1. **引言**

**1.1 作品简介**

本项目是基于Java技术开发的学生宿舍管理系统，在设计该系统时，考虑到此系统涉及的用户，本系统运行时的登录用户为学生和宿管人员；系统要求用户输入正确的账号和密码才能进入系统。学生用户进入系统后，可对本宿舍的维修信息、快递信息进行相关操作，也可对个人的晚归记录进行查看，另外，本系统还实现了快件信息的定时提醒功能，用户可以看到自己未领取的新快件；宿管进入系统后，可对宿舍楼的信息、本宿舍楼所有宿舍的具体情况、学生的晚归记录进行相关操作，同时可以查看学生的相关信息。每个用户可对自己的个人信息进行修改，也可以对自己的密码进行设置。

* 1. **参考资料**

1. 结合课本《Java从入门到精通》的数据库知识进行开发
2. 结合<http://www.runoob.com/w3cnote/java-swing-demo-intro.html>

内Java Swing相关知识进行界面的设计

1. 根据<https://www.yiibai.com/jdbc/jdbc_quick_guide.html>进行JDBC的开发
2. 结合了宿舍管理系统的例子进行设计和借鉴：

<https://blog.csdn.net/acm_hmj/article/details/70225137>

1. **项目概述**

**2.1 开发内容**

① 登录界面模块（学生与宿舍管理员两种用户）

② 学生用户部分

1. 学生个人信息模块
2. 密码设置功能模块
3. 报修信息模块；
4. 快件信息模块；
5. 晚归信息模块；
6. 帮助文档介绍模块；
7. 关于模块

③ 宿管用户部分

1. 宿管个人信息模块；
2. 密码设置修改功能模块；
3. 宿舍楼信息模块；
4. 宿舍分配信息模块；
5. 学生信息模块；
6. 夜归信息模块；
7. 帮助文档介绍模块；
8. 关于模块
   1. **性能要求**

由于此系统是以Java技术为核心开发的小项目，所以对于性能方面的要求不是很高。

* 1. **条件与限制**

开发平台：eclipse，Microsoft SQL Server Management Studio 2017

开发语言：Java

JRE：jdk1.8.0\_191

* 1. **运行环境以及配置要求**

本项目所使用的数据库表是使用SQL Server Management Studio 2017进行设计的，要求将数据库表导入此数据库系统管理软件，系统需配置好java运行环境。本系统使用的数据库名为DMS，连接数据库的用户名为sa,连接数据库的密码为root。具体配置流程为：打开SQL Server Management Studio 2017，使用Windows Authentication登录进去后，进行相关的配置。sa用户是sql server自带的用户，所以我们在装数据库的时候只需要设置一个密码就可以了。配置过程可参照<https://jingyan.baidu.com/article/6766299787b69c54d51b84bb.html>进行配置。

* 1. **数据库设计**

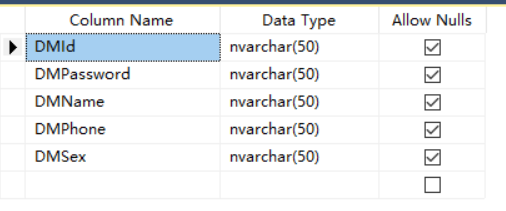
**（一千个字顶不上一张图，直接上图）**

**① 本系统涉及到的数据库表有：**

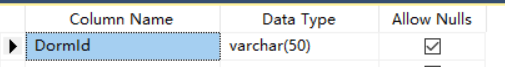


**② 每个表的介绍：**

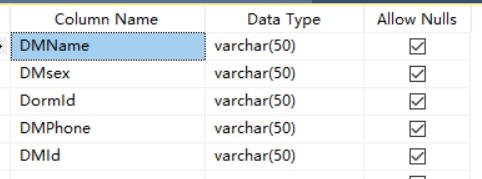
DMAccountPassword表：(用于记录宿管工号密码等信息)



