动校验，因此 ActionForm 不能继承普通的 ActionForm，必须继承自 ValidatorForm 或 ValidatorActionForm。二者区别：

- ValidatorForm 将<action>的 name 属性，为当前 ActionForm 调用相应的校验规则。
- ValidatorActionForm 将<action>的 path 属性，为当前 ActionForm 调用相应的校验规则。

我们可以看到对于 ActionForm 没有太大变化，而且也不需要实现它的 validate 方法，因为这些由 Validator 框架完成了。

为 CheckForm 编写，validation.xml 配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC "-//Apache Software Foundation//DTD Commons
Validator Rules Configuration 1.0//EN" "validator_1_0.dtd" >
<form-validation>
    <global>
        <constant>
            <constant-name>numericchar</constant-name>
            <constant-value>^\w+$</constant-value>
        </constant>
    </global>

    <formset>
        <form name="CheckForm">
            <field property="userName" depends="required,mask">
                <arg0 key="userName" />
                <var>
                    <var-name>mask</var-name>
                    <var-value>${numericchar}</var-value>
                </var>
            </field>

            <field property="passWord" depends="required,mask,minmaxlength">
                <arg0 key="passWord" ></arg0>
                <arg1 key="6"  resource="false"></arg1>
                <arg2 key="12" resource="false"></arg2>
                <var>
                    <var-name>mask</var-name>
                    <var-value>${numericchar}</var-value>
                </var>
                <var>
                    <var-name>maxlength</var-name>
                    <var-value>12</var-value>
                </var>
                <var>
                    <var-name>minlength</var-name>
                    <var-value>6</var-value>
                </var>
            </field>
            <field property="age" depends="range">
                <arg0 key="age" />
                <arg1 key="18" resource="false" />
                <arg2 key="30" resource="false" />
```

```
                <var>
                    <var-name>min</var-name>
                    <var-value>18</var-value>
                </var>
                <var>
                    <var-name>max</var-name>
                    <var-value>30</var-value>
                </var>
            </field>
            <field property="email" depends="email">
                <arg0 key="email"></arg0>
            </field>
        </form>
    </formset>
</form-validation>
```

本示例仅有一个 ActionForm，所以只有一个<form>元素，若有多个 ActionForm 则<form>元素<form>也应该有多个。

- <form>元素的 name 属性与 struts-config.xml 中的配置命名有很大的关系。当需要校验的 ActionForm 继承 ValidatorForm 时，<form>元素的 name 属性应该和 struts-config.xml 中声明一个 ActionForm 所给予的标识相匹配；当需要验证的 ActionForm 继承于 ValidatorActionForm 时，<form>元素的 name 属性和 struts-config.xml 中提交的<action>元素的 path 属性相匹配。主要通过<form>元素和 ActionForm 进行绑定，为 ActionForm 中的每个字段提供声明校验的实现。
- <form>元素有多个<field>子元素，以此来实现对每个定义在 ActionForm 中变量属性的校验。
- <field>元素用 property 属性来指明 ActionForm 中的一个变量，因此 property 属性的内容应该与定义在 ActionForm 中的变量属性相匹配，而且如果在 JSP 中使用标签库来显示校验失败的报错信息，那么它也将作为标签的属性给出。
- <field>元素的 depends 属性指明了该变量属性所依赖的校验规则，该规则应该在 validator-rules.xml 中被定义，由于在 validator-rules.xml 中为每个校验规则提供了从资源配置文件得到的报错消息，因此当依赖的校验规则出错后，就会以报错信息作为本校验规则出错的提示。Depends 属性中的校验规则可以有多个，并以“，”分开。
- <field>元素的<arg0>、<arg1>、<arg2>、<arg3>四个子元素会作为校验信息显示的替换参数所使用，它将替换本条校验规则报错所取得的报错信息的{0}、{1}、{2}、{3}

每个<argX>元素都有 key、name、resource 3 个属性，key 属性指定了资源配置文件中的一个“健”，因此替换报错信息时也会从资源配置文件中得到<argX>的内容。

- name 属性的内容应该是<field>元素的 depends 属性校验规则中的一个，它将指定为某个校验规则进行报错信息的替换。当然，name 属性也可以不出现，这样框架默认就会以<argX>来替换校验规则所需要的报错信息。

- 若不需要使用配置文件中的信息来替换<argX>，而直接使用字面文字，那么就应该出现 resource 属性，并将它设置为“false”，这时 key 属性的内容就会直接作为字面文字来实现替换了。
- <msg>子元素允许为<field>元素的校验规则提供另一条报错信息，而不是默认的从 validator-rules.xml 得到的报错消息。<msg>属性同样拥有 key、name 和 resource3 个属性，它们的作用和<argX>元素的 3 个属性一样。
- <var>子元素允许为校验规则提供额外的值。引文通常来说，校验并不是简单的无值或固定值校验，而是需要通过外间传递过来的值进行判断，<var>元素就提供了这个功能。
- <var>元素通过它本身的<var-name>和<var-value>两个子元素来传入校验需要值。

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 mybbs 数据库及 userinfo 表和 MyBBS 工程。（10 分钟）
2. 编写 register.jsp 页面。（15 分钟）
3. 编写 validation.xml 和 ApplicationResources.properties。（15 分钟）
4. 编写用于数据访问的 JdbcDao.java,UserinfoDao.java 类。（10 分钟）
5. 编写业务逻辑类 UserinfoService。（10 分钟）
6. 编写 InserUserinfoAction 和 InserUserinfoActionForm 类。（20 分钟）
7. 配置 struts-config.xml 文件，并调试运行程序。（10 分钟）

6. 作业

Struts 框架实现注册功能，并利用 Validatror 实现用户信息验证。

专题二 Hibernate

教学目标

理解 Hibernate 工作原理

掌握 Hibernate 核心 API

掌握复杂关联关系

掌握 Hibernate 高级查询

案例一 Hibernate 初探

1. 教学目标

1.1 理解 Hibernate 基本原理

1.2 了解编写实体映射文件

1.3 学会使用 Hibernate 完成数据增加的操作

1.4 掌握使用 MyEclipse Database Explorer 管理数据库

1.5 掌握使用 Myeclipse6.0 简化 Hibernate 开发

2. 工作任务

2.1 手工配置 Hibernate 框架。

2.2 使用 DB Brower 进行数据库管理。

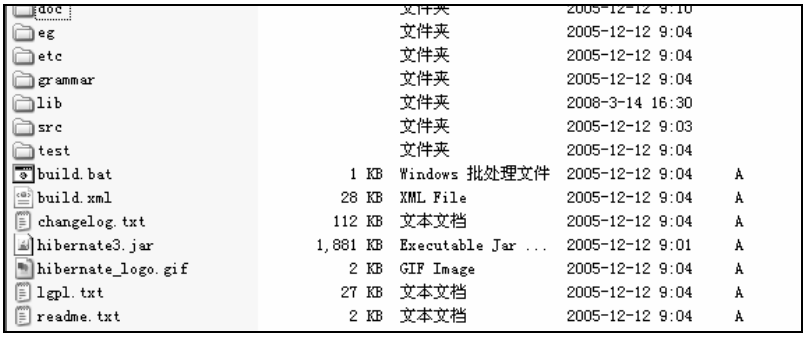
2.3 使用 Hibernate 框架实现论坛注册。

3. 相关实践知识

3.1 手工配置 Hibernate 框架

从 Hibernate 官方网站 www.hibernate.org 下载 Hibernate3.1 版本。

解压后看到的目录如下



doc		文件夹	2005-12-12 9:10	
eg		文件夹	2005-12-12 9:04	
etc		文件夹	2005-12-12 9:04	
grammar		文件夹	2005-12-12 9:04	
lib		文件夹	2008-3-14 16:30	
src		文件夹	2005-12-12 9:03	
test		文件夹	2005-12-12 9:04	
build.bat	1 KB	Windows 批处理文件	2005-12-12 9:04	A
build.xml	28 KB	XML File	2005-12-12 9:04	A
changelog.txt	112 KB	文本文档	2005-12-12 9:04	A
hibernate3.jar	1,881 KB	Executable Jar ...	2005-12-12 9:01	A
hibernate_logo.gif	2 KB	GIF Image	2005-12-12 9:04	A
lgpl.txt	27 KB	文本文档	2005-12-12 9:04	A
readme.txt	2 KB	文本文档	2005-12-12 9:04	A

图 2-1 Hibernate 解压目录

hibernate3.jar 是 Hibernate 的核心库，doc 目录是相关文档，eg 目录有相关示例，lib 是和 Hibernate 相关的库文件。打开 lib 目录，可以看到如下 jar 文件：

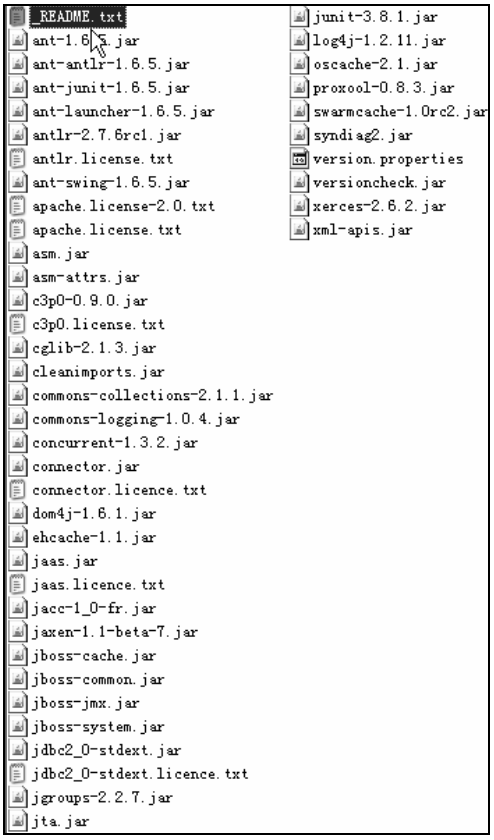


图 2-2 lib 目录下的 jar 文件

hibernate3.jar 和 lib 目录下的 jar 文件是开发中所需的类库。

打开 src 目录下的 org 目录到 hibernate 目录可以看到两个重要的文件：

hibernate-configuration-3.0.dtd　　这是 hibernate 配置文件 dtd 文档；

hibernate-mapping-3.0.dtd　　这是 hibernatd 映射文件 dtd 文档。

这两个文件是开发过程中编写配置文件及映射文件的依据。

1. 打开 Myeclipse，新建一个 Java Project 工程，工程名字为“hibernatedemo”。

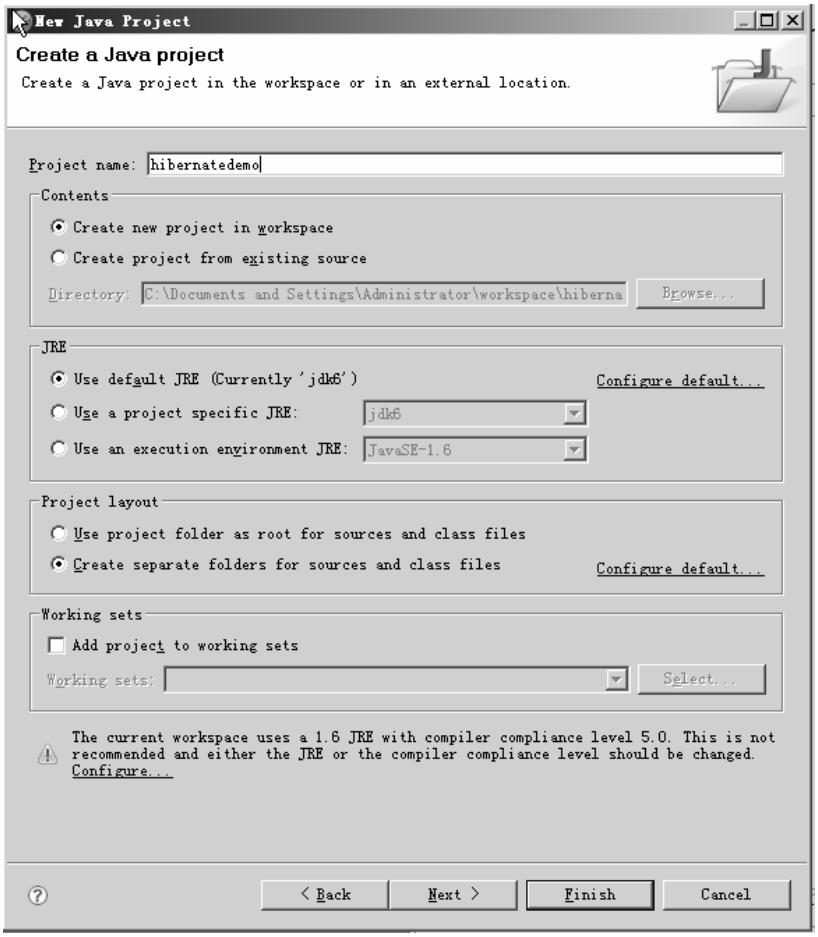


图 2-3 新建 hibernatedemo 工程

2. 右键工程名→选择【Properties】。

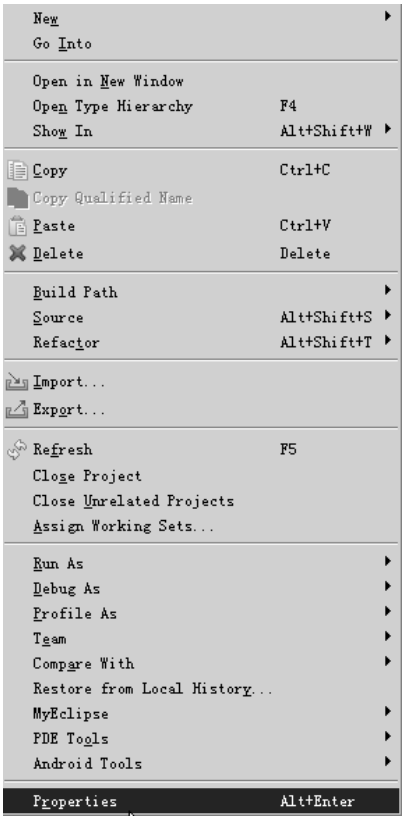


图 2-4 选择工程的 Properties 属性

3. 选择【Properties for hibernatedemo】对话框左边的【Java Build Path】选项，在对话框的右边会显示具体内容。选择【Libraries】选项卡，点击【Add External JARS】，添加 hibernate 的类库。如下图所示：

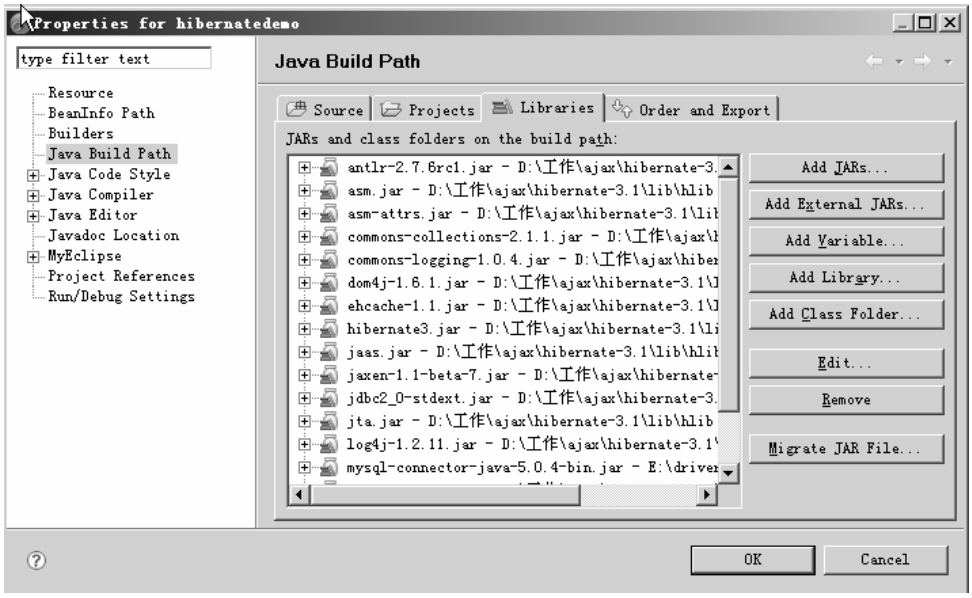


图 2-5 向 hibernatedemo 工程中添加相关类库

此处添加的 jar 包有 hibernate3.0.jar，hibernate 解压后 lib 目录下所有类库及 mysql 驱动程序。

4. 在 hibernate 工程 src 目录下，编写 hibernate 配置文件 hibernate.cfg.xml 文件。

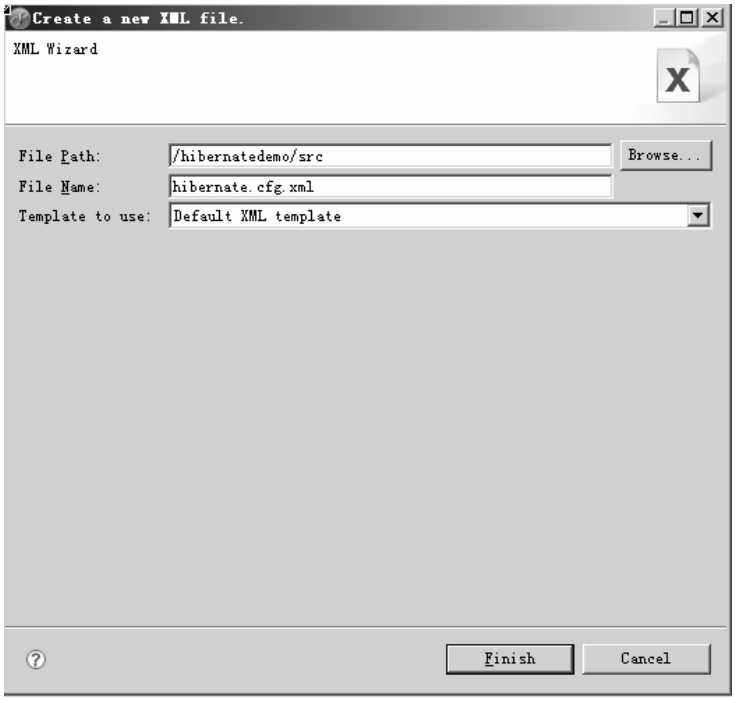


图 2-6 在 hibernatedemo 工程 src 目录下创建 hibernate.cfg.xml

打开 hibernate-configuration-3.0.dtd 文档，将

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

这段 dtd 文档的引入复制,此文件是编写 hibernate 配置文件的规范，规范了文件中出现的元素及元素间的关系。然后在 hibernate.cfg.xml 中粘贴 dtd 文档中复制的内容。

5. 根据 dtd 的规范进行编写 hibernate.cfg.xml。

```
  <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
    <hibernate-configuration>
       <session-factory>
         <property name="connection.username">root</property>
        <property name="connection.url">
            jdbc:mysql://localhost:3306/mybbs
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="connection.password">sa</property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <property name="show_sql">true</property>
              </session-factory>
    </hibernate-configuration>
```

6. 编写用户信息实体类。首先在工程中建三个包，包名分别为：com.mybbs.pojo；com.mybbs.dao；com.mybbs.test 如图所示

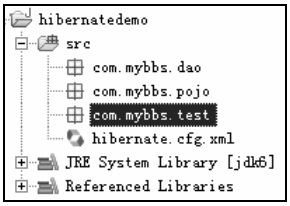


图 2-7 hibernatedemo 工程包结构

在 com.mybbs.pojo 包中编写 UserInfo 类，此类要实现 java.io.Serializable 接口：

```
package com.mybbs.pojo;
/**
 * 用户信息实体类
 * @author Administrator
 *
 */
public class UserInfo implements java.io.Serializable {
   //用户唯一标识
    private int userId;
    //用户名
    private String userName;
    //密码
    private String userPwd;
    //找回密码问题
    private String userQues;
    //找回密码答案
    private String userAns;
    //用户积分
    private int integral;
    //用户等级
    private String grade;
    //用户 email
    private String userMail;
    //用户性别
    private byte sex;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserPwd() {
        return userPwd;
    }

    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }
```

```
    public String getUserQues() {
        return userQues;
    }

    public void setUserQues(String userQues) {
        this.userQues = userQues;
    }

    public String getUserAns() {
        return userAns;
    }

    public void setUserAns(String userAns) {
        this.userAns = userAns;
    }

    public int getIntegral() {
        return integral;
    }

    public void setIntegral(int integral) {
        this.integral = integral;
    }

    public String getGrade() {
        return grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public String getUserMail() {
        return userMail;
    }

    public void setUserMail(String userMail) {
        this.userMail = userMail;
    }

    public byte getSex() {
        return sex;
```

```
    }

    public void setSex(byte sex) {
        this.sex = sex;
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }
}
```

要完成论坛注册，就是向 userinfo 表中添加一条记录。注意，此处的数据类型必须和数据库中对应的表数据类型保持一致。

7. 在 com.mybbs.pojo 包中编写 UserInfo.hbm.xml 映射文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
   Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
   <class name="com.mybbs.pojo.UserInfo" table="userinfo" catalog="mybbs">
      <id name="userId" type="java.lang.Integer">
         <column name="userid" />
         <generator class="native" />
      </id>
      <property name="userName" type="java.lang.String">
         <column name="username" length="20" not-null="true" />
      </property>
      <property name="userPwd" type="java.lang.String">
         <column name="userpwd" length="20" not-null="true" />
      </property>
      <property name="userQues" type="java.lang.String">
         <column name="userques" length="50" not-null="true" />
      </property>
      <property name="userAns" type="java.lang.String">
         <column name="userans" length="50" not-null="true" />
      </property>
```

```
        <property name="integral" type="java.lang.Integer">
            <column name="integral" not-null="true" />
        </property>
        <property name="grade" type="java.lang.String">
            <column name="grade" length="20" not-null="true" />
        </property>
        <property name="userMail" type="java.lang.String">
            <column name="useremail" length="20" not-null="true" />
        </property>
        <property name="sex" type="java.lang.Byte">
             <column name="sex" />
         </property>
    </class>
</hibernate-mapping>
```

打开 hibernate-mapping-3.0.dtd 文件，这是 hibernate 映射文件 dtd 规范。将

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN""http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

这段代码进行复制。在 UserInfo.hb.xml 文件中粘贴。

8. 在 hibernate.cfg.xml 文件中加入下面这段：

```
<mapping resource="com/mybbs/pojo/UserInfo.hbm.xml"/>
```

通过这段配置，将实体映射文件配置到 hibernate 配置文件中来。通过配置文件自动加载映射文件。

8. 在 com.mybbs.dao 中创建 UserInfoDAO 类。

```
package com.mybbs.dao;
import org.hibernate.cfg.Configuration;
import org.hibernate.*;
import com.mybbs.pojo.UserInfo;
public class UserInfoDAO {
    //添加一个用户
     public void save(UserInfo user){
     try{
     //第一步：通过 Configuration 类实例化自动加载 hibernate.cfg.xml
    Configuration cfg=new Configuration();
    //第二步：通过配置文件信息，得到 SessionFactory
    SessionFactory sf=cfg.configure().buildSessionFactory();
    //第三步：通过 SessionFactory 打开一个 Session
    Session s=sf.openSession();
    //第四步：通过 Session 开始事务
```

```
    Transaction tx=s.beginTransaction();
    //第五步：将用户保存到 Session（持久化操作）
    s.save(user);
    //第六步：提交事务
    tx.commit();
    //第七步：关闭 session
    s.close();
    }catch(Exception ex){
     ex.printStackTrace();
    }
    }
}
```

9. 在 com.mybbs.test 中创建 UserInfoTest 类，测试能够插入数据。

```
package com.mybbs.test;
/**
 * UserInfoDAO 测试类
 * @author Administrator
 *
 */
import com.mybbs.pojo.UserInfo;
import com.mybbs.dao.UserInfoDAO;
public class UserInfoTest {
    /**
     * @param args
     */
    public static void main(String[] args) {
         UserInfo user=new UserInfo();
        user.setUserName("mike");
        user.setUserPwd("mike23");
        user.setIntegral(0);
        user.setGrade("普通用户");
        user.setUserQues("你喜欢的球员是谁？");
        user.setUserAns("科比");

        user.setSex(Byte.parseByte("0"));
        user.setUserMail("mike@yahoo.com");
        UserInfoDAO uDao=new UserInfoDAO();
        uDao.save(user);
    }
}
```

执行后可以发现数据库中多一条记录，本次数据插入成功。

通过以上步骤的实施，发现手动配置 Hibernate 框架，需要三个准备，七个步骤。

准备一、导入 Hibernate 库

准备二、添加配置文件 hibernate.cfg.xml

准备三、添加映射文件 UserInfo.hbm.xml

步骤一、创建 Configuration

步骤二、创建 SessionFactory

步骤三、打开 Session

步骤四、开始一个事务

步骤五、持久化操作

步骤六、提交事务

步骤七、关闭 Session

通过上面的学习我们对于 Hibernate 有了更深层次的了解，但我们也不难发现在做 hibernatedemo 示例中，配置 hibernate 很麻烦，有没有简单的方法呢？有的，下面就让我们一起体验一下 Myeclipse6.0 开发 Hibernate 工程的便捷之旅。

3.2 使用 DB Brower 进行数据库管理

1. 在 Myeclipse 菜单栏中选 Window→Open Perpective→MyEclipse Database Explorer。

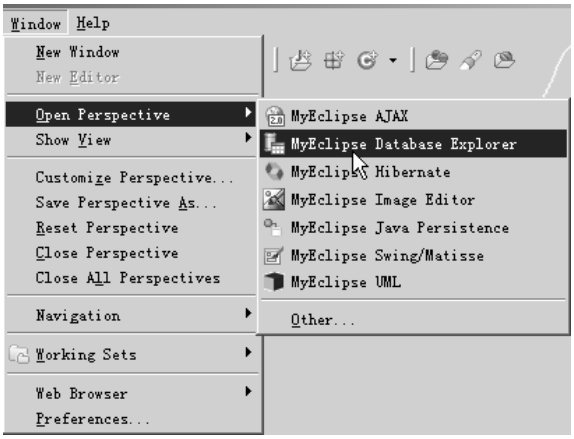


图 2-8 打开 MyEclipse Database Explorer 视图

而后就会到 MyEclipse Database Explorer 视图下

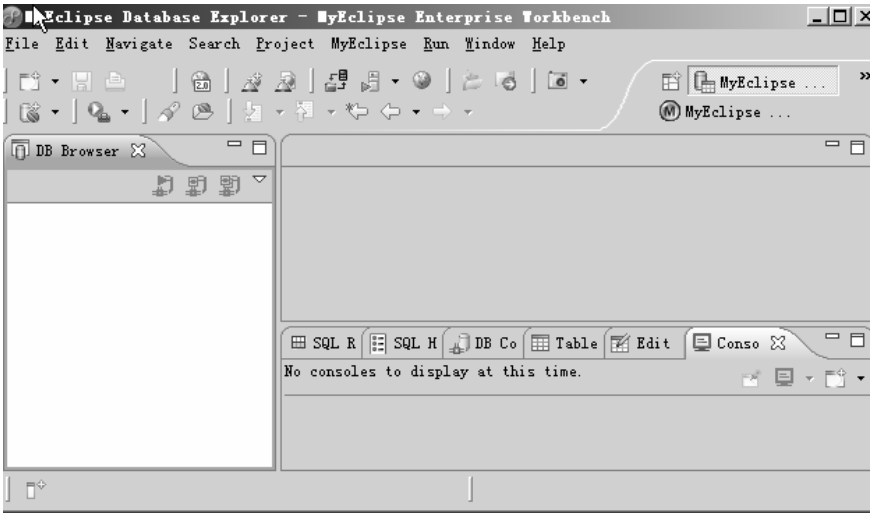


图 2-9 MyEclipse Database Explorer 视图

2. 建立一个数据库连接，在【DB Brower】空白处右键→new。

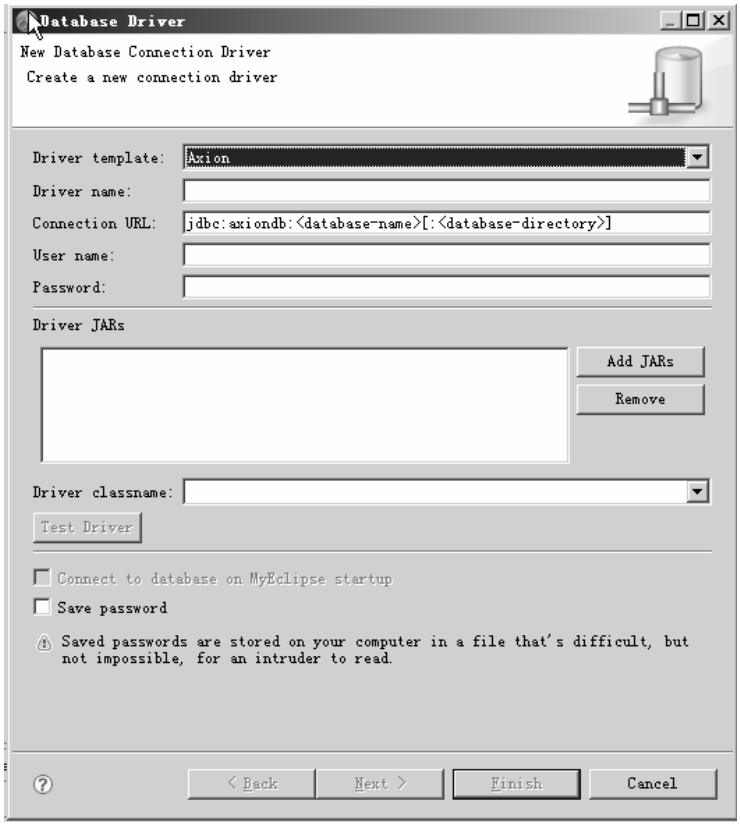


图 2-10 建立数据库连接视图

- Driver template 选择驱动模板选择要连接的数据库
- Driver name 给指定数据库连接取名字
- Connection Url 数据库的连接字符串
- User Name 数据库用户名
- Password 数据库密码
- Driver JARs 添加使用的数据库驱动

这里配置 mysql 数据库，配置后如图所示：

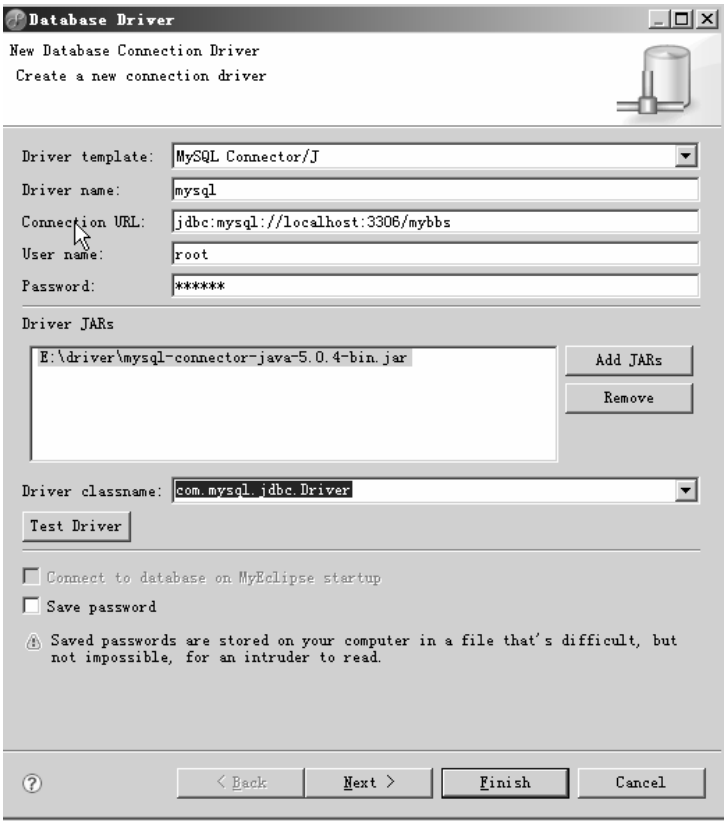


图 2-11 配置数据库连接

3. 点击【Next】显示数据库相关内容。

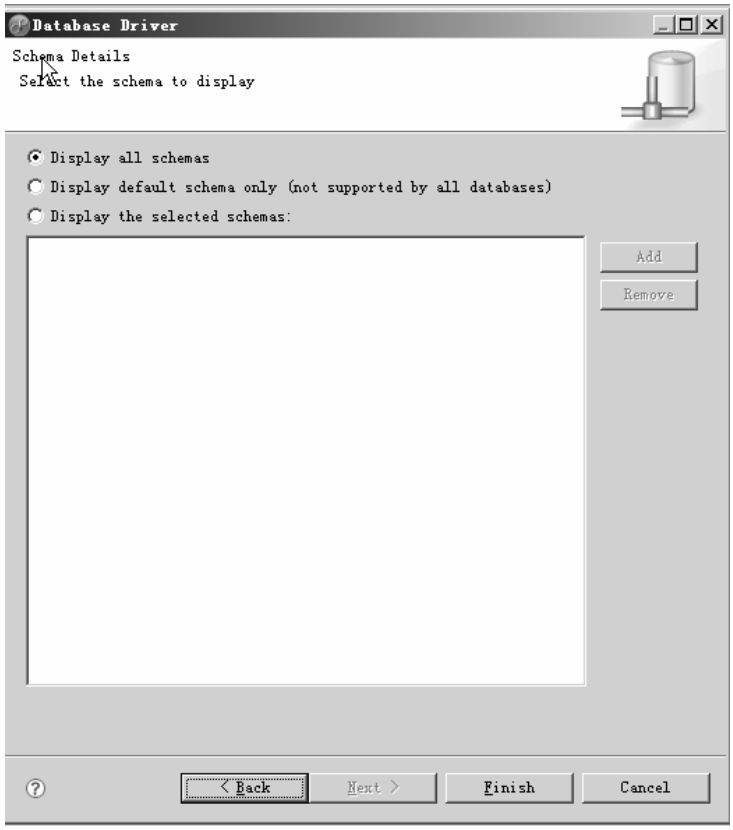


图 2-12 显示数据库相关信息

选择默认选项，显示所有的 Schemas，即显示数据库所有的信息。

4. 显示数据库连接。点击【Finish】后，在【DB Brower】视图区如下：

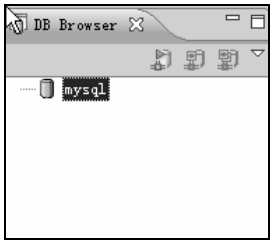


图 2-13 显示配置的数据库连接

5. 打开配置好的数据库连接。右键 mysql 图标。

图 2-14 打开配置好的数据库连接

6. 选择【Open Connection】选项，显示连接数据库的用户名和密码选项。

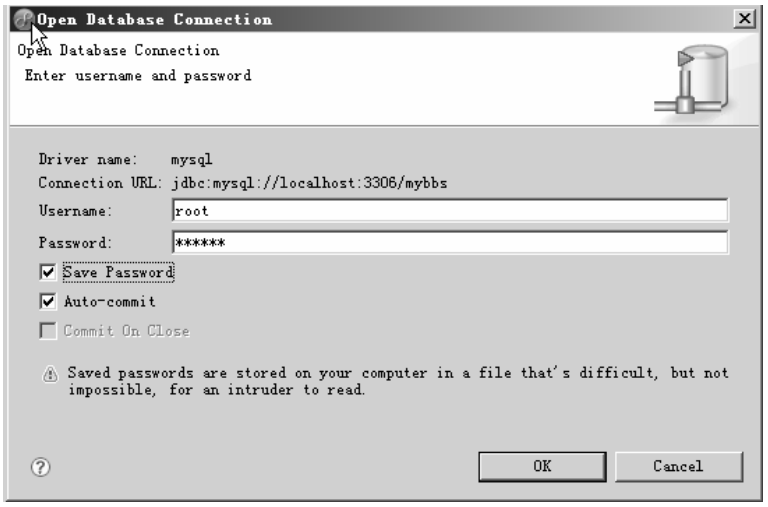


图 2-15 显示连接数据库的用户名密码

【Save Password】选项选中后，在以后操作过程中可以不再输入密码。

7. 选择【OK】，显示当前连接数据库的所有数据库信息。

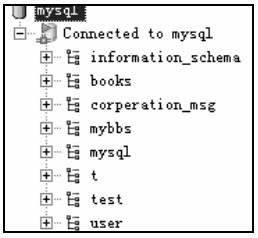


图 2-16 显示连接数据库的所有数据库信息

8. 选择 mybbs 数据库，看到 mybbs 中所有表的信息。

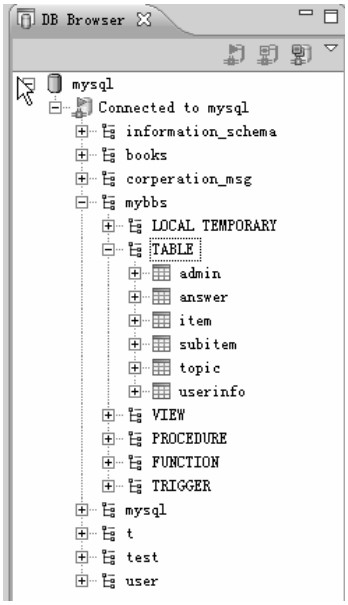


图 2-17 显示 mybbs 数据库所有的表

在 MyEclipse Database Explorer 视图中，很方便、快捷的进行建库，建表，查看记录等等相关操作，此项不是本书重点，在此不做累述。

3.3 使用 Hibernate 框架实现论坛注册

1. 导入 struts 案例四的 mybbs 工程。

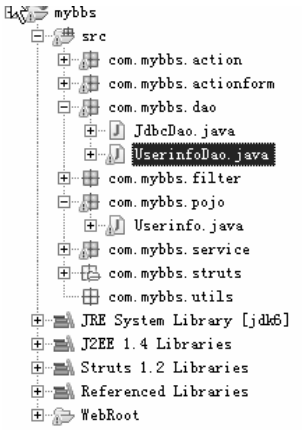


图 2-18 导入 struts 案例四的 mybbs 工程

然后将 dao、pojo 包内的类删除掉，相关引用地方进行注释。

2. 然后像加入 struts 那样加入 Hibernate 扩展。

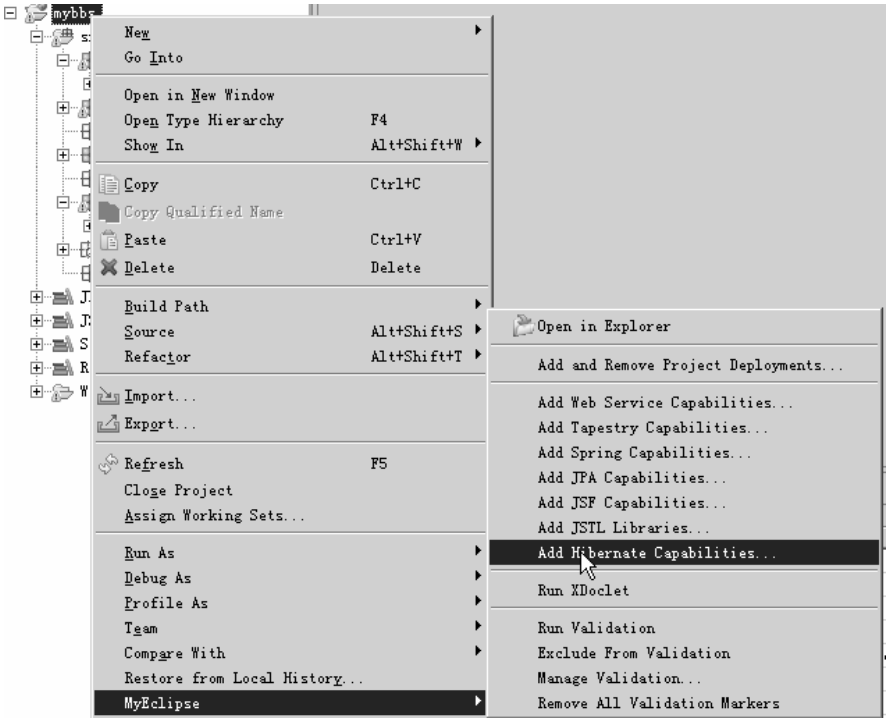


图 2-19 向 mybbs 工程加入 Hibernate 扩展

3. 现在请你选择 Hibernate 框架版本。

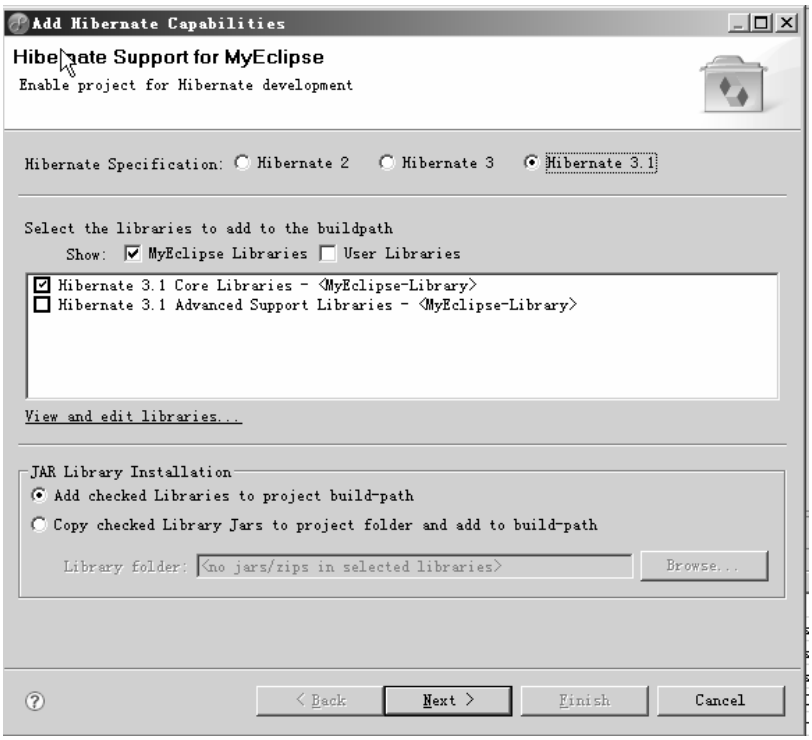


图 2-20 选择要加入 Hibernate 框架的版本信息

- Hibernate Specification：要添加到项目中的 Hibernate 版本支持功能。为了最大限度的利用 MyEclipse Hibernate 工具，推荐 Hibernate3.1。
- MyEclipse Libraries/User Libraries：可以添加到你的项目的构造路径的类库集合。一般选择 MyEclipse Libraries。
- Hibernate3.1 Core Libraries：Hibernate 的核心库。这是必须的。
- Hibernate3.1 Advanced Support Libraries：Hibernate 的扩展库，可选。
- Jar Library Installation 要把相关类库放在什么地方。
- Add checked Libraries to project build-path：JAR 文件将不会复制到你的项目中，这些 JAR 文件将在发布程序时复制。
- Copy checked Library Jars to project folder and add to build-path：选中的类库 JAR 文件将会被复制到你的项目并添加到构造路径中去（这个方式在开发不依赖于 MyEclipse 的项目的时候，或者解决 JAR 包冲突的时候很有用）。为了防止冲突，推荐选用这种方式。
- Library Folder：相对于现在项目的路径，可以新建或者使用现有目录，Hibernate 类库将会被向导复制到这里。推荐放在/WebRoot/WEB-INF/lib 目录下。

4. 选择【Next】，生成 Hibernate 配置文件。

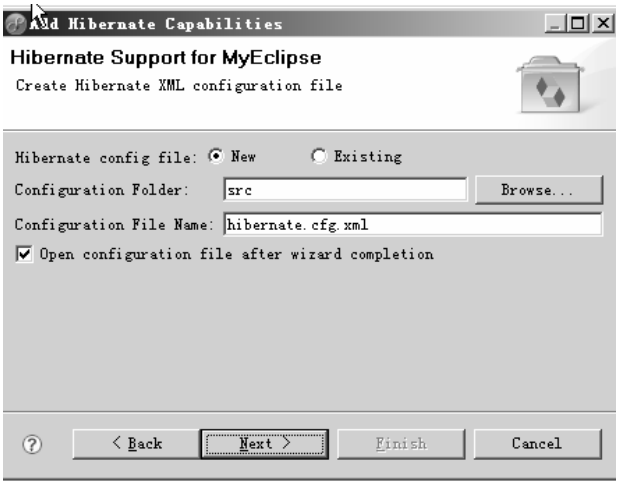


图 2-21 生成 Hibernate 配置文件

如果是新项目保持默认设置然后点击【Next】按钮就可以了。反过来如果以前有存在的 Hibernate 配置文件的话，可以点击选中【Existing】单选钮后选择现有 Hibernate 配置文件的路径即可，然后点击【Next】按钮。

5. 选择【Next】，设置 Hibernate 的连接属性。

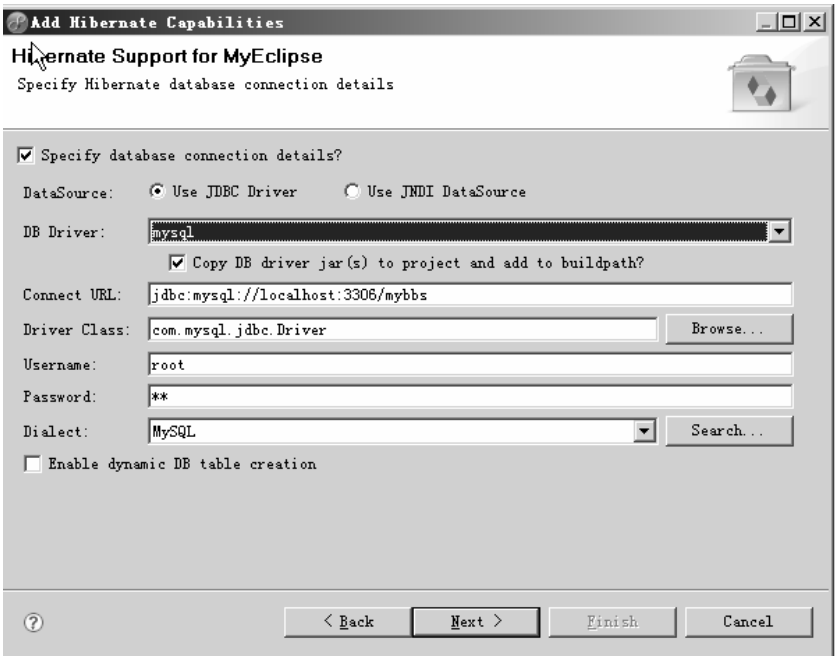


图 2-22 配置 Hibernate 连接属性

选择【DB Driver】右侧的现有数据库连接列表，选择以前在 MyEclipse Database Explore 创建好的数据库连接例如 mysql。这时候相关的连接信息将会自动填入到对话框中，对应的 Hibernate 方言也会选择好（Dialect）右侧的下拉列表可以选择一种 Hibernate 所支持的方言），

当然也可以根据情况手工调整这些输入框的值。

复选框 Copy DB driver jar(s) to project and add to buildpath

选中后则会自动加入相应的数据库驱动类库 jar 文件到项目的类路径中。之后选择【Next】按钮即可。

注意：如果你不想现在就设置数据库连接属性，去掉 Specify database connection details？前面的复选框即可跳过。

注意：Enable dynamic DB table creation 复选框如果选中，那么 Hibernate 将会自动根据映射文件来动态生成建表语句然后执行，然而这种方法不是很可靠，所以一般来说不要用。

6. 选择【Next】，创建工厂类。

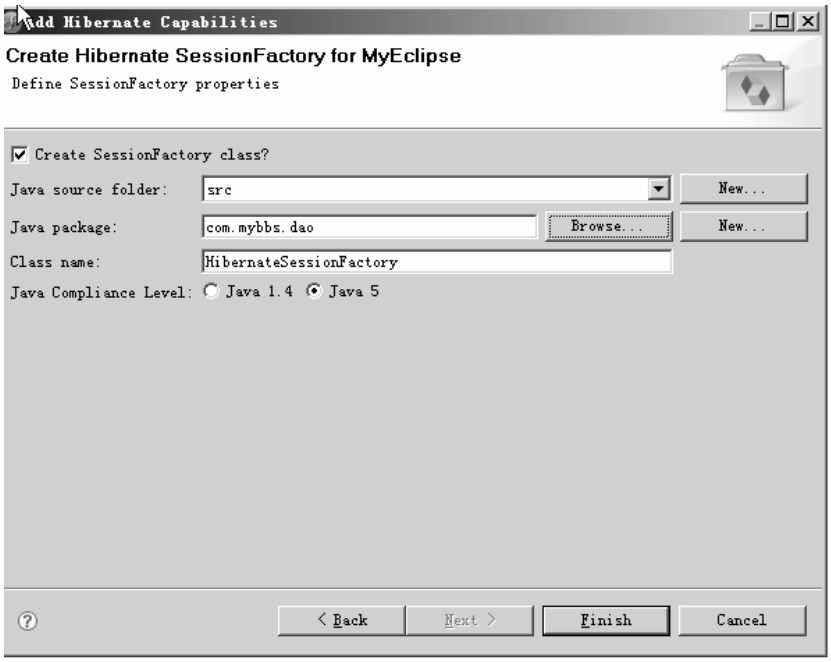


图 2-23 生成工厂类

这一步骤可选，不创建也是可以的。现在我们创建一个 SessionFactory，放到 com.mybbs.dao 中去。注意，工厂类一般不要放在工程的 src 目录下，最好指定一个包名。

7. 选择【Finish】，完成了 Hibernate 框架的配置工作。

现在可以看到 hibernate.cfg.xml 已经自动生成。

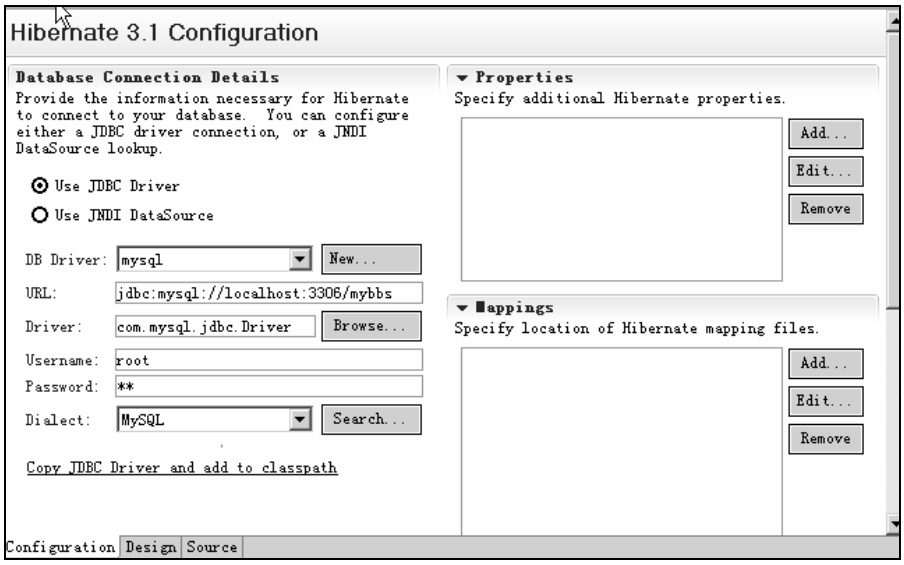


图 2-24 hibernate.cfg.xml 设计视图

我们可以看到配置视图，可以通过可视化的界面进行 Hibernate 文件配置，而且它还有 Design 视图和 Source 视图。打开 Source 视图，可以看到：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<!-- Generated by MyEclipse Hibernate Tools.                 -->
<hibernate-configuration>

   <session-factory>
      <property name="connection.username">root</property>
      <property
name="connection.url">jdbc:mysql://localhost:3306/mybbs</property>
      <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="myeclipse.connection.profile">mysql</property>
      <property name="connection.password">sa</property>
      <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>

   </session-factory>
</hibernate-configuration>
```

与我们手工写的配置文件内容一样，但这样做可以简化开发过程，缩短开发时间。

现在打开【Refernced Libraries】目录，发现 Hibernate 相关的库文件自动被引入。

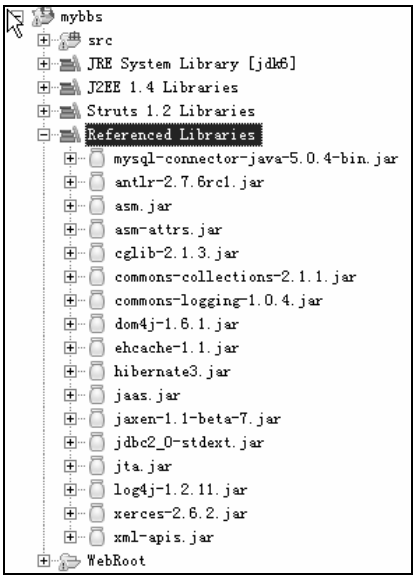


图 2-25 mybbs 工程【Refernced Libraries】目录

这样对于我们开发者来说，又少了一些工作。

7. 切换到 MyEclipse Database Explorer 视图，选择 mybbs 数据库找到要操作 userinfo 表。然后右键表名，选择【Hibernate Reverse Engineering】翻转引擎。

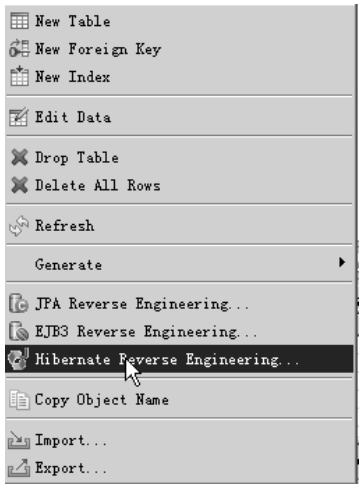


图 2-26 选择 Hibernate Reverse Engineering 翻转引擎

8. 【Hibernate Reverse Engineering】设置。

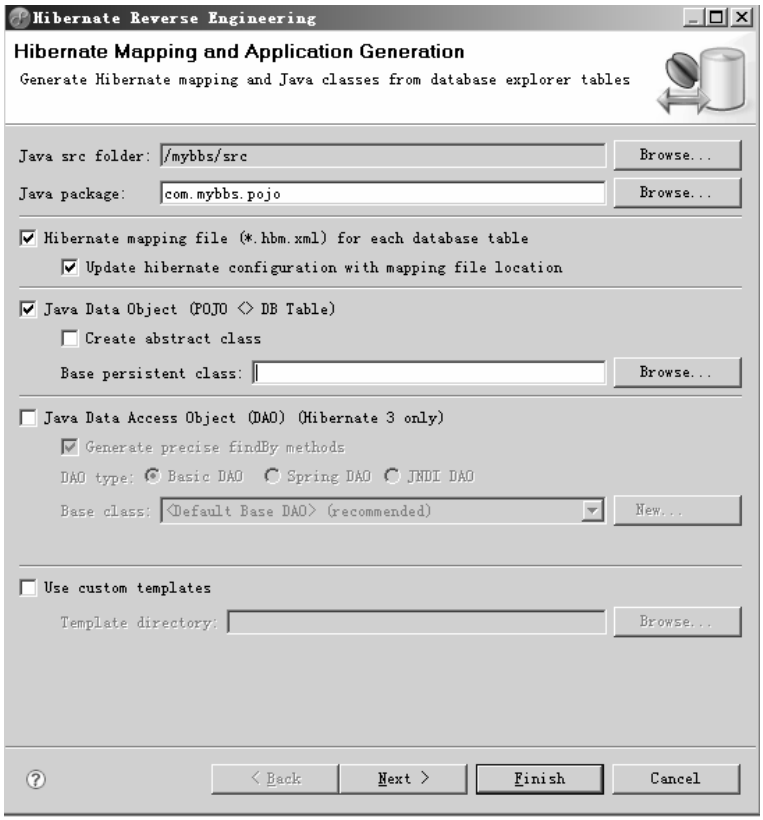


图 2-27【Hibernate Reverse Engineering】设置

【Java src folder】最右侧的【Browse】按钮，可以查看可用的 Hibernate 项目以及源码目录，这些目录将用来存放最终生成的文件。这里选中 mybbs/src。

【Java package】输入框右侧的【Browse】按钮，选中 com.mybbs.pojo 包，或者新建一个其它的包来存放生成的代码及文件。

【Hibernate mapping file(*.hbm.xml) for each database table】

【Update hibernate configuration with mapping file location】

这两个复选框选中，这样将为每个数据库表生成 Hibernate 映射文件（*.hbm.xml），并在 hibernate.cfg.xml 中将新生成的映射文件加入。

【Java Data Object（POJO<> DB Table）】

选中复选框【Java Data Object（POJO <> DB Table）】，这样为映射文件和表格生成对应的数据对象（POJO）。

【Create abstract class】是创建一个抽象类，一般我们不用创建。

【Java Data Access Object(DAO)(Hibernate 3 only)】这样将能生成普通的 DAO 类。

注意：只有使用 Hibernate 3 才能生成便于访问映射后类的数据访问对象。

注意：Spring DAO 这个单选钮是灰色，如果可用则能生成 Spring＋Hibernate 的 DAO。

最后，为了简化，直接点击 Finish 按钮就可以结束代码的生成。只需轻轻点击几下鼠标，就生成了 Hibernate 实体类，映射文件。

打开 com.mybbs.pojo 包可以看到自动生成的实体类。

```
package com.mybbs.pojo;

/**
 * Userinfo entity.
 *
 * @author MyEclipse Persistence Tools
 */

public class Userinfo implements java.io.Serializable {

    // Fields

    private Integer userid;
    private String username;
    private String userpwd;
    private String userques;
    private String userans;
    private Integer integral;
    private String grade;
    private String useremail;
    private Byte sex;

    // Constructors

    /** default constructor */
    public Userinfo() {
    }

    /** minimal constructor */
    public Userinfo(Integer userid, String username, String userpwd,
            String userques, String userans, Integer integral, String grade,
            String useremail) {
        this.userid = userid;
        this.username = username;
        this.userpwd = userpwd;
        this.userques = userques;
        this.userans = userans;
        this.integral = integral;
```

```
        this.grade = grade;
        this.useremail = useremail;
    }
    /** full constructor */
    public Userinfo(Integer userid, String username, String userpwd,
            String userques, String userans, Integer integral, String grade,
            String useremail, Byte sex) {
        this.userid = userid;
        this.username = username;
        this.userpwd = userpwd;
        this.userques = userques;
        this.userans = userans;
        this.integral = integral;
        this.grade = grade;
        this.useremail = useremail;
        this.sex = sex;
    }
    // Property accessors
    public Integer getUserid() {
        return this.userid;
    }
    public void setUserid(Integer userid) {
        this.userid = userid;
    }
    public String getUsername() {
        return this.username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getUserpwd() {
        return this.userpwd;
    }
    public void setUserpwd(String userpwd) {
        this.userpwd = userpwd;
    }
    public String getUserques() {
        return this.userques;
    }
    public void setUserques(String userques) {
        this.userques = userques;
    }
```

```
    public String getUserans() {
        return this.userans;
    }
    public void setUserans(String userans) {
        this.userans = userans;
    }
    public Integer getIntegral() {
        return this.integral;
    }
    public void setIntegral(Integer integral) {
        this.integral = integral;
    }
    public String getGrade() {
        return this.grade;
    }
    public void setGrade(String grade) {
        this.grade = grade;
    }
    public String getUseremail() {
        return this.useremail;
    }
    public void setUseremail(String useremail) {
        this.useremail = useremail;
    }
    public Byte getSex() {
        return this.sex;
    }
    public void setSex(Byte sex) {
        this.sex = sex;
    }
```

同时也可以看到生成的 Userinfo.hbm.xml 文件。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.mybbs.pojo.Userinfo" table="userinfo" catalog="mybbs">
      <!—注意主键的生成策略要手动修改，默认是 assigned.因为数据表字段是自增的，改成最常用
的 native。-->
```

```
<id name="userid" type="java.lang.Integer">
          <column name="userid" />
          <generator class="native" />
      </id>
      <property name="username" type="java.lang.String">
          <column name="username" length="20" not-null="true" />
      </property>
      <property name="userpwd" type="java.lang.String">
          <column name="userpwd" length="20" not-null="true" />
      </property>
      <property name="userques" type="java.lang.String">
          <column name="userques" length="50" not-null="true" />
      </property>
      <property name="userans" type="java.lang.String">
          <column name="userans" length="50" not-null="true" />
      </property>
      <property name="integral" type="java.lang.Integer">
          <column name="integral" not-null="true" />
      </property>
      <property name="grade" type="java.lang.String">
          <column name="grade" length="20" not-null="true" />
      </property>
      <property name="useremail" type="java.lang.String">
          <column name="useremail" length="20" not-null="true" />
      </property>
      <property name="sex" type="java.lang.Byte">
          <column name="sex" />
      </property>
   </class>
</hibernate-mapping>
```

同时在 hibernate.cfg.xml 中自动添加映射。

```
<mapping resource="com/mybbs/pojo/Userinfo.hbm.xml" />
```

如此简单就完成了相关的配置，开发人员只需根据业务进行 DAO 类的编写即可以了

9. 根据业务需要编写 UserinfoDAO 类。

```
import  com.mybbs.pojo.Userinfo;
import org.hibernate.*;
/**
 * 对用户进行操作业务逻辑
 * @author Administrator
 *
```

```
 */
public class UserinfoDAO {
  //添加一个注册用户
   public void insertUserinfo(Userinfo user){
      //通过 session 工厂得到一个 session
    Session s=HibernateSessionFactory.getSession();
      //开始事务
    Transaction tx=s.beginTransaction();
      //进行持久化操作
    s.save(user);
      //提交事务
    tx.commit();
    //关闭 session
    s.close();
   }
}
```

