Eclipse Web 开发从入门到精通（实例版）

目录

第 3 篇　综合案例

前言

如今，Eclipse 越来越成为众多 Java 程序开发者首选的集成开发环境。层出不穷的插件和应用不断丰富着 Eclipse 的世界。在 SUN、IBM 等公司的积极推动下，“开源”之势在 Java 社区中日新月异，Hibernate、Struts、HSQLDB 等一大批优秀的开源框架脱颖而出，同时基于这些成熟框架而构建的成功企业应用也越来越多。

在实际的企业应用中，面对如此众多的优秀框架，通常需要解决两个问题：应该选择什么框架和如何使用这些框架。本书的目录是对第一个问题很好的回答，它总结了目前基于 Eclipse 平台的所有优秀框架，范围涉及从数据持久层、应用逻辑层到用户表示层的所有方面。我们每天都在不停地做着各种 DEMO，面对一个新的框架，第一步要做的是能调通一个 Hello World 级别的 DEMO。每当这时我们都希望能有一篇“step by step”的文章。

本书的目的就是力求让读者尽可能快地熟悉如何基于 Eclipse 平台进行企业应用开发。我们不求数学上的“一题多解”，但求寻找一种最快的方法解决问题。我们希望在本书的帮助下，让公司的新员工在三天之内熟悉 Eclipse 成为可能。

本书特点

1. 实例讲解，易于学习

书中的每一章都精选了经典的实例进行讲解，每个实例都是一个完整的应用。

2. 讲解通俗，步骤详细

本书的讲解始终贯穿着“跟我做”的思想，认真记录下键盘鼠标的每一个动作，加上丰富的插图和备注说明，让每个知识点都一目了然。

3. 配视频演示光盘，加速学习

配书光盘中包含了所有的源代码，以方便读者使用。同时，光盘中还配带了作者专门制作的典型配置的多媒体演示，让读者快速入门。

4. 完善的服务

作者的 E-mail 是 hongwulian@gmail.com，如果您在学习的过程中遇到什么困难，可以给作者发信，作者会及时回复。

本书内容

第 1 章带领读者了解 Eclipse 平台，包括下载并安装 Eclipse、Eclipse 的汉化，并且基于 Eclipse 完成了第一个 Java 实例。

第 2 章介绍 Eclipse 强大的代码重构功能。掌握这些技巧可以大大提高开发者的开发效率。

第 3 章介绍几个经典的 Eclipse 插件，包括 XML 文件的编写、直接查看字节码等基本内容。它对实际的应用开发很有益处。

第 4 章介绍如何在 Eclipse 中进行资源构建。内容包括 Eclipse 和 Ant 的集成，常用 Ant 操作等内容。

第 5 章以学生成绩管理系统为例，介绍如何基于 HSQLDB 进行数据库应用开发。

第 6 章以第 5 章为基础，介绍如何基于 JBoss 插件进行 Web 应用的开发。

第 7 章以在线留言板为例，介绍如何基于 Eclipse 开发 Struts 应用。

第 8 章以图书管理系统为例，介绍如何基于 Hibernate 框架进行应用开发。

第 9 章以第 8 章为基础，介绍如何使用 JUnit 框架对图书管理系统进行单元测试。

第 10 章详细介绍 JBossAOP 插件，并实现了第一个 AOP 实例。

第 11 章介绍 CVS 版本控制的使用。CVS 越来越成为团队开发不可或缺的工具。

第 12～14 章通过 3 个综合实例的开发全面应用本书涉及的开发技术，以达到进一步提高的目的。

读者对象

本书具有知识全面、实例典型、指导性强等特点，力求以全面的知识性及丰富的实例来指导读者透彻学习 Eclipse 各方面的技术。本书适合如下读者：

- 有一定 Java 基础的开发人员；
- JSP 开发人员；
- Web 开发人员；
- Eclipse 初、中级读者；
- J2EE 程序员；
- 培训学校的老师和学生；
- 大专院校的老师和学生。

本书作者

本书由强锋科技统筹，由连洪武编写。其他参与编写、资料整理及光盘制作的人员有王龙、王拥东、吴善才、徐砚颖、尹健慧、詹涵林、张薇、张小强、张运端、赵玉荣、郑慧、朱博、朱朝坤、邹小红、陈强、陈燕、丁凤霞、丁礼友、范忠诚、黄俊灿、贾伟、李喜彤、林垚、尚文谊、孙亮亮、唐崇敏、陶则熙等。在此对大家的辛勤工作一并表示感谢！

作者

2007 年 5 月

第 5 章 数据库开发实例——学生成绩管理系统

HSQLDB 是一个开源的纯 Java 嵌入式关系数据库管理系统，小巧方便，具有标准的 SQL 语法和 Java 接口，可以作为内存数据库、独立数据库和 C/S 数据库，支持索引、事务处理、Java 存储过程、完整性引用和约束等功能。

本章介绍 Eclipse 环境下的 HSQLDB 数据库应用开发，包括 HSQLDB 数据库的安装和配置、SqlExplorer 数据库插件的安装和配置、常见数据库操作的封装，最后通过学生成绩管理系统介绍了基于 HSQLDB 进行数据库应用开发的具体步骤。

5.1 HSQLDB 数据库的使用

5.1.1 下载并安装 HSQLDB 数据库

在使用 HSQLDB 数据库之前，本小节首先介绍 HSQLDB 数据库的下载和安装。与大多数 Java 应用程序一样，只需解压缩安装包即可完成 HSQLDB 数据库的安装。

跟我做

（1）登录 HSQLDB 的官方网站 http://www.hsqldb.org，下载 HSQLDB 数据库的安装包 hsqldb_1_8_0_x.zip。

（2）将下载的安装包解压缩到设定的安装目录，如 d:\hsqldb。

（3）将 D:\hsqldb\lib 目录下的 hsqldb.jar 文件加入到 CLASSPATH 环境变量中，HSQLDB 安装完毕。安装后其目录结构如图 5-1 所示。

图 5-1 HSQLDB 数据库的目录结构

%注意：设置 CLASSPATH 环境变量的方法参见 1.1 节，所有的 HSQLDB 组件如数据库引擎、服务器进程、JDBC 驱动程序、文档以及一些实用工具都放在 hsqldb.jar 文件中。

5.1.2 使用 Memory 模式运行 HSQLDB

下面介绍 HSQLDB 的几种运行模式。

- 独立服务器模式：类似于其他关系数据库的标准客户机/服务器数据库配置，允许出现使用 TCP 套接字的并发连接。
- 独立 Web 服务器模式：作为 Web 服务器通过 HTTP 接受 SQL 查询，也能作为任何标准 Web 容器中的 Servlet 来运行。由于 HTTP 是无状态的，所以本模式中不存在事务。
- 单机模式：是许多嵌入式应用程序的首选模式，该模式下应用程序使用 JDBC 创建一个数据库连接，HSQLDB 引擎也运行在该应用程序中。
- Memory 模式：所有数据库表和索引都放在内存中，数据不进行外存储，没有持久性。

本小节将以 Memory 模式为例，介绍如何基于 HSQLDB 数据库进行应用的开发。

跟我做

（1）启动 Eclipse，创建名字为 hsqldbdemo 的 Java 工程，并创建 Java 类 MemoryDB.java。

%注意：切记将 hsqldb.jar 加到工程的构建路径上。

（2）编辑 MemoryDB.java 文件。输入如下代码：

```
try {
    //加载 HSQLDB 数据库 JDBC 驱动
            Class.forName("org.hsqldb.jdbcDriver");
    //在内存中建立临时数据库 score，用户名为 sa，密码为空
            Connection connect = DriverManager.getConnection("jdbc:hsqldb:mem:score",
                    "sa", "");
    System.out.println("在此行上设置一个断点");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
        }
```

在内存中建立临时数据库“score”，用户名为“sa”，密码为空。上述程序片断是典型的通过 JDBC 连接数据库的方法，其中数据库 URL“jdbc:hsqldb:mem:score”中的“mem”部分定义了 HSQLDB 数据库工作在 Memory 模式下。一旦跟数据库的连接建立后，数据库引擎就启动起来了，接下来即可创建 Table 表。

5.2 使用 SQLExplorer 插件连接数据库

SQLExplorer插件可以通过JDBC访问常用的关系数据库，同时也支持像Hibernate这样的工具访问数据库。其官方站点为http://sourceforge.net/projects/eclipsesql。

本节介绍如何使用 SQLExplorer 插件，查看 5.1.2 小节建立的 score 内存数据库的具体内容。首先介绍 SQLExplorer 插件的安装，然后介绍 SQLExplorer 插件的具体使用方法。

工作在内存模式下的 HSQLDB 数据库，会随着程序的退出而关闭，所以在下面的操作中 MemoryDB 要始终保持运行状态。本章在程序中设置断点，调试运行，使程序保持运行状态。

跟我做

（1）打开 MemoryDB.java 文件，在程序行“System.out.println（“在此行上设置一个断点”）;”前设置一个断点。

（2）右击“MemoryDB.java”文件，在快捷菜单中选择【调试方式】|【Java 应用程序】命令，MemoryDB.java 程序调试运行至断点处，建立了内存数据库 score。

（3）单击【窗口】菜单，依次选择【打开透视图】|【其它…】命令，打开【选择透视图】对话框，选择“SQLExplorer”，打开 SQLExplorer 透视图，如图 5-2 所示。

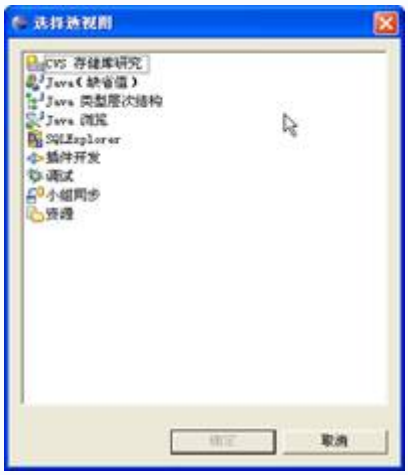

图 5-2　选择 SQLExplorer 透视图

SQLExplorer 透视图有 7 项内容，分别如下：

- Aliases 别名，用来标识数据库连接串。
- Connection Info 连接信息，用来显示连接数据库时的相关信息，如数据库产品名称、版本、JDBC 驱动程序的名称、版本、用户名、连接串、是否自动提交等。
- Connections 显示活动的连接情况。
- Database Structure View 显示数据库结构。
- Drivers 配置驱动程序用。
- SQL History 执行 SQL 的历史记录。
- SQL Results 执行 SQL 的结果集。

（4）打开如图 5-3 所示的 Drivers 视图，右击“HSQLDB In-Memory”，在快捷菜单中选择【Change the selected Driver】命令，打开 Modify Driver 窗口。

（5）选择【Extra Class Path】选项卡，单击【Add】按钮，在【打开】窗口中选择 d:\hsqldb\lib\hsqldb.jar，将 HSQLDB 数据库的驱动程序加入到 classpath 中。

（6）在【Example URL】文本框中输入“jdbc:hsqldb:mem:score”，单击【确定】按钮，如图 5-4 所示。这时，Drivers 视图中的“HSQLDB In-Memory”由 变成 ，表示 HSQLDB 数据库的驱动程序配置成功。

图 5-3　Drivers 视图

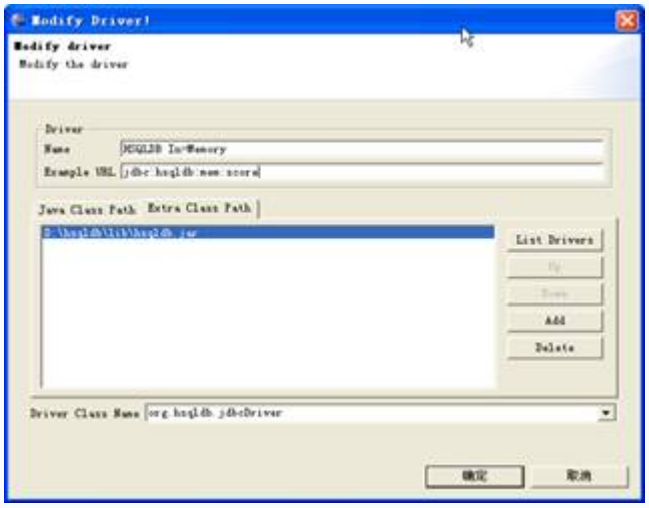

图 5-4　配置数据库驱动

（7）打开 SQLExplorer 插件的 Aliases 别名视图，单击【创建】图标 ，打开【Create new Alias】对话框。

（8）在【Name】文本框中输入“hsqlMemoryDB”，选择 HSQLDB In-Memory 驱动，在【URL】文本框中输入“jdbc:hsqldb:mem:score”，在【User Name】文本框中输入“sa”，单击【确定】按钮，如图 5-5 所示。在 Aliases 别名视图中出现刚建立的“hsqlMemoryDB”连接。

（9）右击“hsqlMemoryDB”，在快捷菜单中选择【Open…】命令，弹出有关数据库连接的确认框，可以更改用户名与密码，也可以设置是否自动提交，这里保持所有的选项为默认值。

（10）单击【确定】按钮，在 Database Structure View 视图中即可看到 Database，展开 Database 树形结构，如图 5-6 所示。

图 5-5　创建数据库连接别名

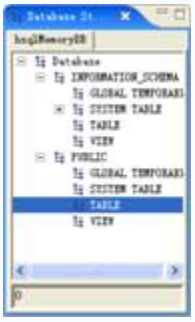

图 5-6　Database Structure View 视图

5.3　创建 Score 成绩表

至此通过 SQLExplorer 插件已经建立了与 HSQLDB 数据库的连接。本节通过在 SQLExplorer 中编辑和运行 SQL 脚本建立 Score 表，为后续几节的数据库操作做准备。

5.3.1　编写脚本

跟我做

（1）打开 SQLExplorer 插件的 Connections 视图，其中显示当前数据库的连接情况，这里有一个活动的连接，如图 5-7 所示。

图 5-7　Connections 视图

（2）单击 Connections 视图中的按钮，选择【New SQL Editor】命令，创建一个新的 SQL 编辑器。

（3）在 SQL 编辑器中输入如下 SQL 语句。

```
CREATE TABLE Score
  (SNO CHAR(7)  NOT NULL,
  CNO CHAR(6) NOT NULL,
  GRADE NUMERIC(4,1),
 PRIMARY KEY(SNO,CNO));
```

该 SQL 语句在 score 内存数据库中创建一张名为 Score 的表。该表包括 3 个字段：学号（SNO）、课程

编号（CNO）、课程分数（GRADE），并且以 SNO、CNO 两个字段作为主键。

5.3.2 在 SQLExplorer 中运行脚本

SQL Editor 是 SQLExplorer 插件提供的功能强大、使用方便的 SQL 语句编辑器，如图 5-8 所示。SQL Editor 提供编辑 SQL 语句，选中并执行部分 SQL 语句，打开和保存 SQL 脚本文件等功能。本小节介绍如何在 SQLExplorer 中运行 SQL 语句。

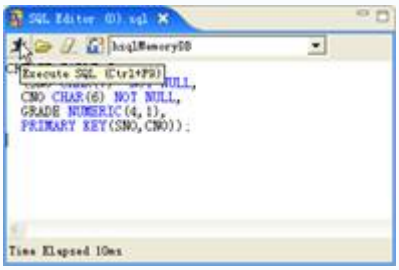

图 5-8 SQL Editor

跟我做

（1）单击【Execute SQL】按钮，执行所输入的 SQL 语句。

%注意：如果 SQL Editor 中存在多条 SQL 语句，首先选中想要执行的语句，然后单击【Execute SQL】按钮，如图 5-9 所示。

（2）从打开 SQLExplorer 插件的 Database Structure View 视图中可以看到刚才创建的表格。选中该表，可以看到该表的详细结构，如图 5-10 所示。

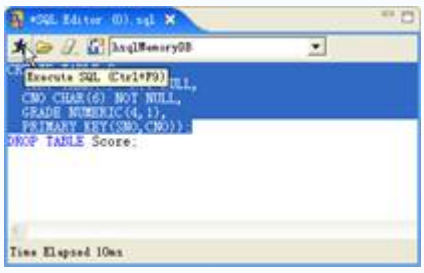

图 5-9 执行多条 SQL 语句中的某一条

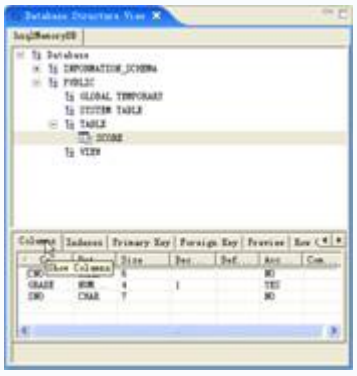

图 5-10 新创建的 Score 表

（3）打开 SQLExplorer 插件的 SQL History 视图，可以看到执行过的所有 SQL 语句列表，如图 5-11 所示。

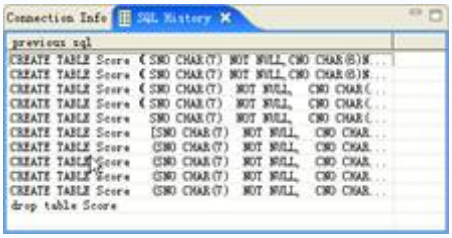

图 5-11　SQL History 视图

5.4　使用 JavaBean 映射成绩表

对象关系映射（ORM）是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。其本质就是将数据从关系数据库的二维表格形式改换一种形式，以更加容易理解的面向对象形式来表现具有某种关系的数据。本节将遵循 ORM 的思想，以 Score 表为例说明如何以 JavaBean 的方式来映射 HSQLDB 数据库中的 Score 表。

5.4.1　实现 Score 类

表 Score 共包括 3 个字段：SNO（学号）、CNO（课程编号）、GRADE（分数）。下面创建一个 JavaBean 来映射这个表。

跟我做

（1）右击“hsqldbdemo”工程的“src”源文件夹，在快捷菜单中选择【新建】|【包】命令，打开【新建 Java 包】窗口。

（2）在【名称】文本框中输入“hsqldb.javabean”，单击【完成】按钮，建立名为“hsqldb.javabean”的 package。

（3）在包“hsqldb.javabean”中创建 Java 类 Score.java。

（4）编辑 Score.java 文件，输入如下代码：

```
package hsqldb.javabean;
//映射 HSQLDB 数据库 Score 表的 JavaBean 类
public class Score {
    // 学号
    private String SNO;
```

```
    // 课程编号
    private String CNO;
    // 课程分数
    private float GRADE;
}
```

Score 类为标准的 JavaBean，为 Score 类声明了 3 个属性：SNO、CNO、GRADE，分别表示学号、课程编号和课程分数。

5.4.2 添加 getter/setter 方法

为了能够实现查询、修改等数据库常见操作，需要为 Score 类加入 getter/setter 方法。

跟我做

（1）打开 Score.java 文件，右击文件中的任何空白区域，在快捷菜单中选择【源代码】|【生成 Getter 和 Setter】命令，打开【生成 Getter 和 Setter】窗口，如图 5-12 所示。

（2）单击【全部选中】按钮，全部选中 3 个属性，单击【确定】按钮。生成完整 Score 类代码如下：

```
package hsqldb.javabean;
//映射 Score 表的 Javabean 类
public class Score {
    // 学号
    private String SNO;
    // 课程编号
    private String CNO;
    // 课程分数
    private float GRADE;
  //CNO 属性的 getter 方法
    public String getCNO() {
        return CNO;
    }
  //CNO 属性的 setter 方法
    public void setCNO(String cno) {
```

```
            CNO = cno;
        }
    //GRADE 属性的 getter 方法
        public float getGRADE() {
            return GRADE;
        }
    //GRADE 属性的 setter 方法
        public void setGRADE(float grade) {
            GRADE = grade;
        }
    //SNO 属性的 getter 方法
        public String getSNO() {
            return SNO;
        }
    //SNO 属性的 setter 方法
        public void setSNO(String sno) {
            SNO = sno;
        }
}
```

图 5-12 【生成 Getter 和 Setter】窗口

为 Score 类的 3 个属性添加 Getter/Setter 方法。Score 类为标准的 JavaBean。

5.5 使用 ScoreDAO 管理成绩

DAO 是数据访问接口 Data Access Object 的简称，在业务逻辑与数据库资源之间。DAO 是一种常用的设计模式，可以用来封装数据库的驱动、数据库 URL、用户名和密码。以后要更改数据库的类型，如把 MSSQL 换成 Oracle，则只需要更改 DAOFactory 里面的相关信息即可。另外，DAO 把对数据库的操作如最基本的查询、更新、删除和插入全部封装在里面，如加入一个学生一门课的成绩，只要调用 DAO 中的 insertScore（Score score）方法即可，省去了编写复杂的 SQL 语句的麻烦，使得操作数据库的动作更加方便。

