第6章 Struts 2、Hibernate和Spring整合应用

- 6.1 项目创建及功能描述
- 6.2 层次划分
- 6.3 添加Spring开发
- 6.4 实现Hibernate持久层
- 6.5 实现DAO
- 6.6 实现业务逻辑层
- 6.7 实现Web层

6.1 项目创建及功能描述

创建一个Web项目，命名为“xscjManage”。该项目要实现学生、课程及成绩的增加、删除、修改、查找功能，需要三个表，即XSB表、KCB表、CJB表。其中XSB表中含有该学生所属专业的ID，且作为外键，故还应该有一个ZYB表。在登录学生成绩管理系统时，如果没有登录成功，就回到登录界面，登录成功后方可进行各种操作，所以还要有个DLB表。具体的表结构及数据准备见附录A。

选择填空—（40），简答（40），编程（20）

6.2 层次划分

Java EE的体系结构有三层：表示层、业务逻辑层和数据持久层。开发一个SSH（Struts，Hibernate，Spring）项目，要遵循这三层模式。根据前面知识的学习，可以分别用SSH实现这样的目的：用Hibernate来完成数据的持久层应用，用Spring的Bean来管理组件（主要是DAO、业务逻辑和Struts的Action），而用Struts来完成页面的控制跳转。该项目完成后的业务逻辑层及数据池目录如图6.1所示。

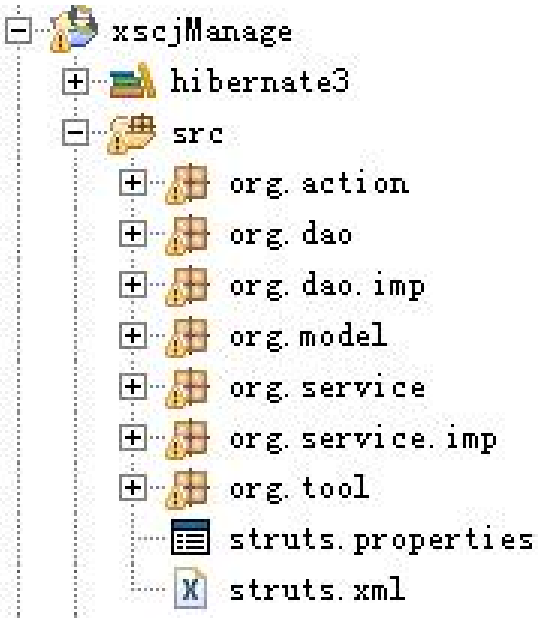

图6.1 项目部分目录

6.2 层次划分

- org.action：放置对应的用户自定义的Action类。由Action类调用业务逻辑来处理用户请求，然后控制跳转。
- org.dao：放置DAO（数据访问对象）的接口，接口中的方法用来和数据库进行交互，这些方法由实现它们的类来实现。
- org.dao.imp：放置实现DAO接口的类。
- org.model：放置表对应的POJO类及映射文件*.hbm.xml。
- org.service：放置业务逻辑接口。接口中的方法用来处理用户请求，这些方法由实现接口的类来实现。
- org.service.imp：放置实现业务逻辑接口的类。
- org.tool：放置公用的工具类，如分页类。
- struts.properties：实现Struts 2和Spring整合。
- struts.xml：配置Action。

6.3 添加Spring开发

在添加Spring开发能力之前先介绍如何自定义User Labraries。右击项目名xscjManage，选择【Build Path】→【Configure Build Path】菜单项，出现如图6.2所示的对话框。