然后在 action 中进行调用即可。这样在我们实际编写代码的过程中只需关注 DAO 的编写，其他的方面，编译器自动帮我们完成，大大加快了开发的速度和效率。

4. 相关理论知识

4.1 什么是 Hibernate，为什么要用 Hibernate

什么是 Hibernate？

Hibernate 是一个 JDO（Java Data Objects）工具。它的工作原理是通过文件把值对象和数据库表之间建立起一个映射关系。这样只需要通过操作这些值对象和 Hibernate 提供的一些基本类，就可以达到使用数据库的目的。例如，Hibernate 的查询，可以直接返回包含某个值对象的列表，而不必向传统的 JDBC 访问方式一样把结果集的数据逐一装载到一个值对象中，为编码节省了大量时间。另外，Hibernate 提供的 HQL 是一种类 SQL 语言，它提供对象化的数据查询方式，并且 HQL 在功能上和使用方式上都非常接近标准的 SQL。

Hibernate 的作用就是介于 Java 与 JDBC 之间的一个持久层，它通过建立与数据库表之间的映射来操纵数据库。

什么是持久层呢？在了解这个概念之前需要搞清楚以下三个概念：

瞬时状态：保存在内存的程序数据，程序退出后，数据就消失了，称为瞬时状态。

持久状态：保存在磁盘上的程序数据，程序退出后依然存在，称为程序数据的持久状态。

持久化：将程序数据在瞬时状态和持久状态之间转换的机制。

持久层就是把数据库实现当作一个独立逻辑拿出来，即数据库程序是在内存中的，为了使程序运行结束后状态得以保存，就要保存到数据库。持久层是在系统逻辑的层面上，专注于实现数据库持久化的一个相对独立的单元。

为什么要使用 Hibernate？

要解决这个问题必须了解操作数据库的三个阶段：

操作 JDBC 阶段

我们可以回顾一下 struts 讲解中的相关程序。

在使用 Java 进行数据库开发的最初阶段，都是使用 JDBC 来操作数据库的。我们经常会写这样的代码：

```
public class Test {
  public void getConn(){
      try{
          Class.forName(drivername);
          Connection conn=DriverManager.getConnection(url);
          Statement stm=conn.createStatement();
          ResultSet rs=stm.executeQuery(sql);
          if(rs.next()){}
          rs.close();
          stm.close();
          conn.close();
      }catch(Exception ex){
          ex.printStackTrace();
      }
  }
}
```

这段操作可以称为经典的 JDBC 数据库操作流程：

加载数据库驱动。

创建数据库连接。

创建申明对象。

执行查询或更新。

关闭结果集对象、申明对象、数据库连接对象。

同时要捕获异常信息。

在这段代码中存在两个问题：

代码过度重复：每一次数据库操作时都要编写这些代码，而代码基本相似。

不能突出业务：即这个代码都是数据库处理的代码，体现不出业务逻辑。

封装 JDBC 阶段

为了改进这些不足。我们会经常编写 DBAccess 类，将相关操作进行封装。

Public boolean createConn()：加载数据库驱动程序，创建数据库连接对象；

Public boolean update(String sql)：执行更新语句；

Public void query(String sql)：执行查询语句等等。

这样通过调用本类中相关的方法就可以完成相关的操作，实现了对 JDBC

的封装，让开发者只需注意业务逻辑。

ORM 阶段

在对 JDBC 进行封装之后，能够方便的实现数据库的操作。但是，在面向对象编程开发中，数据库的操作与普通的面向对象编程是两种不同的思路。能不能再进行改进，使得操作数据库就像操作普通的 Java 类一样呢？于是 ORM 应运而生。

ORM 是 Objiect Relational Mapping 的简称，即对象关系映射。它是为了解决面向对象与关系数据库存在的互相不匹配的现象的技术。简单的说，ORM 是通过使用描述对象和数据库之间映射的原数据，将 Java 程序中的对象自动持久化到关系数据库中，本质上就是将数据从一种形式转换为另一种形式。

我们通过 mybbs 项目的管理员表来剖析一下用户实体、数据表、Java 类三者之间的映射关系。

数据实体的 3 种不同的表示形式，即数据实体，数据表，映射对象。此三者的详细解释如下：

批注 [w2]: 删除一段

用户实体是数据库概念设计阶段的产物，表示在系统设计的最初阶段所抽象的基本数据对象。每一个数据实体都包含一些字段信息。

将数据实体映射为数据表，将实体的字段信息映射为数据表字段，就形成了数据库中的可表示的对象-数据表。

映射对象是普通的 Java 类，它用 Java 类来代表数据实体，类名和表名对应，变量与表的字段对应，ORM 就是数据表和映射对象之间的映射关系。

由此可见，ORM 实现了数据表到 Java 对象的映射，这正是 ORM 的作用。ORM 是随着面向对象的软件开发方法而产生的。面向对象的开发方法是当今企业级应用开发环境主流的开发方法，关系型数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和数据表是业务实体的两种不同表现形式，业务实体在内存中表现为对象，在数据库中表现为数据表。内存中的对象之间存在关联和继承关系，而在数据库中无法直接表达多对多关联和继承关系。因此，对象—关系映射（ORM）系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

几乎所有的程序里面，都存在对象和关系数据库。在业务逻辑层和用户界面层中，都是面向对象的。当对象信息发生变化的时候，我们需要把对象信息保存在关系数据库中。在传统的开发过程中，我们会写很多数据访问层代码及大量的 SQL 语句，用来对数据库进行操作。但当引入 ORM 对象后，只需要用 O/R Mapping 保存、删除、读取对象，由 O/R Mapping 负责生成 SQL，我们只需要关注对象就可以了。这就实现了数据库到面向对象编程的完美过渡。

对象关系映射成功运用在不同的面向对象持久层的产品中，一般 ORM 包括以下 4 个部分：

一个对持久类进行增、删、改、查操作的 API。

一个语言或 API 用来规定类和类属性相关的查询。

一个规定 mapping metadata 工具。

一种可以让 ORM 的实现同事务对象一起进行 dirty checking,lazy associating fetching 及其他优化操作。

目前众多厂商和开源社区都提供了 ORM 的实现，目前主要的 ORM 产品如下：

Apache OJB http://db.apache.org/ojb/

Hibernate http://www.hibernate.org

IBatis http://www.ibatic.com

TopLink http://otn.oracle.com/products/ias/toplink/index.html 等等。其中 Hiberante 的轻量级 ORM 模型在 Java ORM 架构中逐步确立了主导地位。

知识点：

Hibernate 是一个主流的持久化框架，是一个优秀的 ORM（对象—关系映射）机制。

通过这个框架，使用这种机制我们在开发的过程中更多的关注业务逻辑及对象的层面，简化对数据库层面的关注，加快开发速度和效率。

ORM 是 Objiect Relational Mapping 的简称，即对象—关系映射。它一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射，实现了数据库到面向对象编程的完美过渡。

4.2 hibernate.cfg.xml 文件元素分析

<hibernate-configuration>是文件的根元素，所有相关的配置都要在此元素下配置。

<session-factory>session 工厂配置，相当于 JDBC 中的 DriverManager。

<property name="connection.username">root</property>指定连接数据库的用户名。

<property name="connection.password">922526</property>指定连接数据库的密码。

<property name="connection.driver_class">com.mysql.jdbc.Driver</property>指定连接数据库的驱动程序。

<property name="connection.url">jdbc:mysql://localhost:3306/mybbs</property>指定连接数据库 URL。

<property name="dialect">org.hibernate.dialect.MySQLDialect</property>指定数据库使用的 SQL 方言。

Hibernate 能够访问多种数据库，如 MySQL、Oracle 和 Sybase 等。尽管多数数据库都支持标准的 SQL 语言，但是它们往往还有各自的 SQL 方言，就像不同地区的人既能说标准普通话，还能将各自的方言一样。

<property name="show_sql">true</property>显示 SQL 语句

4.3 UserInfo.hbm.xml 文件元素分析

UserInfo.hbm.xml 作用是告诉 Hibernate 怎么来做对象映射。向哪个表插入数据，每个属性的数据类型，以及对应数据表里的列名。

<hibernate-mapping>元素是文件的根元素所有的配置信息都要在此元素下进行配置。

```
<class name="com.mybbs.pojo.UserInfo" table="userinfo" catalog="mybb />
```

<class>元素表示要和哪个实体对象进行映射，在一个 hbm.xml 文件中可以配置多个实体，但是一般来说为每一个 hbm.xml 中只配置一个实体信息。

Name 属性：表示要映射的实体，注意必须标明在那个包中。

Table 属性：表示和数据库中那个表进行映射。

Catalog 属性：数据库的名字。

```
<id name="userId" type="java.lang.Integer">
       <column name="userid" />
       <generator class="native" />
   </id>
```

<id>元素：主键字段配置，hibernate 为我们生成主键 id，必须定义。

name 属性：对应实体的 userId 字段，主键字段。

type 属性：本字段的数据类型。此处有两种写法。一种是 java.lang.Integer 使用 Java 的数据类型，另外一种使用 Hibernate 的映射类型的写法，如：Hibernate type: string。一般采取 Java 数据类型写法。下面将 Hibernate 映射类型，对应的 Java 类型及对应标准 SQL 类型进行匹配。

Hibernate 映射类型	Java 类型	标准 SQL 类型
Integer/int	Java.lang.Integer/int	INTEGER
Long	Java.lang.Long/long	BIGINT
Short	Java.lang.Short/short	SMALLINT
Byte	Java.lang.Byte/byte	TINYNIT
Float	Java.lang.Float/float	FLOAT
Double	Java.lang.Double/double	DOUBLE
Big_decimal	Java.math.BigDecimal	NUMERIC
Character	Java.lang.Character/java.lang.String	CHAR
String	Java.lang.String	VARCHAR
Boolean/yes_no/true_false	Java.lang.Boolean/Boolean	BIT
Date	Java.uitl.Date/java.sql.Date	DATE
Timestamp	Java.uitl.Date/java.uitl.Timestamp	TIMESTAMP
Calendar	Java.uitl.Calendar	TIMESTAMP
Calendar_date	Java.uitl.Calendar	DATE
Binary	Baty[]	BLOB
Text	Java.lang.String	TEXT
Serializable	实现 java.io.Serializable 接口的 java 类	BLOB
Clob	Java.sql.Clob	CLOB
Blob	Java.sql.Blob	BLOB
Class	Java.lang.Class	VARCHAR
Locale	Java.util.Locale	VARCHAR

Timezone	Java.util.TimeZone	VARCHAR
Currency	Java.util.Currency	VARCHAR

<column>元素：与表中哪个字段形成映射。

<generator>元素：表示主键的生成策略。

Class 属性：主键策略生成方式。生成方式有以下几种：

1. assigned

主键由外部程序负责生成，无需 Hibernate 参与，因此需要应用程序在执行保存之前为对象分配一个标识符。这是<generator>元素没有指定时的默认生成策略。

2. hilo

使用 hi/lo 高/低位算法高效的生成 long，short 或 int 类型的标识符。高/低位算法生成的标识符只在一个特定的数据库中是唯一的，需要额外的数据库表或字段提供高位的值来源。

3. seqhilo

与 hilo 类似，通过 hi/lo 算法实现的主键生成机制，只是主键历史状态在 sequence 中，适合支持 sequence 的数据库，如 Oracle。

4. increment

increment 标识主键按数值顺序递增。此方式的实现机制为，在当前应用实例中维持一个变量，以保存当前的最大值，之后每次需要生成主键的时候将此值加 1 作为主键。

注意：这种方式存在的问题是如果当前有多个实例访问同一个数据库，那么由于各个实例各自维护主键的状态，不同实例可能生成相同的主键，从而造成主键重复异常。因此，如果同一个数据库有多个实例访问，此方式避免使用。

5. identity

Identity 采用数据库提供的主键生成机制，如 DB2，MySQL，Sybase，Ms SQL 中的主键生成机制，这需要这些数据库内部支持标识字段。

6. sequence

sequence 采用数据库提供的 sequence 机制生成主键，如 Oralce 中的 sequence。

7. native

此方式由 Hibernate 根据底层数据库自行判断采用 identity，hilo 和 sequence 其中一种作为主键生成方式。此方式常用。

8. uuid.hex

由 Hibernate 基于 128 位唯一产生算法 UUID 生成十六进制数值（编码后以长度 32 的字符串表示）作为主键。

9. uuid.string

与 uuid.hex 类似，只是生成的主键未进行编码。这种方式在某些数据库中可能存在问题（如 PostgreSQL）。

10. foreign

使用另外一个相关联的对象的标识符。通常和<one-to-one>联合使用。

11. select

通过数据库触发器选择一些唯一主键的行并返回主键值来分配一个主键。

```
<property name="userName" type="java.lang.String">
<column name="username" length="20" not-null="true" />
</property>
```

<property>元素定义了该实体的属性

Type 属性：属性的类型。

<column>元素：对应数据库字段。

Name 属性：数据库字段名字。

Length 属性：字段长度。

Not-null 属性：字段是否可以为空。True 代表不能为空，false 代表可以为空。

4.4 知识扩展

Hibernate 之父：Gavin King

JBoss 核心成员之一

EJB3.0 专家委员会成员

《Hibernate In Action》作者

2001 年开始开发 Hibernate

2003 年 Hibernate 发展为 Java 世界主流持久层框架。

Gavin King 1974 年出生于澳大利亚，现在居住在澳大利亚默尔本市。Hibernate 是 Java 平台上的一种流行的、容易使用的开放源代码对象关系（OR）映射框架。Hibernate 诞生在 2001 年 11 月，在短短的两年多时间就发展成为 Java 世界主流的持久层框架软件，令人侧目。Hibernate 是一个雄心勃勃的项目，它的目标是成为 Java 中管理持续性数据问题的一种完整的解决方案。它协调应用与关系数据库的交互，让开发者解放出来专注于手边的业务问题。Hibernate 是一种非强迫的解决方案。

Gavin King 开发 Hibernate 的动机有两个：发现 CMP 太滥；与对老板的争执。Gavin King 当时没有任何用 SQL 开发数据库的经验，Gavin King 开发 Hibernate 的第一件事是去街上买了本 SQL 基础的书。

也许 Cirrus Technologies 的老板做梦也想不到两年以后，这个小伙子开发出的那个产品会成为全世界最流行的 O/R Mapping 工具，而那个对 SQL 和数据库一窍不通的小伙子居然会成为全世界 J2EE 数据库解决方案的领导者。

这就是 Gavin King，一个充满激情、脾气很倔、永不言败的人。他的成就也许全世界搞 Java 的人都知道：他是 Hibernate 的创始人；他是 EJB 3.0 的 Entity bean specification 的实际领导人（SUN 任命的领导人应该是 Linda DeMichiel）；他也是那本经典的书 Hibernate in action 的作者；他也参加了 XDoclet 和 Middlegen 的开发；他在全世界各种著名的会议（TheServerSide Symposium 等）进行演讲和讲座。

充满激情、脾气倔强、永不言败的 Gavin King 创造了 Hibernate，我们是否能够通过我们的努力，创造更多更好的解决方案呢？

批注 [w3]: 删除一行

5. 实验

1. 手工配置 Hibernate 框架完成书上示例。（20 分钟）

2. 使用 MyEclipse Database Explorer 对数据库进行基本操作。建立 mybbs 数据库，建 userinfo 表。（30 分钟）

3. 使用 MyElipse 进行 Hibernate 框架开发，完成注册功能。（40 分钟）

6. 课后作业

1. 完成 mybbs 论坛栏目的添加。

案例二 Hibernate 核心 API

1. 教学目标

1.1 掌握 Hibernate6 个核心接口 Session，SesssionFactory，Configuration，Transaction，Query 和 Criteria。

1.2 掌握使用 Hibernate 对数据的进行查询、删除和修改。

2. 工作任务

2.1 使用 Hibernate 完成登录任务。

2.2 使用 Hibernate 对表进行修改和删除。

3. 相关实践知识

3.1 使用 Hibernate 完成登录任务

1. 在 MyEclipse 中导入 Struts 案例中的登录示例。

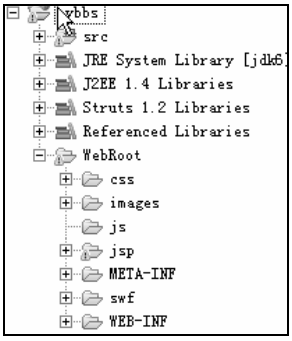


图 2-28 导入 mybbs 工程目录

2. 将 com.mybbs.dao 和 com.mybbs.pojo 中的类删除，时将程序中相关的引用进行注释。

3. 右键点击工程名字“mybbs”，加入 Hibernate 框架。按照上次课程完成 Hibernate 框架配置。

4. 打开【MyEclipse Database Explorer】视图，可以看到配置好的连接。如果上次课程配置的连接本地机器没有保存，请参照上次课程的相关内容进行数据库连接的配置。

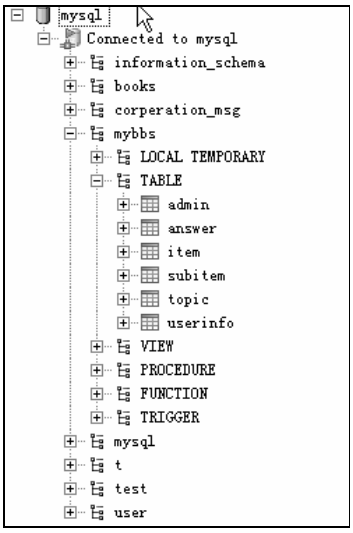


图 2-29 配置好的数据库连接

5. 右键点击 userinfo 表，选择【Hibernate Reverse Engineering】翻转引擎。选择生成 pojo，放在 com.mybbs.pojo 包里。如图所示：

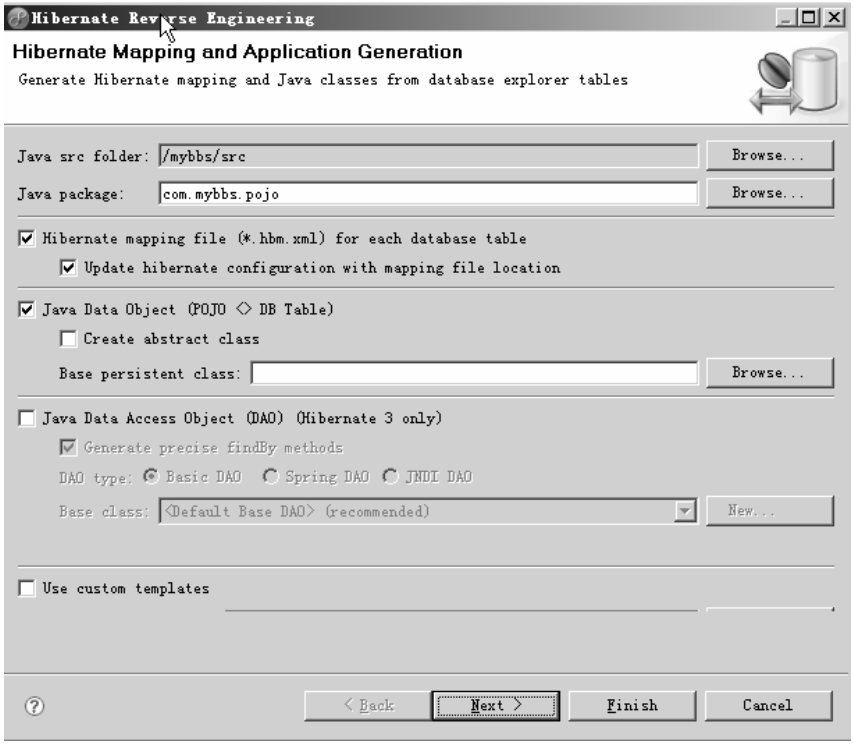


图 2-30 【Hibernate Reverse Engineering】设置

6. 选择【Finish】，在 com.mybbs.pojo 包中自动生成了 Userinfo 类。同时生成了 Userinfo.hbm.xml 映射文件，而且在 hibernate.cfg.xml 中自动添加。

```
<mapping resource="com/mybbs/pojo/Userinfo.hbm.xml" />
```

代码，自动完成 Userinfo 类和 Hibernate 框架的映射。

7. 打开 Userinfo.hbm.xml 文件将主键的生成策略进行修改，默认的主键生成策略为：assigned，改成 native。

```
<id name="userid" type="java.lang.Integer">
          <column name="userid" />
          <generator class="native" />
 </id>
```

8. 在 com.mybbs.dao 中编写 UserinfoDAO 类，完成登录业务处理。

```
package com.mybbs.dao;
//导入 Hibernate 核心包
import  org.hibernate.SessionFactory;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;
import org.hibernate.Query;
import org.hibernate.Transaction;
import java.util.List;
import com.mybbs.pojo.Userinfo;
public class UserinfoDAO {
    //检查用户是否存在
    public boolean isUser(Userinfo user){
     //设定一个标识符
     boolean flag=false;
     //通过 Configuration 类实例化自动加载 hibernate.cfg.xml
     Configuration cf=new Configuration();
     //通过配置文件信息，得到 SessionFactory
     SessionFactory sf=cf.configure().buildSessionFactory();
     //通过 SessionFactory 打开一个 Session
     Session s=sf.openSession();
     //查询语句
     String hql="from Userinfo as u  where u.username=:name and u.userpwd=:pwd";
     //根据 HQL 语句通过 session 得到查询对象 Query
     Query q=s.createQuery(hql);
     //设定查询语句中的参数
     q.setString("name", user.getUsername());
     q.setString("pwd", user.getUserpwd());
     //根据查询语句得到查询结果的集合
```

```
    List <Userinfo> ls=q.list();
    //判断集合中是否有值
    if(ls.size()==0){
        //如果集合为空则方法返回 false
        return flag;
    }else{
       //如果集合不为空则返回 true
       flag=true;
    return flag;
    }
   }
}
```

9. 在 LoginAction 中完成业务逻辑的调用，调用部分代码如下：

```
Userinfo user=new Userinfo();
       user.setUsername(LoginActionForm.getName());
       System.out.println(user.getUsername());
       user.setUserpwd(LoginActionForm.getPassword());
       System.out.println(user.getUserpwd());
      UserinfoDAO uDAO=new UserinfoDAO();
       if(uDAO.isUser(user))
       {
           return mapping.findForward("success");
       }
       else
       {
           return mapping.findForward("error");
       }
   }
```

10. 启动 tomcat，在地址栏输入 http://localhost:8080/mybbs/jsp/login.jsp 进行测试。此时测试的结果和使用 JDBC 操作的结果一样。

通过上面示例，我们对于 Hibernate 的查询有了一定的了解，上次课程也学习 Hibernate 的添加示例。下面的示例让我们一起学习怎样使用 Hibernate 进行数据的删除和修改。

3.2 使用 Hibernate 对表进行修改和删除

1. 在 MyEclipse 中创建一个【Java Project】工程，工程名字为 hibernatedemo2。同时在 src 目录下建立三个包 com.mybbs.pojo，com.mybbs.dao，com.mybbs.test。结果如下图所示：

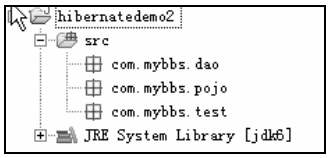


图 2-31 hibernatedemo2 工程目录

2. 在 hibernatedemo2 工程中加入 Hibernate 框架。按照前面章节进行操作即可但需注意以下一点：

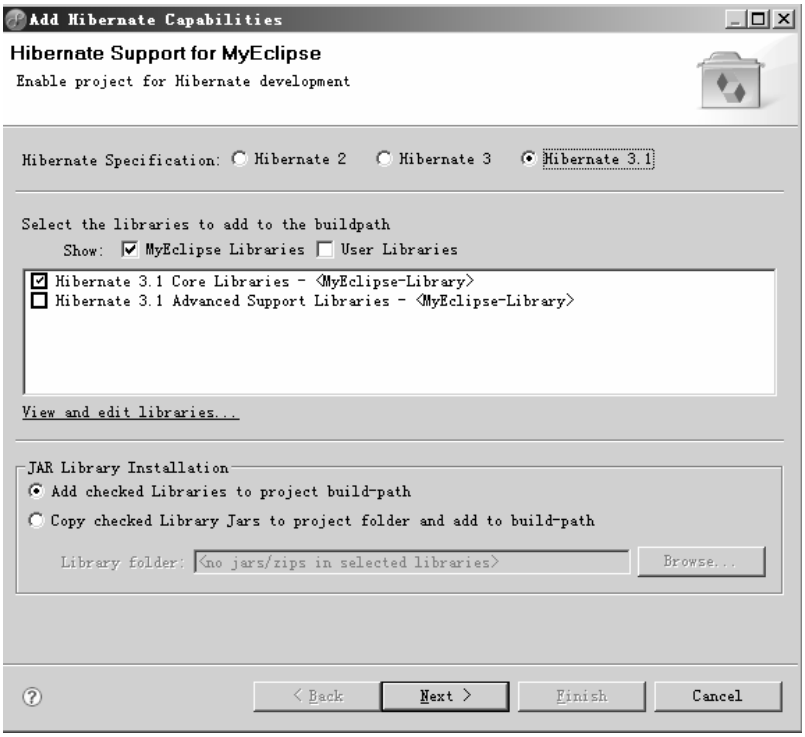


图 2-32 向工程中加入 Hibernate 框架

【Java Project】工程引入 Hibernate 框架【JAR Library Installation】选项，要选择默认项【Add checked Libraries to project build-path】。否则会出现问题。

3. 右键点击 userinfo 表，选择【Hibernate Reverse Engineering】翻转引擎。选择生成 pojo，放在 com.mybbs.pojo 包里。不要忘记修改 userinfo.hbm.xml 中的主键生成策略。

```
<id name="userid" type="java.lang.Integer">
          <column name="userid" />
          <generator class="native" />
</id>
```

4. 在 com.mybbs.dao 中编写 UserinfoDAO 类，编写删除用户和修改用户信息的方法。

```
package com.mybbs.dao;
//导入 HIbernate 核心包
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.Query;
import com.mybbs.pojo.Userinfo;
public class UserinfoDAO {
  //根据主键删除记录
    public void deleteUser(int id){
       //通过 Configuration 类实例化自动加载 hibernate.cfg.xml
       Configuration cf=new Configuration();
       //通过配置文件信息，得到 SessionFactory
       SessionFactory sf=cf.configure().buildSessionFactory();
       //通过 SessionFactory 打开一个 Session
       Session s=sf.openSession();
       //根据 session，通过 id,得到一个 userinfo
       Userinfo user=(Userinfo) s.get(Userinfo.class, id);
       //通过 session 开始事务
       Transaction tx=s.beginTransaction();
       //删除得到的 userinfo
       s.delete(user);
       //提交事务
       tx.commit();
       //关闭 session
       s.close();
  }
   //根据主键修改记录
  public void updateUser(int id){
       //通过 Configuration 类实例化自动加载 hibernate.cfg.xml
       Configuration cf=new Configuration();
     //通过配置文件信息，得到 SessionFactory
       SessionFactory sf=cf.configure().buildSessionFactory();
       //通过 SessionFactory 打开一个 Session
       Session s=sf.openSession();
       //根据 session，通过 id,得到一个 userinfo
       Userinfo us=(Userinfo)s.get(Userinfo.class, id);
       //重新设置当前用户的问题
       us.setUserques("你的偶像是谁？");
       //重新设置当前用户的的答案
       us.setUserans("比尔.盖茨");
```

```
        //开始事务
        Transaction tx=s.beginTransaction();
        //更新当前用户
        s.update(us);
        //提交事务
        tx.commit();
        //关闭 session
        s.close();
    }
}
```

5. 在 com.mybbs.test 包中建立测试类 UserinfoDAOTest。

```
package com.mybbs.test;
import com.mybbs.pojo.Userinfo;
import com.mybbs.dao.UserinfoDAO;
import java.util.Scanner;
//这是一个测试类
public class UserinfoDAOTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("请输入你要删除用户的 id 号");
        Scanner sc=new Scanner(System.in);
        int id=sc.nextInt();
        UserinfoDAO uDAO=new UserinfoDAO();
        uDAO.deleteUser(id);

//        System.out.println("请输入你要修改用户的 id 号");
//        Scanner s=new Scanner(System.in);
//        int uid=s.nextInt();
//        UserinfoDAO uDAO=new UserinfoDAO();
//        uDAO.updateUser(uid);
    }
}
```

取消注释两个方法轮流测试。查看数据库的信息进行比对，可以看到成功的进行了相关操作。

4. 相关理论知识

应用程序可以直接通过 Hibernate API 访问数据库。Hibernate API 中的接口可分为以下几类。

- 提供访问数据库的操作（如保存、更新、删除和查询对象）的接口。这些接口

包括：Session、Transaction 和 Query 接口。

- 用于配置 Hibernate 的接口：Configuration。
- 回调接口，使应用程序接受 Hibernate 内部发生的事件，并做出相关的回应。这些接口包括：Interceptor,Lifecycle 和 Validatable 接口。
- 用于扩展 Hibernate 的功能接口，如 UserType、CompositeUserType 和 Identifier Generator 接口。如果需要的话可以扩展这些接口。

Hibernate 内部封装了 JDBC、JTA（Java Transaction API）和 JNDI（Java Naming and Directory Interface）。JDBC 提供底层的数据访问操作，只要用户提供了相应的 JDBC 驱动程序，Hibernate 可以访问任何一个数据库系统。JNDI 和 JTA 使 Hibernate 能够和 J2EE 应用服务集成。

虽然有这么多接口但是对于开发者来说我们只需掌握常用的核心的接口，下面介绍 Hibernate 的六大核心接口。

所有的 Hibernate 应用中都会访问 Hibernate 的六大核心接口。

- Configuration 接口：配置 Hibernate，启动 Hibernate，创建 SessionFactory 对象。
- SessionFactory 接口：初始化 Hibernate，充当数据存储源的代理，创建 Session 对象。
- Session 接口：负责保存、更新、删除、加载和查询对象。
- Transaction：管理事务。
- Query 和 Critetia：执行数据库查询。

下面具体的阐述各个核心类的主要功能，不会详细介绍 Hibernate API 接口中的所有方法，如果想了解具体方法请查看 Hibernate 的文档。

批注 [w4]: 删除一行

- Configuration 接口的作用是对 Hibernate 进行配置，以及对它进行启动。在 Hibernate 的启动过程中，Configuration 类的实例首先定位映射文档的位置，读取这些配置，然后创建一个 SessionFactory 对象。一个 Configuration 实例代表了一个应用程序中 Java 类型到 SQL 数据库的完整集合。Configuration 被用来构建一个不可变的 SessionFactory，映射定义则由不同的 XML 映射定义编译而来。要使用一个 Configuration，需要为它设置三个方面的内容：
  1. 数据库连接属性（在 hibernate.cfg.xml 中配置）
  2. hbm.xml 文件
  3. POJO 类

Configuration 实例是一个启动期间的对象，一旦 SessionFactory 创建完成，它就被丢弃了。

SessionFactory 实例对应一个数据存储源，应用从 SessionFactory 中获得 Session 实例。SessionFactory 用到了一个设计模式-工厂模式，用户程序从工厂类 SessionFactory 中取得 Session 的实例，设计者的意图是让它能在整个应用程序中共享。SessionFactroy 有以下特点：

它是线程安全的，意味着他的同一个实例可以被多个线程共享。

它是重量级的，这意味着不能随意创建或销毁它的实例。一般来说连接一个数据库我们

只需创建一个 SessionFactory 实例，如果同时访问多个数据库，则需要为每个数据库创建一个单独的 Session 实例。

使用 Session 操纵数据库。Session 对于开发者来说是最重要的一个接口。在 Hibernate 中实例化的 Session 是一个轻量级的类，创建和销毁它不需要太多的资源。在实际的项目中，经常会不断地创建和销毁 Session 对象。如果 Session 的开销太大，会给系统带来不良的影响。但是值得我们关注的是 Session 对象是非线程安全的，因此在设计中，最好是一个线程只创建一个 Session 对象。

Session 的主要功能是提供对映射的实体类的增删改查的操作。实例可能以下面的三种状态存在：

自由状态（transient）：不曾进行持久化，未与任何 Session 相关联；

持久化状态（persistent）：仅与一个 Session 相关联；

游离状态（detached）：已经进行过持久化，但当前未与任何 Session 相关联。

游离状态的实例可以通过调用 save()、persist()或者 saveOrUpdate()方法进行持久化。持久化实例可以通过调用 delete()方法变成游离状态。通过 get()或 load()方法得到的实例都是持久化状态的。游离或者自由状态下的实例可以通过调用 merge()方法成为一个新的持久化对象。

save()和 persist()将会引发 SQLINSERT，delete()会引发 SQLDELETE，而 update()或 merge()会引发 SQLUPDATE。对持久化（persistent）实例的修改在刷新提交的时候会被检测到，也会引起 SQLUPDATE。saveOrUpdate()或者 replicate()会引发 SQLINSERT 或者 SQLUPDATE。

下面具体分析一下 Session 是如何操纵数据库。

使用 save()保存对象

save()用于将一个新实例化的对象持久化，保存到数据库中。看看上一章的用户注册示例代码片段。通过 HibernateSessionFactory 创建一个 Session 对象，然后将一个 Userinfo 实例通过 save()方法将该对象保存到数据库中。

```
public void save(UserInfo user){
    try{
    //第一步：通过 Configuration 类实例化自动加载 hibernate.cfg.xml
   Configuration cfg=new Configuration();
   //第二步：通过配置文件信息，得到 SessionFactory
   SessionFactory sf=cfg.configure().buildSessionFactory();
   //第三步：通过 SessionFactory 打开一个 Session
   Session s=sf.openSession();
   //第四步：通过 Session 开始事务
   Transaction tx=s.beginTransaction();
   //第五步：将用户保存到 Session（持久化操作）
   s.save(user);
   //第六步：提交事务
   tx.commit();
```

```
    //第七步：关闭 session
    s.close();
    }catch(Exception ex){
     ex.printStackTrace();
    }
    }
```

使用 load()装载对象

如果知道某个实例的持久化标识（主键），就可以通过 Session 的 load()方法来获取它。load()的另一个参数是指定类的 Class 对象。本方法会创建指定类的持久化实例，并从数据库中加载其数据。比如我们已知 userinfo 表中有一个主键值为“1”的对象存在，我们可以采取下面的方式得到这个主键为“1”的实例：

```
//第一步：通过 Configuration 类实例化自动加载 hibernate.cfg.xml
Configuration cfg=new Configuration();
//第二步：通过配置文件信息，得到 SessionFactory
SessionFactory sf=cfg.configure().buildSessionFactory();
//第三步：通过 SessionFactory 打开一个 Session
Session s=sf.openSession();
//第四步：通过 load()方法加载数据库中主键 id 为“1”的对象
Userinfo user=(Userinfo)s.load(Userinfo.class,new Integer("1"));
```

注意：如果没有匹配的数据，load()方法可能抛出无法恢复的异常（unrecoverable exception）

使用 get()装载对象

如果不能确定数据库中是否有匹配的记录存在，应该使用 get()方法，它会立刻访问数据库，如果没有对应的记录，会返回 null。如下面代码片段所示，如果 userinfo 表中没有 id 为“4”的记录，get()装载的结果会返回 null；而如果本记录不存在使用 get()装载则会出现异常。

```
//第一步：通过 Configuration 类实例化自动加载 hibernate.cfg.xml
Configuration cfg=new Configuration();
//第二步：通过配置文件信息，得到 SessionFactory
SessionFactory sf=cfg.configure().buildSessionFactory();
//第三步：通过 SessionFactory 打开一个 Session
Session s=sf.openSession();
//第四步：通过 load()方法加载数据库中主键 id 为“1”的对象
Userinfo user=(Userinfo)s.get(Userinfo.class,new Integer("4"));
```

使用 update()提交游离对象

update()用于根据给定的 detached（游离状态）对象实例的标识更新对应的持久化实例。

一般而言，传递给 update 的对象要是处于游离状态的对象。如果传入一个持久化对象那么 update 方法是多余的，因为 Hibernate 的检查机制会自动根据对象属性的值变化，自动向数据库发送一条 update 语句，此时使用 save 方法即可。本次课程 hibernatedemo2 工程中的示例：

```
public void updateUser(int id){
```

```
        //通过 Configuration 类实例化自动加载 hibernate.cfg.xml
        Configuration cf=new Configuration();
      //通过配置文件信息，得到 SessionFactory
        SessionFactory sf=cf.configure().buildSessionFactory();
        //通过 SessionFactory 打开一个 Session
        Session s=sf.openSession();
        //根据 session，通过 id,得到一个 userinfo
        Userinfo us=(Userinfo)s.get(Userinfo.class, id);
        //重新设置当前用户的问题
        us.setUserques("你的偶像是谁？");
        //重新设置当前用户的的答案
        us.setUserans("比尔.盖茨");
        //开始事务
        Transaction tx=s.beginTransaction();
        //更新当前用户
        s.update(us);
        //提交事务
        tx.commit();
        //关闭 session
        s.close();
    }
```

其中的 update 方法完全可以使用 save 方法进行替换，因为为了演示修改，所以此处使用了 update 方法。

如果传入的对象处于临时状态，那么此时 Hibernate 应该会抛出异常。因为 Hibernate 在更新数据时会根据对象的 ID 去数据库中查找相应的记录并将它更新，而在数据库中是没有记录与这个临时对象相关联，因此 Hibernate 就会抛出异常。所以当要对一个临时对象进行 update 操作的时候一定要给这个临时对象指定一个主键。

在执行 update 方法的时候，Hibernate 会首先把传入的对象放入 Session 的缓存中，使之持久化，然后计划执行一个 update 语句。

使用 delete()删除持久化对象

此方法用于从数据库中删除对象对应的记录。如本章示例代码所示：

```
//通过 Configuration 类实例化自动加载 hibernate.cfg.xml
        Configuration cf=new Configuration();
        //通过配置文件信息，得到 SessionFactory
        SessionFactory sf=cf.configure().buildSessionFactory();
        //通过 SessionFactory 打开一个 Session
        Session s=sf.openSession();
        //根据 session，通过 id,得到一个 userinfo
        Userinfo user=(Userinfo) s.get(Userinfo.class, id);
```

```
//通过 session 开始事务
Transaction tx=s.beginTransaction();
//删除得到的 userinfo
s.delete(user);
//提交事务
tx.commit();
//关闭 session
s.close();
```

Transaction 接口是 Hibernate 的数据库事务接口，它对底层的事务接口做了封装，底层事务接口包括：

JDBC API

JTA（Java Transaction API）

CORBA（Common Object Request Broke Architecture）API

之所以这样设计，是为了开发者能够使用一个统一事务的操作界面，使得自己的项目可以在不同的环境和容器之间方便地移植。

一个典型的事务应该这样使用，在创建完 Session 对象后即使用 beginTranstransaction()启动事务，从此开始直到 commit()之间的代码，都会处于同一个事务中。这两个函数之间的所有数据库代码都会在 commit()时一次性提交，在提交时如果某一句代码执行出现异常，就会回滚这次事务之间的所有执行代码。

使用 Query 进行 HQL 查询

Query 接口让你方便地对数据库及持久对象进行查询，它可以有两种表达方式：HQL 语言或本地数据库的 SQL 语句。Query 经常被用来绑定查询参数、限定查询记录数量，并最终执行查询操作。

要取得 Query 对象，要使用 Session 的 createQuery()方法来执行查询，查询的参数是基于 HQL 语法的，其查询的对象是 Hibernate 的持久化对象名，Hbernate 会根据该对象名找到要查找的表名。

下面具体的阐释一下如何利用 Query 来执行查询。

不带参数的查询

最简单的查询就是不带任何参数的查询，查询的语句是“from POJO”的形式，其中的 POJO 为持久化类而不是表名字。如下所示：

```
Query query=session.CreateQuery(“from Userinfo”);
```

带参数的查询

接口 Query 提供了对命名参数、JDBC 风格的问号（?）参数两种绑定方法。一般来说最好使用命名参数方式，命名参数在查询字符串中形式为：name。命名参数的优点是：

命名参数与其在查询字符串中出现的位置顺序无关；

它们可以在一个查询字符串中多次出现；

它们本身是自我说明的；

如登录实例代码片段如下：

```
//查询语句
String hql="from Userinfo as u  where u.username=:name and u.userpwd=:pwd";
//根据 HQL 语句通过 session 得到查询对象 Query
Query q=s.createQuery(hql);
//设定查询语句中的参数
q.setString("name", user.getUsername());
q.setString("pwd", user.getUserpwd());
//根据查询语句得到查询结果的集合
List <Userinfo> ls=q.list();
```

其中下面这段代码中：

```
String hql="from Userinfo as u  where u.username=:name and u.userpwd=:pwd";
```

“Userinfo”代表的是持久化类的名字，“as u”给这个类起个别名为“u”而后的“u.username”和“u.userpwd”是这个类中的字段名，“name”和“pwd”是绑定的查询命名参数，然后通过 Query 对象的 setXXX 方法给其赋值。

Query 使用问号参数时与 JDBC 不同，Hibernate 对参数从 0 开始计数。上面的示例可以改为如下形式：

```
//查询语句
String hql="from Userinfo as u  where u.username=? and u.userpwd=?";
//根据 HQL 语句通过 session 得到查询对象 Query
Query q=s.createQuery(hql);
//设定查询语句中的参数
q.setString(0, user.getUsername());
q.setString(1, user.getUserpwd());
//根据查询语句得到查询结果的集合
List <Userinfo> ls=q.list();
```

这样与带参数的查询效果是一样的。

取得一个对象

如果已经知道当前查询只会返回一个对象，可以使用 uniqueResult()来取得一个对象。例如登录示例，我们可以修改成这样效果：

```
//查询语句
String hql="from Userinfo as u  where u.username=? and u.userpwd=?";
//根据 HQL 语句通过 session 得到查询对象 Query
Query q=s.createQuery(hql);
//设定查询语句中的参数
q.setString(0, user.getUsername());
```

```
q.setString(1, user.getUserpwd());
//根据查询语句得到查询结果的集合
Userinfo user=q. uniqueResult();
```

创建 SQL 查询

可以使用 createSQLQuery()方法，用普通的 SQL 来描述，并由 Hibernate 处理将结果转换成对象的工作。这样我们可以通过手写 SQL 来完成所有的 create，update，delete 和 load 操作。如下示例：

```
List user=s.createSQLQuery("select {user.*} from userinfo {"user"}").list();
```

其中的 SQL 别名需要用大括号包围起来。和 Hibernate 查询一样，SQL 查询也可以包含命名参数和占位参数。

使用 Criteria 进行条件查询

Criteria 接口完全封装了基于字符串形式的查询语句，比 Query 接口更加面向对象，Criteria 接口擅长于执行动态查询。如下面代码所示：

```
    Criteria c=s.createCriteria(Userinfo.class);
    Criterion cr=Expression.eq("pwd",new Integer("2") );
    c=c.add(cr);
    Userinfo user=(Userinfo)c.uniqueResult();
```

使用步骤如下：

调用 Session 的 createCriteria()方法创建一个 Criteria 对象。

设定查询条件。Expression 类提供了一系列用于设定查询条件的静态方法，这些静态方法都返回 Criterion 实例，每个 Criterion 实例代表一个查询条件。Criterion 的 add 方法用于加入查询条件。

调用 Criteria 的 uniqueResult()方法执行查询语句。该方法返回一个 Userinfo 实例。

在更多的查询应用过程中，更多的使用 Query 类完成相关的查询工作。

5. 实验

1. 完成实验部分示例代码编写。（30 分钟）
2. 使用 SQLSERVER2000 数据库，建立如下两张表：

```
CREATE TABLE [dbo].[houseinfo] (
    [hid] [int] IDENTITY (1, 1) NOT NULL ,
    [streetid] [int] NULL ,
    [shi] [int] NULL ,
    [ting] [int] NULL ,
    [houseinfo] [varchar] (2000) COLLATE Chinese_PRC_CI_AS NULL ,
    [zj] [money] NULL ,
```

```
    [title] [varchar] (70) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [date] [datetime] NULL ,
    [telephone] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL ,
    [lxr] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[street] (
    [streetid] [int] IDENTITY (6, 1) NOT NULL ,
    [streetname] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
```

Houseinfo 是房屋信息表，street 是街道表，其中两表通过 streedid 建立主外键关系。插入相关测试数据。

使用 HQL 查询。

1. 查询所有房屋信息。（5 分钟）
2. 查询所有房屋信息的 date 属性和 title 属性。（20 分钟）

   提示：查询部分属性返回值是一个 Object 类型的数组而不是 bean 对象。

3. 参数查询 比如查询房屋的 title 带有“天然气”的房屋信息。（10 分钟）

   提示：使用 like 关键字。

4. 询所有房屋信息记录数，计算房屋租金平均价格。（25 分钟）

   提示：使用相关函数比如 count()，avg()。

6. 作业

1. 使用 Hibernate 进行分页查询

提示：

1. 需要两个变量一个当前页数，一个每页显示多少行
2. 使用 Query 中的 setFirstResult()，setMaxResults()。

案例三 Hibernate 关联映射

1. 教学目标

1.1 掌握单向 many-to-one 关联

1.2 掌握双向 one-to-many 关联

1.3 掌握双向 many-to-many 关联

2. 工作任务

2.1. 配置 Hibernate 关联通过二级栏目得到栏目的信息

2.2. 添加一个栏目“Ruby”，增加 Ruby 下属的三个子栏目

2.3. 删除一级栏目中“.net”项

2.4. 将“书籍教程”移动到“Java”栏目下

2.5. 建立学生表和教师表完成双向多对多关系的映射

3. 相关实践知识

为了便于本章知识的学习和测试，本章建立的工程类型为【Java Project】。

3.1 配置 Hibernate 关联通过二级栏目得到栏目的信息

1. 新建一个【Java Project】，工程名字为 hibernate3。

2. 在 hibernate3 工程下建立 2 个包，包名分别为 com.hibernate3.pojo，com.hibernate3.dao。建立后工程结构如图所示：

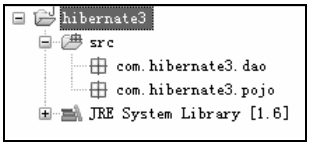


图 2-33 建立 hibernate3 工程

3. 添加 Hibernate3.1 扩展库。

图 2-34 添加 Hibernate3.1 工程

4. 点击【Next】创建 hibernate.cfg.xml 文件。

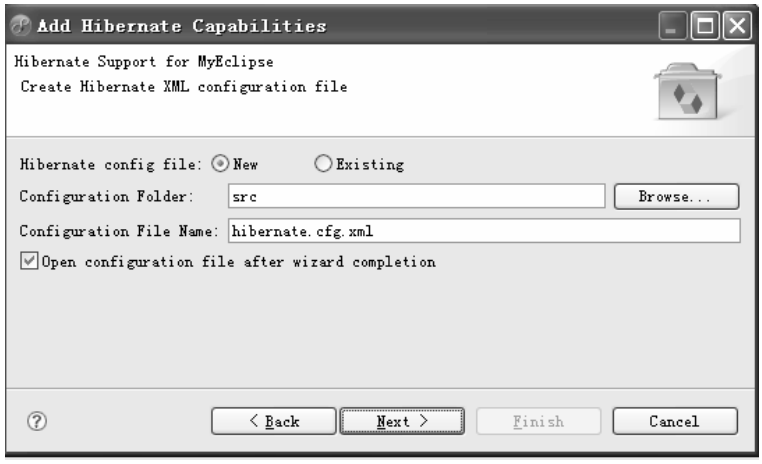


图 2-35 创建 hibernate.cfg.xml 文件

5. 点击【Next】，选择在【MyEclipse Database Explorer】中配置的数据库连接。

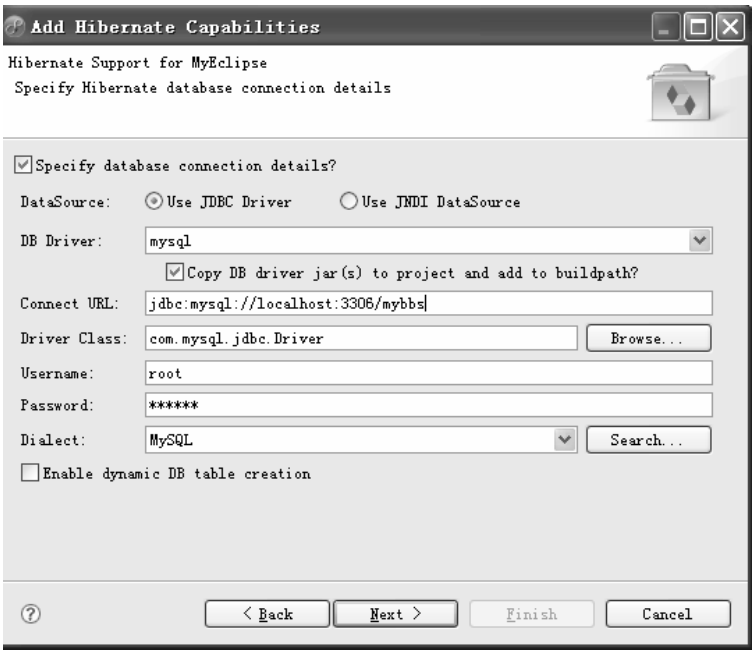


图 2-36 选择在【MyEclipse Database Explorer】配置的数据库连接

6. 点击【Next】，在 com.hibernate3.dao 包下建立 HibernateSessionFactory 类。

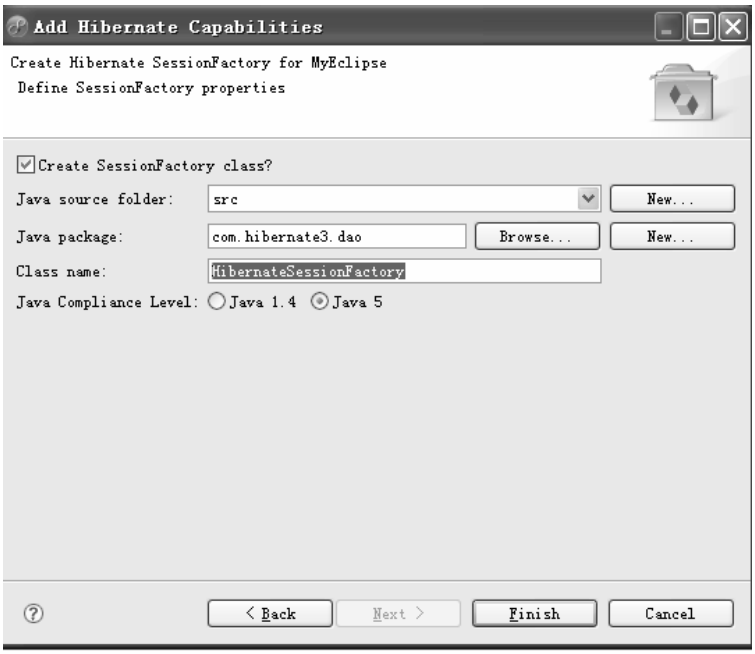


图 2-37 创建 HibernateSessionFactory 类

7. 点击【Finish】完成。

8. 生成 HibernateSessionFactory 代码如下：

```
package com.hibernate3.dao;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;

/**
 * Configures and provides access to Hibernate sessions, tied to the
 * current thread of execution.  Follows the Thread Local Session
 * pattern, see {@link http://hibernate.org/42.html }.
 */
public class HibernateSessionFactory {

   /**
    * Location of hibernate.cfg.xml file.
    * Location should be on the classpath as Hibernate uses
    * #resourceAsStream style lookup for its configuration file.
    * The default classpath location of the hibernate config file is
    * in the default package. Use #setConfigFile() to update
    * the location of the configuration file for the current session.
    */
   private static String CONFIG_FILE_LOCATION = "/hibernate.cfg.xml";
    private   static   final   ThreadLocal<Session>   threadLocal   =   new
ThreadLocal<Session>();
   private  static Configuration configuration = new Configuration();
   private static org.hibernate.SessionFactory sessionFactory;
   private static String configFile = CONFIG_FILE_LOCATION;

    static {
    try {
            configuration.configure(configFile);
            sessionFactory = configuration.buildSessionFactory();
        } catch (Exception e) {
            System.err
                    .println("%%%% Error Creating SessionFactory %%%%");
            e.printStackTrace();
        }
   }
   private HibernateSessionFactory() {
   }
```

```
/**
 * Returns the ThreadLocal Session instance.  Lazy initialize
 * the <code>SessionFactory</code> if needed.
 *
 *  @return Session
 *  @throws HibernateException
 */
public static Session getSession() throws HibernateException {
   Session session = (Session) threadLocal.get();

    if (session == null || !session.isOpen()) {
        if (sessionFactory == null) {
            rebuildSessionFactory();
        }
        session = (sessionFactory != null) ? sessionFactory.openSession()
                : null;
        threadLocal.set(session);
    }

   return session;
}

/**
 *  Rebuild hibernate session factory
 *
 */
public static void rebuildSessionFactory() {
    try {
        configuration.configure(configFile);
        sessionFactory = configuration.buildSessionFactory();
    } catch (Exception e) {
        System.err
                .println("%%%% Error Creating SessionFactory %%%%");
        e.printStackTrace();
    }
}
/**
 *  Close the single hibernate session instance.
 *
 *  @throws HibernateException
 */
```

```
    public static void closeSession() throws HibernateException {
        Session session = (Session) threadLocal.get();
        threadLocal.set(null);

        if (session != null) {
            session.close();
        }
    }

    /**
     *  return session factory
     *
     */
    public static org.hibernate.SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    /**
     *  return session factory
     *
     *   session factory will be rebuilded in the next call
     */
    public static void setConfigFile(String configFile) {
        HibernateSessionFactory.configFile = configFile;
        sessionFactory = null;
    }

    /**
     *  return hibernate configuration
     *
     */
    public static Configuration getConfiguration() {
        return configuration;
    }
}
```

9. 在 com.hibernate3.pojo 包下编写 Item 类。代码如下：

```
package com.hibernate3.pojo;

/**
 * Item entity.
 *
```

```
 * @author admin
 */

public class Item implements java.io.Serializable {

    // Fields

    private Integer itemid;
    private String itemname;
    private String itemcode;

    // Constructors

    /** default constructor */
    public Item() {
    }

    /** full constructor */
    public Item(String itemname, String itemcode) {
        this.itemname = itemname;
        this.itemcode = itemcode;
    }

    // Property accessors

    public Integer getItemid() {
        return this.itemid;
    }

    public void setItemid(Integer itemid) {
        this.itemid = itemid;
    }

    public String getItemname() {
        return this.itemname;
    }

    public void setItemname(String itemname) {
        this.itemname = itemname;
    }

    public String getItemcode() {
```

```
        return this.itemcode;
    }

    public void setItemcode(String itemcode) {
        this.itemcode = itemcode;
    }
}
```

10. 在 com.hibernate.pojo 包下编写 Item.hbm.xml 映射文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
   Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
   <class name="com.hibernate3.pojo.Item" table="item" catalog="mybbs">
      <id name="itemid" type="java.lang.Integer">
         <column name="itemid" />
         <generator class="native" />
      </id>
      <property name="itemname" type="java.lang.String">
         <column name="itemname" length="50" not-null="true" />
      </property>
      <property name="itemcode" type="java.lang.String">
         <column name="itemcode" length="50" not-null="true" />
      </property>
   </class>
</hibernate-mapping>
```

11. 在 com.hibernate.pojo 包下编写 Subitem 类，代码如下：

```
package com.hibernate3.pojo;

/**
 * Subitem entity.
 *
 * @author MyEclipse Persistence Tools
 */

public class Subitem implements java.io.Serializable {
```

```
    // Fields

    private Integer subid;
    private String subname;
    private Item item;
    private Integer subcode;

    // Constructors

    /** default constructor */
    public Subitem() {
    }

    // Property accessors

    public Integer getSubid() {
        return this.subid;
    }

    public void setSubid(Integer subid) {
        this.subid = subid;
    }

    public String getSubname() {
        return this.subname;
    }

    public void setSubname(String subname) {
        this.subname = subname;
    }

    public Integer getSubcode() {
        return this.subcode;
    }

    public void setSubcode(Integer subcode) {
        this.subcode = subcode;
    }

    public Item getItem() {
        return item;
    }
```

```
    public void setItem(Item item) {
        this.item = item;
    }
}
```

注意：在本类中有一个 item 字段，类型为 Item。

12. 在 com.hibernate.pojo 包下编写 Subitem.hbm.xml 映射文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.hibernate3.pojo.Subitem" table="subitem"
                                                    catalog="mybbs">
        <id name="subid" type="java.lang.Integer">
            <column name="subid" />
            <generator class="native" />
        </id>
        <property name="subname" type="java.lang.String">
            <column name="subname" length="20" not-null="true" />
        </property>
        <property name="subcode" type="java.lang.Integer">
            <column name="subcode" not-null="true" />
        </property>
        <many-to-one name="item" column="itemid"
                                  class="com.hibernate3.pojo.Item"/>
    </class>
</hibernate-mapping>
```

注意：<.many-to-one>元素配置。

13. 在 hibernate.cfg.xml 进行映射文件的配置，添加内容如下：

```
<mapping resource="com/hibernate3/pojo/Item.hbm.xml" />
<mapping resource="com/hibernate3/pojo/Subitem.hbm.xml" />
```

14. 在 com.hibernate3.dao 包内编写 TestDAO 类，代码如下：

```
package com.hibernate3.dao;
import com.hibernate3.pojo.*;
import org.hibernate.*;
```

```
public class TestDAO {
    public static void main(String[] args){
     new TestDAO().manyToOne();
    }
    public void manyToOne(){
     //得到 session
     Session session=HibernateSessionFactory.getSession();
     //取出 subid 为 1 的 Subitem
     Subitem subItem=(Subitem)session.get(Subitem.class, 1);
        //通过配置相关映射可以得到栏目的相关信息
    String itemName=subItem.getItem().getItemname();
     String itemCode=subItem.getItem().getItemcode();
    System.out.println("栏目信息"+itemName+"\t"+itemCode);   }
}
```

15. 运行结果如下：

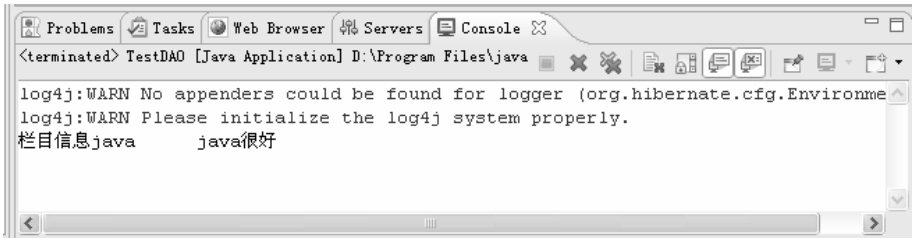


图 2-38 TestDAO 运行结果

3.2 添加一个栏目“Ruby”，增加 Ruby 下属的三个子栏目

1. 向 Item 类中添加一个字段，并生成 get，set 访问器，代码如下：

```
private Set subItems=new HashSet();
public Set getSubItems() {
        return subItems;
    }

    public void setSubItems(Set subItems) {
        this.subItems = subItems;
    }
```

2. 在 Item.hbm.xml 文件中添加如下内容：

```
<set name="subItems" >
          <key column="itemid"/>
          <one-to-many class="com.hibernate3.pojo.Subitem"/>
 </set>
```

3. 在 TestDAO 中写一个方法 addItem()，添加一个“Ruby”栏目。代码如下：

```
public void addNewItem(){
    Session session=HibernateSessionFactory.getSession();
    Transaction tx=session.beginTransaction();
    Item item=new Item();
    item.setItemcode("Ruby 迅速发展");
    item.setItemname("Ruby");
    session.save(item);
    tx.commit();
    session.close();
   }
```

4. 在 TestDAO 中 main()方法中调用 addNewItem()方法，这样就向数据库中新增一个栏目名字为“Ruby”。

5. 在 TestDAO 中编写 oneToMany()方法，在“Ruby”栏目添加三个子项，分别为：“源码下载”，“在线学习”，“书籍教程”，代码如下：

```
public void oneToMany(){
    Set subItems=new HashSet();
    Session session=HibernateSessionFactory.getSession();
    Transaction tx=session.beginTransaction();
    Item item=(Item)session.get(Item.class,3);
    Subitem subItem1=new Subitem();
    subItem1.setSubname("源码下载");
    subItem1.setItem(item);
    subItem1.setSubcode(3);
    subItems.add(subItem1);

    Subitem subItem2=new Subitem();
    subItem2.setSubname("在线学习");
    subItem2.setItem(item);
    subItem2.setSubcode(3);
    subItems.add(subItem2);

    Subitem subItem3=new Subitem();
    subItem3.setSubname("书籍教程");
    subItem3.setItem(item);
    subItem3.setSubcode(3);
    subItems.add(subItem3);

    item.setSubItems(subItems);
    session.save(item);
```

```
    tx.commit();
    session.close();
}
```

6. 在 main()方法中调用 oneToMany()方法，可以看到在数据库中插入相关记录。

3.3 删除一级栏目中的“.net”项

注意：在数据库中插入两条语句：

```
Insert into item(itemname,itemcode)values('.net','.net 也不错');
Insert into subitem(subname,itemid,subcode)values('ajax',8,8);
```

1. 在 TestDAO 中编写 deleteItem()方法，代码如下：

```
public void deleteItem(){
    Session session=HibernateSessionFactory.getSession();
    Transaction tx=session.beginTransaction();
    Item item=(Item)session.get(Item.class,2);
    session.delete(item);
    tx.commit();
    session.close();
   }
```

2. 此时在 main()方法中调用 deleteItem()方法，在控制台可以看到如下信息：

```
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select item0_.itemid as itemid0_0_, item0_.itemname as itemname0_0_, item0_.itemcode as itemcode0_0_ fro
Exception in thread "main" java.lang.IllegalArgumentException: attempt to create delete event with null entity
        at org.hibernate.event.DeleteEvent.<init>(DeleteEvent.java:24)
        at org.hibernate.impl.SessionImpl.delete(SessionImpl.java:739)
        at com.hibernate3.dao.TestDAO.deleteItem(TestDAO.java:60)
        at com.hibernate3.dao.TestDAO.main(TestDAO.java:9)
```