跟我做

（1）建立 ScoreDAOFactory 类。通过该工厂类建立了和数据库的连接，可以关闭与数据库的连接，另外通过该工厂类可以取得一个 ScoreDAO 的实例。代码如下：

```
package hsqldb.dbo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
//创建 ScoreDAO 类的工厂类
public class ScoreDAOFactory {
    //HSQLDB 数据库的 Driver 名称
        public static final String DRIVER = "org.hsqldb.jdbcDriver";
    //将建立的内存数据库的 URL
        public static final String URL = "jdbc:hsqldb:mem:score";
    //Connection 对象，表示到数据库的连接
        private static Connection connection = null;
        /**
         * 建立到内存数据库的连接
         * @return
         */
        public static Connection createConnection() {
                if (connection == null) {
                        try {
```

```
        //加载数据库驱动程序
                Class.forName(DRIVER);
        //建立到数据库的连接
                connection = DriverManager.getConnection(URL, "sa", "");
            } catch (SQLException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }
    /**
     * 释放到内存数据库的连接
     */
    public static void closeConnection() {
        if (connection != null)
            try {
        //释放到数据库的连接
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
    }
    /**
     * 取得一个 ScoreDAO 的实例
     * @return
     */
    public static ScoreDAO getScoreDAO() {
```

```
            return new ScoreDAO();
        }
}
```

该类为 ScoreDAO 的工厂类，用来取得一个 ScoreDAO 的实例。

（2）建立 ScoreDAO 类，封装常见的数据库操作。编写代码如下：

```
package hsqldb.dbo;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import hsqldb.javabean.Score;
//ScoreDAO 类，用来封装对数据库的常见操作
public class ScoreDAO {
        // 数据库的 JDBC 连接
        private Connection connection;
      //  Statement 对象
        private Statement statement;
        // 需要执行的 SQL 语句
        private String sql;
        /**
         * 构造函数，建立到 score 数据库的连接，同时在 score 数据库中建立 Score 表
         */
        public ScoreDAO() {
                // 建立到 score 数据库的连接
                connection = ScoreDAOFactory.createConnection();
                try {
            //创建 statement 对象
                        statement = connection.createStatement();
            //创建 Score 表的 SQL 语句
```

```
                sql = "CREATE TABLE Score (SNO CHAR(7)  NOT NULL,CNO CHAR(6) NOT NU
LL,GRADE NUMERIC(4,1),PRIMARY KEY(SNO,CNO))";
            //执行 SQL 语句
                statement.execute(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
}
```

首先取得数据库的连接，然后执行“CREATE TABLE”SQL 语句，创建 Score 表。该表具有 3 个字段：学号、课程编号和分数。

5.5.1 添加 InsertScore 方法增加成绩

“INSERT INTO”语句用来向数据库中插入新的记录。为了简化插入记录操作，完全以面向对象的方式实现对数据库的插入操作，本小节介绍如何对插入语句进行封装。

跟我做

封装 INSERT INTO 标准 SQL 语句。编写代码如下：

```
/**
 * 将 Score 的一个对象插入到数据库中
 *
 * @param score
 */
public void insertScore(Score score) {
//将参数 score 类的各个属性拼接成插入记录的 SQL 语句
        sql = "INSERT INTO SCORE VALUES(" + "\'" + score.getSNO() + "\',"
                + "\'" + score.getCNO() + "\'," + score.getGRADE() + ")";
        try {
```

```
        //程序 INSERT INTO 语句为 Score 类插入一条记录，记录的三个字段值对应着 score 类的三个
属性
                statement.execute(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

将 Score 类的属性拼接成 INSERT 语句，这样进行数据库的插入操作只需传入一个 Score 类的实例，完全以面向对象的方式往数据库中插入记录。

5.5.2 添加 SelectScore 方法查询成绩

查询操作是数据库的常见操作，这里通过对 SELECT SQL 语句的封装实现对指定记录的查询以及遍历查询两个方法。

跟我做

（1）对指定记录的查询。在建立表格 Score 时，指定了以 SNO 和 CNO 作为该表的主键。该方法实现了查询 SNO 和 CNO 字段值与给定 Score 对象 SNO 和 CNO 属性值相同的记录。代码如下：

```
    /**
     * 在数据库中查询包含某个 Score 对象信息的记录
     *
     * @param score
     * @return
     */
    public ResultSet selectScore(Score score) {
    //查询后的结果集
        ResultSet result = null;
    //将参数 Score 类的各个属性拼接成查询记录的 SQL 语句
        sql = "select * from score where SNO=" + score.getSNO() + "and CNO="
                + score.getCNO();
        try {
```

```
        //执行查询，返回与 Score 类的 SNO、CNO 属性值相同的记录
                result = statement.executeQuery(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return result;
        }
```

（2）对所有记录的查询。为了更加方便地查询数据库目前记录情况，这里实现了简单返回所有记录的遍历查询。编写代码如下：

```
/**
     * 查询数据库中的所有记录
     *
     * @return
     */
    public ResultSet selectAll() {
      //查询后的结果集
            ResultSet result = null;
        //查询所有记录的 SQL 语句
            sql = "select * from score";
            try {
        //执行查询，返回所有的记录
                result = statement.executeQuery(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return result;
        }
```

5.5.3 添加 DeleteScore 方法删除成绩

“DELETE”语句用来从数据库中删除记录。为了简化删除记录操作，完全以面向对象的方式实现对数据库的删除操作，本小节介绍如何对 DELETE 语句进行封装。

跟我做

封装 DELETE SQL 语句，删除数据库中 SNO 和 CNO 字段值与指定的 Score 对象的 SNO、CNO 值相同的记录。编写代码如下：

```
/**
 * 删除某个数据库记录
 *
 * @param score
 */
public void deleteScore(Score score) {
//将参数 Score 类的各个属性拼接成删除记录的 SQL 语句
        sql = "delete from score where SNO=" + score.getSNO() + "and CNO="
                + score.getCNO();
        try {
    //执行删除操作的 SQL 语句
            statement.execute(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

5.5.4 添加 UpdateScore 方法修改成绩

“UPDATE”语句用来更新数据库中的某条记录。为了简化更新操作，完全以面向对象的方式实现对数据库的更新，本小节介绍如何对 UPDATE 语句进行封装。

跟我做

封装 UPDATE SQL 语句将数据库中某条记录更新为指定的 Score 类的 SNO、CNO 和 GRADE 值。编写代码如下：

```
/**
 * 更新数据库中的某条记录
 *
 * @param oldScore
 * @param newScore
 */
public void updateScore(Score oldScore, Score newScore) {
//将数据库中满足条件的记录更新为 newScore 类的 SNO、CNO 和 GRADE 属性值
        sql = "update score set SNO=" + newScore.getSNO() + ",CNO="
                + newScore.getCNO() + ",GRADE=" + newScore.getGRADE()
                + " where SNO=" + oldScore.getSNO() + "and CNO="
                + oldScore.getCNO();
        try {
    //执行数据库更新语句
            statement.executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

5.6 编写测试客户端

采用 DAO 模式的目的就在于简化对数据库的操作。本节将以一个操作数据库的实例来说明以 DAO 方式访问数据库的具体步骤，从而深刻体会 DAO 模式的实用性。

跟我做

（1）通过 ScoreDAOFactory 取得一个 ScoreDAO 的实例，同时在内存中创建了 score 数据库，并且建立了具有 3 个字段的 Score 表。

（2）准备 3 个 Score 类的对象，分别为 firstScore、secondScore 和 thirdScore。编写代码如下：

```
    //firstScore 实例，
   Score firstScore = new Score();
   //CNO 字段值为 34
   firstScore.setCNO("34");
   //SNO 字段值为 1
        firstScore.setSNO("1");
  //GRADE 字段值为 2.5
        firstScore.setGRADE((float) 2.5);
  // secondScore 实例
        Score secondScore = new Score();
  //CNO 字段值为 45
        secondScore.setCNO("45");
  //GRADE 字段值为 67.9
        secondScore.setGRADE((float) 67.9);
  //SNO 字段值为 2
       secondScore.setSNO("2");
  // thirdScore 实例
        Score thirdScore = new Score();
  //CNO 字段值为 78
        thirdScore.setCNO("78");
  //SNO 字段值为 3
        thirdScore.setSNO("3");
  //GRADE 字段值为 89
        thirdScore.setGRADE((float) 89.0);
```

（3）将 firstScore 插入到数据库中。编写代码如下：

```
// 通过 ScoreDAO 的实例执行插入操作
scoreDAO.insertScore(firstScore);
// 查询数据库中的所有记录
        result = scoreDAO.selectAll();
// 输出所有记录信息
        info(result);
```

运行结果如图 5-13 所示。

图 5-13　插入 firstScore 记录的执行结果

（4）将 secondScore 插入到数据库中。编写代码如下：

```
// 通过 ScoreDAO 的实例执行插入操作
scoreDAO.insertScore(secondScore);
// 查询数据库中的所有记录
        result = scoreDAO.selectAll();
// 输出所有记录信息
        info(result);
```

运行结果如图 5-14 所示。

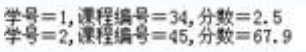

图 5-14　插入 secondScore 记录的执行结果

（5）将 secondScore 记录修改为 thirdScore。编写代码如下：

```
//通过 ScoreDAO 的实例执行更新操作
scoreDAO.updateScore(secondScore, thirdScore);
//查询数据库中的所有记录
        result = scoreDAO.selectAll();
//输出记录信息
        info(result);
```

运行结果如图 5-15 所示。

图 5-15　更新 secondScore 后的执行结果

（6）将 firstScore 记录删除。编写代码如下：

```
//通过 ScoreDAO 的实例执行删除操作
scoreDAO.deleteScore(firstScore);
//查询数据库中的所有记录
    result = scoreDAO.selectAll();
//输出记录信息
    info(result);
```

运行结果如图 5-16 所示。

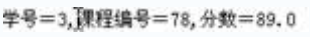

图 5-16　删除 firstScore 后的执行结果

以上简单的实例实现了数据库的查询、插入、删除和更新操作。整个操作完全以面向对象的形式进行，屏蔽了复杂的 SQL 语句，提高了程序的可读性和开发效率。

第 6 章 Web 开发实例——学生成绩管理系统的改进

J2EE 技术是目前使用最广泛的 Web 应用开发技术。JBoss 推出的 Eclipse IDE 开发工具支持 J2EE 的 Web 和 EJB 开发，提供 Ant 和 Xdoclet 自动提示功能，还提供 Hibernate、JBoss AOP 的开发，内置 JSP、HTML 和 XML 编辑器，提供方便快捷的开发向导提高 J2EE 应用开发的效率。

本章讲述如何利用 JBoss Eclipse IDE 开发 Web 版本的学生成绩管理系统，包括 JBoss Eclipse IDE 插件的安装和配置、JSP 文件的开发实例、Servlet 的开发实例，以及将开发的 Web 应用发布到 JBoss 应用服务器上的具体步骤。

6.1 下载并安装 JBoss 插件

Eclipse 提供一种简单的插件安装方法——安装/更新机制。经过第一次手工安装插件后，Eclipse 可以随时检查该插件是否有新版本，如果存在就会下载并安装新版本的插件。本节介绍如何通过 Eclipse 的安装/更新机制安装 JBoss-IDE，并自动更新插件。

跟我做

（1）启动 Eclipse。

（2）选择【帮助】|【软件更新】|【查找并安装】命令，打开【安装/更新】对话框。

（3）选中【搜索要安装的新功能部件】单选按钮，如图 6-1 所示。

图 6-1 【安装/更新】对话框

（4）单击【下一步】按钮打开【安装】对话框，单击对话框右侧的【新建远程站点】按钮，打开【新建更新站点】对话框。

（5）将站点的名称设置为JBoss Eclipse IDE，URL设置为http://download.jboss.org/ jbosside/updates/stable，如图 6-2 所示。

图 6-2　新建更新站点

（6）单击【确定】按钮返回【安装】对话框。选中刚才建立的 JBoss EclipseIDE 站点，单击【完成】按钮。

（7）展开 JBoss EclipseIDE 树，选中【JBossIDE 1.6.0 GA】复选框，如图 6-3 所示。

（8）单击【下一步】按钮，Eclipse 会询问是否同意 JBoss EclipseIDE 的许可条例。如果同意，选择【我接受许可协议中的条款】。

图 6-3　【更新】对话框

（9）单击【下一步】按钮，Eclipse 将列出所有将要安装的特征（Feature），如图 6-4 所示。

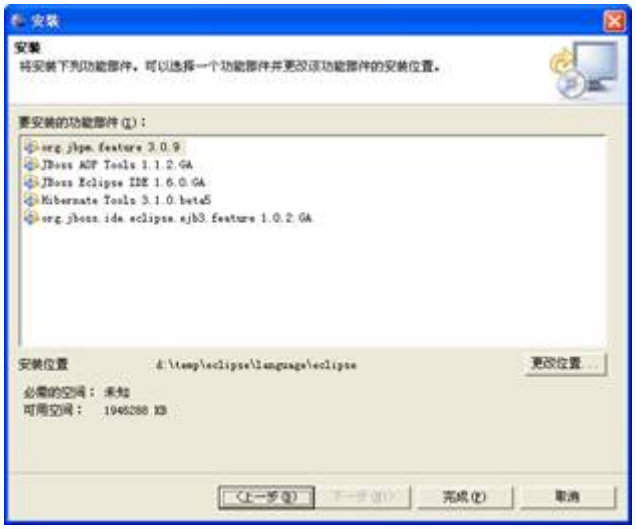

图 6-4　将要安装的特征

（10）单击【完成】按钮开始安装。

（11）在所有的特征下载完毕后，Eclipse 会提示是否想要安装每个特征。这里选择【全部安装】选项。

（12）在安装完成后，会提示是否重新启动 Eclipse。单击【是】按钮重新启动，安装完毕。

6.2　配置并运行 JBoss 应用服务器

JBoss是纯Java的Web应用服务器，为了保证JBoss服务器的正常运行，在安装JBoss之前首先要确保系统已经安装了JDK。可以从http://labs.jboss.com/portal/jbossas/download/ index.html下载JBoss应用服务器，本章选用JBoss 4.0.4 版本。

跟我做

（1）将下载的压缩包解压至本地磁盘，例如 C:\jboss 4.0.4。解压后的 JBoss 目录结构如图 6-5 所示。

%说明：bin 目录主要包含 run.jar、shutdowm.jar 等文件，用于启动、停止服务器脚本；client 目录主要包含与客户端相关的文件；docs 目录主要包含 JBoss 服务器的文档；server 目录主要包含与服务器有关的配置文件。

（2）运行 bin 目录下的 run.bat 文件，如果 DOS 界面出现类似如图 6-6 所示的信息就说明 JBoss 服务器已经启动。

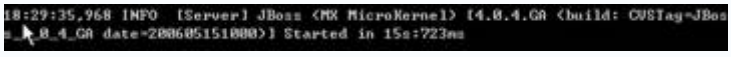

图 6-5　JBoss 目录结构　　图 6-6　JBoss 成功启动信息

（3）单击工具栏中的【调试】按钮，选择【调试】命令，打开【调试】对话框。

（4）在【调试】对话框中选择 JBoss 4.0.x（本章采用的是 JBoss 4.0.4），单击【新建】按钮。对话框右侧出现配置信息交互界面。

（5）在【名称】文本框中输入“JBoss 4.0.4”，在【JBoss 4.0.x Home Directory】文本框中选择 JBoss 应用服务器的根目录为 C：\jboss-4.0.4，【Server Configuration】选择“default”，如图 6-7 所示。

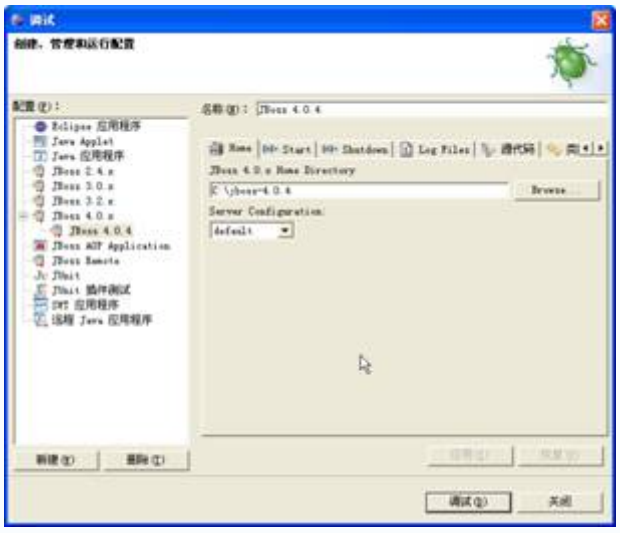

图 6-7 【调试】对话框

（6）单击【调试】按钮，JBoss 应用服务器开始启动，JBoss 服务器启动信息如图 6-8 所示。JBoss 应用服务器安装完毕。

图 6-8 JBoss 服务器启动信息

6.3 在 Eclipse 中开发 Jsp

JBoss 应用服务器配置完成后，为了开发学生成绩管理 Web 应用系统，在图 6-9 中，选择 J2EE 1.4 Project，新建一个名字为 ScoreWeb 的工程。

另外，在第 5 章中介绍了 HSQLDB 的内存数据库运行模式，本章 Web 应用系统采用 HSQLDB 的独立数据库模式。

6.3.1 Eclipse 内置 JSP 编辑器的使用

Eclipse 内置了 JSP 编辑器，功能强大的 JSP 编辑器可以降低 JSP 开发的难度，增加页面开发的速度。JSP 编辑器的属性窗口可以修改 HTML 标签的相应属性，通过大纲视图可以方便地添加子标签和兄弟标签，同时编辑器具有强大的代码辅助功能。下面讲解常用的方法。

跟我做

（1）利用 Eclipse 的属性视图，如图 6-10 所示，将鼠标放在每个 HTML 标签上时，属性窗口中就会出现该标签的所有属性。

图 6-9　新建 J2EE 1.4 工程

图 6-10　属性视图

%提示：可以在属性视图中直接编辑相应属性的值。

（2）在大纲视图中方便地添加每个 HTML 标签的属性和子标签、兄弟标签。

（3）JSP 编辑器提供强大的代码辅助功能，如图 6-11 所示。

图 6-11　代码辅助功能

6.3.2 启动数据库和创建表格

通过命令行

```
java org.hsqldb.Server –database.0 file:Scoredb –dbname.0 scoredb
```

可以启动 hsqldb 数据库。通过 hsqldb 的数据库管理器，可以执行 SQL 语句，完成创建表格、查询等数据库常用操作。

跟我做

（1）在环境变量中加入 hsqldb.jar，将%JAVA_HOME%/bin 路径设置在 Path 环境变量中。

（2）打开 CMD 窗口，输入如下命令：

```
java org.hsqldb.Server –database.0 file:Scoredb –dbname.0 scoredb
```

出现如图 6-12 所示信息，表示数据库已经成功启动。

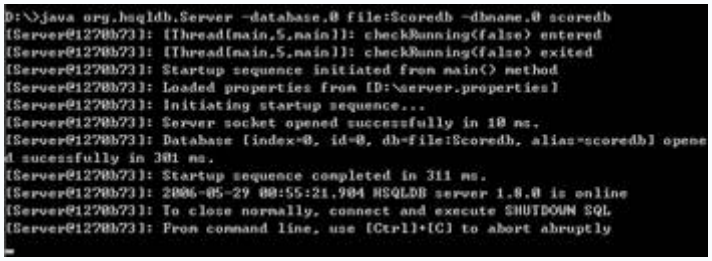

图 6-12　HSQLDB 数据库启动画面

这样就创建了一个数据为 Scoredb，别名为 scoredb 的数据库。

（3）通过 HSQLDB 自带的数据库管理工具管理数据库。在另一个 CMD 窗口输入如下命名：

```
java org.hsqldb.util.DatabaseManager
```

在如图 6-13 所示的窗口中输入如下连接信息。

（4）单击【OK】按钮，如果连接成功，出现如图 6-14 所示的 HSQL 数据库管理窗口。

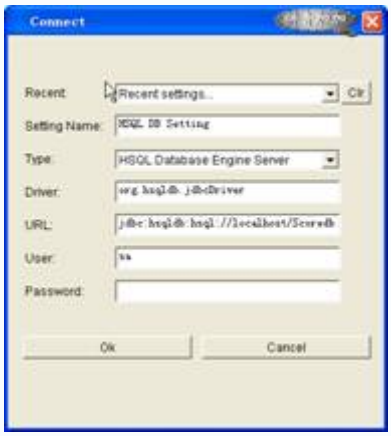

图 6-13　HSQLDB 连接信息

图 6-14　HSQL 数据库管理器

（5）在 SQL 语句编辑器中输入如下 SQL 语句：

```
CREATE TABLE Score
  (SNO CHAR(7)  NOT NULL,
  CNO CHAR(6) NOT NULL,
  GRADE NUMERIC(4,1),
  PRIMARY KEY(SNO,CNO));
```

（6）单击【Execute】按钮，建立表格 Score。

至此为下面章节的学习做好了所有的准备工作。

6.3.3　创建 scoreForm.jsp 录入成绩

在数据提交页面中输入相关信息，如图 6-15 所示。单击【提交】按钮后，在 HSQLDB DatabaseManager 中输入如下 SQL 语句：