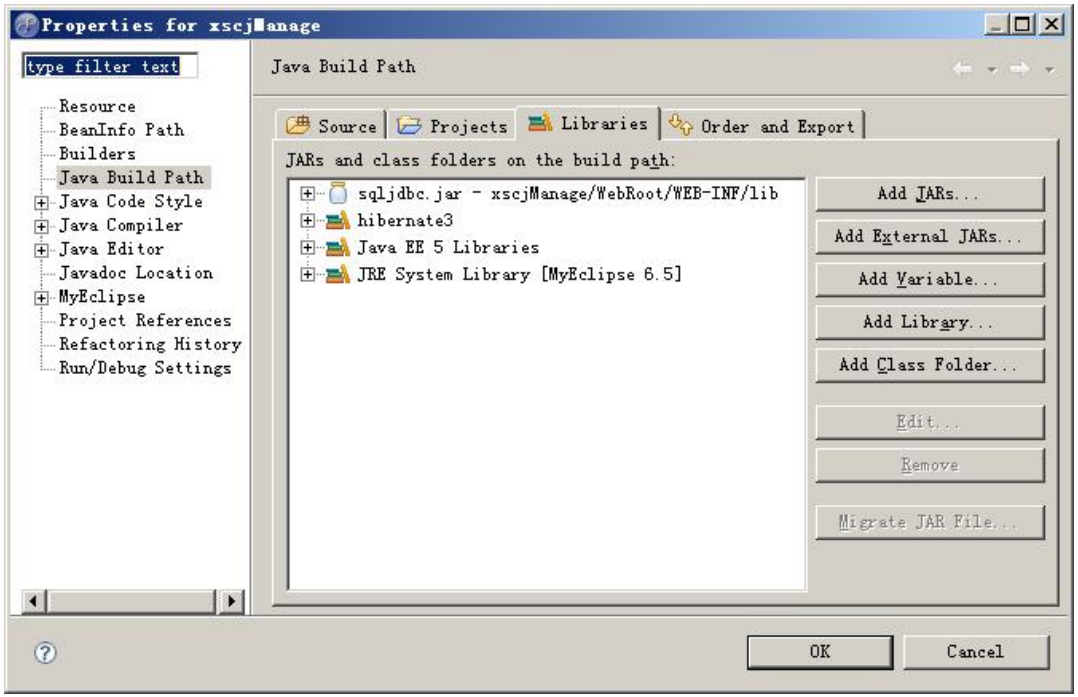

图6.2 Properties for xscjManage对话框

6.3 添加Spring开发

然后单击【Add Library...】按钮，出现如图6.3所示的对话框。

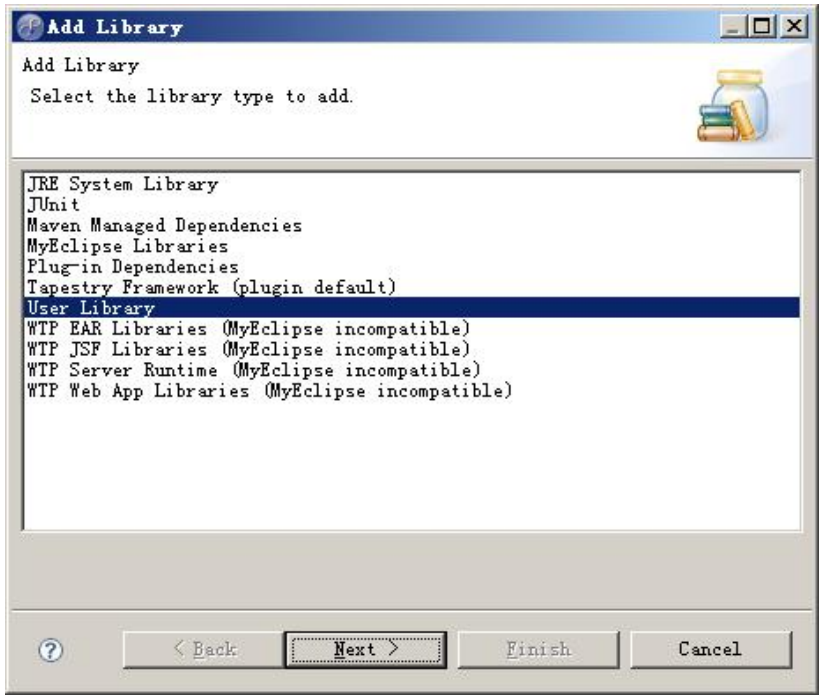

图6.3 Add Library对话框

6.3 添加Spring开发

选中【User Library】后，单击【Next】按钮，出现如图6.4所示的对话框。

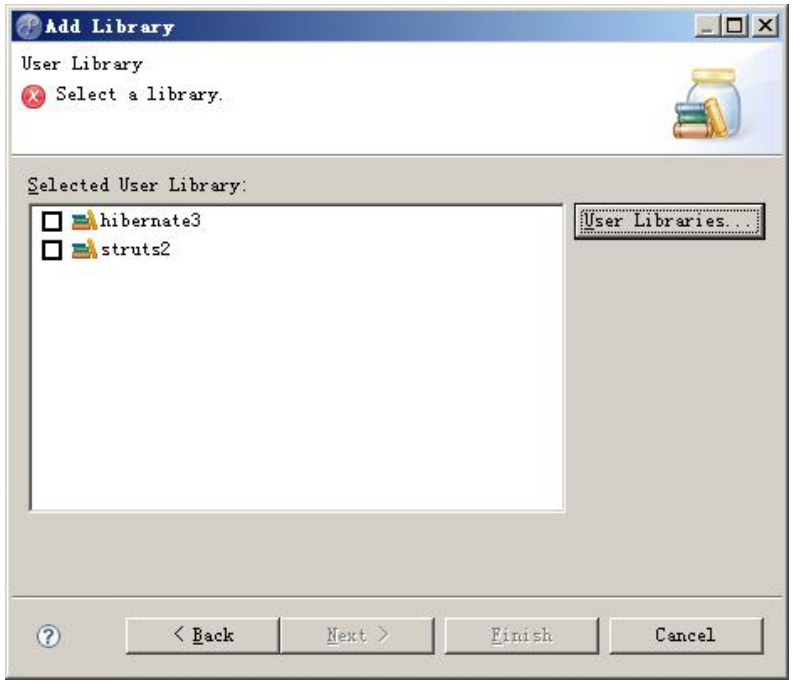

图6.4 用户定义Library对话框

6.3 添加Spring开发

由于笔者这里已经定义过User Library，所以出现了两个自定义的Library。如果用户是第一次定义，应该没有列表项，单击【User Libraries...】按钮，出现如图6.5所示的对话框。

