# 管家婆家庭记账软件

## 项目介绍

### 项目安排

#### 项目目标

本项目为JAVAEE基础班综合项目，包含了若干个知识点，达到将基础班所学知识综合使用，提高了我们对项目的理解与知识点的运用。

掌握View层、Controller层、Service层、Dao层等之间的方法相互调用操作、

掌握dbutils操作数据库表完成增删改查

掌握事件监听处理

通过本项目，了解公司项目开发的流程，充分的掌握项目需求分析、设计与功能的代码实现。提高同学们独立分析需求与功能实现的能力。

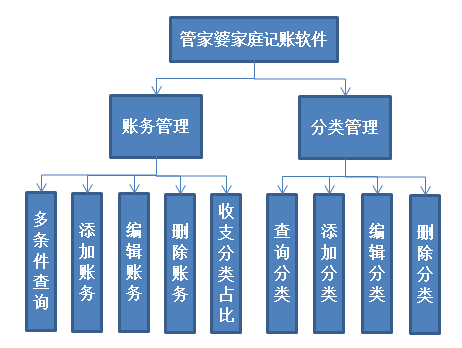
#### 项目时间

本项目通过5天课程时间，边讲边练的方式完成。

### 项目功能分析

#### 项目演示

#### 项目各模块功能介绍



* 分类模块
* 显示所有分类
* 添加分类
* 编辑分类
* 删除分类
* 账务模块
* 多条件组合查询账务
* 添加账务
* 编辑账务
* 删除账务
* 收入分类占比图，支出分类占比图（饼形图）

## 项目环境搭建

### 技术选型和jar包介绍

每个项目都要使用一些已经成熟的技术，它们通常是由一些专业组织或团队所提供的开源免费技术。在今后的学习过程中，学员会逐渐对这些专业组织有所了解。本项目中使用的技术如下：

* jfreechart-1.0.9.jar：用来绘制图形的框架，例如：柱状图、饼形图等；
* jcommon-1.0.12.jar：jfreechart依赖该jar包；
* apache的commons组件：
* commons-beanutils-1.8.3.jar：封装并简化了反射；
* commons-dbutils-1.4.jar：封装并简化了JDBC；
* commons-dbcp-1.4.jar：apache commons提供的数据库连接池组件，命名为DBCP；
* commons.pool-1.3.jar：DBCP连接池依赖该jar包；
* commons-logging-1.1.1.jar：beanutils依赖该jar包；
* mysql-connector-java-5.1.28-bin.jar：MySQL的JDBC驱动包，用JDBC连接MySQL数据库必须使用该JAR包。

### 工具类介绍

每个项目都会用到一些工具类，本项目也不例外。我们不会教大家如何实现这些类，而是让大家知道在项目我们如何使用这些工具类，下面是本项目中所使用的工具类以及介绍：

* DateChooser.java：用来显示日期选择框，属于对GUI的扩展；
* DateUtils.java：字符串与日期类之间的转换类；
* GUITools.java：该类可以把窗口居中显示；
* JFreeChartUtils：用来简化JFreeChart的使用；
* ListTableModel：用来简化JTable组件的使用；
* JDBCUtils：用来创建数据库连接池对象；

在项目的实现过程中，我们会说明工具类的使用，这里就不再过多介绍。

### 数据表创建

对一个项目而言，表设计是非常重要的，因为应用程序中所有的操作都是基于数据库表而进行的，所以我们第一步就是创建数据库表。

项目分层后，每层之间的关系是，上层依赖下层，表现层（view）依赖控制层（controller），控制层依赖业务层（service），业务层依赖数据层（dao），而数据层依赖数据库（DB）。

管家婆项目的数据库设计很简单，学员只需找到gjp.sql文件，然后执行之即可。下面是创建库及表的SQL语句：

#### 创建数据库

创建管家婆数据库 gjp

|  |
| --- |
| CREATE DATABASE gjp; |

#### 创建数据库表

创建分类表 gjp\_sort

|  |
| --- |
| CREATE TABLE gjp\_sort(  sid INT PRIMARY KEY AUTO\_INCREMENT, -- 分类ID  sname VARCHAR(100), -- 分类名称  parent VARCHAR(100), -- 所属父分类  sdesc VARCHAR(10000) -- 描述  ); |