```
Select  *  from  score;
```

可以看到 HSQLDB 数据库中的记录内容，如图 6-16 所示。

图 6-15　数据提交表单

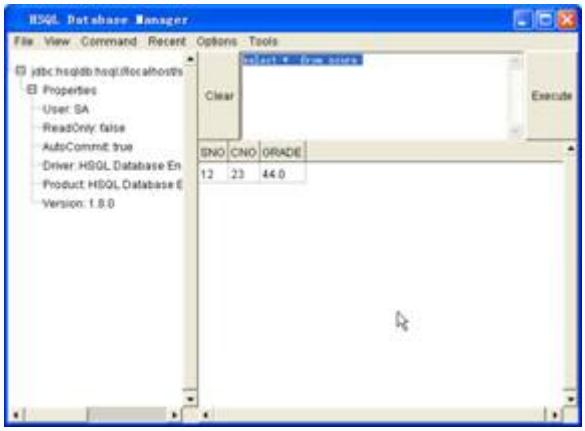

图 6-16　提交后的数据库状态

通过上面的准备，下面介绍在 JBoss Eclipse IDE 中开发 JSP 页面的具体步骤。

跟我做

（1）右击已经建立的 ScoreWeb 工程，在快捷菜单中选择【新建】|【源文件夹】命令，打开【新建源文件夹】对话框。

（2）在【文件夹名】文本框中输入“src”，单击【确定】按钮，在 ScoreWeb 工程中生成一个名字为“src”的源文件夹。

（3）右击“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（4）在【向导】树形结构中选择【其他】|【JSP】选项。单击【下一步】按钮，在【文件名】文本框中输入“scoreFrom.jsp”。

（5）单击【下一步】按钮，选择 JSP 模板“New JSP File（html）”，如图 6-17 所示。

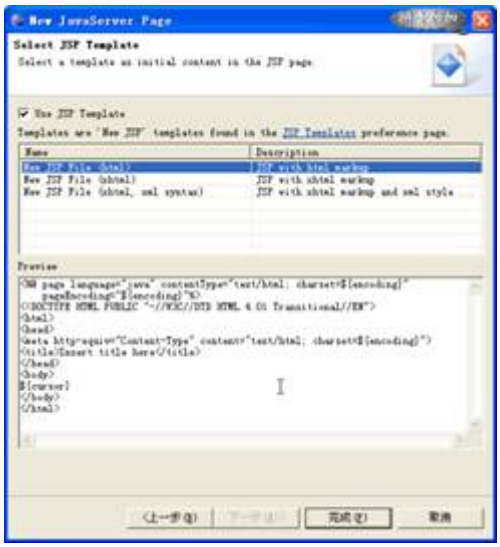

图 6-17　选择 JSP 模板

（6）单击【完成】按钮，在 ScoreWeb 工程中生成 scoreForm.jsp 文件。

（7）打开 scoreForm.jsp 文件，完成如下 JSP 代码：

```
<%@pagelanguage="java"contentType="text/html;charset=UTF-8"pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>学生成绩管理系统 Web 版</title>
  </head>
  <body>
    <!—以 POST 的方式提交学号、课号、成绩信息-->
    <form action="addScore.jsp" name="addScore" method="POST">
      <p>学号：<input type="text" name="SNO" value="0"></p>
      <p>课号：<input type="text" name="CNO" value="0"></p>
      <p>成绩：<input type="text" name="Score" value="0"></p>
      <p align="left"><input name="submit" type="submit" value="提交"></p>
    </form>
  </body>
</html>
```

由“%@...%”包含的信息是设定与 JSP 页面有关的信息，比如设定页面所使用的字符集是“UTF-8”。通过 form 的 action 属性指定表单提交后重定向到“addScore.jsp”页面。

（8）addScore.jsp 将收到的请求信息提交到 HSQLDB 数据库中。其 JSP 代码如下所示：

```
<%@pagelanguage="java"contentType="text/html;charset=ISO-8859-1"pageEncoding="ISO-8859-1"
%>
<%@page import="java.sql.*"%>
<%@page import="java.sql.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <%String SNO = "", CNO = "";
      float score;
   //取得 form 提交的 SNO、CNO 和 Score 参数的值
```

图 6.12 查询成绩

```
            SNO = request.getParameter("SNO");
            CNO = request.getParameter("CNO");
            score = Float.parseFloat(request.getParameter("Score"));
            try {
        //加载 HSQLDB 数据库 JDBC 驱动
        Class.forName("org.hsqldb.jdbcDriver");
        //连接 score 数据库，用户名为 sa，密码为空
        Connection connect = DriverManager.getConnection(
                    "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
    //产生 Statement 对象
        Statement statement = connect.createStatement();
    //将取得的参数值写入到关系数据库中
        String sql = "insert into score values(\'" + SNO + "\'," + "\'"+ CNO + "\'," + score + ")";
        //执行 insert 语句
      statement.execute(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        %>
  </head>
  <body>
  </body>
</html>
```

6.3.4 创建 scoreList.jsp 显示成绩列表

6.3.3 小节介绍了将提交的用户数据保存到数据库中，本小节介绍通过 JSP 页面查询数据库中的信息，

并将查询结果显示在浏览器页面中。在浏览器中访问 scoreList.jsp 页面，查询成绩的显示效果如图 6-18 所示。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，选择【新建】|【其他】命令，打开【新建】对话框，选择“JSP”。

（2）单击【下一步】按钮，在【文件名】文本框中输入“scoreList.jsp”。

（3）单击【下一步】按钮，选择 JSP 模板“New JSP File（html）”，如图 6-17 所示。

（4）单击【完成】按钮，scoreList.jsp 文件被建立。

（5）完成如下 JSP 代码：

```
<html>
  <title>查询结果</title>
  <body>
    <%@ page contentType="text/html;charset=gb2312"%>
    <%@ page import="java.sql.*"%>
    <%
     //装载 HSQL 数据库的驱动程序
     Class.forName("org.hsqldb.jdbcDriver");
    //根据 score 数据库的 URL、用户名和密码取得数据库的连接
       Connection con = DriverManager.getConnection(
                                "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
       //创建 Statement 对象
     Statement smt = con.createStatement();
       //查询所有记录的 SQL 语句
       String sql = "select * from score";
    //执行查询
       rs = smt.executeQuery(sql);
    //将查询结果发送到客户端
       out.println("查询结果如下:" + "<hr>");
       out.println("<table border='1'>");
       out.println("<tr bgcolor='gray'><th>学号</th><th>课程编号</th><th>成绩</th></tr>");
```

```
        while(rs.next()){
            out.println("<tr><td>" + rs.getString(1) + "</td><td>"
                                + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                                + "</td></tr>");
                    }
        con.close();
            %>
        </table>
    </body>
</html>
```

6.4 在 Eclipse 中开发 Servlet

Servlet 是运行在 Web 服务器或应用服务器上的 Java 程序，它是一个中间层，负责连接来自 HTTP 客户程序的请求和服务器上的数据库或应用程序。从某种程度上，可以将 Servlet 看作是有 HTML 的 Java 程序。本节介绍根据给定的学号查询该学生的所有课程编号以及该课程的分数的 Servlet 的实现方法和编程步骤。

6.4.1 创建 ScoreFindServlet 类查询成绩

在浏览器中输入“http://localhost:8080/scoreFind?SNO=12”，将“SNO=12”参数传递给服务器，服务器查询 HSQLDB 数据库将结果返回到客户端，其效果如图 6-19 所示。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【HTTP Servlet】命令。

（3）单击【下一步】按钮，进入【New HTTP Servlet】对话框。

（4）在【名称】文本框中输入“ScoreFindServlet”，选中【doGet()method】复选框，如图 6-20 所示。

图 6-19　scoreFind 的运行效果

图 6-20　【New HTTP Servlet】对话框

（5）单击【完成】按钮，生成“ScoreFindServlet.java”。

（6）在生成的 doGet 函数体中输入如下代码：

```
    //取得请求 URL 中 SNO 参数的值
    String SNO = req.getParameter("SNO");
  //加载 HSQLDB 数据库 JDBC 驱动程序
        Class.forName("org.hsqldb.jdbcDriver");
  //建立到数据库的连接
        Connection  con = DriverManager.getConnection(
                    "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
  //新建 Statement 对象
        Statement smt = con.createStatement();
        ResultSet rs =null;
  //查询 SQL 语句
        String sql = "select * from score where SNO=\'" + SNO + "\'";
  //执行数据库查询 SQL 语句
        rs = smt.executeQuery(sql);
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
```

```
            out.println("*****RESULT*****" + "<hr>");
            out.println("<table border='1'>");
            out.println("<tr bgcolor='gray'><th>SNO</th><th>CNO</th><th>GRADE</th></tr>
");
                while (rs.next()) {
                    out.println("<tr><td>" + rs.getString(1) + "</td><td>"
                            + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                            + "</td></tr>");
                }
        con.close();
```

从请求 URL 中取得 SNO 参数的值，根据取得的 SNO 参数值从数据库中查询符合条件的所有记录并以表格的形式在客户端显示。

6.4.2 创建 ScoreDeleteServlet 类删除成绩

本小节创建一个 Servlet，将某个学号的所有成绩都删除。

（1）在浏览器的地址栏中输入“http://localhost:8080/ScoreDelete?SNO=13”，如果数据库中不存在 SNO=13 的记录就会出现如图 6-21 所示的提示。

（2）在浏览器的地址栏中输入“http://localhost:8080/ScoreDelete?SNO=12”，如果数据库中存在 SNO=12 的记录就会出现如图 6-22 所示的返回状态。

图 6-21　记录不存在的返回状态

图 6-22　删除记录

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【HTTP Servlet】命令。

（3）单击【下一步】按钮，进入下一级对话框。在【名称】文本框中输入“ScoreDeleteServlet”，这里只选中【doGet(method) 】。

（4）单击【完成】按钮，生成“ScoreDeleteServlet.java”。

（5）在生成的 doGet 函数体中输入如下代码：

```
//从请求中取得 SNO 参数的值
String SNO = req.getParameter("SNO");
//装载 HSQLDB 数据库的驱动程序
Class.forName("org.hsqldb.jdbcDriver");
//建立到数据库的连接
Connection con = DriverManager.getConnection(
                                "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
  //生成 Statement 对象
  Statement smt = con.createStatement();
  ResultSet rs = null;
  //查询 SNO 字段等于参数值的 SQL 语句
  String sql = "select * from score where SNO=\'" + SNO + "\'";
  //执行 SQL 查询语句
  rs = smt.executeQuery(sql);
  //设定响应的内容类型
  resp.setContentType("text/html");
  PrintWriter out = resp.getWriter();
  //判断数据库中是否存在满足删除条件的记录
  if (!rs.next()) {
        out.println("RECORD:SNO=" + SNO + " does not exist!");
   } else {
        out.println("*****Following Records Deleted*****" + "<hr>");
        out.println("<table border='1'>");
        out.println("<tr bgcolor='gray'><th>SNO</th><th>CNO</th><th>GRADE</th></tr>
");
  //将即将删除的记录发送到客户端
     do {
          out.println("<tr><td>" + rs.getString(1) + "</td><td>"
```

```
                                        + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                                        + "</td></tr>");
                } while (rs.next());
    //删除 SNO 字段值等于给定值的记录
            sql = "delete from score where SNO=\'" + SNO + "\'";
     //删除记录
        smt.execute(sql);
      }
    //关闭连接
      con.close();
```

从 Request 请求中取得参数 SNO 的值，根据取得的 SNO 值从数据库中取得所有满足条件的记录。判断返回的结果集中记录的数目。如果数目为 0，表示要删除的记录不存在，否则显示所有被删除的记录。

6.5 在 Eclipse 中开发 Filter

Filter（过滤器）是 Servlet 2.3 规范中引入的新组件类型，可以截取发送至 Servlet、JSP 页面或静态页面的请求，也可以在响应发送至客户之前先行截获。这样就可以很容易地将应用于所有请求的任务或应用所提供的服务集中起来。过滤器可以全权访问请求及响应的体和首部，因此可以完成各种不同的转换。本节假定要限制网络地址处于“10.105.20”段的用户访问上一节实现的 ScoreFindServlet，就可以为该应用添加一个 Filter。

运行效果

在浏览器的地址栏中输入“http://10.105.20.117:8080/ScoreWeb/ScoreFind?SNO=1”，因为该 URL 处于“10.105.20”网段上，所以出现如图 6-23 所示的信息。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【Filter】命令。

（3）单击【下一步】按钮，进入【New Web Filter】对话框。

（4）在【名称】文本框中输入“ScoreFilter”，其他保持默认状态，如图 6-24 所示。

图 6-23　过滤器显示的信息

图 6-24　新建过滤器

（5）单击【完成】按钮，生成 ScoreFilter 类，该类实现了 Filter 接口。

（6）在生成的 doFilter 函数中完成如下代码：

```
    //取得客户端的 IP 地址
    String  remoteIP=request.getRemoteAddr();
        int  index=remoteIP.lastIndexOf(".");
  //取得客户端所处的网段
        String  subIP=remoteIP.substring(0,index);
  //判断是否在 10.105.20 网段上
        if(subIP.equals("10.105.20")){
            PrintWriter  out=response.getWriter();
            out.println("<html><head></head><body>");
            out.println("<h1>Sorry,you  can  not  access  our  application!</h1>");
            out.println("</body></html>");
            out.flush();
            return;
        }
  //将控制权交给下一个 Filter
        chain.doFilter(request,response);
```

%注意：每一个 Filter 从 doFilter()方法中得到当前的 request 及 response。在这个方法中，可以进行任何的针对 request 及 response 的操作(包括收集数据,包装数据等)。Filter 调用 chain.doFilter()方法

把控制权交给下一个 Filter。一个 Filter 在 doFilter()方法中结束。如果一个 Filter 想停止 request 处理而获得对 response 的完全控制，则它可以不调用下一个 Filter。

ScoreFilter 根据取得的客户端 IP 地址，判断客户端所处的网段，如果其在“10.105.20”网段上就会向客户端发送提示信息。

6.6 在 Eclipse 中开发 Listener

Listener 是 Servlet 的监听器，它可以监听客户端的请求、服务端的操作。通过监听器，可以自动激发一些操作，比如监听在线用户的数量。本节将实现一个会话事件 Listener，可以跟踪一个应用活动 Session 的个数。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框。

（2）在【名称】文本框中输入“ScoreSessionListener”，作为该 Listener 类的名字。

（3）单击【添加】按钮，打开【选择已实现的接口】对话框。

（4）在【选择接口】文本框中输入“HttpSessionListener”，使得该类实现“HttpSessionListener”接口。

（5）单击【确定】按钮，返回【新建 Java 类】对话框。

（6）其他保持默认状态，如图 6-25 所示。单击【完成】按钮，生成 ScoreSessionListener 类。

图 6-25　新建 Listener 类

（7）在生成的 ScoreSessionListener 类中完成如下代码：

```
public class ScoreSessionListener implements HttpSessionListener {
    //会话个数属性的名字
        private static final String SESSION_COUNTER = "session_counter";
    //将计数器递增，该计数器作为一个 Servlet 上下文属性得到维护
        public void sessionCreated(HttpSessionEvent se) {
                int[] num = getNumber(se);
                num[0]++;
        }
    private int[] getNumber(HttpSessionEvent se) {
                HttpSession session = se.getSession();
        //取得 Session 的上下文对象
                ServletContext context = session.getServletContext();
        //取得上下文属性，名称为“session_counter”
                int[] num = (int[]) context.getAttribute(SESSION_COUNTER);
                if (num == null) {
                        num = new int[1];
         //将 session_counter 属性保存到 Session 的上下文中
                        context.setAttribute(SESSION_COUNTER, num);
                }
                return num;
        }
  //将计算器递减
        public void sessionDestroyed(HttpSessionEvent se) {
                int[] num = getNumber(se);
                num[0]--;
        }
}
```

ScoreSessionListener 在 Servlet 的上下文属性中维护一个计数器，当新的 Session 产生时就将计数器的值加 1，反之当 Session 完成时就将计数器的值减 1，从而监听活动 Session 的个数。

6.7 配置 web.xml 文件

为了部署 Servlet 需要配置 web.xml 文件，根据这个配置文件，JBoss 才会知道如何访问 Servlet。本节介绍如何配置 web.xml 文件，为 6.9 节的部署 Web 应用做准备。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【文件夹】命令，打开【新建文件夹】对话框。

（2）在对话框底部的【文件夹名称】文本框中输入“WEB-INF”。

（3）单击【完成】按钮，在 ScoreWeb 工程的“src”文件夹中生成“WEB-INF”文件夹。

（4）右击“WEB-INF”文件夹，在快捷菜单中选择【新建】|【文件】命令，打开【新建文件】对话框。

（5）在对话框底部的【文件名称】文本框中输入“web.xml”。

（6）单击【完成】按钮，在“WEB-INF”文件夹中生成“web.xml”文件。

（7）编辑 web.xml 输入如下内容：

```
<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN' 'http://java.sun.com/j2ee/dtds/web-app_2.2.dtd'>
<web-app>
    <servlet>
     <!-------配置名字为 ScoreFindServlet 的 Servlet------>
        <servlet-name>ScoreFindServlet</servlet-name>
     <!------指定 ScoreFindServlet 的实现类------>
        <servlet-class>ScoreFindServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ScoreFindServlet</servlet-name>
```

```
    <!------指定 ScoreFindServlet 的 URL------>
        <url-pattern>/ScoreFind</url-pattern>
</servlet-mapping>
<servlet>
  <!-------配置名字为 ScoreDeleteServlet 的 Servlet------>
        <servlet-name>ScoreDeleteServlet</servlet-name>
  <!------指定 ScoreDeleteServlet 的实现类------>
        <servlet-class>ScoreDeleteServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>ScoreDeleteServlet</servlet-name>
    <!------指定 ScoreDeleteServlet 的 URL------>
        <url-pattern>/ScoreDelete</url-pattern>
</servlet-mapping>
<filter>
    <!------配置名字为 ScoreFilter 的 Filter------>
        <filter-name>ScoreFilter</filter-name>
    <!------指定 ScoreFilter 的实现类------>
        <filter-class>ScoreFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>ScoreFilter</filter-name>
    <!------指定 ScoreFilter 的 URL------>
        <url-pattern>/ScoreFind</url-pattern>
</filter-mapping>
<listener>
    <!------配置名字为 ScoreFilter 的 Filter------>
        <listener-class>ScoreSessionListener</listener-class>
</listener>
```

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

6.8 WAR 文件的打包生成

Web 应用通常都是以 WAR 文件的形式部署到 JBoss 应用服务器上。通过 JBossIDE 提供的“打包配置”功能可以定义 WAR 文件中所包含的内容。配置完成后运行该“打包配置”就可以生成所需要的 WAR 文件。本节首先介绍如何产生一个“打包配置”，然后介绍如何通过“打包配置”将本章的实例打包成一个 WAR 文件。

跟我做

（1）右击“ScoreWeb”工程，在快捷菜单中选择【属性】命令，打开【ScoreWeb 的属性】窗口。

（2）在窗口左侧的树形结构中选择【Packaging Configurations】选项，窗口右侧出现“打包配置”的具体信息。

（3）单击窗口右侧的【Add Standard…】按钮，打开【Choose Packaging configurations】对话框。

（4）选择 Standard-WAR.war，并且在【Name】文本框中输入“ScoreWeb.war”，如图 6-26 所示。

（5）单击【确定】按钮，生成名字为“ScoreWeb.war”的配置信息。

（6）展开“ScoreWeb.war”配置，选择【MANIFEST.MF】选项，单击【Remove】按钮将其删除，其最终状态如图 6-27 所示。

图 6-26 选择“打包配置”类型

图 6-27 打包配置

（7）单击【确定】按钮，在“ScoreWeb”工程中生成“package-build.xml”文件。

（8）右击“ScoreWeb”工程，在快捷菜单中选择【Run Packaging】命令。该命令执行后就在“ScoreWeb”工程中生成“ScoreWeb.war”文件。

6.9 部署 Web 应用

本节将产生的 WAR 发布到 JBoss 上。

跟我做

（1）单击工具栏上的【调试】按钮，打开【调试】对话框。

（2）在对话框左侧的“配置”树形结构中选择【JBoss 4.0.x】选项，单击【新建】按钮，对话框右侧出现“JBoss 4.0.x”的配置信息。

（3）在【名称】文本框中输入“JBoss4”，作为新建“JBoss 4.0.x”的名称，单击【Browse】按钮选择 JBoss 4 的安装目录，比如“C:\jboss-4.0.4”。在【Server Configuration】下拉列表框中选择【default】选项，如图 6-28 所示。

（4）单击【应用】按钮，配置完成。

（5）右击“ScoreWeb.war”文件，在快捷菜单中选择【Deployment】|【Deploy to】命令，打开【Target Choice】对话框。

（6）选择【JBoss4】选项，如图 6-29 所示。

图 6-28 新建 JBoss 4.0.x 配置

图 6-29 选择目标 JBoss 服务器

（7）单击【确定】按钮，“ScoreWeb.war”文件被部署到“C:\jboss-4.0.4\server\ default\deploy”目录下。

第 8 章 Hibernate 开发实例——图书管理系统

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲地使用对象编程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用，最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，完成数据持久化的重任。

本章以图书管理系统为例详细介绍了在 Eclipse 中开发 Hibernate 实例的具体步骤，内容包括数据库的配置，创建持久化对象，生成 Hibernate 映射文件以及生成配置文件等内容，在本章的最后通过一个图书管理系统详细介绍了通过 Hibernate 进行数据库操作的详细
步骤。

8.1 下载并安装 Hibernate Synchronizer 插件

Hibernate Synchronizer 是一个 Eclipse 插件，可以自动生成*.hbm 文件、持久化类和 DAO，大大降低开发 Hibernate 应用的难度。本节介绍如何下载和安装 Hibernate Synchronizer 插件。

JBoss Eclipse IDE 插件中包括 Hibernate Tools，按照 6.1 节介绍的步骤安装 JBoss Eclipse IDE 插件后即可完成 Hibernate Synchronizer 插件的安装，打开如图 8-1 所示的【关于 Eclipse SDK 插件】窗口，其中包含了 4 个 Hibernate 插件，说明插件安装成功。

图 8-1 【关于 Eclipse SDK 插件】窗口

8.2 图书管理系统需求分析

图书管理系统分为用户管理和图书管理两大部分，分别具有如下功能：

- 用户分为系统管理员、书籍管理员和借阅管理员 3 种角色，不同角色具有不同的权限。
- 用户登录和用户管理功能。
- 图书管理包括增加图书信息、删除图书信息和修改图书信息功能。
- 借书和还书管理，修改借书和还书记录信息。
- 查询所有书籍列表、书籍借阅情况和所有用户列表。

其运行界面如图 8-2 所示，整个系统分为系统管理、书籍管理、借书管理、还书管理和信息查询 5 大部分。

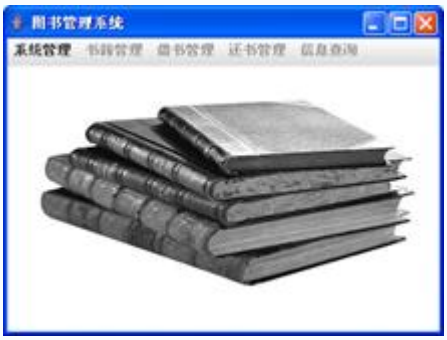

图 8-2　图书管理系统运行界面

系统管理完成对用户登录和用户权限的管理。用户权限分为“系统管理员”、“书籍管理员”和“借阅管理员”3 种。用户管理可以增加用户、修改用户信息和删除用户。

书籍管理完成对所有书籍信息的维护。分为“添加书籍”、“修改书籍”和“删除书籍”3 部分功能。借书管理完成对所有已出借图书信息的维护，分为“出借图书”和“修改出借图书信息”两部分功能。

8.3　配置数据库

本节在 SQL Server 2000 中创建图书管理系统的数据库，关于 SQL Server 数据库的相关知识可参见其相应文档。

跟我做

（1）打开 SQL Server 企业管理器，创建名称为“demo”的数据库，如图 8-3 所示。

图 8-3　新建 demo 数据库

（2）打开 SQL Server 的 SQL 查询分析器，选择默认数据库为刚才创建的“demo”数据库，输入如下 SQL 脚本：