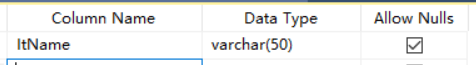
DormitoryInfo表：（记录本校区的宿舍楼号）



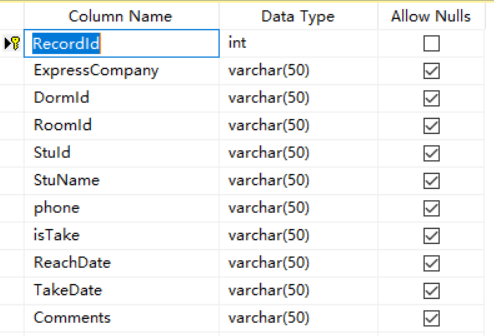
AllDormManagerInfo表：（记录所有宿管的个人信息）



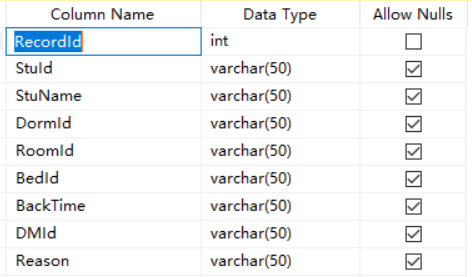
ItemInfo表：（记录宿舍可报修的物品信息）



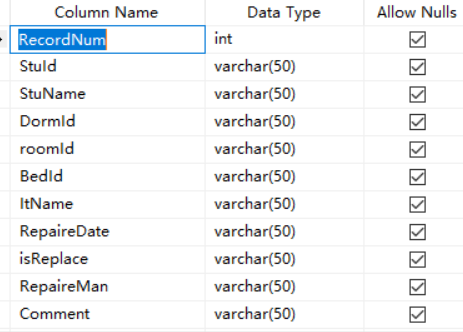
NewExpressInfo表：（记录新快递的相关信息）



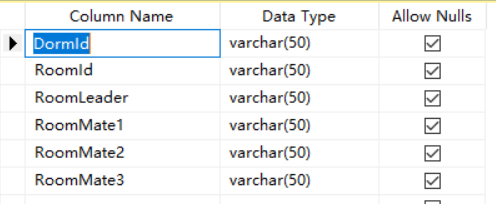
NightBackInfo表：（记录学生晚归的信息）



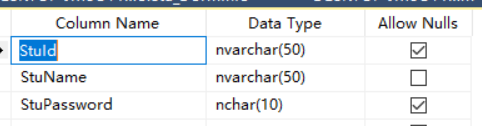
RepairInfo表：（记录学生用户本宿舍内的报修情况信息）



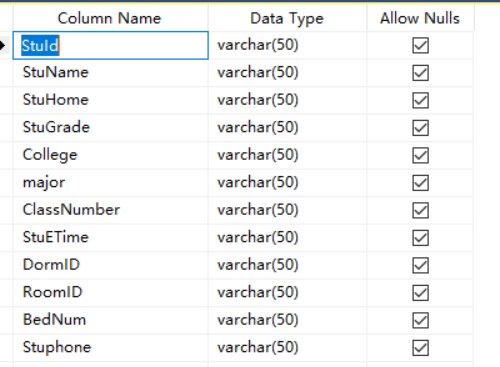
Stu\_DormInfo表：（记录学生的宿舍以及成员信息）



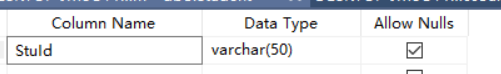
StuAccountPassword表：（记录学生用户的学号和密码等信息）



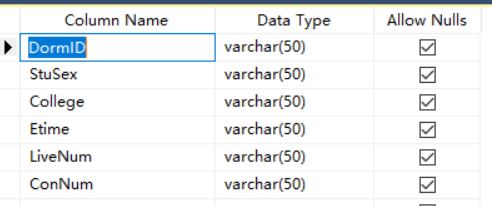
StudentInfo表：（记录学生详细的个人信息）



StuAccountId表：（记录学生用户的学号信息）