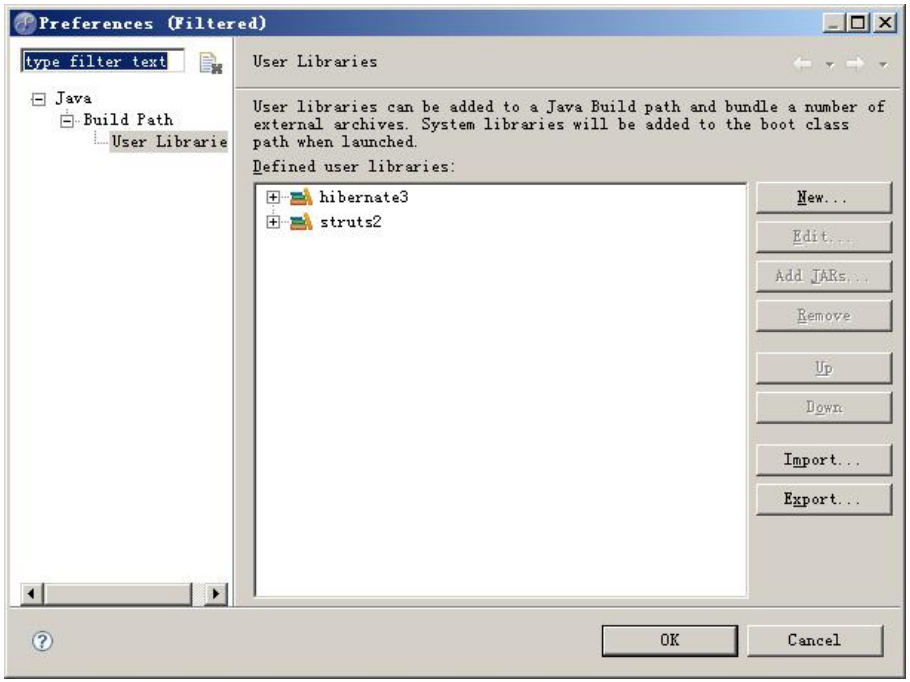

图6.5 添加User Libraries对话框

6.3 添加Spring开发

单击【New】按钮，出现如图6.6所示的对话框。

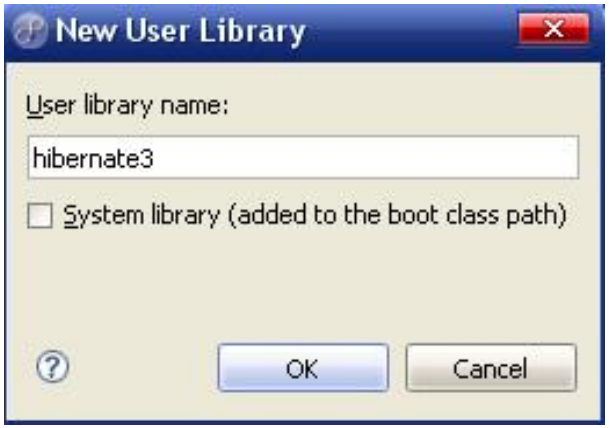

图6.6 New User Library对话框

6.3 添加Spring开发

现在可以为项目添加Spring开发能力。选择【MyEclipse】→【Add Spring Capabilities...】菜单项，出现选择版本及类库的对话框，如图6.7所示。

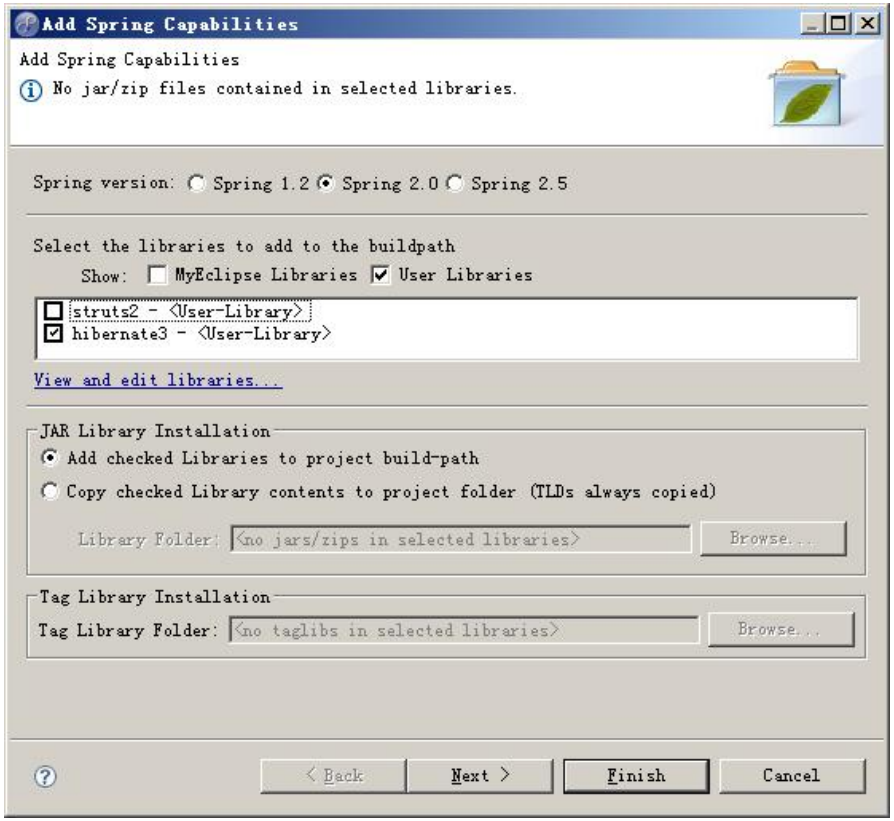

图6.7 Spring选择类库

6.4 实现Hibernate持久层

首先建立与SQL Server的连接，步骤见4.2.1节的第2步。建成后的连接如图6.8所示。

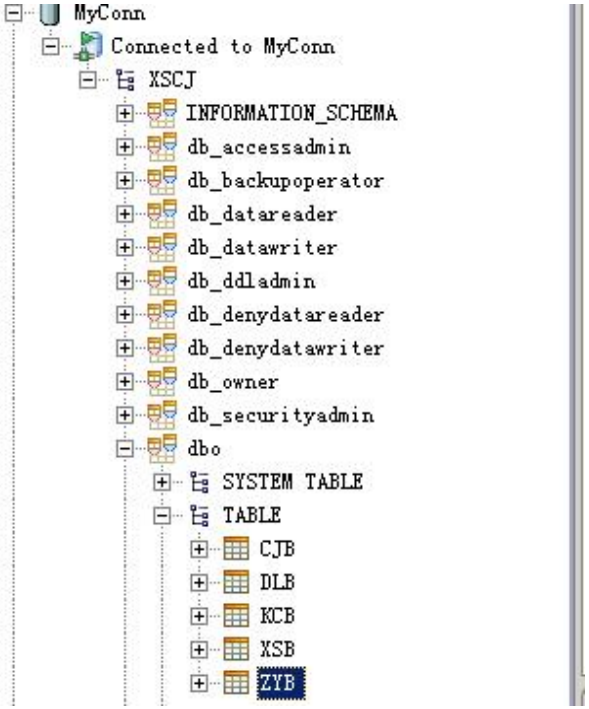

图6.8 与SQL Server的连接

6.4 实现Hibernate持久层

- Dlb.java文件代码。

对应映射文件Dlb.hbm.xml：

```
<?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="org.model.Dlb" table="DLB" schema="dbo" catalog="XSCJ">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="xh" type="java.lang.String">
            <column name="XH" length="6" not-null="true" />
        </property>
        <property name="kl" type="java.lang.String">
            <column name="KL" length="50" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

- Xsb.java文件代码。

6.4 实现Hibernate持久层

- 对应映射文件Xsb.hbm.xml。
- Kcb.java文件代码如下：

