《精通 J2EE--Eclipse、Struts、Hibernate 及 Spring 整合应用案例》

精通 J2EE——Eclipse、Struts、Hibernate 及 Spring 整合应用案例
【作者】计磊 李里 周伟 编著
【出版社】人民邮电出版社
【ISBN】 7-115-15033-8/TP.5571
【出版日期】2006 年 8 月

章节精彩阅读：
http://book.csdn.net/bookfiles/94/index.html
购书网址：
http://www.china-pub.com/computers/common/info.asp?id=31561
http://www.dearbook.com.cn/book/110849

特点：

囊括 J2EE 开发最新技术，代表 Java 的发展方向
8 个典型案例，具有很强的代表性
贯穿了大量的开发思想与技术要点，具有很强的实用价值
按照实际开发流程进行编写，让读者体会其中的奥妙
代码规范，层次清楚，注释丰富，易于理解

本书关键字：
J2EE
MVC 模式
Ajax 技术

Struts 框架
Spring 框架
Hibernate 持久层
MySQL 数据库
JSP 技术
Servlet 技术
JDBC 数据库开发
CVS 版本控制
JavaBean
Eclipse 开发工具
项目开发

本书包含的系统：

第 11 章　网上订购子系统——JSP+JavaBean
第 12 章　公告管理系统——Eclipse+Struts
第 13 章　学生课程及成绩管理系统——Struts+Hibernate
第 14 章　科研信息发布平台——JSP+Servlet
第 15 章　通用论坛 BBS——Eclipse+Struts
第 16 章　网上书店——Struts+Hibernate
第 17 章　个性化定制系统——Ajax+Spring+Hibernate
第 18 章　网上文件管理系统——Eclipse+Struts

读者对象：

相关专业毕业设计的学生
有一定 Java 基础的程序员
J2EE 架构师和开发者
Web 应用（B/S 结构）开发人员

目录

18.17　小结

第 4 章　Hibernate 高级技术

在上一章中，通过一个简单实例介绍了使用 Hibernate 开发的流程。在本章中将进一步深入学习 Hibernate。首先向读者介绍 Hibernate 的核心接口，然后介绍 Hibernate 配置文件的应用，最后将介绍常用的映射关系。

4.1　Hibernate 核心接口

在项目中使用 Hibernate 框架，非常关键的一点就是要了解 Hibernate 的核心接口。Hibernate 接口位于业务层和持久化层，如图 4.1 所示。

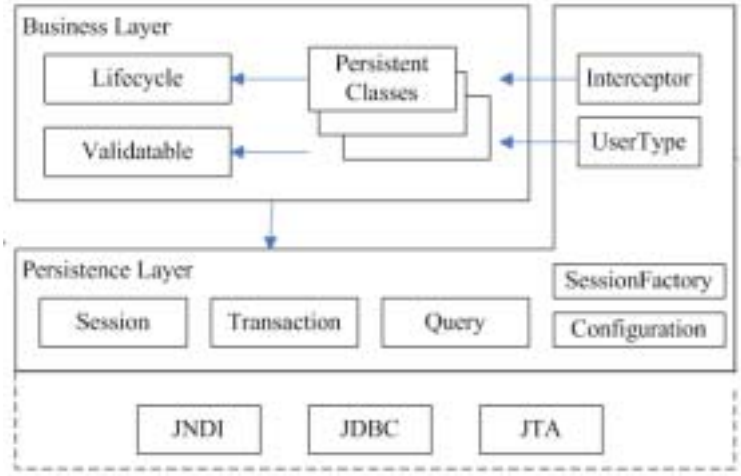

图 4.1　Hibernate 核心接口的层次架构关系

Hibernate 的核心接口一共有 5 个，分别为：Session、SessionFactory、Transaction、Query 和 Configuration。这 5 个核心接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。下面对这五的核心接口分别加以介绍。

Session 接口：Session 接口负责执行被持久化对象的 CRUD 操作（CRUD 的任务是完成与数据库的交流，包含了很多常见的 SQL 语句。）。但需要注意的是 Session 对象是非线程安全的。同时，Hibernate 的 session 不同于 JSP 应用中的 HttpSession。这里当使用 session 这个术语时，其实指的是 Hibernate 中的 session，而以后会将 HttpSesion 对象称为用户 session。

SessionFactory 接口：SessionFactroy 接口负责初始化 Hibernate。它充当数据存储源的代理，并负责创建 Session 对象。这里用到了工厂模式。需要注意的是 SessionFactory 并不是轻量级的，因为一般情况下，一个项目通常只需要一个 SessionFactory 就够，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory。

Configuration 接口：Configuration 接口负责配置并启动 Hibernate，创建 SessionFactory

对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先定位映射文档位置、读取配置，然后创建 SessionFactory 对象。

Transaction 接口：Transaction 接口负责事务相关的操作。它是可选的，可发人员也可以设计编写自己的底层事务处理代码。

Query 和 Criteria 接口：Query 和 Criteria 接口负责执行各种数据库查询。它可以使用 HQL 语言或 SQL 语句两种表达方式。

4.2 Hibernate 的配置文件应用

这一节中将讲述 Hibernate 的基本配置及配置文件的应用，这对于正确熟练使用 Hibernate 是相当关键的。

4.2.1 配置文件中映射元素详解

对象关系的映射是用一个 XML 文档来说明的。映射文档可以使用工具来生成，如 XDoclet，Middlegen 和 AndroMDA 等。下面从一个映射的例子开始讲解映射元素，映射文件的代码如下。

```
<?xml version="1.0"?>
<!--
      所有的 XML 映射文件都需要定义如下所示的 DOCTYPE。
      Hibernate 会先在它的类路径（classptah）中搜索 DTD 文件。
-->
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
            "http：//hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!--
      hibernate-mapping 有几个可选的属性：
      schema 属性指明了这个映射的表所在的 schema 名称。
      default-cascade 属性指定了默认的级联风格  可取值有  none、save、update。
      auto-import 属性默认让我们在查询语言中可以使用非全限定名的类名  可取值有  true、false。
      package 属性指定一个包前缀。
-->
<hibernate-mapping   schema="schemaName"   default-cascade="none"
      auto-import="true"     package="test">

          <!--用 class 元素来定义一个持久化类  -->
          <class name="People"    table="person">

                    <!-- id 元素定义了属性到数据库表主键字段的映射。-->
                    <id name="id">
                              <!--  用来为该持久化类的实例生成唯一的标识  -->
                              <generator class="native"/>
                    </id>

                    <!-- discriminator 识别器  是一种定义继承关系的映射方法-->
```

```
                <discriminator column="subclass" type="character"/>

                <!-- property 元素为类声明了一个持久化的，JavaBean 风格的属性-->
                <property name="name" type="string">
                    <column name="name" length="64" not-null="true" />
                </property>

                <property name="sex"
                    not-null="true"
                    update="false"/>

                <!--多对一映射关系-->
                <many-to-one name="friend"
                    column="friend_id"
                    update="false"/>

                 <!--设置关联关系-->
                 <set name="friends"
                     inverse="true"
                     order-by="id">
                         <key column="friend_id"/>
                         <!—一对多映射-->
                         <one-to-many class="Cat"/>
                 </set>
        </class>
</hibernate-mapping>
```

4.2.2 组件应用的方法

组件有两种类型，即组件（component）和动态组件（dynamic-component）。在配置文件中，component 元素为子对象的元素与父类对应表的字段建立起映射关系。然后组件可以声明它们自己的属性、组件或者集合。component 元素的定义如下所示：

```
<component
        name="propertyName"
        class="className"
        insert="true|false"
        upate="true|false"
        access="field|property|ClassName">
        <property ...../>
        <many-to-one .... />
        ........
</component>
```

在这段代码中，name 是指属性名，class 是类的名字，insert 指的是被映射的字段是否出现在 SQL 的 INSERT 语句中，upate 指出被映射的字段是否出现在 SQL 的 UPDATE 语句中，access 指出访问属性的策略。

4.2.3 Hiebernate 的基本配置

Hibernate 的数据库连接信息是从配置文件中加载的。Hibernate 的配置文件有两种形式：一种是 XML 格式的文件，一种是 properties 属性文件。properties 形式的配置文件默认文件名是 hibernate.properties，一个 properties 形式的配置文件内容如下所示：

```
#指定数据库使用的驱动类
hibernate.connection.driver_class = com.mysql.jdbc.Driver r

#指定数据库连接串
hibernate.connection.url = jdbc:mysql://localhost:3306/db

#指定数据库连接的用户名
hibernate.connection.username = user

#指定数据库连接的密码
hibernate.connection.password = password

#指定数据库使用的方言
hibernate.dialect = net.sf.hibernate.dialect.MySQLDialect

#指定是否打印 SQL 语句
hibernate.show_sql=true
```

在配置文件中包含了一系列属性的配置，Hibernate 将根据这些属性来连接数据库。

在 XML 格式的配置文件中，除了基本的 Hibernate 配置信息，还可以指定具体的持久化类的映射文件，这可以避免将持久化类的配置文件硬编码在程序中。XML 格式的配置文件的默认文件名为 hibernate.cfg.xml，一个 XML 配置文件的示例如下所示：

```
<?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>
        <!--显示执行的 SQL 语句-->
        <property name="show_sql">true</property>

        <!--连接字符串-->
        <property name="connection.url">jdbc:mysql://localhost:3306/STU</property>

        <!--连接数据库的用户名-->
        <property name="connection.username">root</property>

        <!--数据库用户密码-->
        <property name="connection.password">root</property>

        <!--数据库驱动-->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>

        <!--选择使用的方言-->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
        <!--映射文件  -->
        <mapping resource="com/stuman/domain/Admin.hbm.xml" />

        <!--映射文件-->
        <mapping resource="com/stuman/domain/Student.hbm.xml" />

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

properties 形式的配置文件和 XML 格式的配置文件可以同时使用。当同时使用两种类型的配置文件时，XML 配置文件中的设置会覆盖 properties 配置文件的相同的属性。

4.2.4 对象标识符号

在关系数据库表中，主键（Primary Key）用来识别记录，并保证每条记录的唯一性。在 Java 语言中，通过比较两个变量所引用对象的内存地址是否相同，或者比较两个变量引用的对象值是否相同来判断两对象是否相等。Hibernate 为了解决两者之间的不同，使用对象标识符（OID）来标识对象的唯一性。OID 是关系数据库中主键在 Java 对象模型中的等价物。在运行时，Hibernate 根据 OID 来维持 Java 对象和数据库表中记录的对应关系。如下代码所示，三次调用了 Session 的 load()方法，分别加载 OID 为 1 或 3 的 User 对象。