DormitoryBuildingInfo表：（记录宿管所管宿舍楼的相关信息）

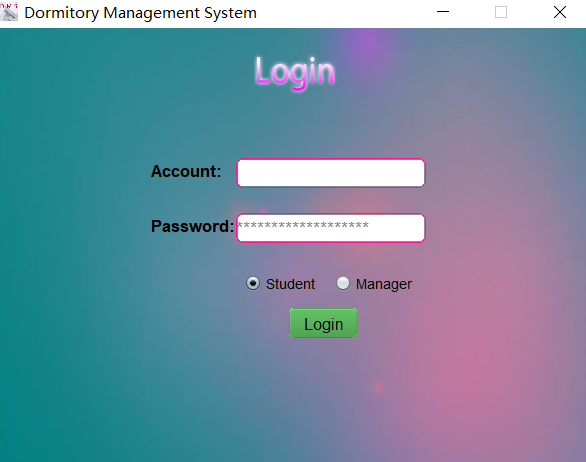


ALLDormRoomInfo表：（记录宿管所管宿舍楼的学生信息）

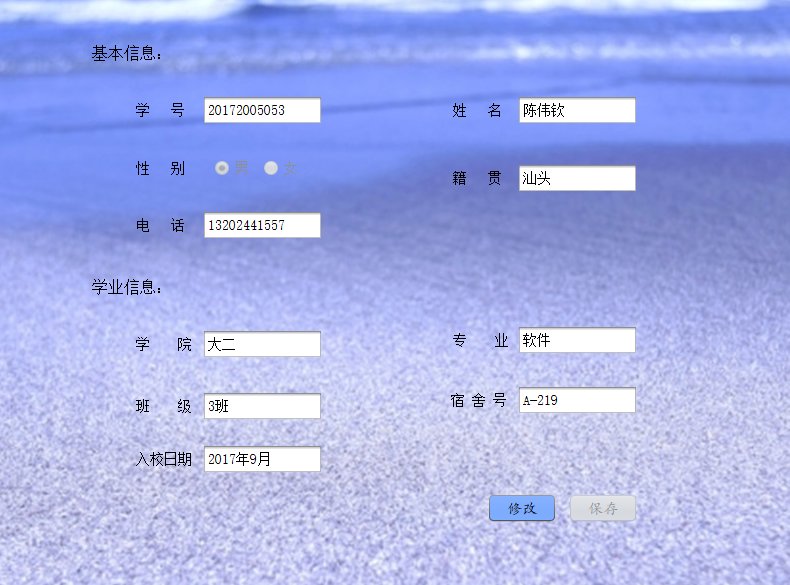


* 1. **模块设计**

1. **登录界面**



1. **学生用户方面的模块设计**
2. **个人信息**



1. **修改密码**



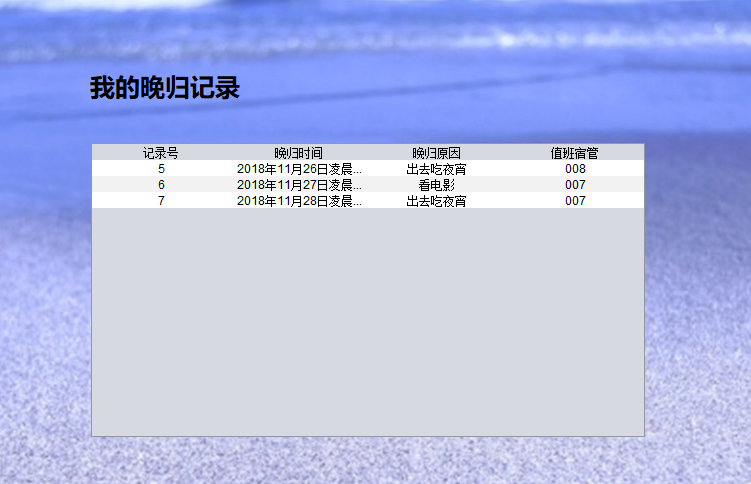
1. **报修信息**



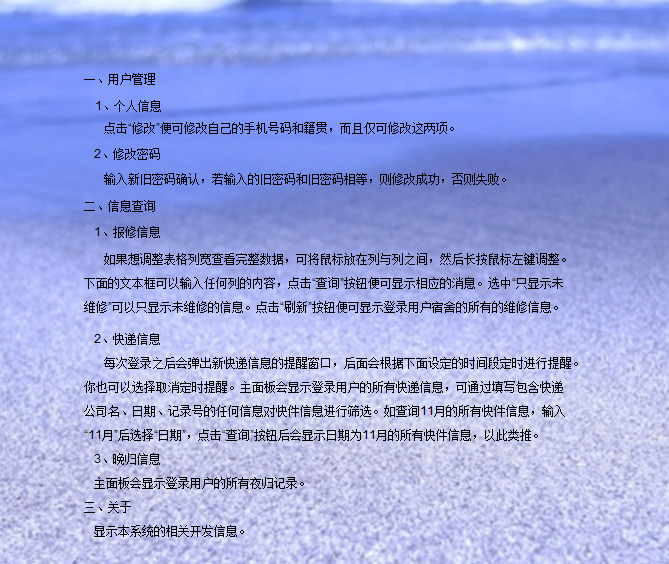
1. **快递信息**



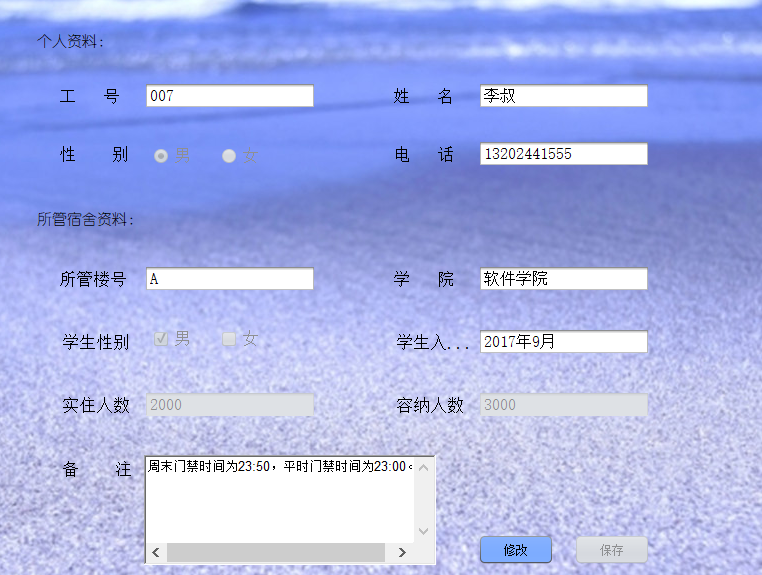
1. **晚归信息**



1. **使用帮助**



1. **宿舍管理员方面的模块设计**
2. **个人信息**



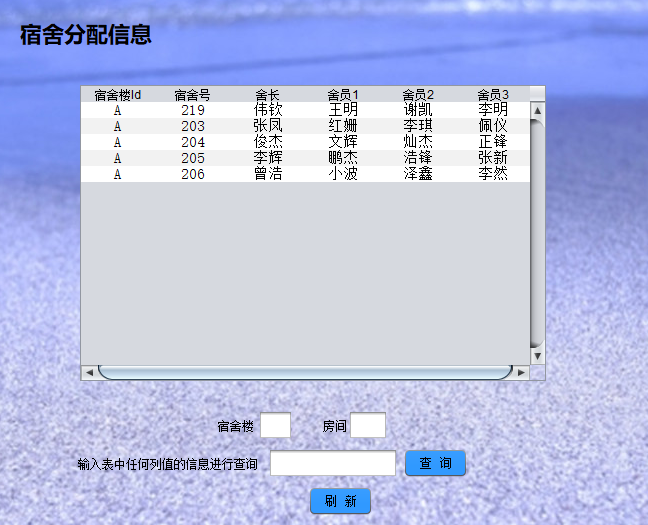
1. **密码修改**



1. **宿舍楼信息**



1. **宿舍分配信息**



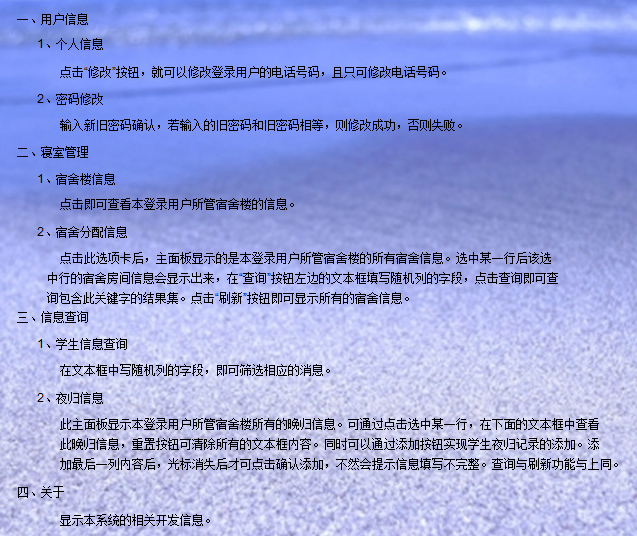
1. **学生信息查询**



1. **学生晚归信息**



1. **使用帮助**



* 1. **核心代码说明**
  2. **ShowInfo方法以及managerService接口**

1. **public** **void** showInfo() **throws** DataBaseException, QueryResultIsNullException {
2. AllDormitoryInfo allAllDormitoryInfo = manager.getAllDormitoryInfo();
3. DefaultTableModel defaultTableModel.setDataVector(allAllDormitoryInfo.
4. getdormitoryInfos(), headvector);
5. }

对于以上第二行的代码，其来源为：

1. **public**  AllDormitoryInfo getAllDormitoryInfo() **throws** DataBaseException,
2. QueryResultIsNullException {
3. AllDormitoryInfo allDormitoryInfo =**null**;
4. allDormitoryInfo = managerService.getAllDormitoryInfo(**this**.getDormId());

managerService为com.dms.people包下实现宿舍管理员服务接口ManagerService的类ManagerServiceImpl的对象，下面列举ManagerService接口定义的部分抽象方法：