```
package org.model;
public class Kcb implements java.io.Serializable {
        private String kch;                    //课程号
        private String kcm;                    //课程名
        private Short kxxq;                    //开学学期
        private Short xs;                                  //学时
        private int xf;                                    //学分
        public Kcb() {
        }
        public Kcb(String kcm, Short kxxq, Short xs, int xf) {
                this.kcm = kcm;
                this.kxxq = kxxq;
                this.xs = xs;
                this.xf = xf;
        }
        //省略上述属性的getter和setter方法
}
```

6.4 实现Hibernate持久层

- 对应映射文件Kcb.hbm.xml。
- Zyb.java文件代码如下：

```
package org.model;
import java.util.HashSet;
import java.util.Set;
public class Zyb implements java.io.Serializable {
        private Integer id;                                    //ID
        private String zym;                                    //专业名
        private Integer rs;                                    //人数
        private String fdy;                                    //辅导员
        //这个是单向多对一，所以Zyb.java里没有Xsb的Set
        public Zyb() {
        }
        //省略上述属性的getter和setter方法
}
```

6.4 实现Hibernate持久层

- 对应映射文件Zyb.hbm.xml。
- CjbId.java文件代码如下：

```
package org.model;
public class CjbId implements java.io.Serializable {
        private String xh;
        private String kch;
        public CjbId() {
        }
        public CjbId(String xh, String kch) {
                this.xh = xh;
                this.kch = kch;
        }
        //省略主键对应getter和setter方法
}
```

6.4 实现Hibernate持久层

- Cjb.java文件代码如下：

```
package org.model;
public class Cjb implements java.io.Serializable {
        private CjbId id;
        private int cj;
        private int xf;
        public Cjb() {
        }
        public Cjb(CjbId id) {
                this.id = id;
        }
        public Cjb(CjbId id, int cj, int xf) {
                this.id = id;
                this.cj = cj;
                this.xf = xf;
        }
        //省略上述属性setter和getter方法
}
```

6.4 实现Hibernate持久层

- 对应映射文件Cjb.hbm.xml。
- 对应文件完成后，还要在Spring配置文件中注册，Spring配置文件application Context.xml。

6.5 实现DAO

下面介绍这5个类对应的DAO组件的实现。这里的DAO实现类中用到了Spring整合Hibernate后提供的HibernateDaoSupport类。

- 登录类对应DAO层实现，DlDao.java接口：

```
package org.dao;
import java.util.List;
import org.model.Dlb;
public interface DlDao {
        //插入用户
        public void save(Dlb user);
        //根据学号和口令查找
        public Dlb find(String xh,String kl);
        //是否存在该学号的用户
        public boolean existXh(String xh);
}
```

- 对应实现类DlDaoImp.java。

6.5 实现DAO

- 学生类对应DAO层实现，XsDao.java接口：

```
package org.dao;
import java.util.List;
import org.model.Xsb;
public interface XsDao {
        //插入学生
        public void save(Xsb xs);
        //根据学号删除学生
        public void delete(String xh);
        //修改学生信息
        public void update(Xsb xs);
        //根据学号查询学生信息
        public Xsb find(String xh);
        //分页显示学生信息
        public List findAll(int pageNow,int pageSize);
        //查询一共多少条学生记录
        public int findXsSize();
}
```

- 对应实现类XsDaoImp.java。

6.5 实现DAO

课程类对应DAO层实现，这里只列举在成绩信息中应用到的课程信息方法。

KcDao.java接口：

```
package org.dao;
import java.util.List;
import org.model.Kcb;
public interface KcDao {
    //根据课程号查找课程信息
    public Kcb find(String kch);
    //分页查询
    public List findAll(int pageNow,int pageSize);
    //查询一共多少条课程记录
    public int findKcSize();
}
```

- 对应实现类KcDaoImp.java：

6.5 实现DAO

- 专业类对应DAO层实现，ZyDao.java接口：

```
package org.dao;
import java.util.List;
import org.model.Zyb;
public interface ZyDao {
        //插入专业信息
        public void save(Zyb zy);
        //根据专业ID查找专业信息
        public Zyb getOneZy(Integer zyId);
        //查找所有专业信息
        public List getAll();
}
```

6.5 实现DAO

- 对应实现类ZyDaoImp.java：

```
package org.dao.imp;
import java.util.List;
import org.dao.ZyDao;
import org.model.Zyb;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
public class ZyDaoImp extends HibernateDaoSupport implements ZyDao{
        public List getAll() {
                return this.getHibernateTemplate().find("from Zyb");
        }
        public Zyb getOneZy(Integer zyId) {
                return (Zyb)getHibernateTemplate().find("from Zyb where
id=?",zyId).get(0);
        }
        public void save(Zyb zy) {
                getHibernateTemplate().save(zy);
        }
}
```

6.5 实现DAO

- 成绩类对应DAO层实现，CjDao.java接口。
- 对应实现类CjDaoImp.java。

基于HibernateDaoSupport的DAO组件必须获得一个SessionFactory的引用，然后才可以完成持久化访问。因此把这些DAO组件交由Spring容器的Bean来管理，而且在后面的业务逻辑中也要用到这些组件，所以要在Spring配置文件中加入下列代码。

6.6 实现业务逻辑层

下面列举它们对应的业务逻辑接口及实现类。

- DlService.java接口：