```
Transaction tx = session.beginTransaction();
User user1 =   (User)session.load(User.class,new Long(1));
User user2 =   (User)session.load(User.class,new Long(1));
User user3 =   (User)session.load(User.class,new Long(3));
System.out.println( user1 == user2 );
System.out.println( user1 == user3 );
```

应用程序在执行上述代码时，第一次加载 OID 为 1 的 User 对象，从数据库中查找 ID 为 1 的记录，然后创建相应的 User 实例，并把它保存在 Session 缓存中，最后将该实例的引用赋值给变量 user1。第二次加载 OID 为 1 的对象时，直接把 Session 缓存中 OID 为 1 的实例的引用赋值给变量 user2。因此，表达式 user1==user2 的结果为 true。
标识的生成可以使用不同的策略，表 4.1 为 Hibernate 内置的标识生成策略。

表 4.1 Hibernate标识生成策略

标识符生成器	描述
increment	适用于代理主键。由Hibernate自动以递增方式生成。
identity	适用于代理主键。由底层数据库生成标识符。
sequence	适用于代理主键。Hibernate根据底层数据库的序列生成标识符，这要求底层数据库支持序列。
hilo	适用于代理主键。Hibernate分局high/low算法生成标识符。
seqhilo	适用于代理主键。使用一个高/低位算法来高效的生成long，short或者int类型的标识符。
native	适用于代理主键。根据底层数据库对自动生成标识符的方式，自动选择identity、sequence或hilo。
uuid.hex	适用于代理主键。Hibernate采用128位的UUID算法生成标识符。
uuid.string	适用于代理主键。UUID被编码成一个16字符长的字符串。
assigned	适用于自然主键。由Java应用程序负责生成标识符。
foreign	适用于代理主键。使用另外一个相关联的对象的标识符。

4.2.5 Hibernate 映射类型

在对象/关系映射文件中，Hibernate 采用映射类型作为 Java 类型和 SQL 类型的桥梁。Hibernate 映射类型分为 2 种：内置映射类型和自定义映射类型。

1 内置映射类型

Hibernate 对所有的 Java 原生类型、常用的 Java 类型如 String、Date 等都定义了内置的映射类型。表 4.2 列出了 Hibernate 映射类型、对应的 Java 类型以及对应的标准 SQL 类型。

表 4.2 Hibernate内置映射类型

Hibernate映射类型	Java类型	标准SQL类型	大小
integer/int	java.lang.Integer/int	INTEGER	4字节
long	java.lang.Long/long	BIGINT	8字节
short	java.lang.Short/short	SMALLINT	2字节
byte	java.lang.Byte/byte	TINYINT	1字节
float	java.lang.Float/float	FLOAT	4字节
double	java.lang.Double/double	DOUBLE	8字节
big_decimal	java.math.BigDecimal	NUMERIC	
character	java.lang.Character/java.lang.String/char	CHAR(1)	定长字符
string	java.lang.String	VARCHAR	变长字符
boolean/ yes_no/true_false	java.lang.Boolean/Boolean	BIT	布尔类型
date	java.util.Date/java.sql.Date	DATE	日期
timestamp	java.util.Date/java.util.Timestamp	TIMESTAMP	日期
calendar	java.util.Calendar	TIMESTAMP	日期
calendar_date	java.util.Calendar	DATE	日期
binary	byte[]	BLOB	BLOB
text	java.lang.String	TEXT	CLOB
serializable	实现java.io.Serializablej接口的任意Java类	BLOB	BLOB
clob	java.sql.Clob	CLOB	CLOB
blob	java.sql.Blob	BLOB	BLOB
class	java.lang.Class	VARCHAR	定长字符
locale	java.util.Locale	VARCHAR	定长字符
timezone	java.util.TimeZone	VARCHAR	定长字符
currency	java.util.Currency	VARCHAR	定长字符

2 自定义映射类型

Hibernate 提供了自定义映射类型接口，允许用户以编程的方式创建自定义的映射类型。用户自定义的映射类型需要实现 net.sf.hibernate.UserType 或 net.sf.hibernate.CompositeUserType 接口。具体的创建自定义映射类型的方法请参考 hibernate 官方文档或相关资料，这里不再详细介绍。

4.3 Hibernate 数据映射

这一节中将为读者讲述 Hibernate 中的两种映射方式，即一对一映射和多对一映射。

4.3.1 一对一映射

Hibernate 中，持久化对象间的一对一关联是通过 one-to-one 元素定义的。其定义方式如下所示。

```
<one-to-one
        name="propertyName"
        class="ClassName"
        cascade="all|none|save-update|delete"
        constrained="true|false"
        outer-join="true|false|auto"
        property-ref="propertyNameFromAssociatedClass"
        access="field|property|ClassName"
/>
```

在这段代码中，name 是指属性的名字，class 指的是被关联的类的名字。cascade 是可选项，表明操作是否从父对象级联到被关联的对象。constrained 表明该类对应的表对应的数据库表，和被关联的对象所对应的数据库表之间，通过一个外键引用对主键进行约束。
outer-join 也是可选项，它指的是当设置 hibernate.use_outer_join 的时候，对这个关联允许外连接抓取。property-ref 指定关联类的一个属性，这个属性将会和本外键相对应。如果没有指定，会使用对方关联类的主键。access 是 Hibernate 用来访问属性的策略。
一对一关联有两种不同的形式，即主键关联和唯一外键关联：

1 主键关联

主键关联不需要额外的表字段。如下所示的代码，将 Student 和 Person 表的主键进行一对一关联：

```
<one-to-one name="person" class="Person"/>
<one-to-one name="student" class="Student" constrained="true"/>
```

Person 和 Student 两个表中相关字段是对等的。在下面的代码中，Person 表的主键将采用 foreign 的方式生成。

```
<class name="Person" table=" Person ">
    <id name="id" column=" Person _ID">
        <generator class="foreign">
            <param name="property">employee</param>
        </generator>
    </id>
    ...
    <one-to-one name="Student"
        class="Student"
        constrained="true"/>
</class>
```

2 惟一外键关联

此种关联方式是指一个外键与一个惟一的关键字相关联，上面的 Student 和 Person 的例子，如果使这种关联方式，应该表达成：

```
<many-to-one name=" Person " class="Person" column=" Person _ID" unique="true"/>
```

4.3.2 多对一映射

持久化对象间的多对一关联是通过 many-to-one 元素定义的。其定义方式如下所示。

```
<many-to-one
        name="propertyName"
        column="column_name"
        class="ClassName"
        cascade="all|none|save-update|delete"
        outer-join="true|false|auto"
        update="true|false"
        insert="true|false"
        property-ref="propertyNameFromAssociatedClass"
        access="field|property|ClassName"
/>
```

在这段代码中，name 指类的属性名，column 指表的字段名，class 是可选项，指关联的类的名字。cascade 指明哪些操作会从父对象级联到关联的对象，outer-join 指的是当设置 hibernate.use_outer_join 的时候，对这个关联允许外连接抓取。update 用于指定对应的字段是否用于 UPDATE，property-ref 用于指定关联类的一个属性，这个属性将会和本外键相对应，access 为可选项，指访问属性的策略。

4.4 Hibernate 检索方式

Hibernate 的检索方式主要有 HQL、QBC、QBE 检索等几种方式。Hibernate 查询语言（Hibernate Query Language，简称 HQL）是一种面向对象的查询语言。HQL 功能强大且简单易学，它具有以下功能。

在查询语句中设定各种查询条件。
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group by关键字。
提供内置聚集函数，如sum()、min()、max()等。
能够调用用户自定义的SQL函数。
支持子查询，即嵌入式查询。
支持动态绑定参数。

Session 接口的 find()方法及 Query 接口都支持 HQL 检索方式。如下代码是一个使用 Query 接口查询的示例。

```
//创建一个 Query 对象
Query query = session.createQuery ( “form User as u where u.name=:userName” + “ and u.age=:userAge” ) ;

//动态绑定参数
query.setString( “userName” , “Bush” );
query.setString( “userAge” , “50” );
```