```
//创建 books 表，保存所有有关书籍的信息
create table books (
BookName varchar(20),
Press varchar(20),
Author varchar(20),
address varchar(50),
PressDate datetime,
Price float,
Com varchar(20),
books_count int,
borrowed_count int,
constraint ID_Contraint_PK primary key ( BookName ));
//创建表 bookBrowse，保存所有书籍借阅情况信息
create table bookBrowse (
StudentName varchar(40),
BookName varchar(40),
ReturnDate datetime,
BorrowDate datetime,
Com varchar(40),
Is_Returned char(2),
constraint ID_BookBrowse_Containt primary key ( StudentName ));
```

```
//创建表 UserTable，保存所有的用户信息
create table UserTable(
UserName varchar(40),
Password varchar(40),
Power varchar(40),
constraint ID_User_Containt primary key ( UserName ));
```

（3）单击 按钮，执行上述 SQL 脚本，生成 3 个表：books 表、bookBrowse 表和 UserTable 表，分别保存书籍信息、书籍出借信息和用户信息，如图 8-4～图 8-6 所示。

%注意：为了简单起见，books 表和 UserTable 表分别以书籍名和用户名作为主键，所以不允许出现重名的书籍和用户。

列名	数据类型	长度	允许空
BookName	varchar	20	
Press	varchar	20	✓
Author	varchar	20	✓
address	varchar	50	✓
PressDate	datetime	8	✓
Price	float	8	✓
Com	varchar	20	✓
books_count	int	4	✓
borrowed_count	int	4	✓

图 8-4　表 books

列名	数据类型	长度	允许空
StudentName	varchar	40	
BookName	varchar	40	✓
ReturnDate	datetime	8	✓
BorrowDate	datetime	8	✓
Com	varchar	40	✓
Is_Returned	char	2	✓

图 8-5　表 bookBrowse

列名	数据类型	长度	允许空
UserName	varchar	40	
Password	varchar	40	✓
Power	varchar	40	✓

图 8-6　表 UserTable

books 表记录书籍的书名、出版社、作者、地址、出版日期、价格、书籍数量和被借阅数量；bookBrowse 表记录学生姓名、书籍书名、借阅日期、应还日期和是否归还；UserTable 表中记录用户的姓名、密码和角色。

传统的对关系数据库表的访问都是通过 SQL 语句进行的，本章利用 Hibernate 框架来封装对关系数据库的访问，使得可以完全以面向对象的方式对上述表格进行读写操作，从而提高数据库开发效率。

8.4　生成配置文件 hibernate.cfg.xml

Hibernate 运行时需要获取一些底层实现的基本信息，包括数据库 URL、数据库用户、数据库用户密码、数据库 JDBC 驱动类和数据库 dialect 等。Hibernate 同时支持 xml 格式的配置文件，以及传统的 properties 文件配置方式。本章采用基于 xml 格式文件的配置方式，这些信息都包含在默认名称为 hibernate.cfg.xml 的文件中。本节介绍如何在 Eclipse 中快速生成 hibernate.cfg.xml 文件。

跟我做

（1）创建名称为“Library”的 Java 工程。单击【文件】菜单，选择【新建】|【其他】命令，打开如图 8-7 所示的【新建】对话框。

（2）选择【Hibernate Configuration File】选项，单击【下一步】按钮，在图 8-8 的【输入或选择父文件夹】文本框中选择“Library”工程，单击【下一步】按钮，打开数据库配置对话框。

图 8-7 【新建】对话框

图 8-8 选择工程名称

（3）在数据源配置对话框中输入如下数据库配置信息，如图 8-9 所示。

- Database dialect：SQLServer
- Driver class：com.microsoft.jdbc.sqlserver.SQLServerDriver
- Connection URL：jdbc:microsoft:sqlserver://localhost:1433;databaseName=demo
- Username：sa（根据实际配置）

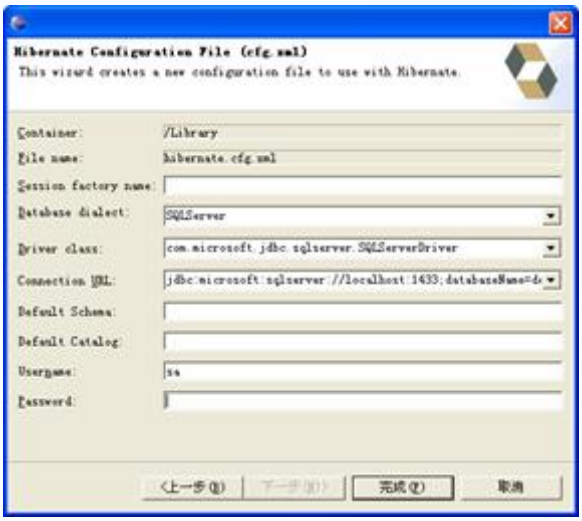

图 8-9 数据库配置窗对话框

（4）单击【完成】按钮，在 Library 工程的根目录下生成 hibernate.cfg.xml 文件，其内容如下所示：

```
<?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 >
    <!--数据库 JDBC 驱动类-->
      <property name="hibernate.connection.driver_class">com.microsoft.jdbc.sqlserver.SQLServerDri
ver</property>
    <!--数据库密码-->
      <property name="hibernate.connection.password"></property>
    <!--数据库的 URL-->
      <property name="hibernate.connection.url">jdbc:microsoft:sqlserver://localhost:1433;database
Name=demo</property>
    <!--数据库的用户名-->
      <property name="hibernate.connection.username">sa</property>
    <!--每个数据库都有其对应的 Dialet 以匹配其平台特性-->
      <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
   </session-factory>
</hibernate-configuration>
```

hibernate.cfg.xml 文件可以包含构建 SessionFactory 实例的所有配置信息。当使用如下代码

```
SessionFactory sessions=new Configuration().configure().buildSessionFactory();
```

初始化 Hibernate 时，Hibernate 会在 classpath 中寻找文件名为 hibernate.cfg.xml 的文件。

%注意：如果运行时出现如图 8-10 所示的错误信息，这是因为在配置文件中设置了 session-factory 的 name 属性，这样 hibernate 会试图把这个 sessionfactory 注册到 jndi 中去，从而报告错误。去掉 name 属性即可，所以在 hibernate.cfg.xml 中不要设置 session-factory 的 name 属性。

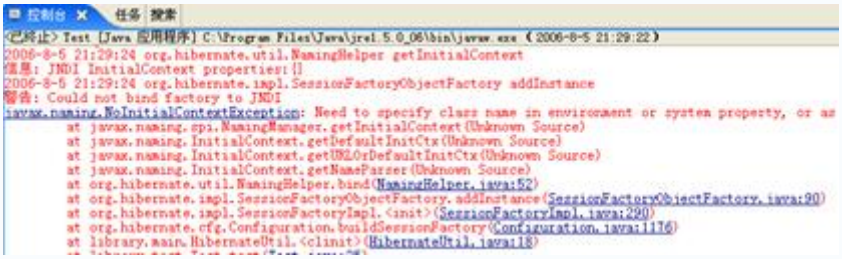

图 8-10　配置 session-factory name 属性后的出错信息

8.5　创建持久化对象

Hibernate 从本质上来讲是一种“对象－关系型数据映射”（Object Relational Mapping 简称 ORM）。POJO 在这里体现的就是 ORM 中 Object 层的语义，而映射（Mapping）文件则是将对象（Object）与关系型数据（Relational）相关联的纽带，在 Hibernate 中，映射文件通常以“.hbm.xml”作为后缀。

8.5.1　生成映射文件和持久化对象

本小节介绍如何在 Eclipse 中根据数据库中的表结构生成映射文件和持久化对象。通过 Hibernate Synchronizer 插件可以方便地生成映射文件和持久化对象，方便 Hibernate 应用的开发。

跟我做

（1）单击 Eclipse 的【窗口】菜单，选择【打开透视图】|【其他】命令，打开【选择透视图】对话框，如图 8-11 所示。选择【Hibernate Console】选项，单击【确定】按钮，打开 Hibernate Console 透视图。

Hibernate Console 透视图包括 Hibernate Configuration、Hibernate Dynamic Query Translator、Hibernate Entity Model 和 Hibernate Query Result 等几个视图。

（2）右击 Hibernate Configuration 视图的空白区域，在快捷菜单中选择【Add configuration】命令，打开【Create Hibernate Console Configuration】对话框，如图 8-12 所示。

（3）单击【Configuration file】文本框右侧的Browse...按钮并选择“\Library\hibernate. cfg.xml”，单击“Classpath”组中的Add JAR/Dir...按钮将 SQL Server 数据库的 JDBC 驱动类 msbase.jar、mssqlserver.jar 等加入到 classpath 中；在【Name】文本框中输入“Library”，单击【完成】按钮，创建名称为“Library”的配置，如图 8-13 所示。

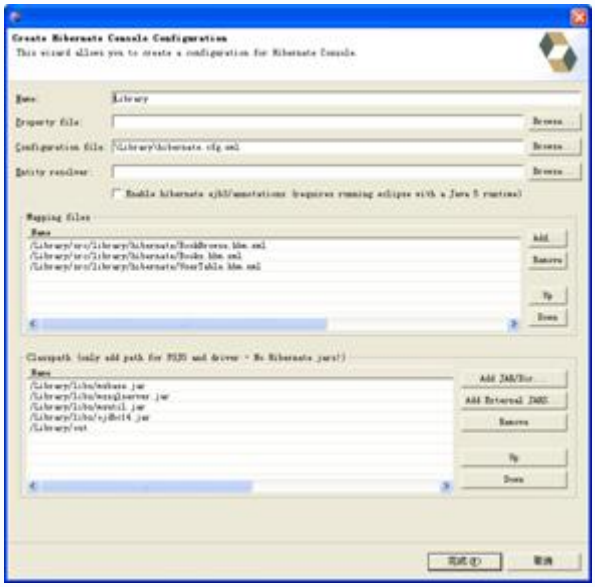

图 8-12　生成 hibernate Console Configuration

图 8-13　Hibernate Configuration 视图

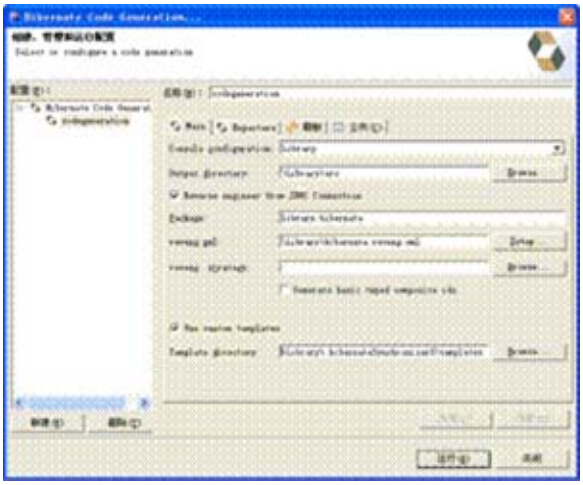

图 8-14 【Hibernate Code Generation】对话框

（4）单击工具栏中的按钮右边的下拉箭头，在下拉菜单中选择【Hibernate Code Generation】命令，打开如图 8-14 所示的【Hibernate Code Generation】对话框。

（5）在【Hibernate Code Generation】对话框中的【Main】选项卡中输入如下配置信息：

- 名称：codegeneration
- Console configuration：Library
- Output directory：\Library\src
- Package：library.hibernate

创建名称为“codegeneration”的 Hibernate Code Generation 配置，生成的目标代码存放在“\library\src”目录下，其包名为“library.hibernate”。

（6）单击【Exporters】选项卡选中如下各项，如图 8-15 所示。

- Generate domain code(.java)
- JDK1.5 Constructs(generics,etc.)
- Generate DAO code(.java)
- Generate mappings(hbm.xml)

（7）单击【运行】按钮，在 Library 工程的“library.hibernate”包中生成如图 8-16 所示的文件。

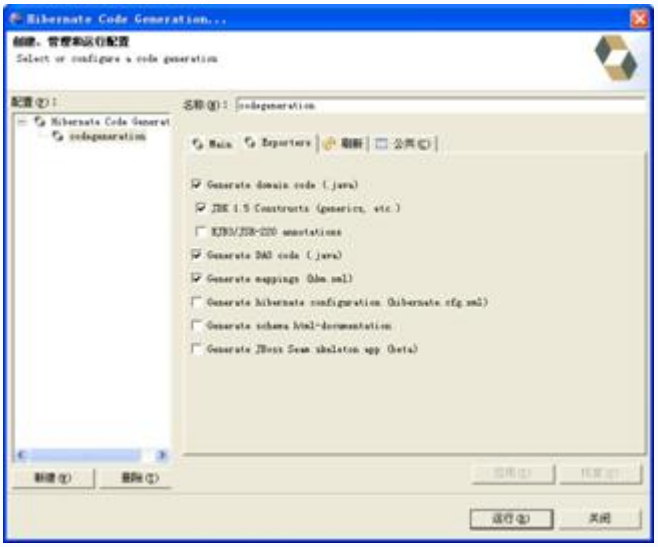

图 8-15 Exporters 配置

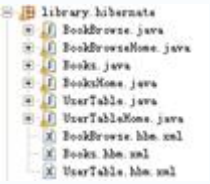

图 8-16 生成的映射文件和持久化对象

8.5.2 对持久化对象的分析

BookBrowse.java、Books.java 和 UserTable.java 3 个生成的类是标准的 JavaBean，Hibernate 插件根据数据库的表结构自动生成了 3 个持久化对象，对于每个属性都有其对应的 getter/setter 方法。

%注意：这 3 个类中的英文注释是 Hibernate 插件自动生成的，为了便于读者的理解，这里加上了部分中文注释。

（1）BookBrowse.java 文件是标准的 JavaBean 对应于 demo 数据库中的 bookbrowse 表，包括学生姓名、书名、归还日期、借阅日期、备注和是否归还等信息。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3
import java.util.Date;
/**
 * BookBrowse generated by hbm2java
 */
public class BookBrowse  implements java.io.Serializable {
    // Fields
     //学生名字
     private String studentName;
     //书名
     private String bookName;
     //归还日期
     private Date returnDate;
     //借阅日期
     private Date borrowDate;
     //备注，评论信息
     private String com;
     //是否归还
     private String isReturned;
    // Constructors
    /** 默认构造函数 */
    public BookBrowse() {
    }
        /** minimal constructor */
    public BookBrowse(String studentName) {
       this.studentName = studentName;
```

```
    }

    /** full constructor */
    public BookBrowse(String studentName, String bookName, Date returnDate, Date borrowDate, S
tring com, String isReturned) {
        this.studentName = studentName;
        this.bookName = bookName;
        this.returnDate = returnDate;
        this.borrowDate = borrowDate;
        this.com = com;
        this.isReturned = isReturned;
    }

    // Property accessors 取得学生姓名
    public String getStudentName() {
        return this.studentName;
    }
    //设置学生姓名
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    //取得书名字段的值
    public String getBookName() {
        return this.bookName;
    }
    //设置书名字段
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
```

```
//取得归还日期字段
public Date getReturnDate() {
    return this.returnDate;
}
//设置归还日期字段
public void setReturnDate(Date returnDate) {
    this.returnDate = returnDate;
}
//取得借阅日期
public Date getBorrowDate() {
    return this.borrowDate;
}
//设置借阅日期
public void setBorrowDate(Date borrowDate) {
    this.borrowDate = borrowDate;
}
//取得备注信息
public String getCom() {
    return this.com;
}
//设置备注信息
public void setCom(String com) {
    this.com = com;
}
//取得是否归还状态
public String getIsReturned() {
    return this.isReturned;
}
//设置是否归还状态
```

```
    public void setIsReturned(String isReturned) {
        this.isReturned = isReturned;
    }
}
```

（2）Books.java 类对应于 demo 数据库中的 books 表，是标准的 JavaBean，包括书名、出版社、作者、地址、出版日期和价格等相关信息。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3
import java.util.Date;
/**
 * Books generated by hbm2java
 */
public class Books  implements java.io.Serializable {
    // Fields
     //书名
     private String bookName;
     //出版社
     private String press;
     //作者
     private String author;
     //地址
     private String address;
     //出版日期
     private Date pressDate;
     //价格
     private Double price;
     //备注
     private String com;
     //图书数量
```

```
    private Integer booksCount;
    //已借阅数量
    private Integer borrowedCount;
    // Constructors
    /** default constructor */
    public Books() {
    }
        /** minimal constructor */
    public Books(String bookName) {
        this.bookName = bookName;
    }

    /** full constructor */
    public Books(String bookName, String press, String author, String address, Date pressDate, Do
uble price, String com, Integer booksCount, Integer borrowedCount) {
        this.bookName = bookName;
        this.press = press;
        this.author = author;
        this.address = address;
        this.pressDate = pressDate;
        this.price = price;
        this.com = com;
        this.booksCount = booksCount;
        this.borrowedCount = borrowedCount;
    }


    // Property accessors
    //取得书名属性的值
```

```
public String getBookName() {
    return this.bookName;
}
//设置书名属性
public void setBookName(String bookName) {
    this.bookName = bookName;
}
//取得出版社字段的值
public String getPress() {
    return this.press;
}
//设置出版社字段
public void setPress(String press) {
    this.press = press;
}
//取得作者信息
public String getAuthor() {
    return this.author;
}
//设置作者信息
public void setAuthor(String author) {
    this.author = author;
}
//取得地址信息
public String getAddress() {
    return this.address;
}
//设置地址信息
public void setAddress(String address) {
```