```
package org.service;
import org.model.Dlb;
public interface DlService {
        //插入用户
        public void save(Dlb user);
        //根据学号和口令查找
        public Dlb find(String xh,String kl);
        //是否存在该学号的用户
        public boolean existXh(String xh);
}
```

6.6 实现业务逻辑层

- 对应实现类DlServiceManage.java：

```
package org.service.imp;
import org.dao.DlDao;
import org.model.Dlb;
import org.service.DlService;
public class DlServiceManage implements DlService{
        //对DlDao进行依赖注入
        private DlDao dlDao;
        public void setDlDao(DlDao dlDao) {
                this.dlDao = dlDao;
        }
        public boolean existXh(String xh) {
                return dlDao.existXh(xh);
        }
        public Dlb find(String xh, String kl) {
                return dlDao.find(xh, kl);
        }
        public void save(Dlb user) {
                dlDao.save(user);
        }
}
```

6.6 实现业务逻辑层

- XsService.java接口：

```
package org.service;
import java.util.List;
import org.model.Xsb;
public interface XsService {
        //插入学生
        public void save(Xsb xs);
        //根据学号删除学生
        public void delete(String xh);
        //修改学生信息
        public void update(Xsb xs);
        //根据学号查询学生信息
        public Xsb find(String xh);
        //分页显示学生信息
        public List findAll(int pageNow,int pageSize);
        //查询一共多少条学生记录
        public int findXsSize();
}
```

6.6 实现业务逻辑层

- 对应实现类XsServiceImp.java。
- ZyService.java接口：

```
package org.service;
import java.util.List;
import org.model.Zyb;
public interface ZyService {
        //插入专业信息
        public void save(Zyb zy);
        //根据专业ID查找专业信息
        public Zyb getOneZy(Integer zyId);
        //查找所有专业信息
        public List getAll();
}
```

6.6 实现业务逻辑层

- 对应实现类ZyServiceManage.java：

```
package org.service.imp;
import java.util.List;
import org.dao.ZyDao;
import org.model.Zyb;
public class ZyServiceManage implements org.service.ZyService {
        //对ZyDao进行依赖注入
        private ZyDao zyDao;
        public void setZyDao(ZyDao zyDao) {
                this.zyDao = zyDao;
        }
        public List getAll() {
                return zyDao.getAll();
        }
        public Zyb getOneZy(Integer zyId) {
                return zyDao.getOneZy(zyId);
        }
        public void save(Zyb zy) {
                zyDao.save(zy);
        }
}
```

6.6 实现业务逻辑层

- KcService.java接口：

```
package org.service;
import java.util.List;
import org.model.Kcb;
public interface KcService {
//根据课程号查询
public Kcb find(String kch);
//插入课程信息
public void save(Kcb kc);
//分页显示课程信息
public List findAll(int pageNow,int pageSize);
//查询一共多少条课程记录
public int findKcSize();
}
```

6.6 实现业务逻辑层

- 对应实现类KcServiceManage.java：

```
package org.service.imp;
import java.util.List;
import org.dao.CjDao;
import org.dao.KcDao;
import org.model.Kcb;
import org.service.KcService;
public class KcServiceManage implements KcService {
        private KcDao kcDao;
        public void setKcDao(KcDao kcDao) {
                this.kcDao = kcDao;
        }
        public Kcb find(String kch) {
                return kcDao.find(kch);
        }
        public List findAll(int pageNow, int pageSize) {
                return kcDao.findAll(pageNow, pageSize);
        }
        public int findKcSize() {
                return kcDao.findKcSize();
        }
        public void save(Kcb kc) {
        // TODO Auto-generated method stub
        }
}
```

6.6 实现业务逻辑层

- CjService.java接口：

```
package org.service;
import java.util.List;
import org.model.Cjb;
public interface CjService {
        //插入学生成绩
        public void saveorupdateCj(Cjb cj);
        //根据学号和课程号删除学生成绩
        public void deleteCj(String xh,String kch);
        //根据学号和课程号查询学生成绩
        public Cjb getXsCj(String xh,String kch);
        //分页显示所有学生成绩
        public List findAllCj(int pageNow,int pageSize);
        //查询某学生成绩
        public List getXsCjList(String xh);
        //查询某门课程的学生成绩
        public List getKcCjList(String kch);
        //删除某学生的成绩
        public void deleteOneXsCj(String xh);
        //删除某门课程的成绩
        public void deleteOneKcInfo(String kch);
        //查询一共多少条成绩记录
        public int findCjSize();
}
```

6.6 实现业务逻辑层

- 对应实现类CjServiceManage.java。

  把业务逻辑交由Spring容器的Bean管理，在Spring配置文件中加入以下代码。

  在我们的操作中，必须要用到事务管理，所以这里要用到Spring的事务，在Spring配置文件中加入以下代码来对业务逻辑进行事务管理。

6.7 实现Web层

- **6.7.1 web.xml及struts.properties配置文件**

既然要用到Struts 2及Spring，就需要在web.xml中配置相应的过滤器及监听器。其代码。

还要配置Struts 2与Spring整合用到的struts.properties文件，其代码如下：

struts.objectFactory=spring

- **6.7.2 分页实现**

从前面的方法可以看出，项目运用了分页技术，在查询结果中，一般要有首页、前一页、后一页及尾页。所以这里要先写一个Pager.java类，实现页面分页操作。代码设计。

6.7.3 系统主界面

系统运行后，会出现如图6.9所示的主界面。

图6.9 运行主界面

6.7.3 系统主界面

该主界面分为4个部分，分别是头部head.jsp、左边部分left.jsp、右边部分rigtht.jsp和尾部foot.jsp。然后用main.jsp把它们整合在一起。下面是实现它们的代码。

- head.jsp代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
    <title>学生成绩管理系统</title>
</head>
<body bgcolor="#D9DFAA">
    <img src="/xscjManage/images/head.gif" />
</body>
</html>
```