```
//执行查询语句，返回查询结果
List results = query.list();
```

Query By Criteria（QBC）和 Query By Example（QBE）提供了检索对象的其他方式，这些检索方式主要由 Criteria 接口、Criterion 接口、Expression 类和 Example 类组成，支持在运行时动态生成查询语句。关于 QBC 和 QBE 的内容可以参考 Hibernate 官方文档或相关书籍，这里不再详细介绍。

4.5 小结

在这一章中讲述了 Hibernate 的相关概念和知识，想要在开发中使用 Hibernate，这些知识都是必不可少的。尽管更加深入的剖析 Hibernate 更有助于读者理解和使用 Hibernate，但是本节的讲解并没有过于深入，这是因为本书是以实用为标准而且主要面向初学者，所以如果读者希望深入研究 Hibernate 的技术细节和内幕请参考其他相关书籍。

第 13 章 学生课程及成绩管理系统——Struts+Hibernate

本章将要实现的是学生课程及成绩管理系统，它是整个教务管理系统的一个子系统，但它的作用却相当关键。系统方便了学生选课和查分，方便了教师的教学管理和学生成绩的录入，更为主要的是系统方便了学校的教务管理。

本系统将采用 MVC 三层架构的模式，在开发过程中会使用到 Struts 和 Hibernate 来处理页面逻辑和对象的持久化工作。本系统的开发并没有单纯地使用 JSP+Servlet 进行开发，而是结合了 Struts 和 Hibernate，这是为了使系统的结构更加清晰同时简化开发工作。

在开始系统开发前，读者应该有网页设计的基本知识，可以熟练的使用数据库（本章中使用 MySQL）和 Web 服务器（本章中使用 Tomcat），以及熟练掌握 Struts 框架和 Hibernate 的使用方法。如果您对 Struts 和 Hibernate 还不了解请先阅读本书第一部分。

13.1 系统概述

本章要实现的是一个学生课程和成绩管理系统，它是高校教务管理系统中的一个子系统。主要用户是学生、教师和管理员。管理员管理系统的基本信息（如课程信息等），学生和教师通过系统完成不同的工作。系统需要实现的功能总体来说共有三个，分别为：

管理员维护系统基本信息。

学生对应功能。

教师对应功能。

以上三部分加上登录验证模块共同组成了这个系统。为了给读者一个直观的印象，先给出系统的登录页面，如图 13.1 所示。

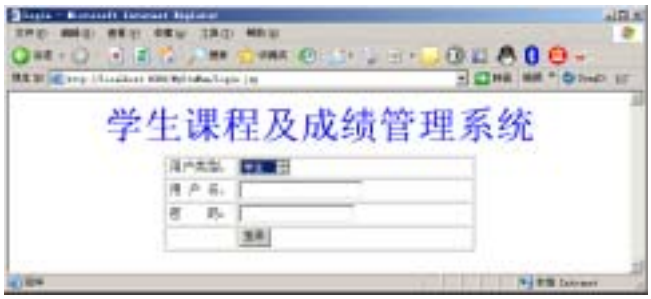

图 13.1　学生课程和成绩管理系统

图 13.1 所示页面为系统的登录页面（本例中并没有对页面做过多的美化，如果读者有兴趣可以自行美化页面），也就是系统的入口。用户首先选择用户类型，然后输入用户名和密码进入系统，再继续其他操作。

13.2　需求分析

学生课程和成绩管理系统的设计目的，是要将学生选择的课程和学生成绩通过网络进行管理。为学生、教师和教务管理人员提供便利。系统的用户共有三种类型，分别为系统管理员、学生及教师，系统对于一个用户只允许以一种身份登录。系统管理员登录系统后可以对系统进行管理，其主要操作是维护学生、教师、课程和班级的基本信息。学生登录后的主要操作是选课和个人信息维护。教师登录后的主要操作是选择学生并为学生登录成绩。

将系统需求加以总结，得出系统需求列表如下：

系统可以运行在 Windows 操作系统平台上，并通过友好的用户界面。

系统用户类型为：管理员、教师、学生。

系统对于一个用户只允许以一种身份登录。

只有管理员可以维护学生、教师、课程、班级的基本信息。

学生可以选课并维护自己的个人信息。

教师可以选择上课的学生并为学生登录成绩。

13.3　系统功能预览

通过前面的分析已经明确系统用户共有三类：

管理员：管理学生、教师、课程和班级信息。

学生：选课、查看成绩、修改个人信息。

教师：选择学生、登录成绩。

不同的用户可以通过系统进行不同的操作，每一操作都是一个功能的体现，下面给出系统需要实现的具体功能。

13.3.1　用户登录功能

不同用户登录系统时首先选择对应的用户类型，然后输入用户名及密码登录系统。系统的管理员由系统内部设定，学生和教师由管理员添加（或者与教务管理系统中其他子系统共用数

据)。用户登录系统的页面如图 13.2 所示。

图 13.2　用户登录页面

13.3.2　管理员登录后选择功能

管理员登录后，会得到欢迎信息表示登录成功，如果登录失败，则会有错误提示信息。管理员可以在如图 13.3 所示的页面中单击“ 学生 ”、“ 教师 ”、“ 课程 ”和“ 班级 ”这四个链接进入不同页面继续下一步的操作。

图 13.3　管理员管理首页

13.3.3　管理员管理学生功能（查看、添加、编辑以及删除学生信息）

管理员管理学生包括查看、添加、编辑以及删除学生的信息。为了让读者更加清楚系统的功能，下面将系统功能与页面效果相结合，逐一加以描述。

1　管理员查看学生信息

当管理员进行学生信息管理操作时，即在图 13.3 所示页面中单击“ 学生 ”链接，将跳转到一个显示所有学生信息列表的页面，页面效果如图 13.4 所示。

2　管理员添加学生信息

在如图 13.4 所示的页面上，单击学生列表右上方的“ 新加学生 ”链接，将会跳转到如图 13.5 所示的增加学生页面，系统管理员可以通过这个页面向系统中添加学生信息。

图 13.4　显示学生列表界面

图 13.5　添加学生信息

3　管理员编辑学生信息

在图 13.4 所示页面上单击学生列表中某个学生信息中的“ 编辑 ”链接时，将会跳转到如图 13.6 所示的修改学生信息页面，系统管理员可以通过这个页面编辑、更新学生的信息。

图 13.6　修改学生界面设计

4　管理员删除学生信息

在图 13.4 中每个学生记录的后面都有一个“ 删除 ”链接单击它，系统将删除这位学生的信息。

13.3.4　管理员管理教师功能（查看、添加、编辑以及删除教师信息）

管理员管理教师包括查看、添加、编辑以及删除教师的信息。为了让读者更加清楚系统的功能，下面将系统功能与页面效果相结合，逐一加以描述。

1　管理员查看教师信息

当管理员进行教师信息管理操作时，即在图 13.3 所示页面中单击“ 教师 ”链接，将跳转到一个显示所有教师信息列表的页面，页面效果如图 13.7 所示。

2　管理员添加教师信息

在图 13.7 所示页面上单击教师列表右上方的“ 新增教师 ”链接，将会跳转到如图 13.8 所示的新增教师信息页面，系统管理员可以通过这个页面向系统中添加教师信息。

图 13.7　教师列表

图 13.8　添加新教师界面设计

3　管理员编辑教师信息

单击教师列中某个教师信息中的“ 编辑 ”链接时，将会跳转到如图 13.9 所示的更新教师信息页面，系统管理员可以通过这个页面编辑、更新教师的信息。

图 13.9　添加新教师界面设计

4　管理员删除教师信息

在图 13.7 中每个教师记录的后面都有一个“ 删除 ”链接单击它，系统将删除这位教师的信息。

13.3.5　管理员管理课程功能（查看、添加、编辑以及删除课程信息）

管理员管理课程包括查看、添加、编辑以及删除课程的信息。为了让读者更加清楚系统的功能，下面将系统功能结合页面效果逐一加以描述。

1　管理员查看课程信息

当管理员进行课程信息管理操作时，即在图 13.3 所示页面中单击“ 课程 ”链接，将跳转到一个显示所有课程信息列表的页面，页面效果如图 13.10 所示。

2　管理员添加课程信息

在图 13.10 所示页面上单击课程列表右上方的“ 新增课程 ”链接，将会跳转到如图 13.11 所示的新增课程信息页面，系统管理员可以通过这个页面向系统中添加课程信息。

图 13.10　课程列表

图 13.11　添加新课程界面设计

3　管理员编辑课程信息

单击课程列中某个课程信息中的“ 编辑 ”链接时，将会跳转到如图 13.12 所示的更新课程信息页面，系统管理员可以通过这个页面编辑、更新课程的信息。

图 13.12　添加新课程界面设计

4　管理员删除课程信息

在图 13.10 中每个课程记录的后面都有一个“ 删除 ”链接单击它，删除这个课程的信息。

13.3.6　管理员管理班级功能（查看、添加、编辑以及删除班级信息）

管理员管理班级包括查看、添加、编辑以及删除班级的信息。为了让读者更加清楚系统的功能，下面将系统功能与页面效果相结合，逐一加以描述。

1　管理员查看班级信息

当管理员进行班级信息管理操作时，即在图 13.3 所示页面中单击“ 班级 ”链接，将跳转到一个显示所有班级信息列表的页面，页面效果如图 13.13 所示。

2　管理员添加班级信息

在图 13.13 所示页面上单击班级列表右上方的“ 新增班级 ”链接，将会跳转到如图 13.14 所示的新增班级信息页面，系统管理员可以通过这个页面向系统中添加班级信息。

图 13.13　班级列表

图 13.14　添加新班级界面设计

3　管理员编辑班级信息

在图 13.13 所示页面上单击班级列中某个班级信息中的‘ 编辑 ’链接时，将会跳转到如图 13.15 所示的更新班级信息页面，系统管理员可以通过这个页面编辑、更新班级的信息。

图 13.15　添加新班级界面设计

4　管理员删除班级信息

在图 13.13 中每个班级记录的后面都有一个“ 删除 ”链接单击它，系统将删除这个班级的信息。

13.3.7　学生用户登录后选择功能

学生登录本系统后，会得到欢迎信息表示登录成功，如果登录失败，则会有错误提示信息。学生可以在如图 13.16 所示的页面中单击“ 选修课程 ”、“ 查看学分 ”和“ 更改信息 ”这三个链接进入不同页面继续下一步的操作。

13.3.8　学生选修课程功能

学生登录成功后，通过单击图 13.16 所示页面中的“ 选修课程 ”链接可以跳转到如图 13.17 所示页面。这个页面显示了学生所有能够选择的课程列表。并在每条课程信息之后放置了一个“ 注册 ”链接，学生可以通过单击这个链接来选修这门课程。

图 13.16　学生选择功能

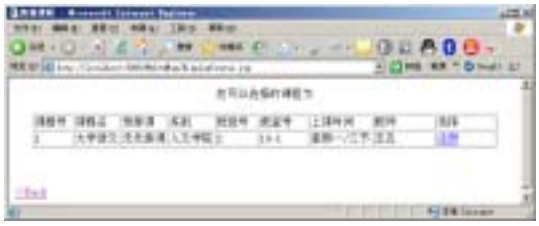

图 13.17　学生选课

13.3.9　学生查看成绩功能

通过单击图 13.16 所示页面中的“ 查看成绩 ”链接可以跳转到如图 13.18 所示页面。这个页面显示了学生所有课程的成绩列表。

13.3.10　学生更新个人信息功能

单击“ 更改信息 ”链接可以进入到如图 13.19 所示页面。学生可以通过这个页面修改自己的一些信息。

图 13.18　学生查看成绩

图 13.19　学生修改信息

13.3.11　教师用户登录后选择功能