图 2-39 TestDAO 执行 deleteItem()方法运行结果

3. 错误信息显示的 Sql 语句看到根本没有 delete 语句。如果想主表时级联删除从表怎么办呢？修改 Item.hbm.xml 文件，需要修改内容如下：

```
<set name="subItems" inverse="true" inverse="true">
          <key column="itemid"/>
          <one-to-many class="com.hibernate3.pojo.Subitem"/>
 </set>
```

4. 修改配置文件后保存，在执行该方法，可以看到顺利的将从表信息删除。

在实际的开发过程中，我们不会这样操作，在从表有数据的时候，是不允许将主表删除的，在这里只是通过这一另类的操作，引出配置文件中的两个属性 cascade 和 inverse。具体两个属性起到什么作用，在理论部分会有详细介绍。

3.4. 将“Ruby”子项“书籍教程”移动到“Java”项目中

分析：应该把“书籍教程”从“Ruby”中删除，然后再添加到“Java”子项中来。但是

使用 Hibernate 后，将配置文件写好，完成这件事情会很轻松。代码如下：

```
public void updateSubItem(){
    Session session=HibernateSessionFactory.getSession();
    Transaction tx=session.beginTransaction();
    //根据 sid，得到“书籍源码”
    Subitem item=(Subitem)session.get(Subitem.class,6);
       //得到“Java”的栏目项
    Item it=(Item)session.get(Item.class,1);
    item.setItem(it);
    tx.commit();
    session.close();
   }
```

执行此方法，会发现数据库端已经发生改变。

3.5. 建立学生表和教师表完成多对多关系映射

1. 在 mysql 中建立学生表和教师表，Sql 代码如下：

```
create table `mybbs`.`student`(
       `stuid` int not null auto_increment,
      `stuname` varchar(50) default '' not null,
       primary key (`stuid`)
   )TYPE=INNODB,
    default character set gbk;
create table `mybbs`.`teacher`(
       `teaid` int not null auto_increment,
      `teaname` varchar(50) default '' not null,
       primary key (`teaid`)
   )TYPE=INNODB,
    default character set gbk;
```

2. 要完成多对多的映射，需要一张中间表，建立中间表 tea_stu_relation。Sql 代码如下：

```
   create table `mybbs`.`tea_stu_relation`(
       `rid` int not null auto_increment,
      `stuid` int default '' not null,
      `teaid` int default '' not null,
       primary key (`rid`)
   );

   alter table `mybbs`.`tea_stu_relation`
       add index `fk_rt`(`teaid`),
       add constraint `fk_rt`
```

```
    foreign key (`teaid`)
    references `mybbs`.`teacher`(`teaid`)
    on delete cascade
    on update cascade ;
alter table `mybbs`.`tea_stu_relation`
    add index `fk_rs`(`stuid`),
    add constraint `fk_rs`
    foreign key (`stuid`)
    references `mybbs`.`student`(`stuid`)
    on delete cascade
    on update cascade ;
create unique index `PRIMARY` on `mybbs`.`tea_stu_relation`(`rid`);
create index `fk_rs` on `mybbs`.`tea_stu_relation`(`stuid`);
create index `fk_rt` on `mybbs`.`tea_stu_relation`(`teaid`);
```

3. 在 com.hibernate3.pojo 包内建立 Student，Teacher 两个类，同时在该包内建立 Student.hbm.xml 和 Teacher.java

Student 类代码如下：

```
public class Student implements java.io.Serializable {

    // Fields
    private Integer stuid;
    private String stuname;
   private  Set teachers=new HashSet();
    // Constructors
    /** default constructor */
    public Student() {
    }

    /** full constructor */
    public Student(String stuname) {
        this.stuname = stuname;
    }

    // Property accessors

    public Integer getStuid() {
        return this.stuid;
    }

    public void setStuid(Integer stuid) {
        this.stuid = stuid;
```

```
    }

    public String getStuname() {
        return this.stuname;
    }

    public void setStuname(String stuname) {
        this.stuname = stuname;
    }

    public Set getTeachers() {
        return teachers;
    }

    public void setTeachers(Set teachers) {
        this.teachers = teachers;
    }
}
```

注意：private Set teachers=new HashSet();

Student.hbm.xml 配置文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.hibernate3.pojo.Student" table="student"
                                                    catalog="mybbs">
        <id name="stuid" type="java.lang.Integer">
            <column name="stuid" />
            <generator class="native" />
        </id>
        <property name="stuname" type="java.lang.String">
            <column name="stuname" length="50" not-null="true" />
        </property>
        <set name="teachers" table="tea_stu_relation">
          <key column="stuid"/>
          <many-to-many class="com.hibernate3.pojo.Teacher" column="teaid"/>
        </set>
    </class>
```

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

注意：<set>元素中的配置

Teacher 类代码如下：

```
public class Teacher implements java.io.Serializable {

    // Fields

    private Integer teaid;
    private String teaname;
   private Set students=new HashSet();
    // Constructors

    public Set getStudents() {
        return students;
    }

    public void setStudents(Set students) {
        this.students = students;
    }

    /** default constructor */
    public Teacher() {
    }

    /** full constructor */
    public Teacher(String teaname) {
        this.teaname = teaname;
    }

    // Property accessors

    public Integer getTeaid() {
        return this.teaid;
    }

    public void setTeaid(Integer teaid) {
        this.teaid = teaid;
    }

    public String getTeaname() {
        return this.teaname;
```

```
    }

    public void setTeaname(String teaname) {
        this.teaname = teaname;
    }

}
```

注意：private Set students=new HashSet();

Teacher.hbm.xml 配置文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
   Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
   <class         name="com.hibernate3.pojo.Teacher"         table="teacher"
catalog="mybbs">
       <id name="teaid" type="java.lang.Integer">
          <column name="teaid" />
          <generator class="native" />
       </id>
       <property name="teaname" type="java.lang.String">
          <column name="teaname" length="50" not-null="true" />
       </property>
       <set name="students" table="tea_stu_relation" inverse="true">
         <key column="teaid"/>
         <many-to-many class="com.hibernate3.pojo.Student" column="stuid"/>
       </set>
   </class>
</hibernate-mapping>
```

注意：<set>元素配置。

4. 在 TestDAO 类中编写两个方法：一个是 teacherStu()方法查询某个老师教了哪些学生，另一个是 studentTea()方法查询某个学生由哪些老师教。

teacherStu()方法代码如下：

```
 public void teacherStu(){
   Session session=HibernateSessionFactory.getSession();
      Teacher teacher=(Teacher)session.get(Teacher.class, 1);
      Iterator item=teacher.getStudents().iterator();
```

```
        System.out.println("教师"+teacher.getTeaname()+"的学生有:");
        while(item.hasNext()){
         Student stu=(Student)item.next();
         System.out.println(stu.getStuname()+"\t");
        }
    }
```

测试该方法可以得到教师 id 为 1 的所有学生。

studentTea()方法代码如下：

```
public void studentTea(){
    Session session=HibernateSessionFactory.getSession();
       Student student=(Student)session.get(Student.class, 1);
       Iterator item=student.getTeachers().iterator();
       System.out.println("教授学生"+student.getStuname()+"的老师有:");
       while(item.hasNext()){
        Teacher tea=(Teacher)item.next();
        System.out.println(tea.getTeaname()+"\t");
       }
    }
```

测试该方法可以得到学生 id 为 1 的学生由哪些老师教授。

5. 由于工作需要教师 id 为 1 的老师调离工作，那么就要将这位老师从教师表中删除，同时也要把其教授的学生也要从 tea_stu_relation 表删除。在 TestDAO 中编写 deleteTeacher()方法，代码如下：

```
public void deleteTeacher(){
    Session session=HibernateSessionFactory.getSession();
     Teacher teacher=(Teacher)session.get(Teacher.class, 1);
     Transaction tx=session.beginTransaction();
     session.delete(teacher);
     tx.commit();
     session.close();
     }
```

测试运行此方法，可以看到 teacher 表中该教师数据被删除，同时 tea_stu_relation 表中涉及到该教师和与该教师相关的学生的信息也被删除了。为什么能够产生这样的效果，原因是在 Teacher.hbm.xml 文件中配置了 inverse="true"这个属性。

4. 相关理论知识

4.1 一对多关联

首先回顾一下面向对象领域内对象和对象间的关联关系。以我的论坛系统中栏目和子栏

目为例，栏目和子栏目是典型的一对多关系。一个栏目下有多个子栏目，多个子栏目同属于一个栏目。

另外，关联是有方向的。如图所示：

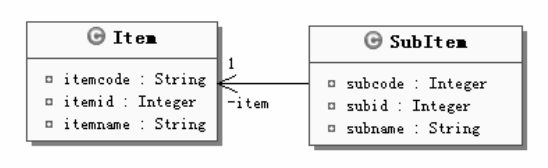


图 7-8 单向关联

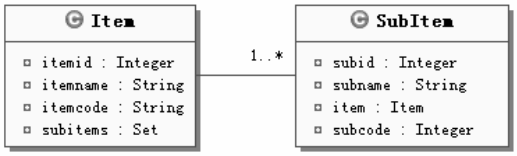


图 2-40 双向关联

如果仅有从 Item 到 Subitem 的关联，或者仅有从 Subitem 到 Item 的关联，就称之为单向关联。如果同时包含两种关联，就称之为双向关联。

4.1.1 多对一单向关联

在类与类之间的各种关系中，多对一的单向关联和关系数据库的外键参照关系最匹配了。如 3.1 示例中的 Item 类和 Subitem 类。

```
public class Item implements java.io.Serializable {
    private Integer itemid;
    private String itemname;
    private String itemcode;
}
```

```
public class Subitem implements java.io.Serializable {
    // Fields
    private Integer subid;
    private String subname;
    private Item item;
    private Integer subcode;
}
```

Item 类所有的属性和 item 表中的字段一一对应，因此把 Item 类映射到 item 表非常简单，如在 3.1 中的 Item.hbm.xml 映射文件所示：

```
<hibernate-mapping>
   <class name="com.hibernate3.pojo.Item" table="item" catalog="mybbs">
```

```
        <id name="itemid" type="java.lang.Integer">
            <column name="itemid" />
            <generator class="native" />
        </id>
        <property name="itemname" type="java.lang.String">
            <column name="itemname" length="50" not-null="true" />
        </property>
        <property name="itemcode" type="java.lang.String">
            <column name="itemcode" length="50" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

而 Subitem 类中，其中有个属性 item 属性其为 Item 类型，所以需要在配置文件中进行特殊处理，使用<many-to-one>元素来进行配置。配置如下：

```
<hibernate-mapping>
    <class name="com.hibernate3.pojo.Subitem" table="subitem"
                                                    catalog="mybbs">
        <id name="subid" type="java.lang.Integer">
            <column name="subid" />
            <generator class="native" />
        </id>
        <property name="subname" type="java.lang.String">
            <column name="subname" length="20" not-null="true" />
        </property>
        <property name="subcode" type="java.lang.Integer">
            <column name="subcode" not-null="true" />
        </property>
        <many-to-one name="item" column="itemid"
                                  class="com.hibernate3.pojo.Item"/>
    </class>
</hibernate-mapping>
```

<many-to-one>元素建立了 item 属性和 subitem 表的外键 itemid 的映射关系，其包括以下属性：

- name：设定待映射的持久化类的属性名，此处为 Subitem 类的 item 属性。
- column：设定和持久化类的属性对应的外键，此处为 subitem 表的外键 itemid。
- class：设定持久化类的属性类型，此处设定 item 类型为 Item。

通过以上步骤就配置成了单向的多对一的关系，也就实现了可以通过 Many 方得到 One 方相关的数据。

4.1.2 双向一对多关联

当建立了类与类之间的关系，就可以方便地从一个对象导航到另一个对象或者与其相关联的对象。例如在 3.1 示例中我们可以通过

```
Item item=subitem.getItem();
```

得到一个 Item 对象。现在如果想得到给定的 Item 对象想关联的 Subitem 对象怎么办呢？

这个时候需要在 Item 对象中建立一种一对多的关联。如在 3.2 示例中，Item 类中编写下：

```
private Set subItems=new HashSet();
public Set getSubItems() {
        return subItems;
    }

    public void setSubItems(Set subItems) {
        this.subItems = subItems;
    }
```

这样就可以查询到特定的 Item 下所有的 Subitem 信息。那么在配置文件中怎样进行映射呢？在 3.2 中映射代码如下：

```
<set name="subItems" >
          <key column="itemid"/>
          <one-to-many class="com.hibernate3.pojo.Subitem"/>
 </set>
```

<set>元素建立了 subitem 属性在数据库中的映射关系。

- set：表明 Item 类的 subitem 属性为 java.util.Set 集合类型。
- name：设定待映射的持久化类的属性名，此处为 Item 类的 subitem 属性。
- key：表明 subitem 表通过外键 itemid 参照 item 表
- one-to-many：表明 subItems 集合中放的是一组 Subitem 对象。

这样在 3.1 示例中在 Subitem 中添加了一个 Item 类型的属性，并在映射文件中进行了配置，现在又在 Item 中添加了一个集合属性，集合中是一组 Subitem 对象，也在映射文件中进行了配置，这样双向的一对多关系就映射成功了。

注意：既然是双向关联，“一对多的双向关联”和“多对一的双向关联”是同一回事，只不过“一对多关联”更加顺口而已。

通过这种配置就实现了 3.2 中配置的添加多个 Subitem 对象的示例。

在示例 3.3 中要删除一个 Item 元素，那么其相关的 Subitem 元素也要删除，在图 7-7 中看到如果进行配置的话，根本不能进行操作。当在 Item.hbm.xml 文件中配置如下属性：

```
<set name="subItems" inverse="true" cascade="all" >
          <key column="itemid"/>
```

```
            <one-to-many class="com.hibernate3.pojo.Subitem"/>
 </set>
```

即可完成删除操作。

inverse：表示关系的维护由谁来执行。true 表示不由自己执行，而由对应的另外一方执行。false 则相反，表示由自己维护关系。所以这里设置成 true。另外，

inverse 属性在<one-to-many>中，如果关系由 one 来维护，那么性能会非常低。

many 方每次进行增、删、改时都会多一次 update 操作，因为关系维护设在 one 这一方，所以对于 many 每一次的操作，one 这一方都要维护一次双方的关系。

cascade：表示是否进行级联操作。all：表示所有的操作都进行级联。

4.2 双向多对多关联

对于双向多对多关联，必须把其中一端的属性的 inverse 属性配置为 true,关联的两端都可以使用<set>元素。

如示例 3.5 在学校中存在一个老师可以教授多个学生，一个学生要上多个老师的课。这样学生和老师之间构成多对多的关系。如图：（uml 图制作中）

在数据库设计的时候，需要设计一个关联表 tea_stu_relation，通过关联表描述学生表和老师表的多对多关系。

数据库设计完毕，在程序中对于教师，学生两个实体中需要互相包含对方一个集合。

Student 类代码如下：

```
public class Student implements java.io.Serializable {
    // Fields
    private Integer stuid;
    private String stuname;
   private  Set teachers=new HashSet();
}
```

Teacher 类代码如下：

```
public class Teacher implements java.io.Serializable {
    private Integer teaid;
    private String teaname;
   private Set students=new HashSet();
}
```

这样其映射文件的配置方式和一对多很类似，也需要两个属性一个是 class 属性用来设置关联的属性的类型，column 属性用来设定用哪个字段去做外键关联，最后根据业务的需要，将某一方的 inverse 属性设为 false。

Teacher.hbm.xml 配置文件如下：

```
<set name="students" table="tea_stu_relation" inverse="true" cascade=all>
```

```
        <key column="teaid"/>
        <many-to-many class="com.hibernate3.pojo.Student" column="stuid"/>
      </set>
```

这里比配置一对多关联多一个 table 属性，table 指向数据库建立的关联的那张表。

Key 中的 column：关联表中和 teacher 表发生关系的字段。

Many-to-many 中的 column 指的是关联表中与 class（com.hibernate3.pojo.Student）关联的字段。

Student.hbm.xml 配置文件如下：

```
<set name="teachers" table="tea_stu_relation">
        <key column="stuid"/>
        <many-to-many class="com.hibernate3.pojo.Teacher" column="teaid"/>
</set>
```

通过上述配置就完成了双向多对多的配置。

5. 实验

1. 完成实践知识示例 3.1。（10 分钟）
2. 完成实践知识示例 3.2。（20 分钟）
3. 完成实践知识示例 3.3。（20 分钟）
4. 完成实践知识示例 3.4。（20 分钟）
5. 完成实践知识示例 3.5。（20 分钟）

6. 作业

1. 新来一位老师，他负责教授张无忌、赵敏、段誉。
2. 新来了一位同学，他要上欧阳锋老师，黄老邪老师，周伯通老师的课。
3. 同学杨过转学了，需要将这位同学从班级删除，同时告知教授他的老师。

案例四 Hibernate 高级查询

1. 教学目标

1.1 Hibernate 高级查询

2. 工作任务

2.1 使用 Hibernate 实现论坛版块管理

3. 相关实践知识

3.1 本章相关实例介绍

本章我使用 Struts 和 Hibernate 共同实现，论坛中论坛版块的管理。本章使用前面章节给出的 mybbs 数据库当中的 item 表。

其创建脚本如下

```
use mybbs;
create table `mybbs`.`item`(
       `itemid` int not null auto_increment,
      `itemname` varchar(50) default '' not null,
      `itemcode` varchar(50) default '' not null,
       primary key (`itemid`)
   )TYPE=INNODB,
    default character set gbk;
```

css 文件和程序页面我们会以课件形式发放到学员手中。

3.2 使用 MyEclipse6.0 实现实例功能

1. 打开 MyEclipse6.0，建立 Web 工程，并添加 Struts 和 Hibernate 支持。

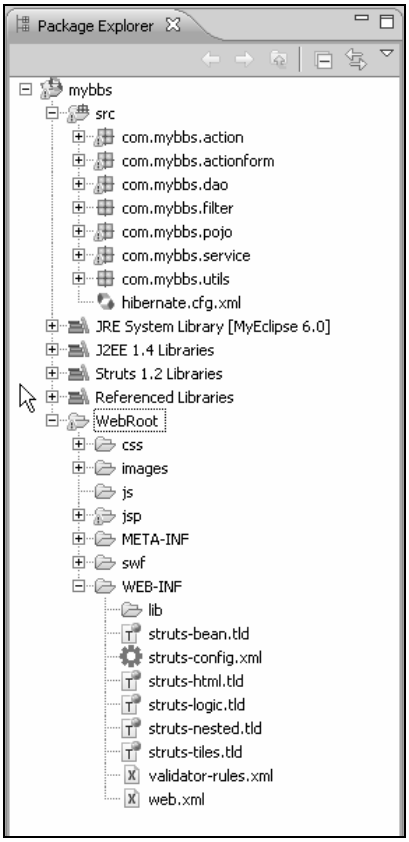


图 2-41 添加了 Struts 和 Hibernate 的 Web 工程

2. 实现这样的功能，我们首先需要一个 Web 页面，用以将论坛版块信息以表格的形式展现到客户端，在 WebRoot 下建立 Jsp 文件夹，并在其中建立 leibie.jsp：

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<%@ page import="com.mybbs.pojo.Item"%>
<%@ page import="java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
        <title>MyBBS 后台管理页面</title>
        <link href="<%=request.getContextPath()%>/css/main.css"
            rel="stylesheet" type="text/css" />
        <link href="<%=request.getContextPath()%>/css/lb.css"
                                rel="stylesheet" type="text/css" />
```

```
</head>
<body>
    <div id="bgmain">
        <div id="top">
            <div class="l">
                <a href="#" class="a">MyBBS 我的论坛</a>
            </div>
            <div class="r">
                <a href="#" class="a">登录论坛 </a>
            </div>
        </div>
        <div class="header">
        </div>
        <div id="content_tb">
            <h3 class="boxhd">
                帖子类别管理
            </h3>
            <div class="rtj">
                <a class="al"
href="<%=request.getContextPath()%>/jsp/addleibie.jsp"> 添加帖子类别</a>
            </div>
            <div class="rows">
                <ul>
                    <li class="t_leftli">
                        编号
                    </li>
                    <li class="t_leftli">
                        类别名称
                    </li>
                    <li class="t_leftli">
                        类别编码
                    </li>
                    <li class="t_leftli">
                        删除
                    </li>
                </ul>
                <%
                java.util.List list = (List)request.getAttribute("list");
                java.util.Iterator it = list.iterator();
                    while (it.hasNext()) {
                        Item item = (Item) it.next();
                %>
```

```
                    <ul>
                        <li class="leftli">
                            <%=item.getItemid()%>
                        </li>
                        <li class="leftli">
                            <%=item.getItemname()%>
                        </li>
                        <li class="leftli">
                            <%=item.getItemcode()%>
                        </li>
                        <li class="leftli">
                            <a
    href="<%=request.getContextPath()%>/ItemAction.do?method=beforeUpdateIt
em&id=<%=item.getItemid()%>&name=<%=item.getItemname()%>&code=<%=item.getIt
emcode()%>"class="al">修改</a>
                        </li>
                        <li class="leftli">
                            <a
    href="<%=request.getContextPath()%>/ItemAction.do?method=deleteItem&id=
<%=item.getItemid()%>"  class="al">删除</a>

                                                        </li>
                    </ul>
                    <%
                        }
                    %>
                </div>
                <div class="rtj">
                    <%
                        if
(!request.getAttribute("display").toString().equals("notdisplay")) {
                    %>
                    <a class="al"
    href="<%=request.getContextPath()%>/ItemAction.do?method=getItem&operat
er=first">
                        首页</a>
                    <%} %>

                    <%
                        if
(!request.getAttribute("display").toString().equals("notdisplay")         &&
Integer.parseInt(session.getAttribute("pageCount").toString()) != 1 ) {
```

```
                    %>
                    <a class="al"
    href="<%=request.getContextPath()%>/ItemAction.do?method=getItem&operat
er=previous">
                        上一页</a>
                    <%
                        }
                    %>
                    <%
                        if
(!request.getAttribute("display").toString().equals("notdisplay")         &&
Integer.parseInt(session.getAttribute("pageCount").toString()) != 0) {
                    %>
                    <a class="al"
    href="<%=request.getContextPath()%>/ItemAction.do?method=getItem&operat
er=next">
                        下一页</a>
                    <%
                        }
                    %>
                    <%
                        if
(!request.getAttribute("display").toString().equals("notdisplay")) {
                    %>
                    <a class="al"

    href="<%=request.getContextPath()%>/ItemAction.do?method=getItem&operat
er=end">
                        尾页</a>
                    <%} %>
                </div>
            </div>

            <div id="menu">
                <ul>
                    <li>
                        <a href="leibie.html" class="a2">类别管理</a>
                    </li>
                </ul>
                <div class="bt">
                        论坛内部信息
                </div>
```

```
                <div class="text_area">
                    <textarea name="textarea" cols="17" rows="8"></textarea>
                </div>
            </div>

            <div id="footer">
                <p>
                    版权所有：MyBBS 我的论坛
                </p>
            </div>
        </div>

    </body>
</html>
```

3. 建立用以添加论坛版块的页面 addleibie.jsp。

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
        <title>商业资讯后台管理主页面</title>
        <link href="<%=request.getContextPath()%>/css/main.css"
            rel="stylesheet" type="text/css" />
        <link href="<%=request.getContextPath()%>/css/tjgl.css"
            rel="stylesheet" type="text/css" />
    </head>

    <body>
        <div id="bgmain">
            <div id="top">
                <div class="l">
                    <a href="#" class="a">MyBBS 我的论坛</a>
                </div>
                <div class="r">
                    <a href="#" class="a">登录论坛 </a>
                </div>
            </div>
            <div class="header">
            </div>
```

```
            <div id="content_tb">
                <h3 class="boxhd">
                    帖子类别管理
                </h3>
                <div class="xbt">
                    添加类别
                </div>
    <formaction="<%=request.getContextPath()
%>/ItemAction.do?method=addItem" method="post"  >
                <div class="row02">
                    <div class="row01_l">
                        类别名称:
                    </div>
                    <div class="row01_r">
                    <input name="subname" class="box" id=xname size="20" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row01_l">
                        类别编码:
                    </div>
                    <div class="row01_r">
                    <input name="subcode" class="box" id=xname size="20" />
                    </div>
                </div>
                <div class="row02">
                    <div class="row03_c">
                        <input type="submit"  value="保 存" />
                         
                        <input type="reset"  value="取 消" />
                    </div>
                </div>
                </form>
            </div>
            <div id="menu">
                <ul>
                    <li>
                        <a href="leibie.html" class="a2">类别管理</a>
                    </li>
                </ul>
                <div class="bt">
                        论坛内部信息
```

```
                </div>
                <div class="text_area">
                    <textarea name="textarea" cols="17" rows="8"></textarea>
                </div>
            </div>
            <div id="footer">
                <p>
                    版权所有：MyBBS 我的论坛
                </p>
            </div>
        </div>

    </body>
</html>
```

4. 新建 com.mybbs.pojo 包并在包下建立叫作 Item 的 pojo 类，这个类就代表帖子的版块描述信息，其中包括如下字段：

```
package com.mybbs.pojo;
public class Item implements java.io.Serializable {
    //版块 id
    private Integer itemid;
    //版块名称
    private String itemname;
    //版块编号
    private String itemcode;

    public Item() {
    }

    public Item(Integer itemid, String itemname, String itemcode) {
        this.itemid = itemid;
        this.itemname = itemname;
        this.itemcode = itemcode;
    }

    public Integer getItemid() {
        return this.itemid;
    }

    public void setItemid(Integer itemid) {
        this.itemid = itemid;
```

```
    }

    public String getItemname() {
        return this.itemname;
    }

    public void setItemname(String itemname) {
        this.itemname = itemname;
    }

    public String getItemcode() {
        return this.itemcode;
    }

    public void setItemcode(String itemcode) {
        this.itemcode = itemcode;
    }
}
```

5. 新建 Item.hbm.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="com.mybbs.pojo.Item" table="item" catalog="mybbs">
      <id name="itemid" type="java.lang.Integer">
         <column name="itemid" />
         <generator class="increment" />
      </id>
      <property name="itemname" type="java.lang.String">
         <column name="itemname" length="50" not-null="true" />
      </property>
      <property name="itemcode" type="java.lang.String">
         <column name="itemcode" length="50" not-null="true" />
      </property>
   </class>
</hibernate-mapping>
```

6. 新建 com.mybbs.dao 包并在包下建立 HibernateDAO 类，用于操作数据库：

```
package com.mybbs.dao;
```

```
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.mybbs.pojo.Item;
import com.mybbs.utils.PageUtil;

public class HibernateDAO {
    //用于单例模式
    private static HibernateDAO instance = null;
    //Session 工厂用于生产 Session
    private static SessionFactory fac = null;
    //静态块
    static {
        try {
            Configuration cfg = new Configuration().configure();
            fac = cfg.buildSessionFactory();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private HibernateDAO() {
    }

    public static HibernateDAO getInstance() {
        if (instance == null) {
            instance = new HibernateDAO();
        }
        return instance;
    }
    //得到版块信息
    public List selectItem(int start) {
        //打开 Session
        Session session = fac.openSession();
        List itemList = null;
        try {
            //开始事务
            Transaction tra = session.beginTransaction();
```

```
            //使用 Hibernate 的分页查询
            Query query = session.createQuery("from Item");
            query.setFirstResult(start);
            query.setMaxResults(PageUtil.pageSize);
            //获得数据
            itemList = query.list();
            //提交事务
            tra.commit();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭 Session
            session.close();
        }
        return itemList;
    }

    // 得到所有帖子版块信息
    public List selectAllItem() {
        //打开 Session
        Session session = fac.openSession();
        List itemList = null;
        try {
            //开始事务
            Transaction tra = session.beginTransaction();
            Query query = session.createQuery("from Item");
            //获得数据
            itemList = query.list();
            //提交事务
            tra.commit();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭 Session
            session.close();
        }
        return itemList;
    }

    // 添加一个论坛版块
    public void insertItem(Item item) {
        //打开 Session
```

```
        Session session = fac.openSession();
        try {
            //开始事务
            Transaction tra = session.beginTransaction();
            //保存对象
            session.save(item);
            //提交事务
            tra.commit();

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭 Session
            session.close();
        }
    }

    // 删除一个论坛版块
    public void deleteItem(Item item) {
        //打开 Session
        Session session = fac.openSession();
        try {
            //开始事务
            Transaction tra = session.beginTransaction();
            //删除数据
            Query query = session.createQuery("delete Item i where
i.itemid=?");
            query.setInteger(0, item.getItemid());
            query.executeUpdate();
            //提交事务
            tra.commit();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭 Session
            session.close();
        }
    }
}
```

7. 新建 com.mybbs.service 包并在包下建立 ItemService 类，用于封装持久化代码，进而完成对论坛版块信息的增删查，其中键入如下代码：

```
package com.mybbs.service;

import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import com.mybbs.dao.HibernateDAO;
import com.mybbs.pojo.Item;
import com.mybbs.utils.PageUtil;

public class ItemService {

    //分页显示版块信息
    public List selectItem(HttpSession session, HttpServletRequest request) {
        //本页显示第一条数据在数据库中的索引
        int start = 0;
        //当前页索引
        int pageCount = PageUtil.firstPageIndex;
        //最大页数
        int maxPageCount = 0;
        //获得数据库中所有数据
        List list = HibernateDAO.getInstance().selectAllItem();
        int count = list.size();
        //计算最大页数
        if (count % PageUtil.pageSize == 0) {
            maxPageCount = count / PageUtil.pageSize;
        } else {
            maxPageCount = count / PageUtil.pageSize + 1;
        }

        //如果最大页数大于1,分页显示数据
        if (maxPageCount > 1) {
            //分页按钮显示
            request.setAttribute("display", "display");
            //将当前页索引信息存入session，如果为空说明第一次访问本页面，创建session，
将1存入
            if (session.getAttribute("pageCount") == null) {
                session.setAttribute("pageCount", PageUtil.firstPageIndex);
            } else {
                //如果点击首页按钮
                if
(request.getParameter("operater").toString().equals("first")) {
                    //第一页
```

```
                    pageCount = PageUtil.firstPageIndex;
                    session.setAttribute("pageCount", pageCount);
                }//点击最后一页按钮
                else if (request.getParameter("operater").toString().equals(
                        "end")) {
                    //最后一页
                    pageCount = maxPageCount;
                    session.setAttribute("pageCount",
PageUtil.endPageIndex);
                }//点击上一页
                else if (request.getParameter("operater").toString().equals(
                        "previous")) {
                    //如果当前页是最后一页，我们用 0 代表最后一页的页数
                    if (Integer.parseInt(session.getAttribute("pageCount")
                            .toString()) == 0) {
                        //当前页减一
                        pageCount = maxPageCount - 1;
                        session.setAttribute("pageCount", pageCount);
                    } else {
                        session.setAttribute("pageCount", Integer
                                .parseInt(session.getAttribute("pageCount")
                                        .toString()) - 1);
                        pageCount = Integer.parseInt(session.getAttribute(
                                "pageCount").toString());
                    }

                }//点击下一页
                else if (request.getParameter("operater").toString().equals(
                        "next")) {
                    //当前页加一
                    session.setAttribute("pageCount",
Integer.parseInt(session
                            .getAttribute("pageCount").toString()) + 1);
                    pageCount = Integer.parseInt(session.getAttribute(
                            "pageCount").toString());
                    //如果当前页数等于最大页数，说明是最后一页。我们将 session 中的
pageCount 设为 0
                    if (pageCount == maxPageCount) {
                        session.setAttribute("pageCount",
PageUtil.endPageIndex);
                    }
                }
```

```
            }
        }
        else
        {
            //数据不够两页所以不用分页
            pageCount=maxPageCount;
            request.setAttribute("display", "notdisplay");
        }
        start = (pageCount * PageUtil.pageSize) - PageUtil.pageSize;
        return HibernateDAO.getInstance().selectItem(start);
    }

    //添加一个论坛版块
    public void addItem(Item item)
    {
        HibernateDAO.getInstance().insertItem(item);
    }
    //删除一个论坛版块
    public void deleteItem(Item item)
    {
        HibernateDAO.getInstance().deleteItem(item);
    }
}
```

8. 新建 com.mybbs.actionform 包并在包下建立 Itemform 类，键入如下代码：

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

public class Itemform extends ActionForm {
    //帖子类别 id
    private int    subid;
    //帖子类别名称
    private String subname;
    //帖子编号
    private int    subcode;

    public int getSubid() {
        return subid;
    }
    public void setSubid(int subid) {
```

```
        this.subid = subid;
    }
    public String getSubname() {
        return subname;
    }
    public void setSubname(String subname) {
        this.subname = subname;
    }
    public int getSubcode() {
        return subcode;
    }
    public void setSubcode(int subcode) {
        this.subcode = subcode;
    }
    //actioform 特有的验证方法
    public  ActionErrors  validate(ActionMapping  mapping,HttpServletRequest
request)
    {
        return null;
    }
    //actionform 特有的重置方法
    public void reset(ActionMapping mapping,HttpServletRequest request)
    {
    }
}
```

9. 新建 com.mybbs.utils 包并在包下建立 pageUtil 类，键入如下代码：

```
package com.mybbs.utils;

public class PageUtil {
    //每页显示数据条数
    public static final int pageSize=3;
    //第一页
    public static final int firstPageIndex=1;
    //最后一页
    public static final int endPageIndex=0;
}
```

10. 新建 com.mybbs.action 包并在包下建立 ItemAction 类，注意此类继承自 org.apache.struts.actions.DispatchAction，在其中键入如下代码：

```
package com.mybbs.action;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.mybbs.actionform.Itemform;
import com.mybbs.pojo.Item;
import com.mybbs.service.ItemService;
import javax.servlet.http.HttpSession;
import org.apache.struts.actions.DispatchAction;
import org.apache.struts.action.Action;

public class ItemAction extends DispatchAction {
    //添加帖子类别
    public ActionForward addItem(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        HttpSession session = request.getSession();
        //得到有关的 actionform 对象
        Itemform itemform = (Itemform) form;
        //声明有关帖子类别的 pojo 对象
        Item item = new Item();
        item.setItemname(itemform.getSubname());
        item.setItemcode(itemform.getSubcode());
        //声明业务逻辑对象
        ItemService itemSerivce = new ItemService();
        itemSerivce.addItem(item);
        session.setAttribute("pageCount", null);
        return mapping.findForward("updatesuccess");
    }

    public ActionForward deleteItem(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        HttpSession session = request.getSession();
        String itemid = request.getParameter("id");
        //声明有关帖子类别的 pojo 对象
        Item item = new Item();
        item.setItemid(Integer.parseInt(itemid));
        //声明业务逻辑对象
        ItemService itemSerivce = new ItemService();
        itemSerivce.deleteItem(item);
```

```
        session.setAttribute("pageCount", null);
        return mapping.findForward("deletesuccess");
    }

    public ActionForward getItem(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {

        //声明业务逻辑对象
        ItemService itemSerivce = new ItemService();
        //获得 session 对象
        HttpSession session = request.getSession();
        java.util.List<Item> list = itemSerivce.selectItem(session, request);
        request.setAttribute("list", list);
        return mapping.findForward("success");
    }
}
```

11. 打开 WEB-INF 当中的，struts-config.xml 文件，在其中键入如下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration                                                    1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  <form-beans >
    <form-bean name="ItemForm" type="com.mybbs.actionform.Itemform" />
  </form-beans>
  <action-mappings >
   <action
      attribute="ItemForm"
      name="ItemForm"
      path="/ItemAction"
      scope="request"
      parameter="method"
      type="com.mybbs.action.ItemAction" >
         <forward name="success" path="/jsp/leibie.jsp"></forward>
         <forward name="update" path="/jsp/updateleibie.jsp"></forward>
         <forward  name="updatesuccess"  path="/ItemAction.do?method=getItem"
></forward>
        <forward  name="deletesuccess"  path="/ItemAction.do?method=getItem">
        </forward>
    </action>
   </action-mappings>
```

```
  <message-resources parameter="com.mybbs.struts.ApplicationResources" />
</struts-config>
```

12. hibernate.cfg.xml 配置信息如下：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                 -->
<hibernate-configuration>

    <session-factory>
        <property name="connection.username">root</property>
        <property name="connection.url">
            jdbc:mysql://localhost:3306/mybbs
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="myeclipse.connection.profile">MySQL</property>
        <property name="connection.password">sa</property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <mapping resource="com/mybbs/pojo/Item.hbm.xml" />

    </session-factory>

</hibernate-configuration>
<message-resources parameter="com.mybbs.struts.ApplicationResources" />
</struts-config>
```

13. 部署运行程序，在地址栏里输入 http://localhost:8080/mybbs/ItemAction.do?method=getItem 打开查询页面，点击红色链接，为论坛添加版块。

图 2-42 查询论坛版块信息

14. 输入版块名称和编码后，点击“保存”。

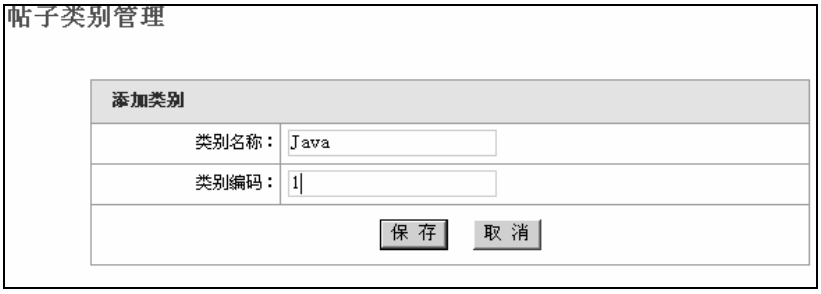


图 2-43 添加论坛版块

15. 要删除论坛版块点击红色“删除”链接便可。

图 2-44 删除论坛版块

16. 如果数据量大的话程序会分页显示。

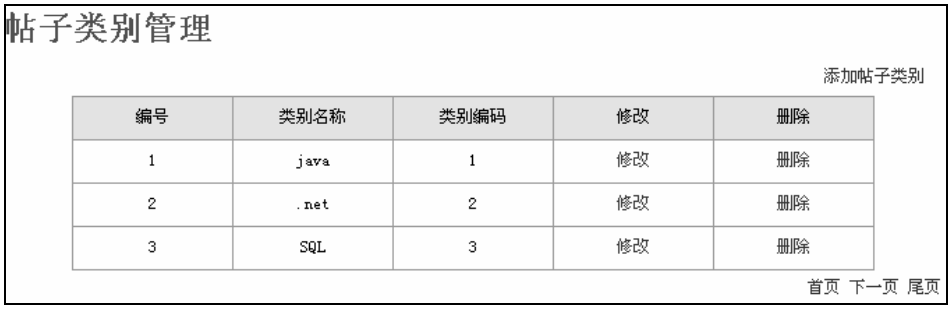


图 2-45 分页显示数据

4. 相关理论知识

4.1 Hibernate 的检索方式

1. OID 检索方式

OID 对象标始符，按照对象的 OID 来检索对象。Session 的 get()和 load()方法提供了这种功能。如果在应用程序中事先知道了 OID，就可以使用这种检索对象的方式。

2. HQL 检索方式

使用面向对象的 HQL 查询语言。Session 的 find()方法用于执行 HQL 查询语句。此外，Hibernate 还提供了 Query 接口，它是 Hibernate 提供的专门的 HQL 查询接口，能够执行各种复

杂的 HQL 查询语句。

3. QBC 检索方式

使用（Query By Criteria）API 来检索对象。这种 API 封装了基于字符串形式的查询语句，提供了更面向对象的接口。

4. 本地 SQL 检索方式

使用本地数据库的 SQL 查询语句。Hibernate 会负责把检索到的 JDBC ResultSet 结果集映射成为对象图。

我们将在本章主要介绍 HQL 检索方式、QBC 检索方式和本地 SQL 检索方式，重点介绍了 HQL 查询的语法，此外还介绍了各种检索方式的优缺点，指出了各自的使用场合。

4.2 HQL 检索方式

HQL（Hibernate Query Language）是面向对象的查询语言，它和 SQL 查询语言有些相似。在 Hibernate 提供的各种检索方式中，HQL 是使用最广泛的一种检索方式。它具有以下功能：

- 在查询语句中设置各种查询条件。
- 支持投影查询，即仅检索出对象的部分属性。
- 支持分页查询。
- 支持连接查询。
- 支持分组查询，允许使用 having 和 ground by 关键字。
- 提供内置聚集函数，如 sum()、min()和 max()。
- 能够调用用户定义的 SQL 函数。
- 支持子查询，即嵌入式查询。
- 支持动态绑定参数。

Session 类的 find()方法及 Query 接口都支持 HQL 检索方式。这两者的区别在于，前者只是执行一些简单 HQL 查询语句和便捷方法，它不具有动态绑定参数的功能，而且在将来新的 Hibernate 中，有可能淘汰 find()方法；而 Query()接口才是真正的 HQL 查询接口，它提供了以上列出的各种查询功能。

以下程序代码用于检索姓名为“soctt”，并且密码为“tiger”的 Userinfo 对象。

```
//创建一个 Query 对象。
Query query=session.createQuery("from Userinfo as u where u.username=? and
u.userpwd=?");
//动态绑定参数
query.setString(0, "scott");
query.setString(1,"tiger");
//执行查询语句，返回结果集
List result=query.list();
```

HQL 检索方式包括以下步骤：

1. 通过 Session 的 createQuery()方法创建一个 Query 对象，它包含一个 HQL 查询语句。HQL 查询语句可以包含占位符“？”。

2. 动态绑定参数。Query 接口提供各种类型的参数赋值语法，例如 setString()方法用于为 HQL 语句中的“username”参数赋值。

3. 调用 Query 的 list()方法执行查询语句。该方法返回 List 类型的查询结果。在 List 集合中存放了符合查询条件的持久化对象。执行以上代码，当运行 Query 的 list()方法时，Hibernate 执行以下 SQL 查询语句：

```
Select * from Userinfo where username=”scott” and userpwd=”tiger”
```

Query 接口支持方法链编程风格，它的 setString()方法及其他的 setXXX()方法都返回自身实例，而不是返回 void 类型。以下是 Query 接口的实现类 setString()方法的源程序。

```
public Query setString(int position,String val)
{
          setParameter(position,val,Hibernate.STRING);
          retrun this;
}
```

如果采用方法链编程风格，将按以下形式访问 Query 接口：

```
Query query=session.createQuery("from Userinfo as u where u.username=? and
u.userpwd=?").setString(0, "scott").setString(1,"tiger");
```

可见，方法链编程风格能使程序代码更加简洁。

4.3 QBC 检索方式

采用 HQL 检索方式时，在应用程序中需要定义基于字符串形式的 HQL 查询语句。QBC API 提供了检索对象的另一种方法，它主要由 Criteria 接口、Criterion 接口和 Expression 类组成，它支持在运行时动态生成查询语句。

```
//创建一个 Criteria 对象。
Criteria criteria=session.createCriteria(Userinfo.class);
//设置查询条件，然后把查询条件加入 Criteria 中
Criterion criterion1=Expression.like("username","s%");
Criterion criterion2=Expression.like("userpwd","tiger");

criteria.add(criterion1);
criteria.add(criterion2);

//执行查询语句，返回结果集
List result= criteria.list();
```

QBC 检索方式包括以下步骤：

1. 通过 Session 的 createCriteria()方法创建一个 Criteria 对象。

2. 设定查询条件。Expression 类提供了一系列用于设定查询条件的静态方法，这些静态方法都返回 Criterion 的实例，每个 Criterion 代表一个查询条件。Criterion 的 add()方法用于加入查询条件。

3. 调用 Criteria 的 list()方法执行查询语句。该方法返回 List 类型的查询结果，在 List 集合中存放了符合查询条件的持久化对象。对于执行以上的代码，当运行 Criteria 的 list()方法时，Hibernate 执行的 SQL 查询语句为：

```
Select * from Userinfo where username like ”%s” and userpwd=”tiger”
```

Criteria 接口支持方法链编程风格，它的 add()方法返回自身实例，而不是返回 void 类型。以下是 Criteria 接口的实现类 add()方法的源程序：

```
public Criteria add(Criterion expression)
{
CriteriaImpl.this.add(rootAlias,expression);
return this;
}
```

如果采用编程链编程风格，将按以下形式访问 Criteria 接口：

```
List                                                                    result=
session.createCriteria(Userinfo.class).add(Expression.like("username","s%")
).add(Expression.like("userpwd","tiger")).list();
```

Hibernate 还提供了 QBE（Query By Example）检索方式，它是 QBC 的子功能。QBE 允许先创建一个对象样板，然后检索出所有和这个样板相同的对象。

```
Userinfo user=new Userinfo();
            user.setUsername("scott");
            user.setUserpwd("tiger");
            //执行查询语句，返回结果集
            List result=
   session.createCriteria(Userinfo.class).add(Example.create(user)).list();
```

在 QBE API 中，Example 接口的静态方法 create()创建一个 Criterion 对象，它代表按照样板对象的属性来比较的查询条件。对以上程序，Hibernate 执行的 SQL 查询语句为：

```
Select * from Userinfo where username=”scott” and userpwd=”tiger”
```

从以上查询语句看出，在默认条件下，Hibernate 能把 Userinfo 对象中所有不为 Null 的属性作为查询条件。

QBE 的功能不是特别强大，仅在某些场合下有用。一个典型的使用场合是在查询窗口中让用户输入一系列的查询条件，然后返回匹配的对象。

4.4 SQL 检索方式

采用 HQL 或 QBC 检索方式时，Hibernate 会生成标准的 SQL 查询语句，适用于所有的数

据库平台，因此这两种检索方式是跨平台的。

有的应用程序可能需要根据底层数据库的 SQL 方言，来生成一些特殊的查询语句。在这种特殊的情况下。可以利用 Hibernate 提供的 SQL 检索方式。以下程序用于检索用户名为“soctt”的用户的密码。

```
//创建 Query 对象
Query query= session.createSQLQuery("Select u.userpwd from Userinfo u where
username=?");
//动态绑定参数
query.setString(0, "scott");
//执行 SQL select 语句，返回结果集
List list=query.list();
```

从以上代码看出，SQL 检索方式与 HQL 检索方式都使用 Query 接口，区别在于 SQL 检索方式通过 Session 的 createSQLQuery()方式创建 Query 对象，这个方法的参数指定一个 SQL 查询语句，该语句使用本地数据库的 SQL 方言。

4.5 使用别名

最简单的查询检索一个持久化类的所有实例。例如：

```
//采用 HQL 检索方式
List result=session.createQuery("from Userinfo").list();
//采用 QBC 方式
List secondresult=session.createCriteria(Userinfo.class).list();
```

通过 HQL 检索一个类的实例时，如果查询语句的其他地方需要引用它，应该为这个类指定一个别名，例如：

```
from Userinfo as u where u.username=?
```

as 关键字用于设定别名，也可以将 as 关键字省略：

```
from Userinfo  u where u.username=?
```

在实际应用中，建议使用别名与类名相同，例如 Userinfo 类赋予别名“userinfo”，而不是别名“u”：

```
from Userinfo as userinfo where userinfo.username=?
```

QBC 检索方式不需要由应用程序显示指定类的别名，Hibernate 会自动把查询语句中的根节点实体赋予别名“this”。例如以下程序代码中 Expression 类的 eq()方法的第一个参数为“username”，它代表 Userinfo 类的 username 属性：

```
List result=session.createCriteria(Userinfo.class)
.add(Expression.eq("username","scott"))
.list();
```

Hibernate 为 Userinfo 类自动赋予别名“this”，因此程序中也可以按照“this.username”的形式引用 Userinfo 的 username 属性：

```
List result=session.createCriteria(Userinfo.class)
.add(Expression.eq("this.username","scott"))
.list();
```

4.6 多态查询

HQL 和 QBC 都支持多态查询，多态查询是指查询出当前类及所有子类的实例，对于以下代码：

```
//采用 HQL 查询
session.createQuery(“from Employee”);
//采用 QBC 检索方式
session.createCriteria(Employee.class);
```

加入 Employee 有两个子类：HourlyEmployee 和 SalariedEmployee，那么这个查询语句会查询出所有 Employee 类的实例，以及 HourlyEmployee 类和 SalariedEmployee 类的实例。如果只想检索某个特定类的实例，可以使用如下方式 ：

```
//采用 HQL 查询
session.createQuery(“from HourlyEmployee”);
//采用 QBC 检索方式
session.createCriteria(HourlyEmployee.class);
```

以下 HQL 查询语句将检索出所有的持久化对象：

```
from java.lang.Object
```

多态查询对接口也适用。例如，以下查询语句查询出所有实现 Serializable 接口的实例：

```
from java.io.Serializable
```

Hibernate 不仅对 from 子句中显示指定的类进行多态查询，而且对其他关联的类也会进行多态查询。

4.7 对查询结果排序

HQL 和 QBC 都支持对查询结果的排序。HQL 采用 order by 关键字对查询结果排序，而 QBC 采用 org.hibernate.expression.Order 类对查询结果排序，下面举例说明它们的用法。

1. 查询结果按照用户名升序排列：

```
//HQL 查询方式
Query query=session.createQuery(“form Userinfo as u order by  u.username”);
//QBC 查询方式
Criteria criteria=session.createCriteria(Userinfo.class);
criteria.addOrder(Order.asc(“username”));
```

2. 查询结果按用户名升序排序，并且按照密码倒序排列：

```
//HQL 查询方式
Query query=
session.createQuery(“form Userinfo as u order by  u.username asc,u.userpwd
desc”);
//QBC 查询方式
Criteria criteria=session.createCriteria(Userinfo.class);
criteria.addOrder(Order.asc(“username”));
criteria.addOrder(Order.asc(“userpwd”));
```

如果应用程序中既通过 import 语句引入了 org.hibernate.expression.Order 类，又通过 import 语句引入了 mypack.Order 类，在程序中引用这两个类名时，必须给出完整的包路径，以便区分这两个 Order 类，例如：

```
import  org.hibernate.expression.Order;
import  mypack.Order;
Criteria criteria=session.createCriteria(mypack.Orde.class);
criteria.addOrder(org.hibernate.expression.Order.asc(“username”));
```

4.8 分页查询

当批量查询数据库时，如果数据量很大，会导致无法在用户终端的单个页面上显示所有的查询结果，此时需要对查询结果分页。假如 userinfo 表中有 99 条记录，可以在用户终端上分 10 页来显示数据，每页最多显示 10 个 userinfo 对象，用户可以导航到下一页。Query 和 Criteria 接口都提供了用于分页显示查询结果的方法。

1. setFirstResult(int firstResult)：设置从那个对象开始检索，参数 firstResult 表示这个对象在查询结果中的索引位置，索引位置的起始值为 0。在默认情况下，Query 和 Criteria 接口从查询结果中的第一个对象，也就是索引位置为 0 的对象开始检索。

2. setMaxResult(int maxResults)：设定一次最多检索出的对象数目。在默认情况下，Query 和 Criteria 接口检索出查询结果中所有对象。

以下代码从查询结果的起始对象开始，共检索出 5 个 userinfo 对象，查询结果按照 username 属性排序：

```
//采用 HQL 检索方式
Query query= session.createQuery("from Userinfo u order by u.username");
query.setFirstResult(0);
query.setMaxResults(5);
List list=query.list();
//采用 QBC 检索方式
Criteria criteria=session.createCriteria(Userinfo.class);
criteria.addOrder(Order.asc("username"));
criteria.setFirstResult(0);
criteria.setMaxResults(5);
```

```
List list=criteria.list();
```

如果查询结果中共有 99 个 userinfo 对象，那么在 list 中包含 5 个 userinfo 对象，第 1 个对象在查询结果中的索引位置为 0，第 5 个对象在查询结果中的索引位置为 4。

以下代码从查询结果中索引位置为 97 的对象开始，共检索出 10 个 userinfo 对象：

```
//采用 HQL 检索方式
Query query= session.createQuery("from Userinfo u order by u.username");
query.setFirstResult(97);
query.setMaxResults(10);
List list=query.list();
//采用 QBC 检索方式
Criteria criteria=session.createCriteria(Userinfo.class);
criteria.addOrder(Order.asc("username"));
criteria.setFirstResult(97);
criteria.setMaxResults(10);
List list=criteria.list();
```

如果查询结果中共有 99 个对象，那么在 result 中只包含两个 userinfo 对象，它们的索引为 97 和 98。

对于以上程序代码，也可以采用方法链编程风格：

```
//采用 HQL 检索方式
Query query= session.createQuery("from Userinfo as u order by u.username")
.setFirstResult(0)
.setMaxResults(5);
//采用 QBC 检索方式
Criteria criteria=session.createCriteria(Userinfo.class)
.addOrder(Order.asc("username"))
.setFirstResult(0)
.setMaxResults(5);
```

4.9 检索单个对象

Query 和 Criteria 接口都提供了以下用于执行查询语句并返回查询结果的方法。

1. list()方法：返回一个 List 类型的查询结果，在 List 集合中存放了所有满足查询条件的持久化对象。

2. uniqueResult()方法：返回单个对象。

在某种情况下，如果希望检索出一个对象，可与先调用 Query 或 Criteria 接口的 setMaxResult(1)方法，把最大检索数目设为 1：

```
setMaxResult(1);
```

接下来调用 uniqueResult 方法，该方法返回一个 Object 类型的对象：

```
//采用 HQL 检索方式
Userinfo user= (Userinfo)session.createQuery("from Userinfo as u order by
u.username")
.setMaxResults(1)
.uniqueResult();
//采用 QBC 检索方式
Userinfo user=(Userinfo)session.createCriteria(Userinfo.class)
.addOrder(Order.asc("username"))
.setMaxResults(1)
.uniqueResult();
```

如果明确知道查询结果只会包含一个对象，可以不调用 setMaxResults(1)方法，例如：

```
Userinfo user= (Userinfo)session.createQuery("from Userinfo as u where
u.userid=1").uniqueResult();
```

以下查询结果包含多个 Customer 对象，但是没有调用 setMaxResults(1)方法：

```
Userinfo user= (Userinfo)session.createQuery("from Userinfo").uniqueResult();
```

执行以上 uniqueResult()方法时，会抛出 NonUniqueResultException()异常：
org.hibernate.NonUniqueResultException: query did not return a unique result: 13

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 在 mybbs 数据库中建立 item 表并建立 MyBBS 工程。（10 分钟）
2. 编写 leibie.jsp，addleibie.jsp 页面。（30 分钟）
3. 编写用于数据访问的 HibernateDAO 类。（10 分钟）
4. 编写业务逻辑类 ItemService 和 pageUtil 工具类。（10 分钟）
5. 编写 ItemAction 和 Itemform 类。（20 分钟）
6. 配置 struts-config.xml，hibernate.cfg.xml 文件，并调试运行程序。（10 分钟）

6. 课后作业

完成 mybbs 论坛版块管理中的修改功能。

专题三 Spring

教学目标

掌握 Spring IoC 原理

掌握 Spring AOP 原理

案例一 Spring IoC 基础

1. 教学目标

1.1 使用工具如何建立一个 Spring 工程

1.2 Spring 的 IoC

2. 工作任务

2.1 使用 MyEclipse 添加 Spring 开发环境

2.2 Spring 第一个“Hello World”程序

2.3 Spring IoC 实现简单员工管理程序

3. 相关实践知识

3.1 使用 MyEclipse 添加 Spring 开发环境

MyEclipse 为我们提供了 Spring 支持环境的构建向导，打开 MyEclipse6.0，创建步骤具体如下：

1. 首先建立一个 Web 项目，点击菜单栏【File】➔【New】➔【Web Project】，在 Project Name 当中输入项目名称“SpringTest”，然后点击【Finish】。

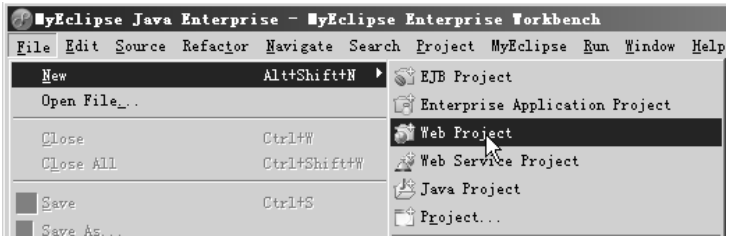


图 3-1 建立 Web 工程

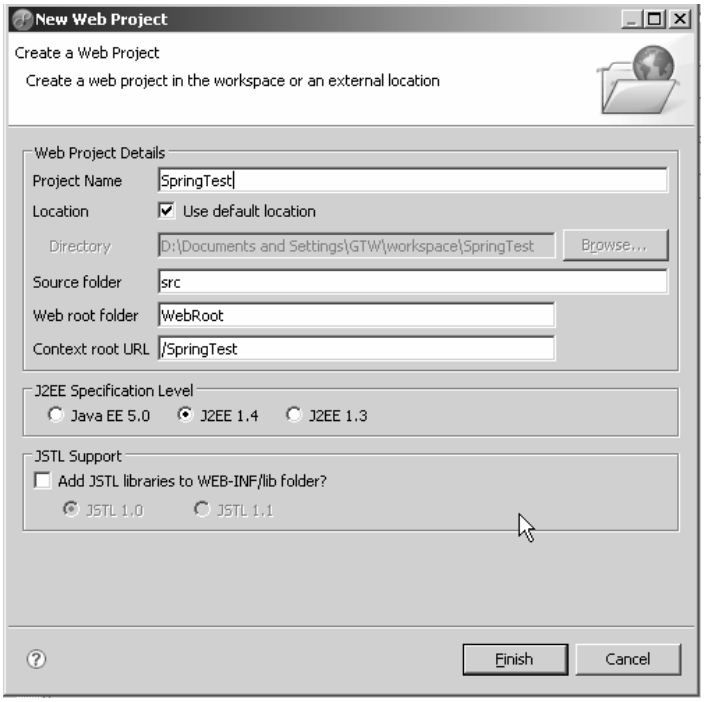


图 3-2 命名 Web 工程

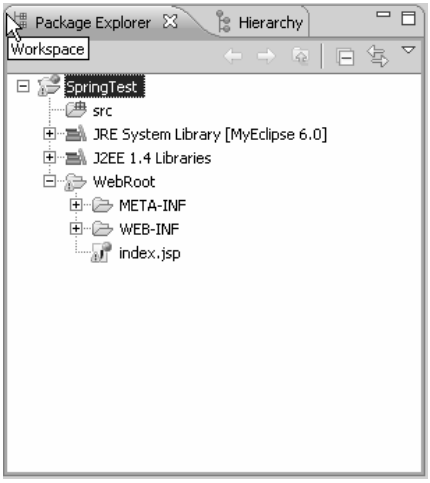


图 3-3 Web 项目结构

2. 然后在项目中加入 Spring 支持，右键点击项目根节点【Spring】➔【MyEclipse】➔【Add Spring Capabilities】。

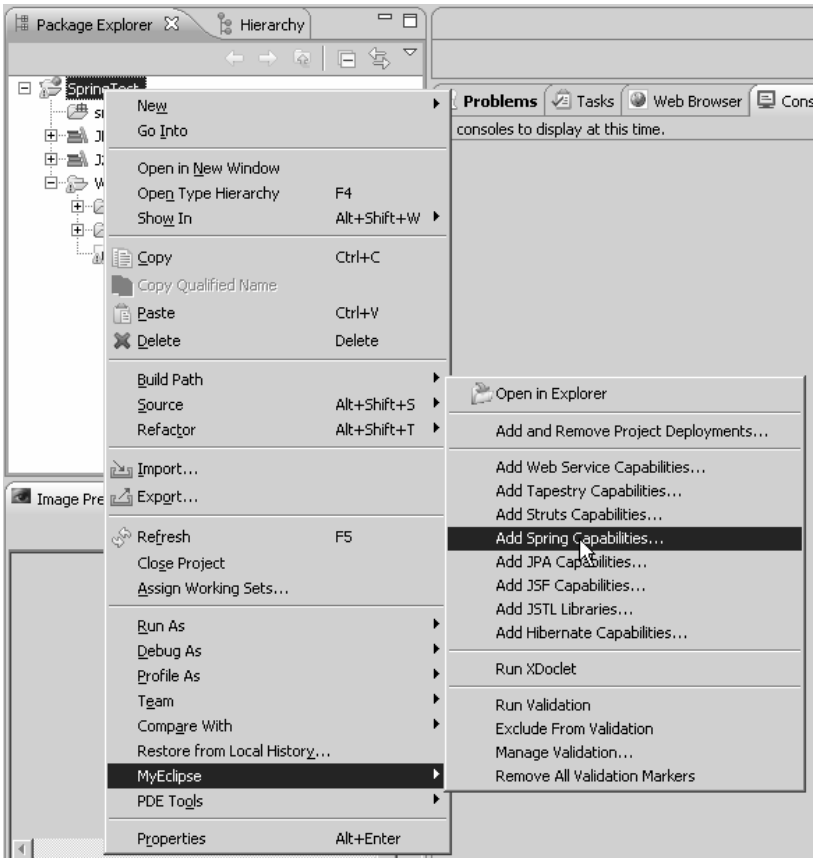


图 3-4 添加 Spring 支持

3. 弹出如图 1-5 所示的设置窗口，在窗口中依次包括以下内容：

- Spring version：选择 Spring 2；
- Select the libraries to add the buildpath：选择 MyEclipse 提供的 Spring 的库，默认只有 Spring 2.0 core libraries 被选中；
- JAR Library Installation：指定将 Spring 的 jar 文件复制到的目录，默认使用“Add checked Libraries to project build-path”；
- Tag Library Installation：指定 Spring 的 Tag 标签文件复制到的目录，默认为不采用 Spring Tag。

设置后的结果如图 9-5 所示：

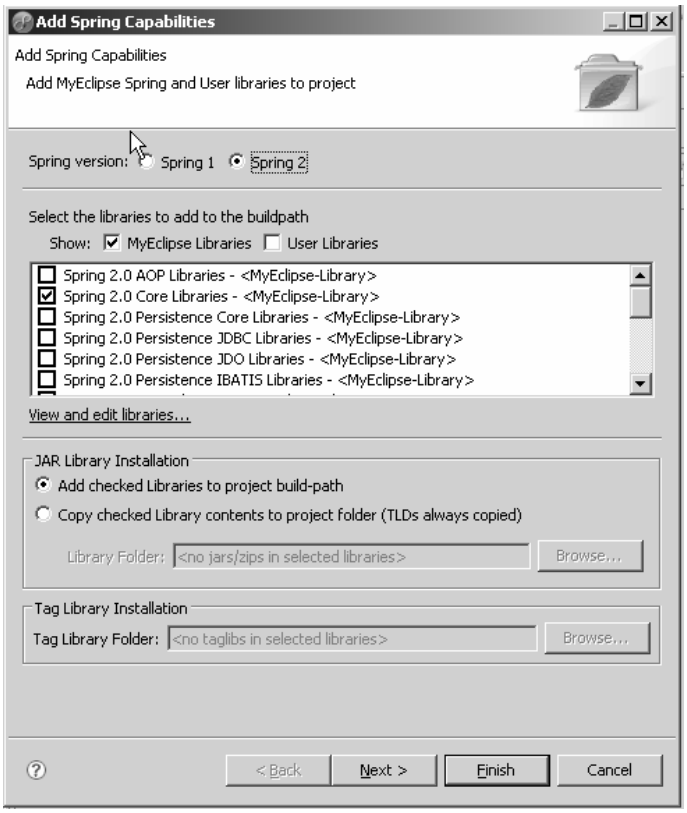


图 3-5 添加 Spring 支持

4. 单击图 9-5 中的【Next】按钮，进入如图 9-6 所示的窗口，此时选中图中的复选框，并进行如下的设置。

- Folder：选择 Bean 配置文件存放的路径，点击【Browse】按钮，选择路径，默认路径为 src；
- File：设置 Bean 配置文件的名称，默认为“applicationContext.xml”；

设置后的结果如图 9-6 所示。

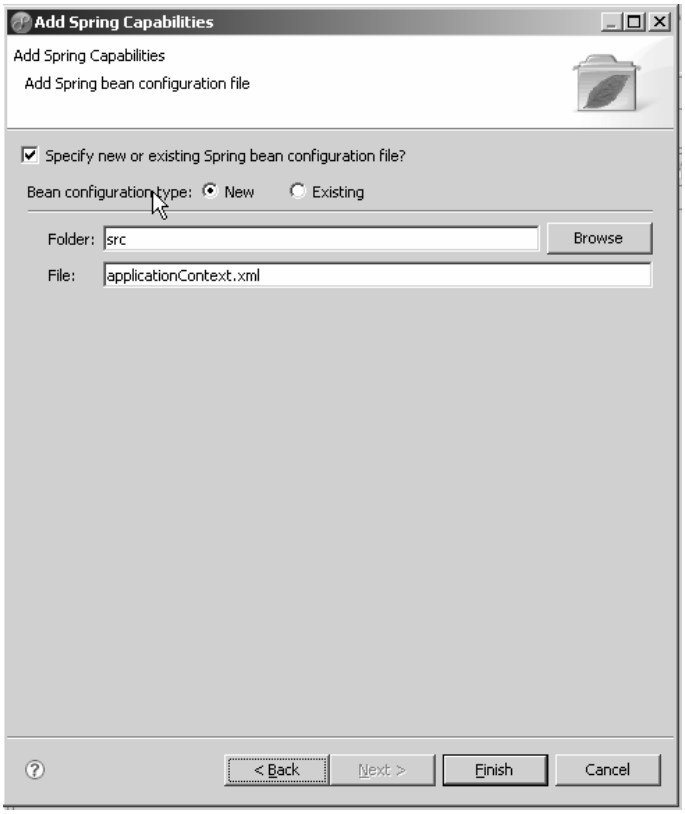


图 3-6 设置 Spring 配置文件

配置完成后，单击【Finish】按钮即可完成 Spring 支持环境的添加。

5. 现在项目“SpringTest”已经能够支持 Spring 了，也就是说已经可以添加 Spring 的相关的功能代码了。

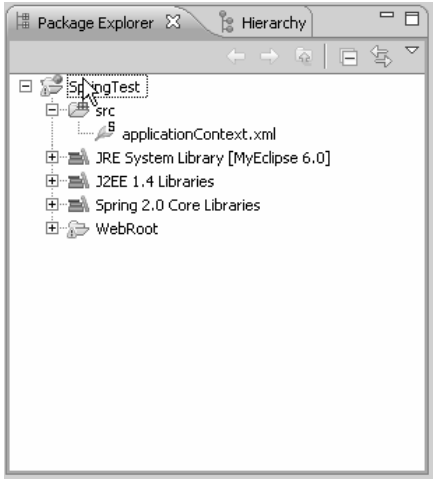


图 3-7 添加 Spring 支持

3.2 Spring 第一个“Hello Word”程序

1. 新建“HelloWorld”工程，并添加 Spring 支持。如图：

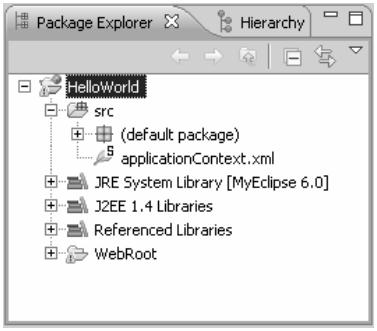


图 3-8 添加了 Spring 支持的 HelloWord 工程

2. 在跟目录下新建 HelloWord.java，代码如下：