1. /\*\*
2. \*
3. \*
4. \* 获取所有学生的信息
5. \* @param dmid
6. \* @return
7. \* @throws QueryResultIsNullException
8. \* @throws DataBaseException
9. \*/
10. **public** Vector<Object> getAllStudentInfo(String dmid)
11. **throws** QueryResultIsNullException, DataBaseException; // 得到所有学生的信息
13. /\*\*
14. \*
15. \*
16. \* @param studentId
17. \* @return
18. \* @throws QueryResultIsNullException
19. \* @throws DataBaseException
20. \*/
21. **public** StudentInfo queryStudentInfoById(String studentId)
22. **throws** QueryResultIsNullException, DataBaseException; // 查询学生信息by学生id
24. **public** **void** queryStudentInfoByName(String studentName); // 查询学生信息by学生Name
26. **public** **void** queryStudentInfoByDormitoryId(String dormitoryId); // 按宿舍查询学生信息
27. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*维修信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//
28. /\*\*
29. \*
30. \*
31. \* @param dormId
32. \* @return
33. \* @throws DataBaseException
34. \* @throws QueryResultIsNullException
35. \*/
36. **public** AllMaintanceRecord getAllMaintenanceRecord(String dormId) **throws**
37. DataBaseException, QueryResultIsNullException; // 得到所有维修信息
39. **public** **void** getFixedInfo();
41. **public** **void** queryMaintenanceInfoByDormitoryId(String DormitoryId);
43. /\*\*
44. \*
45. \*
46. \* 查询舍管所在楼栋宿舍信息
47. \* @param DMId
48. \* @return
49. \* @throws DataBaseException
50. \* @throws QueryResultIsNullException
51. \*/
52. **public** DormitoryBuildingInfo getDormitory(String DMId)
53. **throws** DataBaseException, QueryResultIsNullException;
55. /\*\*
56. \*
57. \*
58. \* 得到本栋楼宿舍信息
59. \* @param string
60. \* @return
61. \* @throws DataBaseException
62. \* @throws QueryResultIsNullException
63. \*/
64. **public** AllDormitoryInfo getAllDormitoryInfo(String DMid) **throws** DataBaseException,
65. QueryResultIsNullException;
67. /\*\*
68. \*
69. \*
70. \* 得到所有快递信息
71. \* @return
72. \* @throws DataBaseException
73. \* @throws QueryResultIsNullException
74. \*/
75. **public** Vector<Object> getAllExpressInfo(String dmid) **throws** DataBaseException,
76. QueryResultIsNullException;
78. **public** **boolean** updateExpressTransceiver(**int** Record, String Data);

此接口是获取相关表信息的重要桥梁，对于getAllStudentInfo方法：

在ManagerServiceImpl的具体实现如下：

1. /\*\*
2. \* 获取所有学生的信息
3. \* @param dmid
4. \* @return
5. \* @throws QueryResultIsNullException
6. \* @throws DataBaseException
7. \*/
8. @Override
9. **public** Vector<Object> getAllStudentInfo(String dmid)
10. **throws** QueryResultIsNullException, DataBaseException {
11. Statement stmt = DB.CreateStatement();
12. String sql = "select \* from DMS..ALLDormRoomInfo where DormId = '" + dmid + "'";
13. **try** {
14. ResultSet rs = stmt.executeQuery(sql);
15. **if**(rs!=**null**&&!rs.next()){
16. **throw** **new** QueryResultIsNullException();
17. }
18. Vector<Object> vectors = **new** Vector<Object>();
19. Vector<Object> vector = **new** Vector<Object>();
20. **for**(**int** i=1;i<=9;i++){
21. vector.add(rs.getObject(i));
22. }
23. vectors.add(vector);
24. **while**(rs.next()){
25. vector = **new** Vector<Object>();
26. **for**(**int** i=1;i<=9;i++){
27. vector.add(rs.getObject(i));
28. }
29. vectors.add(vector);
30. }
31. **return** vectors;
32. } **catch** (SQLException e) {
33. e.printStackTrace();
34. **throw** **new** DataBaseException();
35. }
37. }

查询ALLDormRoomInfo 表中DormId为此登录宿舍管理员工号的所有记录，使用Vector数组vectors存储所有的行集，使用数组vector存储每一行，每存储一行就将其存入vectors数组，最后将行集数组vectors返回给调用者使用，该方法调用者为com.dms.managerui.DormitoryBuildingInfo.java中的showInfo方法，下面给出具体实现：

1. **public** **void** showInfo() **throws** DataBaseException, QueryResultIsNullException{
3. Vector<Object> vector = manager.getAllStudentInfo(manager.getDormId());
4. **for**(**int** i=0;i<vector.size();i++)
5. defaultTableModel.setDataVector(vector, headvector);
6. }