- left.jsp代码。

6.7.3 系统主界面

- right.jsp代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
    <title>学生成绩管理系统</title>
</head>
<body bgcolor="#D9DFAA">
</body>
</html>
```

- foot.jsp代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
    <title>学生成绩管理系统</title>
</head>
<body bgcolor="#D9DFAA">
    <img src="/xscjManage/images/foot.gif" />
</body>
</html>
```

6.7.3 系统主界面

- main.jsp代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
   <title>学生成绩管理系统</title>
</head>
      <frameset rows="24%,68%,*" border="0">
                  <frame src="head.jsp">
                  <frameset cols="15%,*">
                              <frame src="left.jsp">
                              <frame src="right.jsp" name="right">
                  </frameset>
                  <frame src="foot.jsp">
      </frameset>
<body>
</body>
</html>
```

6.7.4 “登录”功能实现

在6.7.3节的主界面中，如果用户单击左边的任意一个超链接时还没有登录，都会跳转到登录界面，如图6.10所示。

图6.10 登录界面

6.7.4 “登录”功能实现

原来，Struts 2可以自己配置拦截器，用来验证用户是否已经登录，如果没有登录就跳转到登录界面。而登录成功后就会跳转到成功界面，如图6.11所示。

图6.11 登录成功界面

6.7.4 “登录”功能实现

- 下面是拦截器代码：

```
package org.tool;
import java.util.Map;
import org.model.Dlb;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class MyFilter extends AbstractInterceptor{
        public String intercept(ActionInvocation arg0) throws Exception {
                Map session=arg0.getInvocationContext().getSession();
                Dlb user=(Dlb) session.get("user");
                if(user==null){
                        return Action.LOGIN;
                }
                return arg0.invoke();
        }
}
```

6.7.4 “登录”功能实现

下面是实现left.jsp中超链接跳转的Action在Struts的核心配置文件struts.xml中的配置。

如果判断出没有登录就会跳转到登录界面。下面是登录界面login.jsp的实现：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s"%>
<html>
<head>
   <title>学生成绩管理系统</title>
</head>
<body bgcolor="#D9DFAA">
                    <table width="700" align="center">
        <tr>
                    <td>
        <s:form action="login.action" method="post">
                    <s:textfield name="dl.xh" label="学号" size="20"/>
                    <s:password name="dl.kl" label="口令" size="21"/>
                    <s:submit value="登录"/>
        </s:form>
        </td>
        </tr>
        </table>
</body>
</html>
```

6.7.4 “登录”功能实现

- 在struts.xml中的配置：

```
<!-- 登录 -->
<action name="login" class="dlAction">
        <result name="success">/login_success.jsp</result>
        <result name="error">/login.jsp</result>
</action>
```

- 对应Action实现类DlAction.java。

6.7.4 “登录”功能实现

- 由于该Action实例是由Spring完成的，所以在applicationContext.xml文件中加入代码：

```
<bean id="dlAction" class="org.action.DlAction">
    <property name="dlService">
        <ref bean="dlService"/>
    </property>
</bean>
```

- 登录成功后的界面login_success.jsp：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
</head>
<body bgcolor="#D9DFAA">
    登录成功！你可以进行相关操作了！
</body>
</html>
```

6.7.5 “学生信息管理”功能实现

◈ **1. 显示所有学生信息**

在left.jsp中有一个【学生信息查询】超链接，如果登录后单击它，则会分页列举出所有学生信息，如图6.12所示。

图6.12 所有学生信息

6.7.5 “学生信息管理”功能实现

- 核心配置中的action配置在前面的拦截器中已经列出，对应Action类实现代码。

该Action类也是由Spring管理的，在该实现类中，实现添加学生信息功能时用到了专业信息的业务逻辑，所以这里先列出，后面用到时就不必列举了。

```
<bean id="xsAction" class="org.action.XsAction">
    <property name="xsService">
        <ref bean="xsService"/>
    </property>
    <property name="zyService">
        <ref bean="zyService"/>
    </property>
</bean>
```

- 成功后跳转到xsInfo.jsp，分页显示所有学生信息，代码。

6.7.5 “学生信息管理”功能实现

◈ **2. 显示某个学生的详细信息**

在xsInfo.jsp中有【详细信息】超链接。单击它会显示该学生的详细信息，如图6.13所示。

图6.13 显示某学生详细信息界面

6.7.5 “学生信息管理”功能实现

- 按照我们的开发顺序，列举出实现该功能的代码。Action配置如下：

```
<!-- 某个学生详细信息 -->
<action name="findXs" class="xsAction" method="findXs">
    <result name="success">/moretail.jsp</result>
</action>
<action name="getImage" class="xsAction" method="getImage">
</action>
```

注意，这里有对照片的读取，故有getImage的Action配置。在XsAction类中加入下面的实现代码。

- 显示页面moretail.jsp代码。

6.7.5 “学生信息管理”功能实现

- **3. 删除某学生信息**
- 在xsInfo.jsp中，有下面的代码：

```
<td> <a href="deleteXs.action?xs.xh=<s:property value="#xs.xh"/>"
                    onClick="if(!confirm('确定删除该信息吗？'))return false;else
return true;">删除</a>
 </td>
```

6.7.5 “学生信息管理”功能实现

为了防止操作人员无意中单击【删除】超链接，故加入了上面的确定消息框，当用户单击【删除】超链接时，会出现如图6.14所示的界面。

图6.14 删除学生信息界面

6.7.5 “学生信息管理”功能实现

- 单击【确定】按钮，提交信息到deleteXs.action。对应的Action配置如下：

```
<!-- 删除学生 -->
<action name="deleteXs" class="xsAction" method="deleteXs">
        <result name="success" >/success.jsp</result>
</action>
```

- 对应XsAction类中实现的代码如下：

```
public String deleteXs() throws Exception{
        String xh=xs.getXh();
        xsService.delete(xh);
        return SUCCESS;
}
```

6.7.5 “学生信息管理”功能实现

- 操作成功后会跳转到成功界面success.jsp：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
</head>
<body bgcolor="#D9DFAA">
    恭喜你，操作成功！
</body>
</html>
```