教师登录后，会得到欢迎信息表示登录成功，如果登录失败，则会有错误提示信息。教师可以在如图 13.19 所示的页面中单击“ 选择学生 ”和“ 公布成绩 ”这两个链接进入不同页面继续下一步的操作。

13.3.12　教师选择学生功能

通过单击图 13.20 所示页面中的“ 选择学生 ”链接可以跳转到如图 13.21 所示页面。这个页面显示了教师所教授的课程列表。并在每条课程信息之后放置了一个“ 选择 ”链接，教师可以通过单击这个链接来选择报名这门课程的学生。

图 13.20　教师选择功能

图 13.21　教师教授课程列表

单击图 13.21 所示页面中的“ 选择 ”链接后，将跳转到如图 13.22 所示页面，在这个页面中列出了所有报名这门课程的学生信息。教师可以通过单击某条学生信息后的“ 接收 ”链接来选择这名学生。

13.3.13　教师登录成绩功能

单击 “ 公布成绩 ”链接跳转课程列表页面。这个页面显示了教师所教授的课程列表。并在每条课程信息之后放置了一个“ 选择 ”链接，教师可以通过单击这个链条转到学生列表页面，再通过单击某位学生信息后的链接，跳转到如图 13.23 所示的成绩录入页面（这个过程与上一小节“ 教师选择学生功能 ”的操作顺序类似）。教师用这个页面给学生打分。

图 13.22　教师选择学生

图 13.23　教师选择学生

13.4　系统分析

需求确定之后需要对系统进行整体分析和设计。这包括系统功能的描述、对功能模块的划分和对系统流程的分析。下面首先对系统功能进行描述。

13.4.1　系统功能模块划分

模块分析是描述系统需求的一个过程，需要将需求分析中的感性描述进行抽象，提取出要实现的功能，这是整个系统开发的一个关键过程。分析的根本目的是在开发者和提出需求的人之间，建立一种理解和沟通的机制。因此，学生课程及成绩管理系统的需求分析，也应该由开发人员和用户或者客户一起完成。这里的例子只用于帮助读者学习系统开发的过程及方法，所以对于将要开发实现的学生课程及成绩管理系统，实际上并没有真正的用户或客户，在开发过程中假定笔者就是系统的使用者，并由此提出具体需求。

需求分析的第一步，是描述学生课程及成绩管理系统的功能，以此确定系统的功能需求。学生课程及成绩管理系统的角色是管理员、学生和教师，管理员对学生、教师、课程和班级信息进行维护，学生选择想要上的课程，查看所选的学分以及修改个人信息，教师决定上课的学生以及给学生学分。

根据以上的用户操作需求，将系统划分为如下三大功能，并对其模块的划分和功能进行描述。

管理员功能

登录：登录

学生管理：列表、增加、修改、删除

教师管理：列表、增加、修改、删除

课程管理：列表、增加、修改、删除

班级管理：列表、增加、修改、删除

学生功能

登录：登录

选课：选课

学分：查看

个人信息：修改

教师功能

登录：登录

选择学生：课程列表、学生列表、选择

公布成绩：课程列表、学生列表、成绩

整个系统的模块结构如图 13.24 所示。

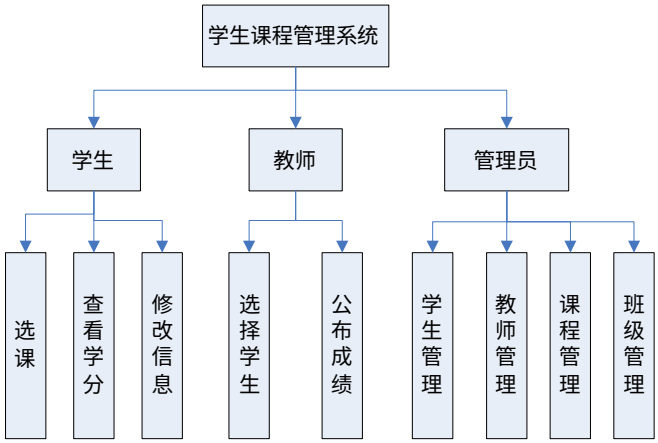

图 13.24　系统模块结构图

13.4.2　系统流程分析

本系统中的中心对象是学生和教师。根据以上的模块划分和功能分析可知，该系统的流程主要描述的是学生选择课程后，教师根据选课的学生决定选哪些学生，最后教师给学生学分。该系统的适用对象包括学生、教师和管理员，因此包括三个基本的流程。

图 13.25 描述的是管理员的操作流程：首先管理员要进行学生、教师、课程和班级数据的初始化，也就是说由管理员将学生和教师的信息，加入到系统的数据库中。当然，也可以不通过系统而直接通过数据库将已有的学生和教师信息导入，这种做法与前一种做法的效果相同。之后学生和教师就可以登录并使用系统了。在系统使用过程中，管理员进行管理工作。

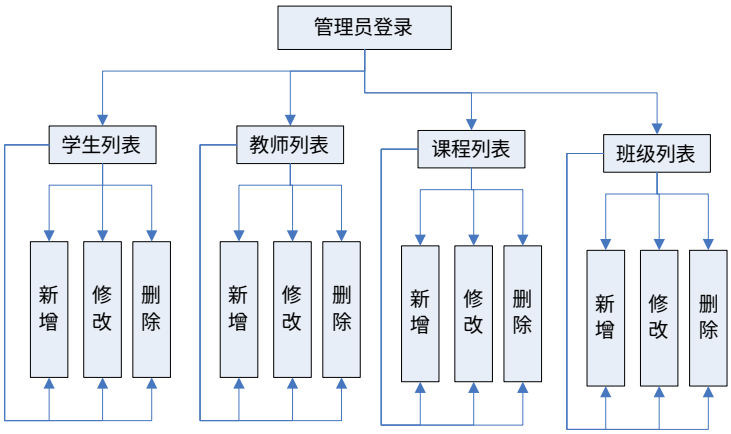

图 13.25　系统流程图—管理员

图 13.26 描述的是学生的操作流程：学生根据学生号和密码登录系统，初始的密码由管理员提供。学生登录系统后，可以修改个人信息、选课和查看学分。

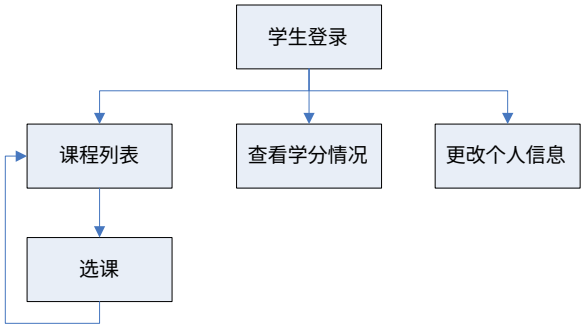

图 13.26　系统流程图—学生

图 13.27 描述的是教师的操作流程：教师根据教室号和密码登录系统，初始的密码由管理员提供。教师登录系统后，选择学习本课程的学生和给学生学分。

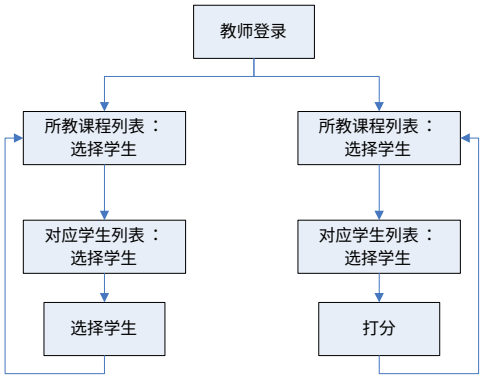

图 13.27　系统流程图—教师

13.5　系统设计

系统分析完成后，进入系统设计阶段，这是整个系统实现过程呢个中非常重要的一个阶段。

良好的系统设计将会使开发顺利地进行。

13.5.1 数据库逻辑结构设计

数据库设计是系统设计中非常重要的一个环节。数据是一切系统设计的基础，通俗的说数据库设计就像高楼大厦的根基一样，如果设计不合理、不完善，将在系统开发过程中、甚至到后期的系统维护、功能变更和功能扩充时，引起较多问题，严重时甚至要重新设计，重做大量已完成工作。

根据功能模块划分的结果可知，本系统的用户有三类：管理员、学生和教师。由于管理员、学生和教师的权限和操作功能大不相同，因此在本系统中需要分别进行数据记录。首先需要如下 3 个数据实体：

管理员数据实体：只需要记录管理员的登录名、姓名和密码，其中登录名和密码是管理功能模块登录验证时所必需的。

学生数据实体：包括学生号、密码、学生姓名、性别、学生所在系、籍贯、联系电话、电子邮件。这些信息中，密码、联系电话和电子邮件由学生自己进行维护，管理员在学生入学时根据填写的信息初始化学生信息，在以后的维护过程中，仅在特殊情况下对信息进行修改操作。

教师数据实体：包括教师号、密码、教师姓名、职称。这些信息有管理员初始化好，如果有所改动都要由管理员维护。

除了以上三个系统用户实体外，学生课程及成绩管理系统还要对学生课程和班级进行管理，这就又需要如下的两个数据实体：

课程数据实体：用于记录所有课程的基本信息，包括课程的课程号、课程名、学分、系别和预选修情况。这些信息由学校的工作人员已管理员的身份登录后进行维护。

班级数据实体：用于记录班级的基本信息，包括班级号、教师、课程、教室、和上课时间。这些数据由管理员进行录入和维护（如果与学校排课系统等结合，数据就由那些系统来提供）。

以上的 5 个实体是基本的数据实体。作为学生课程及成绩管理系统，还要记录学生选课和学分情况，因此又有如下的实体：

学生课程及成绩数据实体：包括学生号、所上课班级、是否被老师接收和所给学分。

根据以上的分析，设定每一个数据实体都有一个 ID 作为它的惟一标识，那么这六个数据实体的关联关系如图 13.28 所示（其中管理员数据实体相对独立，这里没有列出关系图）。

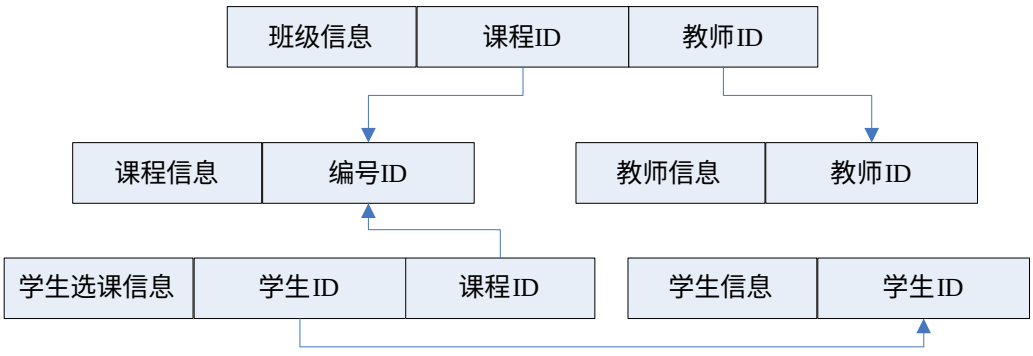

图 13.28　数据实体关系图

基于上面的设计，开始设计表，表与表之间相互关联，共同存储着系统所需要的数据。在设计数据库表的过程中，一般要遵循几条原则：

数据库的一个表最好只存储一个实体或对象的相关信息，不同的实体最好存储在不同的数据表中，如果实体还可以再划分，实体的划分原则是，划分后得实体比当前系统要开发实体的复杂度小；

数据表的信息结构一定要合适，表的字段的数量一般不要过多；

扩充信息和动态变化的信息一定要分别放在不同的表里；

多对多的表关系尽量不出现。

设计数据库表及表间关系，通过下图 13.29 表示。

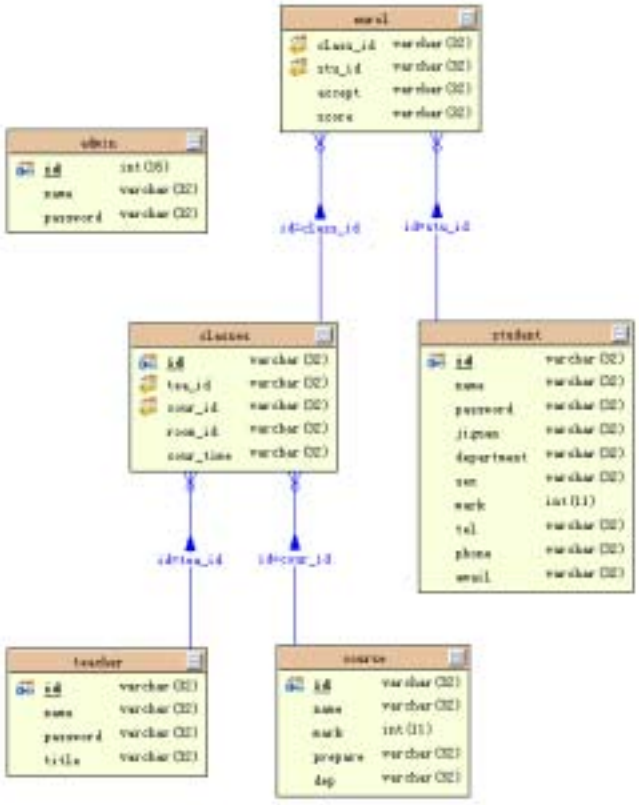

图 13.29　数据库表关系图

13.5.2　创建数据库

首先要创建一个数据库，可以使用 MySQL 的辅助图形化界面工具 SQLyog，这个工具的下载安装已经在前面章节介绍过了，如果读者还不熟悉 MySQL 或 SQLyog 这个工具，可以参考前面的相关章节。使用 SQLyog 连接到 MySQL，然后创建数据库 STU，并为新建的数据库添加用户并设置权限（单击菜单【Tools】|【User Manager】|【Add user】或直接按快捷键 Ctrl+U）。可以将对于这个新建数据库的所有权限全都赋给这个用户。接下来要在这个数据库中创建数据表，由前面的分析得知这个系统中需要建立 6 张数据表，它们分别为

管理员表（admin）：用于存放管理员用户的数据记录。

学生信息表（student）：用于存放学生的基本信息。

教师信息表（teacher）：用于存放所有上课教师的基本信息。

课程信息表（course）：用于存放所有开课课程的数据记录基本信息。

班级信息表（classes）：用于存放所有与班级相关的信息。

学生课程及成绩表（enrol）：用于存放所有学生课程及成绩信息。

这 6 张数据表的字段说明如表 13.1~表 13.6 所示。

表 13.1　admin管理员表

序号	字段	含义	类型
1	Id	管理员编号	Int
2	Name	姓名	Varchar
3	Password	密码	Varchar

表 13.2　student学生信息表

序号	字段	含义	*类型*
1	Id	学生编号	Varchar
2	Name	姓名	Varchar
3	Password	密码	Varchar
4	Jiguan	籍贯	Varchar
5	Department	所在系	Varchar
6	Sex	性别	Varchar
7	mark	学分	Int
8	tel	联系电话	Varchar
9	email	电子邮箱	Varchar

表 13.2　classes班级信息表

序号	字段	含义	类型
1	id	班级编号	Varchar
2	tea_id	姓名	Varchar
3	cour_id	密码	Varchar
4	room_id	教室号	Varchar
5	cour_time	上课时间	Varchar

表 13.3　course课程信息表

序号	字段	含义	*类型*
1	id	课程编号	Varchar
2	name	课程名	Varchar
3	mark	学分	Int
4	prepare	预选课程	Varchar
5	dep	所在系	Varchar

表 13.4　teacher教师信息表

序号	字段	含义	*类型*
1	id	教师编号	Varchar
2	name	姓名	Varchar
3	title	职称	Varchar
4	pssword	密码	Varchar

表 13.6　enrol学生课程及成绩信息表

序号	字段	含义	*类型*
1	stu_id	学生编号	Varchar
2	class_id	班级编号	Varchar
3	accept	是否被接收	Varchar

4	score	成绩	Varchar

13.5.3 创建表的脚本文件

根据数据库字段设计，编写的创建数据库表的 SQL 语句如下：
创建数据表 admin 的 SQL 语句：