其中defaultTableModel为DefaultTableModel类的对象，此模型结合JTable使用可以显示数据库表的内容，模型的数据通过setDataVector方法来实现，该方法有两个参数，第一个为数据行集（用Vector存储），第二个为表头数据集：

1. //头部信息
2. headvector = **new** Vector<Object>();
3. headvector.add("学号");
4. headvector.add("姓名");
5. headvector.add("性别");
6. headvector.add("入学时间");
7. headvector.add("学院");
8. headvector.add("专业");
9. headvector.add("班级");
10. headvector.add("籍贯");
11. headvector.add("联系电话");

ShowInfo方法在点击确认添加或者刷新之后进行使用,起到重新查询刷新数据的作用：

1. confirmadd=**new** JButton("确认添加");
2. confirmadd.addActionListener(**new** ActionListener() {
4. @Override
5. **public** **void** actionPerformed(ActionEvent e) {
6. // TODO Auto-generated method stub
7. **if**(input()) {
8. currentRows=table.getRowCount()-1;
9. currentColumns=table.getColumnCount();
10. confirmadd.setEnabled(**false**);
11. add.setEnabled(**true**);
12. String sql="INSERT INTO DMS..NightBackInfo(StuId,StuName,DormId,RoomId,BedId,BackTime,DMId,Reason) VALUES('"+stuid+"','"+stuname+"','"+dormid+"','"+roomid+"','"+bedid+"','"+backtime+"','"+dmid+"','"+reason+"')";
13. //                               System.out.println(sql);   //test
14. PreparedStatement stmt = DB.CreatePrepare(sql);
15. **try** {
16. stmt.execute();
17. JOptionPane.showMessageDialog(**null**, "添加成功",
18. "信息提示", JOptionPane.WARNING\_MESSAGE);
19. add.setEnabled(**true**);
20. confirmadd.setEnabled(**false**);
21. **try** {
22. showInfo();
23. } **catch** (DataBaseException | QueryResultIsNullException e1) {
24. // TODO Auto-generated catch block
25. e1.printStackTrace();
26. }
27. } **catch** (SQLException e1) {
28. // TODO Auto-generated catch block
29. JOptionPane.showMessageDialog(**null**, "添加失败",
30. "信息提示", JOptionPane.ERROR\_MESSAGE);
31. e1.printStackTrace();
32. }
33. }
34. }
35. });
    1. **数据的“增删查改”**

在添加数据方面，此处使用了“添加”和“确认添加”两个按钮对添加过程进行控制。对于“添加”，点击之后模型会出现一行数据，新的一行可能有某些列值已有初始值，某些为了获取前一行的id值实现计算新行的id，需要计算当前模型的总行数，下面给出“添加”部分的实例代码：

1. add.addActionListener(**new** ActionListener() {
3. @Override
4. **public** **void** actionPerformed(ActionEvent e) {
5. // TODO Auto-generated method stub
6. Vector newRow;
7. **int** selectrow=table.getSelectedRow();
9. stuid="";stuname="";dormid="";roomid="";bedid="";backtime="";reason="";
10. add.setEnabled(**false**);
11. confirmadd.setEnabled(**true**);
12. currentRows=table.getRowCount();
13. currentColumns=table.getColumnCount();
14. dormid = manager.getDormId();
15. dmid = manager.getId();
16. newRow=**new** Vector();
17. **for**(**int** i=0;i<currentColumns;i++)   //小于当前列数
18. {
19. **if**(i==0) {
20. recordid=Integer.parseInt(table.getValueAt(currentRows-1, 0).toString());
21. //                         System.out.println(recordId);
22. newRow.add(i, recordid+1);
23. }
24. **else** **if**(i==3||i==7) {
25. **if**(i==3)
26. newRow.add(i,dormid);
27. **else** **if**(i==7)
28. newRow.add(i,dmid);
29. }
30. **else**
31. newRow.add("");
32. }
33. ((DefaultTableModel) table.getModel()).addRow(newRow);
34. }
35. });