```
public class HelloWorldBean {
    //定义 sayHello 方法,在控制台打印 HELLO WORLD
    public void sayHello()
    {
        System.out.println("HELLO WORLD");
    }
}
```

3. 在跟目录下新建 BeanTest.java，作为运行的主程序，代码如下：

```
import org.springframework.context.ApplicationContext;
import  org.springframework.context.support.
ClassPathXmlApplicationContext;
public class BeanTest {
    //程序入口
    public static void main(String args [])
    {
        //创建 Spring 的 ApplicationContext,用于测试
        ApplicationContext                                    context=new
ClassPathXmlApplicationContext("applicationContext.xml");
        //使用 Spring 上下文获得 HelloWorldBean 实例
        HelloWorldBean
bean=(HelloWorldBean)context.getBean("helloWorldBean");
        //调用 HelloWorldBean 的 sayHello 方法
        bean.sayHello();
    }
}
```

4. 编写简单的 Spring 配置文件，配置文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- spring 配置文件的根元素 -->
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 定义一个 bean，该 bean 的 id 是 helloWorldBean，class 指定该 bean 实例的实现类
-->
    <bean id="helloWorldBean" class="HelloWorldBean"></bean>
</beans>
```

5. 运行程序在控制台输出“HELLO WORLD”，运行结果：

HELLO WORLD

3.3 Spring IoC 实现简单员工管理程序

1. 新建“EmployeeManager”工程，并添加 Spring 支持。如图：

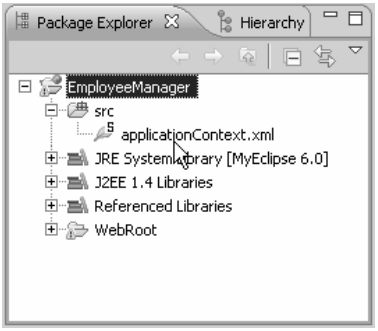


图 3-9 添加了 Spring 支持的 EmployeeManager 工程

2. 新建 Computer.java 接口，代码如下：

```
//定义 Computer 接口
public interface Computer {
    //Computer 接口里有个打字方法
    public void codeing();
}
```

3. 新建 Employee.java 接口，代码如下：

```
//定义 Employee 接口
public interface Employee
{
    //Employee 接口里定义一个使用电脑的方法
```

```
    public void userComputer();
}
```

4. 新建 Age.java 类用于提供一个方法返回整型值，代码如下：

```
public class Age {
    public int getValue()
    {
        return 23;
    }
}
```

5. 新建 NotebookPC.java 类继承 Computer 接口，代码如下

```
//Computer 的第一个实现类 NotebookPC
public class NotebookPC implements Computer {
    //默认构造函数
    public NotebookPC()
    {
    }
    //实现 Computer 接口的 codeing 方法
    public void codeing()
    {
        System.out.println("笔记本电脑打字");
    }
}
```

6. 新建 Programmer.java 类用于用于描述员工信息，代码如下：

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Programmer implements Employee {
    //姓名
    private String name;
    //年龄
    private int age;
    //下面是系列集合属性，分别表示此人的学校，成绩，健康情况
    private List schools=new ArrayList();
    private Map  score=new HashMap();
```

```
    private Properties health=new Properties();
    //面向 Computer 接口编程，而不是具体的实现类
    private Computer computer;
    //默认构造函数
    public Programmer()
    {
    }
    public Computer getComputer() {
        return computer;
    }
    //设置注入需要的 setter 方法
    public void setComputer(Computer computer) {
        this.computer = computer;
    }
    //实现 Employee 接口的 userComputer 方法
    public void userComputer()
    {
        computer.codeing();
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public List getSchools() {
        return schools;
    }
    public void setSchools(List schools) {
        this.schools = schools;
    }
    public Map getScore() {
        return score;
    }
    public void setScore(Map score) {
        this.score = score;
```

```
    }
    public Properties getHealth() {
        return health;
    }
    public void setHealth(Properties health) {
        this.health = health;
    }
}
```

7. Spring 配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 定义第一个 bean，该 bean 的 id 是 notebookPC,
    Class 指定 bean 实例的实现类
    -->
    <bean id="notebookPC" class="NotebookPC"></bean>
    <!-- 定义第二个 bean，该 bean 的 id 是 age,
    Class 指定 bean 实例的实现类
    -->
    <bean id="age" class="Age"></bean>
    <!-- 定义第三个 bean，该 bean 的 id 是 programmer,
    Class 指定 bean 实例的实现类
    -->
    <bean id="programmer" class="Programmer">
    <property name="name">
        <value>李明</value>
    </property>
    <property name="age">
            <!-- 以下是访问 bean 属性的方式 -->
            <bean
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
                <property name="targetObject"><ref local="age"/></property>
                <property
name="targetMethod"><value>getValue</value></property>
            </bean>
    </property>
  <property name="schools">
            <!-- 定义 List 属性，使用 list 元素 -->
```

```
            <list>
                <!-- list 元素里使用 value 元素确定系列值 -->
                <value>小学</value>
                <value>中学</value>
                <value>大学</value>
            </list>
        </property>
        <property name="score">
            <!-- 定义 Map 属性，使用 map 元素 -->
            <map>
                <!-- Map 属性必须是 key-value 对 -->
                <entry key="数学">
                    <value>87</value>
                </entry>
                <entry key="英语">
                    <value>89</value>
                </entry>
                <entry key="语文">
                    <value>82</value>
                </entry>
            </map>
        </property>
        <property name="health">
                <!-- 定义 Properties 属性，使用 prop 元素 -->
                <props>
                    <!-- Properties 属性必须是 key-value 对 -->
                    <prop key="血压">正常</prop>
                    <prop key="身高">175</prop>
                </props>
        </property>
        <!-- property 元素用来指定需要容器注入的属性，
        computer 属性需要容器注入，
              此处是设值注入，因此 Programmer 必须有 setComputer 方法
         -->
         <property name="computer">
            <ref local="notebookPC"/>
        </property>
    </bean>
</beans>
```

8. 编写主程序，用于测试,代码如下：

```
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class TestBean {
    public static void main(String args [])
    {
        //因为是独立的应用程序，显示地实例化 Spring 的上下文
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //利用 Spring 上下文得到 Programmer 类的实例
        Programmer programmer=(Programmer)context.getBean("programmer");
        //打印姓名
        System.out.println(programmer.getName());
        //打印年龄
        System.out.println(programmer.getAge());
        //打印学校信息
        System.out.println(programmer.getSchools());
        //打印成绩信息
        System.out.println(programmer.getScore());
        //打印健康情况
        System.out.println(programmer.getHealth());
        //使用电脑
        programmer.userComputer();
    }
}
```

9. 运行程序，运行结果如下：

```
李明
23
[小学, 中学, 大学]
{数学=87, 英语=89, 语文=82}
{血压=正常, 身高=175}
笔记本电脑打字
```

4. 相关理论知识

4.1 Spring 概念

Spring 为企业应用的开发提供一个轻量级的解决方案。该解决方案包括基于依赖注入的核心机制，基于 AOP 的申明式事务管理，与多种持久层技术的整合，以及优秀的 Web Mvc 框架等。

Spring 的设计哲学是：在尽量不影响 Java 对象的情况下，将 Java 对象加入框架的管理中。Spring 是一种低侵入式的框架。

Spring 支持对 POJO（plain old Java Object，指最传统的 Java 对象，和任何模式都无关）的管理，能将 J2EE 应用各层的对象“组装”在一起，甚至这些对象无须标准的 JavaBean。

Spring 也是高度组件化的框架。它允许开发者选用该框架的部分或全部，并不强制开发者选择 Spring 的全部。无论怎样，绝不会影响 Spring 的独特魅力。

4.2 Spring 历史

Spring 的基础架构起源于 2000 年早期，它是 Rod Johnson 在一些成功的商业项目中构建的基础设施。

在 2002 后期，Rod Johnson 发布了《Expert One-on-One J2EE Design and Development》一书，并随书提供了一个初步的开发框架实现——interface21 开发包，interface21 就是书中阐述的思想的具体实现。后来，Rod Johnson 在 interface21 开发包的基础之上，进行了进一步的改造和扩充，使其发展为一个更加开放、清晰、全面、高效的开发框架——Spring。

2003 年 2 月 Spring 框架正式成为一个开源项目，并发布于 SourceForge。

4.3 Spring 包含的模块

Spring 框架由七个定义明确的模块组成。

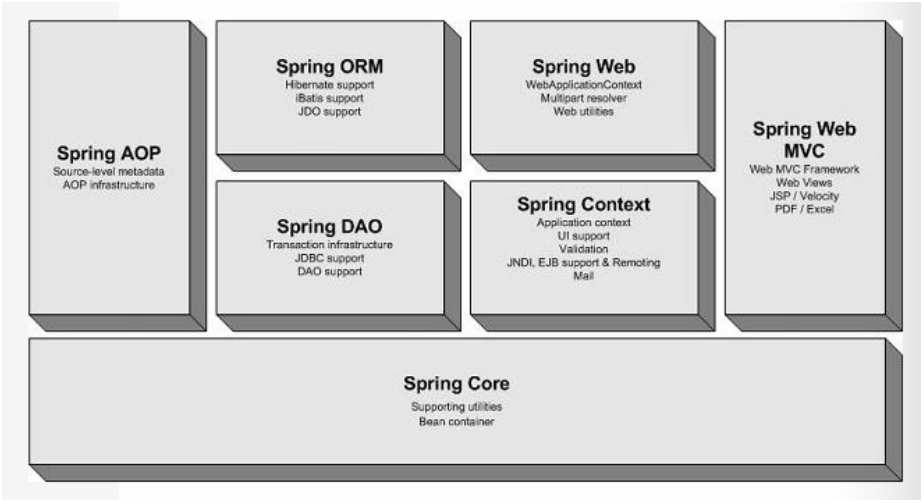


图 3-10 Spring 模块

组成 Spring 框架的每个模块（或组件）都可以单独存在，或者与其他一个或多个模块联合实现。每个模块的功能如下：

1. **核心容器：**核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转（IoC）模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

2. **Spring Context（上下文）：**Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

4. **Spring DAO：**JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理

异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。

5. **Spring ORM**：Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。

6. **Spring Web 模块**：Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Jakarta Struts 的集成。Web 模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。

7. **Spring MVC 框架**：MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

Spring 框架的功能可以用在任何 J2EE 服务器中。Spring 的核心要点是：支持不绑定到特定 J2EE 服务的可重用业务和数据访问对象。毫无疑问，这样的对象可以在不同 J2EE 环境（Web 或 EJB）、独立应用程序、测试环境之间重用。

4.4 Spring 的 Ioc 容器

IoC，Inversion of Control，控制反转又名依赖注入，其原理是基于 OO 设计原则的 The Hollywood Principle：Don't call us, we'll call you。也就是说，所有的组件都是被动的（Passive），所有的组件初始化和调用都由容器负责。组件处在一个容器当中，由容器负责管理。

Spring 以 bean 的方式组织，管理 Java 应用中各组件，组件之间的依赖关系是松耦合运行良好。Spring 使用 BeanFactory 作为应用中负责产生和管理各组件的工厂，同时也是组件运行时的容器。BeanFactory 根据配置文件确定容器中 bean 的实现，管理 bean 之间的依赖关系。

bean 之间的依赖关系由 BeanFactory 管理，bean 的具体实例化过程也由 BeanFactory 来管理。将 bean 和 bean 实现类的依赖解耦，变成对接口的依赖。程序从面向具体类的编程，转向编程面向接口编程。极大地降低应用中组件的耦合。

ApplicationContext 是 BeanFactory 的加强。ApplicationContext 接口提供在 J2EE 应用中的大量增强功能，比如随 Web 应用的自动创建，程序国际化等。

ApplicationContext 通常使用如下两个实现类：

- FileSystemXmlApplicationContext：以指定路径的 XML 配置文件创建 ApplicationContext。
- ClassPathXmlApplicationContext：以 CLASSPATH 路径下的 XML 配置文件创建 ApplicationContext。

依赖注入（Dependency Injection）和控制反转（Inversion of Control）是同一个概念，具体的含义：当某个角色（可能是一个 Java 实例，调用者）需要另一个角色（另一个 Java 实例，被调用者）的协助时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在 Spring 里，创建被调用者的工作不再由调用者来完成，因此成为控制反转，创建被调用者的实例的工作通常由 Spring 容器来完成，然后注入调用者，因此也称为依赖注入。

不管是依赖注入，还是控制反转，都说明 Spring 采用动态,灵活的方式来管理各种对象。

对象与对象之间的具体实现互相透明。依赖注入通常有两种：

- 设值注入
- 构造注入

4.4.1 设值注入

设值注入是指通过 setter 方法传入被调用者的实例。这种注入方式简单，直观，因而在 Spring 的依赖注入里大量使用。看下面的代码，这是 Employee 的接口：

```
//定义 Employee 接口
public interface Employee
{
    //Employee 接口里定义一个使用电脑的方法
    public void userComputer();
}
```

然后是 Computer 的接口：

```
//定义 Computer 接口
public interface Computer {
    //Computer 接口里有个打字方法
    public void codeing();
}
```

Employee 的实现类如下：

```
Public class Programmer implements Employee {
    //面向 Computer 接口编程，而不是具体的实现类
    private Computer computer;
    //默认构造函数
    public Programmer()
    {
    }

    public Computer getComputer() {
        return computer;
    }
    //设置注入需要的 setter 方法
    public void setComputer(Computer computer) {
        this.computer = computer;
    }
    //实现 Employee 接口的 userComputer 方法
    public void userComputer()
    {
```

```
        computer.codeing();
    }
}
```

Computer 的一个实现类 NotebookPC：

```
//Computer 的第一个实现类 NotebookPC
public class NotebookPC implements Computer {
    //默认构造函数
    public NotebookPC()
    {
    }
    //实现 Computer 接口的 codeing 方法
    public void codeing()
    {
        System.out.println("笔记本电脑打字");
    }
}
```

下面采用 Spring 的配置文件将 Employee 和 Computer 实例组织在一起。配置文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 定义第一个 bean，该 bean 的 id 是 notebookPC,
Class 指定 bean 实例的实现类
-->
    <bean id="notebookPC" class="NotebookPC"></bean>
    <!-- 定义第二个 bean，该 bean 的 id 是 programmer,
Class 指定 bean 实例的实现类
-->
    <bean id="programmer" class="Programmer">
        <!-- property 元素用来指定需要容器注入的属性，
computer 属性需要容器注入，
              此处是设值注入，因此 Programmer 必须有 setComputer 方法
         -->
        <property name="computer">
            <ref local="notebookPC"/>
        </property>
```

```
    </bean>
</beans>
```

从配置文件中，可以看到 Spring 管理 bean 的灵巧性，bean 与 bean 之间的依赖关系放在配置文件里组织，而不是写在代码里。通过配置文件的指定，Spring 能够精确地为每个 bean 注入属性。因此，配置文件里的 bean 的 class 元素，必须是真正的实现类。

Spring 会自动接管每个 bean 定义里的 property 元素定义。Spring 会在执行无参的构造器和创建默认的 bean 实例后，调用对应的 setter 方法为程序注入属性值。Property 定义的属性值将不再由该 bean 主动创建，管理，而改为被动接收 Spring 的注入。

每个 bean 的 id 属性是该 bean 的唯一标识。程序通过 id 属性访问 bean。bean 与 bean 的依赖关系也是通过 id 属性完成。

下面看主程序部分:

```
Import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
ClassPathXmlApplicationContext;

public class BeanTest {
    //主方法，程序入口
    public static void main(String args [])
    {
        //因为是独立的应用程序，显示地实例化 Spring 的上下文
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //Employee bean 的 id 来获取 bean 的实例，面向接口编程，因此
        //此处强制类型转换为接口类型
        Employee em=(Employee)context.getBean("programmer");
        //直接执行 Employee 的 userComputer()方法
        em.userComputer();
    }
}
```

程序执行结果:

笔记本电脑打字

主程序调用 Employee 的 userComputer()方法时，该方法的方法体内需要使用 Computer 的实例，但程序里没有任何地方将特定的 Employee 实例和 Computer 实例耦合在一起。或者说，程序里没有为 Employee 实例传入 Computer 的实例，Computer 实例由 Spring 在运行期间动态注入。

Employee 实例不仅不需要了解 Computer 实例的具体实现，甚至无须了解 Computer 的创建过程。程序在运行到需要 Computer 时，Spring 创建了 Computer 实例，然后注入给需要的 Computer 的调用者。Employee 实例运行到需要 Computer 实例的地方，自然就产生了 Computer 实例，用来供 Employee 实例使用。

4.4.2 构造注入

所谓构造注入，指通过构造函数来完成依赖关系的，而不是通过 setter 方法。对前面代码 Programmer 类作简单的修改，修改后的代码如下：

```
Public class Programmer implements Employee {
    //面向 Computer 接口编程，而不是具体的实现类
    private Computer computer;
    //默认构造函数
    public Programmer(Computer computer)
    {
        this.computer=computer;
    }
    //实现 Employee 接口的 userComputer 方法
    public void userComputer()
    {
        computer.codeing();
    }
}
```

此时无须 Programmer 类里的 setComputer 方法，构造 Programmer 实例的时候，Spring 为 Programmer 实例注入所依赖的 computer 实例。构造注入的配置文件也需做简单的修改，修改后的配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 定义第一个 bean，该 bean 的 id 是 notebookPC,Class 指定 bean 实例的实现类 -->
    <bean id="notebookPC" class="NotebookPC"></bean>
    <!-- 定义第二个 bean，该 bean 的 id 是 programmer,Class 指定 bean 实例的实现类 -->
    <bean id="programmer" class="Programmer">
        <constructor-arg>
        <ref bean="notebookPC"/>
        </constructor-arg>
    </bean>
</beans>
```

执行效果与使用设值注入时的执行效果完全一样，区别在于：创建 Programmer 实例中 computer 属性的时机不同，设值注入是先创建一个默认的实例，然后调用对应的 setter 方法注入依赖关系。而构造注入则在创建 bean 实例时，已经完成依赖关系的注入。

设值注入和构造注入，都是 Spring 支持的依赖注入模式。也是目前主流的依赖注入模式。

两种注入各有优点。建议采用以设值注入为主，构造注入为辅的注入策略。对于依赖关系无须变化的注入，尽量采用构造注入；而其他的依赖关系的注入，则考虑采用设置注入。

4.4.3 bean 的基本行为

bean 在 Spring 的容器中有两种基本行为：

- singleton：单态
- non-singleton 或 prototype：原形

如果一个 bean 被设置成 singleton 的，整个 Spring 容器里只有一个共享实例存在，程序每次请求该 id 的 bean，Spring 都会返回该 bean 的共享实例。

如果一个 bean 被设置成 non-singleton 行为，程序每次请求该 id 的 bean，Spring 都会创建一个 bean 实例，然后返回给程序。通常，要求 Web 应用程序的控制器 bean 被配置成 non-singleton 行为，因为每次 HttpServletRequest 都需要系统启动一个新 Action 来处理用户请求。

如果不指定 bean 的行为，Spring 默认使用 singleton 行为。Java 在创建 Java 实例时，需要进行内存申请；销毁实例时，需要完成垃圾回收。这些工作都会导致系统开销的增加。因此，non-single 行为的 bean 创建、销毁时代价较大。而 singleton 行为的 bean 在实例后，可以重复使用。因此，除非必要，尽量避免将 bean 设置成 non-singleton 行为。

设置 bean 的基本行为，通过 scope 属性来指定，scope 属性值只接受 singleton 或 prototype。下面配置 singleton 和 non-singleton bean 各一个。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 定义第一个 bean，该 bean 的 id 是 programmer,Class 指定 bean 实例的实现类 -->
    <bean id="programmer" class="Programmer"></bean>
<!-- 定义第二个bean,该bean的id是anthorprogrammer,Class指定bean实例的实现类 -->
    <bean id="anthorprogrammer" class="Programmer" scope=″ prototype″>
    </bean>
</beans>
```

主程序通过如下代码来测试两个 bean 的区别：

```
Import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
ClassPathXmlApplicationContext;

public class BeanTest {
    //主方法，程序入口
    public static void main(String args [])
```

```
    {
        //因为是独立的应用程序，显示地实例化 Spring 的上下文
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //判断两次请求 singleton 行为的 bean 实例是否相等
    System.out.println(context.getBean(“programmer”)
==context.getBean(“programmer”));
        //判断两次请求 non-singleton 行为的 bean 实例是否相等
    System.out.println(context.getBean(“anthorprogrammer”)
==context.getBean(“anthorprogrammer”));
    }
}
```

程序运行结果：

true

flase

4.4.4 bean 依赖关系可接受的元素

- value
- ref
- bean
- list，set，map 以及 props

4.3.4.1 value 元素

value 元素用于确定字符串参数。这些参数由 PropertyEditors 完成转换：从 java.lang.String 类型转化为所需的参数类型。基本数据类型，通常可以正确转换。下面代码演示 value 元素正确属性值的情况：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<
    <bean id="exampleBean" class="ExampleBean">
       <property name=”integerProperty”>
              <value>1</value>
</property>
<property name=”doubleProperty”>
              <value>2.3</value>
```

```
</property>
</bean>
</beans>
```

下面是该 bean 的实现类，接口不再列出：

```
Public class ExampleBean {
    private int integerProperty;
   private double doubleProperty;
    //默认构造函数
    public void setIntegerProperty(int i)
{
     this.integerProperty=i;
}
//默认构造函数
    public void setDoubleProperty (double d)
{
     this.doubleProperty =d;
}
}
```

测试本程序与之前的程序差别不大，在此不再赘述。执行结果是：

1

2．3

value 元素主要传入字符串参数，基本数据类型参数。也可以传入合作者 bean，但不推荐使用 value 元素。可以指定为空，指定空属性值通过<null/>元素，代码如下：

```
<bean id="exampleBean" class="ExampleBean">
 <property name=”email”>
         <value></value>
</property>
</bean>
```

它采用如下配置文件的效果相同:

```
<bean id="exampleBean" class="ExampleBean">
       <property name=”email”>
              <value><null/></value>
</property>
</bean>
```

4.4.4.2 ref 元素

ref 元素：用于指定属性值的 Spring 容器中的其他 bean。ref 元素是一种简写，并可防止

出现引用错误，用于设置属性值为容器中其他 bean。通常，对属性值为容器中其他 bean 的情况，推荐采用 ref 元素指定，而不是 value 元素。看下面的配置片段：

```
<bean id="notebookPC" class="NotebookPC"></bean>
<bean id="programmer" class="Programmer">
       <property name=”computer”>
                 <!—应用容器中另一个 bean-->
                 <ref local=”notebookPC”/>
       </property>
</bean>
```

于下面的配置片段效果完全一样：

```
<bean id="notebookPC" class="NotebookPC"></bean>
<bean id="programmer" class="Programmer">
       <!—应用容器中另一个 bean-->
       <value=”notebookPC”/>
</bean>
```

第一种形式比第二种形式更好的原因：使用 ref 标记，可以让 Spring 容器在部署时验证依赖的 bean 是否真正存在；在第二种形式中，notebookPC 属性值仅在创建 bean 时作验证，会导致错误的延迟出现。而且，第二种还有额外的类型转换开销。因此，合作者 bean 属性值的传入，推荐采用 ref 元素指定。ref 元素通常有两个属性：

- bean
- local

bean 用于确定不在同一个 XML 配置文件中的 bean。而 local 用于确定同一个 XML 配置文件中的其他 bean，并且 local 属性值只能是其他 bean 的 id 属性。

local 属性让 Spring 在解析 XML 时，验证 bean 的名称。

4.3.4.3 bean 元素

bean 元素用来定义 bean 实例属性是个嵌套 bean，而不是在 Spring 容器中已经存在的 bean。嵌套 bean 只对它的外部 bean 有效，因此嵌套 bean 没有 id 属性。修改上面的配置文件，使之变成嵌套 bean 的形式，示例如下：

```
<bean id="programmer" class="Programmer">
       <!—-属性为嵌套 bean,嵌套 bean,不能由 Spring 容器直接访问,因此嵌套 bean 没有 id
属性-->
       <bean class=” NotebookPC”/>
</bean>
```

嵌套 bean 的配置形式，保证嵌套 bean 不能被容器访问，因此不用担心其他程序修改嵌套 bean。外部 bean 的用法与之前的用法完全一样，使用结果也没有区别。

嵌套 bean 损失了部分性，提高了程序的内聚性。

4.3.4.4 list，set，map 以及 props 元素

list，set，map 和 props 元素分别用来设置类型为 List，Set，Map 和 Properties 的属性值。分别用来为 bean 传入集合值。看如下示例代码：

```
import java.util.*;
public class Programmer implements Employee {
    //下面是系列集合属性，分别表示此人的学校，成绩，健康和电脑
    private List schools=new ArrayList();
    private Map  score=new HashMap();
    private Properties health=new Properties();
    private Set       Computer=new HashSet();

    public Programmer()
    {
        System.out.println("Spring 实例化主调 bean：Programmer 实例...");
    }

    public List getSchools() {
        return schools;
    }

    public void setSchools(List schools) {
        this.schools = schools;
    }

    public Map getScore() {
        return score;
    }

    public void setScore(Map score) {
        this.score = score;
    }

    public Properties getHealth() {
        return health;
    }

    public void setHealth(Properties health) {
        this.health = health;
    }

    public Set getComputer() {
```

```
        return computer;
    }

    public void setComputer(Set computer) {
        this.computer = computer;
    }

    public void test()
    {
        System.out.println(schools);
        System.out.println(score);
        System.out.println(health);
        System.out.println(computer);
    }
}
```

下面是 Spring 的配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 定义第一个 bean，该 bean 的 id 是 notebookPC,Class 指定 bean 实例的实现类 -->
<bean id="notebookPC" class="NotebookPC"></bean>
    <!-- 定义第二个 bean，该 bean 的 id 是 programmer,Class 指定 bean 实例的实现类 -->
    <bean id="programmer" class="Programmer">
        <property name="schools">
            <!-- 定义 List 属性，使用 list 元素 -->
            <list>
                <!-- list 元素里使用 value 元素确定系列值 -->
                <value>小学</value>
                <value>中学</value>
                <value>大学</value>
            </list>
        </property>
        <property name="score">
            <!-- 定义 Map 属性，使用 map 元素 -->
            <map>
                <!-- Map 属性必须是 key-value 对 -->
                <entry key="数学">
                    <value>87</value>
```

```
                </entry>
                <entry key="英语">
                    <value>89</value>
                </entry>
                <entry key="语文">
                    <value>82</value>
                </entry>
            </map>
        </property>
        <property name="health">
                <!-- 定义 Properties 属性，使用 prop 元素 -->
                <props>
                    <!-- Properties 属性必须是 key-value 对 -->
                    <prop key="血压">正常</prop>
                    <prop key="身高">175</prop>
                </props>
        </property>
        <property name="computer">
            <!-- 定义 Set 属性，使用 value,bean,ref 等指定系列值 -->
            <set>
                <value>字符串电脑</value>
                <!-- bean 用于指定嵌套 bean 作为属性值 -->
                <bean class="NotebookPC"></bean>
                <!-- 确定容器中另一个 bean 为属性值 -->
                <ref local="notebookPC"/>
            </set>
        </property>
    </bean>
</beans>
```

如上配置文件所示：set 元素的值可以通过 value，bean，ref 确定值。map 元素 entry 的值、set 元素的值都可以使用如下元素：

- value：确定基本数据类型值，或字符串类型值。
- local：确定另一个 bean 为属性值。
- bean：确定一个嵌套 bean 为属性值。
- list，set，map 以及 props：确定集合值为属性值。

4.4.5 高级依赖注入

前面介绍的依赖注入，要么是 JavaBean 式的值，要么是直接依赖于其他 bean。实际应用当中，某个实例的属性可能是某个方法的返回值、类的 field 值，或是属性值。这种常见的注入方式，Spring 同样提供支持。Spring 甚至支持将 bean 实例的属性值、方法返回值、field 值，

直接赋给一个变量。

4.4.5.1 属性值的依赖注入

属性值的注入，是通过 PropertyPathFactoryBean 类完成的。PropertyPathFactoryBean 用来获得目标 bean 的属性值。获得的值可注入其他 bean，也可以直接定义成新的 bean。看如下配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <!-- 定义第一个 bean，该 bean 的 id 是 programmer,Class 指定 bean 实例的实现类 -->
    <bean id="programmer" class="Programmer">
        <property name="age">
            <value>12</value>
        </property>
    </bean>
    <bean id="teacher" class="Teacher">
        <property name="age">
            <!-- 以下是访问 bean 属性的方式 -->
            <bean id="programmer.age"
  class="org.springframework.beans.factory.config.PropertyPathFactoryBean">
            </bean>
        </property>
    </bean>
</beans>
```

主程序程序如下：

```
Public static void main(String args [])
    {
        //因为是独立的应用程序，显示地实例化 Spring 的上下文
        ApplicationContext context=
               new ClassPathXmlApplicationContext("applicationContext.xml");
        //打印出 teacher 的 age 值
        System.out.println(((Teacher)context.getBean("teacher"))
.getAge());
}
```

执行结果如下：

12

teacher 这个 bean 的 age 属性来自于 programmer bean 的 age 属性。一个 bean 实例的属性，可以注入另一个 bean。bean 实例的属性值可以直接定义成 bean 实例，也可以通过 PropertyPathFactoryBean 完成的。对上面的配置文件增加如下代码：

```
<bean id="age"
  class="org.springframework.beans.factory.config.PropertyPathFactoryBean">
        <!-- 确定目标 bean -->
        <property name="targetBeanName">
            <value>programmer</value>
        <!-- 确定属性名 -->
        </property>
        <property name="propertyPath">
            <value>age</value>
        </property>
</bean>
```

主程序部分增加如下输出：

```
System.out.println(context.getBean("age"));
```

此语句的执行效果如下：

```
12
```

使用 PropertyPathFactoryBean 必须指定如下两个属性：

- tragetBeanName：用于指定目标 bean，确定获取哪个 bean 的属性值。
- propertyPath：用于指定属性，确定获取目标 bean 的哪个属性值。

4.4.5.2 field 值的依赖注入

field 值的注入，是通过 FieldRetrievingFactoryBean 类完成的。FieldRetrievingFactoryBean 用来获得目标 bean 的 field 值。获得的值可注入其他 bean，也可以直接定义成新的 bean。看如下配置文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="teacher" class="Teacher">
        <property name="age">
            <!-- 以下是访问 bean 属性的方式 -->
            <bean                                                id="Age.age"
class="org.springframework.beans.factory.config.FieldRetrievingFactoryBean"
></bean>
```

```
        </property>
    </bean>
</beans>
```

Age 类代码如下：

```
Public class Age {
    public static final int age=23;
}
```

主程序如下：

```
Public class BeanTest {
    //主方法，程序入口
    public static void main(String args [])
    {
        //因为是独立的应用程序，显示的实例化 Spring 的上下文
        ApplicationContext context=
               new ClassPathXmlApplicationContext("applicationContext.xml");
        //打印出 teacher 的 age 值
    System.out.println(((Teacher)context.getBean("teacher")).getAge());
    }
}
```

程序的执行结果如下：

23

field 值也可以配置成 bean 实例。在配置文件中增加如下一段：

```
<!-- 将 field 值定义成 bean 实例 -->
    <bean id="age" class=
     "org.springframework.beans.factory.config.FieldRetrievingFactoryBean">
        <!-- 确定该 bean 的值来自于 staticField -->
        <property name="staticField">
            <value>Age.age</value>
        </property>
</bean>
```

主程序部分增加如下输出：

System.out.println(context.getBean("age"));

此语句的执行效果如下：

23

4.4.5.3 方法返回值的依赖注入

方法返回值的注入是通过 MethodInvokingFacotyBean 类完成的。MethodInvokingFacoty-

Bean 用来获得某个方法的返回值，该方法即可以是静态方法，也可以是实例方法。方法的返回值可以注入 bean 实例属性，也可以直接定义成 bean 实例。看如下配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 以下定义目标 bean，后面的 bean 将使用该 bean 的方法返回值注入属性 -->
    <bean id="age" class="Age"></bean>
    <!-- 该 bean 的 age 属性不是直接注入的，而是依赖于其他 bean 的返回值 -->
    <bean id="teacher" class="Teacher">
        <property name="age">
            <!-- 以下是访问 bean 属性的方式 -->
            <bean
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
                <property name="targetObject"><ref local="age"/></property>
                <property
name="targetMethod"><value>getValue</value></property>
            </bean>
        </property>
    </bean>
</beans>
```

下面给出 axe 的代码部分。axe 包含一个实例方法，返回整型值，代码如下:

```
Public class Age {
    public int getValue()
    {
        return 23;
    }
   public static int getStaticValue()
    {
        return 15;
    }
}
```

主程序如下：

```
Public class BeanTest {
    //主方法，程序入口
    public static void main(String args [])
    {
```

```
        //因为是独立的应用程序，显示的实例化 Spring 的上下文
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //打印出 teacher 的 age 值
    System.out.println(((Teacher)context.getBean("teacher")).getAge());
    }
}
```

执行结果如下：

23

bean teacher 的 age 属性值来自于 age 类的实例方法返回值。使用 bean 实例的方法返回值注入，通过 MethodInvokingFactoryBean 完成，必须指定如下两个属性。

- targetObject：确定目标 bean，该 bean 可以是容器中已有的 bean，也可以是嵌套 bean。
- targetMethod：确定目标方法，确定通过目标 bean 的那个方法返回值注入。

如果使用静态方法返回值注入，则无须指定 targetObject，但需要指定目标的 class,指定目标 class 的属性通过通过 targetClass 属性。使用静态方法注入，需指定如下两个属性。

- targetClass：确定目标 class
- targetMethod：确定目标方法，确定通过目标 bean 的那个方法返回值注入。

在配置文件中增加如下一段：

```
<!-- 该 bean 的 age 属性不是直接注入的，而是依赖于其他 bean 的返回值 -->
    <bean id="anotherteacher" class="Teacher">
        <property name="age">
            <!-- 以下是访问 bean 属性的方式 -->
            <bean
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
                <!-- targetClass 确定目标类，确定调用哪个类 -->
                <property name="targetClass"><value>Age</value></property>
                <!-- targetMethod 确定目标方法，该方法必须是静态方法 -->
                <property
name="targetMethod"><value>getStaticValue</value></property>
            </bean>
        </property>
    </bean>
```

主程序中增加如下输出：

```
//打印出 anotherteacher 的 age 值
System.out.println(((Teacher)context.getBean("anotherteacher")).getAge());
```

程序执行结果如下：

15

当然，也可以将方法返回值直接定义成 bean，配置文件中增加如下一段：

```
<!-- 将实例方法返回值直接定义成 bean -->
    <bean id="agevalue"
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
        <!-- targetObject 确定目标 bean，确定调用那个 bean -->
        <property name="targetObject"><ref local="age" /></property>
        <!-- targetMethod 确定目标方法，确定调用目标 bean 的哪个方法 -->
        <property name="targetMethod"><value>getValue</value></property>
    </bean>
```

主程序部分增加如下输出：

System.out.println(context.getBean("agevalue"));

此语句的执行效果如下：

23

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 HelloWorld 工程，添加 Spring 的支持。（10 分钟）

2. 编写 HelloWorldBean，BeanTest 和 applicationContext.xml 配置文件并运行程序。 （20 分钟）

3. 建立 EmployeeManager 工程，添加 Spring 的支持。（10 分钟）

4. 编写 Computer 接口和 NotebookPC 类。（10 分钟）

5. 编写 Employee 接口、Age 和 Programmer 类。（15 分钟）

6. 编写 TestBean 类和 applicationContext.xml 配置文件并运行程序。（25 分钟）

6. 作业

利用 Spring IoC 实现员工管理程序。

案例二 Spring AOP 基础

1. 教学目标

1.1 AOP 编程

2. 工作任务

使用 Spring AOP 打印程序日志。

3. 相关实践知识

使用 Spring AOP 打印程序日志

本示例我们操作 mybbs 数据库中的 userinfo 表，模拟增删改论坛用户，并在控制台打印出操作日志。

1. 首先建立一个 Web 项目 UserInfoManager，添加 Spring 支持。

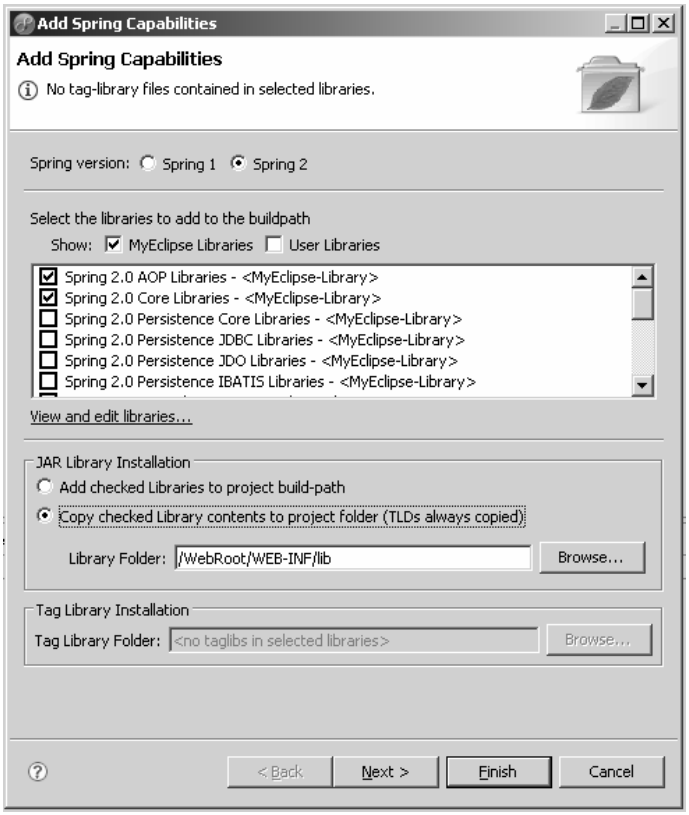


图 3-11 建立 Spring 工程

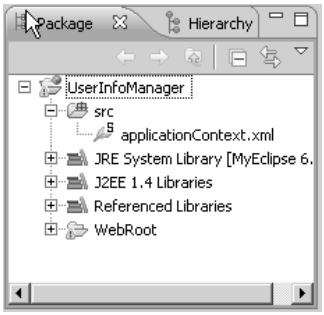


图 3-12 添加完 Spring 支持的 UserInfoManager 工程

2. 新建 com.pojo 包，并在包下建立 Userinfo.java 类和 Userinfo.hbm.xml 配置文件。

```
package com.pojo;
public class Userinfo implements java.io.Serializable {
    //用户 id
    private Integer userid;
    //用户名
    private String username;
    //密码
    private String userpwd;
    //密码找回问题
    private String userques;
    //密码找回答案
    private String userans;
    //积分
    private Integer integral;
    //等级
    private String grade;
    //email
    private String useremail;
    //性别
    private Byte sex;

    public Userinfo() {
    }

    public Userinfo(Integer userid, String username, String userpwd,
            String userques, String userans, Integer integral, String grade,
            String useremail, Byte sex) {
        this.userid = userid;
        this.username = username;
        this.userpwd = userpwd;
```

```
        this.userques = userques;
        this.userans = userans;
        this.integral = integral;
        this.grade = grade;
        this.useremail = useremail;
        this.sex = sex;
    }

    public Integer getUserid() {
        return this.userid;
    }

    public void setUserid(Integer userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUserpwd() {
        return this.userpwd;
    }

    public void setUserpwd(String userpwd) {
        this.userpwd = userpwd;
    }

    public String getUserques() {
        return this.userques;
    }

    public void setUserques(String userques) {
        this.userques = userques;
    }

    public String getUserans() {
        return this.userans;
```

```
    }

    public void setUserans(String userans) {
        this.userans = userans;
    }

    public Integer getIntegral() {
        return this.integral;
    }

    public void setIntegral(Integer integral) {
        this.integral = integral;
    }

    public String getGrade() {
        return this.grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public String getUseremail() {
        return this.useremail;
    }

    public void setUseremail(String useremail) {
        this.useremail = useremail;
    }

    public Byte getSex() {
        return this.sex;
    }

    public void setSex(Byte sex) {
        this.sex = sex;
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="com.pojo.Userinfo" table="userinfo" catalog="mybbs">
      <id name="userid" type="java.lang.Integer">
         <column name="userid" />
         <generator class="assigned" />
      </id>
      <property name="username" type="java.lang.String">
         <column name="username" length="20" not-null="true" />
      </property>
      <property name="userpwd" type="java.lang.String">
         <column name="userpwd" length="20" not-null="true" />
      </property>
      <property name="userques" type="java.lang.String">
         <column name="userques" length="50" not-null="true" />
      </property>
      <property name="userans" type="java.lang.String">
         <column name="userans" length="50" not-null="true" />
      </property>
      <property name="integral" type="java.lang.Integer">
         <column name="integral" not-null="true" />
      </property>
      <property name="grade" type="java.lang.String">
         <column name="grade" length="20" not-null="true" />
      </property>
      <property name="useremail" type="java.lang.String">
         <column name="useremail" length="20" not-null="true" />
      </property>
      <property name="sex" type="java.lang.Byte">
         <column name="sex" />
      </property>
   </class>
</hibernate-mapping>
```

3. 新建 com.dao 包，并在包下建立 UserinfoDao.java 用以操作数据库。

```
package com.dao;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import com.pojo.Userinfo;
import org.hibernate.Query;
```

```
public class UserInfoDao {
    //定义 session 用以操作数据库
    private Session session;
    public UserInfoDao()
    {
        //打开一个 session 连接
        Configuration config=new Configuration().configure();
        SessionFactory factory=config.buildSessionFactory();
        session=factory.openSession();
    }
    //添加一个用户
    public void add(Userinfo userinfo)
    {
        try
        {
            Transaction tra=session.beginTransaction();
            session.save(userinfo);
            tra.commit();
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    //修改一个用户
    public void update(Userinfo userinfo)
    {
        try
        {
            Transaction tra=session.beginTransaction();
            session.update(userinfo);
            tra.commit();
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    //查询一个用户
    public Userinfo select(int id)
    {
        Userinfo user=null;
```

```
        try
        {
            Transaction tra=session.beginTransaction();
            user=(Userinfo)session.load(Userinfo.class, id);
            tra.commit();
        }catch(Exception e)
        {
            e.printStackTrace();
        }
        return user;
    }

    //删除一个用户
    public void delete(int id)
    {
        try
        {
            Transaction tra=session.beginTransaction();
            Query query=session.createQuery("delete Userinfo as u where
u.userid=?");
            query.setInteger(0, id);
            query.executeUpdate();
            tra.commit();
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

4. 新建 com.serivce 包，在包下建立 UserServiceInterfaces.java 接口和 UserService.java 类封装业务逻辑。

```
package com.service;
import com.pojo.Userinfo;
public interface UserServiceInterface {
    //添加用户
    public void addUser(Userinfo user);
    //修改用户
    public void updateUser(Userinfo user);
    //查询用户
    public Userinfo getUser(int id);
    //删除用户
```

```
    public void deleteUser(int id);
}
```

```
package com.service;
import com.dao.UserInfoDao;
import com.pojo.Userinfo;
public class UserService implements UserServiceInterface{
    private UserInfoDao userInfoDao=null;
    public UserService()
    {
        userInfoDao=new UserInfoDao();
    }
    //添加用户
    public void addUser(Userinfo user)
    {
        userInfoDao.add(user);
    }
    //修改用户
    public void updateUser(Userinfo user)
    {
        userInfoDao.update(user);
    }
    //查询用户
    public Userinfo getUser(int id)
    {
        Userinfo user=userInfoDao.select(id);
        return user;
    }
    //删除用户
    public void deleteUser(int id)
    {
        userInfoDao.delete(id);
    }
}
```

5. 根目录下建立 MyBeforeAdvisor.java 类，用于打印日志。

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;
//before 处理实现类
public class MyBeforeAdvisor implements MethodBeforeAdvice {

    //实现 MethodBeforeAdvice 接口必须实现的方法，该方法在连接点之前执行
```

```
    public void before(Method arg0, Object[] arg1, Object arg2)
            throws Throwable {
        if(arg0.getName().equals("addUser"))
        {
            System.out.println("添加了一个用户");
        }
        else if(arg0.getName().equals("updateUser"))
        {
            System.out.println("更新了一个用户");
        }
        else if(arg0.getName().equals("getUser"))
        {
            System.out.println("查询了一个用户");
        }
        else if(arg0.getName().equals("deleteUser"))
        {
            System.out.println("删除了一个用户");
        }
    }
}
```

6. 编写 Spring 的配置文件 applicationContext.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="userService" class="com.service.UserService"></bean>
    <!-- 配置 before 处理 bean -->
    <bean id="myBeforeAdvisor" class="MyBeforeAdvisor"></bean>
    <!-- 配置工厂 bean -->
    <bean id="user"
        class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces">
            <value>com.service.UserServiceInterface</value>
        </property>
        <property name="target">
            <ref local="userService" />
        </property>
        <property name="interceptorNames">
            <list>
```

```
                <value>myBeforeAdvisor</value>
            </list>
        </property>
    </bean>
</beans>
```

7. 编写 Hibernate 的配置文件 hibernate.cfg.xml。

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                 -->
<hibernate-configuration>
    <session-factory>
        <property name="connection.username">root</property>
        <property name="connection.url">
            jdbc:mysql://localhost:3306/mybbs
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="myeclipse.connection.profile">MySQL</property>
        <property name="connection.password">sa</property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <mapping resource="com/pojo/Userinfo.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

8. 编写测试程序。

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.pojo.Userinfo;
import com.service.UserServiceInterface;
public class TestBean {
    public static void main(String args [])
    {
        ApplicationContext                                      context=new
```

```
ClassPathXmlApplicationContext("applicationContext.xml");
        //拿到代理对象
        UserServiceInterface
userService=(UserServiceInterface)context.getBean("user");
        //生成一则用户信息
        Userinfo user=new Userinfo();
        user.setUserid(11);
        user.setUsername("scott");
        user.setUserpwd("soctt");
        user.setUserques("how old are you");
        user.setUserans("32");
        user.setUseremail("hao123@.com");
        user.setIntegral(50);
        user.setGrade("初级用户");
        user.setSex((byte)1);
        //添加一个用户
        userService.addUser(user);
        //查询用户
        user=userService.getUser(11);
        //修改用户
        user.setUsername("anthorscott");
        userService.updateUser(user);
        //删除用户
        userService.deleteUser(11);
    }
}
```

9. 运行测试程序会在控制台打印：

添加了一个用户

查询了一个用户

更新了一个用户

删除了一个用户

4. 相关理论知识

4.1 Spring AOP 简介

AOP 是 OOP 的延续，是 Aspect Oriented Programming 的缩写，意思是面向切面编程。AOP 实际是 GoF 设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP 可以说也是这种目标的一种实现。

面向切面编程并不会取代面向对象编程，而是作为面向对象编程的补充。AOP 从动态的

角度考虑程序的结构，从而使 OOP 更加完善。面向对象编程将程序分解成各个层次的对象，而面向切面编程将程序运行过程分解成各个切面。

AOP 从程序动态运行角度考虑程序的结构，提取业务处理过程的切面。AOP 面对程序运行中各个步骤，以降低各步骤之间的耦合，从而提高步骤之间的隔离。

OOP 是静态的抽象，它对应用中的实体及属性、行为进行抽象，从而获得清晰高效的单元划分；而 AOP 是动态的抽象，它对应用的执行过程的步骤进行抽象，从而获得步骤之间的逻辑划分。AOP 框架并不与特定的代码耦合，能处理程序执行中的特定点，而不是某个具体的程序。AOP 框架具有以下两个特征：

- 各步骤之间的良好隔离。
- 源代码无关性。

AOP 框架是 Spring 的一个关键组件。AOP 框架是 Spring IoC 的完善和补充，使之成为更加有效的中间件。当然，IoC 容器（BeanFactory 和 ApplicationContext）并不依赖于 AOP 框架，因此，AOP 框架并不是必须的。

Spring AOP 的目标是提供与 Spring IoC 容器紧密结合的 AOP 框架，Spring AOP 并不致力于提供完善的 AOP 实现，而是以“实用主义”为目标，提供 AOP 框架与 IoC 容器的完美结合。

4.2 AOP 中的概念

下面是关于 AOP 一些术语解释。

1. 切面（Aspect）：一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是 J2EE 应用中一个关于横切关注点的很好的例子。

2. 连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用，或者异常的抛出。Spring AOP 中连节点总是方法的调用，Spring 并没有显示地使用连接点。

3. 处理（Porcess）：AOP 框架在特定的的连接点执行的动作。处理包括“around”、“before”、“throws”等类型。大部分框架都以拦截器作为处理模型。

4. 切入点（Pointcut）：系列连接点的集合，它确定处理触发的时机。AOP 框架允许开发者自己定义切入点：例如使用正则表达式。

5. 引入（Introduction）：添加方法或字段到被处理的类。Spring 允许引入新的接口到任何被处理的对象。例如，可以使用一个引入，使任何对象实现 IsModified 接口，以此来简化缓存的应用。

6. 目标对象（Target Object）：包含连接点的对象，也称为被处理对象或者被代理对象。

7. AOP 代理（AOP Proxy）：AOP 框架创建的对象，包含处理。Spring 中的 AOP 代理可以是 JDK 动态代理，也可以是 CGLIB 代理。前者为实现接口的目标对象代理，后者为不实现接口的目标对象代理。

4.3 AOP 代理

AOP 代理是 AOP 框架创建的对象。通常，AOP 代理可以作为目标对象的替代品，而 AOP 代理提供比目标对象更加强大的功能。一个典型的例子就是 Spring 中的事务代理 bean。通常，目标 bean 的方法不是事务性的，而 AOP 代理包含目标 bean 的全部方法，而且这些方

法经过加强，变成了事务性方法。简单的说，目标对象是蓝本，AOP 代理是目标对象的加强，在目标对象的基础上，增加属性和方法，提供更强大的功能。

目标对象包含系列的切入点。切入点是可以触发处理连接点的集合。用户可以自定义切入点，比如使用正则表达式。AOP 代理包装目标对象，在切入点加入的处理，使目标对象的方法功能更强。

Spring 默认使用 JDK 动态代理实现 AOP 代理，主要用于代理接口，也可以使用 CGLIB 代理实现。实现类的代理，而不是接口。如果业务对象没有实现接口，默认使用 CGLIB 代理。面向接口编程是良好的习惯，尽量不要面向具体的类编程。因此，业务对象通常应实现一个或多个接口。下面是一个动态代理模式的示例，首先有个 Employee 的接口，接口如下：

```
//定义 Employee 接口
public interface Employee
{
    //info 方法声明
    public void info();
    //codeing 方法声明
    public void codeing();
}
```

然后给出该接口的实现的实现类，实现类必须实现两个方法，代码如下：

```
public class Programmer implements Employee {
    //info 方法实现，仅仅打印一个字符串
    public void info()
    {
        System.out.println("我是一名程序员");
    }
    //codeing 方法实现,仅仅打印一个字符串
    public void codeing()
    {
        System.out.println("我快乐的敲打着键盘");
    }
}
```

上面的代码没有丝毫独特之处，是典型的面向接口编程的模型，为了更好的解耦，采用工厂来创建 Programmer 实例。工厂源代码如下:

```
public class ProgrammerFactory {
    //工厂本身是单态模式，因此将 ProgrammerFactory 作为静态成员变量保存
    private static ProgrammerFactory instance;
    //将 Programmer 实例缓存
    private Programmer programmer;
    //默认的构造函数，单态模式需要构造器是 private
```

```
    private ProgrammerFactory()
    {}
    //单态模式所需的静态方法，该方法是创建本类实例的唯一方法点
    public static ProgrammerFactory getInstance()
    {
        if(instance==null)
        {
            instance=new ProgrammerFactory();
        }
        return instance;
    }
    //获得 Programmer 类的实例
    public Employee getProgrammer(String employeeName)
    {
        //根据字符串参数决定返回的实例
        if(employeeName.equalsIgnoreCase("programmer"))
        {
            //返回 Programmer 实例前，先判断缓存的 Programmer 是否存在，如果不存在创建，
            //否则直接返回缓存的 Programmer 实例
            if(programmer==null)
            {
                programmer=new Programmer();
            }
            return programmer;
        }
        return null;
    }
}
```

下面是一个普通的处理类。该处理类没有与任何特定的类耦合，它可以处理所有的目标对象。从 JDK1.3 起，Java 的 import java.lang.reflect 下增加 InvocationHandler 接口，该接口是所有处理类的根接口。该处理类的源代码如下：

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
public class ProxyHandler implements InvocationHandler {

    //需被代理的对象
    private Object target;
    //执行代理的目标方法时，该 invoke 方法会被自动调用
    public Object invoke(Object proxy,Method method,Object [] args) throws
Exception
    {
```

```
        Object result=null;
        if(method.getName().equals("info"))
        {
            System.out.println("事务开始");
            result=method.invoke(target, args);
            System.out.println("事务提交");
        }
        else
        {
            result=method.invoke(target, args);
        }
        return result;
    }
    //通过该方法，设置目标对象
    public void setTarget(Object o)
    {
        this.target=o;
    }
}
```

该处理类实现了 InvocationHandler 接口，实现该接口必须实现 invoke(Object proxy,Method method,Object [] args)方法，程序调用代理的目标方法，自动变成调用 invoke 方法。

该处理类并未与任何接口或类耦合，它完全是通用的，它的目标实例是 Object 类型的，可以是任何类型。在 invoke 方法内，对目标方法对象的 info 方法进行简单加强，在开始执行目标方法之前，先打印事务开始，执行方法之后，打印事务提交。

通过 method 对象的 invoke 方法，可以完成目标对象的方法调用，执行代码如下：

```
result=method.invoke(target, args);
```

下面是代理工厂：

```
import java.lang.reflect.Proxy;
public class ProxyFactory {

    public static Object getProxy(Object object)
    {
        //代理的处理类
        ProxyHandler handler=new ProxyHandler();
        //把该 programmer 类的实例委托给代理操作
        handler.setTarget(object);
        //第一个参数用来创建动态代理的 ClassLoader 对象,只要该对象能访问 Employee 接口即可
        //第二个参数是接口数组，正是代理该接口数组
```

```
        //第三个参数是代理包含的处理实例
        return     Proxy.newProxyInstance(Programmer.class.getClassLoader(),
object.getClass().getInterfaces(), handler);
    }
}
```

代理工厂里有行代码：

```
Proxy.newProxyInstance(Programmer.class.getClassLoader(),
object.getClass().getInterfaces(), handler);
```

Proxy.newProxyInstance 方法根据接口数组动态创建代理类的实例，接口数组通过 object.getClass().getInterfaces()方法获得，创建的代理类是 JVM 在内存中动态创建，该类实现传入的接口数组的全部接口。

Dynamic Proxy 要求被代理的必须是接口的实现类，否则无法为其构造相应的动态类。因此，Spring 对接口实现类采用 Dynamic Proxy 实现 AOP，而对没有实现任何接口的类，则通过 CGLIB 实现 AOP 代理。下面是主程序：