```
CREATE TABLE `admin` (
    `id` int(16) NOT NULL,
    `name` varchar(32) default NULL,
    `password` varchar(32) default NULL,
    PRIMARY KEY  (`id`)
)
```

创建数据表 student 的 SQL 语句：

```
CREATE TABLE `student` (
    `id` varchar(32) NOT NULL,
    `name` varchar(32) default NULL,
    `password` varchar(32) default NULL,
    `jiguan` varchar(32) default NULL,
    `department` varchar(32) default NULL,
    `sex` varchar(32) default NULL,
    `mark` int(11) default NULL,
    `tel` varchar(32) default NULL,
    `phone` varchar(32) default NULL,
    `email` varchar(32) default NULL,
    PRIMARY KEY  (`id`)
)
```

创建数据表 teacher 的 SQL 语句：

```
CREATE TABLE `teacher` (
    `id` varchar(32) NOT NULL,
    `name` varchar(32) default NULL,
    `password` varchar(32) default NULL,
    `title` varchar(32) default NULL,
    PRIMARY KEY  (`id`)
)
```

创建数据表 course 的 SQL 语句：

```
CREATE TABLE `course` (
    `id` varchar(32) NOT NULL,
    `name` varchar(32) default NULL,
    `mark` int(11) default NULL,
    `prepare` varchar(32) default NULL,
    `dep` varchar(32) default NULL,
    PRIMARY KEY  (`id`)
)
```

创建数据表 classes 的 SQL 语句：

```
CREATE TABLE `classes` (
                `id` varchar(32) NOT NULL,
                `tea_id` varchar(32) default NULL,
                `cour_id` varchar(32) default NULL,
                `room_id` varchar(32) default NULL,
                `cour_time` varchar(32) default NULL,
                PRIMARY KEY  (`id`),
                KEY `FK_Reference_4` (`tea_id`),
                KEY `FK_classes` (`cour_id`),
                CONSTRAINT `classes_ibfk_1` FOREIGN KEY (`cour_id`) REFERENCES `course` (`id`),
                CONSTRAINT `FK_Reference_4` FOREIGN KEY (`tea_id`) REFERENCES `teacher` (`id`)
            )
```

创建数据表 enrol 的 SQL 语句：

```
CREATE TABLE `enrol` (
                `class_id` varchar(32) default NULL,
                `stu_id` varchar(32) default NULL,
                `accept` varchar(32) default NULL,
                `score` varchar(32) default NULL,
                KEY `FK_enrol` (`class_id`),
                KEY `FK_Reference_2` (`stu_id`),
                CONSTRAINT `enrol_ibfk_1` FOREIGN KEY (`class_id`) REFERENCES `classes` (`id`),
                CONSTRAINT `enrol_ibfk_2` FOREIGN KEY (`stu_id`) REFERENCES `student` (`id`)
            )
```

为了方便后面的开发，在这里先象数据库中 admin 表里插入一条数据，SQL 语句如下：

```
insert into `stu`.`admin` ( `id`, `name`, `password` ) values ( '1', 'admin', 'admin' )
```

13.5.4　目录和包结构

在进行程序设计和开发之前，要设计目录和包的结构。良好的结构会使代码逻辑清楚且容易阅读。一般一个设计良好的结构都有其共同的特点，就是逻辑清楚。本系统的目录结构如图 13.30 所示。

图 13.30　目录及包结构

在这个目录结构中，MyStuMan 是项目的根目录也是项目的名称。其下 src 目录用于存放原文件，所有的 Java 类都定义在这个文件夹下。WebRoot 目录是发布时网站的根目录，其下放置 JSP 页面，WEB-INF 目录下存放系统的配置文件，如 web.xml 等。
这个目录结构是通用的目录结构，读者可以根据需要进行相应的修改。

13.5.5　定义 HibernateUtil

本系统采用 Struts+Hibernate 技术进行开发，由 Hibernate 进行数据对象的操作，这里定义一个 HibernateUtil 类负责初始化 Hibernate。由它创建全局的 SessionFactory 实例，并且提供创建 Session 实例、关闭 Session 实例以及打开/关闭事务以及重新创建 SessionFactory 实例的实用方法。而且所有方法均为静态方法。
HibernateUtil 类是用 2 个 ThreadLocal 类型的属性以保持在一次请求过程中共享单一的 Session 和 Trasaction 实例，Session 和 Trasaction 实例可以跨越多个一次请求的多个方法，这有助于实现集合属性的延迟加载等 Hibernate 特性。HibernateUtil 代码如下：

```
package com.stuman.dao.hibernate;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
	private static final SessionFactory sessionFactory;

	static {
		try {
			// 创建 SessionFactory
			sessionFactory = new Configuration().configure()
					.buildSessionFactory();
		} catch (Throwable ex) {
			ex.printStackTrace();
			System.out.println("Initial SessionFactory creation failed.");
			throw new ExceptionInInitializerError(ex);
		}
	}

	public static final ThreadLocal tLocalsess = new ThreadLocal();

	public static final ThreadLocal tLocaltx = new ThreadLocal();

	// 取得 session
	public static Session currentSession() {
		Session session = (Session) tLocalsess.get();

		// 打开一个新的 session，如果当前的不可用.
		try {
			if (session == null || !session.isOpen()) {
```

```
                session = openSession();
                tLocalsess.set(session);
            }
        } catch (HibernateException e) {
            // 抛出 HibernateException 异常
            e.printStackTrace();
        }
        return session;
    }

    //关闭 session
    public static void closeSession() {

        Session session = (Session) tLocalsess.get();
        tLocalsess.set(null);
        try {
            if (session != null && session.isOpen()) {
                session.close();
            }

        } catch (HibernateException e) {
            //抛出 InfrastructureException 异常
        }
    }

     //开始事务
    public static void beginTransaction() {
        //声明 Transaction 类型对象 tx，并付初值
        Transaction tx = (Transaction) tLocaltx.get();
        try {
            if (tx == null) {
                tx = currentSession().beginTransaction();
                tLocaltx.set(tx);
            }
        } catch (HibernateException e) {
            // 抛出 InfrastructureException 异常
        }
    }

    //关闭事务
    public static void commitTransaction() {
        Transaction tx = (Transaction) tLocaltx.get();
        try {
            if (tx != null && !tx.wasCommitted() && !tx.wasRolledBack())
                tx.commit();
            tLocaltx.set(null);
            System.out.println("commit tx");
        } catch (HibernateException e) {
            // 抛出 InfrastructureException 异常
        }
    }

    //事务回滚
```

```
    public static void rollbackTransaction() {
        Transaction tx = (Transaction) tLocaltx.get();
        try {
            tLocaltx.set(null);
            if (tx != null && !tx.wasCommitted() && !tx.wasRolledBack()) {
                tx.rollback();
            }
        } catch (HibernateException e) {
            // 抛出 InfrastructureException 异常
        }
    }

    private static Session openSession() throws HibernateException {
        return getSessionFactory().openSession();
    }

    private static SessionFactory getSessionFactory() throws HibernateException {
        return sessionFactory;/
    }
}
```

因为在一次请求中需要共享单一的 Session 和 Trasaction 实例，因此不能在每个方法后关闭 Session，应该在一次请求全部处理完成后关闭 Session。为此，可以设计一个过滤器，在过滤器中统一关闭 Session。实现该功能的过滤器代码如下所示：

```
package com.stuman.dao.hibernate;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class CloseSessionFilter implements Filter {
    Log logger = LogFactory.getLog(this.getClass());

    protected FilterConfig config;

    //初始化方法
    public void init(FilterConfig arg0) throws ServletException {
        this.config = arg0;
    }

    //doFilter 方法 定义操作
    public void doFilter(
        ServletRequest request,
        ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
```

```
            try{
                chain.doFilter((HttpServletRequest)request, (HttpServletResponse)response);
            }
            finally{
                try{
                    HibernateUtil.commitTransaction();              //提交事务
                }catch (Exception e){
                    HibernateUtil.rollbackTransaction();            //回滚事务
                }finally{
                    HibernateUtil.closeSession();                   //关闭 Session
                }
            }
        }

        //destroy 方法
        public void destroy() {
            this.config = null;
        }
}
```

这样一来，在使用 Hibernate 时更加方便，不用每次为创建 SessionFactory 实例、Session 实例、或是关闭 Session 实例以及打开/关闭事务等操作单独写代码，只要调用上面类的方法就行了，从而简化了操作。

13.5.6 定义 SetCharacterEncodingFilter

在进行 Web 开发时，经常会遇到中文显示出现乱码的情况，这是因为 Java 内置的字符集与页面显示的字符集不一致造成的。为了解决这个问题，开发人员需要转化字符编码，但是如果对所有的输入输出信息都做编码转化显然比较麻烦。通过定义一个 Filter 来自动的实现字符编码的转化是一个比较好的方法。而且实现并不复杂，实现步骤如下：

1 定义 SetCharacterEncodingFilter 类

将这个类放在 com.stuman.util 包下，这个类要实现 Filter 接口，其代码如下：

```
package com.stuman.util;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class SetCharacterEncodingFilter implements Filter {
    protected String encoding = null;
    protected FilterConfig filterConfig = null;
    protected boolean ignore = true;

    //destroy 方法
    public void destroy() {
        this.encoding = null;
```

```
        this.filterConfig = null;
    }

     //选择  设置使用的字符编码
     public void doFilter(ServletRequest request, ServletResponse response,
                                  FilterChain chain)
     throws IOException, ServletException {
          //选择使用的字符编码
          if (ignore || (request.getCharacterEncoding() == null)) {
               String encoding = selectEncoding(request);
               if (encoding != null)
                    request.setCharacterEncoding(encoding);
          }
            //将控制权交给下一个 Filter
            chain.doFilter(request, response);
    }


    //将这个 filter 放置在服务中
    public void init(FilterConfig filterConfig) throws ServletException {
          this.filterConfig = filterConfig;
          this.encoding = filterConfig.getInitParameter("encoding");
          String value = filterConfig.getInitParameter("ignore");
          if (value == null)
               this.ignore = true;
          else if (value.equalsIgnoreCase("true"))
               this.ignore = true;
          else if (value.equalsIgnoreCase("yes"))
               this.ignore = true;
          else
               this.ignore = false;
    }

    //选择适当的字符编码
    protected String selectEncoding(ServletRequest request) {
          return (this.encoding);
    }
}
```

2　配置这个 Fiter

在 web.xml 文件中添加如下代码：

```
    <filter>
          <filter-name>SetCharsetEncodingFilter</filter-name>
          <filter-class>com.stuman.util.SetCharacterEncodingFilter</filter-class>
          <init-param>
               <param-name>encoding</param-name>
               <param-value>utf-8</param-value>
          </init-param>
    </filter>

    <filter-mapping>
          <filter-name>SetCharsetEncodingFilter</filter-name>
```

```
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

通过配置 Filter，就可以实现编码的自动转换了。

13.5.7　数据层设计

一般将业务逻辑分为服务层逻辑和持久化层逻辑。在实现业务层逻辑时，需要尽可能的保持层次之间的松散耦合，面向接口变成是业务逻辑设计一个最重要的原则。
本系统的持久化逻辑采用 Hibernate 作为中间件，并使用 DAO 设计模式实现。DAO 模式是 J2EE 核心模式中的一种，主要是在业务核心方法和具体数据源之间增加一层，这样就减少了两者的耦合。因为具体持久层数据源可能是多样化的，例如可能是 XML 或者是关系数据库，在具体的关系数据库中，也可能有不同的产品如 Oracle 或 MySQL（当然在本系统的开发过程中使用 MySQL），通过使用 DAO 模式，业务核心部分就无须涉及如何具体操作数据库，如图 13.31 所示。