```
    this.address = address;
}
//取得出版日期
public Date getPressDate() {
    return this.pressDate;
}
//设置出版日期
public void setPressDate(Date pressDate) {
    this.pressDate = pressDate;
}
//取得价格信息
public Double getPrice() {
    return this.price;
}
//设置价格信息
public void setPrice(Double price) {
    this.price = price;
}
//取得备注信息
public String getCom() {
    return this.com;
}
//设置备注信息
public void setCom(String com) {
    this.com = com;
}
//取得图书数量
public Integer getBooksCount() {
    return this.booksCount;
```

```
    }
    //设置图书数量
    public void setBooksCount(Integer booksCount) {
        this.booksCount = booksCount;
    }
    //取得已借阅图书数量
    public Integer getBorrowedCount() {
        return this.borrowedCount;
    }
    //设置已借阅图书数量
    public void setBorrowedCount(Integer borrowedCount) {
        this.borrowedCount = borrowedCount;
    }
}
```

（3）UserTable.java 类对应于 demo 数据库中的 usertable 表，包括用户名、密码和用户权限 3 个属性，是标准的 JavaBean。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3
/**
 * UserTable generated by hbm2java
 */
public class UserTable  implements java.io.Serializable {
    // Fields
     //用户名
     private String userName;
     //密码
     private String password;
     //用户权限
```

```
 private String power;
// Constructors
/** default constructor */
public UserTable() {
}
    /** minimal constructor */
public UserTable(String userName) {
   this.userName = userName;
}

/** full constructor */
public UserTable(String userName, String password, String power) {
   this.userName = userName;
   this.password = password;
   this.power = power;
}



// Property accessors
//取得用户名字段
public String getUserName() {
   return this.userName;
}
//设置用户名字段
public void setUserName(String userName) {
   this.userName = userName;
}
//取得密码字段
public String getPassword() {
```

```
        return this.password;
    }
    //设置密码字段
    public void setPassword(String password) {
        this.password = password;
    }
    //取得用户权限
    public String getPower() {
        return this.power;
    }
    //设置用户权限
    public void setPower(String power) {
        this.power = power;
    }

}
```

8.6 创建映射文件

Hibernate 之所以能够判断实体类和数据表之间的对应关系，是因为有 XML 映射文件。通过 Hibernate Code Generation 既可以生成持久化对象也可以生成映射文件。本节介绍在 8.5 节中生成的 BookBrowse.hbm.xml、Books.hbm.xml、UserTable.hbm.xml 3 个映射文件。

BookBrowse.hbm.xml 描述了 BookBrowse 类和 bookbrowse 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
    <!--library.hibernate.BookBrowse 类和 bookbrowse 表对应关系描述-->
```

```
    <classname="library.hibernate.BookBrowse"table="bookBrowse"schema="dbo"catalog=
"demo">
      <!--studentName 属性对应 StudentName 字段-->
        <id name="studentName" type="string">
          <column name="StudentName" length="40" />
          <generator class="assigned" />
      </id>
        <!--bookName 属性对应 BookName 字段-->
      <property name="bookName" type="string">
          <column name="BookName" length="40" />
      </property>
        <!--returnDate 属性对应 ReturnDate 字段-->
      <property name="returnDate" type="timestamp">
          <column name="ReturnDate" length="23" />
      </property>
       <!-- borrowDate 属性对应 BorrowDate 字段-->
      <property name="borrowDate" type="timestamp">
          <column name="BorrowDate" length="23" />
      </property>
          <!-- com 属性对应 Com 字段-->
      <property name="com" type="string">
          <column name="Com" length="40" />
      </property>
          <!-- isReturned 属性对应 Is_Returned 字段-->
      <property name="isReturned" type="string">
          <column name="Is_Returned" length="2" />
      </property>
   </class>
</hibernate-mapping>
```

Books.hbm.xml 描述了 Books 类和 books 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
    <!--library.hibernate.Books 类和 books 表对应关系描述-->
    <class name="library.hibernate.Books" table="books" schema="dbo" catalog="demo">
        <!-- bookName 属性对应 BookName 字段-->
      <id name="bookName" type="string">
        <column name="BookName" length="20" />
        <generator class="assigned" />
      </id>
       <!-- press 属性对应 Press 字段-->
       <property name="press" type="string">
        <column name="Press" length="20" />
      </property>
      <!-- author 属性对应 Author 字段-->
      <property name="author" type="string">
        <column name="Author" length="20" />
      </property>
      <!-- address 属性对应 address 字段-->
      <property name="address" type="string">
        <column name="address" length="50" />
      </property>
      <!-- pressDate 属性对应 PressDate 字段-->
      <property name="pressDate" type="timestamp">
        <column name="PressDate" length="23" />
      </property>
```

```
        <!-- price 属性对应 Price 字段-->
        <property name="price" type="java.lang.Double">
            <column name="Price" precision="53" scale="0" />
        </property>
        <!-- com 属性对应 Com 字段-->
        <property name="com" type="string">
            <column name="Com" length="20" />
        </property>
        <!-- booksCount 属性对应 books_count 字段-->
        <property name="booksCount" type="java.lang.Integer">
            <column name="books_count" />
        </property>
        <!-- borrowedCount 属性对应 borrowed_count 字段-->
        <property name="borrowedCount" type="java.lang.Integer">
            <column name="borrowed_count" />
        </property>
    </class>
</hibernate-mapping>
```

UserTable.hbm.xml 描述了 UserTable 类和 usertable 表之间的对应关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
    <!--library.hibernate.UserTable 类和 usertable 表对应关系描述-->
    <class name="library.hibernate.UserTable" table="UserTable" schema="dbo"catalog="demo">
        <!-- userName 属性对应 UserName 字段-->
```

```
        <id name="userName" type="string">
            <column name="UserName" length="40" />
            <generator class="assigned" />
        </id>
        <!-- password 属性对应 Password 字段-->
        <property name="password" type="string">
            <column name="Password" length="40" />
        </property>
        <!-- power 属性对应 Power 字段-->
        <property name="power" type="string">
            <column name="Power" length="40" />
        </property>
    </class>
</hibernate-mapping>
```

</class>定义实体类和数据表之间的关系，name 属性 library.hibernate.UserTable 是实体类（用类全名）的名字，UserTable 是对应的数据表。</id>定义了主键 id 字段所用的键值生成方法。</property>定义了实体类和表字段的关联，name 属性定义了类的字段，column 属性定义了数据库表字段名。Hibernate 通过这里的设置建立起实体类和数据库表之间的对应关系。

8.7 Hibernate 操作数据库的方法

通过 Hibernate 可以简化对数据库的操作，本节首先创建一个 HibernateUtil 类，用于管理 session，然后介绍如何通过 Hibernate 实现数据库的查询、插入、删除和更新操作。

SessionFactory 用来创建 Session 实例，通过 Configuration 实例构建 SessionFactory。Configuration 实例根据当前的配置信息，构造 SessionFactory 实例并返回。一旦 SessionFactory 构造完毕，即被赋予特定的配置信息。

Session 是持久层操作的基础，相当于 JDBC 的 Connection。通过 SessionFactory 实例构建。Session 实例提供的 saveOrUpdate、delete 和 createQuery 方法分别实现了数据库的插入更新、删除和查询操作，简化了数据库的基本操作。

跟我做

（1）在“Library”工程的“src”文件夹中创建“library.main”包，在“library.main”包中创建 HibernateUtil.java 文件，并输入如下内容：

```
package library.main;
import java.io.File;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class HibernateUtil {
	private static final SessionFactory sessionFactory;
	static {
		try {
	    //hibernate.cfg.xml 文件
			File file = new File(
					"E:\\Eclipsebook\\eclipse\\workspace\\Library\\src\\hibernate.cfg.xml
");
			//根据 hibernate.cfg.xml 中的配置信息创建 SessionFactory
			sessionFactory = new Configuration().configure(file)
					.buildSessionFactory();
		} catch (Throwable ex) {
			//创建 SessionFactory 失败信息
			System.err.println("Initial SessionFactory creation failed." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}
    //得到 SessionFactory 的静态方法
	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
```

```
}
```

（2）在“Library”工程的“src”文件夹中创建“library.test”包，创建 Test.java 文件，在该 Test.java 文件中测试数据库的插入、更新、删除和查询操作。

（3）插入和更新数据库的基本操作。

```
    //取得 SessionFactory 实例
    SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
    //打开一个 Session 实例
        Session session = sessionFactory.openSession();
  //开始事务
        Transaction tx = session.beginTransaction();
        //创建 UserTable 类实例
         UserTable userTable=new UserTable();
  //设置 userName 属性
         userTable.setUserName("张三");
  //设置 password 属性
         userTable.setPassword("123456");
  //设置 power 属性
         userTable.setPower("图书管理员");
  //插入和更新数据库
         session.saveOrUpdate(userTable);
  //提交事务
        tx.commit();
  //关闭会话
        session.close();
```

（4）从数据库中删除记录的基本操作。

```
    //取得 SessionFactory 实例
    SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
    //打开一个 Session 实例
```

```
        Session session = sessionFactory.openSession();
   //开始事务
        Transaction tx = session.beginTransaction();
  //创建 UserTable 类实例
        UserTable userTable=new UserTable();
  //设置 userName 属性
        userTable.setUserName("张三");
  //设置 password 属性
        userTable.setPassword("123456");
  //设置 power 属性
        userTable.setPower("图书管理员");
        // 删除操作
        // session.delete(userTable);
  //提交事务
        tx.commit();
        //关闭会话
    session.close();
```

（5）查询数据库的基本操作。

```
    //取得 SessionFactory 实例
     SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
    //打开一个 Session 实例
        Session session = sessionFactory.openSession();
    //开始事务
        Transaction tx = session.beginTransaction();
        // 查询 hql 语句
        String hql = "from UserTable where UserName='张三' and Password='123456'";
  //执行查询，查询结果为 Query 实例
        Query userList = session.createQuery(hql);
```

```
        //将查询结果放到一个 list 中
                List list = userList.list();
        //提交事务
                tx.commit();
        //关闭 session
                session.close();
```

8.8 系统主界面

系统主界面是整个系统的入口，所有系统功能都是通过主界面来实现的。主界面的运行效果图如图 8-2 所示，本节实现该主界面。

8.8.1 主界面窗体的创建

本小节将利用 Swing 中的图形控件生成主界面的窗体、菜单等内容，并且为每个菜单项添加了事件监听者。关于 Swing 的相关知识超出了本书介绍的范围，读者可以查看相关资料，本小节假设读者对 Swing 有一定的了解。

跟我做

（1）在“Library”工程的“src”文件夹中创建“library.main”包，创建 LibraryWindow.java 文件，继承 Jframe 类，并实现 ActionListener 接口。该文件是整个图书管理系统的入口，完成界面菜单的生成和菜单动作的响应等功能。

（2）编辑 LibraryWindow.java 文件，在其构造函数中输入如下代码：

```
        public LibraryWindow() {
                super("图书管理系统");
                // --系统管理菜单--
                menuBar = new JMenuBar();
                systemMenu = new JMenu("系统管理");
                // --用户管理菜单--
          userMGRMenu = new JMenu("用户管理");
```

```
  // --用户登录菜单--
        userLoginMenuItem = new JMenuItem("用户登录");
// --添加用户菜单--
        userAddMenuItem = new JMenuItem("添加用户");
  // --修改用户菜单--
        userModifyMenuItem = new JMenuItem("修改用户");
//删除用户菜单
        userDeleteMenuItem = new JMenuItem("删除用户");
//退出菜单
        exitMenuItem = new JMenuItem("退出");
// --将“用户登录”子菜单项加入到“系统菜单”中--
        systemMenu.add(userLoginMenuItem);
// --将“添加用户”子菜单项加入到“用户管理”菜单中--
        userMGRMenu.add(userAddMenuItem);
// --将“修改用户”子菜单项加入到“用户管理”菜单中--
        userMGRMenu.add(userModifyMenuItem);
 // --将“删除用户”子菜单项加入到“用户管理”菜单中--
        userMGRMenu.add(userDeleteMenuItem);
 // --将“用户管理”子菜单项加入到“系统管理”菜单中--
        systemMenu.add(userMGRMenu);
 // --将“退出”子菜单项加入到“系统管理”菜单中--
        systemMenu.add(exitMenuItem);
 // --为“用户登录”菜单项添加动作监听者--
        userLoginMenuItem.addActionListener(this);
 // --为“添加用户”菜单项添加动作监听者--
        userAddMenuItem.addActionListener(this);
 // --为“修改用户”菜单项添加动作监听者--
        userModifyMenuItem.addActionListener(this);
 // --为“删除用户”菜单项添加动作监听者--
```

```
        userDeleteMenuItem.addActionListener(this);
 // --为“退出”菜单项添加动作监听者--
        exitMenuItem.addActionListener(this);
 // --将“系统管理”菜单项加入到菜单栏上--
        menuBar.add(systemMenu);
        // ---书籍管理菜单--
        bookMGRMenu = new JMenu("书籍管理");
 // --“添加书籍”菜单--
        bookAddMenuItem = new JMenuItem("添加书籍");
// --“修改书籍”菜单--
        bookModifyMenuItem = new JMenuItem("修改书籍");
// --“删除书籍”菜单--
        bookDeleteMenuItem = new JMenuItem("删除书籍");
// --将“添加书籍”菜单加入到“书籍管理”菜单项中--
        bookMGRMenu.add(bookAddMenuItem);
// --将“修改书籍”菜单加入到“书籍管理”菜单项中--
        bookMGRMenu.add(bookModifyMenuItem);
// --将“删除书籍”菜单加入到“书籍管理”菜单项中--
        bookMGRMenu.add(bookDeleteMenuItem);
        // --将“添加书籍”菜单添加事件监听者--
   bookAddMenuItem.addActionListener(this);
   // --将“修改书籍”菜单添加事件监听者--
        bookModifyMenuItem.addActionListener(this);
// --将“删除书籍”菜单添加事件监听者--
        bookDeleteMenuItem.addActionListener(this);
//将“书籍管理”菜单添加到菜单栏中
        menuBar.add(bookMGRMenu);
        // --借书管理菜单--
        borrowBookMenu = new JMenu("借书管理");
```

```
//“书籍出借”菜单
        borrowBookMenuItem = new JMenuItem("书籍出借");
//“出借信息修改“菜单
        borrowInfoMenuItem = new JMenuItem("出借信息修改");
//将“书籍出借”菜单加入到“书籍管理”菜单中
        borrowBookMenu.add(borrowBookMenuItem);
//将“出借信息修改”菜单加入到“书籍管理”菜单中
        borrowBookMenu.add(borrowInfoMenuItem);
//为“书籍出借”菜单添加事件监听者
        borrowBookMenuItem.addActionListener(this);
//为“出借信息修改”菜单添加事件监听者
        borrowInfoMenuItem.addActionListener(this);
//将“借书管理”菜单加入到菜单栏中
        menuBar.add(borrowBookMenu);
        // --还书管理菜单--
        returnBookMenu = new JMenu("还书管理");
//“书籍还入”菜单
        returnBookMenuItem = new JMenuItem("书籍还入");
//“书籍还入信息修改”菜单
        returnInfoMenuItem = new JMenuItem("书籍还入信息修改");
//将“书籍还入”菜单加入到“还书管理”菜单中
        returnBookMenu.add(returnBookMenuItem);
//将“书籍还入信息修改”菜单加入到“还书管理”菜单中
        returnBookMenu.add(returnInfoMenuItem);
//为“书籍还入”菜单添加事件监听者
        returnBookMenuItem.addActionListener(this);
//为“书籍还入信息修改”菜单添加事件监听者
        returnInfoMenuItem.addActionListener(this);
//将“还书管理”菜单加入到菜单栏中
```

```
        menuBar.add(returnBookMenu);
        // --信息一览菜单--
        infoBrowseMenu = new JMenu("信息查询");
//"书籍列表"菜单
        bookListMenuItem = new JMenuItem("书籍列表");
//"借阅情况表"菜单
        borrowBookListMenuItem = new JMenuItem("借阅情况表");
//"用户列表"菜单
        userListMenuItem = new JMenuItem("用户列表");
//将"书籍列表"菜单添加到"信息一览"菜单中
        infoBrowseMenu.add(bookListMenuItem);
        //将"出借书籍列表"菜单添加到"信息一览"菜单中
  infoBrowseMenu.add(borrowBookListMenuItem);
  //将"用户列表"菜单添加到"信息一览"菜单中
        infoBrowseMenu.add(userListMenuItem);
//为"书籍列表"菜单添加事件监听者
        bookListMenuItem.addActionListener(this);
//为"出借图书列表"菜单添加事件监听者
        borrowBookListMenuItem.addActionListener(this);
//为"用户列表"菜单添加事件监听者
        userListMenuItem.addActionListener(this);
//将"信息一览"菜单加入到菜单栏中
        menuBar.add(infoBrowseMenu);
        //为 Window 添加菜单栏
        setJMenuBar(menuBar);
//图片 Label
        titleLabel = new JLabel(new ImageIcon(".\\pic.jpg"));
        container = getContentPane();
        container.setLayout(new BorderLayout());
```

```
            panel1 = new JPanel();
            panel1.setLayout(new BorderLayout());
            panel1.add(titleLabel, BorderLayout.CENTER);
            container.add(panel1, BorderLayout.CENTER);
            setBounds(100, 50, 400, 300);
            show();
            // --设置初始功能:--
            userMGRMenu.setEnabled(false);
            bookMGRMenu.setEnabled(false);
            borrowBookMenu.setEnabled(false);
            returnBookMenu.setEnabled(false);
            infoBrowseMenu.setEnabled(false);
    }
```

生成了系统管理、书籍管理、借书管理、还书管理和信息一览 5 个菜单项，并且为其子菜单项添加了事件监听者。

8.8.2 为每个菜单项添加响应事件

本小节将在 8.8.1 节的基础上，为主界面的每个菜单项添加事件响应函数，并且控制每个新创建窗口的显示位置。在 actionPerformed（ActionEvent）方法中通过参数 ActionEvent 对象的 getActionCommand() 方法可以得到用户点击的菜单项，然后进行不同的处理。

跟我做

（1）右击 LibraryWindow 类的任意空白区域，在快捷菜单中选择【源代码】|【覆盖/实现方法】命令，打开【覆盖/实现方法】对话框，选择“actionPerformed（ActionEvent）”，单击【确定】按钮，生成 actionPerformed 方法。

（2）编辑 LibraryWindow.java 文件，在 actionPerformed 方法中输入如下代码：

```
    // --设置每个菜单点击后出现的窗口和窗口显示的位置--
    public void actionPerformed(ActionEvent e) {
```

```
//"用户登录"菜单的响应函数
        if (e.getActionCommand() == "用户登录") {
    //创建用户登录窗口
                UserLogin userLoginFrame = new UserLogin(this);
                Dimension frameSize = userLoginFrame.getPreferredSize();
                Dimension mainFrameSize = getSize();
                Point loc = getLocation();
    //设置窗口的位置和大小
                userLoginFrame.setLocation((mainFrameSize.width - frameSize.width)
                        / 2 + loc.x, (mainFrameSize.height - frameSize.height) / 2
                        + loc.y);
                userLoginFrame.pack();
                userLoginFrame.show();
            } else if (e.getActionCommand() == "添加用户") {
    //"添加用户"菜单的响应函数
                UserAdd UserAddFrame = new UserAdd();
                Dimension FrameSize = UserAddFrame.getPreferredSize();
                Dimension MainFrameSize = getSize();
                Point loc = getLocation();
    //设置窗口的大小和位置
                UserAddFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
                UserAddFrame.pack();
                UserAddFrame.show();
            } else if (e.getActionCommand() == "修改用户") {
    //"修改用户"菜单的响应函数
                UserModify UserModifyFrame = new UserModify();
                Dimension FrameSize = UserModifyFrame.getPreferredSize();
```

```
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
        //设置窗口的大小和位置
            UserModifyFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            UserModifyFrame.pack();
            UserModifyFrame.show();
        } else if (e.getActionCommand() == "删除用户") {
        //初始化"删除用户"窗口
            UserDelete UserDeleteFrame = new UserDelete();
            Dimension FrameSize = UserDeleteFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
             Point loc = getLocation();
        //设置窗口的大小和位置
            UserDeleteFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            UserDeleteFrame.pack();
        //显示窗口
            UserDeleteFrame.show();
        } else if (e.getActionCommand() == "添加书籍") {
        //"添加书籍"窗口
            BookAdd BookAddFrame = new BookAdd();
            Dimension FrameSize = BookAddFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
        //设置窗口的大小和位置
            BookAddFrame.setLocation((MainFrameSize.width - FrameSize.width)
```