```
public class BeanTest {
    //主方法，程序入口
    public static void main(String args [])
    {
        Employee employee=null;
        //创建 Employee 实例，该实例将被作为代理对象
        Employee
targetObject=EmployeeFactory.getInstance().getEmployee("programmer");
        //以目标对象创建代理
        Object proxy=ProxyFactory.getProxy(targetObject);
        if(proxy instanceof Employee)
        {
            employee=(Employee)proxy;
        }
        //测试代理方法
        employee.info();
        employee.codeing();
    }
}
```

代理实例会实现目标对象实现的全部接口。因此，代理实例也实现了 Employee 接口，程序运行结果如下：

事务开始

我是一名程序员

事务提交

我快乐的敲打着键盘

代理实例加强了目标对象的方法仅仅打印两行字符串。当然，此种加强没有意义。试想一下程序中打印字符串的地方，换成真实的事务开始，事务提交，则代理实例的方法为目标对象的方法增加了事务性。

4.4 Spring 对 AOP 的支持

Spring AOP 采用纯 Java 实现，因此不需要额外编译。Spring AOP 不需要控制 Class-Loader 层次，因此适用于 J2EE Web 容器或应用服务器。Spring 目前支持拦截方法调用。成员变量拦截器没有实现。

Spring AOP 是对 Spring IoC 的补充，为 Spring 的声明式事务管理提供基础，或者使用 Spring AOP 框架的全部功能来实现自定义切面。Spring AOP 并不想提供完善的 AOP 实现，而是结合 Spring IoC 容器，提供实用的 AOP 实现，从而解决企业应用开发中的常见问题。

因此，Spring AOP 框架通常和 Spring IoC 容器整合使用。AOP 处理采用普通 bean 法来定义，处理和切入点由 Spring IoC 容器管理，这是和其他 AOP 实现的重要区别。

Spring 提供切入点和处理类型抽象的大量类和接口，使用术语 Advisor 来表示代表切面的对象，它包含一个处理和一个特定连接点的切入点。

各种处理类型有 MethodInterceptor，org.springframework.aop 中的处理接口。MethodInterceptor 接口来自 AOP 联盟，org.aoppalliance.aop.Advice 标签接口是所有处理的根接口。Spring 还包括如下的处理类：

- MethodInterceptor
- ThrowsAdvice
- BeforeAdvice
- AfterReturningAdvice

同一个处理，可以处理多个目标对象，这种处理成为 per-class 处理；也可以每个目标对象都有自己的处理，这种处理成为 per-instance 处理。

per-class 处理在应用中广泛的使用。它适合于通用处理，例如事务 Advistor。per-class 处理不关心目标对象的状态，也不会修改目标对象的状态，这种处理仅仅对方法和方法参数起作用，per-instance 处理关心目标对象的状态，处理可能改变目标对象的状态。per-class 处理和 per-instance 处理，可以在同一个 AOP 代理中混合使用。Spring 中实用的处理类型有如下 5 种：

- Around 处理
- Before 处理
- Throws 处理
- After Returing 处理
- Introduction 处理

4.4.1 Around 处理

Around 处理是 Spring 中最基本的处理类型。Spring 的 Interceptor Around 处理和 AOP 联

盟兼容。实现 Around 处理需要实现 MethodInterceptor 接口。MethodInterceptor 由 AOP 联盟提供，MethodInterceptor 接口的源代码如下：

```
public interface MethodInterceptor extends Interceptor{
    Object invoke(MethodInvocation invocation) throws Throwable;
}
```

程序调用代理实例的方法，invoke 方法完成实际的方法体。该方法包含 MethodInvocation 参数，MethodInvocation 也是 AOP 联盟提供的。该参数当中包含被调用的方法、目标连接点、AOP 代理和调用方法的参数，通过 MethodInvocation 可以执行目标方法。invoke()方法返回调用的结果，即连接点的返回值。下面是一个简单的 Around 处理源代码：

```
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
//创建 Around 处理应该实现 MethodInterceptor 接口
public class MyAroundInterceptor implements MethodInterceptor{

    //实现 MethodInterceptor 接口必须实现的方法，该方法处理完成的动作
    public Object invoke(MethodInvocation invocation) throws Throwable
    {
        //执行目标方法之前的方法
        System.out.println("调用方法前:invocation 对象["+invocation+"]");
        //执行目标方法
        Object rval=invocation.proceed();
        //执行目标方法后的处理
        System.out.println("调用结束....");
        return rval;
    }
}
```

目标类和所实现的接口如下：

```
//目标类所实现的接口
public interface Person {
    //info 方法声明
    void info();
}
//目标类，实现接口
public class PersonImpl implements Person {
    //两个成员属性
    private String name;
    private int age;

    //name 属性依赖注入所需的 setter 方法
```

```
    public void setName(String name) {
        this.name = name;
    }
    //age 属性依赖注入所需的 setter 方法
    public void setAge(int age) {
        this.age = age;
    }
    //测试用的 info 方法，该方法仅仅打印 name 属性和 age 属性的值
    public void info()
    {
        System.out.println("我的名字是:"+name+",我今年的年龄为:"+age);
    }
}
```

下面是完整的配置文件（配置文件配置了 bean、处理 bean、代理工厂 bean、Spring Ioc 容器负责管理 bean 之间的依赖关系）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 配置目标 bean 实例 -->
<bean id="personTarget" class="PersonImpl">
<!-- 下面是依赖注入的属性值 -->
<property name="name">
<value>李明</value>
</property>
<property name="age">
<value>51</value>
</property>
</bean>
<!-- 配置 Around 处理 bean -->
<bean id="myAroundInterceptor" class="MyAroundInterceptor"></bean>
<!-- 配置工厂 bean -->
<bean id="person"
            class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>Person</value>
</property>
<property name="target">
<ref local="personTarget"/>
```

```
</property>
<property name="interceptorNames">
<list>
<value>myAroundInterceptor</value>
</list>
</property>
</bean>
</beans>
```

注意工厂 bean 的配置，工厂 bean 采用 ProxyFactoryBean。如果程序中请求获得工厂 bean ID，实际将不会获得 bean 的实例，而是获得工厂 bean 的 getObject()返回对象。ProxyFactoryBean 工厂 bean 将返回目标对象的 AOP 代理，主程序如下：

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
ClassPathXmlApplicationContext;
public class BeanTest {
    public static void main(String args [])
    {
        //创建 Spring 容器实例
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //获取 AOP 代理实例
        Person p=(Person)context.getBean("person");
        //执行 AOP 代理方法
        p.info();
    }
}
```

主程序没有太多的变化，与之前的区别在于：原来直接请求目标 bean ID；使用 AOP 代理后，改为请求代理工厂 bean ID，IoC 不会返回工厂实例，而是返回工厂生成的代理实例，程序的执行结果如下：

调用方法前：invocation 对象[ReflectiveMethodInvocation: public abstract void Person.info(); target is of class [PersonImpl]]

我的名字是:李明,我今年的年龄为:51

调用结束....

将该示例的执行结果与 4.3 节的示例的执行结果相比，效果相似。代理在调用方法之前，调用方法之后，加入了处理。这种处理，可以是相当简单的，也可以是相当复杂。比如增加事务管理，或执行权限检查。

4.4.2 Before 处理

Before 处理是一种简单的处理，这种处理不需要 MethodInvocation 对象，因为它仅在方

法调用之前执行处理，而目标方法的继续执行无须处理关心。

Before 处理不需要调用 Proceed（）方法，不会产生忘记调用目标方法的错误，不会停止拦截器链的继续执行。实现 Before 处理必须实现 MethodBeforeAdvice 接口，该接口的源代码如下：

```
//before 处理实现类
public interface MethodBeforeAdvice extends BeforeAdvice
{
     //该方法在连接点之前被执行
     void before(Method m,Object[] args,Object target) throws Throwable;
}
```

before 方法的返回值是 void，意味着 Before 处理在连接点之前执行处理，但不能改变方法的返回值。如果 Before 处理抛出异常，拦截器链的执行被中断，该异常将沿着拦截器链向上传播。如果异常是 unchecked 的，或者出现在目标方法声明中，异常直接返回给客户端；否则，异常由 AOP 代理包装成 unchecked。Before 处理可以被用于任何类型的切入点。下面是一个简单的 Before 处理实现类：

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;
//before 处理实现类
public class MyBeforeAdvisor implements MethodBeforeAdvice {

    //实现 MethodBeforeAdvice 接口必须实现的方法，该方法在连接点之前执行
    public void before(Method arg0, Object[] arg1, Object arg2)
            throws Throwable {
        System.out.println("方法调用前...");
        System.out.println("下面是方法调用的信息：");
        System.out.println("所执行的方法是:"+arg0);
        System.out.println("调用方法的参数:"+arg1);
        System.out.println("目标对象是:"+arg2);
    }
}
```

程序所有的接口和目标类，与之前的实例非常相似：

```
//目标类所实现的接口
public interface Person {
    //run 方法声明
    void run();
}
//目标类，实现接口
public class PersonImpl implements Person {
    //两个成员属性
```

```
    private String name;
    private int age;

    //name 属性依赖注入所需的 setter 方法
    public void setName(String name) {
        this.name = name;
    }
    //age 属性依赖注入所需的 setter 方法
    public void setAge(int age) {
        this.age = age;
    }
    //实现 Person 接口必须实现的 run 方法
    public void run()
    {
        if(age<45)
        {
            System.out.println("我还年轻，奔跑迅速...");
        }
        else
        {
            System.out.println("我年老体弱，只能慢跑...");
        }
    }
}
```

将目标 bean 和处理 bean 部署在容器中，同时也部署代理工厂 bean。完整的配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 配置目标 bean 实例 -->
<bean id="personTarget" class="PersonImpl">
<!-- 下面是依赖注入的属性值 -->
<property name="name">
<value>李明</value>
</property>
<property name="age">
<value>28</value>
</property>
```

```
</bean>
<!-- 配置 before 处理 bean -->
<bean id="myBeforeAdvisor" class="MyBeforeAdvisor"></bean>
<!-- 配置工厂 bean -->
<bean id="person"
      class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>Person</value>
</property>
<property name="target">
<ref local="personTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>myBeforeAdvisor</value>
</list>
</property>
</bean>
</beans>
```

程序和上例主程序的程序差别不大，只是上示例中的执行 info 方法，本示例中换成执行 run 方法，此处不在给出主程序，程序运行结果如下：

方法调用前...

下面是方法调用的信息：

所执行的方法是:public abstract void Person.run()

调用方法的参数:[Ljava.lang.Object;@bad8a8

目标对象是:PersonImpl@5e13ad

我还年轻，奔跑迅速...

虽然 Before 处理无须理会目标方法的执行，但程序中依然可以获得目标方法的相关信息，如目标方法的方法名、方法调用的参数、目标对象。

4.4.3 After Returning 处理

After Returning 处理和 Before 处理非常的相似，不需要 MethodInvocation 对象，目标方法的继续执行无须处理关心。实现 After Returning 处理，必须实现 org.springframework.aop.AfterReturningAdvice 接口，该接口的源代码如下所示：

```
public interface AfterReturningAdvice extends Advice
{
    void afterReturning(Object returnvalue,Method m,Object[] args,Object
target) throws Throwable;
}
```

看上面的源代码，afterReturning 方法在目标方法返回之后自动调用，afterReturning 可以访问目标方法的返回值，但不能修改。afterReturning 也可以访问目标方法、方法参数和目标对象。

After Returning 处理可被用于任何类型的切入点。下面是一个简单的 After Returning 处理实现类：

```
import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
//实现自定义处理类，必须实现 AfterReturningAdvice 接口
public class MyAfterAdvisor implements AfterReturningAdvice {
    //实现 AfterReturningAdvice 接口必须实现的 afterReturning 方法
    public void afterReturning(Object arg0, Method arg1, Object[] arg2,
            Object arg3) throws Throwable {
        System.out.println("方法调用结束......");
        System.out.println("目标方法的返回值是:"+arg0);
        System.out.println("目标方法是:"+arg1);
        System.out.println("目标方法的参数是:"+arg2);
        System.out.println("目标对象是:"+arg3);
    }
}
```

程序所有的接口和目标类也非常简单：

```
//目标类所实现的接口
public interface Person {
    //test 方法声明
    String test();
}
//目标类，实现接口
public class PersonImpl implements Person {
    //两个成员属性
    private String name;
    private int   age;

    //name 属性依赖注入所需的 setter 方法
    public void setName(String name) {
        this.name = name;
    }
    //age 属性依赖注入所需的 setter 方法
    public void setAge(int age) {
        this.age = age;
    }
    //实现 Person 接口必须实现的 test 方法
```

```
    public String test()
    {
        System.out.println("我的名字是:"+name+",今年的年龄为:"+age);
        return "我的名字是:"+name+",今年的年龄为:"+age;
    }
}
```

将目标 bean 和处理 bean 部署到容器中，也部署代理工厂 bean。本实例的处理与切入点配置在同一个 bean 之中，切入点采用正则表达式切入点配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 配置目标 bean 实例 -->
<bean id="personTarget" class="PersonImpl">
<!-- 下面是依赖注入的属性值 -->
<property name="name">
<value>李明</value>
</property>
<property name="age">
<value>51</value>
</property>
</bean>
<!-- 配置 after return 处理 bean,该处理 bean 与正则表达式切入点一起配置 -->
<bean                                                    id="testAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
<property name="advice">
<bean class="MyAfterAdvisor"></bean>
</property>
<property name="patterns">
<list>
<value>.*test.*</value>
</list>
</property>
</bean>
<!-- 配置工厂 bean -->
<bean                                                  id="person"
class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>Person</value>
```

```
</property>
<property name="target">
<ref local="personTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>testAdvisor</value>
</list>
</property>
</bean>
</beans>
```

主程序与前面示例的主程序差别不大，本示例换成执行 test 方法，此处不在给出主程序，程序运行结果如下：

我的名字是:李明,今年的年龄为:51

方法调用结束.....

目标方法的返回值是:我的名字是:李明,今年的年龄为:51

目标方法是:public abstract java.lang.String Person.test()

目标方法的参数是:null

目标对象是:PersonImpl@156b6b9

4.4.4 Throws 处理

当连接点抛出异常，Throws 处理被调用。实现 Throws 异常，必须实现 org.springframework.aop.ThrowsAdvice 接口，该接口不包含任何方法，但实现该接口必须实现如下形式的方法。

```
afterThrowing([Method],[args],[target],Throwable subclass)
```

可以实现一个或多个这样的方法。这些方法中，只有第四个参数是必需的，前三个参数可选，取决于应用是否关心抛出异常的宿主。最后一个参数确定连接点抛出哪种异常，处理被调用。Throws 处理可被用于任何类型的切入点。下面的示例代理中，当连接点抛出 RemoteException 异常时，该处理被调用：

```
public class RemoteThrowsAdvice implements ThrowsAdvice
{
Public void afterThrowing(RemoteException ex) throws Throwable
{}
}
```

下面的示例，当连接点抛出 RemoteException 异常时，该处理被调用，与上面的处理不同的是，该处理声明四个参数。因此，该处理可以访问被调用的方法，方法的参数和目标对象：

```
public class RemoteThrowsAdvice implements ThrowsAdvice
```

```
{
Public void afterThrowing(Method m,Object[] args,Object target,RemoteException
ex) throws Throwable
{}
}
```

一个完整的示例，该实例中的处理组合了两个 afterThrowing 方法，该处理能同时处理两类异常。事实上，一个处理中，可以组合任意多个 afterThrowing 方法。该处理的源代码如下：

```
import org.springframework.aop.ThrowsAdvice;
import java.lang.reflect.Method;
import java.lang.ArrayIndexOutOfBoundsException;
//Throws 异常应该实现 ThrowsAdvice 接口
public class MyExceptionAdvice implements ThrowsAdvice {

    //该方法处理 ArrayIndexOutOfBoundsException 异常
    public  void  afterThrowing(ArrayIndexOutOfBoundsException  ex)  throws
Throwable
    {
        System.out.println("系统抛出 ArrayIndexOutOfBoundsException 异常，异常提
示为："+ex.getMessage());
    }
    //该方法处理 ArithmeticException 异常。
    //而且访问抛出异常的方法，方法参数，目标对象
    public void afterThrowing(Method m,Object[] args,Object target,
                                                    ArithmeticException ex)
    {
        System.out.println("系统抛出 ArithmeticException 异常，异常提示为：
"+ex.getMessage());
        System.out.println("抛出异常的方法为:"+m);
        System.out.println("抛出异常的方法参数为:"+args);
        System.out.println("抛出异常的目标对象为:"+target);
    }
}
```

程序所有的接口和目标类也非常简单：

```
//目标类所实现的接口
public interface TestBean {
    void test();
    void anthortest();
}
//目标类，实现接口
```

```
public class TestBeanImpl implements TestBean{
    //该方法抛出 ArithmeticException 异常
public void test()
    {
        int i=5/0;
    }
   //该方法抛出 ArrayIndexOutOfBoundsException 异常
public void anthortest()
    {
        Integer [] arr=new Integer[5];
        for(int i=0;i<7;i++)
        {
            arr[i]=i;
        }
    }
}
```

配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 配置目标 bean 实例 -->
<bean id="testTarget" class="TestBeanImpl"></bean>
<!-- 定义处理 bean -->
<bean id="myExceptionAdvice" class="MyExceptionAdvice" ></bean>
<!-- 配置工厂 bean -->
<bean id="test" class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>TestBean</value>
</property>
<property name="target">
<ref local="testTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>myExceptionAdvice</value>
</list>
</property>
</bean>
```

```
</beans>
```

当主程序调用 test 方法时的执行结果：

系统抛出 ArithmeticException 异常，异常提示为：/ by zero

抛出异常的方法为:public abstract void TestBean.test()

抛出异常的方法参数为:null

抛出异常的目标对象为:TestBeanImpl@b5dac4

当主程序调用 anthortest 方法时的执行结果:

系统抛出 ArrayIndexOutOfBoundsException 异常，异常提示为：5

4.4.5 Introduction 处理

Introduction 处理是一种特殊类型的拦截处理。它不能作用于任何切入点，因为它只作用于类级别方法。实现 Introduction 处理，需要实现 IntroductionAdvisor 和 IntroductionInterceptor 接口。IntroductionInterceptor 接口的源代码如下：

```
public interface IntroductionInterceptor extends MethodInterceptor
{
        Boolean implmentsInterfaces(Class intf);
}
// IntroductionInterceptor 继承 AOP 联盟 MethodInterceptor 接口，接口源代码如下:
public interface  MethodInterceptor
{
   public  java.lang.Object  invoke(MethodInvocation  invocation)  throws
java.lang.Throwable
}
```

实现 Introduction 处理必须实现 invoke（）方法。Introduction 处理不能调用 proceed（）方法。Introduction 处理不能作用于任何切入点，通常建议使用 IntroductionAdvisor 包装该处理。IntroductionAdvisor 的源代码如下：

```
//包装 IntroductionInterceptor 的 IntroductionAdvisor
public interface  IntroductionAdvisor extends Advisor,IntroductionInfo
{
    ClassFilter getClassFilter():
    void validateInterfaces() throws IllegalArgumentException;
}
```

其父接口 IntroductionInfo 的源代码如下：

```
public interfaces IntroductionInfo
{
  //该方法返回 Introduction 的接口
  Class [] getInterfaces();
```

```
}
```

getInterfaces（）方法返回 Advistor 导入的接口。Introduction 处理能将目标对象转换成代理接口，不管目标对象原本的类型，Introduction 处理后的代理能够直接调用代理接口中的方法。如下是代理将实现的接口的源代码：

```
public interface Lockable
{
   //锁定目标 bean 的属性
   void lock();
//解锁
void unlock();
//返回目标 bean 的属性是否被锁定
boolean locked();
}
```

实例程序的代理将实现该接口，目标 bean 的代理可执行该接口里的方法。一旦执行 lock 方法，目标 bean 的属性被锁定，将在控制台显示错误信息。

下面是 Introduction 处理的实现类，让其继承 DelegatingIntroductionInterceptor 类。实现 Introduction 处理可直接实现 IntroductionInterceptor 接口,但实现 DelegatingIntroductionInterceptor 类更简单、方便，并让其实现代理的全部接口。Introduction 处理的源代码如下：

```
import org.aopalliance.intercept.MethodInvocation;
import org.springframework.aop.support.DelegatingIntroductionInterceptor;
//Introduction 处理，继承 DelegatingIntroductionInterceptor，实现代理接口
public class LockMixin extends DelegatingIntroductionInterceptor implements
        Lockable {
    //locked 是个标始，该标始表示目标 bean 的属性是否可修改
    //该标始会作为代理的状态，因此 Introduction 处理是代理的一部分
    private boolean locked;
    //锁定目标 bean 的属性
    public void lock()
    {
        this.locked=true;
    }

    public boolean locked()
    {
        return this.locked;
    }

    public void unlock()
    {
```

```
        this.locked=false;
    }
    //不能调用 peoceed 方法，必须实现 invoke 方法
    public Object invoke(MethodInvocation invocation) throws Throwable
    {
        if(locked() && invocation.getMethod().getName().indexOf("set")==0)
        {
            System.out.println("属性加锁，无法修改");
        }
        return super.invoke(invocation);
    }
}
```

LockMixin 继承 DelegatingIntroductionInterceptor 并实现 Lockable。通过这种方式，可以导入任意数量的接口。通常不需要改写 invoke()方法，只需实现 DelegatingIntroductionInterceptor。如果是导入的方法，DelegatingIntroductionInterceptor 实现委托方法，否则沿着连接点继续处理。在示例程序中增加检查，如果属性被锁定，且客户端代码调用 setter 方法，则打印错误。

Introduction Advistor 必须有一个独立的 LockMixin 实例，并指定导入的接口 Lockable。本示例中通过 new 直接创建实例。实际应用中，通过 IoC 容器为其注入 Introduction 拦截器。

IntroductionAdvisor 的源代码如下：

```
import org.springframework.aop.support.DefaultIntroductionAdvisor;
public class LockMixinAdvisor extends DefaultIntroductionAdvisor {
    public LockMixinAdvisor()
    {
        super(new LockMixin(),Lockable.class);
    }
}
```

将 Introduction Advisor 部署子容器中，不需要额外的属性注入。Advistor 必须针对每个实例，并且是有状态的。实例代码中，每个目标 bean 都有对应的 LockMixinAdvisor，Advisor 是目标对象的一部分，因此该 Advisor 应部署成 prototype。

下面是完整的配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

<!-- 配置目标 bean 实例 -->
```

```
<bean id="personTarget" class="PersonImpl"  >
<!-- 下面是依赖注入的属性值 -->
<property name="name">
<value>李明</value>
</property>
<property name="age">
<value>51</value>
</property>
</bean>
<!-- 配置 Around 处理 bean -->
<bean id="lockMixinAdvisor" class="LockMixinAdvisor" scope=”prototype” />
<!-- 配置工厂 bean -->
<bean id="person"
      class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>Person</value>
</property>
<property name="target">
<ref local="personTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>lockMixinAdvisor</value>
</list>
</property>
</bean>
</beans>
```

程序所有的接口和目标类也非常简单：

```
//目标类所实现的接口
public interface Person {
    //test 方法声明
    String test();
   void setAge(int age);
}
//目标类，实现接口
public class PersonImpl implements Person {
    //两个成员属性
    private String name;
    private int    age;

    //name 属性依赖注入所需的 setter 方法
```

```
    public void setName(String name) {
        this.name = name;
    }
    //age 属性依赖注入所需的 setter 方法
    public void setAge(int age) {
        this.age = age;
    }
    //实现 Person 接口必须实现的 test 方法
    public String test()
    {
        System.out.println("我的名字是:"+name+",今年的年龄为:"+age);
        return "我的名字是:"+name+",今年的年龄为:"+age;
    }
}
```

主程序中，让代理执行 lock()方法锁定目标 bean 的属性，然后调用 setter 方法修改属性。主程序源代码如下：

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class BeanTest {
    public static void main(String args [])
    {
        //创建 Spring 容器实例
        ApplicationContext context=
              new ClassPathXmlApplicationContext("applicationContext.xml");
        //获取 AOP 代理实例
        Person p=(Person)context.getBean("person");
        //执行 AOP 代理方法
        p.test();
        //修改代理 bean 的属性，此时并未调用 lock 方法，修改成功
        p.setAge(45);
        p.test();
        //将代理强制转型成 Lockable
        Lockable lp=(Lockable)p;
        //加锁
        lp.lock();
        //修改属性，会打印错误信息
        p.setAge(34);
    }
}
```

程序执行结果：

我的名字是:李明,今年的年龄为:51

我的名字是:李明,今年的年龄为:45

属性加锁，无法修改

4.4.6 Advistor

Advistor 等于切入点加处理。Advistor 是切面的模块化表示。除了 Introduction 处理，Advisor 可被用于任何处理。DefaultPointcutAdvisor 是最常用的 Advistor。例如，它可以和 MethodInterceptor,BeforeAdvice 和 ThrowsAdvice 一起使用。Spring 可在一个 AOP 代理中混合使用 Advistor 和处理,Spring 将自动创建拦截器链。

下面示例中，有两个 Advice bean，一个 Advisor bean。Spring 创建拦截器链，拦截器的源代码就是上面示例中的 Around 处理、Before 处理、After 处理的源代码，此处不在给出。目标对象的类和接口如下 ：

```
//目标类所实现的接口
public interface Person {
    //info 方法声明
    void info();
    //run 方法声明
    void run();
 }
//目标类，实现接口
public class PersonImpl implements Person {
    //两个成员属性
    private String name;
    private int   age;

    //name 属性依赖注入所需的 setter 方法
    public void setName(String name) {
        this.name = name;
    }
    //age 属性依赖注入所需的 setter 方法
    public void setAge(int age) {
        this.age = age;
    }
    //实现 Person 接口必须实现的 info 方法
    public void info()
    {
        System.out.println("我的名字是:"+name+",今年的年龄为:"+age);
    }
    //实现 Person 接口必须实现的 run 方法
    public void run()
```

```
    {
        if(age>45)
        {
            System.out.println("我还轻，奔跑迅速");
        }
        else
        {
            System.out.println("我年老体弱，只能慢跑");
        }
    }
}
```

完整的配置文件如下（配置文件的代理工厂里，同时配置了三个拦截器，三个拦截器形成拦截链）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

<!-- 配置目标 bean 实例 -->
<bean id="personTarget" class="PersonImpl"  >
<!-- 下面是依赖注入的属性值 -->
<property name="name">
<value>李明</value>
</property>
<property name="age">
<value>51</value>
</property>
</bean>
<!-- 配置 Around 处理 bean -->
<bean id="myAroundInterceptor" class="MyAroundInterceptor"></bean>
<!-- 配置 before 处理 bean -->
<bean id="myBeforeAdvisor" class="MyBeforeAdvisor"></bean>
<!-- 配置 after return 处理 bean,该处理 bean 与正则表达式切入点一起配置 -->
<bean id="testAdvisor"
     class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
<property name="advice">
<bean class="MyAfterAdvisor"></bean>
</property>
<property name="patterns">
```

```
<list>
<value>.*run.*</value>
</list>
</property>
</bean>
<!-- 配置工厂 bean -->
<bean id="person"
       class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>Person</value>
</property>
<property name="target">
<ref local="personTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>myAroundInterceptor</value>
<value>myBeforeAdvisor</value>
<value>testAdvisor</value>
</list>
</property>
</bean>
</beans>
```

拦截器的定义顺序很重要。拦截器链前面的处理先被调用，链后面的处理后被调用。看程序运行结果：

调用方法前:invocation 对象[ReflectiveMethodInvocation: public abstract void Person.run(); target is of class [PersonImpl]]

方法调用前...

下面是方法调用的信息：

所执行的方法是:public abstract void Person.run()

掉用方法的参数:[Ljava.lang.Object;@1977b9b

目标对象是:PersonImpl@195d4fe

我还轻，奔跑迅速

方法调用结束.....

目标方法的返回值是:null

目标方法是:public abstract void Person.run()

目标方法的参数是:[Ljava.lang.Object;@340101

目标对象是:PersonImpl@195d4fe

调用结束....

执行结果是目标方法执行之前：Around 处理先执行，Before 处理后执行；目标方法执行之后：先执行正则表达式 after 处理，后执行 Around 处理。

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 UserInfoManager 工程并同时添加 Hibernate 和 Spring 支持。（10 分钟）
2. 编写 Userinfo.java 类和 Userinfo.hbm.xml 配置文件。（15 分钟）
3. 编写 UserinfoDao.java。（10 分钟）
4. 编写 UserServiceInterfaces.java 接口和 UserService.java 类。（15 分钟）
5. 编写 MyBeforeAdvisor.java 类。（10 分钟）
6. 编写 applicationContext.xml 和 hibernate.cfg.xml 配置文件。（20 分钟）
7. 编写主程序,并调试运行程序。（10 分钟）

6. 作业

利用 Struts aop 打印程序日志。

专题四 SSH 整合

1. 教学目标

1.1 Spring 与 Hibernate 整合

1.2 Spring 与 Struts 整合

2. 工作任务

SSH 整合实现论坛登录

3. 相关实践知识

SSH 整合实现论坛登录

1. 首先建立一个 Web 项目，命名 SSHLogin。

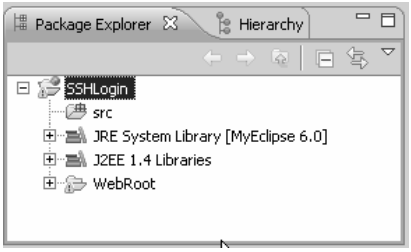


图 4-1 建立 Web 工程

2. 然后在项目中加入 Struts 支持。

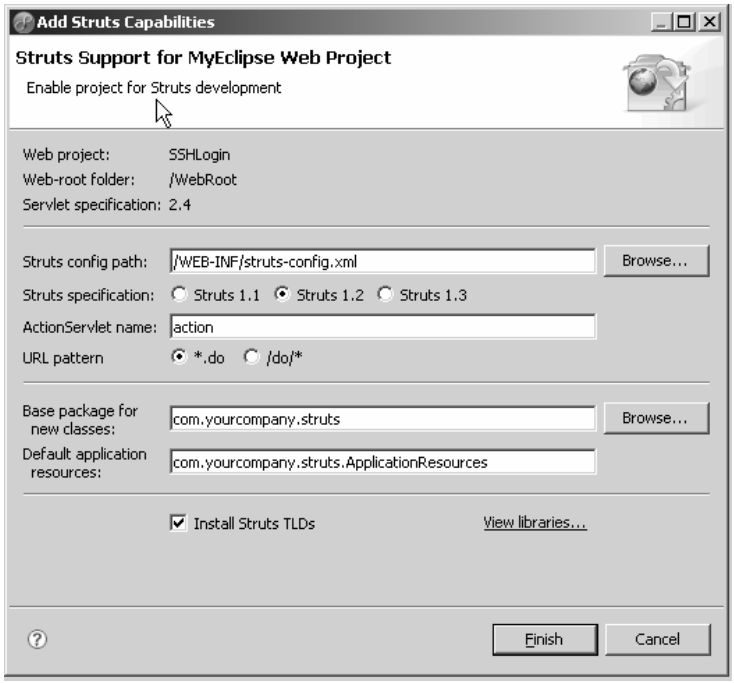


图 4-2 添加 Struts 支持

3. 添加 Spring 支持，右键选中工程【MyEclipse】➔【Add Springcapabilities】，选中四个 Libraries，分别是 Spring 2.0 AOP Libraries、Spring 2.0 Core Libraries、Spring 2.0 Persistence Libraries、Spring 2.0 Web Libraries，点击【Next】。

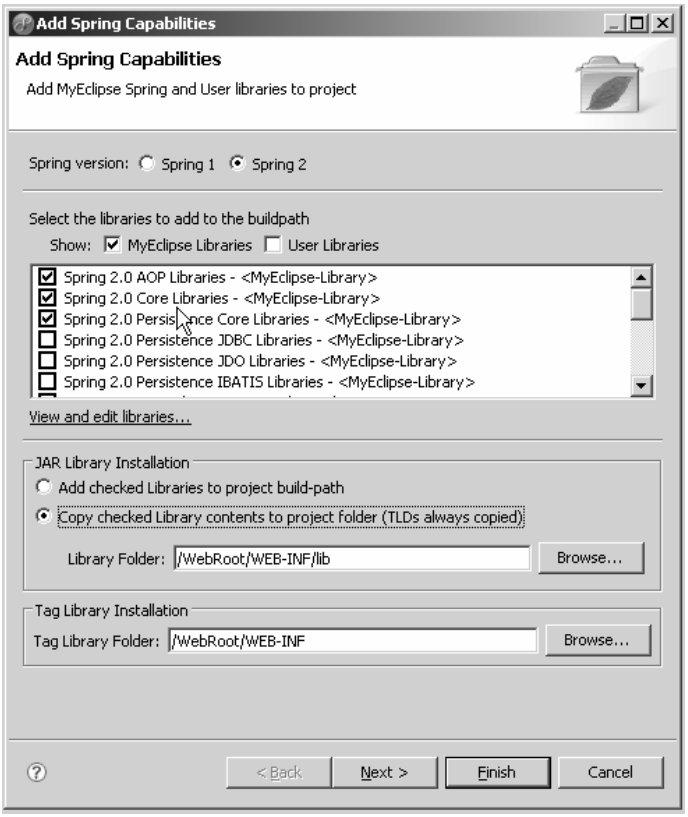


图 4-3 添加 Spring 支持

4. 将 applicationContext.xml 放在 WEB-INF 目录当中，点击【Finish】，完成 Spring 支持环境的添加。

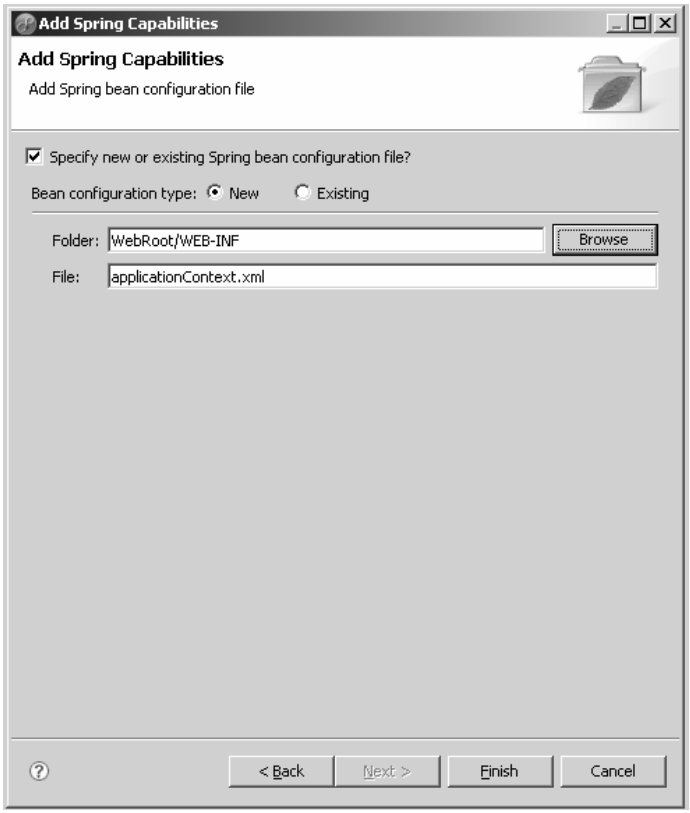


图 4-4 将 Spring 配置文件放在 WEB-INF 下

5. 添加 Hibernate 支持，右键选中工程【MyEclipse】➔【Add Hibernatecapabilities】,选中三个 Libraries，分别是 Hibernate 3.1 Core Libraries、Hibernate 3.1 Advanced Support Libraries、Spring 2.0 Persistence Libraries，点击【Next】。

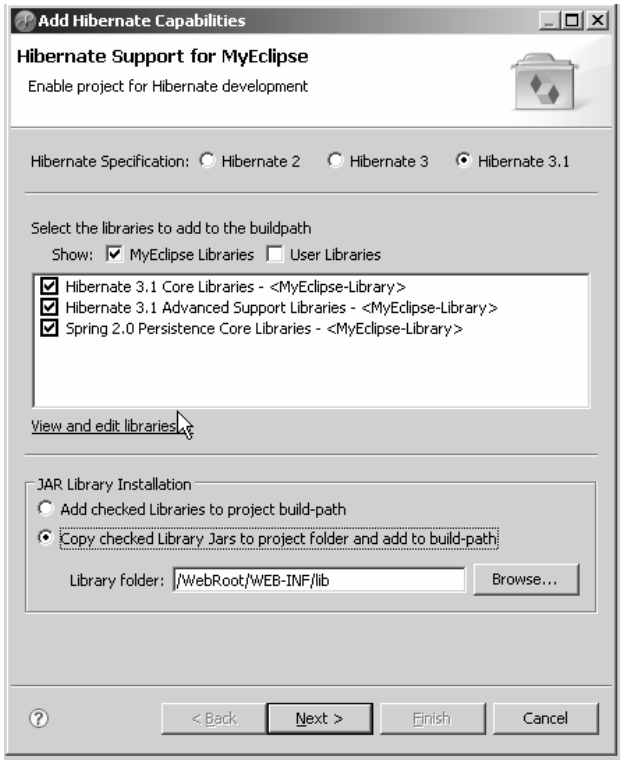


图 4-5 添加 Hibernate 支持

6. 选中 Spring configuration file(applicationContext.xml)，点击【Next】。

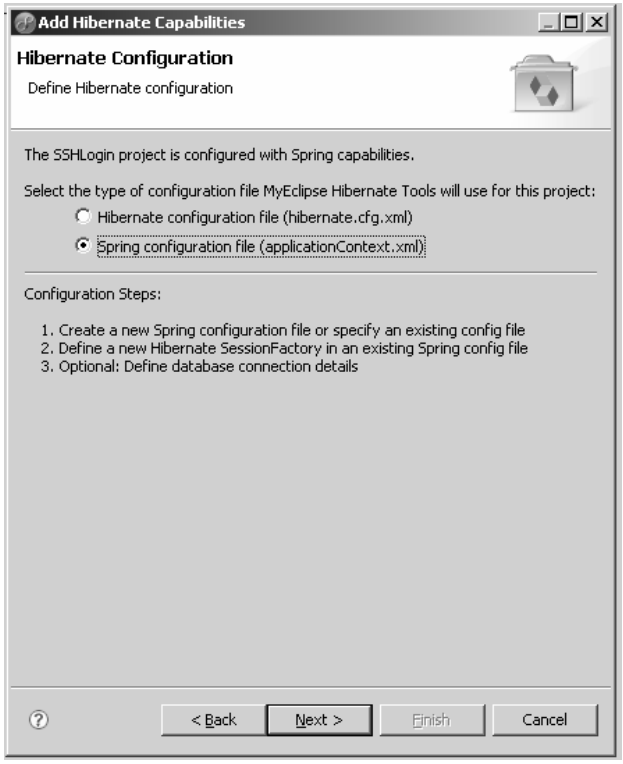


图 4-6 选择数据源采用的配置文件

7. 选中 Existing Spring configuration file，在 SessionFactory ID 中填写“sessionFactory”，点击【Next】。

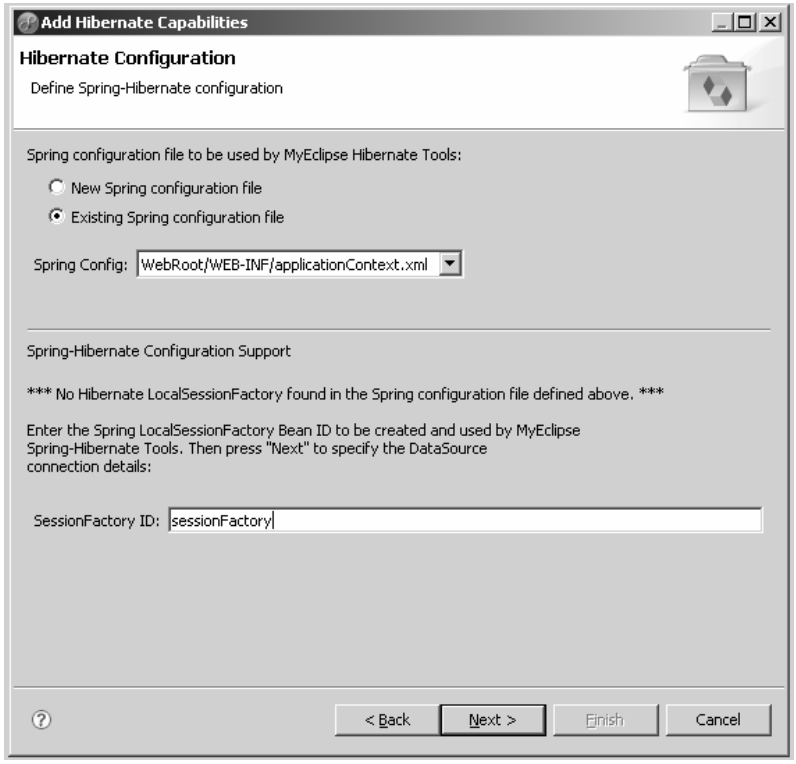


图 4-7 填写 SessionFactory ID

8. Bean Id 填写 dataSource，DB Driver 选择我们建好的驱动，点击【Next】。

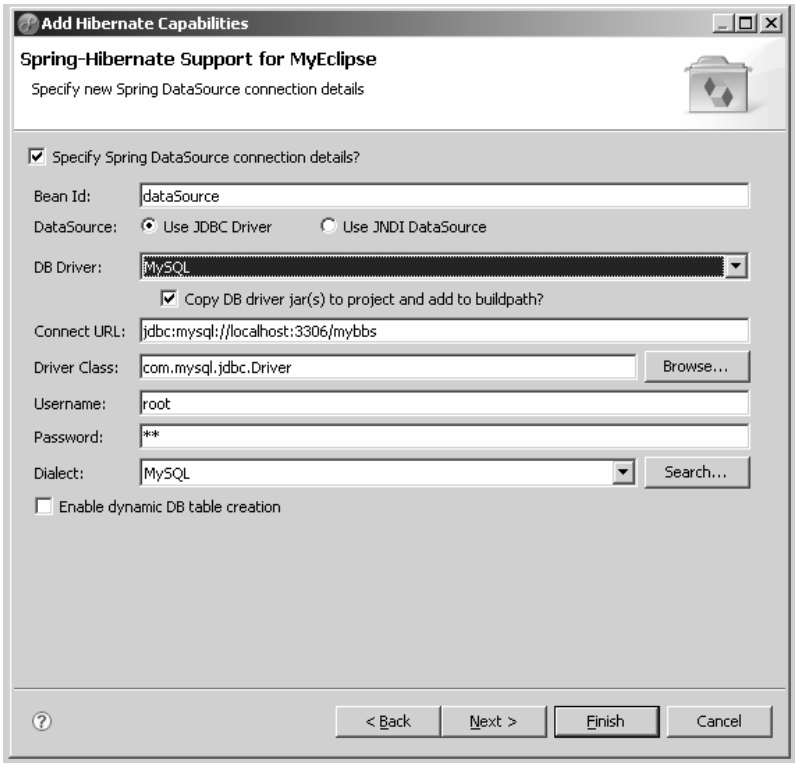


图 4-8 配置数据源

9. 将“Create SesssionFactory Class?”的钩去掉，点击【Finish】。

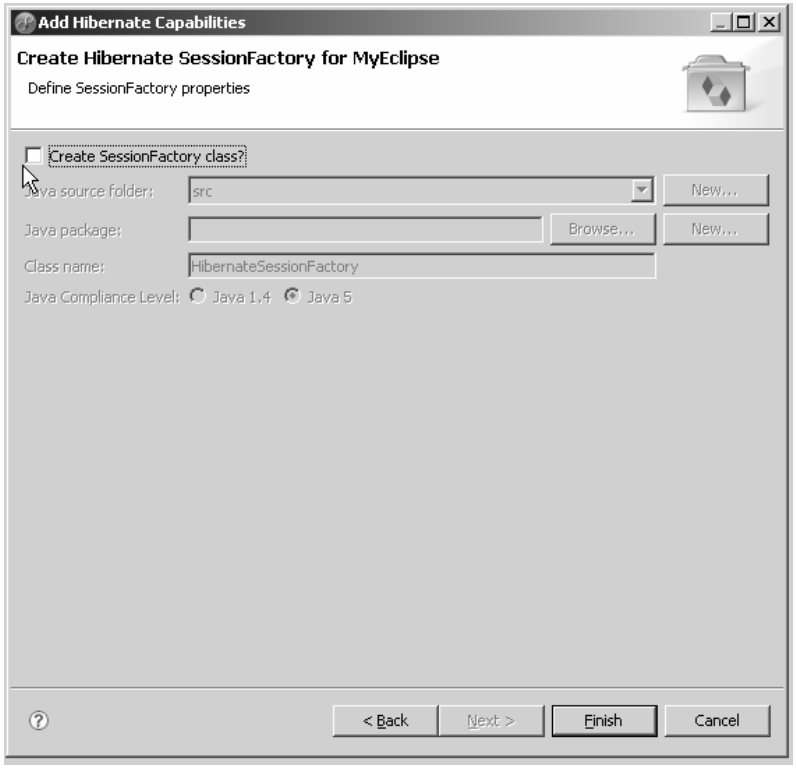


图 4-9 去掉“Create SesssionFactory Class?”的钩

10. 工具会弹出对话框，我们选中【Keep Existing】点击。

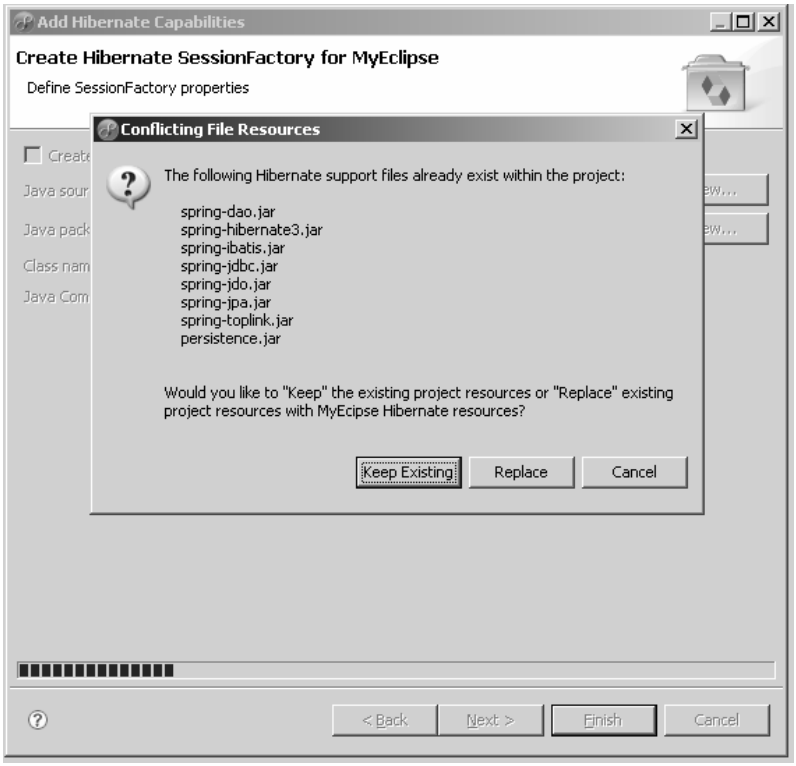


图 4-10 点击 Keep Existing

11. 此时 applicationContext.xml 会报错，因为缺少 commons-pool.jar 和 commons-dbcp.jar，我们手动添加。

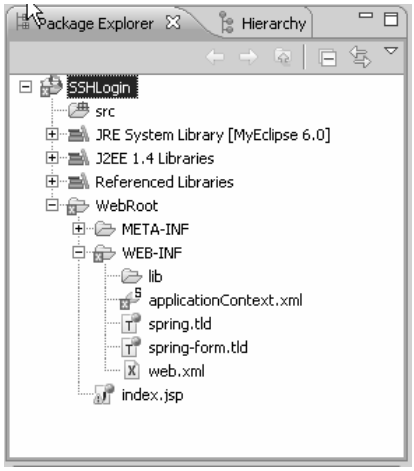


图 4-11 缺少 jar 包 applicationContext.xml 报错

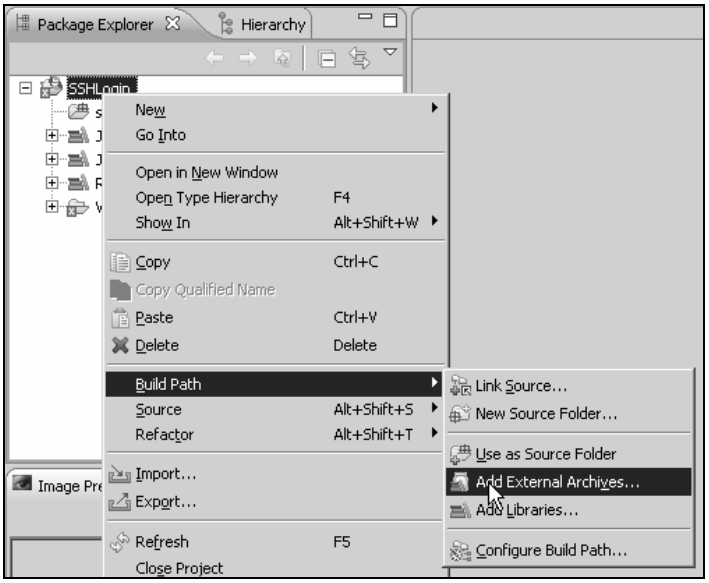


图 4-12 手动添加 jar 包

12.为了解决 SSH 整合 jar 包冲突问题，我们需要删除三个 jar 包 asm-2.2.3.jar，asm-commons-2.2.3.jar，asm-util-2.2.3.jar，此时工程建立完毕。

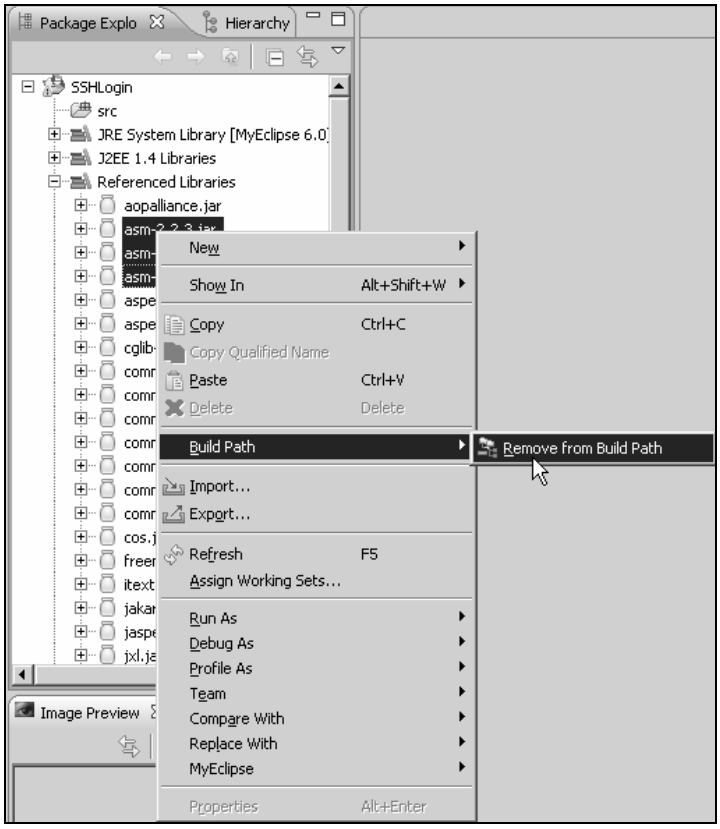


图 4-13 删除冲突 jar 包

13. 新建 com.pojo 和 com.dao 包，利用 MyEclipse 当中的 MyEclipse Database Explorer 工具在 com.pojo 包下生成 Userinfo.java 类，UserinfoDAO.java 类和 Userinfo.hbm.xml，并将 UserinfoDAO.java 类放入 com.dao 包中。

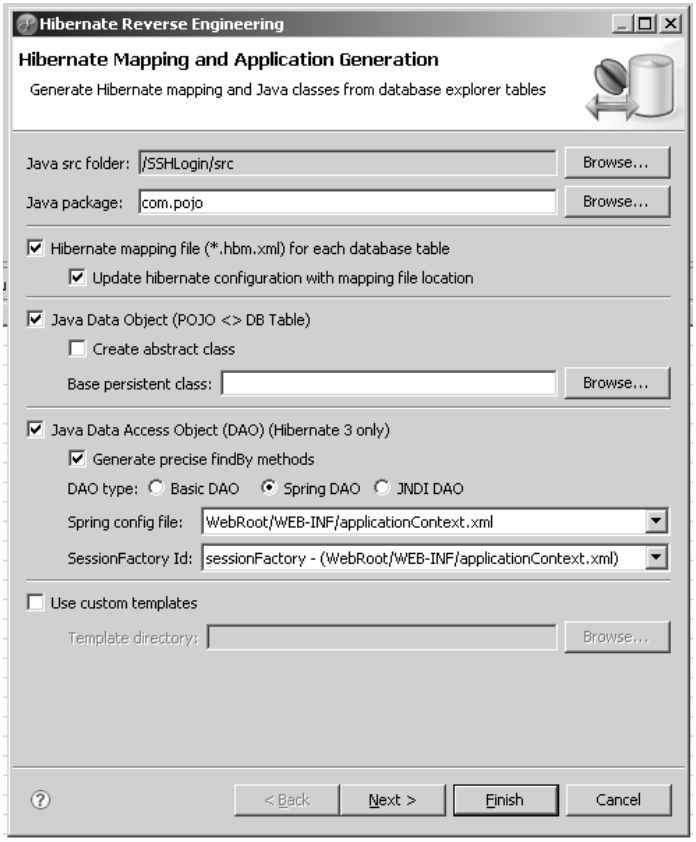


图 4-14 生成数据访问类

```
package com.pojo;
public class Userinfo implements java.io.Serializable {
    //用户 id
    private Integer userid;
    //用户名
    private String username;
    //密码
    private String userpwd;
    //密码找回问题
    private String userques;
    //密码找回答案
    private String userans;
    //积分
    private Integer integral;
    //等级
    private String grade;
```

```
    //email
    private String useremail;
    //性别
    private Byte sex;

    public Userinfo() {
    }

    public Userinfo(Integer userid, String username, String userpwd,
            String userques, String userans, Integer integral, String grade,
            String useremail) {
        this.userid = userid;
        this.username = username;
        this.userpwd = userpwd;
        this.userques = userques;
        this.userans = userans;
        this.integral = integral;
        this.grade = grade;
        this.useremail = useremail;
    }

    public Userinfo(Integer userid, String username, String userpwd,
            String userques, String userans, Integer integral, String grade,
            String useremail, Byte sex) {
        this.userid = userid;
        this.username = username;
        this.userpwd = userpwd;
        this.userques = userques;
        this.userans = userans;
        this.integral = integral;
        this.grade = grade;
        this.useremail = useremail;
        this.sex = sex;
    }

    public Integer getUserid() {
        return this.userid;
    }

    public void setUserid(Integer userid) {
        this.userid = userid;
    }
```

```
public String getUsername() {
    return this.username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getUserpwd() {
    return this.userpwd;
}

public void setUserpwd(String userpwd) {
    this.userpwd = userpwd;
}

public String getUserques() {
    return this.userques;
}

public void setUserques(String userques) {
    this.userques = userques;
}

public String getUserans() {
    return this.userans;
}

public void setUserans(String userans) {
    this.userans = userans;
}

public Integer getIntegral() {
    return this.integral;
}

public void setIntegral(Integer integral) {
    this.integral = integral;
}

public String getGrade() {
```

```
        return this.grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public String getUseremail() {
        return this.useremail;
    }

    public void setUseremail(String useremail) {
        this.useremail = useremail;
    }

    public Byte getSex() {
        return this.sex;
    }

    public void setSex(Byte sex) {
        this.sex = sex;
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="com.pojo.Userinfo" table="userinfo" catalog="mybbs">
      <id name="userid" type="java.lang.Integer">
         <column name="userid" />
         <generator class="assigned" />
      </id>
      <property name="username" type="java.lang.String">
         <column name="username" length="20" not-null="true" />
      </property>
      <property name="userpwd" type="java.lang.String">
         <column name="userpwd" length="20" not-null="true" />
      </property>
      <property name="userques" type="java.lang.String">
         <column name="userques" length="50" not-null="true" />
```

```
        </property>
        <property name="userans" type="java.lang.String">
            <column name="userans" length="50" not-null="true" />
        </property>
        <property name="integral" type="java.lang.Integer">
            <column name="integral" not-null="true" />
        </property>
        <property name="grade" type="java.lang.String">
            <column name="grade" length="20" not-null="true" />
        </property>
        <property name="useremail" type="java.lang.String">
            <column name="useremail" length="20" not-null="true" />
        </property>
        <property name="sex" type="java.lang.Byte">
            <column name="sex" />
        </property>
    </class>
</hibernate-mapping>
```

```
package com.dao;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.LockMode;
import org.springframework.context.ApplicationContext;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import com.pojo.Userinfo;
public class UserinfoDAO extends HibernateDaoSupport {
    private static final Log log = LogFactory.getLog(UserinfoDAO.class);

    public static final String USERNAME = "username";
    public static final String USERPWD = "userpwd";
    public static final String USERQUES = "userques";
    public static final String USERANS = "userans";
    public static final String INTEGRAL = "integral";
    public static final String GRADE = "grade";
    public static final String USEREMAIL = "useremail";
    public static final String SEX = "sex";

    protected void initDao() {

    }
```

```
public void save(Userinfo transientInstance) {
    log.debug("saving Userinfo instance");
    try {
        getHibernateTemplate().save(transientInstance);
        log.debug("save successful");
    } catch (RuntimeException re) {
        log.error("save failed", re);
        throw re;
    }
}

public void delete(Userinfo persistentInstance) {
    log.debug("deleting Userinfo instance");
    try {
        getHibernateTemplate().delete(persistentInstance);
        log.debug("delete successful");
    } catch (RuntimeException re) {
        log.error("delete failed", re);
        throw re;
    }
}

public Userinfo findById(java.lang.Integer id) {
    log.debug("getting Userinfo instance with id: " + id);
    try {
        Userinfo instance = (Userinfo) getHibernateTemplate().get(
                "com.pojo.Userinfo", id);
        return instance;
    } catch (RuntimeException re) {
        log.error("get failed", re);
        throw re;
    }
}

public List findByExample(Userinfo instance) {
    log.debug("finding Userinfo instance by example");
    try {
        List results = getHibernateTemplate().findByExample(instance);
        log.debug("find by example successful, result size: "
                + results.size());
        return results;
```

```
        } catch (RuntimeException re) {
            log.error("find by example failed", re);
            throw re;
        }
    }


    public List findByProperty(String propertyName, Object value) {
        log.debug("finding Userinfo instance with property: " + propertyName
                + ", value: " + value);
        try {
            String queryString = "from Userinfo as model where model."
                    + propertyName + "= ?";
            return getHibernateTemplate().find(queryString, value);
        } catch (RuntimeException re) {
            log.error("find by property name failed", re);
            throw re;
        }
    }

    public List findByUsername(Object username) {
        return findByProperty(USERNAME, username);
    }

    public List findByUserpwd(Object userpwd) {
        return findByProperty(USERPWD, userpwd);
    }

    public List findByUserques(Object userques) {
        return findByProperty(USERQUES, userques);
    }

    public List findByUserans(Object userans) {
        return findByProperty(USERANS, userans);
    }

    public List findByIntegral(Object integral) {
        return findByProperty(INTEGRAL, integral);
    }

    public List findByGrade(Object grade) {
        return findByProperty(GRADE, grade);
```

```
    }

    public List findByUseremail(Object useremail) {
        return findByProperty(USEREMAIL, useremail);
    }

    public List findBySex(Object sex) {
        return findByProperty(SEX, sex);
    }

    public List findAll() {
        log.debug("finding all Userinfo instances");
        try {
            String queryString = "from Userinfo";
            return getHibernateTemplate().find(queryString);
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }

    public Userinfo merge(Userinfo detachedInstance) {
        log.debug("merging Userinfo instance");
        try {
            Userinfo result = (Userinfo) getHibernateTemplate().merge(
                    detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
        }
    }

    public void attachDirty(Userinfo instance) {
        log.debug("attaching dirty Userinfo instance");
        try {
            getHibernateTemplate().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
```

```
        }
    }

    public void attachClean(Userinfo instance) {
        log.debug("attaching clean Userinfo instance");
        try {
            getHibernateTemplate().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }
    //判断登录
    public boolean verify(Userinfo user)
    {
        String hql = "from Userinfo as u where u.username=? and u.userpwd=?";
        List list = this.getHibernateTemplate().find(hql,
                        new String[]{user.getUsername(),user.getUserpwd()});
        if(list.size() > 0)
        {
            return true;
        }
        return false;
    }

    public  static  UserinfoDAO  getFromApplicationContext(ApplicationContext
ctx) {
        return (UserinfoDAO) ctx.getBean("UserinfoDAO");
    }
}
```

14. 新建 com.service 包，在包中建立 UserinfoService.java 类，代码如下：

```
package com.service;
import com.pojo.Userinfo;
import com.dao.UserinfoDAO;
public class UserinfoService {
    //依赖注入 UserinfoDAO
    private UserinfoDAO userinfoDAO;

    public UserinfoDAO getUserinfoDAO() {
        return userinfoDAO;
```

```
    }

    public void setUserinfoDAO(UserinfoDAO userinfoDAO) {
        this.userinfoDAO = userinfoDAO;
    }
    //判断登录
    public boolean Login(Userinfo user)
    {
        return userinfoDAO.verify(user);
    }
}
```