6.7.5 “学生信息管理”功能实现

◈ **4. 修改学生信息**

修改学生信息要首先跳转到修改学生信息界面，并且获得该学生的信息，如图6.15所示。

图6.15 修改学生信息界面

6.7.5 “学生信息管理”功能实现

下面看功能实现代码。

```
<td> <a href="updateXsView.action?xs.xh=<s:property value="#xs.xh"/>">修改</a> </td>
```

从上面的代码中可以看出，该超链接交给Action且传递了该学生的学号作为参数。

```
<!-- 进入修改学生界面 -->
<action name="updateXsView" class="xsAction" method="updateXsView">
        <result name="success">/updateXsView.jsp</result>
</action>
```

XsAction类的方法实现如下：

```
public String updateXsView() throws Exception{
        String xh=xs.getXh();
        Xsb xsInfo=xsService.find(xh);
        List zys=zyService.getAll();
        Map request=(Map)ActionContext.getContext().get("request");
        request.put("xsInfo", xsInfo);
        request.put("zys", zys);
        return SUCCESS;
 }
```

6.7.5 “学生信息管理”功能实现

由于这里要查询出所有的专业信息，用到了专业的业务逻辑，所以要在该Action类中加入专业的业务逻辑的setter方法，进行依赖注入。

```
private ZyService zyService;
public void setZyService(ZyService zyService) {
    this.zyService = zyService;
}
```

- 在Action类处理完毕后，跳转到修改页面updateXsView.jsp，实现代码。
- 当填写好要修改的内容后，单击【修改】按钮，提交到updateXs.action。

```
<!-- 修改学生信息 -->
<action name="updateXs" class="xsAction" method="updateXs">
    <result name="success">/success.jsp</result>
</action>
```

6.7.5 “学生信息管理”功能实现

- XsAction类的实现方法：

```
public String updateXs() throws Exception{
        Xsb xs1=xsService.find(xs.getXh());
        xs1.setXm(xs.getXm());
        xs1.setXb(xs.getXb());
        xs1.setZyb(zyService.getOneZy(xs.getZyb().getId()));
        xs1.setCssj(xs.getCssj());
        xs1.setZxf(xs.getZxf());
        xs1.setBz(xs.getBz());
        if(this.getZpfile()!=null){
                FileInputStream fis=new FileInputStream(this.getZpfile());
                byte[] buffer=new byte[fis.available()];
                fis.read(buffer);
                xs1.setZp(buffer);
      }
       Map request=(Map)ActionContext.getContext().get("request");
       xsService.update(xs1);
       return SUCCESS;
}
```

6.7.5 “学生信息管理”功能实现

- 由于要获取照片文件，所以要在XsAction类中加入下列属性：

```
private File zpfile;
public File getZpfile() {
        return zpfile;
}
public void setZpfile(File zpfile) {
        this.zpfile = zpfile;
}
```

6.7.5 “学生信息管理”功能实现

◈ **5. 添加学生信息**

在添加学生信息时首先要进入添加学生信息界面。单击left.jsp中的【学生信息录入】超链接，出现如图6.16所示界面。

图6.16 添加学生信息界面

6.7.5 “学生信息管理”功能实现

下面看其实现代码。首先是超链接提交的Action配置，它在前面拦截器配置中已经给出，这里就不再列举。其次是在XsAction类中的实现方法：

```
public String addXsView() throws Exception{
        return SUCCESS;
 }
```

在XsAction类中添加一个List属性，并生成其getter和setter方法，用来保存专业集合，这样在页面中直接调用即可，非常方便，下面是其实现代码：

```
//存放专业集合
private List list;
public void setList(List list) {
      this.list = list;
}
 public List getList(){
        return zyService.getAll();              //返回专业的集合
 }
```

6.7.5 “学生信息管理”功能实现

该方法获取专业的所有信息，以便在显示页面中放入下拉列表中。显示页面addXsInfo.jsp代码。

在输入框中输入要添加学生的信息，然后单击【添加】按钮，提交给addXs.action：

```
<!-- 添加学生 -->
<action name="addXs" class="xsAction" method="addXs">
    <result name="success">/success.jsp</result>
    <result name="error">/existXs.jsp</result>
</action>
```

6.7.5 “学生信息管理”功能实现

- 对应在XsAction类中方法的实现。

在Action配置中可以看出，如果Action类返回ERROR，就会跳转到existXs.jsp，它是通知该学生已经存在的页面：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
</head>
<body bgcolor="#D9DFAA">
                学号已经存在！
</body>
</html>
```

6.7.6 “学生成绩信息管理”功能实现

1. 学生成绩录入

学生成绩录入，要先进入学生成绩录入界面，输入学生姓名、课程名及成绩。由于在录入学生成绩时，学生名和课程名是不能随意填写的，不允许用户填写一个不存在的学生和课程名，所以要从数据库中查询出学生及课程名。可在成绩录入页面中将它们设计成下拉列表，供选择使用，如图6.17所示。

图6.17 成绩录入界面

6.7.6 “学生成绩信息管理”功能实现

但是如果用户选择的学生及课程都是存在的，并且有成绩，这样就会有冲突，所以这里把录入操作设计成“添加”或“修改”操作，从前面列出的DAO实现类中的方法可以发现：

```
public void saveorupdateCj(Cjb cj) {
    getHibernateTemplate().saveOrUpdate(cj);
}
```

在left.jsp中单击【成绩信息录入】超链接，如果用户已经登录就会跳转到如图6.17所示的界面，该Action配置如下：

```
<!-- 进入添加或修改学生成绩界面 -->
<action name="addXscjView" class="cjAction">
    <result name="success">/addCj.jsp</result>
    <interceptor-ref name="defaultStack"></interceptor-ref>
    <interceptor-ref name="myFilter"/>
</action>
```

6.7.6 “学生成绩信息管理”功能实现

对应的Action类的实现代码。

把该CjAction类交由Spring管理（这里把后面要用到的业务逻辑同时列出）：