```
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
                BookAddFrame.pack();
        //显示添加书籍窗口
                BookAddFrame.show();
            } else if (e.getActionCommand() == "修改书籍") {
        //初始化"修改书籍"窗口
                BookModify BookModifyFrame = new BookModify();
                Dimension FrameSize = BookModifyFrame.getPreferredSize();
                Dimension MainFrameSize = getSize();
                Point loc = getLocation();
        //设置窗口的大小和位置
                BookModifyFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
                BookModifyFrame.pack();
        //显示"修改书籍"窗口
                BookModifyFrame.show();
            } else if (e.getActionCommand() == "删除书籍") {
        //初始化"删除书籍"窗口
                BookDelete BookDeleteFrame = new BookDelete();
                Dimension FrameSize = BookDeleteFrame.getPreferredSize();
                Dimension MainFrameSize = getSize();
                Point loc = getLocation();
        //设置窗口的大小和位置
                BookDeleteFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
                BookDeleteFrame.pack();
```

```
//显示窗口
            BookDeleteFrame.show();
        } else if (e.getActionCommand() == "书籍出借") {
//初始化“书籍出借”窗口
            BorrowBook BorrowBookFrame = new BorrowBook();
            Dimension FrameSize = BorrowBookFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
//设置窗口的大小和位置
            BorrowBookFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BorrowBookFrame.pack();
//显示窗口
            BorrowBookFrame.show();
        } else if (e.getActionCommand() == "出借信息修改") {
//初始化“出借信息修改”窗口
            BorrowInfo BorrowInfoFrame = new BorrowInfo();
            Dimension FrameSize = BorrowInfoFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
//设置窗口的大小和位置
            BorrowInfoFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BorrowInfoFrame.pack();
//显示窗口
            BorrowInfoFrame.show();
        } else if (e.getActionCommand() == "书籍还入") {
```

```
//初始化“书籍还入”窗口
            ReturnBook returnBookFrame = new ReturnBook();
            Dimension FrameSize = returnBookFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
//设置窗口的大小和位置
            returnBookFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
            returnBookFrame.pack();
//显示窗口
            returnBookFrame.show();
        } else if (e.getActionCommand() == "书籍还入信息修改") {
//初始化“书籍还入信息修改”窗口
            ReturnInfo returnInfoFrame = new ReturnInfo();
            Dimension FrameSize = returnInfoFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
//设置窗口的大小和位置
            returnInfoFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
            returnInfoFrame.pack();
//显示窗口
            returnInfoFrame.show();
        } else if (e.getActionCommand() == "书籍列表") {
//初始化“书籍列表”窗口
            BookList BookListFrame = new BookList();
            Dimension FrameSize = BookListFrame.getPreferredSize();
```

```
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
        //设置窗口的大小和位置
            BookListFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BookListFrame.pack();
        //显示窗口
            BookListFrame.show();
        } else if (e.getActionCommand() == "借阅情况表") {
        //初始化“借阅情况表”窗口
            BorrowBookList BorrowBookListFrame = new BorrowBookList();
            Dimension FrameSize = BorrowBookListFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
        //设置窗口的大小和位置
            BorrowBookListFrame.setLocation(
                    (MainFrameSize.width - FrameSize.width) / 2 + loc.x,
                    (MainFrameSize.height - FrameSize.height) / 2 + loc.y);
            BorrowBookListFrame.pack();
        //显示窗口
            BorrowBookListFrame.show();
        } else if (e.getActionCommand() == "用户列表") {
        //显示“用户列表”窗口
            UserList UserListFrame = new UserList();
            Dimension FrameSize = UserListFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
        //设置窗口的大小和位置
```

```
                UserListFrame.setLocation((MainFrameSize.width - FrameSize.width)
                        / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                        + loc.y);
                UserListFrame.pack();
        //显示窗口
                UserListFrame.show();
            } else if (e.getActionCommand() == "退出") {
                this.dispose();
        //系统退出
                System.exit(0);
            }
        }
```

该方法为书籍管理系统的各个菜单项添加了事件响应函数，并设置了窗口的大小和位置。

8.8.3 为系统增加权限控制

本小节为系统增加权限控制，从而保证系统的安全性，共分为系统管理员、书籍管理员和借阅管理员 3 个角色。通过 Menu 对象的 setEnabled（true）方法，根据不同角色的权限分配设置各个菜单项的显示与否。其中的系统管理员具有全部的权限。

跟我做

在“LibraryWindow.java”中创建“setEnable”方法，编辑该方法，输入如下代码：

```
    public void setEnable(String powerType) {
            if (powerType.trim().equals("系统管理员")) {
        //系统管理员具有全部的权限
                userMGRMenu.setEnabled(true);
                bookMGRMenu.setEnabled(true);
                borrowBookMenu.setEnabled(true);
```

```
            returnBookMenu.setEnabled(true);
            infoBrowseMenu.setEnabled(true);
            userListMenuItem.setEnabled(true);
        } else if (powerType.trim().equals("书籍管理员")) {
            //书籍管理员拥有书籍管理和信息查询权限
            userMGRMenu.setEnabled(false);
            bookMGRMenu.setEnabled(true);
            borrowBookMenu.setEnabled(false);
            returnBookMenu.setEnabled(false);
            infoBrowseMenu.setEnabled(true);
            userListMenuItem.setEnabled(false);
        } else if (powerType.trim().equals("借阅管理员")) {
            //借阅管理员拥有借书管理、还书管理和信息查询权限
            userMGRMenu.setEnabled(false);
            bookMGRMenu.setEnabled(false);
            borrowBookMenu.setEnabled(true);
            returnBookMenu.setEnabled(true);
            infoBrowseMenu.setEnabled(true);
            userListMenuItem.setEnabled(false);
        } else if (powerType.trim().equals("else")) {
            //其他角色没有任何权限
            userMGRMenu.setEnabled(false);
            bookMGRMenu.setEnabled(false);
            borrowBookMenu.setEnabled(false);
            returnBookMenu.setEnabled(false);
            infoBrowseMenu.setEnabled(false);
        }
    }
```

8.9 用户管理

用户管理包括用户登录和用户登录信息维护两大部分，用户管理又包括添加用户、删除用户和修改用户 3 部分功能。本节实现用户管理功能。用户管理功能是典型的数据库操作，包括记录的添加、删除和修改等操作。经过本节的学习可以熟练掌握通过 Hibernate 实现对数据库的操作。

8.9.1 用户登录功能的实现

根据实现了图书管理系统的用户登录类，根据用户名和密码查询用户相应的权限。根据查询的权限结果，设置相应菜单项的显示与否。取得用户名和用户密码之后，通过如下代码段从数据库中查询相应的用户信息，如果查询的结果为空，说明该用户没有注册，将提示用户为非法用户。

```
String hql = "from UserTable where UserName='"
                                + userNameText.getText().trim() + "' and Password='"
                                + passwordStr + "'";
// 执行查询
Query userList = session.createQuery(hql);
```

跟我做

在“Library”工程的“src”文件夹中创建“library.user”包，并在其中创建“UserLogin.java”文件，在该文件中输入如下代码：

```
package library.user;
/**
 * 用户登录类，根据用户名和密码查询用户相应的权限
 *
 * @author lianhw
 *
 */
public class UserLogin extends JFrame implements ActionListener {
        LibraryWindow libraryWindow;
```

```
JPanel panel1, panel2;
JLabel userNameLabel, passwordLabel;
JTextField userNameText;
JPasswordField passwordText;
JButton yesButton, cancelButton;
Container container;
ResultSet rs;
public UserLogin(LibraryWindow mainFrame) {
        super("用户登录");
        this.libraryWindow = mainFrame;
        // “用户名”标签
        userNameLabel = new JLabel("用户名", JLabel.CENTER);
        // “密码”标签
        passwordLabel = new JLabel("密码", JLabel.CENTER);
        // 用户名输入框
        userNameText = new JTextField(10);
        // 密码输入框
        passwordText = new JPasswordField(10);
        // “确定”按钮
        yesButton = new JButton("确定");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“确定”按钮增加事件监听者
        yesButton.addActionListener(this);
        // 为“取消”按钮增加事件监听者
        cancelButton.addActionListener(this);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 2));
        panel2 = new JPanel();
```

```
        container = getContentPane();
        container.setLayout(new BorderLayout());
        panel1.add(userNameLabel);
        panel1.add(userNameText);
        panel1.add(passwordLabel);
        panel1.add(passwordText);
        container.add(panel1, BorderLayout.CENTER);
        panel2.add(yesButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
        // 设置窗口大小
        setSize(300, 300);
    }
    /**
     * 动作响应方法，完成数据库的查询和权限的设置功能
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 如果单击“取消”按钮后执行的动作
            libraryWindow.setEnable("else");
            this.dispose();
        } else {
            // 如果单击“确定”按钮后执行的动作
            char[] password = passwordText.getPassword();
            // 将用户输入的密码由字符数组转换成字符串
            String passwordStr = new String(password);
            // 判断用户输入的用户名是否为空
```

```
if (userNameText.getText().trim().equals("")) {
        JOptionPane.showMessageDialog(null, "用户名不可为空!");
        return;
}
// 判断用户输入的密码是否为空
if (passwordStr.equals("")) {
        JOptionPane.showMessageDialog(null, "密码不可为空!");
        return;
}
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// hsql 执行语句
String hql = "from UserTable where UserName='"
            + userNameText.getText().trim() + "' and Password='"
            + passwordStr + "'";
// 执行查询
Query userList = session.createQuery(hql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
// 标记该用户是否存在
boolean isExist = false;
// 如果 list 的长度为 0，表示该用户不存在
if (list.size() > 0)
        isExist = true;
if (!isExist) {
```

图 8-18 新增用户效果图

```
                                        // 提示用户名和密码不正确
                                        JOptionPane.showMessageDialog(null,
"用户名不存在或者密码不正确!");
                        libraryWindow.setEnable("else");
                    } else {
                        // 取得该用户的权限级别
                        UserTable user = (UserTable) list.get(0);
                        // 设置权限
                        libraryWindow.setEnable(user.getPower().trim());
                        // 事务提交
                        tx.commit();
                        // 关闭 session
                        session.close();
                        this.dispose();
                    }
                }
            }
        }
```

该类实现了用户登录功能，其运行效果如图 8-17 所示。根据用户输入的用户名和密码查询数据库中的 UserTable 表验证该用户是否是注册用户。

该类使用 Hibernate 从数据库中查询需要的信息，通过验证后根据用户不同的权限设置其相应权限。

8.9.2 添加用户类的实现

本小节实现增加用户类，将新用户信息保存到数据库中，从而完成用户的注册。其运行效果如图 8-18 所示。将新用户的用户名、密码和登录权限等信息提交到数据库中保存。

该类使用 Hibernate 实现了数据库的插入或更新操作，通过本类可以看出 Hibernate 操作数据库完全以面向对象的方式来实现。

跟我做

在“Library”工程的“library.user”包中创建“UserAdd.java”文件，在该文件中输入如下代码：

```
package library.user;
/**
 * 增加用户类，将用户信息保存到数据库中
 *
 * @author lianhw
 *
 */
public class UserAdd extends JFrame implements ActionListener {



	Container container;
	JPanel panel1, panel2;
	JLabel userNameLabel, passwordLabel, passwordConfirmLabel, loginPrivelegeLabel;
	JTextField userNameText;
	JPasswordField passwordText, passwordConfirmText;
	JComboBox loginPrivelegeComboBox;
	JButton addButton, cancelButton;
	public UserAdd() {
		super("添加用户");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		//“用户名”标签
		userNameLabel = new JLabel("用户名", JLabel.CENTER);
		//“密码”标签
		passwordLabel = new JLabel("密码", JLabel.CENTER);
		//“确认密码”标签
		passwordConfirmLabel = new JLabel("确认密码", JLabel.CENTER);
		//“登录权限”标签
		loginPrivelegeLabel = new JLabel("登录权限", JLabel.CENTER);
```

```
//输入用户名的文本框
userNameText = new JTextField(10);
//输入密码的文本框
passwordText = new JPasswordField(10);
//密码确认文本框
passwordConfirmText = new JPasswordField(10);
//选择用户权限
loginPrivelegeComboBox = new JComboBox();
loginPrivelegeComboBox.addItem("系统管理员");
loginPrivelegeComboBox.addItem("书籍管理员");
loginPrivelegeComboBox.addItem("借阅管理员");
//“添加”按钮
addButton = new JButton("添加");
//“取消”按钮
cancelButton = new JButton("取消");
//为“添加”按钮加入事件监听者
addButton.addActionListener(this);
//为“取消”按钮加入事件监听者
cancelButton.addActionListener(this);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(4, 2));
panel1.add(userNameLabel);
panel1.add(userNameText);
panel1.add(passwordLabel);
panel1.add(passwordText);
panel1.add(passwordConfirmLabel);
panel1.add(passwordConfirmText);
panel1.add(loginPrivelegeLabel);
panel1.add(loginPrivelegeComboBox);
```

```
        container.add(panel1, BorderLayout.CENTER);
        panel2 = new JPanel();
        panel2.add(addButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
        //设置窗口的大小
        setSize(300, 300);
    }
    /**
     * 动作响应方法，将新增的用户信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 单击“取消”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == addButton) {
            // 单击“添加”按钮后将用户信息提交到数据库中
            if (userNameText.getText().trim().equals("")) {
                // 判断用户名是否为空
                JOptionPane.showMessageDialog(null, "用户名不能为空！");
            } else if (passwordText.getText().trim().equals("")) {
                // 判断密码是否为空
                JOptionPane.showMessageDialog(null, "密码不能为空！");
            } else if (!passwordText.getText().trim().equals(
                    passwordConfirmText.getText().trim())) {
                // 判断两次输入的密码是否一致
                JOptionPane.showMessageDialog(null, "两次输入的密码不一致！");
```

```
                } else {
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil
                                        .getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        //创建 UserTable 对象
                        UserTable user = new UserTable();
                        //设置 user 对象的名字
                        user.setUserName(userNameText.getText().trim());
                        //设置 user 对象的密码
                        user.setPassword(passwordText.getText().trim());
                        //设置 usr 对象的权限
                        user.setPower(loginPrivelegeComboBox.getSelectedItem() + "");
                        //保存 user 对象
                        session.saveOrUpdate(user);
                        // 事务提交
                        tx.commit();
                        // 关闭 session
                        session.close();
                        this.dispose();
                }
            }
        }
}
```

图8-19 修改用户信息效果图

8.9.3 修改用户信息类的实现

本小节实现了修改用户信息类，将新的用户注册信息保存到数据库中。其运行效果如图 8-19 所示。

该类通过 Hibernatyye 完全以面向对象的方式实现对数据库的更新操作。

跟我做

在“Library”工程的“library.user”包中创建“UserModify.java”文件，在该文件中输入如下代码：

```
package library.user;
/**
 * 修改用户信息类，将新的用户注册信息保存到数据库中
 *
 * @author lianhw
 *
 */
public class UserModify extends JFrame implements ActionListener {
        JPanel panel1, panel2;
        Container container;
        JLabeluserLabel,passwordLabel,newPasswordLabel,passwordConfirmLabel,loginPrivelege
Label;
        JTextField userNameText;
        JPasswordField passwordText, newPasswordText, passwordConfirmText;
        JButton updateButton, cancelButton;
        JComboBox loginPrivelegeComboBox;
        public UserModify() {
                super("更改密码");
                container = getContentPane();
                container.setLayout(new BorderLayout());
                // “用户名”标签
                userLabel = new JLabel("用户名", JLabel.CENTER);
```

```
// “原密码”标签
passwordLabel = new JLabel("原密码", JLabel.CENTER);
// “新密码”标签
newPasswordLabel = new JLabel("新密码", JLabel.CENTER);
// “确认新密码”标签
passwordConfirmLabel = new JLabel("确认新密码", JLabel.CENTER);
//“选择权限”标签
loginPrivelegeLabel=new JLabel("登录权限", JLabel.CENTER);
// 输入用户名文本框
userNameText = new JTextField(10);
// 输入密码文本框
passwordText = new JPasswordField(10);
// 输入新密码文本框
newPasswordText = new JPasswordField(10);
// 密码确认文本框
passwordConfirmText = new JPasswordField(10);
// 选择用户权限
loginPrivelegeComboBox = new JComboBox();
loginPrivelegeComboBox.addItem("系统管理员");
loginPrivelegeComboBox.addItem("书籍管理员");
loginPrivelegeComboBox.addItem("借阅管理员");
// “更新”按钮
updateButton = new JButton("更新");
// “取消”按钮
cancelButton = new JButton("取消");
// 为“更新”按钮添加动作监听者
updateButton.addActionListener(this);
// 为“取消”按钮添加动作监听者
cancelButton.addActionListener(this);
```

```
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(5, 2));
        panel1.add(userLabel);
        panel1.add(userNameText);
        panel1.add(passwordLabel);
        panel1.add(passwordText);
        panel1.add(newPasswordLabel);
        panel1.add(newPasswordText);
        panel1.add(passwordConfirmLabel);
        panel1.add(passwordConfirmText);
        panel1.add(loginPrivelegeLabel);
        panel1.add(loginPrivelegeComboBox);
        panel2 = new JPanel();
        panel2.add(updateButton);
        panel2.add(cancelButton);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
        // 设置窗口的大小
        setSize(300, 300);
    }
    /**
     * 动作响应方法，修改用户的注册信息
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 单击"取消"按钮不作任何事情
            this.dispose();
```

```
} else if (action.getSource() == updateButton) {
        // 单击"更新"按钮将新的用户信息保存到数据库中
        char[] password = passwordText.getPassword();
        // 将密码转成字符串
        String passwordStr = new String(password);
        char[] newPassword = newPasswordText.getPassword();
        String newPasswordStr = new String(newPassword);
        char[] confirmPassword = passwordConfirmText.getPassword();
        String confirmPasswordStr = new String(confirmPassword);
        // 判断输入的用户名是否为空
        if (userNameText.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "用户名不能为空！");
        } else if (passwordStr.equals("")) {
            // 判断输入的密码是否为空
            JOptionPane.showMessageDialog(null, "原密码不能为空！");
        } else if (!newPasswordStr.equals(confirmPasswordStr)) {
            // 判断输入的新密码和确认密码是否一致
            JOptionPane.showMessageDialog(null, "两次输入的新密码不一致！");
        } else {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil
                    .getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // 创建 UserTable 对象
            UserTable user = new UserTable();
            // 设置 user 对象的名字
```


```
                                    user.setUserName(userNameText.getText().trim());
                                    // 设置 user 对象的密码
                    user.setPassword(newPasswordText.getText().trim());
                    // 设置 user 对象的权限
                    user.setPower(loginPrivelegeComboBox.getSelectedItem()+"");
                    // 保存 user 对象
                    session.saveOrUpdate(user);
                    // 事务提交
                    tx.commit();
                    // 关闭 session
                    session.close();
                    this.dispose();
                }
            }
        }
}
```

8.9.4 删除用户类的实现

本小节实现了删除用户类，将一些用户从系统中删除。其运行效果如图 8-20 所示。

该类通过 Hibernate 实现了从数据库中删除记录的功能。

跟我做

在“Library”工程的 “library.user”包中创建“UserDelete.java”文件，在该文件中输入如下代码：

```
package  library.user;
/**
 *
 * 删除用户类，将一些用户从系统中删除
 *
```

```
 * @author lianhw
 *
 */
public class UserDelete extends JFrame implements ActionListener {
	JPanel panel1, panel2;
	Container container;
	JLabel userLabel, passwordLabel;
	JTextField userText;
	JPasswordField passwordText;
	JButton yesButton, cancelButton;
	public UserDelete() {
		super("删除用户");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		//“用户名”标签
		userLabel = new JLabel("用户名", JLabel.CENTER);
		//“密码”标签
		passwordLabel = new JLabel("密码", JLabel.CENTER);
		//输入用户名文本框
		userText = new JTextField(10);
		//输入密码文本框
		passwordText = new JPasswordField(10);
		//“确定”按钮
		yesButton = new JButton("确定");
		//“取消”按钮
		cancelButton = new JButton("取消");
		//为“确定”按钮添加事件监听者
		yesButton.addActionListener(this);
		//为“取消”按钮添加事件监听者
```

```
        cancelButton.addActionListener(this);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 2));
        panel1.add(userLabel);
        panel1.add(userText);
        panel1.add(passwordLabel);
        panel1.add(passwordText);
        panel2 = new JPanel();
        panel2.add(yesButton);
        panel2.add(cancelButton);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
        //设置窗口的大小
        setSize(300, 300);
}
/**
 * 动作响应方法，将指定用户从数据库中删除
 *
 * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
 */
public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
                // 如果单击"取消"按钮后执行的动作
                this.dispose();
        } else if (action.getSource() == yesButton) {
                // 单击"确定"按钮后将用户从数据库中删除
                char[] password = passwordText.getPassword();
                // 将用户密码转换成字符串
                String passwordStr = new String(password);
```