15. 新建 com.struts 包，在包中建立 LoginAction.java 和 LoginActionForm.java 类，代码如下：

```
package com.struts;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.pojo.Userinfo;
import com.service.UserinfoService;

public class LoginAction extends Action {
    //依赖注入 UserinfoService
    private UserinfoService userinfoService;

    public UserinfoService getUserinfoService() {
        return userinfoService;
    }

    public void setUserinfoService(UserinfoService userinfoService) {
        this.userinfoService = userinfoService;
    }

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        LoginActionForm LoginActionForm = (LoginActionForm) form;// TODO
Auto-generated method stub
        //得到 session 对象
```

```
        HttpSession session=request.getSession();
        //判断验证码

    if(!session.getAttribute("rand").toString().equals(LoginActionForm.getR
and()))
        {
            //页面跳转
            return mapping.findForward("self");
        }
        Userinfo user=new Userinfo();
        user.setUsername(LoginActionForm.getName());
        user.setUserpwd(LoginActionForm.getPassword());
        //如果用户名和密码输入正确登录成功
        if(userinfoService.Login(user))
        {
            return mapping.findForward("success");
        }
        //如果输入错误提示错误
        else
        {
            return mapping.findForward("error");
        }

    }
}
```

```
package com.struts;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
public class LoginActionForm extends ActionForm {

    //密码
    private String password;
    //用户名
    private String name;
    //验证码
    private String rand;
    public String getRand() {
        return rand;
    }
```

```
    public void setRand(String rand) {
        this.rand = rand;
    }
    public ActionErrors validate(ActionMapping mapping,
            HttpServletRequest request) {

        return null;
    }
    public void reset(ActionMapping mapping, HttpServletRequest request) {

    }
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

16. struts-config.xml 配置文件信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration                                                    1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans >
    <form-bean name="LoginActionForm" type="com.struts.LoginActionForm" />
  </form-beans>
  <global-exceptions />
  <global-forwards />
  <action-mappings >
```

```
    <action
      attribute="LoginActionForm"
      name="LoginActionForm"
      path="/LoginAction"
      scope="request"
      >
       <forward name="self" path="/jsp/login.jsp"></forward>
        <forward name="success" path="/jsp/success.jsp"></forward>
        <forward name="error" path="/jsp/error.jsp"></forward>
      </action>
    </action-mappings>
    <controller
processorClass="org.springframework.web.struts.DelegatingRequestProcessor">
</controller>
 <message-resources  parameter="com.yourcompany.struts.ApplicationResources"
/>
    <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
                        value="/WEB-INF/applicationContext.xml" />
    </plug-in>
</struts-config>
```

17. applicationContext.xml 配置文件信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">


    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <property name="url"
            value="jdbc:mysql://localhost:3306/mybbs">
        </property>
        <property name="username" value="root"></property>
        <property name="password" value="sa"></property>
    </bean>
```

```
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <property name="mappingResources">
            <list>
                <value>com/pojo/Userinfo.hbm.xml</value></list>
        </property>
</bean>
    <bean id="userinfoDAO" class="com.dao.UserinfoDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
    <bean id="userinfoService" class="com.service.UserinfoService"
        abstract="false" lazy-init="default" autowire="default"
        dependency-check="default">
        <property name="userinfoDAO">
            <ref bean="userinfoDAO" />
        </property>
    </bean>
    <bean name="/LoginAction" class="com.struts.LoginAction" abstract="false"
        lazy-init="default" autowire="default" dependency-check="default">
        <property name="userinfoService">
            <ref bean="userinfoService" />
        </property>
    </bean>
</beans>
```

18. 本程序涉及到的四个 Jsp 页面前面已经给出，部署程序，在地址栏里输入 http://localhost:8080/SSHLogin/jsp/login.jsp 打开登录页面。

图 4-15 登录页面

19. 输入正确的用户名、密码和验证码后点击“提交”，提示“登录成功”。

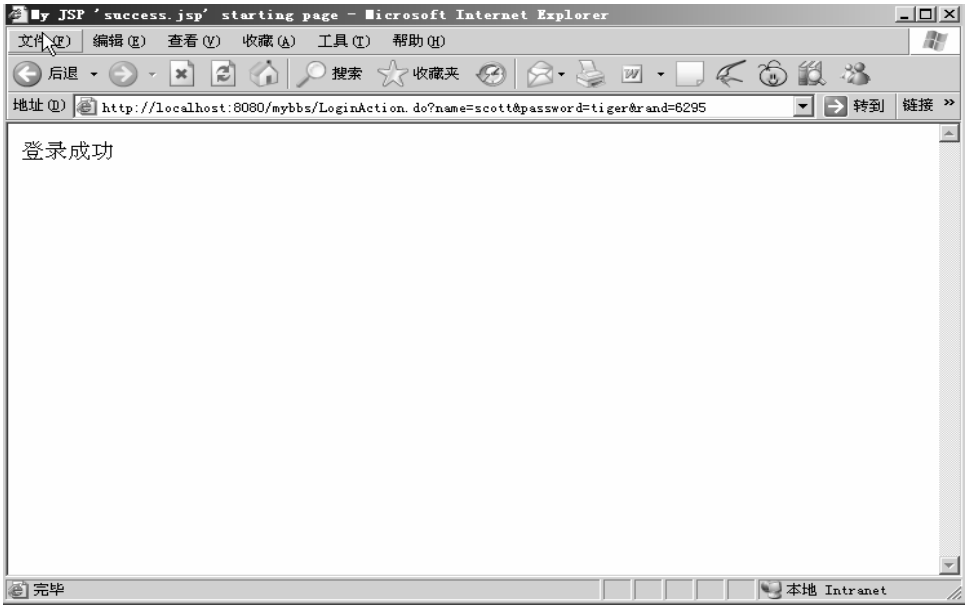


图 4-16 提示登录成功

18. 验证码、用户名或密码输入错误将在登录页面提示用户。

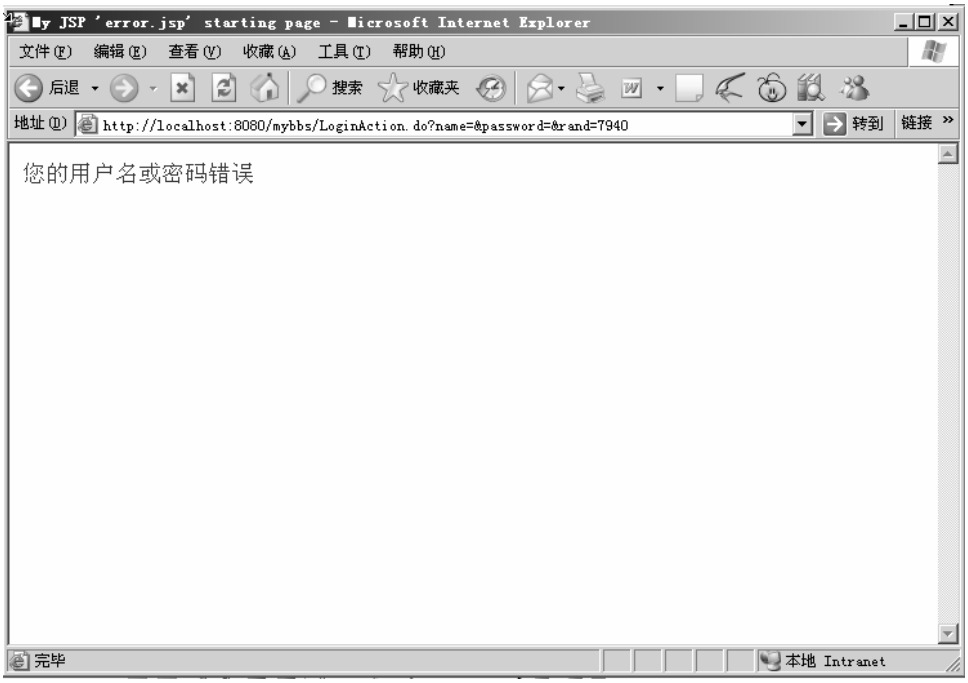


图 4-17 提示用户名密码错误

4. 相关理论知识

4.1 SSH 简介

典型的 J2EE 三层结构，分为表现层、中间层（业务逻辑层）和数据服务层。三层体系将业务规则、数据访问及合法性校验等工作放在中间层处理。客户端不直接与数据库交互，而是通过组件与中间层建立连接，再由中间层与数据库交互。 以前，经常提到的 MVC，其实就是这里所说的中间层的三个组成部分。

表现层采用传统的 JSP 技术，自 1999 年问世以来，经过多年的发展，以广泛的应用和稳定的表现，为其作为表现层技术打下了坚实的基础。

中间层采用的是流行的 Spring+Hibernate，为了将控制层与业务逻辑层分离，又细分为以下几种：

Web 层，就是 MVC 模式里面的“C”（controller），负责控制业务逻辑层与表现层的交互，调用业务逻辑层，并将业务数据返回给表现层作组织表现。SSH 中的“C”采用的是 Struts。

Service 层（就是业务逻辑层），负责实现业务逻辑。业务逻辑层以 DAO 层为基础，通过对 DAO 组件的正面模式包装，完成系统所要求的业务逻辑。

DAO 层，负责与持久化对象交互。该层封装了数据的增、删、查、改的操作。

PO，持久化对象。通过实体关系映射工具将关系型数据库的数据映射成对象，很方便地实现以面向对象方式操作数据库，该系统采用 Hibernate 作为 ORM 框架。

Spring 的作用贯穿了整个中间层，将 Web 层、Service 层、DAO 层及 PO 无缝整合，其

数据服务层用来存放数据。

4.2 Spring 整合 Hibernate

对 Hibernate，Spring 提供很多 IoC 的特性的支持，方便处理大部分典型的 Hibernate 整合的问题。所有的这些，都遵守 Spring 通用的事务和 DAO 异常体系。Spring 整合 Hibernate 后，使持久层的访问更加容易，Spring 管理 Hibernate 持久层有如下优势：

- 通用的资源管理：Spring 的 ApplicationContext 能管理 SessionFactory，使得配置值很容易被管理和修改。无须使用 Hibernate 的配置文件。
- 有效的 Session 管理：Spring 提供了有效，简单和安全的 Hibernate Session 处理。
- 方便的事务管理：Hibernate 的事务管理处理不好，会限制 Hibernate 的表现，而 Spring 的声明式事务管理粒度是方法级。
- 异常包装：Spring 能够包装 Hibernate 异常，把它们从 checked exception 变为 runtime exception。开发者可选择在恰当的层处理数据库的不可恢复异常，从而避免繁琐的 catch/throw 以及异常声明。

4.3 管理 SessionFactory

SessionFactory：单个数据库映射关系编译后的内存镜像。大部分情况下，一个 J2EE 应用对应一个数据库。Spring 通过 ApplicationContext 管理 SessionFactory，无须采用单独 Hiberate 应用必需的 hibernate.cfg.xml 文件。

SessionFactory 与数据库的连接，都由 Spring 的配置管理。实际的 J2EE 应用，通常使用数据源，数据源会采用依赖注入的方式，传给 Hibernate 的 SessionFactory。具体配置如下所示：

```
    <!--定义数据源, 该 bean 的 ID 为 dataSource-->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
         <!--  指定数据库驱动-->
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
         <!--  指定连接数据库的 URL-->
        <property name="url"
            value="jdbc:mysql://localhost:3306/mybbs">
        </property>
           <!--  root 为数据库的用户名-->
        <property name="username" value="root"></property>
          <!--  pass 为数据库密码-->
        <property name="password" value="sa"></property>
    </bean>
    <!--定义 Hibernate 的 SessionFactory-->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
        <!--  依赖注入数据源，注入正是上文定义的 dataSource-->
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <!--定义 Hibernate 的 SessionFactory 的属性 -->
        <property name="hibernateProperties">
            <props>
                  <!--  指定 Hibernate 的连接方言-->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <!--  mappingResouces 属性用来列出全部映射文件-->
        <property name="mappingResources">
            <list>
                <value>com/pojo/Userinfo.hbm.xml</value></list>
        </property>
    </bean>
```

SessionFactory 由 ApplicationContext 管理，会随着应用启动时候自动加载。SessionFactory 可以被处于 ApplicaionContext 管理的任意一个 bean 引用，比如 DAO。Hibernate 的数据库访问需要在 Session 管理下，而 SessionFactory 是 Session 的工厂。Spring 采用依赖注入为 DAO 对象注入 SessionFactory 的引用。

Spring 更提供 Hibernate 的简化访问方式，Spring 采用模板设计模式，提供 Hibernate 访问与其他持久层和访问保持一致。如果需要使用容器管理的数据源，则无须提供数据驱动等信息，只需要提供数据源的 JNDI 即可。对上文的 SessionFactory 无须任何修改，只需将 dataSource 的配置替换成 JNDI 数据源，将原有的 dataSource Bean 替换成如下所示：

```
<!--  此处配置 JNDI 数据源-->
<bean                                                    id="myDataSource"
class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName">
                <!--  指定数据源的 JNDI-->
                <value>java:comp/env/jdbc/myds</value>
        </property>
</bean>
```

4.4 Spring 对 Hibernate 的简化

Spring 对 Hibernate 的简化主要有如下几个方面：

1. 基于依赖注入的 SessionFactory 管理机制。SessionFactory 是执行持久化操作的核心组件。传统 Hibernate 应用中，SessionFactory 必须手动创建；通过依赖注入，代码无须关心 Ses

sionFactory，SessionFactory 的创建，维护由 BeanFactory 负责管理。

2. 更优秀的 Session 管理机制。Spring 提供“每事务一次 Session”的机制，该机制能大大提高系统性能，而且 Spring 对 Session 的管理是透明的，无须在代码中操作 Session。

3. 统一的事务管理。无论是编程式事务，还是声明式事务，Spring 都提供一致的编程模型，无须繁琐的开始事务，显式提交、回滚。如果使用声明式事务管理，事务管理逻辑与代码分离，事务可在全局事务和局部事务之间切换。

4. 统一的异常处理机制。不再强制开发者在持久层捕捉异常，持久层异常被包装成 DataAccessException 异常的子类，开发者可以自己决定在合适的层处理异常，将底层数据库异常包装成业务异常。

5. HibernateTemplate 支持类。HibernateTempate 能完成大量 Hibernate 持久层操作，这些操作大多只需一行代码，非常简洁。

4.5 使用 HibernateTemplate

HibernateTemplate 提供持久层访问模板化，使用 HibernateTemplate 无须实现特定接口，它只需要提供一个 SessionFactory 的引用，就可执行持久化操作。SessionFactory 对象可通过构造参数传入，或通过设值方式传入。如下：

```
//获取 Spring 上下文
ApplicationContext ctx =
             new ClassPathXmlApplicationContext("applicationContext.xml");
//通过上下文获得 SessionFactory
SessionFactory sessionFactory =
                            (SessionFactory) ctx.getBean(“sessionFactory”);
```

然后创建 HibernateTemplate 实例。HibernateTemplate 提供如下三个构造函数

- HibernateTemplate()
- HibernateTemplate(org.hibernate.SessionFactory sessionFactory)
- HibernateTemplate(org.hibernate.SessionFactory sessionFactory, boolean allowCreate)

第一个构造函数，构造一个默认的 HibernateTemplate 实例，因此，使用 HibernateTemplate 实例之前，还必须使用方法 setSessionFactory(SessionFactory sessionFactory)来为 HibernateTemplate 传入 SessionFactory 的引用。

第二个构造函数，在构造时已经传入 SessionFactory 引用。

第三个构造函数，其 boolean 型参数表明：如果当前线程已经存在一个非事务性的 Session，是否直接返回此非事务性的 Session。

对于在 Web 应用，通常启动时自动加载 ApplicationContext，SessionFactory 和 DAO 对象都处在 Spring 上下文管理下，因此无须在代码中显式设置，可采用依赖注入解耦 SessionFactory 和 DAO，依赖关系通过配置文件来设置，如下所示：

```
<?xml version="1.0" encoding="gb2312"?>
<!--  Spring 配置文件的 DTD 定义-->
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<!--  Spring 配置文件的根元素是 beans-->
     <!--定义数据源, 该 bean 的 ID 为 dataSource-->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
         <!--  指定数据库驱动-->
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
         <!--  指定连接数据库的 URL-->
        <property name="url"
            value="jdbc:mysql://localhost:3306/mybbs">
        </property>
           <!--  root 为数据库的用户名-->
        <property name="username" value="root"></property>
          <!--  pass 为数据库密码-->
        <property name="password" value="sa"></property>
    </bean>
    <!--定义 Hibernate 的 SessionFactory-->
    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
         <!--  依赖注入数据源，注入正是上文定义的 dataSource-->
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <!--定义 Hibernate 的 SessionFactory 的属性 -->
        <property name="hibernateProperties">
            <props>
                  <!--  指定 Hibernate 的连接方言-->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <!--  mappingResouces 属性用来列出全部映射文件-->
        <property name="mappingResources">
            <list>
                <value>com/pojo/Userinfo.hbm.xml</value></list>
        </property>
    </bean>
```

```
    <!--  配置 userinfoDAO 持久化类的 DAO bean-->
    <bean id="userinfoDAO" class="com.dao.UserinfoDAO">
         <!--  采用依赖注入来传入 SessionFactory 的引用-->
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
</beans>
```

DAO 实现类中，可采用更简单的方式来取得 HibernateTemplate 的实例。代码如下：

```
public class UserinfoDAO{
    //以私有的成员变量来保存 SessionFactory。
private SessionFactory sessionFactory;
         //设值注入 SessionFactory 必需的 setter 方法
    public void setSessionFactory(SessionFactory sessionFactory)
{
           this.sessionFactory = sessionFactory;
    }

    public List loadUserByName(final String name)
{
           HibernateTemplate hibernateTemplate =
                    new HibernateTemplate(this.sessionFactory);
                   //此处采用 HibernateTemplate 完成数据库访问
    }
}
```

4.5.1 HibernateTemplate 用法

HibernateTemplate 提供非常多的常用方法来完成基本的操作，比如通常的增加、删除、修改、查询等操作，Spring 2.0 更增加对命名 SQL 查询的支持，也增加对分页的支持。大部分情况下，使用 Hibernate 的常规用法，就可完成大多数 DAO 对象的 CRUD 操作。下面是 HibernateTemplate 的常用方法简介：

void delete(Object entity)：删除指定持久化实例

deleteAll(Collection entities)：删除集合内全部持久化类实例

find(String queryString)：根据 HQL 查询字符串来返回实例集合

findByNamedQuery(String queryName)：根据命名查询返回实例集合

get(Class entityClass, Serializable id)：根据主键加载特定持久化类的实例

save(Object entity)：保存新的实例

saveOrUpdate(Object entity)：根据实例状态，选择保存或者更新

update(Object entity)：更新实例的状态，要求 entity 是持久状态

setMaxResults(int maxResults)：设置分页的大小

下面是一个完整 DAO 类的源代码：

```
public class UserinfoDAO {
    //采用 log4j 来完成调试时的日志功能
private static Log log = LogFactory.getLog(UserinfoDAO.class);
    //以私有的成员变量来保存 SessionFactory。
private SessionFactory sessionFactory;
//以私有变量的方式保存 HibernateTemplate
private HibernateTemplate hibernateTemplate = null;
         //设值注入 SessionFactory 必需的 setter 方法
    public void setSessionFactory(SessionFactory sessionFactory)
{
           this.sessionFactory = sessionFactory;
    }
         //初始化本 DAO 所需的 HibernateTemplate
public HIbernateTemplate getHibernateTemplate()
{
         //首先，检查原来的 hibernateTemplate 实例是否还存在
         if ( hibernateTemplate == null)
         {
                   //如果不存在，新建一个 HibernateTemplate 实例
                   hibernateTemplate =
                                      new HibernateTemplate(sessionFactory);
         }
         return hibernateTemplate;
}
         //返回全部用户的实例
    public List getUserinfos()
         {
                   //通过 HibernateTemplate 的 find 方法返回 Userinfo 的全部实例
           return getHibernateTemplate().find("from Userinfo");
    }
         /**
          * 根据主键返回特定实例
          * @ return 特定主键对应的 Userinfo 实例
          * @ param 主键值
    public Userinfo getUserinfo (int userid)
         {
           return (Userinfo)getHibernateTemplate().get(Userinfo.class, new
Integer(userid));
```

```
    }
         /**
          * @ Userinfo 需要保存的 Userinfo 实例
          */
    public void saveUserinfo(Userinfo userinfo)
         {
                   getHibernateTemplate().saveOrUpdate(userinfo);
    }
         /**
          * @ param userid 需要删除 Userinfo 实例的主键
          * /
    public void removeUserinfo (int userid)
         {
                   //先加载特定实例
              Object  u  =  getHibernateTemplate().load(Userinfo.class,  new
Integer(userid));
                   //删除特定实例
           getHibernateTemplate().delete(u);
    }
}
```

4.6 Hibernate 的 DAO 实现

DAO 对象是模块化的数据库访问组件，DAO 对象通常包括：对持久化类的基本 CRUD 操作（插入、查询、更新、删除）操作。Spring 对 Hibernate 的 DAO 实现提供了良好的支持。

Spring 为 Hibernate 的 DAO 提供工具类：HibernateDaoSupport。该类主要提供如下两个方法，方便 DAO 的实现：

- public final HibernateTemplate getHibernateTemplate()
- public final void setSessionFactory(SessionFactory sessionFactory)

其中，setSessionFactory 方法用来接收 Spring 的 ApplicationContext 的依赖注入，可接收配置在 Spring 的 SessionFactory 实例，getHibernateTemplate 方法则用来根据刚才的 SessionFactory 产生 Session，最后生成 HibernateTemplate 来完成数据库访问。

典型的继承 HibernateDaoSupport 的 DAO 实现的代码如下：

```
public class UserinfoDAO extends HibernateDaoSupport
    //采用 log4j 来完成调试时的日志功能
private static Log log = LogFactory.getLog(UserinfoDAO.class);
    //返回全部用户的实例
    public List getUserinfos()
    {
           //通过 HibernateTemplate 的 find 方法返回 Userinfo 的全部实例
           return getHibernateTemplate().find("from Userinfo");
```

```
    }
       /**
          * 根据主键返回特定实例
          * @ return 特定主键对应的 Userinfo 实例
          * @ param 主键值
    public News getUserinfo (int userid)
   {
           return (Userinfo)getHibernateTemplate().get(Userinfo.class, new
Integer(userid));
    }
         /**
          * @ Userinfo 需要保存的 Userinfo 实例
          */
    public void saveUserinfo (Userinfo user)
         {
                   getHibernateTemplate().saveOrUpdate(user);
         }
         /**
          * @ param userid 需要删除 Userinfo 实例的主键
          * /
    public void removeUserinfo (int userid)
         {
           //先加载特定实例
            Object  u  =  getHibernateTemplate().load(Userinfo.class,  new
Integer(userid));
           //删除特定实例
           getHibernateTemplate().delete(p);
    }
}
```

可以与前面的 UserinfoDAO 对比，会发现代码量大大减少。事实上，DAO 的实现依然借助于 HibernateTemplate 的模板访问方式，只是，HibernateDaoSupport 将依赖注入 SessionFactory 的工作已经完成，获取 HibernateTemplate 的工作也已完成。该 DAO 的配置必须依赖于 SessionFactory，具体的配置如下：

```
<?xml version="1.0" encoding="gb2312"?>
<!--  Spring 配置文件的 DTD 定义-->
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<!--  Spring 配置文件的根元素是 beans-->
     <!--定义数据源, 该 bean 的 ID 为 dataSource-->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
```

```
        <!--  指定数据库驱动-->
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <!--  指定连接数据库的 URL-->
        <property name="url"
            value="jdbc:mysql://localhost:3306/mybbs">
        </property>
          <!--  root 为数据库的用户名-->
        <property name="username" value="root"></property>
         <!--  pass 为数据库密码-->
        <property name="password" value="sa"></property>
    </bean>
    <!--定义 Hibernate 的 SessionFactory-->
    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <!--  依赖注入数据源，注入正是上文定义的 dataSource-->
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <!--定义 Hibernate 的 SessionFactory 的属性 -->
        <property name="hibernateProperties">
            <props>
                  <!--  指定 Hibernate 的连接方言-->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <!--  mappingResouces 属性用来列出全部映射文件-->
        <property name="mappingResources">
            <list>
                <value>com/pojo/Userinfo.hbm.xml</value>
</list>
        </property>
    </bean>
    <!--  配置 userinfoDAO 持久化类的 DAO bean-->
    <bean id="userinfoDAO" class="com.dao.UserinfoDAO">
        <!--  采用依赖注入来传入 SessionFactory 的引用-->
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
```

```
        </property>
    </bean>
</beans>
```

程序中可以通过显式的编码来获得 userinfoDAO bean，然后执行 CRUD 操作。也可通过依赖注入，将 personDAO 的实例注入其他 bean 属性，再执行 CRUD 操作。

在继承 HibrnateDaoSupport 的 DAO 实现里，Hibernate Session 的管理完全不需要 Hibernate 代码打开，而由 Spring 来管理。Spring 会根据实际的操作，采用“每次事务打开一次 session”的策略，自动提高数据库访问的性能。

4.7 声明式事务管理

Hibernate 建议所有的数据库访问都应放在事务内进行，即使只进行只读操作。事务又应该尽可能的短，长事务会导致系统长时间无法释放，因而降低系统并发的负载。Spring 同时支持编程式事务和声明式事务。尽量考虑使用声明式事务，声明式事务管理可分离业务逻辑和事务管理逻辑，具备良好的适应性。

关于声明式事务管理的配置方式，通常有如下三种

- 使用 TransactionProxyFactoryBean 为目标 bean 生成事务代理的配置。此方式是最传统，配置文件最臃肿、难以阅读的方式
- 采用 bean 继承的事务代理配置方式，比较简洁，但依然是增量式配置。
- 使用 BeanNameAutoProxyCreator，根据 bean name 自动生成事务代理的方式。

建议采用第三种配置方式，详细的配置代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <!--定义数据源, 该 bean 的 ID 为 dataSource-->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <!--  指定数据库驱动-->
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <!--  指定连接数据库的 URL-->
        <property name="url"
            value="jdbc:mysql://localhost:3306/mybbs">
        </property>
        <!--  root 为数据库的用户名-->
        <property name="username" value="root"></property>
```

```
        <!--  pass 为数据库密码-->
        <property name="password" value="sa"></property>
    </bean>
    <!--定义 Hibernate 的 SessionFactory-->
    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <!--  依赖注入数据源，注入正是上文定义的 dataSource-->
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <!--定义 Hibernate 的 SessionFactory 的属性 -->
        <property name="hibernateProperties">
            <props>
                <!--  指定 Hibernate 的连接方言-->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <!--  mappingResouces 属性用来列出全部映射文件-->
        <property name="mappingResources">
            <list>
                <value>com/pojo/Userinfo.hbm.xml</value>
            </list>
        </property>
    </bean>
    <bean id="transactionManager"

    class="org.springframework.orm.hibernate3.HibernateTransactionManager"
>
        <!--     HibernateTransactionManager     bean  需 要 依 赖 注 入 一 个
SessionFactory bean 的引用-->
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <!--   配置事务拦截器-->
    <bean id="transactionInterceptor"

    class="org.springframework.transaction.interceptor.TransactionIntercep
```

```
tor">
        <!--   事务拦截器 bean 需要依赖注入一个事务管理器 -->
        <property name="transactionManager" ref="transactionManager" />
        <property name="transactionAttributes">
            <!--   下面定义事务传播属性-->
            <props>
                <prop key="insert*">PROPAGATION_REQUIRED</prop>
                <prop key="find*">PROPAGATION_REQUIRED,readOnly</prop>
                <prop key="*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>
    <!-- 定义 BeanNameAutoProxyCreator，该 bean 是个 bean 后处理器，无需被引用，因此
没有 id 属性，这个 bean 后处理器，根据事务拦截器为目标 bean 自动创建事务代理指定对满足哪些
bean name 的 bean 自动生成业务代理 -->
    <bean

    class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCr
eator">
        <property name="beanNames">
            <!--   下面是所有需要自动创建事务代理的 bean-->
            <list>
                <value>userinfoService</value>
            </list>
                <!--   此处可增加其他需要自动创建事务代理的 bean-->
        </property>
        <!--   下面定义 BeanNameAutoProxyCreator 所需的事务拦截器-->
        <property name="interceptorNames">
            <list>
                <value>transactionInterceptor</value>
                <!-- 此处可增加其他新的 Interceptor -->
            </list>
        </property>
    </bean>


    <!--  配置 userinfoDAO 持久化类的 DAO bean-->
    <bean id="userinfoDAO" class="com.dao.UserinfoDAO">
        <!--  采用依赖注入来传入 SessionFactory 的引用-->
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
```

```
    </bean>
    <bean id="userinfoService" class="com.service.UserinfoService"
        abstract="false" lazy-init="default" autowire="default"
        dependency-check="default">
        <property name="userinfoDAO">
            <ref bean="userinfoDAO" />
        </property>
    </bean>
</beans>
```

TranscationInterceptor 是一个事务拦截器 bean，需要传入一个 TransactionManager 的引用。配置中使用 Spring 依赖注入该属性，事务拦截器的事务属性通过 transactionAttributes 来指定，该属性有 props 子元素，配置文件中定义了三个事务传播规则：

所有以 insert 开始的方法，采用 PROPAGATION_REQUIRED 的事务传播规则。程序抛出 MyException 异常及其子异常时，自动回滚事务。所有以 find 开头的方法，采用 PROPAGATION_REQUIRED 事务传播规则，并且只读。其他方法，则采用 PROPAGATION_REQUIRED 的事务传播规则。

BeanNameAutoProxyCreator 是个根据 bean 名生成自动代理的代理创建器，该 bean 通常需要接受两个参数。第一个是 beanNames 属性，该属性用来设置哪些 bean 需要自动生成代理。另一个属性是 interceptorNames，该属性则指定事务拦截器，自动创建事务代理时，系统会根据这些事务拦截器的属性来生成对应的事务代理。

常见的事务传播规则有如下几种：

1. PROPAGATION_MANDATORY:：求调用该方法的线程必须处于事务环境中，否则抛出异常。

2. PROPAGATION_NESTED：如果执行该方法的线程以处于事务环境下，依然启动新的事务，方法在嵌套的事务里执行。如果执行方法的线程并未处于事务中，也启动新的事务，然后执行该方法，此时与 PROPAGATION_REQUIRED 相同。

3. PROPAGATION_NEVER：不允许调用该方法的线程处于事务环境下，如果调用该方法的线程处于事务环境下，则抛出异常。

4. PROPAGATION_NOT_SUPPORTED：如果调用该方法的线程处在事务中，则先暂停当前事务，然后执行方法。

5. PROPAGATION_REQUIRED：要求在事务环境中执行该方法，如果当前执行线程已处于事务中，则直接调用；如果当前执行线程不处于事务中，则启动新的事务后执行该方法。

6. PROPAGATION_REQUIRED_NEW：该方法要求有一个在新的事务环境中运行，如果当前执行线程已处在事务中，先暂停事务，启动新事务后执行方法；如果当前线程不处在事务中，则启动新的事务后执行方法。

7. PROPAGATION_SUPPORTS：如果当前执行线程处于事务中，则使用当前事务，否则不使用事务。

4.8 整合 Struts

Spring 提供与 Struts 的无缝结合。Spring 与 Struts 的整合有如下两种策略：

- 让 Spring IoC 容器管理 Struts 的 Action，Struts 采用 ContextLoaderPlugin 创建 Spring 的 ApplicationContext 实例。
- 采用 Spring 的 ApplicationSupport 类的子类。该支持类具有 getWebApplicationContext()方法可以访问 Spring 的 ApplicationContext。

推荐使用 Spring IoC 容器管理 Struts Action 的方式。采用这种方式能充分利用 Spring 依赖注入的优势，而无须显示地获取 Spring 的 ApplicationContext 实例。让 Spring IoC 容器管理 Action 也有两种方式：

- 使用 DelegatingRequestProcessor
- 使用 DelegatingActionProxy

不管采用哪一种，都需要随应用启动时创建 ApplicationContext 实例。创建 ApplicationContext 实例采用 Spring 的 ContextLoaderPlugin 类，该类实现 org.apache.struts.action.Plugin 接口，Plugin 接口从 Struts1.1 开始才有，用于启动时加载某个模块。

ContextLoaderPlugin 默认加载的配置文件为 servletName-servlet.xml。其中 servletName 是 Struts 的 ActionServlet 对应的 Servlet 名。例如 web.xml 中进行如下定义:

```
<servlet>
   <servlet-name>actionServlet</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>
```

ContextLoaderPlugin 默认加载\WEB-INF\actionServlet-servlet.xml，将该文件作为 Spring 的配置文件。因此如果 Spring 的配置文件只有一份，且文件名为 actionServlet-servlet.xml，则只需在 Struts 配置文件的底部增加如下一行：

```
<plug-in className="org.springframeword.web.struts.ContextLoaderPlugIn" />
```

如果有多个配置文件，或者配置文件的文件名不符合规则，则可采用 contextConfigLocation 属性载入。同样，配置文件之间以“,”隔开。下面是载入多个配置文件的配置片段：

```
<plug-in className="org.springframeword.web.struts.ContextLoaderPlugIn" >
<set-property                                property="contextConfigLocation"
value="\WEB-INF\action-servlet.xml,\WEB-INF\applicationContext.xml"
</plug-in>
```

创建 ApplicationContext 实例后，关键是如何将 ActionServlet 截获的请求转发给 Spring 管理的 bean。Spring IoC 容器不仅管理本身的业务 bean，还负责管理 Struts 的 Action。因此，需要让 ActionServlet 将请求不再转发给 struts-config.xml 配置的 action，而是转发给 ApplicationContext 里配置的 bean。完成这个转发也有两个策略：

- 采用 DelegatingRequestProcessor
- 采用 DelegatingActionProxy

4.8.1 使用 DelegatingRequestProcessor

查看 Struts 的源代码，看到 ActionServlet 调用 RequestProcessor 完成实际的转发。如果想将 Action 截获的请求转发给 ApplicationContext 的 bean，可以扩展 RequestProcessor 完成。

Spring 完成这种扩展。Spring 提供的 DelegatingRequestProcessor 继承 RequestProcessor。为了让 Struts 使用 DelegatingRequestProcessor，还需要在 struts-config.xml 文件中增加如下一行：

```
//使用 Spring 的 RequestProcessor 替换 Struts 原有的 RequestProcessor
<controller
processorClass=”org.springframework.web.struts.DelegatingRequestProcessor”
/>
```

完成这个设置后，Struts 会将截获到的用户请求转发到 Spring context 下的 bean，根据 bean 的 name 属性来匹配。而 Struts 中的 action 配置则无须配置 type 属性，即使配置了 type 属性也没有任何用处，即下面两行配置是完全相同的：

```
//配置 struts action 时，指定了实现类
<action path=”/login” type=”LoginAction”/>
//配置配置 struts action 时，没有指定类
<action path=”/login” />
```

下面一个简单的 struts-config.xml 文件，已经和 Spring 整合在了一起，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration                                                    1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
    <!-- 配置 formbean -->
  <form-beans >
    <form-bean name="LoginActionForm" type="LoginActionForm"></form-bean>
  </form-beans>
    <!-- 定义 action 部分 -->
  <action-mappings >
    <!-- 这里只有一个 action。而且没有指定该 action 的 type 元素 -->
    <action path="/Login" scope="request" name="LoginActionForm"></action>
  </action-mappings>
  <!-- 使用 DelegatingRequestProcessor 替换 RequestProcessor-->
  <controller
processorClass="org.springframework.web.struts.DelegatingRequestProcessor">
</controller>
  <message-resources parameter="com.yourcompany.struts.ApplicationResources"
```

```
/>
  <!-- 装载 spring 配置文件 -->
  <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property                        property="contextConfigLocation"
value="/WEB-INF/applicationContext.xml" />
  </plug-in>
</struts-config>
```

applicationContext.xml 应放置在 WEB-INF 目录下，其详细配置信息如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE        beans        PUBLIC        "-//SPRING//DTD        BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean name="/Login" class="LoginAction" abstract="false" scope=″prototype″
></bean>
</beans>
```

每次请求都应该启动新的 action 处理用户请求，因此应将 action bean 配置成 non-singleton 行为。

当我们访问 Login.do 的时候，DelegatingRequestProcessor 会将请求转发到 action。该 action 已经处于 IoC 容器管理之下，因此，可以方便的访问容器中的其他 bean。

4.8.2 使用 DelegatingActionProxy

使用 DelegatingRequestProcessor 简单方便，但有一个缺点：RequestProcessor 是 Struts 的一个扩展点，也许应用本身就需要扩展 RequestProcessor，而 DelegationRequestProcessor 已经占领了这个扩展点。此时有如下两个做法：

- 应用程序的 RequestProcessor 不再继承 Struts 的 RequestProcessor，改为继承 DelegatingRequestProcessor。
- 使用 DelegatingActionProxy。

前者常常有一些未知的风险，而后者是 Spring 推荐的整和策略。使用 DelegatingActionProxy 与使用 DelegatingRequestProcessor 目的只有一个，即将请求转发给 Spring 管理的 bean。

DelegationRequestProcessor 直接替换原有的 RequestProcessor，在请求转发给 action 之前，转发给 Spring 管理 bean；而 DelegatingActionProxy 则配置成 Struts 的 action，即所有的请求先被 ActionServlet 截获，请求被转发到相应的 action，而 action 实现类全都是 DelegatingActionProxy。DelegatingActionProxy 再将请求转发给 Spring 容器的 bean。

可以看出，DelegatingActionProxy 比使用 DelegationRequestProcessor 要晚一步转发到 Spring 的 context，但通过这种方式可以避免占用扩展点。

与使用 DelegationRequestProcessor 对比，使用 DelegatingActionProxy 仅需去掉 controller 配置元素，并将所有的 action 实现类改为 DelegatingActionProxy。详细配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration                                                    1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
    <!-- 配置 formbean -->
  <form-beans >
    <form-bean name="LoginActionForm" type="LoginActionForm"></form-bean>
  </form-beans>
    <!-- 定义 action 部分 -->
  <action-mappings >
    <!-- 这里只有一个 action。而且没有指定该 action 的 type 元素 -->
    <action path="/Login" scope="request" name="LoginActionForm"
            type="org.springframework.web.struts.DelegatingActionProxy">
</action>
  </action-mappings>
  <message-resources
parameter="com.yourcompany.struts.ApplicationResources" />
  <!-- 装载 spring 配置文件 -->
  <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
                        value="/WEB-INF/applicationContext.xml" />
  </plug-in>
</struts-config>
```

DelegatingActionProxy 接受 ActionServlet 转发过来的请求，然后转发给 ApplicationContext 管理的 bean，这是典型的链式处理。

4.8.3 使用 ActionSupport 代替 Action

还有一种方法可以用于 Spring 和 Struts 的整合，让 Action 在程序中手动获得 Application Context 实例。在这种整合策略下，Sturts 的 Action 不接受 IoC 容器管理，Action 的代码与 Spring API 部分耦合，造成代码污染。Actoin 中访问 ApplicationContext 有两种方法：

- 利用 WebApplicationContextUtils 工具类。
- 利用 ActionSupport 支持类。

WebApplicationContextUtils 可以通过 ServletContext 获得 Spring 容器实例。ActionSupport 类则提供一个简单的 getWebApplicationContext()方法，该方法用于获取 ApplicationContext 实例。Spring 扩展 Strus 的标准类，Spring 的 Action 在 Struts 的 Action 后加上 Support。Spring 的 Action 有如下 4 中：

- ActionSupport
- DispatchActionSupport

- LookUpDispatchActionSupport
- MappingDispatchActionSupport

下面给出利用 ActionSupport 的示例代码：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.springframework.web.struts.ActionSupport;
//新的业务控制器，继承 Spring 的 ActionSupport
public class LoginAction extends ActionSupport {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        LoginActionForm LoginActionForm = (LoginActionForm) form;
        return null;
    }
}
```

这种整合策略下，表现层的控制器组件不再接受 IoC 容器管理。因此没有控制器 context，我们应该修改配置文件将，applicationContext.xml 中的 action bean 删除，修改 action 配置，将 action 配置的 type 元素修改成实际的处理类。这种整合策略也有个好处：代码可读性强，对传统 Struts 应用开发的改变很小，容易使用。

5. 试验

按照上课的顺序依次练习，主要掌握：

1. 建立 SSHLogin 工程，并添加 struts，spring，hibernate 支持。（10 分钟）
2. 编写 login.jsp，success.jsp，error.jsp，image.jsp 页面。（20 分钟）
3. 利用工具生成 userinfo.java、userinfo.hbm.xml 和 userinfoDAO 类。（20 分钟）
4. 编写 userinfoService.java 类。（10 分钟）
5. 编写 LoginAction 和 LoginActionForm 类。（20 分钟）
6. 配置 struts-config.xml 和 application.xml 文件，并调试运行程序。（10 分钟）

6. 作业

整合 SSH，并实现登录功能。

专题五 Ajax

教学目标

理解 Ajax 工作原理

掌握 XmlHttpRequest 对象

使用 DWR 框架进行 Ajax 开发

案例一 AJAX 基础知识

1. 教学目标

1.1 了解 AJAX 及其 AJAX 优点

1.2 理解 AJAX 构成元素

1.3 掌握 XmlHttpRequest 对象

2. 工作任务

2.1 建立一个类似 Google Suggest 应用

2.2 建立一个用户注册时用户名验证应用

3. 相关实践知识

3.1 建立一个类似 Google Suggest 应用

1. 新建 Web 项目，项目名为“ajax”。如下图所示：

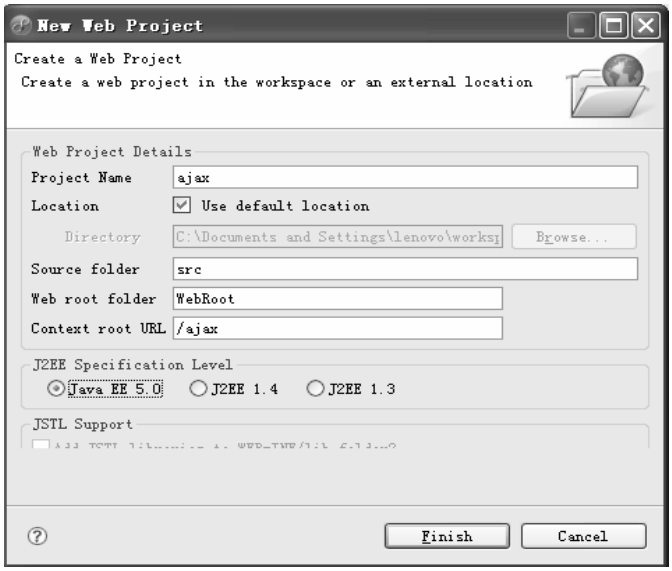


图 5-1 建立 ajax 工程

2. 在项目中建立名为 com.ajax.servlet 的包。
3. 在 com.ajax.servlet 包内建立名为 Suggest 的 servlet。代码如下：

```
public class Suggest extends HttpServlet {
    private ArrayList lib = new ArrayList();
```

```
    //初始化数据集合,可以在这个字库中添加更多词条
    public void init() throws ServletException {
        lib.add("a");
        lib.add("able");
        lib.add("access");
        lib.add("advance");
    }
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
response)
            throws ServletException, IOException {
        //设置生成文件的类型和编码格式
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        String output = "";
        //处理接收到的参数
        String key = request.getParameter("key");
        ArrayList matchList = getMatchString(key);
        if (!matchList.isEmpty()) {
            output += "<response>";
            for (int i = 0; i < matchList.size(); i++) {
                String match = matchList.get(i).toString();
                output += "<item>" + match + "</item>";
            }
            output += "</response>";
        }
        out.println(output);
        out.close();
    }

    public void doGet(HttpServletRequest request,
                                            HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }
    //取得所有匹配的字符串
    public ArrayList getMatchString(String key) {
        ArrayList result = new ArrayList();
        if (!lib.isEmpty()) {
            for (int i = 0; i < lib.size(); i++) {
                String str = lib.get(i).toString();
                if (str.startsWith(key))
```

```
                    result.add(str);
                }
            }
            return result;
        }
}
```

4. 在项目中建立 Suggest.jsp 文件，代码如下：

```
%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
    <head>
        <title>输入提示示例</title>
        <style>
TD {
    FONT-SIZE: 12px
}
</style>
        <script language="javascript">
      //创建 XMLHttpReques 对象
      function createXMLHttpRequest() {
        if (window.XMLHttpRequest) {
            //Mozilla 浏览器
            XMLHttpReq = new XMLHttpRequest();
          } else{
             //IE 浏览器
             if (window.ActiveXObject) {
               try {
                     XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                 }catch (e) {
                    try {
                          XMLHttpReq =
                                    new ActiveXObject("Microsoft.XMLHTTP");
                       }catch (e) { }
                 }
              }
          }
       }
      //处理服务器响应结果
      function handleResponse() {
       //判断对象状态
       if (XMLHttpReq.readyState == 4) {
          //信息已经成功返回，开始处理信息
```

```
        if (XMLHttpReq.status == 200) {
            clearTable();
            var out = "";
            var res = XMLHttpReq.responseXML;
            var items = res.getElementsByTagName("item");
            for(var i=0;i<items.length;i++)
            {
               addRow(items(i).firstChild.nodeValue);
            }
            setDivStyle();
         }
      }
    }
    //清除表格中的结果
    function clearTable()
    {
       var content = document.getElementById("content");
       while(content.childNodes.length>0)
       {
          content.removeChild(content.childNodes[0]);
       }
    }
    //向输入提示的表格中添加一行记录
    function addRow(item)
    {
       var content = document.getElementById("content");
       var row = document.createElement("tr");
       var cell = document.createElement("td");
       cell.appendChild(document.createTextNode(item));
       cell.onmouseover = function(){this.style.background="blue"};
       cell.onmouseout = function(){this.style.background="#f5f5f1"};
       cell.onclick = function(){
       document.getElementById("key").value=this.innerHTML;
       document.getElementById("suggest").style.visibility="hidden"};
       row.appendChild(cell);
       content.appendChild(row);
    }
   //发送客户端的请求
   function sendRequest(url) {
        createXMLHttpRequest();
        XMLHttpReq.open("GET", url, true);
        //指定响应函数
```

```
            XMLHttpReq.onreadystatechange = handleResponse;
            //发送请求
            XMLHttpReq.send(null);
        }
        //调用 AJAX 自动提示功能
        function suggest()
        {
            var key = document.getElementById("key").value;
            sendRequest("Suggest?key="+key);
        }

        //设置输入提示框的位置和风格
        function setDivStyle()
        {
            var suggest = document.getElementById("suggest");
            suggest.style.border = "black 1px solid";
            suggest.style.left = 62;
            suggest.style.top = 50;
            suggest.style.width = 150;
            suggest.style.backgroundColor = "#f5f5f1"
            document.getElementById("suggest").style.visibility="visible"
        }
    </script>
    </head>
    <body>
        <font size="1"> 输入提示示例(可以输入字母 a 开头的字符串进行测试)<br> 请输入：
<input type="text" id="key" name="key" onkeyup="suggest()" />
            <div id="suggest" style="position: absolute">
                <table>
                    <tbody id="content"></tbody>
                </table>
            </div> </font>
    </body>
</html>
```

5. 将项目部署到服务器，启动服务器，打开 IE 浏览器输入地址 http://localhost:8080/ajax/Suggest.jsp。显示内容如下：

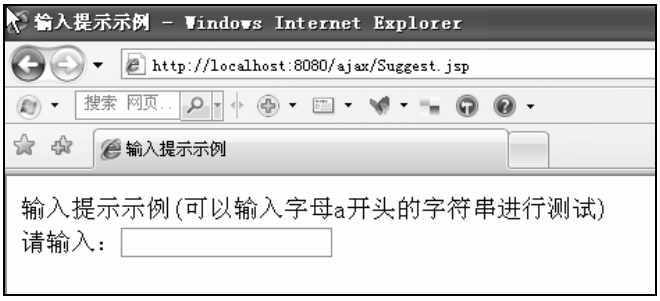


图 5-2 显示 Suggest.jsp 内容

6. 向文本框输入“a”，可以看到下图内容：

图 5-3 输入‘a’字母呈现内容

这样我们就完成了一个类似于 google suggest 的示例。

在上述的示例中我们将相关数据写在 servlet 中。在实际开发过程中，数据再存放到数据库中，下面进行注册时用户名是否存在验证示例。

3.2 建立用户注册时用户名验证应用

1. 使用 Hibernate 示例中的 mybbs 数据库中的 admin 表。admin 表 Sql 语句如下：

```
create table `mybbs`.`admin`(
       `adminid` int not null auto_increment,
      `adminname` varchar(20) default '' not null,
      `adminpwd` varchar(20) default '' not null,
       primary key (`adminid`)
   )TYPE=INNODB,
    default character set gbk;
```

同时向 admin 表中插入一条记录：

```
Insert into admin(adminname,adminpwd) values('admin','admin');
```

2. 将 mysql 的驱动程序 mysql-connector-java-5.0.4-bin.jar 复制到 WEB-INF/lib 目录下。

3. 使用上述示例的“ajax”工程。在“ajax”工程中建立 com.ajax.pojo，com.ajax.dao，com.ajax.util 三个包。工程结构如下图所示：

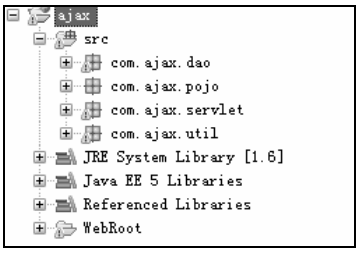


图 5-4 “ajax”工程目录

4. 在 com.ajax.util 包内建立 DBUtil 类，该类得到数据库连接及关闭连接。代码如下：

```
package com.ajax.util;
import java.sql.*;
public class DBUtil {
    private Connection conn;
    private PreparedStatement pstm;
    private ResultSet rs;
    public Connection getConn(){
     try{
         Class.forName("com.mysql.jdbc.Driver");

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/mybbs","root"
,"123456");

     }catch(Exception ex){
          ex.printStackTrace();
     }
     return conn;
    }
    public void closeConn(Connection conn,PreparedStatement pstm,ResultSet
rs){
     if(rs!=null){
         try {
                rs.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
     }
     if(pstm!=null){
```

```
        try {
               pstm.close();
           } catch (SQLException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
           }
    }
    if(conn!=null){
        try {
               conn.close();
           } catch (SQLException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
           }
    }
    }
}
```

5. 在 com.ajax.pojo 中建立 Admin 实体类。代码如下：

```
package com.ajax.pojo;
public class Admin {
    private String adminName;
    private String adminPwd;
    public String getAdminName() {
        return adminName;
    }
    public void setAdminName(String adminName) {
        this.adminName = adminName;
    }
    public String getAdminPwd() {
        return adminPwd;
    }
    public void setAdminPwd(String adminPwd) {
        this.adminPwd = adminPwd;
    }
}
```

6. 在 com.ajax.dao 中建立 AdminDAO 类，该类负责进行业务逻辑的编写。代码如下：

```
package com.ajax.dao;
import java.sql.*;
import com.ajax.pojo.Admin;
```

```
import com.ajax.util.DBUtil;
import java.util.*;
public class AdminDAO {
   private Connection conn;
   private PreparedStatement pstm;
   private ResultSet rs;
   private Admin admin;
   private DBUtil db;
   public ArrayList selectName(){
    db=new DBUtil();
    ArrayList array=new ArrayList();
    String name="";
    try{
        conn=db.getConn();
       String sql="select adminname from admin";
       pstm=conn.prepareStatement(sql);
       rs=pstm.executeQuery();
       while(rs.next()){
        name=rs.getString("adminname");
        array.add(name);
       }
    }catch(Exception ex){
        ex.printStackTrace();
    }
    return array;
   }
}
```

7. 在 com.ajax.servlet 中建立一个名字为 UserCheck 的 servlet。代码如下：

```
package com.ajax.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ajax.dao.AdminDAO;
import java.util.ArrayList;
public class UserCheck extends HttpServlet {
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
```

```
response)
            throws ServletException, IOException {
        //设置生成文件的类型和编码格式
        response.setContentType("text/xml; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        String output = "";
        //处理接收到的参数，生成响应的 XML 文档
        String name = request.getParameter("name");
        AdminDAO admin=new AdminDAO();
        ArrayList<String> array=admin.selectName();
        if(name.length()>0)
        {
            //下面对用户的身份进行判断
            for(int i=0;i<array.size();i++){
           if(name.equals(array.get(i)))
            output ="<response>该用户名已经有人使用，请使用其他用户名</response>";
           else
            output ="<response>恭喜你，该用户名可以使用</response>";

        }
    }
        out.println(output);
        out.close();
    }
    public void doGet(HttpServletRequest request,
                                              HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }
}
```

8. 在工程中建立 UserCheck.jsp。代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<html>
  <head>
    <title>异步身份验证</title>
     <script language="javascript">
       //创建 XMLHttpReques 对象
       function createXMLHttpRequest() {
          if (window.XMLHttpRequest) {
              //Mozilla 浏览器
```

```
            XMLHttpReq = new XMLHttpRequest();
        } else{
            //IE 浏览器
            if (window.ActiveXObject) {
              try {
                    XMLHttpReq = new ActiveXObject("Msxml2.XMLHTTP");
                }catch (e) {
                   try {
                         XMLHttpReq =
                                   new ActiveXObject("Microsoft.XMLHTTP");
                      }catch (e) { }
                }
             }
        }
     }
    //处理服务器响应结果
    function handleResponse() {
     //判断对象状态
     if (XMLHttpReq.readyState == 4) {
        //信息已经成功返回，开始处理信息
       if (XMLHttpReq.status == 200) {
            var out = "";
            var res = XMLHttpReq.responseXML;
            var response=
             res.getElementsByTagName("response")[0].firstChild.nodeValue;
            document.getElementById("result").innerHTML = response;
           }
      }
    }
   //发送客户端的请求
   function sendRequest(url) {
        createXMLHttpRequest();
        XMLHttpReq.open("GET", url, true);
        //指定响应函数
        XMLHttpReq.onreadystatechange = handleResponse;
        //发送请求
        XMLHttpReq.send(null);
    }
    //开始调用 AJAX 的功能
   function userCheck()
   {
      var name = document.getElementById("name").value;
```

```
        var password = document.getElementById("password").value;
        //发送请求
        sendRequest("UserCheck?name="+name+"&password="+password);
      }
    </script>
  </head>
  <body>
  <font size="1">
<br>
  姓名：<input type="text"  id="name" onblur="userCheck()"/><br>
  密码：<input type="text" id="password"/><br>
  <input type="button" value="注册"/>
   <input type="reset" value="取消" />
  <div id="result"></div>
  </font>
  </body>
</html>
```

9. 将工程重新部署到服务器，启动服务器。打开 IE 浏览器在地址栏输入 http://localhost:8080/ajax/UserCheck.jsp，看到如下视图：

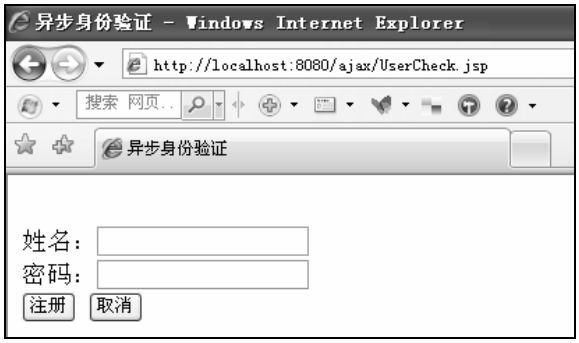


图 5-5 显示 UserCheck.jsp 内容

10. 在用户名中输入“admin”后，将光标放到密码框内，看到如下视图：

图 5-6 输入“admin”后呈现内容

输入其它用户名后，会看到如下视图：

图 5-7 输入其他用户名后呈现内容

通过上面示例就实现了很多论坛注册时用户名不能相同的应用。

4. 相关理论知识

4.1 什么是 AJAX，为什么要使用 AJAX？

1. 什么是 AJAX

AJAX（Asynchronous JavaScript and XML）其实是多种技术的综合，包括 JavaScript、XHTML 和 CSS、DOM、XML 和 XSTL、XMLHttpRequest。其中，使用 XHTML 和 CSS 标准化呈现，使用 DOM 实现动态显示和交互，使用 XML 和 XSTL 进行数据交换与处理，使用 XMLHttpRequest 对象进行异步数据读取，使用 JavaScript 绑定和处理所有数据。

在 AJAX 提出之前，业界对于上述技术都只是单独的使用，没有综合使用，也是由于之前的技术需求所决定的。随着应用的广泛，AJAX 在开发中起到越来越重要的作用。

2. 为什么要使用 AJAX

传统的 Web 应用采用同步交互过程，这种情况下，用户首先向 HTTP 服务器触发一个行为或请求的呼求。反过来，服务器执行某些任务，再向发出请求的用户返回一个 HTML 页面。这是一种不连贯的用户体验，服务器在处理请求的时候，用户多数时间处于等待的状态，屏幕内容也是一片空白。如图所示：

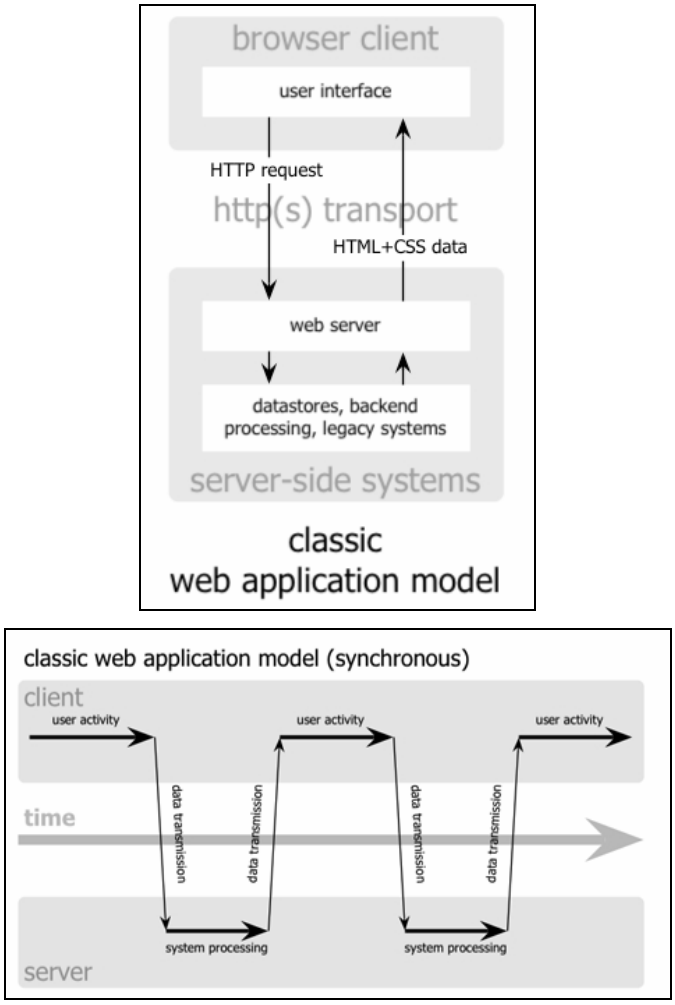
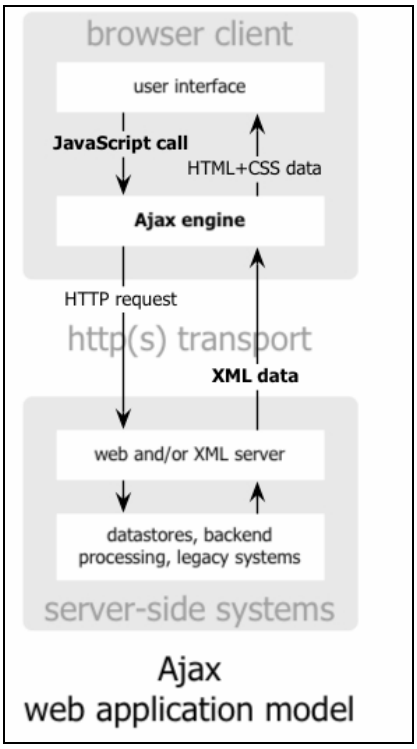


图 5-8 传统 Web 客户端与服务器交互过程

尤其是当负载比较大，响应时间要很长，1 分钟、2 分钟……钟的时候，这种等待就不可忍受了。严重的，超过响应时间，服务器干脆告诉你页面不可用。另外，某些时候，我只是想改变页面一小部分的数据，那为什么我们必须重新加载整个页面呢？当软件设计越来越讲究人性化的时候，这么糟糕的用户体验简直与这种原则背道而驰。为什么总是要让用户等待服务器取数据呢？至少，我们应该减少用户等待的时间。这时候迫切需要一种技术来解决这种问题，AJAX 孕育而生。

与传统的 Web 应用不同，AJAX 采用异步交互过程。AJAX 在用户与服务器之间引入一个中间媒介，从而消除了网络交互过程中的处理—等待—处理—等待缺点。用户的浏览器在执行任务时即装载了 AJAX 引擎。AJAX 引擎用 JavaScript 语言编写，通常藏在一个隐藏的框架中。它负责编译用户界面及与服务器之间的交互。AJAX 引擎允许用户与应用软件之间的交互过程异步进行，独立于用户与网络服务器间的交流。现在，可以用 JavaScript 调用 AJAX 引擎来代替产生一个 HTTP 的用户动作，内存中的数据编辑、页面导航、数据校验这些不需

要重新载入整个页面的需求可以交给 AJAX 来执行。AJAX 交互原理，如图所示：



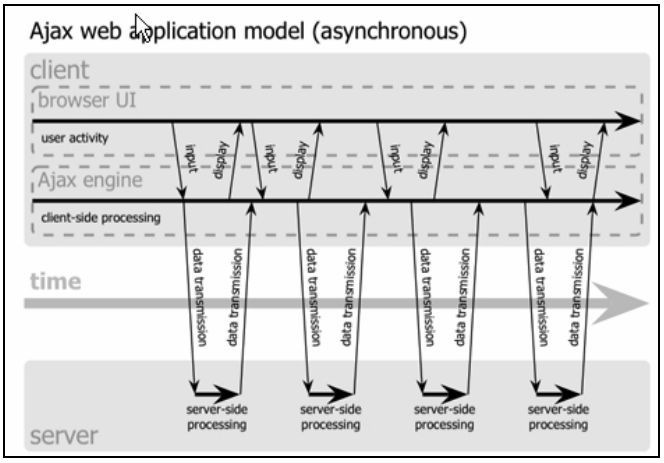


图 5-9 AJAX 客户端与服务器交互过程

使用 AJAX，开发人员、终端用户带来可见的便捷：

- 减轻服务器的负担。AJAX 的原则是“按需取数据”，可以最大程度的减少冗余请求和响应对服务器造成的负担。
- 无刷新更新页面，减少用户心理和实际的等待时间。特别是，当要读取大量的数据的时候，不用像 Reload 那样出现白屏的情况，AJAX 使用 XMLHTTP 对象发送请求

并得到服务器响应，在不重新载入整个页面的情况下用 JavaScript 操作 DOM 最终更新页面。所以在读取数据的过程中，用户所面对的不是白屏，是原来的页面内容（也可以加一个 Loading 的提示框让用户知道处于读取数据过程），只有当数据接收完毕之后才更新相应部分的内容。这种更新是瞬间的，用户几乎感觉不到。

- 带来更好的用户体验。
- 可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。
- 可以调用外部数据。
- 基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。
- 进一步促进页面呈现和数据的分离。

4.2 AJAX 使用的技术

AJAX 不是一项技术，而是几项技术的整合。其中最重要的是 Javascript、XMLHttp-Request、DOM 和 XML 四项技术。

XMLHttpRequest 对象

XMLHttpRequest 是 XMLHTTP 组件的对象，通过这个对象，AJAX 可以像桌面应用程序一样只同服务器进行数据层面的交换，而不用每次都刷新界面，也不用每次将数据处理的工作都交给服务器来做；这样既减轻了服务器负担又加快了响应速度、缩短了用户等待的时间。

IE5.0 开始，开发人员可以在 Web 页面内部使用 XMLHTTP ActiveX 组件扩展自身的功能，不用从当前的 Web 页面导航就可以直接传输数据到服务器或者从服务器接收数据。,Mozilla1.0 以及 NetScape7 则是创建继承 XML 的代理类 XMLHttpRequest。对于大多数情况，XMLHttpRequest 对象和 XMLHTTP 组件很相似，方法和属性类似，只是部分属性不同。

XMLHttpRequest 对象初始化：