创建账务表 gjp\_ledger

|  |
| --- |
| CREATE TABLE gjp\_ledger(  lid INT PRIMARY KEY AUTO\_INCREMENT, -- 账务ID  parent VARCHAR(100), -- 所属父分类  money DOUBLE, -- 金额  sid INT, -- 分类ID  account VARCHAR(100), -- 账户  createtime DATE, -- 创建时间  ldesc VARCHAR(1000) -- 描述  ); |

#### 添加表数据

添加分类表数据

|  |
| --- |
| insert into  gjp\_sort(sid,sname,parent,sdesc)  values (1,'服装支出','支出','买衣服'),  (2,'吃饭支出','支出',''),  (3,'交通支出','支出',''),  (4,'住房支出','支出',''),  (5,'工资收入','收入','fda'),  (6,'股票收入','收入',''),  (7,'礼金支出','支出',''),  (8,'其它支出','支出',''); |

添加账务表数据

|  |
| --- |
| insert into  gjp\_ledger(lid,parent,money,sid,account,createtime,ldesc)  values (1,'支出',247,2,'交通银行','2016-03-02','家庭聚餐'),  (2,'收入',12345,5,'现金','2016-03-15','开工资了'),  (3,'支出',1998,1,'现金','2016-04-02','买衣服'),  (4,'支出',325,2,'现金','2016-06-18','朋友聚餐'),  (10,'收入',8000,6,'工商银行','2016-10-28','股票大涨'),  (11,'收入',5000,6,'工商银行','2016-10-28','股票又大涨'),  (12,'收入',5000,5,'交通银行','2016-10-28','又开工资了'),  (13,'支出',5000,7,'现金','2016-10-28','朋友结婚'),  (14,'支出',1560,8,'现金','2016-10-29','丢钱了'),  (15,'支出',2300,3,'交通银行','2016-10-29','油价还在涨啊'),  (16,'支出',1000,2,'工商银行','2016-10-29','又吃饭'),  (17,'收入',1000,5,'现金','2016-10-30','开资'),  (18,'支出',2000,3,'现金','2016-10-30','机票好贵'),  (19,'收入',5000,5,'现金','2016-10-30','又开资'); |

### 工程创建及包管理

1. 使用Eclipse创建Java工程，命名为gjp
2. 创建工程包
   * cn.itcast.gjp.app: 存放main方法类；
   * cn.itcast.gjp.tool: 存放工具类；
   * cn.itcast.gjp.domain: 存放实体类；
   * cn.itcast.gjp.view: 存放GUI界面，及表现层类；
   * cn.itcast.gjp.controller: 存放控制层类，本项目中该包下的类都会继承view包下的类；
   * cn.itcast.gjp.service: 存放业务层类；
   * cn.itcast.gjp.dao: 存放数据层类
3. 创建lib文件夹，用来存储使用的jar包

## UI模块

### UI介绍

　　本项目中的GUI部分已经完成，学员无需编写。因为GUI只在基础班中使用，而在后期就不再使用，所以我们不会去浪费大量时间去编写复杂的GUI界面。

已为大家提供了6个GUI抽象类，我们只需要去继承这些抽象类就可以得到本项目中所有GUI界面了。你可能会问，为什么都是抽象类呢？原因是每个界面中都包含了按钮，点击按钮所要做的事情在抽象类中并没有提供，所以我们需要编写子类来继承这些抽象类，然后在子类中实现父类中的抽象方法，那么在子类中按钮就有功能了。