```
// 判断用户名是否为空
if (userText.getText().trim().equals("")) {
        JOptionPane.showMessageDialog(null, "用户名不能为空！");
}
// 判断密码是否为空
else if (passwordText.equals("")) {
        JOptionPane.showMessageDialog(null, "密码不能为空！");
} else {
        // 取得 SessionFactory
        SessionFactory sessionFactory = HibernateUtil
                        .getSessionFactory();
        // 打开 session
        Session session = sessionFactory.openSession();
        // 创建一个事务
        Transaction tx = session.beginTransaction();
        // 创建 UserTable 对象
        UserTable user = new UserTable();
        // 设置 user 对象的名字
        user.setUserName(userText.getText().trim());
        // 设置 user 对象的密码
        user.setPassword(passwordText.getText().trim());
        // 删除该用户
        session.delete(user);
        JOptionPane.showMessageDialog(null, "删除成功!");
        // 事务提交
        tx.commit();
        // 关闭 session
        session.close();
        this.dispose();
```

```
                }
            }
        }
}
```

8.9.5 列举所有用户信息类的实现

本小节从数据库中查询所有注册的用户信息，并以表格的方式列出来。其运行效果如图 8-21 所示。该类利用 Hibernate 实现了数据库的查询操作。

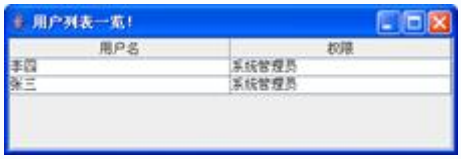

图 8-21 用户列表一览

跟我做

在“Library”工程的“library.user”包中创建“UserList.java”文件，在该文件中输入如下代码：

```
package library.user;
/**
 * 列出数据库中注册的所有用户信息
 *
 * @author lianhw
 *
 */
public class UserList extends JFrame {
        Container container;
        JTable table = null;
        DefaultTableModel defaultModel = null;
        public UserList() {
                super("用户列表一览！");
                container = getContentPane();
                container.setLayout(new BorderLayout());
```

```
// 表的两个列名
String[] name = { "用户名", "权限" };
String[][] data = new String[0][0];
// 表对应的 model
defaultModel = new DefaultTableModel(data, name);
// 新建表格
table = new JTable(defaultModel);
table.setPreferredScrollableViewportSize(new Dimension(400, 80));
JScrollPane scrollPane = new JScrollPane(table);
container.add(scrollPane);
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// hsql 执行语句
String hql = "from UserTable";
// 执行查询
Query userList = session.createQuery(hql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
// 将链表中的数据加入到列表中
for (int index = 0; index < list.size(); index++) {
        Vector insertRow = new Vector();
        insertRow.addElement(((UserTable) list.get(index)).getUserName());
        insertRow.addElement(((UserTable) list.get(index)).getPower());
        defaultModel.addRow(insertRow);
}
```

```
            table.revalidate();
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
        }
}
```

8.10 书籍管理模块

书籍管理模块实现新增书名、修改图书信息和删除图书等功能，本节实现该功能模块。

8.10.1 书籍添加类的实现

本小节实现书籍添加类，将新增加的图书信息保存到数据库中。其运行效果如图 8-22 所示。每本书的信息包括名称、出版社、作者和地址等信息。单击【添加】按钮后将添加的书籍信息通过 Hibernate 插入到数据库中。

图 8-22 添加书籍

跟我做

在“Library”工程的“src”文件夹中创建“library.book”包，并在其中创建“BookAdd.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 书籍添加类，将新增加的图书信息保存到数据库中去
```

```
*
* @author lianhw
*
*/
public class BookAdd extends JFrame implements ActionListener {
	JPanel panel1, panel2;
	JLabel bookNameLabel, pressNameLabel, authorLabel, addressLabel,
		pressDateLabel, priceLabel, bookCountLabel, commentLabel;
	JTextField bookNameText, pressNameText, authorText, addressText,
		pressDateText, priceText, bookCountText, commentText;
	Container container;
	JButton clearButton, addButton, exitButton;
	public BookAdd() {
		super("添加书籍信息");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		//“名称”标签
		bookNameLabel = new JLabel("名称", JLabel.CENTER);
		//“出版社”标签
		pressNameLabel = new JLabel("出版社", JLabel.CENTER);
		//“作者”标签
		authorLabel = new JLabel("作者", JLabel.CENTER);
		//“地址”标签
		addressLabel = new JLabel("地址", JLabel.CENTER);
		//“出版日期”标签
		pressDateLabel = new JLabel("出版日期", JLabel.CENTER);
		//“价格”标签
		priceLabel = new JLabel("价格", JLabel.CENTER);
		//“新书数目”标签
```

```
bookCountLabel = new JLabel("新书数目", JLabel.CENTER);
//“备注”标签
commentLabel = new JLabel("备注", JLabel.CENTER);
//输入书名文本框
bookNameText = new JTextField(15);
//输入出版社名字文本框
pressNameText = new JTextField(15);
//输入作者姓名文本框
authorText = new JTextField(15);
//输入地址文本框
addressText = new JTextField(15);
//输入出版日期文本框
pressDateText = new JTextField(15);
//输入价格文本框
priceText = new JTextField(15);
//输入图书数量文本框
bookCountText = new JTextField(15);
//输入图书备注文本框
commentText = new JTextField(15);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(8, 2));
panel1.add(bookNameLabel);
panel1.add(bookNameText);
panel1.add(pressNameLabel);
panel1.add(pressNameText);
panel1.add(authorLabel);
panel1.add(authorText);
panel1.add(addressLabel);
panel1.add(addressText);
```

```
        panel1.add(pressDateLabel);
        panel1.add(pressDateText);
        panel1.add(priceLabel);
        panel1.add(priceText);
        panel1.add(bookCountLabel);
        panel1.add(bookCountText);
        panel1.add(commentLabel);
        panel1.add(commentText);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 3));
        //“清空”按钮
        clearButton = new JButton("清空");
        //为“清空”按钮添加事件监听者
        clearButton.addActionListener(this);
        //“添加”按钮
        addButton = new JButton("添加");
        //为“添加”按钮添加事件监听者
        addButton.addActionListener(this);
        //“退出”按钮
        exitButton = new JButton("退出");
        //为“退出”按钮添加事件监听者
        exitButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(addButton);
        panel2.add(exitButton);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
```

```
     * 动作响应方法，将新增的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
            // 单击"退出"按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == clearButton) {
            // 单击"清空"按钮将所有输入框清空
            bookNameText.setText("");
            pressNameText.setText("");
            authorText.setText("");
            addressText.setText("");
            pressDateText.setText("");
            priceText.setText("");
            bookCountText.setText("");
            commentText.setText("");
        } else if (action.getSource() == addButton) {
            // 判断书名是否为空
            if (bookNameText.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "书名不能为空！");
            } else if (pressNameText.getText().trim().equals("")) {
                // 判断出版社是否为空
                JOptionPane.showMessageDialog(null, "出版社不能为空！");
            } else if (authorText.getText().trim().equals("")) {
                // 判断作者是否为空
                JOptionPane.showMessageDialog(null, "作者不能为空！");
            } else if (bookCountText.getText().trim().equals("")) {
```

```
            // 判断新书数目是否为空
            JOptionPane.showMessageDialog(null, "新书数目不能为空！");
        } else {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil
                    .getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // 创建 UserTable 对象
            Books book = new Books();
            book.setBookName(bookNameText.getText().trim());
            book.setPress(pressNameText.getText().trim());
            book.setAuthor(authorText.getText().trim());
            book.setAddress(addressText.getText().trim());
            book.setPressDate(new GregorianCalendar().getTime());
            book.setPrice(new Double(priceText.getText().trim()));
            book.setBooksCount(new Integer(bookCountText.getText().trim()));
            book.setCom(commentText.getText().trim());
            session.saveOrUpdate(book);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
            JOptionPane.showMessageDialog(null, "添加书籍成功！");
            this.dispose();
        }
    }
```

```
    }
}
```

8.10.2 书籍信息修改类的实现

本小节实现图书信息修改类，将修改后的图书信息提交到数据库中。其运行效果如图 8-23 所示。该类通过 Hibernate 实现了对数据库的查询和更新操作。

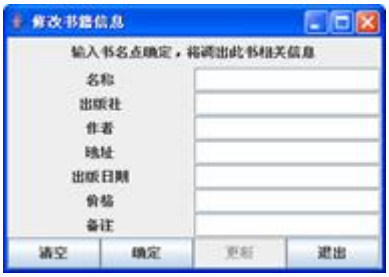

图 8-23 修改图书信息

跟我做

在“library.book”包中创建“BookModify.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 图书信息修改类，将修改后的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BookModify extends JFrame implements ActionListener {
        JPanel panel1, panel2, panel3;
        JLabel tipLabel = new JLabel("输入书名点确定，将调出此书相关信息");
        JLabel bookNameLabel, pressNameLabel, authorLabel, addressLabel,
                        pressDateLabel, priceLabel, commentLabel;
        JTextField bookNameText, pressNameText, authorText, addressText,
                        pressDateText, priceText, commentText;
```

```
Container container;
JButton clearButton, yesButton, updateButton, exitButton;
// 用来保存图书的数量
private int count;
/**
 * 类的构造函数，完成界面的初始化
 */
public BookModify() {
        super("修改书籍信息");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        panel3 = new JPanel();
        panel3.add(tipLabel);
        container.add(panel3, BorderLayout.NORTH);
        // “名称”标签
        bookNameLabel = new JLabel("名称", JLabel.CENTER);
        // “出版社”标签
        pressNameLabel = new JLabel("出版社", JLabel.CENTER);
        // “作者”标签
        authorLabel = new JLabel("作者", JLabel.CENTER);
        // “地址”标签
        addressLabel = new JLabel("地址", JLabel.CENTER);
        // “出版日期”标签
        pressDateLabel = new JLabel("出版日期", JLabel.CENTER);
        // “价格”标签
        priceLabel = new JLabel("价格", JLabel.CENTER);
        // “备注”标签
        commentLabel = new JLabel("备注", JLabel.CENTER);
        // 书籍名称文本框
```

```
bookNameText = new JTextField(15);
// 输入出版社名称文本框
pressNameText = new JTextField(15);
// 输入作者文本框
authorText = new JTextField(15);
// 输入地址文本框
addressText = new JTextField(15);
// 输入出版日期文本框
pressDateText = new JTextField(15);
// 输入价格文本框
priceText = new JTextField(15);
// 输入备注信息文本框
commentText = new JTextField(15);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(7, 2));
panel1.add(bookNameLabel);
panel1.add(bookNameText);
panel1.add(pressNameLabel);
panel1.add(pressNameText);
panel1.add(authorLabel);
panel1.add(authorText);
panel1.add(addressLabel);
panel1.add(addressText);
panel1.add(pressDateLabel);
panel1.add(pressDateText);
panel1.add(priceLabel);
panel1.add(priceText);
panel1.add(commentLabel);
panel1.add(commentText);
```

```
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 4));
        // “清空”按钮
        clearButton = new JButton("清空");
        // “确定”按钮
        yesButton = new JButton("确定");
        // “更新”按钮
        updateButton = new JButton("更新");
        // “退出”按钮
        exitButton = new JButton("退出");
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(updateButton);
        panel2.add(exitButton);
        // 为“清空”按钮添加监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加监听者
        yesButton.addActionListener(this);
        // 为“更新”按钮添加监听者
        updateButton.addActionListener(this);
        // 为“退出”按钮添加监听者
        exitButton.addActionListener(this);
        updateButton.setEnabled(false);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将修改后的图书信息提交到数据库中
     *
```

```
 * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
 */
public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
                // 单击“退出”按钮不作任何事情
                this.dispose();
        } else if (action.getSource() == clearButton) {
                // 单击“清空”按钮将所有文本框中的内容清空
                bookNameText.setText("");
                pressNameText.setText("");
                authorText.setText("");
                addressText.setText("");
                pressDateText.setText("");
                priceText.setText("");
                commentText.setText("");
        } else if (action.getSource() == yesButton) {
                // 单击“确定”按钮将图书信息读出
                // 取得 SessionFactory
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                // 打开 session
                Session session = sessionFactory.openSession();
                // 创建一个事务
                Transaction tx = session.beginTransaction();
                // hsql 执行语句
                String hql = "from Books where bookName='"
                                + bookNameText.getText().trim() + "'";
                // 执行查询
                Query bookInfoList = session.createQuery(hql);
                // 将查询结果放置到一个 list 链表中
```

```
            List list = bookInfoList.list();
            if (bookNameText.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "请输入书名：<*v*>");
            } else if (list.size() == 0) {
                JOptionPane.showMessageDialog(null, "此书没有在书库中...");
            } else {
                Books book = (Books) list.get(0);
                bookNameText.setText(book.getBookName());
                pressNameText.setText(book.getPress());
                authorText.setText(book.getAuthor());
                addressText.setText(book.getAddress());
                pressDateText.setText(book.getPressDate().toString());
                priceText.setText(book.getPrice().toString());
                commentText.setText(book.getCom());
                count = book.getBooksCount().intValue();
                updateButton.setEnabled(true);
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
            }
        } else if (action.getSource() == updateButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // 创建 UserTable 对象
```


```
                Books book = new Books();
                book.setBookName(bookNameText.getText().trim());
            book.setPress(pressNameText.getText().trim());
            book.setAuthor(authorText.getText().trim());
            book.setAddress(addressText.getText().trim());
            book.setPressDate(new GregorianCalendar().getTime());
            book.setPrice(new Double(priceText.getText().trim()));
            book.setCom(commentText.getText().trim());
            book.setBooksCount(new Integer(count));
            session.saveOrUpdate(book);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
            JOptionPane.showMessageDialog(null, "修改书籍成功！");
        }
    }
}
```

8.10.3 书籍删除类的实现

本小节实现书籍删除类，将指定名字的书籍从数据库中删除。其运行效果如图 8-24 所示。该类通过 Hibernate 实现了从数据库中删除记录的操作。

跟我做

在“library.book”包中创建“BookDelete.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 书籍删除类，将指定名字的书籍从数据库中删除
```

```
*
* @author lianhw
*
*/
public class BookDelete extends JFrame implements ActionListener {
	Container container;
	JLabel tipLabel = new JLabel("请输入要删除的书名：", JLabel.CENTER);
	JTextField bookDeleteText = new JTextField(15);
	JButton yesButton, exitButton;
	JPanel panel1 = new JPanel();
	public BookDelete() {
		super("删除书籍信息");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		container.add(tipLabel, BorderLayout.NORTH);
		container.add(bookDeleteText, BorderLayout.CENTER);
		// “确定”按钮
		yesButton = new JButton("确定");
		// “退出”按钮
		exitButton = new JButton("退出");
		// 为“确定”按钮添加事件监听者
		yesButton.addActionListener(this);
		// 为“退出”按钮添加事件监听者
		exitButton.addActionListener(this);
		panel1.add(yesButton);
		panel1.add(exitButton);
		container.add(panel1, BorderLayout.SOUTH);
	}
	/**
```

```
     * 动作响应方法，将修改后的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
            // 单击"退出"按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 单击"确定"按钮将图书从数据库中删除
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            Books book = new Books();
            book.setBookName(bookDeleteText.getText().trim());
            session.delete(book);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
            JOptionPane.showMessageDialog(null, "删除书籍成功！");
            this.dispose();
        }
    }
}
```

8.10.4 图书信息一览类的实现

本小节实现图书信息一览类。从数据库中查询到所有的图书信息，并以表格的形式显示。其运行效果如图 8-25 所示。该类通过 Hibernate 实现了从数据库中查询记录的操作。

图 8-25　书籍信息一览

跟我做

在“library.book”包中创建“BookList.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 图书信息一览类
 *
 * @author lianhw
 *
 */
public class BookList extends JFrame implements ActionListener {
        Container container;
        JPanel panel1, panel2, panel3;
        JLabel bookNameLabel, authorLabel, pressLabel;
        JTextField bookNameText, authorText, pressText;
        JButton searchButton, exitButton;
        JTable table = null;
        DefaultTableModel defaultModel = null;
        public BookList() {
                super("书籍信息一览！");
                container = getContentPane();
```

```
container.setLayout(new BorderLayout());
//“名称”标签
bookNameLabel = new JLabel("名称", JLabel.CENTER);
//“作者”标签
authorLabel = new JLabel("作者", JLabel.CENTER);
//“出版社”标签
pressLabel = new JLabel("出版社", JLabel.CENTER);
//输入书名文本框
bookNameText = new JTextField(15);
//输入作者姓名文本框
authorText = new JTextField(15);
//输入出版社社名文本框
pressText = new JTextField(15);
//“查询”按钮
searchButton = new JButton("查询");
//为“查询”按钮增加事件监听者
searchButton.addActionListener(this);
//“退出”按钮
exitButton = new JButton("退出");
//为“退出”按钮添加事件监听者
exitButton.addActionListener(this);
panel1 = new JPanel();
panel3 = new JPanel();
panel1.add(bookNameLabel);
panel1.add(bookNameText);
panel1.add(authorLabel);
panel1.add(authorText);
panel3.add(pressLabel);
panel3.add(pressText);
```

```
        panel3.add(searchButton);
        panel3.add(exitButton);
        //表格的列名
        String[] name = { "书名", "出版社", "作者", "地址", "出版日期", "定价", "评论" };
        String[][] data = new String[0][0];
        defaultModel = new DefaultTableModel(data, name);
        table = new JTable(defaultModel);
        table.setPreferredScrollableViewportSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table);
        panel2 = new JPanel();
        panel2.add(s);
        container.add(panel1, BorderLayout.NORTH);
        container.add(panel3, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
}
/**
 * 动作响应方法，从数据库中查询所有图书信息
 *
 * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
 */
public void actionPerformed(ActionEvent action) {
        if (action.getSource() == searchButton) {
                String hql = " from Books";
                String strSql = null;
                if (bookNameText.getText().trim().equals("")
                                && authorText.getText().trim().equals("")
                                && pressText.getText().trim().equals("")) {
                        // 如果没有查询条件
                        strSql = hql;
```

```
} else if (bookNameText.getText().trim().equals("")
        && authorText.getText().trim().equals("")) {
    // 按照出版社查询书籍
    strSql = hql + " where press='" + pressText.getText().trim()
        + "'";
} else if (bookNameText.getText().trim().equals("")
        && pressText.getText().trim().equals("")) {
    // 按照作者姓名查询书籍
    strSql = hql + " where author='" + authorText.getText().trim()
        + "'";
} else if (authorText.getText().trim().equals("")
        && pressText.getText().trim().equals("")) {
    // 按照书名查询书籍
    strSql = hql + " where bookName='"
        + bookNameText.getText().trim() + "'";
} else if (bookNameText.getText().trim().equals("")) {
    // 按照作者和出版社两个条件来查询书籍
    strSql = hql + " where author='" + authorText.getText().trim()
        + "'and press='" + pressText.getText().trim() + "'";
} else if (authorText.getText().trim().equals("")) {
    // 按照书名和出版社两个条件来查询书籍
    strSql = hql + " where bookName='"
        + bookNameText.getText().trim() + "'and press='"
        + pressText.getText().trim() + "'";
} else if (pressText.getText().trim().equals("")) {
    // 按照书名和作者两个条件来查询书籍
    strSql = hql + " where bookname='"
        + bookNameText.getText().trim() + "'and author='"
        + authorText.getText().trim() + "'";
```

```
} else {
        // 按照书名、作者和出版社 3 个条件来查询书籍
        strSql = hql + " where bookname='"
                        + bookNameText.getText().trim() + "'and author='"
                        + authorText.getText().trim() + "'and press='"
                        + pressText.getText().trim() + "'";
}
// 首先要删除 table 中的数据
int rowCount = defaultModel.getRowCount() - 1;// 取得 table 中的数据行；
int j = rowCount;
for (int i = 0; i <= rowCount; i++) {
        defaultModel.removeRow(j);// 删除 rowCount 行的数据；
        defaultModel.setRowCount(j);// 重新设置行数；
        j = j - 1;
}
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// 执行查询
Query userList = session.createQuery(strSql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
for (int index = 0; index < list.size(); index++) {
        Vector data = new Vector();
        Books book = (Books) list.get(index);
        data.addElement(book.getBookName());
```