```
<script language=”javascript”>
var http_request = false;
//IE 浏览器
http_request = new ActiveXObject("Msxml2.XMLHTTP");
http_request = new ActiveXObject("Microsoft.XMLHTTP");
//Mozilla 浏览器
http_request = new XMLHttpRequest();
</script>
```

XMLHttpRequest 对象的方法：

方法	描述
abort()	停止当前请求
getAllResponseHeaders()	作为字符串返回完整的 headers
getResponseHeader("headerLabel")	作为字符串返回单个的 header 标签
open("method","URL"[,asyncFlag[,"userName"[,"password"]]])	设置未决的请求的目标 URL，方法，和其他参数

send(content)	发送请求
setRequestHeader("label", "value")	设置 header 并和请求一起发送

XMLHttpRequest 对象的属性：

属性	描述
onreadystatechange	状态改变的事件触发器
readyState	对象状态(integer): 0 = 未初始化 1 = 读取中 2 = 已读取 3 = 交互中 4 = 完成
responseText	服务器进程返回数据的文本版本
responseXML	服务器进程返回数据的兼容 DOM 的 XML 文档对象
status	服务器返回的状态码，如：404 = "文件未找到" 、200 ="成功"
statusText	服务器返回的状态文本信息

JavaScript

JavaScript 一直被定位为客户端的脚本语言，应用最多的地方是表单数据的校验。现在，可以通过 JavasScript 操作 XMLHttpRequest，来跟数据库打交道。

DOM

DOM（Document Object Model）是提供给 HTML 和 XML 使用的一组 API，提供了文件的表述结构，并可以利用它改变其中的内容和可见物。脚本语言通过 DOM 才可以跟页面进行交互。Web 开发人员可操作及建立文件的属性、方法以及事件都以对象来展现。比如：

document 就代表页面对象本身。

DOM 中的 HTML 文档树

在 DOM 中，HTML 跟 XML 一样是一种树形结构的文档，<html>是根（root）节点，<head>、<title>、<body>是<html>的子（children）节点，互相之间是兄弟（sibling）节点；<body>下面才是子节点<table>、、<p>等等。如图所示：

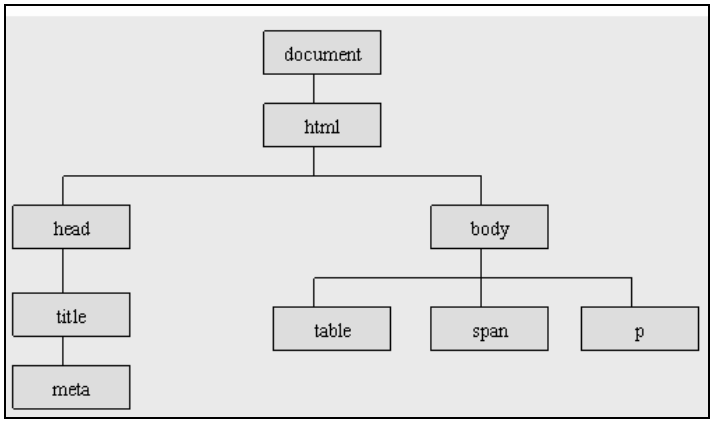


图 5-10 DOM 中的 HTML 文档

DOM 中，HTML 文档各个节点被视为各种类型的 Node 对象。每个 Node 对象都有自己的属性和方法，利用这些属性和方法可以遍历整个文档树。由于 HTML 文档的复杂性，DOM 定义了 nodeType 来表示节点的类型。这里列出 Node 常用的几种节点类型：

接口	nodeType 常量	nodeType 值	备注
Element	Node.ELEMENT_NODE	1	元素节点
Text	Node.TEXT_NODE	3	文本节点
Document	Node.DOCUMENT_NODE	9	document
Comment	Node.COMMENT_NODE	8	注释的文本
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11	document 片断
Attr	Node.ATTRIBUTE_NODE	2	节点属性

下面就对常用的元素进行分析：

1. DOM 树的根节点是个 Document 对象，该对象的 documentElement 属性引用表示文档根元素的 Element 对象（对于 HTML 文档，这个就是<html>标记）。Javascript 操作 HTML 文档的时候，document 即指向整个文档，<body>、<table>等节点类型即为 Element。Comment 类型的节点则是指文档的注释

Document 定义的方法大多数是生产型方法，主要用于创建可以插入文档中的各种类型的节点。常用的 Document 方法有：

方法	描述
createAttribute()	用指定的名字创建新的 Attr 节点。
createComment()	用指定的字符串创建新的 Comment 节点。
createElement()	用指定的标记名创建新的 Element 节点。
createTextNode()	用指定的文本创建新的 TextNode 节点。
getElementById()	返回文档中具有指定 id 属性的 Element 节点。
getElementsByTagName()	返回文档中具有指定标记名的所有 Element 节点。

2. 对于 Element 节点，可以通过调用 getAttribute()、setAttribute()、removeAttribute()方法来查询、设置或者删除一个 Element 节点的性质，比如<table>标记的 border 属性。下面列出

Element 常用的属性：

属性	描述
tagName	元素的标记名称，比如<p>元素为 P。HTML 文档返回的 tabName 均为大写。

Element 常用的方法：

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

3. Attr 对象代表文档元素的属性，有 name、value 等属性，可以通过 Node 接口的 attributes 属性或者调用 Element 接口的 getAttributeNode()方法来获取。不过，在大多数情况下，使用 Element 元素属性的最简单方法是 getAttribute()和 setAttribute()两个方法，而不是 Attr 对象。

4. 使用 DOM 操作 HTML 文档

Node 对象定义了一系列属性和方法，来方便遍历整个文档。用 parentNode 属性和 childNodes[]数组可以在文档树中上下移动；通过遍历 childNodes[]数组或者使用 firstChild 和 nextSibling 属性进行循环操作，也可以使用 lastChild 和 previousSibling 进行逆向循环操作，也可以枚举指定节点的子节点。而调用 appendChild()、insertBefore()、removeChild()、replaceChild()方法可以改变一个节点的子节点从而改变文档树。

需要指出的是，childNodes[]的值实际上是一个 NodeList 对象。因此，可以通过遍历 childNodes[]数组的每个元素，来枚举一个给定节点的所有子节点；通过递归，可以枚举树中的所有节点。下表列出了 Node 对象的一些常用属性和方法。

Node 对象常用属性：

属性	描述
attributes	如果该节点是一个 Element，则以 NamedNodeMap 形式返回该元素的属性。
childNodes	以 Node[]的形式存放当前节点的子节点。如果没有子节点，则返回空数组。
firstChild	以 Node 的形式返回当前节点的第一个子节点。如果没有子节点，则为 null。
lastChild	以 Node 的形式返回当前节点的最后一个子节点。如果没有子节点，则为 null。
nextSibling	以 Node 的形式返回当前节点的兄弟下一个节点。如果没有这样的节点，则返回 null。
nodeName	节点的名字，Element 节点则代表 Element 的标记名称。
nodeType	代表节点的类型。

属性	描述
parentNode	以 Node 的形式返回当前节点的父节点。如果没有父节点，则为 null。
previousSibling	以 Node 的形式返回紧挨当前节点、位于它之前的兄弟节点。如果没有这样的节点，则返回 null。

Node 对象常用方法：

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

5. 遍历文档的节点示例

建立 node.html 文件，代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
var elementName = ""; //全局变量，保存 Element 标记名，使用完毕要清空
function countTotalElement(node) { //参数 node 是一个 Node 对象
    var total = 0;
    if(node.nodeType == 1) { //检查 node 是否为 Element 对象
        total++;    //如果是，计数器加 1
    elementName = elementName + node.tagName + "\r\n"; //保存标记名
    }
    var childrens = node.childNodes;   //获取 node 的全部子节点
    for(var i=0;i<childrens.length;i++) {
    total += countTotalElement(childrens[i]); //在每个子节点上进行递归操作
    }
    return total;
}
</script>
</head>
<body>
<a href="javascript:void(0)"
onClick="alert('标记总数：' + countTotalElement(document) + '\r\n
```

```
全部标记如下：\r\n' + elementName);elementName='';">开始统计</a>
</body>
</html>
```

运行 node.html 文件，结果如下：

图 5-11 node.html 文件运行结果

XML

通过 XML（Extensible Markup Language），可以规范的定义结构化数据，是网上传输的数据和文档符合统一的标准。用 XML 表述的数据和文档，可以很容易的让所有程序共享。

在数据表示方面，XML 文档更加结构化。DOM 在支持 HTML 的基础上提供了一系列的 API，支持针对 XML 的访问和操作。利用这些 API，我们可以从 XML 中提取信息，动态的创建这些信息的 HTML 呈现文档。处理 XML 文档，通常遵循“加载 XML 文档提取信息，加工信息，创建 HTML 文档的过程。下面的例子演示了如何加载并处理 XML 文档。

建立一个名为 employees.xml 文件，代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<employees>
    <employee name="张三">
        <job>Programmer</job>
        <salary>32768</salary>
    </employee>
    <employee name="李四">
        <job>Sales</job>
        <salary>70000</salary>
    </employee>
    <employee name="王五">
        <job>CEO</job>
        <salary>100000</salary>
    </employee>
</employees>
```

建立一个名为 test.html 文件，代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
<script language="javascript">
function loadXML(handler) {
    var url = "employees.xml";
    if(document.implementation&&document.implementation.createDocument) {
        var xmldoc = document.implementation.createDocument("", "", null);
        xmldoc.onload =  handler(xmldoc, url);
        xmldoc.load(url);
    }
    else if(window.ActiveXObject) {
        var xmldoc = new ActiveXObject("Microsoft.XMLDOM");
        xmldoc.onreadystatechange = function() {
            if(xmldoc.readyState == 4) handler(xmldoc, url);
        }
        xmldoc.load(url);
    }
}
function makeTable(xmldoc, url) {
    var table = document.createElement("table");
    table.setAttribute("border","1");
    table.setAttribute("width","600");
    table.setAttribute("class","tab-content");
    document.body.appendChild(table);
    var caption = "Employee Data from " + url;
    table.createCaption().appendChild(document.createTextNode(caption));
    var header = table.createTHead();
    var headerrow = header.insertRow(0);
    headerrow.insertCell(0).appendChild(document.createTextNode("姓名"));
    headerrow.insertCell(1).appendChild(document.createTextNode("职业"));
    headerrow.insertCell(2).appendChild(document.createTextNode("工资"));
    var employees = xmldoc.getElementsByTagName("employee");
    for(var i=0;i<employees.length;i++) {
        var e = employees[i];
        var name = e.getAttribute("name");
        var job = e.getElementsByTagName("job")[0].firstChild.data;
        var salary = e.getElementsByTagName("salary")[0].firstChild.data;
        var row = table.insertRow(i+1);
```

```
        row.insertCell(0).appendChild(document.createTextNode(name));
        row.insertCell(1).appendChild(document.createTextNode(job));
        row.insertCell(2).appendChild(document.createTextNode(salary));
    }
}
</script>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body onLoad="loadXML(makeTable)">
</body>
</html>
```

运行 test.html 文件，结果如下：

Employee Data from employees.xml

姓名	职业	工资
张三	Programmer	32768
李四	Sales	70000
王五	CEO	100000

图 5-12 test.html 运行结果

4.3 AJAX 开发流程

这里，我们通过一步步的解析，来形成一个发送和接收 XMLHttpRequest 请求的程序框架。AJAX 实质上也是遵循 Request/Server 模式，所以这个框架基本的流程也是：对象初始化 a 发送请求 a 服务器接收，a 服务器返回 a 客户端接收 a 修改客户端页面内容。只不过这个过程是异步的。

1. 初始化对象并发出 XMLHttpRequest 请求。

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

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
      http_request = new XMLHttpRequest();
 }
 else if (window.ActiveXObject) { // IE
      http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

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

http_request = new XMLHttpRequest();

http_request.overrideMimeType('text/xml');

3. 指定响应处理函数

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

http_request.onreadystatechange = processRequest;

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

http_request.onreadystatechange = function(){ };

发出 HTTP 请求

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

http_request.open('GET', 'http://www.example.org/some.file', true);

http_request.send(null);

open 的第一个参数是 HTTP 请求的方法，为 Get，Post 或者 Head。

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

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

按照顺序，open 调用完毕之后要调用 send 方法。send 的参数如果是以 Post 方式发出的话，可以是任何想传给服务器的内容。不过，跟 form 一样，如果要传文件或者 Post 内容给服务器，必须先调用 setRequestHeader 方法，修改 MIME 类别。如下：

http_request.setRequestHeader(“Content-Type”,”application/x-www-form-urlencoded”);

这时资料则以查询字符串的形式列出，作为 sned 的参数，例如：

name=value&anothername=othervalue&so=on

4. 处理服务器返回的信息

在第二步我们已经指定了响应处理函数，这一步，来看看这个响应处理函数都应该做什么。

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

```
if (http_request.readyState == 4) {
    //信息已经返回，可以开始处理
} else {
    //信息还没有返回，等待
```

```
}
```

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

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

XMLHttpRequest 对成功返回的信息有两种处理方式：

responseText，将传回的信息当字符串使用。

responseXML，将传回的信息当 XML 文档使用，可以用 DOM 处理。

5. 总结上面的步骤，我们整理出一个初步的开发框架，供以后调用。这里将服务器返回的信息用 window.alert，以字符串的形式显示出来。

```
<script language="javascript">
    var http_request = false;
    function send_request(url) {//初始化、指定处理函数、发送请求的函数
        http_request = false;
        //开始初始化 XMLHttpRequest 对象
        if(window.XMLHttpRequest) { //Mozilla 浏览器
            http_request = new XMLHttpRequest();
            if (http_request.overrideMimeType) {//设置 MiME 类别
                http_request.overrideMimeType("text/xml");
            }
        }
        else if (window.ActiveXObject) { //IE 浏览器
            try {
                http_request = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try {
                    http_request = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {}
            }
        }
        if (!http_request) { //异常，创建对象实例失败
            window.alert("不能创建 XMLHttpRequest 对象实例.");
            return false;
        }
        http_request.onreadystatechange = processRequest;
        //确定发送请求的方式和 URL 以及是否同步执行下段代码
```

```
        http_request.open("GET", url, true);
        http_request.send(null);
    }
    //处理返回信息的函数
    function processRequest() {
        if (http_request.readyState == 4) { //判断对象状态
            if (http_request.status == 200) { //信息已经成功返回，开始处理信息
                alert(http_request.responseText);
            } else { //页面不正常
                alert("您所请求的页面有异常。");
            }
        }
    }
```

以上代码是开发 AJAX 通用的代码，根据相关的业务逻辑修改即可以使用。

5. 实验

1. 完成实践知识部分示例。（50 分钟）
2. 完成理论知识部分示例。（40 分钟）

6. 作业

使用 DOM 操作 HTML 文件，修改 HTML 文件。建立一个 homwork.html 文件，如图所示：

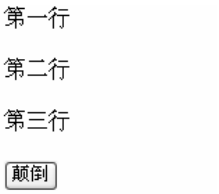


图 5-13 homework.html 文件

当点击“颠倒”按钮后，结果如图所示：

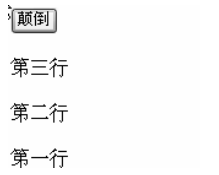


图 5-14 homework.html 文件运行效果

案例二 DWR 框架入门

1. 教学目标

1.1 了解 DWR 框架

1.2 掌握 dwr.xml 配置

1.3 掌握 DWR 框架的基本应用

1.4 熟悉 util.js 中的相关方法

2. 工作任务

2.1 用 DWR 框架建立一个 Hello World 应用

2.2 用 DWR 框架建立一个二级联动菜单应用

3. 相关实践知识

3.1 用 DWR 建立一个 Hello World 应用

1. 从 DWR 官方网站 http://getahead.org/dwr/download 下载 dwr.jar 文件，目前成熟版本 2.0。

2. 从 apache 官方网站 http://commons.apache.org/downloads/download_logging.cgi 下载 commons-logging-1.1.1.jar 日志处理的文件。注意：如果工程中不加入这个 jar 文件，程序可能会出现错误。

3. 在 MyEclipse 中建立一个名为“dwr”的 Web 工程。如图所示：

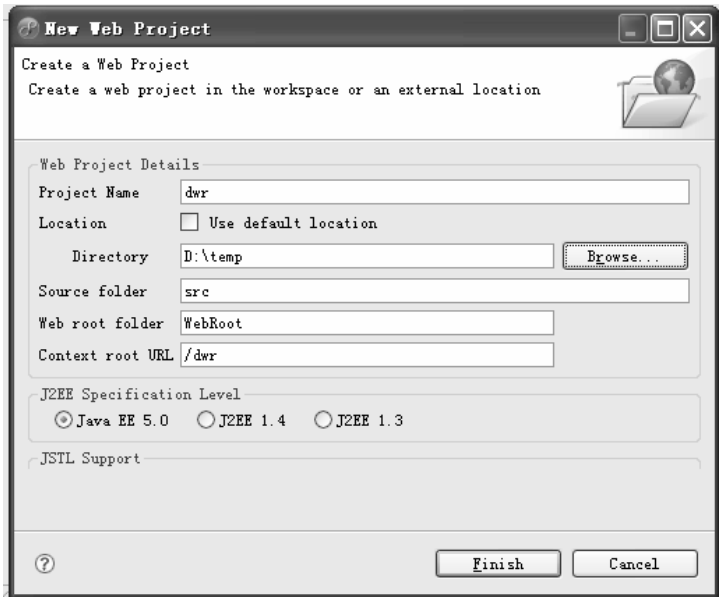


图 5-15 建立“dwr”Web 工程

4. 将下载的 dwr.jar 和 commons-logging-1.1.1.jar 文件，复制到 WEB-INF/lib 目录下。

5. 在“dwr”工程中建立名为“com.dwr.service”包，并在该包下建立名为“Service”的 Java 类。代码如下：

```
package com.dwr.service;

public class Service {
    public String sayHello(String yourName) {
        //可以是访问数据库的复杂代码
        return "HelloWorld " + yourName; }
}
```

6. 配置 web.xml 文件。在 web.xml 加入如下代码，进行 DWR 框架配置。

```
<servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
            org.directwebremoting.servlet.DwrServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
```

7. 编写 dwr.xml 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting 2.0//EN"
"http://www.getahead.ltd.uk/dwr/dwr20.dtd">
<dwr>
    <allow>
   <!-- javascript 的对象是 service,create="new"是 DWR 自己创建类 -->
        <create creator="new" javascript="service" scope="application">
            <param name="class" value="com.dwr.service.Service" />
        </create>
    </allow>
</dwr>
```

8. 将工程部署到 tomcat，启动服务器。在 IE 中输入 http://localhost:8080/dwr/dwr，得到结果如图所示：

图 5-16 配置 DWR 后运行结果图

9. 点击“service”链接，得到结果如图所示：

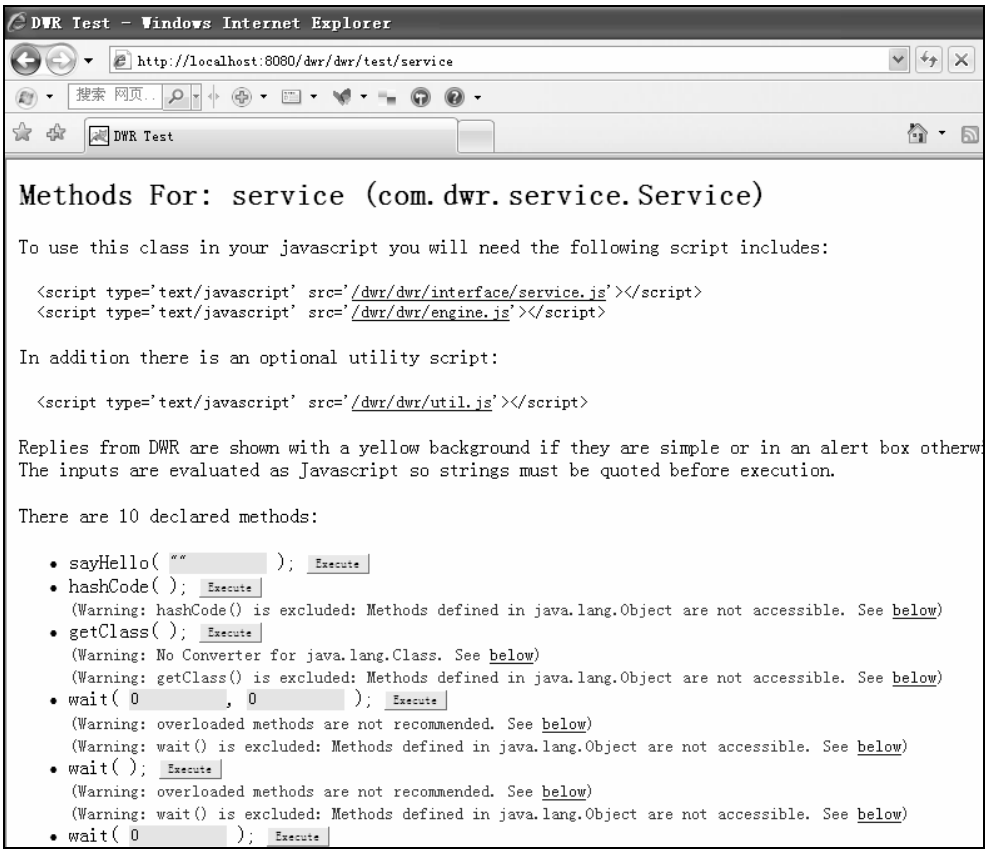


图 5-17 点击“service”后结果图

10. 在图 1-3 中的“sayHello()”方法中输入“dwr”，点击“Execute”按钮。得到结果如图所示：

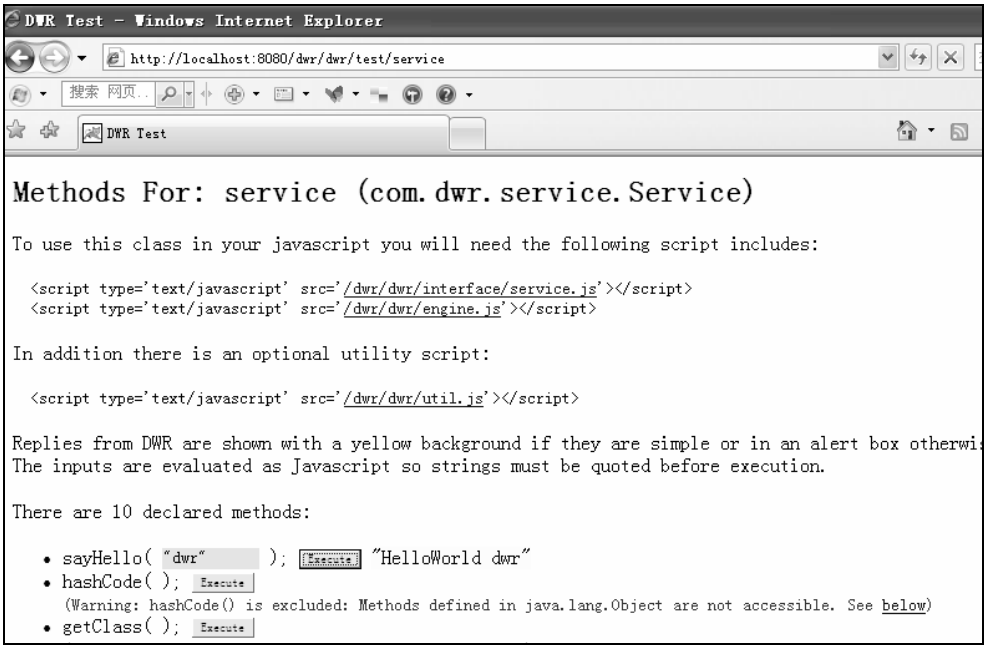


图 5-18 输入“dwr”后点击“Excute”按钮执行结果

可以看到输出了“HelloWorld dwr”这句话。

11. 此时证明框架已经在工程中配置成功，下面编写 Index.jsp 页面进行测试。Index.jsp 代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>My JSP 'first_dwr.jsp' starting page</title>
        <script    type='text/javascript'    src='dwr/interface/service.js'>
</script>
        <script type='text/javascript' src='dwr/util.js'></script>
        <script type='text/javascript' src='dwr/engine.js'></script>

        <script type="text/javascript">
        function firstDwr()
        {
         service.sayHello("dwr",callBackHello);
         }
         function callBackHello(data){
         alert(data);
         }
          </script>
```

```
    </head>
    <body>
        <input type="button" name="button" value="测试" onclick="firstDwr()">
    </body>
</html>
```

12. 在 IE 中输入地址 http://localhost:8080/dwr/index.jsp，到页面如图所示：

图 5-19 运行 index.jsp 后的结果

13. 点击“测试”按钮，得到结果如图所示：

图 5-20 点击“测试”按钮的结果

这样我们就完成了一个简单的 Hello World 程序的应用。

DWR 框架是非常优秀的 AJAX 框架，其应用很广泛。我们已经初次尝试了一个简单应用，下面我们一起尝试一个在实际工作中经常用到的案例，使用 DWR 框架实现。

3.2 用 DWR 框架建立二级菜单联动的应用

1. 新建一个名为“dynamic”的 Web 工程。如图所示：

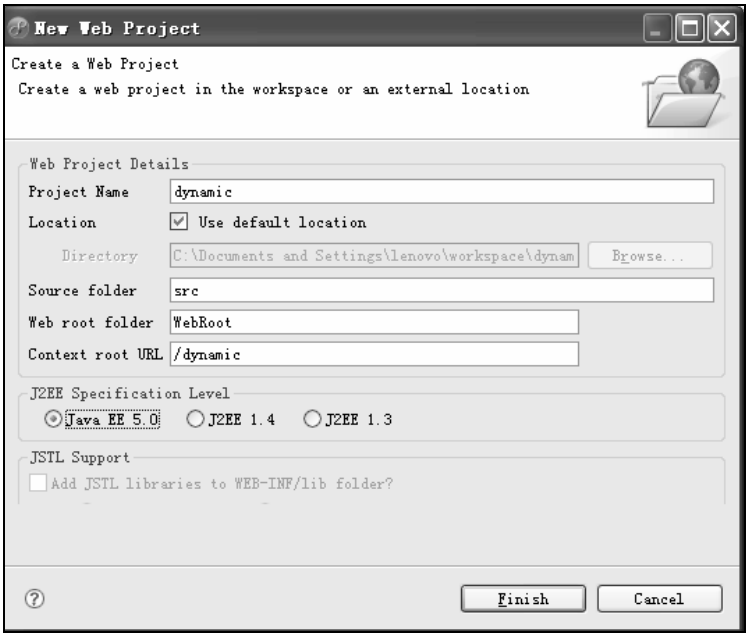


图 5-21 新建“dynamic”Web 工程

2. 在工程中建立名为“com.dwr.service”包，在该包内建立 Bike 类。代码如下：

```
package com.dwr.service;
import java.util.Map;
import java.util.TreeMap;
public class Bike {
    private Map<String, String[]> bikes;
    public Bike() {
        bikes = new TreeMap<String, String[]>();
        bikes.put("2000", new String[] { "2000 T1", "2000 T2", "2000 T3" });
        bikes.put("2001", new String[] { "2001 A1", "2001 A2" });
        bikes.put("2002", new String[] { "2002 BW1", "2002 BW2", "2002 BW" });
        bikes.put("2003", new String[] { "2003 S320" });
        bikes.put("2004", new String[] { "2004 TA1", "2004 TA2", "2004 TA3" });
    }
    public String[] getYears() {
        String[] keys = new String[bikes.size()];
        int i = 0;
        for (String key : bikes.keySet()) {
            keys[i++] = key;
        }
        return keys;
    }
```

```
    public String[] getBikes(String year) {
        return bikes.get(year);
    }
}
```

3. 将 dwr.jar 和 commons-logging-1.1.1.jar 文件，复制到 WEB-INF/lib 目录下。
4. 配置 web.xml 文件。在 web.xml 加入如下代码，进行 DWR 框架配置。

```
<servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
            org.directwebremoting.servlet.DwrServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
```

5. 编写 dwr.xml 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
"http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
    <allow>
        <create creator="new" javascript="Bike" scope="application">
            <param name="class" value="com.dwr.service.Bike" />
        </create>
    </allow>
</dwr>
```

6. 在 WEB-INF 目录下建立“js”文件夹，在“js”文件夹下创建 bike.js 文件。代码如下：

```
function refreshYearList() {
    Bike.getYears(populateYearList);
}
function populateYearList(list) {
    DWRUtil.removeAllOptions("years");
    DWRUtil.addOptions("years", list);
```

```
    refreshBikeList();
}
function refreshBikeList() {
    var year = $("years").value;
    Bike.getBikes(year, populateBikeList);
}
function populateBikeList(list) {
    DWRUtil.removeAllOptions("bikes");
    DWRUtil.addOptions("bikes", list);
}
```

7. 在 WEB-INF 下建立 bike.html 文件。代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Insert title here</title>
        <script type='text/javascript' src='dwr/interface/Bike.js'></script>
        <script type='text/javascript' src='dwr/engine.js'></script>
        <script type='text/javascript' src='dwr/util.js'></script>
        <script type='text/javascript' src='js/bike.js'></script>
    </head>
    <body onload="refreshYearList();">
        年份:
        <select id="years" onchange="refreshBikeList();"></select>
        <br />
        <br />
        型号:
        <select id="bikes"></select>
        <br />
    </body>
</html>
```

8. 将工程部署到 tomcat，启动 tomcat。在 IE 中输入地址 http://localhost:8080/dynamic/bike.html，得到页面如下：

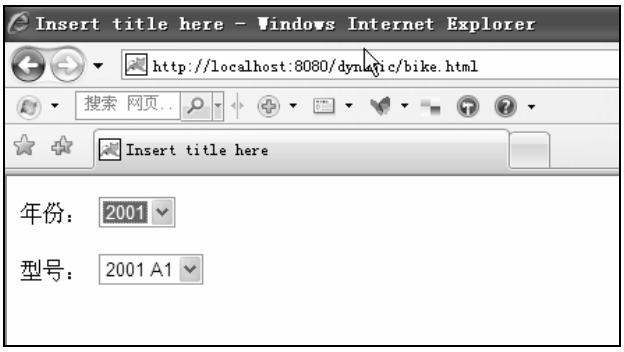


图 5-22 运行 bike.html 后结果

9. 选择不同的“年份”，可以得到属于本年份的“型号”。如图所示：

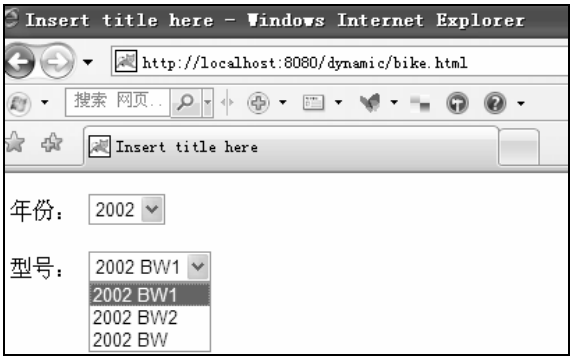


图 5-23 选择不同“年份”后结果

这样我们就轻松的完成了二级菜单的联动。

4. 相关理论知识

4.1 什么是 DWR

DWR 是一个可以允许你去创建 AJAX Web 站点的 Java 开源库，它可以通过浏览器端的 JavaScript 代码去调用服务器端的 Java 代码，看起来就像是 Java 代码运行在浏览器上一样。DWR 是一个完整的异步 AJAX 框架，它隐藏了 XMLHttpRequest 对象，程序员在开发过程中不需要接触 XMLHttpRequest 对象就可以向服务器发送异步请求并通过回调方式处理服务器的返回值。

DWR 包含两个主要部分：

- 运行在服务器端的 servlet 控制器（DwrServlet），它负责接收请求，调用相应业务逻辑进行处理，向客户端返回响应。
- 运行在浏览器端的 JavaScript，它负责向服务器端发送请求，接收响应，动态更新页面。

DWR 工作原理是通过动态的把 Java 类生成为 JavaScript。它的代码就像 Ajax 魔法一样，

你感觉调用就像发生在浏览器端，但是实际上代码调用发生在服务器端，DWR 负责数据的传递和转换。这种从 JavaScript 到 Java 的远程调用功能的方式使 DWR 用起来有种非常像 RMI 或者 SOAP 的常规 RPC 机制，而且 DWR 的优点在于不需要任何的网页浏览器插件就能运行在网页上。

Java 从根本上讲是同步机制，然而 Ajax 却是异步的。所以你调用远程方法时，当数据已从网络上返回的时候，你要提供有回调(callback)来接收数据。

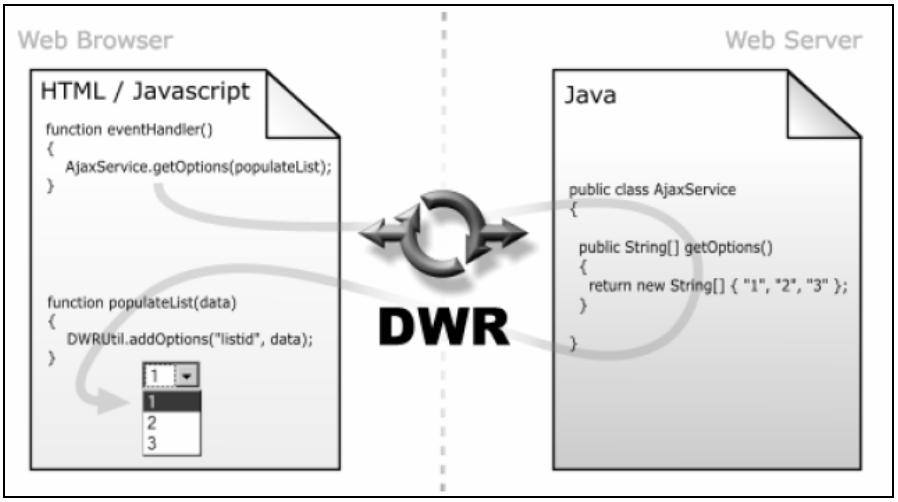


图 5-24 显示 DWR 如何选择一个下拉列表的内容作为 JavaScript 的 onclick 事件的结果

4.2 web.xml 文件配置

1. 主要配置

在 web.xml 中配置 DWR servlet，没有它 DWR 就不起作用。

```
<servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
            org.directwebremoting.servlet.DwrServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
```

注意：在 DWR2.x 里，DwrServlets 是类 org.directwebremoting.servlet.DwrServlet，尽管 uk.ltd.getahead.dwr.DWRServlet 仍然可以用。但在 DWR 1.x 中你不得不使用后者。

有些额外的 servlet 参数，在有些地方很重要，尤其是 debug 参数。这个参数扩展 DWR 的标准结构是使用<init-params>。放在<servlet>内，就像如下使用 ：

```
<servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
            org.directwebremoting.servlet.DwrServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
```

2. 常用<init-param>参数列表

allowGetForSafariButMakeForgeryEasier

开始版本：2.0

默认值：false

描述：设置成 true 使 DWR 工作在 Safari 1.x ，会稍微降低安全性。

crossDomainSessionSecurity

开始版本：2.0

默认值：true

描述：设置成 false 使能够从其他域进行请求。注意，这样做会在安全性上有点冒险。

Debug

开始版本：1.0

默认值：false

描述：设置成 true 使 DWR 能够 debug 和进入测试页面

scriptSessionTimeout

开始版本：2.0

默认值：1800000(30 分钟) 描述：script session 的超时设置

maxCallCount

开始版本：2.0rc2 和 1.1.4

默认值：20

描述：一次批量(batch)允许最大的调用数量。（帮助保护 Dos 攻击）

4.3 dwr.xml 文件配置

dwr.xml 是你用来配置 DWR 的文件，默认是将其放入 WEB-INF 文件夹。

dwr.xml 基本结构：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting 2.0//EN"
"http://www.getahead.ltd.uk/dwr/dwr20.dtd">
<dwr><!-- 仅当需要扩展 DWR 时才需要 -->
    <init>
        <creator id="..." class="..." />
        <converter id="..." class="..." />
    </init><!-- 没有它 DWR 什么也做不了 -->
    <allow>
        <create creator="..." javascript="..." />
        <convert converter="..." match="..." />
    </allow><!-- 有必要告诉 DWR 方法签名 -->
    <signatures>...</signatures>
</dwr>
```

1. <init>标签

这个初始化部分申明被用来创建远程 beans 而且这个类能被用来以某种过程转换。大多数例子你将不需要用它，如果你想去定义一个新的 Creator 或者 Converter，就要在此声明。在 init 部分里有的定义只是告诉 DWR 这些扩展类的存在，给出了如何使用的信息。这时它们还没有被使用。这种方式很像 Java 中的 import 语句。多数类需要在使用前先 import 一下，但是只有 import 语句并不表明这个类已经被使用了。每一个 creator 和 converter 都用 id 属性，以便后面使用。

2. <allow>标签

allow 部分定义了 DWR 能够创建和转换的类。

3. <creator>标签

每一个在类中被调用的方法需要一个<create …>有若干类型的 creator，使用“new”关键字或者 Spring 框架等。create 元素是如下的结构

```
<allow>
    <create creator="..." javascript="..." scope="...">
        <param name="..." value="..." />
        <auth method="..." role="..." />
        <exclude method="..." />
        <include method="..." />
    </create>
    ...
</allow>
```

creator 属性。

new：Java 用“new”关键字创造对象 是 DWR 默认的 creator。

这个 creator 将使用默认构造器创建类的实例，以下是用 new 创建器的好处：安全：DWR 创造的对象生存的时间越短，多次调用中间的值不一致的错误机会越少，内存消耗低。如果你的站点用户量非常大，这个创造器可以减少 VM 的内存溢出。

javascript 属性

在 JavaScript 里给你创建的对象命名。避免使用 JavaScript 保留字。这个名字将在页面里作为 js 被导入。例如本章第一个 Hello World 应用。dwr.xml 配置如下：

```
<create creator="new" javascript="service" scope="application">
```

在 index.jsp 对应引用中，代码如下：

```
<script type='text/javascript' src='dwr/interface/service.js'>
```

scope 属性

和定义在 servlet 的 scope 是一样大的范围，它允许你指定哪个 bean 是可以获得的。选项可以是：application，session，request 和 page。scope 选项是可选的，默认为 page，使用 session 请求 cookies。目前，DWR 还不支持 URL 重写。

<param>元素

被用来指定创造器的其他参数，每种构造器各有不同。例如，“new”创造器需要知道要创建的对象类型是什么。每一个创造器的参数在各自的文档中能找到。

include 和 exclude 元素

允许一个创造器去限制进入类的方法。一个创造器必须指定 include 列表或 exclude 列表之一。如果是 include 列表则暗示默认的访问策略是“拒绝”，include 中的每个方法就是允许访问的方法；如果是 exclude 列表则暗示默认的访问策略是“允许”，exclude 中的每个方法就是拒绝访问的方法。

```
<create creator="new" javascript="Fred">
    <param name="class" value="com.example.Fred" />
    <include method="setWibble" />
</create>
```

说明只能在 DWR 中使用 Fred 的是 setWibble 方法。

4. auth 元素

允许你指定一个 J2EE 的角色作为将来的访问控制检查：

```
<create creator="new" javascript="Fred">
    <param name="class" value="com.example.Fred" />
    <auth method="setWibble" role="admin" />
</create>
```

4.4 engine.js 功能

engine.js 对 DWR 非常重要，因为它是自动态生成的接口的转换后被 JavaScript 函数调用，所以只要用到 DWR 的地方就需要它。每一个页面都需要下面这些语句来引入主

DWR 引擎。在实践知识两个示例中都使用到如下代码：

```
<script type='text/javascript' src='dwr/engine.js'></script>
```

使用 DWREngine.setX() 函数来设置全局相关的一些属性。

4.5 util.js 功能

util.js 包含了一些工具函数来帮助你用 JavaScript 数据(例如从服务器返回的数据)来更新你的 Web 页面。你可以在 DWR 以外使用它，因为它不依赖于 DWR 的其他部分。你可以下载整个 DWR 或者单独下载。它有 4 个基本的操作页面的函数：getValue[s]()和 setValue[s]()可以操作大部分 HTML 元素除了 table，list 和 image。getText()可以操作 select，list。要修改 table 可以用 addRows()和 removeAllRows()。要修改列表(select 列表和 ul,ol 列表)可以用 addOptions()和 removeAllOptions()。 还有一些其他功能不是 DWRUtil 的一部分。但它们也很有用，它们可以用来解决一些小问题，但是它们不是对于所有任务都通用的。

下面介绍 util.js 中常用的函数：

1. $() 函数。

大致上的讲：$ = document.getElementById。因为在 Ajax 程序中，你会需要写很多这样的语句，所以使用$()会更简洁。通过指定的 id 来查找当前 HTML 文档中的元素，如果传递给它多个参数，它会返回找到的元素的数组。所有非 String 类型的参数会被原封不动的返回。这个函数的灵感来自于 prototype 库，但是它可以在更多的浏览器上运行。

2. addOptions and removeAllOptions

DWR 的一个常遇到的任务就是根据选项填充选择列表。下面的例子就是根据输入填充列表。下面将介绍 DWRUtil.addOptions()的几种使用方法。如果你希望在你更新了 select 以后，它仍然保持原来的选择，你要像下面这样做：

```
var sel = DWRUtil.getValue(id);
DWRUtil.removeAllOptions(id);
DWRUtil.addOptions(id, ...);
DWRUtil.setValue(id, sel);
```

DWRUtil.addOptions 有 5 种模式 ：

数组： DWRUtil.addOptions(selectid, array)会创建一堆 option，每个 option

文字和值都是数组元素中的值。

对象数组（指定 text）：DWRUtil.addOptions(selectid, data, prop)用每个数组元素创造一个 option，option 的值和文字都是在 prop 中指定对象的属性。

对象数组（指定 text 和 value 值）：DWRUtil.addOptions(selectid，array，Valueprop，textprop) 用每个数组元素创造一个 option，option 的值是对象的 valueprop 属性，option 的文字是对象的 textprop 属性。

对象：DWRUtil.addOptions(selectid, map, reverse)用每个属性创建一个 option。对象属性名用来作为 option 的值，对象属性值用来作为属性的文字，这听上去有些不对，但是事实上却是正确的方式。如果 reverse 参数被设置为 true，那么对象属性值用来作为选项的值。

对象的 Map：DWRUtil.addOptions(selectid，map，valueprop，textprop) 用 map 中的每一个对象创建一个 option。用对象的 valueprop 属性做为 option 的 value，用对象的 textprop 属性做为 option 的文字。

ol 或 ul 列表：DWRUtil.addOptions(ulid，array)用数组中的元素创建一堆 li 元素，他们的 innerHTML 是数组元素中的值。这种模式可以用来创建 ul 和 ol 列表。

3. addRows and removeAllRows

DWR 通过这两个函数来帮你操作 table：DWRUtil.addRows()和 DWRUtil.removeAllRows()。这个函数的第一个参数都是 table、tbody、thead、tfoot 的 id。一般来说最好使用 tbody，因为这样可以保持你的 header 和 footer 行不变，并且可以防止 Internet Explorer 的 bug。

DWRUtil.removeAllRows()

```
DWRUtil.removeAllRows(id);
```

描述：通过 id 删除 table 中所有行。

参数列表：

id：table 元素的 id（最好是 tbody 元素的 id）

DWRUtil.addRows()

```
DWRUtil.addRows(id, array, cellfuncs, [options]);
```

描述：向指定 id 的 table 元素添加行。它使用数组中的每一个元素在 table 中创建一行。然后用 cellfuncs 数组中的函数创建一个列。单元格是依次用 cellfunc 根据数组中的元素创建出来的。

DWR1.1 开始，addRows()也可以用对象做为数据。如果你用一个对象代替一个数组来创建单元格，这个对象会被传递给 cell 函数。

参数列表：

Id：table 元素的 id（最好是 tbody 元素的 id）。

array：数组（DWR1.1 以后可以是对象），做为更新表格数据。

Cellfuncs：函数数组，从传递过来的行数据中提取单元格数据。

Options：一个包含选项的对象（见下面） 选项包括：rowCreator：一个用来创建行的函数(例如，你希望 tr 加个 css)。默认是返回一个 document.createElement("tr")。cellCreator：一个用来创建单元格的函数(例如：用 th 代替 td)默认返回一个。

```
document.createElement("td")
```

4. getValue

DWRUtil.getValue(id)是 setValue()对应的“读版本”。它可以从 HTML 元素中取出其中的值，而你不用管这个元素是 select 列表还是一个 div。这个函数能操作大多数 HTML 元素，包括 select(去处当前选项的值而不是文字)、input 元素(包括 textarea)、div 和 span。

5.getValues

getValues()和 getValue()非常相似，除了输入的是包含 name/value 对的 Javascript 对象。

name 是 HTML 元素的 ID，value 会被更改为这些 ID 对象元素的内容。这个函数不会返回对象，它只更改传递给它的值。从 DWR1.1 开始 getValues()可以传入一个 HTML 元素(一个 DOM 对象或者 id 字符串)，然后从它生成一个 reply 对象。

6. onReturn

当按下 return 键时，得到通知。当表单中有 input 元素，触发 return 键会导致表单被提交，这往往不是你想要的，所以你需要触发一些 JavaScript 事件，不幸的是不同的浏览器处理这个事件的方式不一样。而 DWRUtil.onReturn 可以修复这个差异，如果你需要提交表单中的一个元素，可以用这样代码实现：

```
<input type="text" onkeypress="DWRUtil.onReturn(event,submitFunction)" />
<input type="button" onclick="submitFunction()" />
```

7. selectRange

选择一个输入框中的一定范围的文字。你可能为了实现类似“Google suggest”类型的功能而需要选择输入框中的一定范围的文字，但是不同浏览器间选择的模型不一样。但是 DWR Util 函数可以帮你实现。

```
    DWRUtil.selectRange(ele, start, end)
```

8. setValue

DWRUtil.setValue(id，value)根据第一个参数中指定的 id 找到相应元素，并根据第二个参数改变其中的值。 这个函数能操作大多数 HTML 元素包括 select、input 元素（包括 textarea）、div 和 span。

9.setValues

setValues()和 setValue()非常相似，除了输入的是包含 name/value 对的 Javascript 对象。name 是 HTML 元素的 ID，value 是你想要设置给相应的元素的值。

4.6 综合分析 Hello World 应用

1. 新建的 Service 类中有一个 sayHello 方法这个方法就是作为远程调用的服务端方法，这个 Service 类也就是远程调用的类。

2. dwr.xml 是 DWR 用来配置服务端类和浏览器端 JavaScript 对象之间的映射。在<allow></allow>里配置映射关系，示范代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web Remoting 2.0//EN"
"http://www.getahead.ltd.uk/dwr/dwr20.dtd">
<dwr>
    <allow>
 <!--javascript 的对象是 service,create="new"是 DWR 自己创建类-->
       <create creator="new" javascript="service" scope="application">
           <param name="class" value="com.dwr.service.Service" />
       </create>
```

```
    </allow>
</dwr>
```

javascript="service"是在浏览器端的 JavaScript 映射的对象名，但不要用 JavaScript 里的关键字，creator="new"是表示这个类是 dwr 自己创建的，如果 creator="new"那么就必须有下面的<param name=”class” value=”类的全路径”></param>，最后 scope=”application”是说这个 pojo 类的范围，和 Jsp 是一样的。

3. 创建 index.jsp。注意的是在 index.jsp 中导入 js 文件的时候注意顺序和路径命名规律，如果自己写 hello.js 一定放在后面，因为要调用其它 js 文件中的函数（例如实践知识第二个示例）。当我们点击按钮的时候，实质调用的是服务器端对应的 Service 类的 sayHello 方法，可以向函数传递参数，不过要在最后加一个回调函数。

```
service.sayHello("dwr",callBackHello);
```

4. 在 web.xml 里加入 DWRServlet 的配置一是为了远程调用，二是自动生成了

<script type='text/javascript' src='dwr/interface/jshello.js'></script>

<script type='text/javascript' src='dwr/engine.js'></script>

<script type='text/javascript' src='dwr/util.js'></script>文件。

同学们根据相关的理论知识，分析实践知识第二个示例。

5. 实验

1. 完成实践知识示例一，使用 DWR 框架构建一个 HelloWorld 应用。（30 分钟）
2. 完成实践知识示例二，使用 DWR 框架构建一个二级联动菜单应用。（60 分钟）

6. 作业

1. 将本章实践知识的联动菜单示例，改成从数据库中查找数据进行联动菜单的实现。

Item 表和 classes 表的 ddl 语句。一个 item 中有多个 classes。

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[item]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[item]
GO

CREATE TABLE [dbo].[item] (
    [tid] [int] NOT NULL ,
    [tname] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[classes]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
```

```
drop table [dbo].[classes]
GO

CREATE TABLE [dbo].[classes] (
    [cid] [int] NOT NULL ,
    [cname] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [tid] [int] NOT NULL
) ON [PRIMARY]
GO
```

专题六 Web Service 基础知识

1. 教学目标

1.1 了解 WebService

1.2 了解 WSDL

1.3 了解 SOAP 协议

1.4 了解 UDDI

1.5 了解 XFire

1.6 掌握使用 XFire 开发 webservice

2. 工作任务

使用 XFire 构建简单的 WebService 应用。

3. 相关实践知识

用 XFire 构建 WebService 应用

1. 新建 Web Service 项目。如图所示：

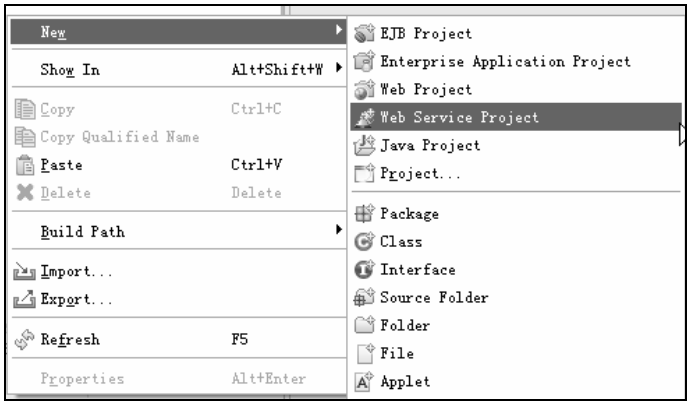


图 6-1 建立 Web Service 工程

2. 将该工程命名为“helloworldservice”。如图所示：

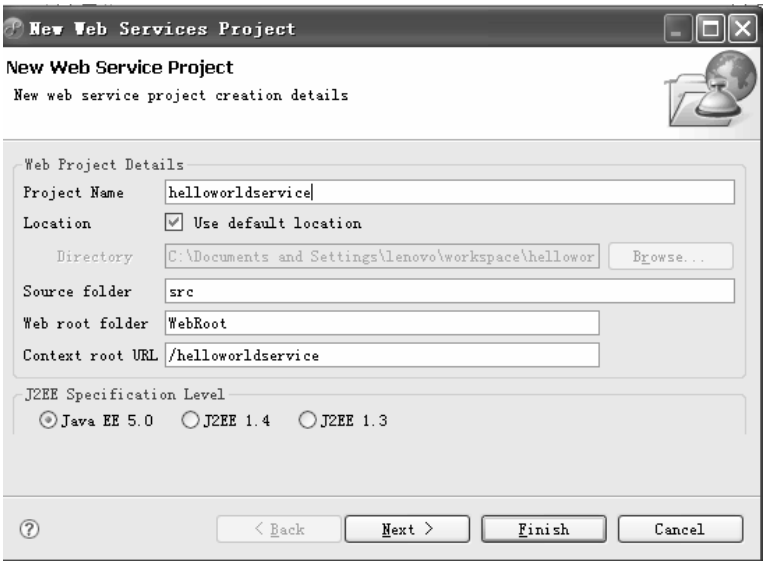


图 6-2 将工程名命名为“helloworldservice”

3. 点击【Next】。看到如下视图：

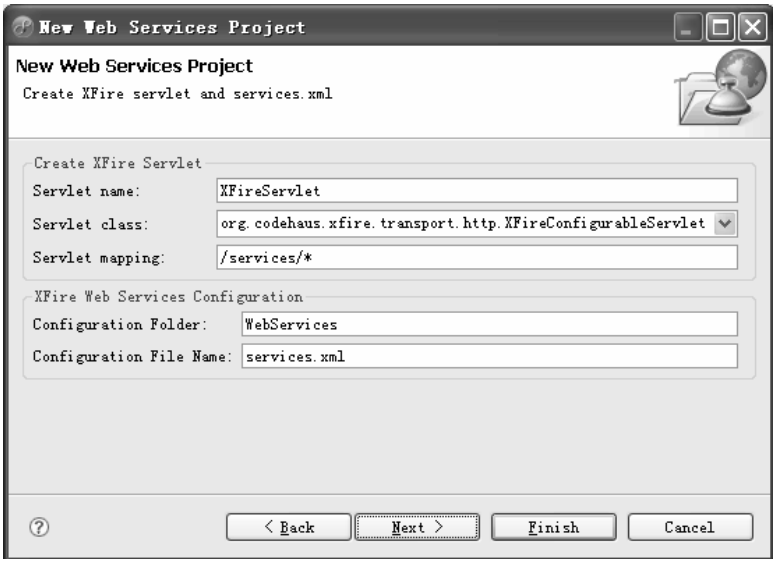


图 6-3 点击【Next】看到视图

4. 点击【Next】。看到如下视图：

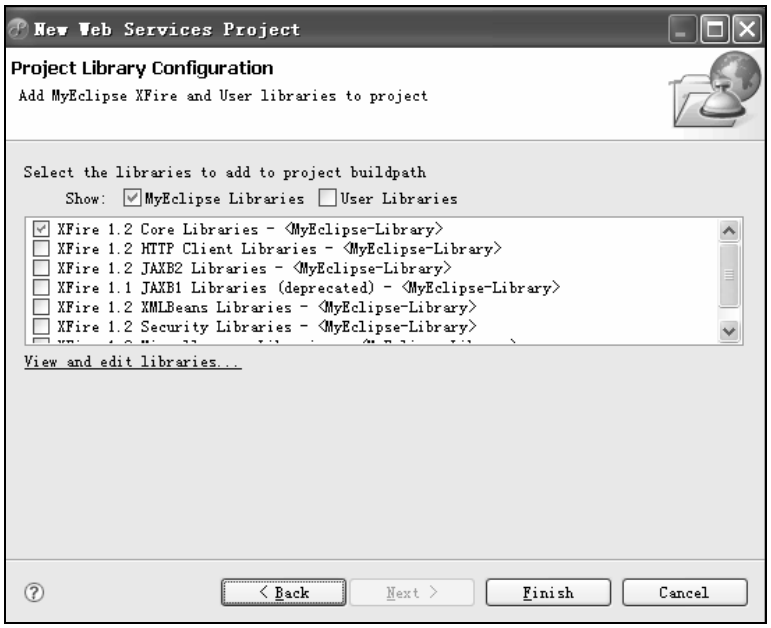


图 6-4 点击【Next】看到视图

5. 点击【Finish】。看到工程结构如图所示：

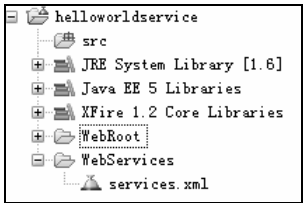


图 6-5“helloworldservice”工程结构

6. 打开 web.xml 文件。web.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                          xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.5"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>XFireServlet</servlet-name>

<servlet-class>org.codehaus.xfire.transport.http.XFireConfigurableServlet</
servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
```

```
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

在 web.xml 配置 org.codehaus.xfire.transport.http.XFireConfigurableServlet。

7. 打开 services.xml 文件。services.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xfire.codehaus.org/config/1.0">

</beans>
```

8. 新建一个 Web Service。如图所示：

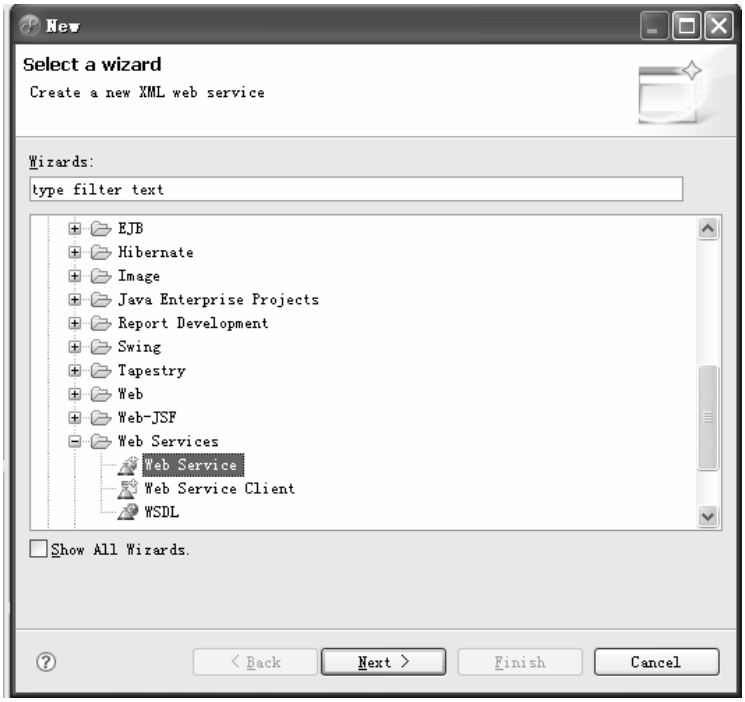


图 6-6 建立 Web Service

9. 点击【Next】。如图所示：

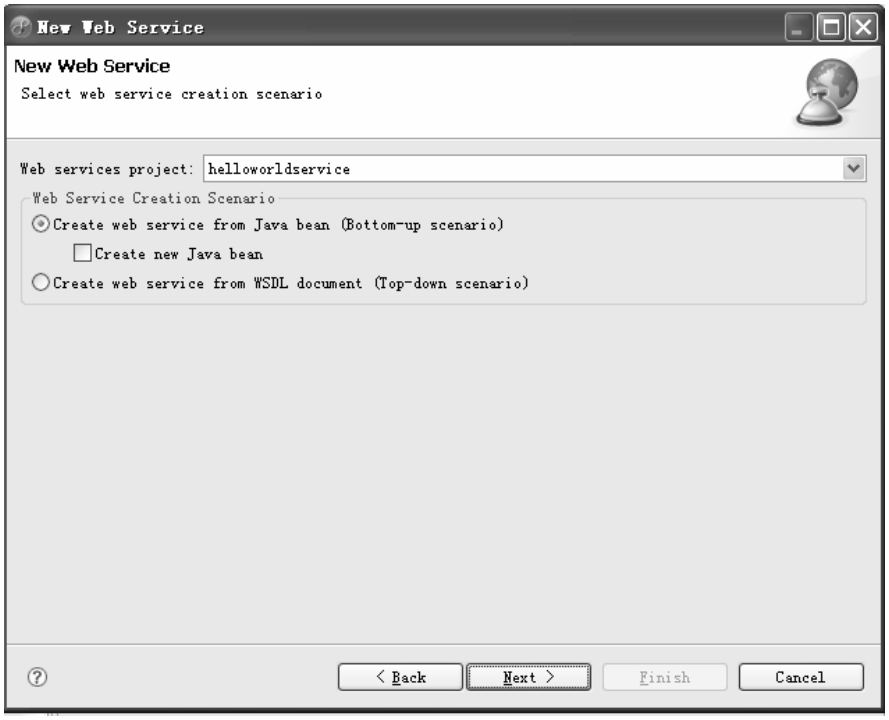


图 6-7 点击【Next】看到视图

10. 选中【Create new Java bean】，点击【Next】。如图所示：

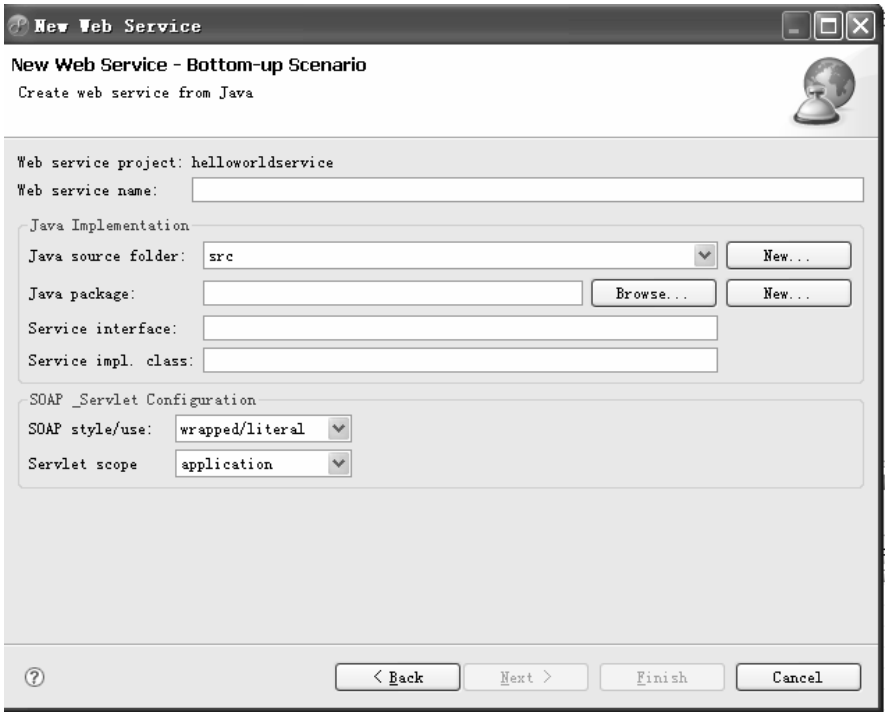


图 6-8 点击【Next】看到视图

11. 【Web service name】命名为“HelloWorld”。如图所示：

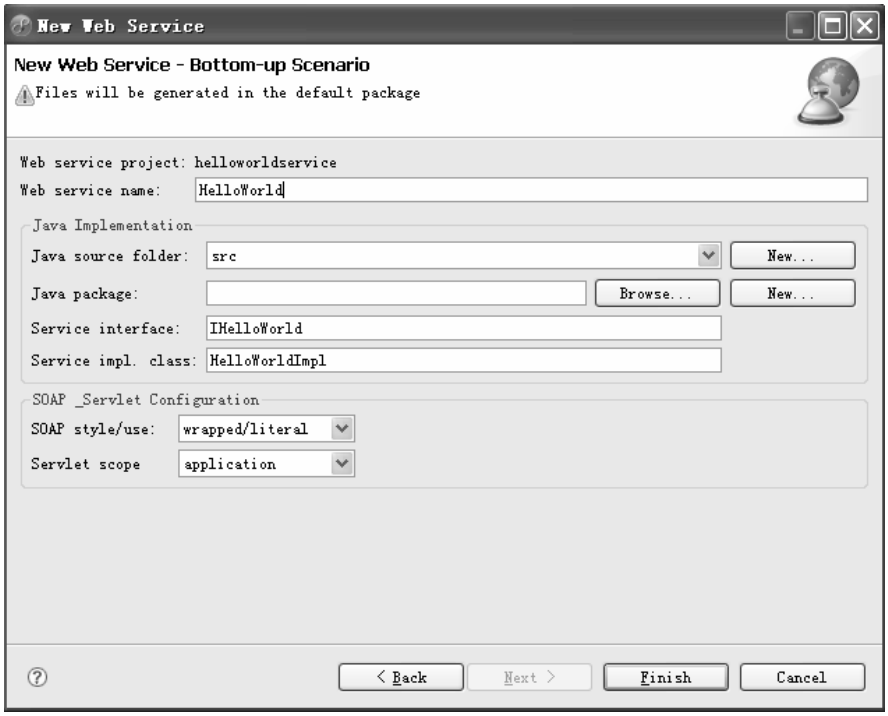


图 6-9 输入“HelloWorld”结果

此时可以看到【Service interface】命名为：“IHelloworld”；【Service impl class】命名为：“HelloWorldImpl”。