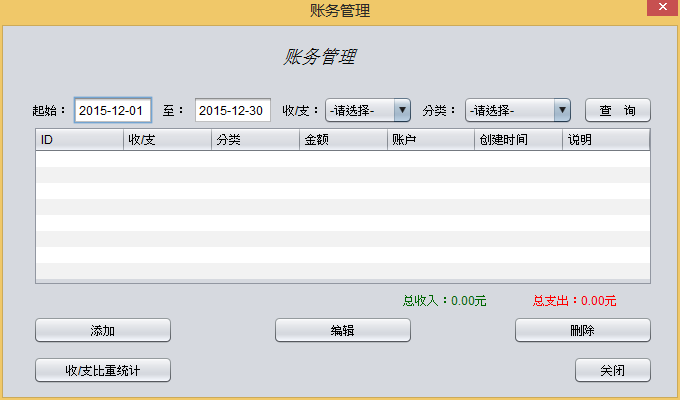
* MainFrameContrller（主界面）：通过继承AbstractMainFrame得到



* SortMngController（分类管理对话框）：通过继承AbstractSortMngDialog得到



* LedgerMngController（账务管理对话框）：通过继承AbstractLedgerMngDialog得到



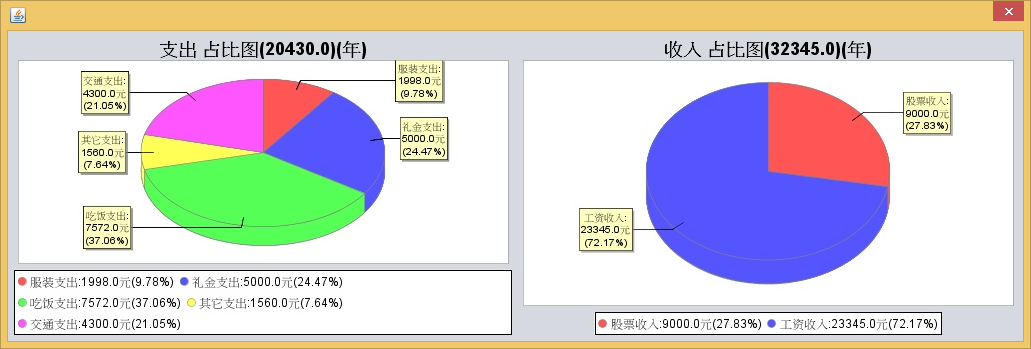
* AddSortController（添加分类对话框）和EditSortController（编辑分类对话框）：通过继承AbstractOperationSortDialog得到



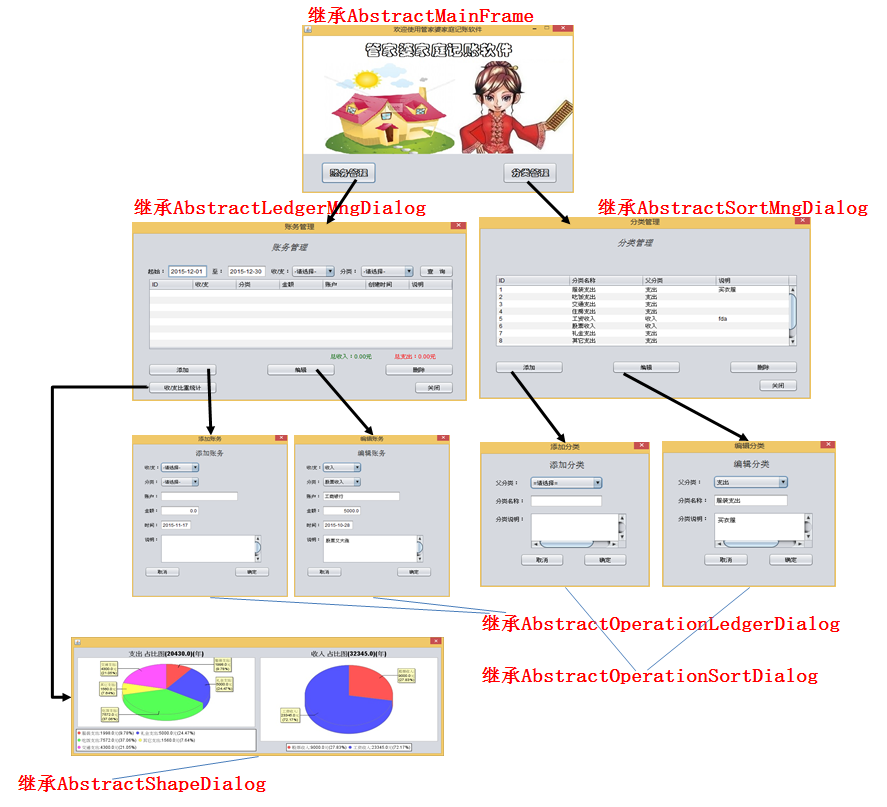
* AddLedgerController（添加账务对话框）和EditLedgerController（编辑账务对话框）：通过继承AbstractOperationLedgerDialog得到