在“确认添加”方面：

实现了先检查后添加的模式，通过input方法对用户输入的数据进行反馈，判断用户输入的数据是否完整或正确，并给出相应的错误提示，下面给出实例代码：

1. //检查数据
2. **private** **boolean** input() {
3. currentRows=table.getRowCount()-1;
4. **if**(stuid.equals("")||stuname.equals("")||dormid.equals("")||roomid.equals("")||
5. backtime.equals("")||reason.equals("")) {
6. JOptionPane.showMessageDialog(**null**, "请输入完整数据",
7. "输入信息不完整", JOptionPane.WARNING\_MESSAGE);
8. **return** **false**;
9. }
10. **return** **true**;
11. }
12. //按钮监听
13. confirmadd.addActionListener(**new** ActionListener() {
15. @Override
16. **public** **void** actionPerformed(ActionEvent e) {
17. // TODO Auto-generated method stub
18. **if**(input()) {
19. currentRows=table.getRowCount()-1;
20. currentColumns=table.getColumnCount();
21. confirmadd.setEnabled(**false**);
22. add.setEnabled(**true**);
23. String sql="INSERT INTO DMS..NightBackInfo(StuId,StuName,DormId,RoomId,BedId,BackTime,DMId,Reason) VALUES('"+stuid+"','"+stuname+"','"+dormid+"','"+roomid+"','"+bedid+"','"+backtime+"','"+dmid+"','"+reason+"')";
24. //                              System.out.println(sql);   //test
25. PreparedStatement stmt = DB.CreatePrepare(sql);
26. **try** {
27. stmt.execute();
28. JOptionPane.showMessageDialog(**null**, "添加成功",
29. "信息提示", JOptionPane.WARNING\_MESSAGE);
30. add.setEnabled(**true**);
31. confirmadd.setEnabled(**false**);
32. **try** {
33. showInfo();
34. } **catch** (DataBaseException | QueryResultIsNullException e1) {
35. // TODO Auto-generated catch block
36. e1.printStackTrace();
37. }
38. } **catch** (SQLException e1) {
39. // TODO Auto-generated catch block
40. JOptionPane.showMessageDialog(**null**, "添加失败",
41. "信息提示", JOptionPane.ERROR\_MESSAGE);
42. e1.printStackTrace();
43. }
44. }
45. }
46. });

对于添加入数据库的正确数据，通过监听模型（如addTableModelListener）以达到数据更新的目的。

在查询功能方面，使用了setRowFilter方法进行正则式的模糊筛选：

1. String text = tfFind.getText();
2. **if**(text.trim().length()==0)
3. sorter.setRowFilter(RowFilter.regexFilter(""));
4. **else**
5. sorter.setRowFilter(RowFilter.regexFilter(text));

在删除方面，为了不删除掉其他未被选中的行，一般根据能唯一标识行数据的列值进行行的删除，示例代码如下：

1. delete=**new** JButton("删除");
2. delete.addActionListener(**new** ActionListener() {
4. @Override
5. **public** **void** actionPerformed(ActionEvent e) {
6. // TODO Auto-generated method stub
7. **int** selectrow=table.getSelectedRow();
8. **int** rowcount = defaultTableModel.getRowCount() - 1;
9. **if**(selectrow>=0)
10. recordid=Integer.parseInt(table.getValueAt(selectrow, 0).toString());
11. //                  System.out.println("id="+id);
12. **if**(selectrow<0)
13. JOptionPane.showMessageDialog(**null**, "请选择一行进行删除",
14. "操作错误", JOptionPane.ERROR\_MESSAGE);
15. **if**(rowcount>=0)
16. {
17. defaultTableModel.removeRow(selectrow);
18. defaultTableModel.setRowCount(rowcount);
19. String sql="delete from DMS..NightBackInfo where RecordId="+recordid;
20. PreparedStatement stmt=DB.CreatePrepare(sql);
21. **try** {
22. stmt.execute();
23. JOptionPane.showMessageDialog(**null**, "删除成功" ,
24. "提示信息" , JOptionPane.WARNING\_MESSAGE);
25. } **catch** (SQLException e1) {
26. // TODO Auto-generated catch block
27. e1.printStackTrace();
28. }
29. sorter.setRowFilter(RowFilter.regexFilter(""));
30. }
31. **else** {
32. JOptionPane.showMessageDialog(**null**, "删除失败",
33. "失败信息", JOptionPane.WARNING\_MESSAGE);
34. }
35. }
36. });

数据更新应用于设置密码方面和个人信息修改等模块上，如定义在com.dms.people.Manager类的方法：

1. @Override
2. **public** **void** PasswordUpdate(UpdatePasswordInfo updatePasswordInfo) **throws** PasswordNotMatchException, DataBaseException, UpdateSuccessException {
3. updatePasswordInfo.setOwnerId(**this**.getId());
4. managerService.PasswordUpdate(updatePasswordInfo);
5. }
7. @Override
8. **public** **void** UpdatePersonalInfo(UpdatePersonInfo updatePersonInfo) **throws** UpdateDataException, DataBaseException {
9. updatePersonInfo.setOwnerId(getId());
10. managerService.UpdatePersonalInfo(updatePersonInfo);
11. **try** {
12. initPersonalInfo();  //初始化个人信息
13. } **catch** (QueryResultIsNullException e) {
14. // TODO 自动生成的 catch 块
15. e.printStackTrace();
16. }
17. }