12. 点击【Java package】后的【New】，新建一个包，包命名为“hellos”。如图所示：

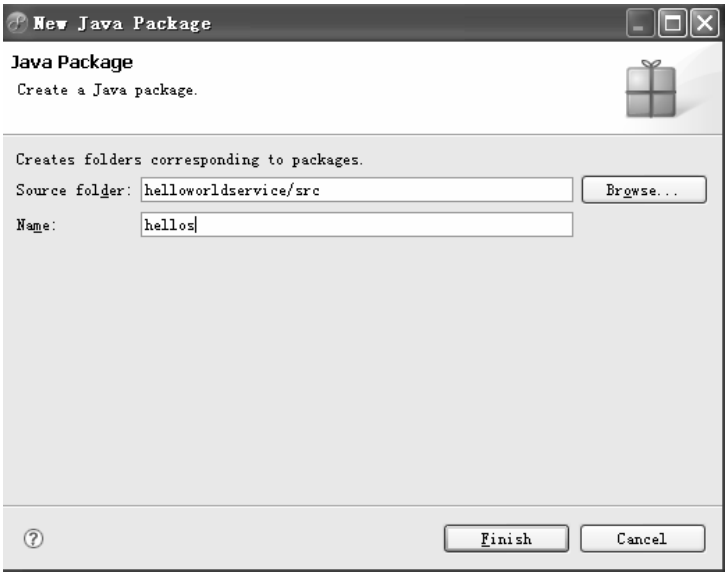


图 6-10 新建名为“hellos”包

13. 建包后结果如图所示：

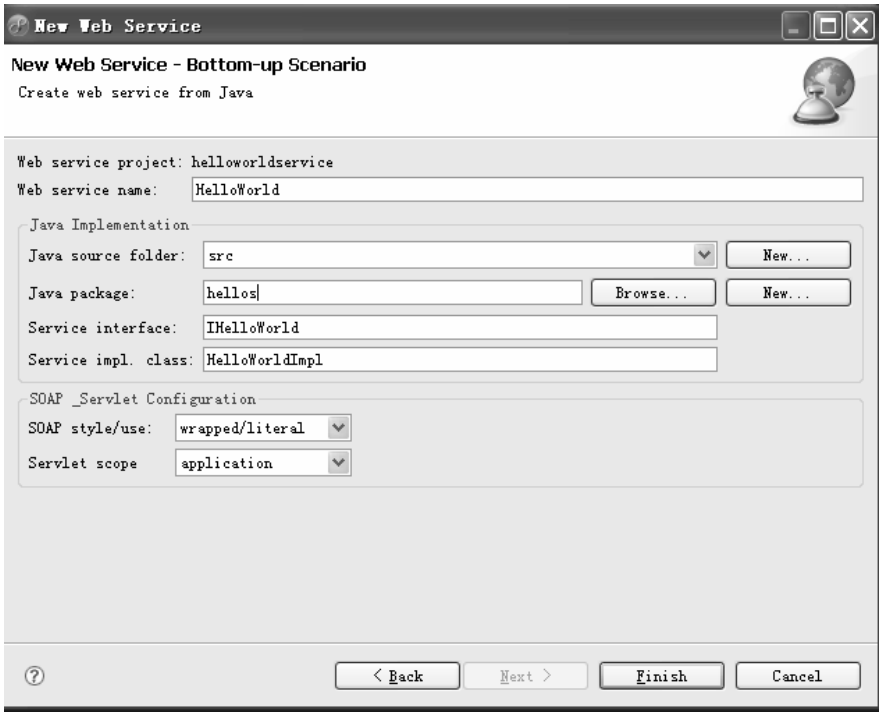


图 6-11 建包后结果视图

14. 点击【Finishi】。工程结构如图所示：

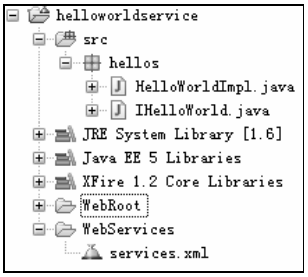


图 6-12 工程结构视图

15. 打开“HelloWorldImpl.java”文件。代码如下：

```
package hellos;
//Generated by MyEclipse

public class HelloWorldImpl implements IHelloWorld {
    public String example(String message) {
        return message;
    }
}
```

16. 打开“HelloWorld.java”文件。代码如下：

```
package hellos;
//Generated by MyEclipse

public interface IHelloWorld {

    public String example(String message);
}
```

17. 打开“services.xml”文件。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xfire.codehaus.org/config/1.0">
    <service>
        <name>HelloWorld</name>
        <serviceClass>hellos.IHelloWorld</serviceClass>
        <implementationClass>hellos.HelloWorldImpl</implementationClass>
        <style>wrapped</style>
        <use>literal</use>
        <scope>application</scope>
    </service></beans>
```

18. 将工程部署到 Tomcat 服务器上，然后启动 Tomcat。打开 IE，在浏览器地址栏输入地址：http://localhost:8080/helloworldservice/services/HelloWorld?wsdl。而后看到页面内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions                              targetNamespace="http://hellos"
xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
xmlns:tns="http://hellos"      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
 <wsdl:types>
 <xsd:schema                     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified"              elementFormDefault="qualified"
targetNamespace="http://hellos">
 <xsd:element name="example">
 <xsd:complexType>
 <xsd:sequence>
   <xsd:element   maxOccurs="1"   minOccurs="1"   name="in0"   nillable="true"
type="xsd:string" />
```

```
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
 <xsd:element name="exampleResponse">
 <xsd:complexType>
 <xsd:sequence>
    <xsd:element   maxOccurs="1"   minOccurs="1"   name="out"   nillable="true"
type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
  </wsdl:types>
 <wsdl:message name="exampleResponse">
  <wsdl:part name="parameters" element="tns:exampleResponse" />
  </wsdl:message>
 <wsdl:message name="exampleRequest">
  <wsdl:part name="parameters" element="tns:example" />
  </wsdl:message>
 <wsdl:portType name="HelloWorldPortType">
 <wsdl:operation name="example">
  <wsdl:input name="exampleRequest" message="tns:exampleRequest" />
  <wsdl:output name="exampleResponse" message="tns:exampleResponse" />
  </wsdl:operation>
  </wsdl:portType>
 <wsdl:binding name="HelloWorldHttpBinding" type="tns:HelloWorldPortType">
  <wsdlsoap:binding style="document"
                      transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="example">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exampleRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
 <wsdl:output name="exampleResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="HelloWorld">
 <wsdl:port name="HelloWorldHttpPort" binding="tns:HelloWorldHttpBinding">
                                                          <wsdlsoap:address
location="http://localhost:8080/helloworldservice/services/HelloWorld" />
```

```
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

此时证明这个 Web Service 开发成功。

18. 测试这个 Web Service。点击 MyEclipse 的【Web Services Explorer】浏览器按钮。如图所示：

图 6-13 【Web Services Explorer】浏览器按钮

19. 点击【Web Services Explorer】按钮后，在 MyEclipse 中可以看到如下视图：

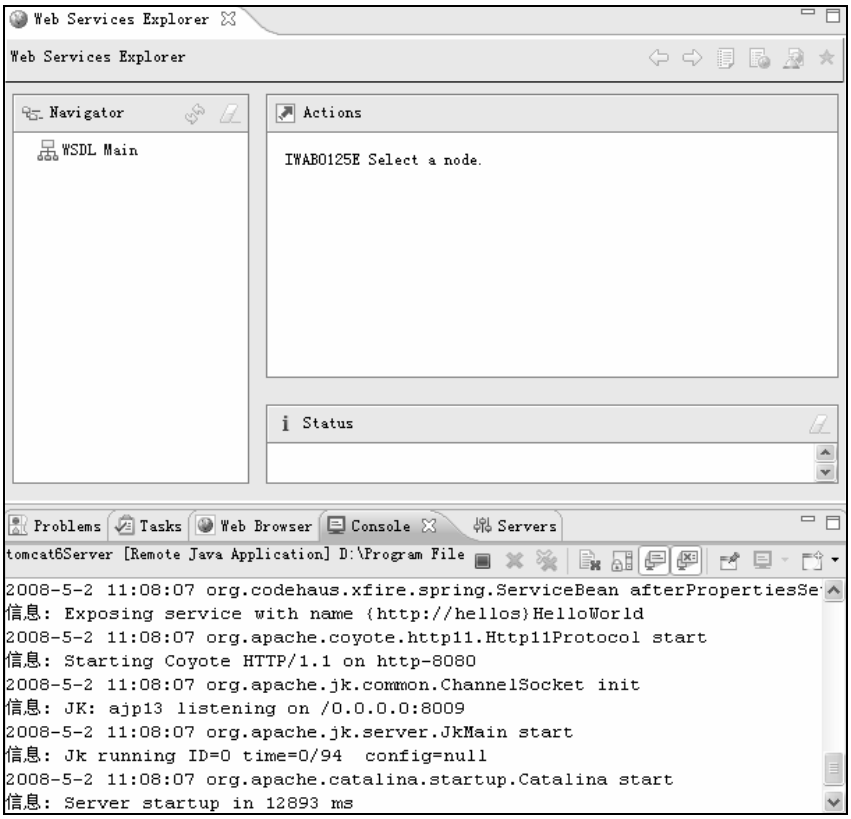


图 6-14 MyEclipse 中【Web Services Explorer】视图

20. 点击【Web Services Explorer】视图中的“WSDL Main”按钮。如图所示：

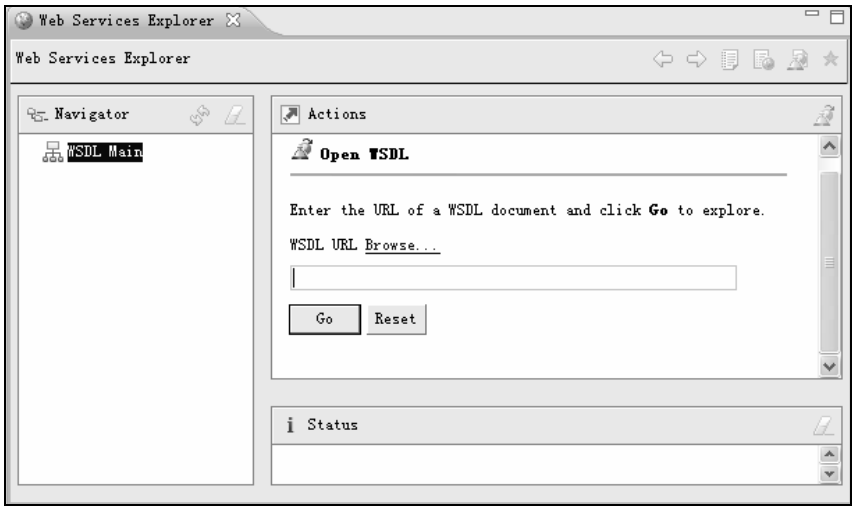


图 6-15 点击“WSDL Main”按钮后的视图

21. 在“WSDL URL Browse…”地址栏输入 http://localhost:8080/helloworldservice/services/HelloWorld?wsdl。如图所示：

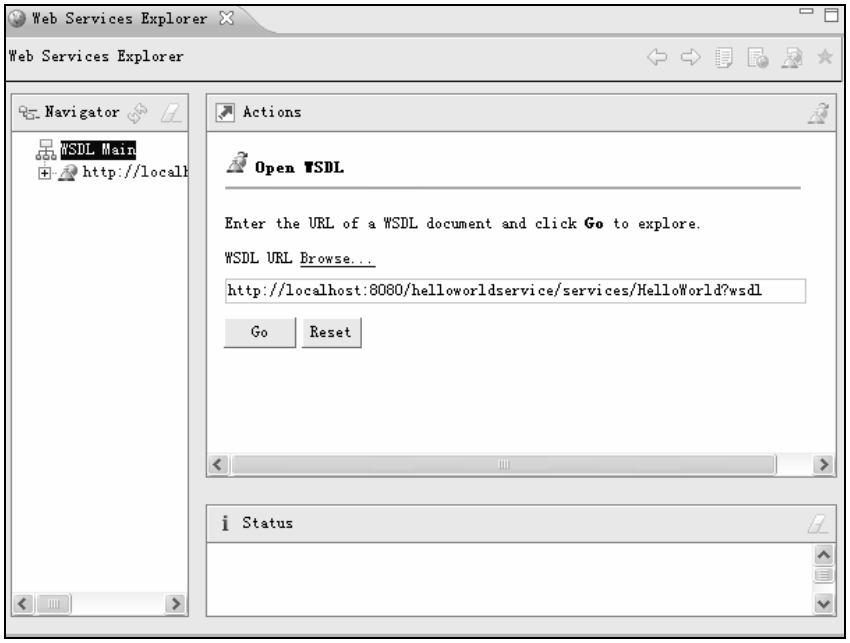


图 6-16 在“WSDL URL Browse..”输入地址

22. 点击【Go】按钮。结果如图所示：

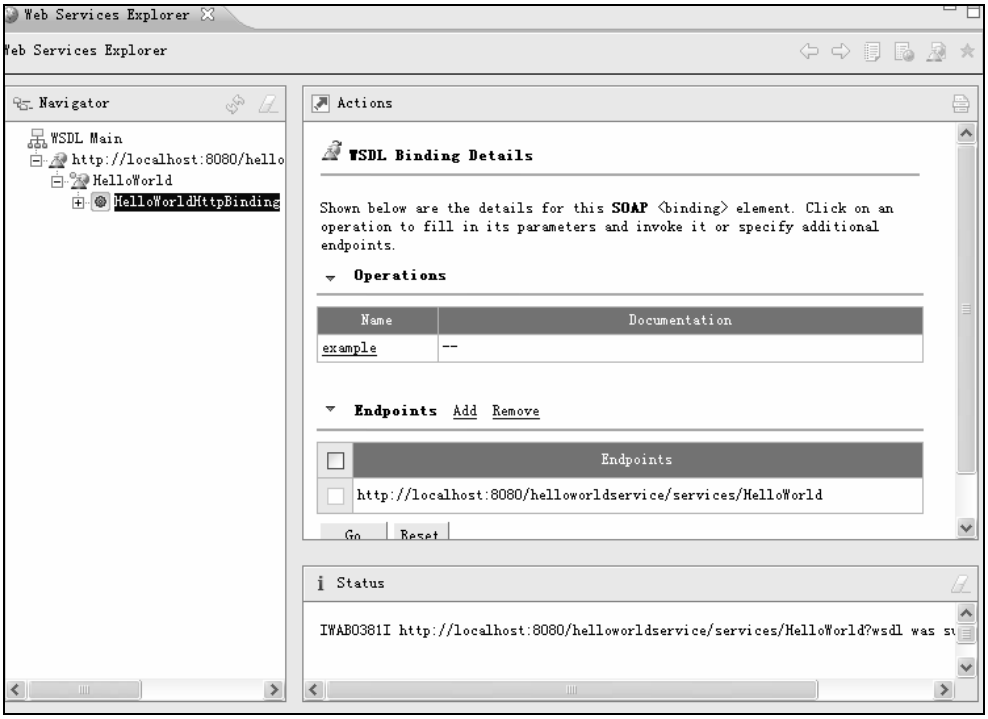


图 6-17 点击【Go】按钮后的视图

23. 点击【example】超连接。结果如图所示：

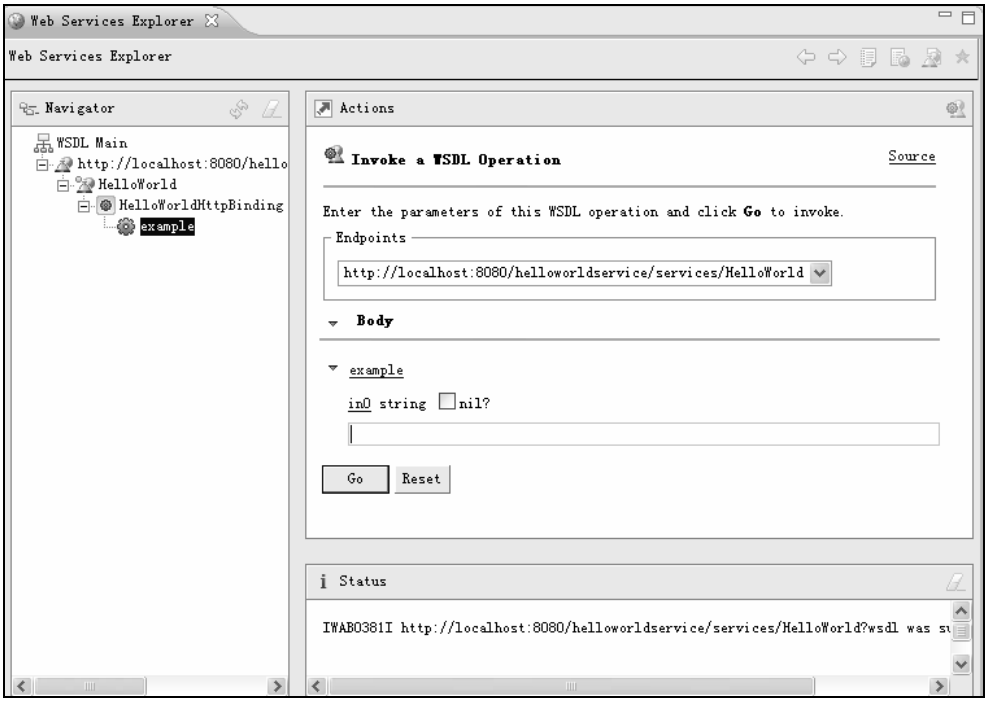


图 6-18 点击【example】超链接后的视图

24. 在【Web Services Explorer】视图的文本框中输入“java web”，点击【Go】按钮。如图所示：

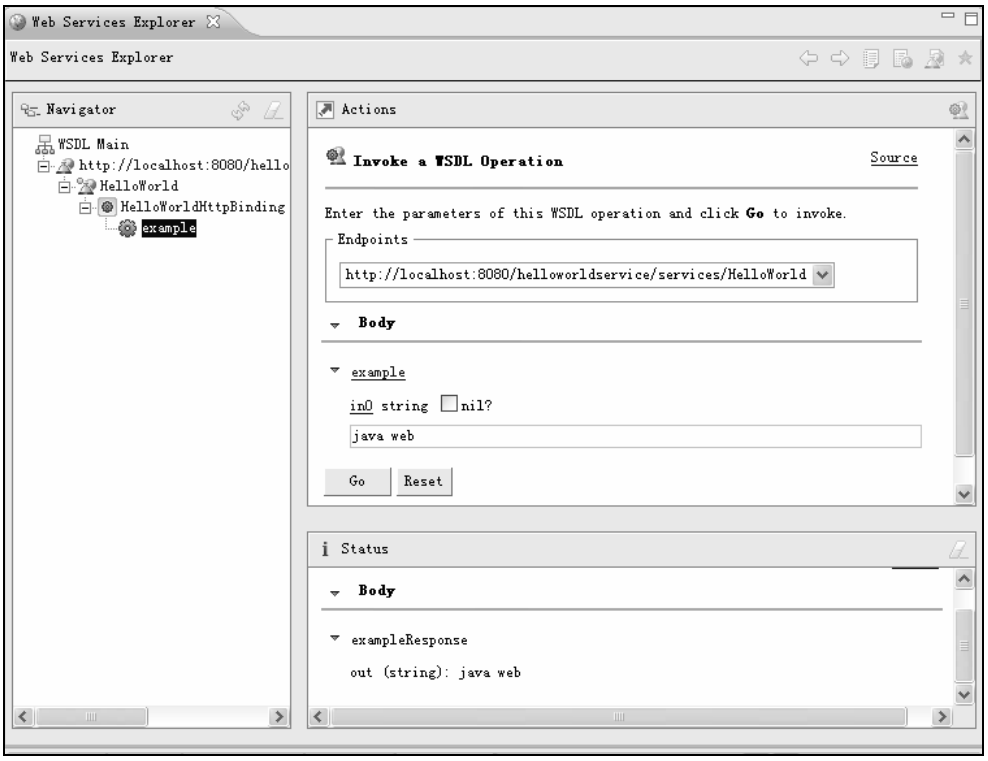


图 6-19 在文本框中输入“java web”点击【Go】按钮

此时在【Status】视图中，可以看到输出结果。这时候证明 Web Service 开发成功。

25. 建立一个工程调用 Web 服务。建立一个“Java Project”命名为“helloworldserviceclient”。如图所示：

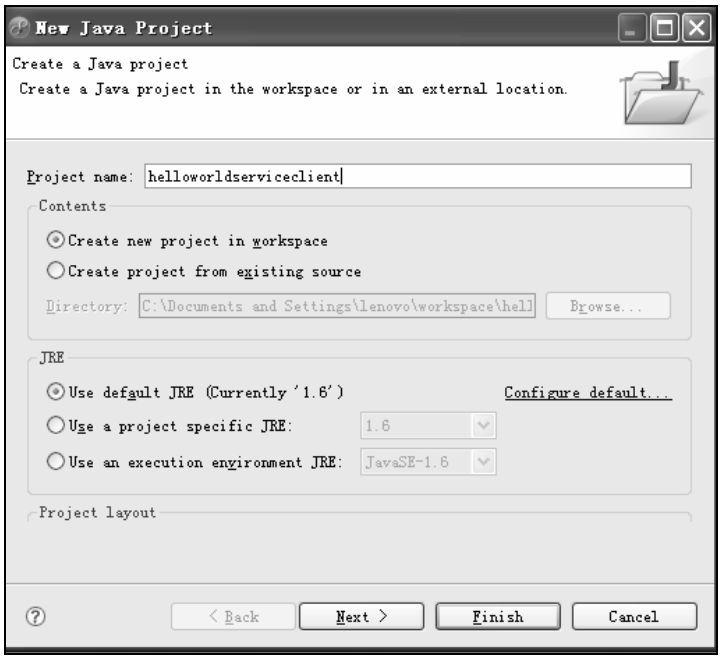


图 6-20 新建“helloworldserivceclient”工程

26. 在“helloworldserviceclient”工程中建立名为“helloclient”的包。

27. 在“helloclient”包内新建【Web Service Client】。如图所示：

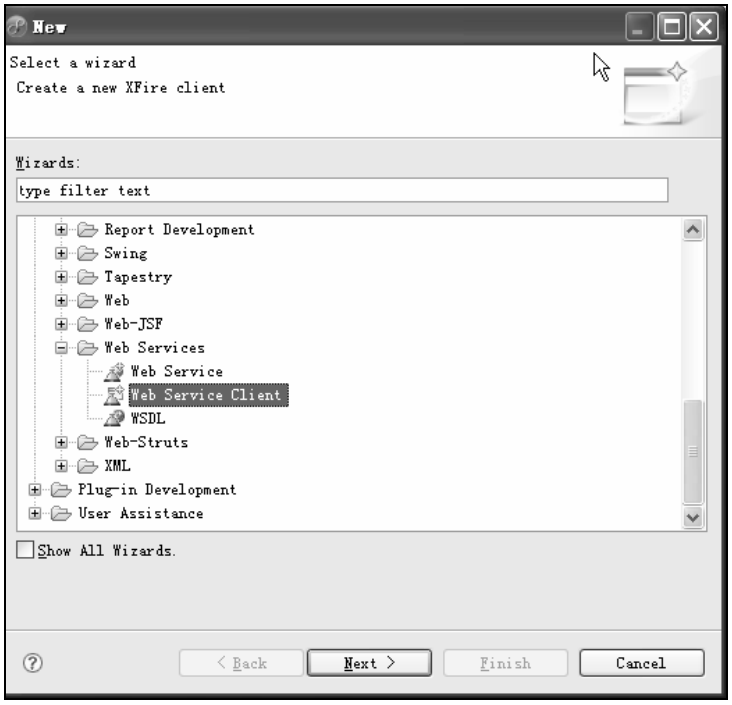


图 6-21 新建【Web Service Client】

28. 点击【Next】。结果如图所示：

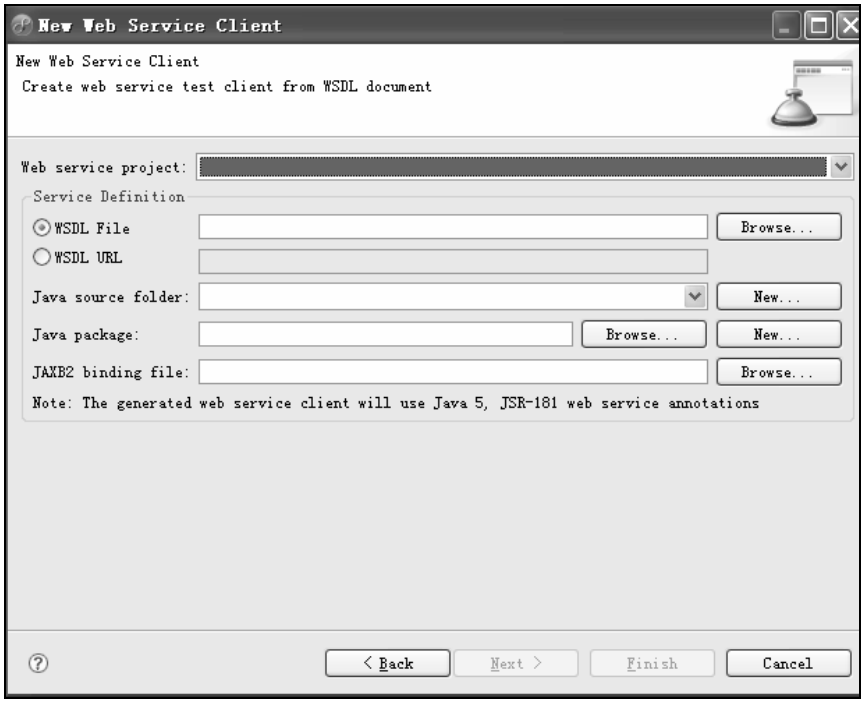


图 6-22 建立“Web Service Client”视图

29. 在【web service project】中选择，刚刚建立过的“helloworldserviceclient”工程。

30. 选中【WSDL URL】这项，在文本框中输入 http://localhost:8080/helloworldservice/services/HelloWorld?wsdl 这个地址。

31.【Java package】为“helloclient”。

32. 将各项填写完毕后，结果如图所示：

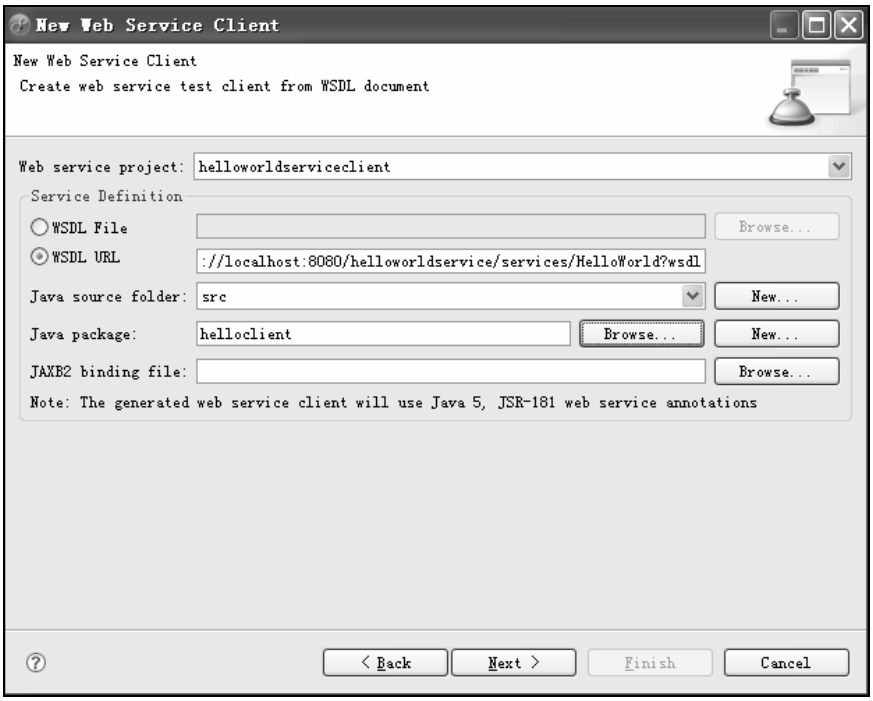


图 6-23 在【Web Service Client】中进行的配置

33. 点击【Next】。结果如图所示：

注意：此处必须保证 Tomcat 服务器处于启动状态，否则会有错误信息。

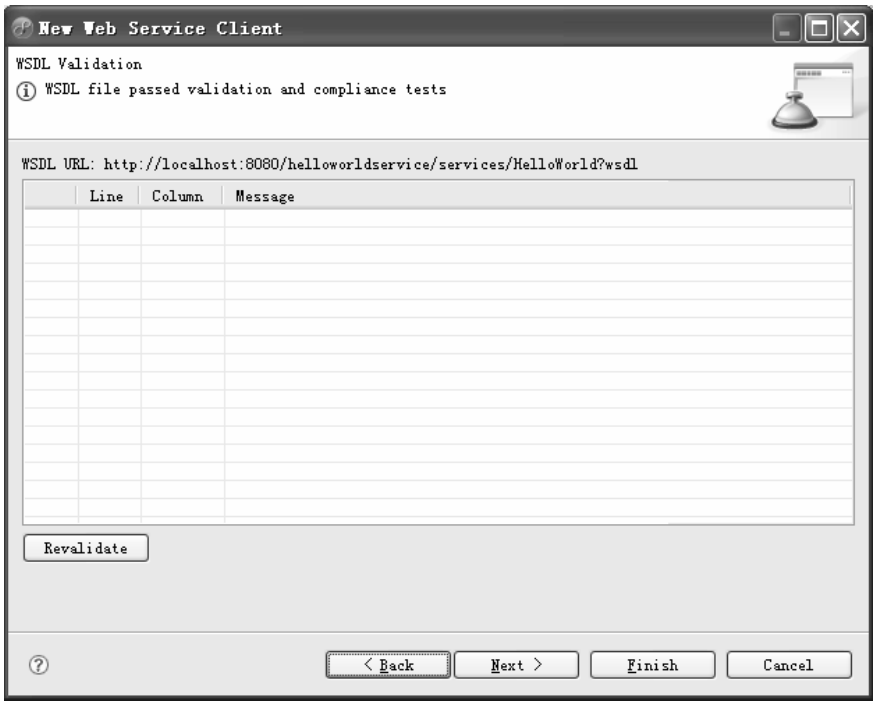


图 6-24 点击【Next】按钮的结果

34. 点击【Next】按钮。结果如图所示：

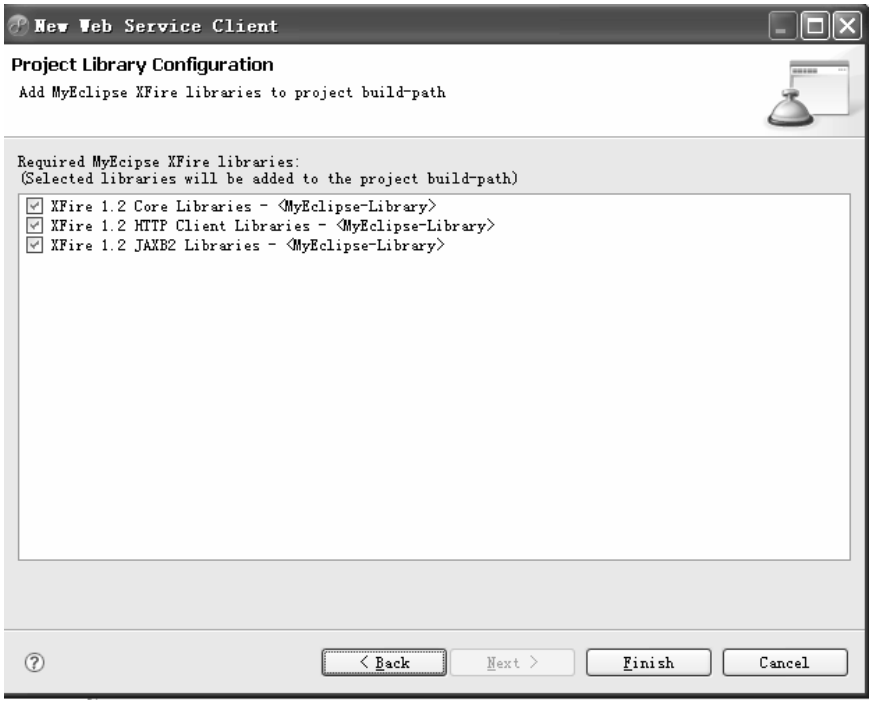


图 6-25 点击【Next】按钮结果

35. 点击【Finish】按钮。查看工程结构如图所示：

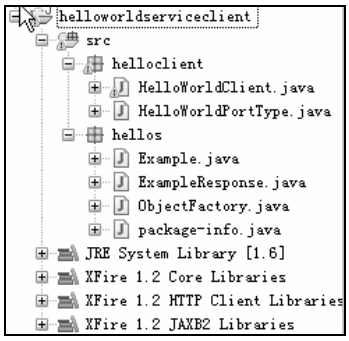


图 6-26 helloworldserviceclient 工程目录

36. 打开“HelloWorldClient.java”文件。该文件代码如下：

```
package helloclient;

import java.net.MalformedURLException;
import java.util.Collection;
import java.util.HashMap;
import javax.xml.namespace.QName;
import org.codehaus.xfire.XFireRuntimeException;
import org.codehaus.xfire.aegis.AegisBindingProvider;
import org.codehaus.xfire.annotations.AnnotationServiceFactory;
import org.codehaus.xfire.annotations.jsr181.Jsr181WebAnnotations;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.jaxb2.JaxbTypeRegistry;
import org.codehaus.xfire.service.Endpoint;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.soap.AbstractSoapBinding;
import org.codehaus.xfire.transport.TransportManager;

public class HelloWorldClient {

   private static XFireProxyFactory proxyFactory = new XFireProxyFactory();
   private HashMap endpoints = new HashMap();
   private Service service0;

   public HelloWorldClient() {
      create0();
      Endpoint HelloWorldPortTypeLocalEndpointEP = service0 .addEndpoint(new
```

```
QName("http://hellos", "HelloWorldPortTypeLocalEndpoint"), new
QName("http://hellos", "HelloWorldPortTypeLocalBinding"),
"xfire.local://HelloWorld");
       endpoints.put(new QName("http://hellos",
"HelloWorldPortTypeLocalEndpoint"), HelloWorldPortTypeLocalEndpointEP);
       Endpoint HelloWorldHttpPortEP = service0 .addEndpoint(new
QName("http://hellos",  "HelloWorldHttpPort"),  new  QName("http://hellos",
                                                  "HelloWorldHttpBinding"),
"http://localhost:8080/helloworldservice/services/HelloWorld");
       endpoints.put(new QName("http://hellos", "HelloWorldHttpPort"),
                                                    HelloWorldHttpPortEP);
   }

   public Object getEndpoint(Endpoint endpoint) {
       try {
          return proxyFactory.create((endpoint).getBinding(),
                                                     (endpoint).getUrl());
       } catch (MalformedURLException e) {
          throw new XFireRuntimeException("Invalid URL", e);
       }
   }

   public Object getEndpoint(QName name) {
       Endpoint endpoint = ((Endpoint) endpoints.get((name)));
       if ((endpoint) == null) {
          throw new IllegalStateException("No such endpoint!");
       }
       return getEndpoint((endpoint));
   }

   public Collection getEndpoints() {
       return endpoints.values();
   }

   private void create0() {
       TransportManager tm =
(org.codehaus.xfire.XFireFactory.newInstance().getXFire().getTransportManag
er());
       HashMap props = new HashMap();
       props.put("annotations.allow.interface", true);
       AnnotationServiceFactory asf = new AnnotationServiceFactory(new
Jsr181WebAnnotations(), tm, new AegisBindingProvider(new
```

```
                                                        JaxbTypeRegistry()))
;
       asf.setBindingCreationEnabled(false);
       service0 = asf.create((helloclient.HelloWorldPortType.class), props);
       {
          AbstractSoapBinding soapBinding = asf.createSoap11Binding(service0,
                      new QName("http://hellos", "HelloWorldHttpBinding"),
                               "http://schemas.xmlsoap.org/soap/http");
       }
       {
          AbstractSoapBinding soapBinding = asf.createSoap11Binding(service0,
            new QName("http://hellos", "HelloWorldPortTypeLocalBinding"),
                                              "urn:xfire:transport:local");
       }
   }

   public HelloWorldPortType getHelloWorldPortTypeLocalEndpoint() {
   return ((HelloWorldPortType)(this).getEndpoint(new QName("http://hellos",
"HelloWorldPortTypeLocalEndpoint")));
   }

   public HelloWorldPortType getHelloWorldPortTypeLocalEndpoint(String url) {
       HelloWorldPortType var = getHelloWorldPortTypeLocalEndpoint();
       org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);
       return var;
   }

   public HelloWorldPortType getHelloWorldHttpPort() {
   return ((HelloWorldPortType)(this).getEndpoint(new QName("http://hellos",
"HelloWorldHttpPort")));
   }

   public HelloWorldPortType getHelloWorldHttpPort(String url) {
       HelloWorldPortType var = getHelloWorldHttpPort();
       org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);
       return var;
   }

   public static void main(String[] args) {


       HelloWorldClient client = new HelloWorldClient();
```

```
        //create a default service endpoint
       HelloWorldPortType service = client.getHelloWorldHttpPort();

        //TODO: Add custom client code here
            //
            //service.yourServiceOperationHere();
        System.out.println("test client completed");
            System.exit(0);
    }
}
```

37. 在“HelloWorldClient.java”中的 main 方法中加入测试代码。代码如下：

```
public static void main(String[] args) {



       HelloWorldClient client = new HelloWorldClient();

        //create a default service endpoint
       HelloWorldPortType service = client.getHelloWorldHttpPort();

        //TODO: Add custom client code here
        //
       //service.yourServiceOperationHere();
      System.out.println(service.example("这是我的第一个 webservice 测试程序"));
        System.out.println("test client completed");
            System.exit(0);
    }
```

38. 运行“HelloWorldClient.java”，在控制台可以看到输出结果如下：

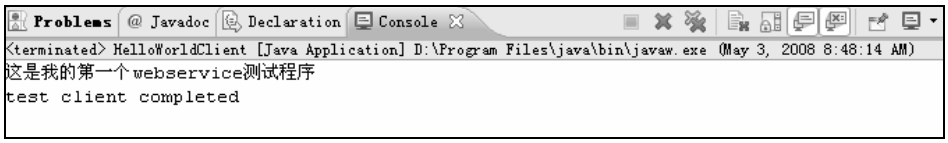


图 6-27 “HelloWorldClient.java”运行结果

通过以上步骤完成了一个简单的 Web Service 的开发与应用。

4. 相关理论知识

4.1 WebService 起源

Web 应用的巨大成功和不断发展，使其渗透到商业领域和个人生活的各个方面。人们只

要使用浏览器，就可以享受到各种各样的 Web 服务，例如网上购物、网上交易、网络游戏、预定车票、网上聊天和交友等等。与此同时，由于 Web 技术所带来的优势（统一的客户端和较好的维护性），使一些传统的应用纷纷转型到 B/S 结构上。

然而，在发展中，逐步暴露了一些问题。所有这些 Web 页面都是为人准备的，是让人去阅读、去输入、去判断。因此各种反映视觉效果的内容占用了大量的网络带宽，例如各种图片，字体信息，文字排版样式等。而真正含有高价值的一些信息，被深深埋在这些显示信息中，很难被其他应用和程序所使用。更重要的是，各种 web 服务之间缺少交互和通讯的机制。

程序之间的互相通讯很重要吗？简单举一个例子。

假设你经常去国外出差，在你回国以后，第一件事就是费用报销了。而你们公司有这样的财务规定，所有的报销款，都按报销当天的外汇比价进行结算。因此在你填写报销单的时候必须先填写每一笔在各个国家的花销，然后上网查出当天的外汇比价，填写到报销单上。剩下的事情也就不用你做了，你的报销单填写工具会自动进行换算和统计。

觉得有什么不妥吗？作为 IT 公司的员工，也许都有一个特点，计算机能做的事情，尽量要计算机去做。外汇比价的查询可以让计算机自动去做嘛！然而，让你的程序自动去网页上查找指定的外汇比价可不是一件容易的事。因为这些网页是给人阅读的，人眼和大脑的反应速度有多快，它们可以从一整页信息中快速定位到你所要的内容，而且无论网页怎样变化和改版都不会带来太大的影响。而应用程序想要做同样的事就差得太远了。因此，现在需要的是专门为应用程序制定的 Web 服务。

随着应用程序之间通讯的需求越来越大，这就需要制定统一的标准和协议。HP 公司是最先提出这个观点的公司，他们制定了有关“e-Speak”的标准来保证应用程序之间的交互，并声称将成为下一代 Internet 信息交互的标准。而随后，MicroSoft 意识到此计划的美好前景，便推出了.Net 战略；IBM 很快就发布了 Web Services Toolkit(WSTK)，和 Web Services Development Environment(WSDE)，申明对 Web Services 的全力支持。与此同时，Oracle 也开发出自己的 Dynamic Services，并和 Oracle 8i Release 2 集成在一起。在这以后，W3C 统一制定了 Web Services 的各种标准。而 SUN 公司在宣布了自己的 Web Services 的框架以后，将 Web Services 的标准溶入 J2EE 的环境，使 Web Services 有了广泛支持的基础和平台。

4.2 WebService 概念

使用 Web 服务技术，应用程序可以与平台和编程语言无关的方式相互通信。Web 服务是一个软件接口，它描述了一组可以在网络上通过标准化的 XML 消息传递访问的操作。它使用基于 XML 语言的协议来描述要执行的操作或者要与另一个 Web 服务交换的数据。在面向服务的体系结构（Service-Oriented Architecture，SOA）中，一组以这种方式交互的 Web 服务定义了特定的 Web 服务应用程序。

软件业最终会接受这样的事实：跨多个操作系统、编程语言和硬件平台集成软件应用程序不可能由任何一种专门的环境来解决。传统上，这个问题一直是一个紧耦合问题，调用远程网络的应用程序通过自己发出的函数调用和请求的参数与远程网络紧密地联系在一起。在 Web 服务出现之前，在大多数系统上，采用的是固定的接口，但对于不断变化的环境或需求，这样做缺乏灵活性或适用性。

即 WebService 提供了基于文本（实际上是基于 XML）的方法来往返传递消息，这意味着应用程序不仅不依赖计算机，而且也不依赖于操作系统和编程语言。只要双方遵循相同的

Web 服务标准，就不受各方运行何种软件的影响。

Webserviece 能够做什么

- 让任何平台上的用任何语言编写的服务进行交互。
- 将应用程序功能概念化成任务，从而形成面向任务的开发和工作流。这使得更抽象的软件能够为工作在业务层面具有较少软件分析技术的用户所用。
- 允许松耦合，这意味着，每当其中某个或多个服务在设计或实现中发生改变时，服务应用程序之间的交互不会因此而中断。
- 使现有的应用程序能适应不断变化的业务条件和客户需求。
- 向现有或遗留软件应用程序提供服务接口，而无需改变原来的应用程序，从而使这些应用程序完全可以运行在这种服务环境下。
- 引入其他一些与原有功能无关的管理或操作管理功能，比如可靠性、责任性和安全性等等，从而在业务计算环境中增加其通用性和实用性。

4.3 WebService 工作原理

Web Services 是通过一系列标准和协议来保证程序之间的动态链接。其中最基本的协议包括：SOAP，WSDL，UDDI

SOAP：是“Simple Object Access Protocol”的缩写，SOAP 是消息传递的协议，它规定了 Web Services 之间是怎样传递信息的。简单的说，SOAP 规定了：

1. 传递信息的格式为 XML。这就使 Web Services 能够在任何平台上，用任何语言进行实现。

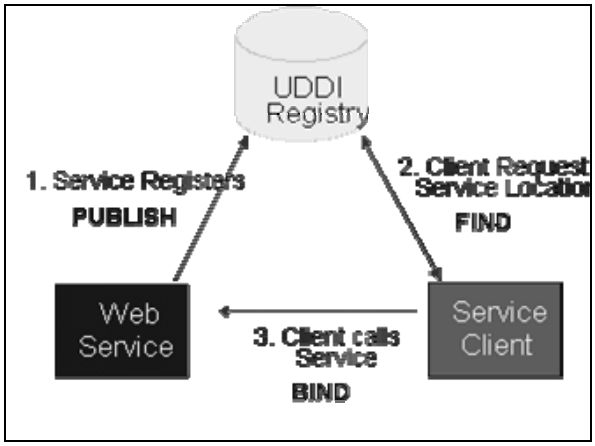
2. 远程对象方法调用的格式。规定了怎样表示被调用对象以及调用的方法名称和参数类型等。

3. 参数类型和 XML 格式之间的映射。这是因为，被调用的方法有时候需要传递一个复杂的参数，例如，一个 Person 对象。怎样用 XML 来表示一个对象参数，也是 SOAP 所定义的范围。

4. 异常处理以及其他的相关信息.

WSDL：是“Web Services Description Language”的缩写。意如其名，WSDL 是 Web Services 的定义语言。当你实现了某种服务的时候（如：股票查询服务），为了让别的程序调用，你必须告诉大家你的服务的接口。例如：服务名称、服务所在的机器名称、监听端口号、传递参数的类型、个数和顺序、返回结果的类型等等。这样别的应用程序才能调用你的服务。WSDL 协议就是规定了有关 Web Services 描述的标准。

UDDI：是 Universal Description, Discovery and Integration 的缩写。简单说，UDDI 用于集中存放和查找 WSDL 描述文件，起着目录服务器的作用。



如上图，一个 Web Services 的生命周期是：

实现一个 Web Services，使其能够接受和响应 SOAP 消息（现在有很多工具都可以帮助实现）。

撰写一个 WSDL 文件用于描述此 Web Services。（现在有很多工具可以自动生成 WSDL 文件）。

将此 WSDL 发布到 UDDI 上。

其他的应用程序（客户端）从 UDDI 上搜索到你的 WSDL。

根据你的 WSDL，客户端可以编写程序（现在有很多工具可以自动生成调用程序）调用你的 Web Services。

4.4 什么是 SOAP

SOAP 把 XML 的使用代码化为请求和响应参数编码模式，并用 HTTP 作传输。这似乎有点抽象。具体地讲，一个 SOAP 方法可以简单地看作遵循 SOAP 编码规则的 HTTP 请求和响应。一个 SOAP 终端则可以看作一个基于 HTTP 的 URL，它用来识别方法调用的目标。

SOAP 请求是一个 HTTP POST 请求。SOAP 请求的 content-type 必须用 text/xml。而且它必须包含一个请求-URI。服务器怎样解释这个请求-URI 是与实现相关的，但是许多实现中可能用它来映射到一个类或者一个对象。一个 SOAP 请求也必须用 SOAPMethodName HTTP 头来指明将被调用的方法。简单地讲，SOAPMethodName 头是被 URI 指定范围的应用相关的方法名，它是用#符作为分隔符将方法名与 URI 分割，如下：

```
SOAPMethodName: urn:strings-com:IString#reverse
```

这个头表明方法名是 reverse，范围 URI 是 urn:strings-com:Istring。在 SOAP 中，规定方法名范围的名域 URI 在功能上等同于在 DCOM 或 IIOP 中规定方法名范围的接口 ID。

简单的说，一个 SOAP 请求的 HTTP 体是一个 XML 文档，它包含方法中[in]和[in,out]参数的值。这些值被编码成为一个显著的调用元素的子元素，这个调用元素具有 SOAPMethod Name HTTP 头的方法名和名域 URI。调用元素必须出现在标准的 SOAP＜Envelope＞和＜Body＞元素。下面是一个最简单的 SOAP 方法请求：

```
POST /string_server/Object17 HTTP/1.1
Host: 209.110.197.2
Content-Type: text/xml
Content-Length: 152
SOAPMethodName: urn:strings-com:IString#reverse
<Envelope>
<Body>
<m:reverse xmlns:m= ‘urn:strings-com:IString ‘>
<theString>Hello, World</theString>
</m:reverse>
</Body>
</Envelope>
```

SOAPMethodName 头必须与<Body>下的第一个子元素相匹配，否则调用将被拒绝。这允许防火墙管理员在不解析 XML 的情况下有效地过滤对一个具体方法的调用。

SOAP 响应的格式类似于请求格式。响应体包含方法的[out]和[in,out]参数，这个方法被编码为一个显著的响应元素的子元素。这个元素的名字与请求的调用元素的名字相同，但以 Response 后缀来连接。下面是对前面的 SOAP 请求的 SOAP 响应：

```
200 OK
Content-Type: text/xml
Content-Length: 162
<Envelope>
<Body>
<m:reverseResponse xmlns:m= ‘urn:strings-com:IString ‘>
<result>dlroW ,olleH</result>
</m:reverseResponse>
</Body>
</Envelope>
```

这里响应元素被命名为 reverseResponse，它是方法名紧跟 Response 后缀。要注意的是这里是没有 SOAPMethodName HTTP 头的。这个头只在请求消息中需要，在响应消息中并不需要。

4.5 什么是 WSDL

Web Services Description Language 的缩写，是一个用来描述 Web 服务和说明如何与 Web 服务通信的 XML 语言。

它来描述下列问题的答案：

- 您的在线业务提供什么服务？
- 您如何调用业务服务？
- 当用户调用您的业务服务时，该业务服务需要他 / 她提供什么信息？
- 用户将如何提供这些必需信息？

- 服务将以什么格式发送返回给用户的信息？

它主要由五个要素构成：

Types：定义 WSDL 定义中所用到的数据类型，即 XML Schema Types。

Message：对一组消息的输入和输出参数的定义。

portType：定义 Web 服务的操作。

Binding：描述特定服务接口的协议、数据格式、安全性和其它属性。

Services：制定特定服务的 URL 和提供的调用接口，包含一组端口元素。

这样可以分析示例中的 WSDL 文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions                          targetNamespace="http://hellos"
xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
xmlns:tns="http://hellos"      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
 <wsdl:types>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         attributeFormDefault="qualified" elementFormDefault="qualified"
                                          targetNamespace="http://hellos">
 <xsd:element name="example">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element maxOccurs="1" minOccurs="1" name="in0" nillable="true"
                                                    type="xsd:string" />
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
 <xsd:element name="exampleResponse">
<xsd:complexType>
 <xsd:sequence>
 <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
                                                     type="xsd:string" />
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
 </xsd:schema>
 </wsdl:types>
```

```
 <wsdl:message name="exampleResponse">
 <wsdl:part name="parameters" element="tns:exampleResponse" />
 </wsdl:message>
 <wsdl:message name="exampleRequest">
 <wsdl:part name="parameters" element="tns:example" />
 </wsdl:message>
 <wsdl:portType name="HelloWorldPortType">
 <wsdl:operation name="example">
 <wsdl:input name="exampleRequest" message="tns:exampleRequest" />
 <wsdl:output name="exampleResponse" message="tns:exampleResponse" />
 </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="HelloWorldHttpBinding" type="tns:HelloWorldPortType">
  <wsdlsoap:binding style=
            "document" transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="example">
 <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exampleRequest">
 <wsdlsoap:body use="literal" />
 </wsdl:input>
 <wsdl:output name="exampleResponse">
 <wsdlsoap:body use="literal" />
 </wsdl:output>
 </wsdl:operation>
 </wsdl:binding>
<wsdl:service name="HelloWorld">
 <wsdl:port name="HelloWorldHttpPort" binding="tns:HelloWorldHttpBinding">
 <wsdlsoap:address
location="http://localhost:8080/helloworldservice/services/HelloWorld" />
 </wsdl:port>
 </wsdl:service>
 </wsdl:definitions>
```

4.6 什么是 UDDI

Universal Description Discovery and Integration 即统一描述、发现和集成协议。

UDDI 始于 2000 年,由 Ariba, IBM, Microsoft 和其他 33 家公司创立.UDDI registries 提供了一个机制，以一种有效的方式来浏览，发现 Web Services 以及它们之间的相互作用。

UDDI 计划是一个广泛的，开放的行业计划，它使得商业实体能够（1）彼此发现，（2）定义他们怎样在 internet 上互相作用，并在一个全球的注册体系架构中共享信息。UDDI 是这样一种基础的系统构筑模块，他使商业实体能够快速，方便地使用他们自身的企业应用软件来发现合适的商业对等实体，并与其实施电子化的商业贸易。

UDDI 同时也是 Web 服务集成的一个体系框架。它包含了服务描述与发现的标准规范。UDDI 规范利用了 W3C 和 Internet 工程任务组织（IETF）的很多标准作为其实现基础，比如扩展标注语言（XML），HTTP 和域名服务（DNS）这些协议。另外，在跨平台的设计特性中，UDDI 主要采用了已经被提议给 W3C 的 SOAP（Simple Object Access Protocol，简单对象访问协议）规范的早期版本。

4.7 什么是 Axis 引擎

Apache Axis 是 Apache Web Service 项目中的子项目之一，它是 Apache SOAP 项目的延续。Axis 的主要功能是作为一个 SOAP 的实现来让开发者通过它来构建自己的 Web Ser_vice（支持 Java 和 C++）。随着 Web Services 以及 SOA 的不断发展，越来越多的人投入到了相关技术的实现、开发和标准的制定的工作中。Apache Axis 凭借它强大的功能和稳定性倍受开发人员的青睐，成为了用于实现 Web Services 的主要途径。

Axis 总体上是一个 SOAP 引擎，它提供了创建 SOAP 客户机、服务器端、网关 SOAP 的基本框架。Axis 是 SOAP 的后续版本（该项目的前身是 IBM 的“SOAP4J”），我们介绍的 Axis 版本是 Axis1.1for Java 版，这个版本是 java 的 Web Service 实现。

Axis 除了提供 SOAP 引擎之外，它还是一个可插入到 servlet 引擎中的独立的服务器，并且还提供了以下工具：将 WSDL 转化为 java 类的工具和一个 TCP/IP 数据包监视工具。

Axis1.1 for Java 版本相比它的以前版本有了重大的改进，这些改进让 Axis 灵活性更好、管理更方便、同时支持 SOAP 和 W3C 中 XML 的新规范，Axis1.1 for Java 提供了以下重要特性：

1. 速度提高

Axis1.1 for Java 版本使用了 SAX（事件驱动）来解析 XML 文档。

2. 灵活

Axis1.1 for Java 版允许用户灵活地定制扩展 Axis 的功能，用户可以加入他们想要增加的任何特性：个性化的信息头处理、扩展的系统管理和其他特性。

3. 稳定性提高

Axis 中定义了一套稳定的发布接口（publicshed interface），他们不会随着 Axis 的更新而更新，除非相关的标准发生了改变，这样减少了 Axis 新发布版本中发生重构时对用户的影响。比如 axis 中处理 JAVA-RPC 的接口就是一套稳定的发布接口，它支持 Java-RPC1.1 规范，除非 Java-RPC 规范发生改变，否则这些接口都不会发生变化。

4. 提供面向组件的部署

用户可以轻松地建立可重用的网络环境来实现处理工作的一些模式，或者将这些发布给自己的合作伙伴。

5. 提供一个简洁的传输抽象框

Axis1.1 版本中的网络传输设计中使用彻底的、简化的抽象，SOAP 消息的发送方和接收方可以使用多种传输协议比如 SMTP、FTP 或者消息中间件等，Axis1.1 版本的核心引擎完全是独立于传输协议的，从而使用户可以更加灵活地选择用何种协议来传输。

6. 支持 WSDL1.1

Axis1.1 版本支持 WSDL1.1 版本，用户通过使用这个特性可以非常简单地创建访问远程的 Web 服务需要用到的类文件，也可以使用该特性从你的 Java 类中自动生成正确的 WSDL 文档。

Axis 现在的版本是 2.0，基于篇幅的问题，具体的使用实现在这里不做过多的介绍。我们主要介绍另一种引擎 XFire。

4.8 XFire 构建 WebService 步骤

XFire 是新一代的 Java Web 服务引擎，XFire 使得在 JavaEE 应用中发布 Web 服务变得轻而易举。和其他 Web 服务引擎相比，XFire 的配置非常简单，可以非常容易地和 Spring 集成，它使得 Java 开发人员终于可以获得和.Net 开发人员一样的开发效率。

由于使用了 STAX（the Streaming API for XML，基于流的 XML 解析）作为 XML 解析器，XFire 的运行速度又有了质的提高，并且 XFire 支持最新的 JSR 181 的 Web 服务注解。如果使用 Java5，只需要在源代码中编写相应的 JSR 181 注解，XFire 就可以根据 Java5 注解自动提取所需的全部信息。由于 JSR 181 也是 JavaEE Web 服务标准的一部分，使用它最大的好处在于不仅极大地简化了配置，而且避免了配置文件和某个特定的 Web 服务引擎的锁定

- 支持多个重要的 WebService 标准,包括 SOAP、WSDL、WS-I BasicProfile、WS-Addressing、WS-Security 等。
- 高性能的 SOAP 栈。
- 可选的绑定方式，如 POJO、XMLBeansJAXB1.1、JAXB2.0、Castor 和 JiBX 等。
- 支持 JSR181 API
- 多种传输方式，如 HTTP、JMS、XMPP、In-JVM 等。
- 灵活的接口
- 支持多个容器，如 Spring、Pico、Plexus、Loom。
- 支持 JBI
- 客户端和服务器代码生成。

在 MyEclipse 下使用 XFire 构建 WebService 步骤如下：

1. 建立 WebService 项目，添加 XFire 库。如图 14-1 到 14-5
2. 配置 web.xml 文件。
3. 在项目中建立一个 WebService，创建相关接口和实现类。如图 14-8 到 14-13
4. 配置 services.xml。
5. 测试 Web 服务是否发布成功。
6. 编写客户端程序访问创建的 WebService。如图 14-21 到 14-28

5. 实验

1. 完成实践知识的示例。（40 分钟）
2. 使用 WebService 开发一个简易计算器。（50 分钟）

6. 作业

使用 WebService 做一个获取生日信息的服务。

提示：客户输入身份证号，取出其出生的年月。