```
                    data.addElement(book.getPress());
                    data.addElement(book.getAuthor());
                    data.addElement(book.getAddress());
                    data.addElement(book.getPressDate());
                    data.addElement(book.getPrice());
                    data.addElement(book.getCom());
                    defaultModel.addRow(data);
                }
                table.revalidate();
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
            } else if (action.getSource() == exitButton) {
                this.dispose();
            }
        }
}
```

8.11 借书管理模块

借书管理实现了对出借图书的信息维护。包括书籍出借和出借信息修改两部分功能。本节实现该功能模块。

8.11.1 借阅图书类的实现

本小节实现借阅图书类，将借阅的图书信息提交到数据库中。其运行效果如图 8-26 所示。该类通过 Hibernate 实现了对数据库的插入操作。

图 8-26　书籍出借窗口

跟我做

在“library.book”包中创建“BorrowBook.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 借阅图书类，将借阅的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BorrowBook extends JFrame implements ActionListener {
        JPanel panel1, panel2;
        Container container;
        JLabel borrowedBookStudentLabel, borrowedBookNameLabel, borrowedDateLabel,
                        returnDateLabel, borrowedCommentLabel;
        JTextField borrowedBookStudentText, borrowedDateText, returnDateText,
                        borrowedCommentText;
        JButton clearButton, yesButton, cancelButton;
        JComboBox bookNameComboBox = new JComboBox();
        /**
         * 类的构造函数，完成界面的初始化
         */
        public BorrowBook() {
                super("书籍出借");
```

```
container = getContentPane();
container.setLayout(new BorderLayout());
// “借阅者姓名”标签
borrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
// “书名”标签
borrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
// “借阅日期”标签
borrowedDateLabel = new JLabel("借阅日期", JLabel.CENTER);
// “归还日期”标签
returnDateLabel = new JLabel("归还日期", JLabel.CENTER);
// “备注”标签
borrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
// 输入借阅者姓名文本框
borrowedBookStudentText = new JTextField(15);
// 输入借阅日期文本框
borrowedDateText = new JTextField(15);
// 输入归还日期文本框
returnDateText = new JTextField(15);
// 输入备注信息文本框
borrowedCommentText = new JTextField(15);
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// hsql 执行语句
String hql = "from Books where books_count>borrowed_count";
// 执行查询
```

```
Query userList = session.createQuery(hql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
// 将查询到所有符合条件的图书加入到出借列表中
for (int index = 0; index < list.size(); index++) {
        bookNameComboBox.addItem(((Books) list.get(index)).getBookName());
}
// 事务提交
tx.commit();
// 关闭 session
session.close();
panel1 = new JPanel();
panel1.setLayout(new GridLayout(5, 2));
panel1.add(borrowedBookStudentLabel);
panel1.add(borrowedBookStudentText);
panel1.add(borrowedBookNameLabel);
panel1.add(bookNameComboBox);
panel1.add(borrowedDateLabel);
panel1.add(borrowedDateText);
panel1.add(returnDateLabel);
panel1.add(returnDateText);
panel1.add(borrowedCommentLabel);
panel1.add(borrowedCommentText);
container.add(panel1, BorderLayout.CENTER);
panel2 = new JPanel();
panel2.setLayout(new GridLayout(1, 3));
// “清空”按钮
clearButton = new JButton("清空");
// “确定”按钮
```

```
        yesButton = new JButton("确定");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“清空”按钮添加信息监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加信息监听者
        yesButton.addActionListener(this);
        // 为“取消”按钮添加信息监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将出借的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == clearButton) {
            // 单击“清空”按钮将文本框中的信息清空
            borrowedBookStudentText.setText("");
            borrowedDateText.setText("");
            borrowedCommentText.setText("");
        } else if (action.getSource() == yesButton) {
```

```
                // 验证输入的借阅者姓名是否为空
                if (borrowedBookStudentText.getText().trim().equals("")) {
                        JOptionPane.showMessageDialog(null, "请输入借阅者的姓名...");
                } else if (bookNameComboBox.getSelectedItem().equals("")) {
                        // 验证现在是否有书可以借阅
                        JOptionPane.showMessageDialog(null, "对不起，现在书库里没有书，\n 你现
在不能借书!");
                } else {
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil
                                        .getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        // 创建 UserTable 对象
                        BookBrowse browse = new BookBrowse();
                         browse.setBookName(bookNameComboBox.getSelectedItem() + "");
                        browse.setStudentName(borrowedBookStudentText.getText().trim());
                        browse.setBorrowDate(new GregorianCalendar().getTime());
                        browse.setCom(borrowedCommentText.getText().trim());
                        browse.setReturnDate(new GregorianCalendar().getTime());
                        session.saveOrUpdate(browse);
                        // 事务提交
                        tx.commit();
                        // 关闭 session
                        session.close();
                        JOptionPane.showMessageDialog(null, "借阅书籍成功！");
                        this.dispose();
```

```
                }
            }
        }
}
```

8.11.2 修改出借图书信息类的实现

本小节实现了修改出借图书信息类，将修改后的图书信息提交到数据库中。其运行效果如图 8-27 所示。该类通过 Hibernate 实现了从数据库中查询记录和更新记录的操作。

图 8-27 修改书籍出借信息

跟我做

在“Library”工程的“src”文件夹中创建“library.info”包，并在其中创建“BorrowInfo.java”文件，在该文件中输入如下代码：

```
package library.info;
/**
 * 修改出借图书信息类，将修改后的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BorrowInfo extends JFrame implements ActionListener {
        JPanel panel1, panel2, panel3;
        Container container;
        JLabel tipLabel = new JLabel("输入借阅者姓名和书名点击确定，将调出此书的相关信息");
        JLabel borrowedBookStudentLabel, borrowedBookNameLabel, borrowedDateLabel,
                borrowedCommentLabel;
```

```
JTextField borrowedBookStudentText, borrowedBookNameText, borrowedDateText,
        borrowedCommentText;
JButton clearButton, yesButton, updateButton, cancelButton;
public BorrowInfo() {
    super("修改书籍出借信息");
    container = getContentPane();
    container.setLayout(new BorderLayout());
    panel3 = new JPanel();
    panel3.add(tipLabel);
    container.add(panel3, BorderLayout.NORTH);
    // “借阅者姓名”标签
    borrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
    // “书名”标签
    borrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
    // “借书日期”标签
    borrowedDateLabel = new JLabel("借书日期", JLabel.CENTER);
    // “备注”标签
    borrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
    // 输入借阅者姓名的文本框
    borrowedBookStudentText = new JTextField(15);
    // 输入书名文本框
    borrowedBookNameText = new JTextField(15);
    // 输入出借日期文本框
    borrowedDateText = new JTextField(15);
    // 输入备注信息文本框
    borrowedCommentText = new JTextField(15);
    panel1 = new JPanel();
    panel1.setLayout(new GridLayout(4, 2));
    panel1.add(borrowedBookStudentLabel);
```

```
panel1.add(borrowedBookStudentText);
panel1.add(borrowedBookNameLabel);
panel1.add(borrowedBookNameText);
panel1.add(borrowedDateLabel);
panel1.add(borrowedDateText);
panel1.add(borrowedCommentLabel);
panel1.add(borrowedCommentText);
container.add(panel1, BorderLayout.CENTER);
panel2 = new JPanel();
panel2.setLayout(new GridLayout(1, 4));
// “清空”按钮
clearButton = new JButton("清空");
// “确定”按钮
yesButton = new JButton("确定");
// “更新”按钮
updateButton = new JButton("更新");
// “取消”按钮
cancelButton = new JButton("取消");
// 为“清空”按钮添加监听者
clearButton.addActionListener(this);
// 为“确定”按钮添加监听者
yesButton.addActionListener(this);
// 为“更新”按钮添加监听者
updateButton.addActionListener(this);
updateButton.setEnabled(false);
// 为“取消”按钮添加监听者
cancelButton.addActionListener(this);
panel2.add(clearButton);
panel2.add(yesButton);
```

```
        panel2.add(updateButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        // 单击"清空"按钮，清空所有的文本框
        if (action.getSource() == clearButton) {
            borrowedBookStudentText.setText("");
            borrowedBookNameText.setText("");
            borrowedDateText.setText("");
            borrowedCommentText.setText("");
        } else if (action.getSource() == cancelButton) {
            // 单击"退出"按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // hsql 执行语句
            String hql = "from BookBrowse where studentName='"
                    + borrowedBookStudentText.getText().trim()
```

```
                    + "'and bookName='" + borrowedBookNameText.getText().trim()
                    + "'";
            // 执行查询
            Query userList = session.createQuery(hql);
            // 将查询结果放置到一个 list 链表中
            List list = userList.list();
            BookBrowse browse = (BookBrowse) list.get(0);
            borrowedBookStudentText.setText(browse.getStudentName());
            borrowedBookNameText.setText(browse.getBookName());
            borrowedDateText.setText(browse.getBorrowDate().toString());
            borrowedCommentText.setText(browse.getCom());
            updateButton.setEnabled(true);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
        } else if (action.getSource() == updateButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            BookBrowse browse2 = new BookBrowse();
            browse2.setBookName(borrowedBookNameText.getText().trim());
            browse2.setStudentName(borrowedBookStudentText.getText().trim());
            browse2.setBorrowDate(new GregorianCalendar().getTime());
            browse2.setCom(borrowedCommentText.getText().trim());
            browse2.setReturnDate(new GregorianCalendar().getTime());
```

```
                    session.saveOrUpdate(browse2);
                    JOptionPane.showMessageDialog(null, "修改书籍成功！");
                    // 事务提交
                    tx.commit();
                    // 关闭 session
                    session.close();
                    this.dispose();
                }
            }
}
```

8.12 还书管理模块

还书管理模块实现了对还书信息的管理。包括书籍还入和书籍还入信息修改两部分功能。本节实现该功能模块。

8.12.1 还书类的实现

本小节实现所有有关还书的功能。该类实现了将还书信息通过 Hibernate 保存到数据库中，其运行效果如图 8-28 所示。

图 8-28 书籍还入

跟我做

在“library.info”包中创建“ReturnedBook.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 还书类
 *
 * @author lianhw
```

```
*
*/
public class ReturnBook extends JFrame implements ActionListener {


        JPanel panel1, panel2;
        Container container;
        JLabel returnedBookStudentLabel, returnedBookNameLabel, returnedDateLabel,
                        returnedCommentLabel;
        JTextField returnedBookStudentText, returnedDateText, returnedCommentText;
        JButton clearButton, yseButton, cancelButton;
        JComboBox bookNameComboBox = new JComboBox();
        // 书名和 BookBrowse 类的映射
        private Map bookName2BookBrowse = new HashMap();
        public ReturnBook() {
                super("书籍还入");
                container = getContentPane();
                container.setLayout(new BorderLayout());
                // “还书者姓名”标签
                returnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);
                // “书名”标签
                returnedBookNameLabel = new JLabel("书名", JLabel.CENTER);
                // “日期”标签
                returnedDateLabel = new JLabel("日期", JLabel.CENTER);
                // “备注”标签
                returnedCommentLabel = new JLabel("备注", JLabel.CENTER);
                // 输入归还者姓名文本框
                returnedBookStudentText = new JTextField(15);
                // 输入归还日期文本框
                returnedDateText = new JTextField(15);
```

```
// 输入备注文本框
returnedCommentText = new JTextField(15);
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// hsql 执行语句
String hql = "from BookBrowse where is_returned='否'";
// 执行查询
Query userList = session.createQuery(hql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
for (int index = 0; index < list.size(); index++) {
        BookBrowse bb = (BookBrowse) list.get(index);
        bookNameComboBox.addItem(bb.getBookName());
        bookName2BookBrowse.put(bb.getBookName(), bb);
}
// 事务提交
tx.commit();
// 关闭 session
session.close();
panel1 = new JPanel();
panel1.setLayout(new GridLayout(4, 2));
panel1.add(returnedBookStudentLabel);
panel1.add(returnedBookStudentText);
panel1.add(returnedBookNameLabel);
panel1.add(bookNameComboBox);
```

```
        panel1.add(returnedDateLabel);
        panel1.add(returnedDateText);
        panel1.add(returnedCommentLabel);
        panel1.add(returnedCommentText);
        container.add(panel1, BorderLayout.CENTER);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 3));
        // “清空”按钮
        clearButton = new JButton("清空");
        // “确定”按钮
        yseButton = new JButton("确定");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“清空”按钮添加事件监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加事件监听者
        yseButton.addActionListener(this);
        // 为“取消”按钮添加事件监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yseButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
```

```
public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
                // 单击"退出"按钮不作任何事情
                this.dispose();
        } else if (action.getSource() == clearButton) {
                // 单击"清空"按钮，清空所有的文本框
                returnedBookStudentText.setText("");
                returnedDateText.setText("");
                returnedCommentText.setText("");
        } else if (action.getSource() == yseButton) {
                // 判断还书者的姓名是否为空
                if (returnedBookStudentText.getText().trim().equals("")) {
                        JOptionPane.showMessageDialog(null, "请输入还书者的姓名...");
                } else if (bookNameComboBox.getSelectedItem().equals("")) {
                        // 判断有没有出借的书
                        JOptionPane.showMessageDialog(null, "图书馆没有出借过书！");
                } else {
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil
                                        .getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        String bookName = bookNameComboBox.getSelectedItem().toString();
                        BookBrowse bookBrowse = (BookBrowse) bookName2BookBrowse
                                        .get(bookName);
                        bookBrowse.setIsReturned("是");
                        bookBrowse.setCom(returnedCommentText.getText().trim());
```

```
                    session.saveOrUpdate(bookBrowse);
                    JOptionPane.showMessageDialog(null, "还书成功！");
                    // 事务提交
                    tx.commit();
                    // 关闭 session
                    session.close();
                    this.dispose();
                }
            }
        }
}
```

8.12.2 修改还书信息类的实现

本小节实现修改还书信息类。该类通过 Hibernate 将修改后的还书信息保存到数据库中，其运行效果如图 8-29 所示。

图 8-29 修改书籍还入信息

跟我做

在“library.info”包中创建 ReturnInfo.java 类，在该文件中输入如下内容：

```
package library.info;
/**
 * 修改还书信息类
 *
 * @author lianhw
 *
 */
```

```
public class ReturnInfo extends JFrame implements ActionListener {
	JPanel panel1, panel2;
	Container container;
	JLabel tipLabel = new JLabel("输入还书者姓名和书名点击确定，将调出此书的相关信息");
	JLabel returnedBookStudentLabel, returnedBookNameLabel, returnedDateLabel,
			returnedCommentLabel;
	JTextField returnedBookStudentText, returnedBookNameText, returnedDateText,
			returnedCommentText;
	JButton clearButton, yesButton, updateButton, cancelButton;
	public ReturnInfo() {
		super("修改书籍还入信息");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		container.add(tipLabel, BorderLayout.NORTH);
		// “还书者姓名”标签
		returnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);
		// “书名”标签
		returnedBookNameLabel = new JLabel("书名", JLabel.CENTER);
		// “还书日期”标签
		returnedDateLabel = new JLabel("还书日期", JLabel.CENTER);
		// “备注”标签
		returnedCommentLabel = new JLabel("备注", JLabel.CENTER);
		// 输入还书学生姓名文本框
		returnedBookStudentText = new JTextField(15);
		// 输入还书名称文本框
		returnedBookNameText = new JTextField(15);
		// 输入还书日期文本框
		returnedDateText = new JTextField(15);
		// 输入还书备注文本框
```

```
returnedCommentText = new JTextField(15);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(4, 2));
panel1.add(returnedBookStudentLabel);
panel1.add(returnedBookStudentText);
panel1.add(returnedBookNameLabel);
panel1.add(returnedBookNameText);
panel1.add(returnedDateLabel);
panel1.add(returnedDateText);
panel1.add(returnedCommentLabel);
panel1.add(returnedCommentText);
container.add(panel1, BorderLayout.CENTER);
panel2 = new JPanel();
panel2.setLayout(new GridLayout(1, 4));
// “清空”按钮
clearButton = new JButton("清空");
// “确定”按钮
yesButton = new JButton("确定");
// “更新”按钮
updateButton = new JButton("更新");
// “取消”按钮
cancelButton = new JButton("取消");
// 为“清空”按钮添加监听者
clearButton.addActionListener(this);
// 为“确定”按钮添加监听者
yesButton.addActionListener(this);
// 为“更新”按钮添加监听者
updateButton.addActionListener(this);
updateButton.setEnabled(false);
```

```
        // 为“取消”按钮添加监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(updateButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == clearButton) {
            // 单击“清空”按钮，清空所有的文本框
            returnedBookStudentText.setText("");
            returnedBookNameText.setText("");
            returnedDateText.setText("");
            returnedCommentText.setText("");
        } else if (action.getSource() == cancelButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
```

```
            Transaction tx = session.beginTransaction();
            // hsql 执行语句
            String hql = "from BookBrowse where studentName='"
                        + returnedBookStudentText.getText().trim()
                        + "'and bookName='" + returnedBookNameText.getText().trim()
                        + "'";
            // 执行查询
            Query userList = session.createQuery(hql);
            // 将查询结果放置到一个 list 链表中
            List list = userList.list();
            BookBrowse browse = (BookBrowse) list.get(0);
            returnedBookStudentText.setText(browse.getStudentName());
            returnedBookNameText.setText(browse.getBookName());
            returnedDateText.setText(browse.getBorrowDate().toString());
            returnedCommentText.setText(browse.getCom());
            updateButton.setEnabled(true);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
        } else if (action.getSource() == updateButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            BookBrowse browse2 = new BookBrowse();
            browse2.setBookName(returnedBookNameText.getText().trim());
```

```
                browse2.setStudentName(returnedBookStudentText.getText().trim());
                browse2.setBorrowDate(new GregorianCalendar().getTime());
                browse2.setCom(returnedCommentText.getText().trim());
                browse2.setReturnDate(new GregorianCalendar().getTime());
                session.saveOrUpdate(browse2);
                JOptionPane.showMessageDialog(null, "修改书籍成功！");
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
                this.dispose();
            }
        }
}
```

8.12.3 借阅图书一览类的实现

本小节实现借阅图书一览。该类通过 Hibernate 实现了从数据库中查询所有相关记录的操作，其运行效果如图 8-30 所示。

图 8-30 书籍借阅一览

跟我做

在“library.book”包中创建 BorrowBookList.java 类，在该类中输入如下内容：

```
package library.book;
public class BorrowBookList extends JFrame implements ActionListener {
```

```
Container container;
JPanel panel1, panel2;
JLabel bookNameLabel, studentNameLabel;
JTextField bookNameText, studentNameText;
JButton searchButton, exitButton;
JTable table = null;
DefaultTableModel defaultModel = null;
public BorrowBookList() {
        super("书籍借阅一览！");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        //“书名”标签
        bookNameLabel = new JLabel("书名", JLabel.CENTER);
        //“借阅者”标签
        studentNameLabel = new JLabel("借阅者", JLabel.CENTER);
        //输入“书名”文本框
        bookNameText = new JTextField(15);
        //输入“学生姓名”文本框
        studentNameText = new JTextField(15);
        //“查询”按钮
        searchButton = new JButton("查询");
        //“退出”按钮
        exitButton = new JButton("退出");
        //为“查询”按钮增加监听者
        searchButton.addActionListener(this);
        //为“退出”按钮增加监听者
        exitButton.addActionListener(this);
        Box box1 = Box.createHorizontalBox();
        box1.add(studentNameLabel);
```

```
        box1.add(studentNameText);
        box1.add(searchButton);
        Box box2 = Box.createHorizontalBox();
        box2.add(bookNameLabel);
        box2.add(bookNameText);
        box2.add(exitButton);
        Box boxH = Box.createVerticalBox();
        boxH.add(box1);
        boxH.add(box2);
        boxH.add(Box.createVerticalGlue());
        panel1 = new JPanel();
        panel1.add(boxH);
        panel2 = new JPanel();
        String[] name = { "借阅者", "书名", "借阅日期", "还入日期", "备注" };
        String[][] data = new String[0][0];
        defaultModel = new DefaultTableModel(data, name);
        table = new JTable(defaultModel);
        table.setPreferredScrollableViewportSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table);
        panel2.add(s);
        container.add(panel1, BorderLayout.NORTH);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，查询所有图书的出借情况
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
```

```
if (action.getSource() == exitButton) {
        // 单击"退出"按钮，不作任何事情
        this.dispose();
} else if (action.getSource() == searchButton) {
        String strSQL = " from BookBrowse";
        String strSql = null;
        if (studentNameText.getText().trim().equals("")
                        && bookNameText.getText().trim().equals("")) {
                // 无条件查询
                strSql = strSQL;
        } else if (studentNameText.getText().trim().equals("")) {
                // 按书名查询
                strSql = strSQL + " where bookName='"
                                + bookNameText.getText().trim() + "'";
        } else if (bookNameText.getText().trim().equals("")) {
                // 按学生姓名查询
                strSql = strSQL + " where studentName='"
                                + studentNameText.getText().trim() + "'";
        } else {
                // 按学生姓名和书名查询
                strSql = strSQL + " where studentName='"
                                + studentNameText.getText().trim() + "'and bookName='"
                                + bookNameText.getText().trim() + "'";
        }
        // 首先要删除 table 中的数据：
        int rowCount = defaultModel.getRowCount() - 1;// 取得 table 中的数据行；
        int j = rowCount;
        for (int i = 0; i <= rowCount; i++) {
                defaultModel.removeRow(j);// 删除 rowCount 行的数据；
```

```
                defaultModel.setRowCount(j);// 重新设置行数；
                j = j - 1;
            }
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // 执行查询
            Query userList = session.createQuery(strSql);
            // 将查询结果放置到一个 list 链表中
            List list = userList.list();
            for (int index = 0; index < list.size(); index++) {
                Vector data = new Vector();
                BookBrowse book = (BookBrowse) list.get(index);
                data.addElement(book.getStudentName());
                data.addElement(book.getBookName());
                data.addElement(book.getBorrowDate());
                data.addElement(book.getReturnDate());
                data.addElement(book.getCom());
                defaultModel.addRow(data);
            }
            table.revalidate();
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
        }
```

```
        }
}
```