其中，UpdatePasswordInfo 是用于处理（setter,getter等）新旧密码的类。

UpdatePersonalInfo是定义于ManagerService接口中的一个方法，在ManagerServiceImpl中给出具体实现，达到对数据库表内容进行更新的目的。

1. /\*\*
2. \* 更新个人信息
3. \*
4. \* @param updatePersonInfo
5. \* @throws UpdateDataException
6. \* @throws DataBaseException
7. \*/
8. @Override
9. **public** **void** UpdatePersonalInfo(UpdatePersonInfo updatePersonInfo)
10. **throws** UpdateDataException, DataBaseException {
11. Connection connection = DB.getConnection();
12. String sql = "update DMS..AllDormManagerInfo set DMPhone=? where
13. DMId='";
14. + updatePersonInfo.getOwnerId()+"'";
15. //  System.out.println(sql);
16. **try** {
17. PreparedStatement pstmt = connection.prepareStatement(sql);
18. pstmt.setString(1, updatePersonInfo.getPhone());
19. pstmt.execute();
21. } **catch** (SQLException e) {
22. e.printStackTrace();
23. **throw** **new** UpdateDataException();
25. }
26. }
    1. **数据库连接**

在com.dms.mydatabase包下的DB.java类定义了数据库的相关操作，实现了数据库连接操作使用的Connection对象、Statement对象、PreparedStatement对象以及数据库驱动加载、数据库关闭的统一管理。详细代码可见项目源码。

* 1. **线程的使用：**

对于新快递信息的弹出窗，结合了线程类进行实现：

在com.dms.expressmsg包有ExpressMassage.java和ExpressMassageRemind.java两个主类，其中ExpressMassage继承JFrame，其对象表示快递信息弹出窗口，ExpressMassageRemind实现Runnable接口为线程类，其方法newExpressMassageRemind构造并返回ExpressMassage对象，并在关闭系统监听处被销毁。下面给出相关代码说明：

1. **public** ExpressMassage newExpressMassageRemind(Student stu){
2. expressMassage = **new** ExpressMassage(stu);
3. **return** expressMassage;
4. }
5. @Override
6. **public** **void** run() {
7. **while** (**true**) {
8. **if**(Thread.currentThread().isInterrupted()) {
9. System.out.println("线程退出");
10. **break**;
11. }
12. **try** {
13. Thread.sleep(expressMassage.getRemindTime()-100000);
14. **if**(expressMassage!=**null**){
15. expressMassage.showInfo();
16. expressMassage.setVisible(**true**);
17. Thread.sleep(1000000);
18. }
20. } **catch** (InterruptedException e) {
21. // TODO 自动生成的 catch 块
22. e.printStackTrace();
23. } **catch** (DataBaseException e) {
24. // TODO 自动生成的 catch 块
25. e.printStackTrace();
27. } **catch** (QueryResultIsNullException e) {
28. // TODO 自动生成的 catch 块
29. e.printStackTrace();
30. }**finally** {
31. Thread.yield();
32. }
33. }
34. }
35. **总结**
    * 1. 在实现此系统之前，对系统有初步的框架意识是很重要的，它关系到数据库表的设计以及系统模块的设计等，有了比较清晰的框架认识，就可使开发有条不紊地进行下去，加快开发进度。
      2. 善于利用网上的资源。如在实现数据插入操作时为了判断插入是否成功，一开始使用execute方法进行语句的执行，但其返回值永远为false，而数据已成功插入数据库表中，通过不懈的baidu，终于发现了问题所在，原来execute方法返回时true表示执行的是查询语句，false表示执行的是insert,delete,update等等。 executeUpdate返回的是int，表示有多少条数据受到了影响。此问题的解决加快了项目的进度，也同时让我获得更多关于JDBC编程的细节。
      3. 要善于调试。由于本系统的开发涉及到较多的数据库查询语句，为了更快的找出bug，一般会在查询语句执行之前将查询语句输出到Console，若查询不到结果，则将此代码行放到SQL Server中进行测试，确定是语句的问题还是数据库执行代码的问题。本项目同时也通过自定义了一些异常类，并在查询语句之处捕获，捕获到的异常信息对问题的解决能提供更加导向性的作用。
      4. 注意备份。无论处于开发哪个阶段，成果无论大小均需要重视备份，在备份目录下应注明开发存在的问题和实现的进度，便于下次的开发。