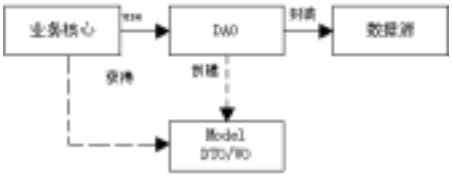

图13.31　DAO模式应用

每个持久化类对应一个 DAO，它实现了持久化类的创建、查询、更新及删除方法，即 CRUD（CREATE、RETRIEVE、UPDATE、DELETE）方法，以及其他访问持久化机制的方法。在相应的 DAO 实现中，调用 Hibernate API 访问持久层。这样只有特定于 Hibernate 的 DAO 实现需要依赖 Hibernate API，当改用其他的持久化机制或持久化中间件时，只需要创建新的 DAO 实现，无需更改应用中其他业务逻辑代码。这便是 DAO 模式的优点。

13.6　界面设计及实现

从本章第一节可知，本系统的界面共分为如下 4 个大的模块：

登录模块：此模块是用于不同系统角色的登录，也是系统的惟一入口。

管理员模块：此模块是用于管理员对学生、教师、课程和班级等信息的管理和维护，包括如下几个部分。

学生管理：增加、修改、删除学生。

教师管理：增加、修改、删除教师。

课程管理：增加、修改、删除课程。

班级管理：增加、修改、删除班级。

学生模块：此模块是学生管理操作界面，包括如下的几个部分。

选报课程：查看可选课程、选课。

查看成绩：查看自己的成绩。

个人信息：修改。

教师模块：此模块是教师管理操作界面，包括如下几个部分。

接收学生选课：查看、接收学生。

打分：查看、打分。

根据这些整体关系设计，下面对每一个部分给出主要的界面及其设计思路。

13.6.1 登录界面

系统的任何用户使用系统，都必须从系统的登录界面进入。这是任何一个信息管理系统的保密性的需要。根据之前的需求分析和概要设计可以知道，系统包括三类用户：学生、教师、和管理员。为了能够让三类用户使用用一个页面登录，在登录界面中，提供了选择登录用户类型的下拉框，由于使用的是 Struts 的标签，下拉框效果并不很好，但足以满足需要。再加上输入用户名和密码的输入框以及【提交】按钮组成了登录页面。效果如图 13.32 所示。

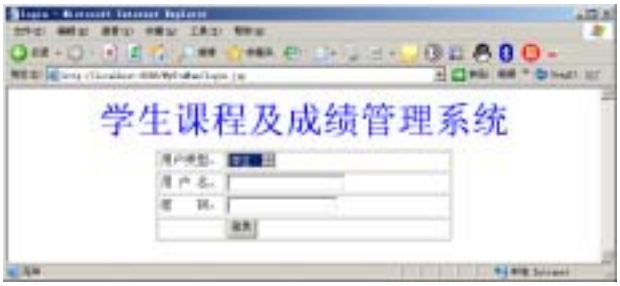

图 13.32 系统登录界面设计

在实现登录界面前，应该先创建一个 web 服务器默认的欢迎页面 index.jsp，以实现自动跳转到登录页面。index.jsp 的代码如下：

```
<%
	response.sendRedirect("login.jsp");
%>
```

接下来创建登录页面，登录页面 login.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html lang="true">
<head>
	<html:base />
	<title>login</title>
<style type="text/css">
<!--
.style1 {
	color: #0000FF;
	font-weight: bold;
	font-size: xx-large;
}
.style2 {color: #FF0000}
-->
	</style>
</head><body>
```

```
<html:form action="/login" method="post">
    <table width="100%" border="0" align="center">
        <tr align="center">
            <td>
                <p class="style1">
                    学生课程及成绩管理系统
                </p>
        </tr>
        <tr>
            <td align="center">
                <span class="style2"> <html:errors /> </span>
        </tr>
        <tr>
            <td align="center">
                <table border="1">
                    <tr>
                        <td align="center">
                            用户类型：
                        <td>
                            <html:select property="sort" multiple="true" size="1"
value="1">
                                <html:option key="a" value="1">学生</html:option>
                                <html:option key="b" value="2">教师</html:option>
                                <html:option key="c" value="3"> 管 理 员
</html:option>
                            </html:select>
                        </td>
                    <tr>
                        <td width="22%" align="center">
                            用&amp;nbsp;&amp;nbsp;户&amp;nbsp;&amp;nbsp;名：
                        </td>
                        <td width="74%">
                            <html:text property="username" />
                            <html:errors property="username" />
                        </td>
                    </tr>
                    <tr>
                        <td align="center">
                            密
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;码：
                        </td>
                        <td>
                            <html:password property="password" />
                            <html:errors property="password" />
                        </td>
                    </tr>
                    <tr>
                        <td align="center">&amp;nbsp;</td>
                        <td>
                            <html:submit value="登录" />
                        </td>
                    </tr>
                </table>
```