```
<bean id="cjAction" class="org.action.CjAction">
        <property name="xsService">
                <ref bean="xsService"/>
        </property>
        <property name="kcService">
                <ref bean="kcService"/>
        </property>
        <property name="cjService">
                <ref bean="cjService"/>
        </property>
</bean>
```

6.7.6 “学生成绩信息管理”功能实现

把该CjAction类交由Spring管理（这里把后面要用到的业务逻辑同时列出）：

```
<bean id="cjAction" class="org.action.CjAction">
        <property name="xsService">
                <ref bean="xsService"/>
        </property>
        <property name="kcService">
                <ref bean="kcService"/>
        </property>
        <property name="cjService">
                <ref bean="cjService"/>
        </property>
</bean>
```

- 显示页面addCj.jsp代码。

6.7.6 “学生成绩信息管理”功能实现

单击录入成绩信息页面的【确定】按钮，交给addorupdateXscj.action，对应Action配置：

```
<!-- 添加或修改学生成绩 -->
<action name="addorupdateXscj" class="cjAction"
method="addorupdateXscj">
        <result name="success">/success.jsp</result>
</action>
```

6.7.6 “学生成绩信息管理”功能实现

- 在CjAction类中的方法实现如下：

```
public String addorupdateXscj() throws Exception{
        Cjb cj1 = null;
        CjbId cjId1=new CjbId();
        cjId1.setXh(cj.getId().getXh());
        cjId1.setKch(cj.getId().getKch());
        if(cjService.getXsCj(cj.getId().getXh(), cj.getId().getKch())==null){
                cj1 = new Cjb();
                cj1.setId(cjId1);
        }else{
                cj1 = cjService.getXsCj(cj.getId().getXh(), cj.getId().getKch());
        }
        Kcb kc1=kcService.find(cj.getId().getKch());
        cj1.setCj(cj.getCj());
        if(cj.getCj()>60||cj.getCj()==60){
                cj1.setXf(kc1.getXf());
        }else
                cj1.setXf(0);
        cjService.saveorupdateCj(cj1);
        return SUCCESS;
}
```

6.7.6 “学生成绩信息管理”功能实现

- 很明显，还要加入下面的属性及方法：

```
private Cjb cj;
private CjService cjService;
public Cjb getCj() {
        return cj;
}
public void setCj(Cjb cj) {
        this.cj = cj;
}
public void setCjService(CjService cjService) {
        this.cjService = cjService;
}
```

6.7.6 “学生成绩信息管理”功能实现

◈ **2. 显示所有学生成绩**

在left.jsp中有一个【学生成绩查询】超链接，如果登录后单击它，就会分页显示所有学生的成绩，如图6.18所示。

图6.18 学生成绩查询界面

6.7.6 “学生成绩信息管理”功能实现

其实现的Action配置已经在解释拦截器时列举出，对应的CjAction类的实现方法如下：

```
public String xscjInfo() throws Exception{
    List list=cjService.findAllCj(this.getPageNow(), this.getPageSize());
    Map request=(Map)ActionContext.getContext().get("request");
    request.put("list",list);
    Pager page=new Pager(this.getPageNow(),cjService.findCjSize());
    System.out.println(cjService.findCjSize());
    request.put("page", page);
    return SUCCESS;
}
```

- 由于用到了分页功能，故需要在该Action中加入下面的属性：

```
private int pageNow=1;                    //默认第一页
private int pageSize=8;                   //每页显示8条记录
//并生成它们的getter和setter方法
```

- 显示页面xscjInfo.jsp代码。

6.7.6 “学生成绩信息管理”功能实现

◈ **3. 查询学生成绩**

在显示所有学生成绩页面中，将学号设计成超链接，单击【学号】超链接，就会显示该学生的所有课程的成绩。如单击学号081101，学生成绩如图6.19所示。

图6.19 显示某学生课程成绩界面

6.7.6 “学生成绩信息管理”功能实现

- 在显示所有学生成绩的页面中，有如下代码：

```
<td><a href="findXscj.action?cj.id.xh=<s:property value="#xscj[0]"/>">
        <s:property value="#xscj[0]"/></a>
</td>
```

从上述代码不难发现，单击【学号】超链接，提交给findXscj.action，Action配置如下：

```
<!-- 某学生课程成绩 -->
<action name="findXscj" class="cjAction" method="findXscj">
        <result name="success">/oneXscj.jsp</result>
        <result name="error">/noXscj.jsp</result>
</action>
```

- 对应的CjAction类的实现方法如下：

```
public String findXscj() throws Exception{
        List list=cjService.getXsCjList(cj.getId().getXh());
        if(list.size()>0){
                Map request=(Map)ActionContext.getContext().get("request");
                request.put("list",list);
                return SUCCESS;
        }else
                return ERROR;
}
```

6.7.6 “学生成绩信息管理”功能实现

- 成功后返回页面oneXscj.jsp。
- 如果失败，会跳转到noXscj.jsp页面：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<body bgcolor="#D9DFAA">
    对不起，不存在该学生成绩！
</body>
</html>
```

6.7.6 “学生成绩信息管理”功能实现

◈ **4. 删除学生课程成绩**

与删除学生信息相同，单击【删除】超链接，提示用户确认，只有用户确定删除才会提交请求。在struts.xml中Action配置如下：

```
<!-- 删除某学生的某门成绩 -->
<action name="deleteOneXscj" class="cjAction"
method="deleteOneXscj">
        <result name="success">/success.jsp</result>
</action>
```

➢ 对应的CjAction类的实现方法如下：

```
public String deleteOneXscj() throws Exception{
        String xh=cj.getId().getXh();
        String kch=cj.getId().getKch();
        cjService.deleteCj(xh, kch);
        return SUCCESS;
}
```