```
                    </td>
                </tr>
            </table>
        </html:form>
</body>
</html:html>
```

由于页面中使用到了 Sturts 的标签，所以在代码开始部分导入了 Sturtsde 标签库。至此登录页面就完成了。

13.6.2　管理员管理首页

管理员登录后，会得到欢迎信息表示登录成功，如果登录失败，则会有错误提示信息。管理员可以通过单击学生、教师、课程和班级这四个链接进入不同页面继续下一步的操作。效果如图 13.33 所示。

图 13.33　管理员管理首页

由于这个页面（admin.jsp）的实现简单，这里就不给出代码了。管理首页设计完成后，接下来就要分别开始设计各个功能页面的界面，首先从管理员管理学生的界面开始。

第 17 章　个性化定制系统——Ajax+Spring+Hibernate

本章将向读者介绍一个结合使用 Ajax、Spring 和 Hibernate 技术的例子。这个例子用于提供个性化相关服务（模拟 Google 的个性化主页：http://www.google.com/ig?hl=zh-CN）。本系统通过使用 Ajax 技术，使得页面的表现能力得到增强，Spring 和 Hibernate 的配合为系统的开发提供了强有力的支持。由于本书前面章节并没有讲述 Ajax 的相关知识，所以在本章的开始部分，用一节的篇幅来向读者介绍 Ajax 的基本知识，然后再进行系统的设计与实现。

17.1　Ajax 技术简介

Ajax 是一种很有用，而且强大好用的客户端和服务器端技术。本节中将通过解释什么是

Ajax，以及为什么要使用 Ajax 等几个问题来帮助读者逐步熟悉和了解 Ajax 技术。

17.1.1 什么是 Ajax

Ajax（Asynchronous JavaScript and XML），用中文来解释就是：异步 Javascript 与 XML。它不是一种单独的技术，而实际上它是几种技术的结合使用。每种技术本身都有其独特之处，把他们结合在一起就成了一个功能强大的“ 新技术 ”——Ajax。Ajax 包括：

使用 XHTML 和 CSS 样式表实现标准化呈现。
使用 DOM 模型实现动态显示和交互。
使用 XML 和 XSLT 进行数据的交换与处理。
使用 XMLHttpRequest 进行异步数据的读取。
使用 JavaScript 绑定和处理数据。

17.1.2 为什么使用 Ajax

为什么使用 Ajax 技术？下面比较一下传统的 Web 应用模型与 Ajax 模型即可得出结论。使用传统的 Web 应用模型时，界面上的用户行为或动作触发一个连接到 Web 服务器的 HTTP 请求。服务器完成处理后再返回一个 HTML 页面到客户端。这个过程有一个十分严重的问题，就是对于每一个动作用户都要等待。
Ajax 是使用客户端脚本与 Web 服务器交换数据的 Web 应用开发方法。Ajax 的工作原理相当于在用户和服务器之间增加了一个中间层，从而使用户操作与服务器响应异步化。这样，Web 页面不用打断交互流程进行重新加载，就可以动态地更新，而且可以只更新需要更新的部分（也就是页面的局部更新）。使用 Ajax，用户可以创建更加直接、更高可用性、更丰富的动态 Web 用户界面。也正是因此，越来越多的 Web 开发使用到了 Ajax 技术。
传统的 Web 应用模型与 Ajax 模型之间的差异通过图 17.1 和图 17.2 可以清晰的看出。

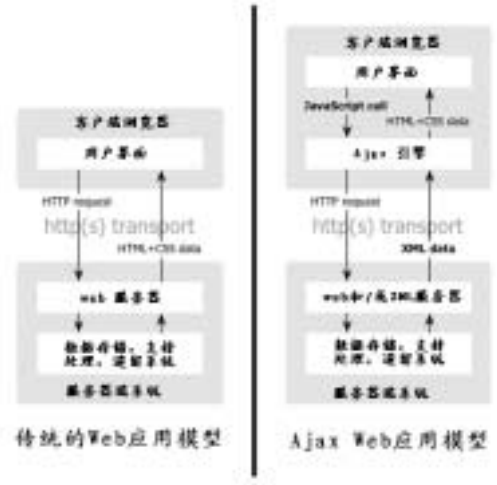

图 17.1　传统 Web 应用模型(左)与 Ajax 模型的(右)

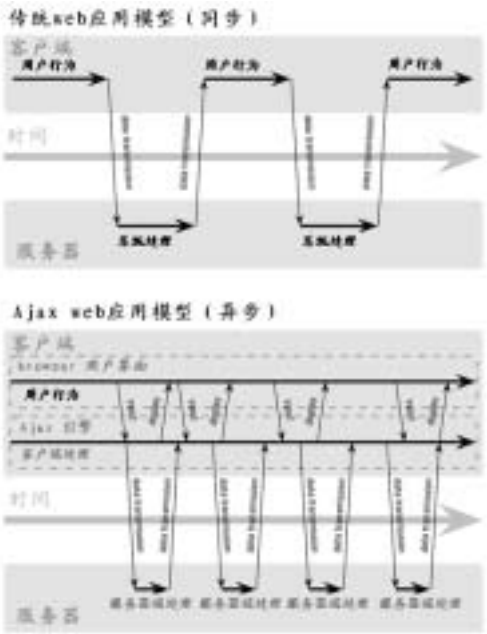

图 17.2　传统 Web 应用的同步交互(上)和 Ajax 异

步交互(下).

17.1.3 如何使用 Ajax

由于 Ajax 并不是一种单纯的技术，而需要几种技术的结合使用。所以想要充分发掘出 Ajax 的能力还是有相当难度的（比如 Google Map 的实现）。但是对于小型应用，Ajax 也并不像想象中的那么困难。在本章后面的系统实现部分将向读者讲解如何应用 Ajax 技术到自己的项目中。如果读者对 Ajax 技术感兴趣，可以查看更多的相关资料和文档。

17.2 系统概述

本章中要完成一个个性化定制系统，其主要特色在于显示给用户的界面。界面实现要模拟和简化 Google 的个性化主页。Google 个性化主页的效果如图 17.3 所示。

图 17.3 Google 个性化主页

使用过 Google 个性化主页的读者知道，只要单击页面左侧窗口中的某个【添加】按钮，就会在页面的右侧部分出现对应的一部分内容。本章中将这新添加进来的部分称作一个新添加的“小窗口”。而且用户可以随意拖动页面右侧部分中的各个“小窗口”。通过添加和拖动，用户可以得到自己需要显示的“小窗口”和“小窗口”的排列顺序。对于本系统，将功能简化，于是得到需要完成的主要功能有三个，分别为：

左侧的动态下拉列表。

“小窗口”的添加。

页面右侧“小窗口”的拖动效果。

实现这些功能的最好的方式就是使用 Ajax。通过 Ajax 令页面局部刷新，从而无需长时间的等待整个页面的刷新。

17.3 需求分析

本章中将要实现的例子其主要目的是：以一个个性化定制系统为基础，向读者讲解如何结合

使用 Ajax、Spring、Hibernate 于自己的项目开发中。而不是实现一个功能完善的个性化定制系统（这样的系统过于庞大和复杂）。因此本例只关注与实现主要的功能，所以在系统的实现过程中可能会忽略一些小功能（比如用户登录等）。如果读者有兴趣可以继续完善这个系统。在此将系统需求加以总结，得出系统需求列表如下：

页面左侧的列表可以隐藏或显示。

页面左侧的下拉列表内容由用户定制，并保存在数据库中。

页面左侧的下拉列表内容动态的从数据库中读取。

通过单击页面左侧的【添加】按钮，将对应内容（“ 小窗口 ”）动态的添加到页面右侧。

通过单击页面右侧某个“ 小窗口 ”右上角的【X】，将这个“ 小窗口 ”从页面的右侧删除。

页面右侧“ 小窗口 ”的拖动效果。

17.4　系统功能预览

需求确定之后需要对系统进行整体分析和设计。这包括系统功能的描述、对功能模块的划分和对系统流程的分析。下面首先对系统功能进行描述。

17.4.1　隐藏或显示页面左侧列表

通过单击浏览页面右侧部分的“ 隐藏/显示 ”链接，来实现页面左侧部分的隐藏和显示。效果如图 17.4 和图 17.5 所示。

图 17.4　显示页面左侧部分

图 17.5　隐藏页面左侧部分

17.4.2　定制列表内容

浏览页面左侧的下拉列表内容由用户定制，并保存在数据库中。用户通过图 17.6 所示的页面来定制分类。用户选中的复选框对应内容将被写入数据库，并在用户单击浏览页面左侧“ 资

源浏览 ” 链接时动态的列出。

图 17.6　定制

17.4.3　动态的下拉列表

当用户单击浏览页面左侧 “ 资源浏览 ” 链接时，将显示出用户定制的分类信息。这些内容都是动态的从数据库中读取的。如图 17.7 所示。

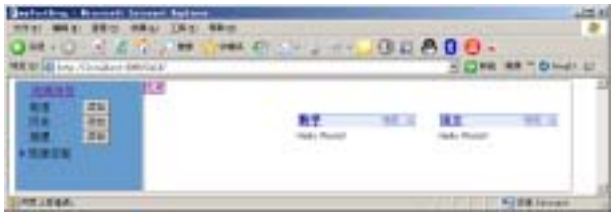

图 17.7　下拉列表

17.4.4　向页面右侧添加 “ 小窗口 ”

通过单击浏览页面左侧的【添加】按钮，将对应 “ 小窗口 ” 动态的添加到页面右侧。如图 17.8 所示，用户在图 17.7 所示页面的基础上，单击页面左侧的 “ 英语 ” 分类后的【添加】按钮，在页面右侧显示相应的 “ 英语 小窗口 ”。由于这个系统只是模拟功能的实现，而没有分类下的实际内容，所以在右侧显示的 “ 小窗口 ” 除了标题外内容都是一样的。

图 17.8　向右侧添加内容

17.4.5　删除页面右侧的 “ 小窗口 ”

用户单击浏览页面右侧某个 “ 小窗口 ” 右上角的【X】链接，可以将这个 “ 小窗口 ” 从页面的右侧删除。效果如图 17.9 所示。

图 17.9　删除“ 小窗口 ”

17.4.6　编辑“ 小窗口 ”

当用户单击页面右侧某个“ 小窗口 ”标题右侧的“ 编辑 ”链接时，“ 小窗口 ”的状态将发生改变，用户可以编辑这个分类下要显示的内容数量（尽管系统没有提供实际内容，但用户的编辑效果将在数据库的用户定制记录中体现出来）。效果如图 17.10 所示。

图 17.10　编辑“ 小窗口 ”

17.4.7　页面右侧“ 小窗口 ”的拖动效果

在网页上实现拖动效果是很令人兴奋的。图 17.11 和图 17.12 展示了一次拖动的效果。用鼠标左键单击一个“ 小窗口 ”的标题部分（页面上显示为淡蓝色的部分）。光标会变成“ 十字 ”形状，这时拖动就会得到如图 17.10 所示的效果，移动的“ 小窗口 ”此时将淡化显示。当用户松开鼠标左键时，“ 小窗口 ”将动态的移动到合适的位置，如图 17.11 所示效果。

图 17.11　开始拖动

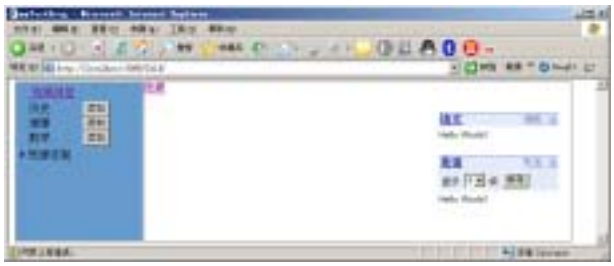

图 17.12　结束拖动

至此读者已经对本系统将要实现的功能有了基本了解，下面进行系统设计。

17.5　系统设计

系统设计在开发一个项目时起着关键的作用，本节将分几步来完成系统的设计。

17.5.1　数据库结构设计与实现

数据库设计是系统设计中非常重要的一个环节。数据是一切系统设计的基础，通俗的说数据库设计就像高楼大厦的根基一样。如果设计不合理，将在系统开发过程中、甚至到后期的系统维护、功能变更和功能扩充时，引起较多问题。严重时甚至要进行重新设计，重做大量已完成的工作。

1　数据库逻辑结构设计

根据前面需求分析和功能描述，可以得知本系统需要创建的实体如下：

用户实体：记录用户的基本信息。虽然系统需求中将用户登录功能省略掉了，但用户定制科目还是需要用户信息的，所以有必要定义用户实体。

科目实体：用于记录用户所要定制的分类科目信息，包括科目的名称、编号等信息。

用户定制信息实体：用于记录用户都定制了那些科目。包含用户编号、科目编号等信息。

以上的 3 个实体对应数据库中三张表。他们的关系如图 17.13 所示。

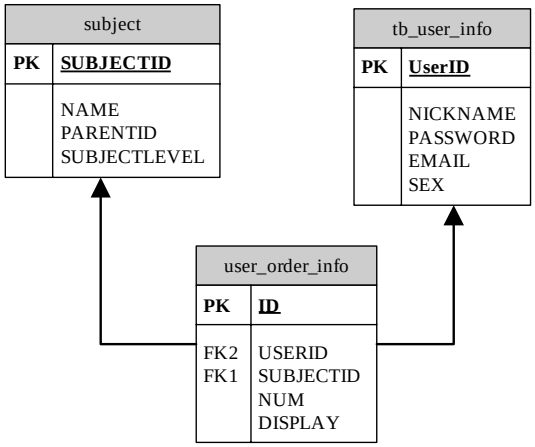

图 17.13　实体关系

2　创建数据库

首先要创建一个数据库，可以使用 MySQL 的辅助图形化界面工具 SQLyog。这个工具的下载安装已经在前面章节介绍过了。如果读者还不熟悉 MySQL 或 SQLyog 这个工具，可以参考前面的相关章节。使用 SQLyog 链接到 MySQL，然后创建数据库 STU，并为新建的数据库添加用户，并设置权限（单击菜单【Tools】|【User Manager】|【Add user】或直接按快捷键 Ctrl+U）。可以将对于这个新建数据库的所有权限全都赋给这个用户。

接下来要在这个数据库中创建数据表。由前面的分析得知这个系统中需要创建 3 张数据表，它们分别为：

用户信息表（TB_USER_INFO）：用于存放用户的信息。

科目表（SUBJECT）：用于存放分类科目信息。

用户定制信息表（USER_ORDER_INFO）：用于用户定制分类科目的信息。

这 3 张数据表的字段说明如表 13.1~13.3 所示。

表 13.1　TB_USER_INFO表

序号	*字段*	*含义*	类型
1	*USERID*	*用户ID*	INTEGER
2	*NICKNAME*	*用户名*	VARCHAR
3	*PASSWORD*	*密码*	VARCHAR
4	*EMAIL*	*用户Email*	VARCHAR
5	SEX	性别	INTEGER

表 13.2　SUBJECT表

序号	*字段*	*含义*	类型
1	*SUBJECTID*	*新闻编号*	VARCHAR
2	*NAME*	*新闻标题*	VARCHAR
3	*PARENTID*	*新闻内容*	VARCHAR
4	SUBJECTLEVEL	新闻添加时间	VARCHAR

表 13.3　USER_ORDER_INFO表

序号	*字段*	*含义*	类型
1	*ID*	*编号*	NUMBER
2	*USERID*	*用户ID*	NUMBER
3	*SUBJECTID*	*科目ID*	Varchar
4	*NUM*	*显示内容条数*	NUMBER
5	DISPALY	是否在页面右侧显示	Varchar

3　创建表的脚本文件

根据数据库字段设计，编写的创建数据库表的 SQL 语句如下：

创建数据表 TB_USER_INFO 的 SQL 语句：

```
CREATE TABLE `tb_user_info` (
                    `USERID` int(11) NOT NULL,
                    `NICKNAME` varchar(40) NOT NULL,
                    `PASSWORD` varchar(50) NOT NULL,
                    `EMAIL` varchar(100) NOT NULL,
                    `SEX` int(11) NOT NULL,
                    PRIMARY KEY  (`USERID`)
                  )
```

创建数据表 SUBJECT 的 SQL 语句：

```
CREATE TABLE `subject` (
              `SUBJECTID` varchar(8) NOT NULL,
              `NAME` varchar(50) NOT NULL,
              `PARENTID` varchar(8) default NULL,
```

```
    `SUBJECTLEVEL` varchar(1) NOT NULL,
    PRIMARY KEY  (`SUBJECTID`)
  )
```

创建数据表 USER_ORDER_INFO 的 SQL 语句：

```
CREATE TABLE `user_order_info` (
    `ID` int(11) NOT NULL,
    `USERID` int(11) NOT NULL,
    `SUBJECTID` varchar(8) NOT NULL,
    `NUM` int(11) NOT NULL default '5',
    `DISPLAY` varchar(2) NOT NULL,
    PRIMARY KEY  (`ID`),
    KEY `USERID` (`USERID`),
    KEY `SUBJECTID` (`SUBJECTID`),
    CONSTRAINT  `user_order_info_ibfk_2`  FOREIGN  KEY  (`USERID`)
        REFERENCES `tb_user_info` (`USERID`),
    CONSTRAINT  `user_order_info_ibfk_1`  FOREIGN  KEY  (`SUBJECTID`)
        REFERENCES `subject` (`SUBJECTID`)
  )
```

17.5.2　目录和包结构

在进行程序设计和开发时，要设计目录和包的结构。良好的结构会使代码逻辑清楚且容易阅读。一般，一个设计良好的结构都有其共同的特点——逻辑清晰。本系统的目录结构如图 13.34 所示。

图 13.14　目录及包结构

在这个目录结构中，Self 是项目的根目录也是项目的名称。其下 src 目录用于存放源文件，所有的 Java 类都定义在这个文件夹下。indi 目录是发布时网站的根目录，其下放置 HTML 和 JSP 页面，WEB-INF 目录下存放系统的配置文件，如 web.xml 等。读者可以根据需要对目录结构进行相应的修改。

17.5.3 数据层设计

一般将业务逻辑分为服务层逻辑和持久化层逻辑。在实现业务层逻辑时，需要尽可能的保持层次之间的松散耦合。面向接口编程是业务逻辑设计的一个最重要的原则。
本例的持久化逻辑由 Spring 和 Hibernate 共同完成，这使得开发过程中可以将数据库操作全都交给 Spring 和 Hibernate 来处理，简化了开发过程。
首先在 applicationContext.xml 文件中进行数据库链接的配置，关键部分代码如下：

```
    <!--
        PropertyPlaceholderConfigurer 作为一个 bean factory post-processor 实现
        将 BeanFactory 定义中的属性值放置到另一个单独的 Java Properties 格式的文件中
        使得用户不用对 BeanFactory 的主 XML 定义文件进行复杂和危险的修改，就可以定制一些基本的属性
    -->
    <bean id="placeholderConfig"
            class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location">
            <!--这里使用 init.properties 文件-->
            <value>/init.properties</value>
        </property>
    </bean>

    <!--从上面指定的文件中读取配置 dataSource 的各个属性值  进行配置-->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName">
            <value>${datasource.driverClassName}</value>
        </property>
        <property name="url">
            <value>${datasource.url}</value>
        </property>
        <property name="username">
            <value>${datasource.username}</value>
        </property>
        <property name="password">
            <value>${datasource.password}</value>
        </property>
        <property name="maxActive">
            <value>${datasource.maxActive}</value>
        </property>
        <property name="maxIdle">
            <value>${datasource.maxIdle}</value>
        </property>
        <property name="maxWait">
            <value>${datasource.maxWait}</value>
        </property>
        <property name="defaultAutoCommit">
            <value>${datasource.defaultAutoCommit}</value>
        </property>
    </bean>

    <!-- 配置 sessionFactory，注意这里引入的包  -->
    <bean                                                                    id="sessionFactory"
```

```
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
            <property name="dataSource">
                <ref local="dataSource" />
            </property>
            <property name="mappingResources">
                <list>
                    <value>org/g12a/hibernate/data/Subject.hbm.xml</value>
                    <value>org/g12a/hibernate/data/TbUserInfo.hbm.xml</value>
                    <value>org/g12a/hibernate/data/UserOrderInfo.hbm.xml</value>
                </list>
            </property>
            <property name="hibernateProperties">
                <props>
                    <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                    <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
                    <prop key="hibernate.jdbc.fetch_size">${hibernate.jdbc.fetch_size}</prop>
                    <prop key="hibernate.jdbc.batch_size">${hibernate.jdbc.batch_size}</prop>
                </props>
            </property>
        </bean>

        <bean id="transactionManager"
             class="org.springframework.orm.hibernate3.HibernateTransactionManager">
            <property name="sessionFactory">
                <ref local="sessionFactory" />
            </property>
        </bean>
```

上述文件中配置 SessionFactory 和 DataSource 时使用的 value 值都是从文件 init.properties 中读取的。这样做使得在更改配置时，更加方便同时也避免了直接修改 applicationContext.xml 文件可能会导致的错误。init.properties 文件内容如下：

```
datasource.driverClassName=com.mysql.jdbc.Driver
datasource.url=jdbc:mysql://127.0.0.1:3306/individuation
datasource.username=root
datasource.password=root
datasource.maxActive=10
datasource.maxIdle=2
datasource.maxWait=120000
datasource.defaultAutoCommit=true
datasource.whenExhaustedAction=1
datasource.validationQuery=select 1 from dual
datasource.testOnBorrow=true
datasource.testOnReturn=false

hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.jdbc.batch_size=25
hibernate.jdbc.fetch_size=50
hibernate.show_sql=true
hibernate.hbm2ddl.auto=create-drop
```

这里出现的各个属性及其对应的值在前面的章节中已经多次介绍，这里不再赘述。

17.5.4 创建 AppContext 类

为了高效的使用 Spring 提供的功能，首先创建 AppContext 类，用于取得 ApplicationContext，其代码如下：

```
package org.g12a.bit.resource;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class AppContext {

	private static AppContext instance;                                    // 定义静态实例 方便使用
	private AbstractApplicationContext appContext;

	//获得 AppContext 实例
	public synchronized static AppContext getInstance() {
		if (instance == null) {
			instance = new AppContext();
		}
		return instance;
	}

	// 获得 AppContext 实例的方法
	private AppContext() {
		this.appContext = new ClassPathXmlApplicationContext(
				"/applicationContext.xml");
	}

	public AbstractApplicationContext getAppContext() {
		return appContext;
	}
}
```

17.6 创建持久化类和映射文件

由于使用到了 Hibernate 所以需要创建持久化类和映射文件。

17.6.1 Subject 的持久化类及映射文件

Subject 的映射文件代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
	<!--持久化类与表的对应-->
	<class name="org.g12a.hibernate.data.Subject" table="SUBJECT" schema="G12A">
		<!--影射属性与列-->
		<id name="subjectid" type="java.lang.String">
			<column name="SUBJECTID" length="8" /><!--
```

```
            <generator class="sequence" />
        --></id>
        <many-to-one name="subject" class="org.g12a.hibernate.data.Subject" fetch="select">
            <column name="PARENTID" length="8" />
        </many-to-one>
        <property name="name" type="java.lang.String">
            <column name="NAME" length="50" not-null="true" />
        </property>
        <property name="subjectlevel" type="java.lang.String">
            <column name="SUBJECTLEVEL" length="1" not-null="true" />
        </property>
        <set name="subjects" inverse="true">
            <key>
                <column name="PARENTID" length="8" />
            </key>
            <one-to-many class="org.g12a.hibernate.data.Subject" />
        </set>
    </class>
</hibernate-mapping>
```

创建 Subject 的持久化类时，为了方便以后的修改维护，首先创建一个抽象类 AbstractSubject，其代码如下：

```
package org.g12a.hibernate.data;
import java.util.Set;
public abstract class AbstractSubject    implements java.io.Serializable {

    private String subjectid;                       // 分类科目编号
    private Subject subject;                        // 科目对象
    private String name;                            // 科目名称
    private String subjectlevel;                    // 科目级别
    private Set subjects;                           // 集合

    /** 默认构造函数 */
    public AbstractSubject() {
    }

    /** 包含编号的构造函数 */
    public AbstractSubject(String subjectid) {
        this.subjectid = subjectid;
    }

    // 数据访问方法
    public String getSubjectid() {
        return this.subjectid;
    }

    public void setSubjectid(String subjectid) {
        this.subjectid = subjectid;
    }

    public Subject getSubject() {
        return this.subject;
    }
```

```
    public void setSubject(Subject subject) {
        this.subject = subject;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSubjectlevel() {
        return this.subjectlevel;
    }

    public void setSubjectlevel(String subjectlevel) {
        this.subjectlevel = subjectlevel;
    }

    public Set getSubjects() {
        return this.subjects;
    }

    public void setSubjects(Set subjects) {
        this.subjects = subjects;
    }

     // 重写 equlas 方法，用于判断两个 subject 是否相等
     public boolean equals(Object subject) {
          if (this.subjectid.equals(((Subject) subject).getSubjectid() )) {
               return true;
          } else {
               return false;
          }
     }
}
```

注意这个类重写了 equals 方法，这个方法中认为只要两个 Subject 对象的 subjectid 属性值相同，那么这两个对象就相同。接下来创建 Subject 类，它继承自 AbstratcSubject，其代码如下：

```
package org.g12a.hibernate.data;
public class Subject extends AbstractSubject implements java.io.Serializable {

    /** 默认构造函数 */
    public Subject() {
    }

    /** 有编号的构造函数 */
    public Subject(String subjectid) {
     super(subjectid);
    }

     // 拷贝一个 subject 对象
```

```
    public void copy(Subject subject) {
        this.setName(subject.getName());
        this.setSubjectlevel(subject.getSubjectlevel());
        this.setSubjects(subject.getSubjects());
        this.setSubject(subject.getSubject());
    }
}
```

至此 Subject 的持久化对象和映射文件就创建完成了。

17.6.2 UserOrderInfo 的持久化类及映射文件

创建 UserOrderInfo 的映射文件，其代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <!--持久化类与表的对应-->
    <class name="org.g12a.hibernate.data.UserOrderInfo" table="USER_ORDER_INFO"
schema="G12A">
        <id name="id" type="java.lang.Long">
            <column name="ID" precision="22" scale="0" />
            <generator class="sequence">
              <param name="sequence">seq_user_rsc_collect_id</param>
            </generator>
        </id>

        <!--多对一关系影射-->
        <many-to-one name="tbUserInfo" class="org.g12a.hibernate.data.TbUserInfo"
fetch="select">
            <column name="USERID" precision="22" scale="0" not-null="true" />
        </many-to-one>

        <!--多对一关系影射-->
        <many-to-one name="subject" class="org.g12a.hibernate.data.Subject" fetch="select">
            <column name="SUBJECTID" precision="22" scale="0" not-null="true" />
        </many-to-one>

        <!--影射属性与列-->
        <property name="num" type="java.lang.Long">
            <column name="NUM" not-null="false" />
        </property>

        <property name="display" type="java.lang.String">
            <column name="DISPLAY" length="2" not-null="true" />
        </property>

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

与创建 Subject 的持久化类时相似，为了方便以后的修改维护，首先创建一个抽象类 AbstractUserOrderInfo，其代码如下：

```
package org.g12a.hibernate.data;
```

```
public abstract class AbstractUserOrderInfo implements java.io.Serializable{

    private Long id;                                    // 编号
    private TbUserInfo tbUserInfo;                      // 用户信息
    private Subject subject;                            // 分类科目
    private Long num;                                   // 显示数量
    private String display;                             // 是否显示标识

    // 默认构造函数
    public AbstractUserOrderInfo()
    {
        num = new Long(5);
        display = "0";
    }

    // 包含 Id 的构造函数
    public AbstractUserOrderInfo(Long id)
    {
        this.id = id;
        num = new Long(5);
        display = "0";
    }

    // 属性访问方法
    public Long getId() {
        return this.id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public TbUserInfo getTbUserInfo() {
        return this.tbUserInfo;
    }

    public void setTbUserInfo(TbUserInfo tbUserInfo) {
        this.tbUserInfo = tbUserInfo;
    }

    public Subject getSubject()
    {
     return this.subject;
    }

    public void setSubject(Subject subject)
    {
     this.subject = subject;
    }

    public Long getNum()
    {
     return this.num;
```

```
    }

    public void setNum(Long num)
    {
     this.num = num;
    }

    public String getDisplay()
    {
     return this.display;
    }

    public void setDisplay(String display)
    {
     this.display = display;
    }
}
```

再创建 Subject 类，它继承自 AbstractUserOrderInfo，其代码如下：

```
package org.g12a.hibernate.data;
public class UserOrderInfo extends AbstractUserOrderInfo implements java.io.Serializable{

    public UserOrderInfo()
    {
        super();
    }

    public UserOrderInfo(Long id)
    {
        super(id);
    }

    // 将属性拼接成一个字符串
    public String toString(){
            return "User id: "+this.getTbUserInfo().getUserid()+", Subject Name:
"+this.getSubject().getName()
            +", User Nickname: "+this.getNum()+",Display: "+this.getDisplay();
        }
}
```

至此 UserOrderInfo 的持久化对象和映射文件就创建完成了。

17.6.3 TbUserInfo 的持久化类及映射文件

TbUserInfo 的映射文件代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <!--持久化类与表的对应-->
    <class name="org.g12a.hibernate.data.TbUserInfo" table="TB_USER_INFO"
schema="G12A">
        <id name="userid" type="java.lang.Long">
```

```
            <column name="USERID" precision="22" scale="0" />
            <generator class="sequence">
              <param name="sequence">seq_user_info_userid</param>
            </generator>
        </id>

        <!--影射属性与列-->
        <property name="nickname" type="java.lang.String">
            <column name="NICKNAME" length="40" not-null="true" />
        </property>
        <property name="password" type="java.lang.String">
            <column name="PASSWORD" length="50" not-null="true" />
        </property>
        </property>
        <property name="email" type="java.lang.String">
            <column name="EMAIL" length="100" not-null="true" />
        </property>
        <property name="sex" type="java.lang.Long">
            <column name="SEX" precision="22" scale="0" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

TbUserInfo 持久化类的创建过程与 Subject 和 UserOrderInfo 的持久化类创建过程相似，这里就不再给出代码了。