* PieController（年收支占比图【饼形图】）：通过继承AbstractShapeDialog得到



**对话框关系图**



### 主界面

#### 主界面相关类创建

1. 复制已编写好的GUI主界面类AbstractMainFrame.java到view包中；
2. 复制已编写好的工具类GUIToos.java 到 tools包中；
3. 在controller控制层，创建类MainFrameController.java，继承AbstractMainFrame用来创建程序主窗口；
4. 在app包中，创建类MainApp.java,编写main主方法；
5. 复制图片gjp.jpg 到项目根目录下

#### 弹出主界面

1. 在类MainApp的主方法中，创建主窗口控制层类（MainFrameController）对象；
2. 通过setVisible（）方法显示主窗口

#### 主界面相关类详解

GUI主界面类（AbstractMainFrame.java）

|  |
| --- |
| /\*\*  \* 主窗口类  \*/  **public** **abstract** **class** AbstractMainFrame **extends** JFrame {  **static** {  **try** {  javax.swing.UIManager.*setLookAndFeel*("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");  } **catch** (Exception e) {  e.printStackTrace();  }  }  /\*  \* 组件  \*/  **private** JLabel titleLabel = **new** JLabel(**new** ImageIcon("gjp.jpg"));//标题  **private** JButton ledgerBtn = **new** JButton("账务管理");//账务管理  **private** JButton sortBtn = **new** JButton("分类管理");//分类管理    **public** AbstractMainFrame() {  **this**.init();// 初始化操作  **this**.addComponent();// 添加组件  **this**.addListener();// 添加监听器  }    // 初始化操作  **private** **void** init() {  **this**.setTitle("欢迎使用管家婆家庭记账软件");// 标题  **this**.setSize(600, 400);// 窗体大小与位置  GUITools.*center*(**this**);//设置窗口在屏幕上的位置  **this**.setResizable(**false**);// 窗体大小固定  **this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);// 关闭窗口默认操作  }      // 添加组件  **private** **void** addComponent() {  **this**.add(**this**.titleLabel, BorderLayout.***NORTH***);  JPanel btnPanel = **new** JPanel();  btnPanel.setLayout(**null**);  **this**.add(btnPanel);    ledgerBtn.setBounds(40, 20, 120, 50);  sortBtn.setBounds(440, 20, 120, 50);    Font font = **new** Font("华文彩云", Font.***BOLD***, 20);    ledgerBtn.setFont(font);  sortBtn.setFont(font);    btnPanel.add(ledgerBtn);  btnPanel.add(sortBtn);  }  // 添加监听器  **private** **void** addListener() {  ledgerBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent e) {  ledgerMng();  }  });  sortBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent e) {  sortMng();  }  });  }    **public** **abstract** **void** ledgerMng();  **public** **abstract** **void** sortMng();  } |

工具类GUIToos.java

|  |
| --- |
| /\*\*  \* UI界面工具类  \*/  **public** **class** GUITools {  **public** **static** **void** center(Component c) {  Toolkit kit = Toolkit.*getDefaultToolkit*();  **int** x = (kit.getScreenSize().width - c.getWidth()) / 2;  **int** y = (kit.getScreenSize().height - c.getHeight()) / 2;  c.setLocation(x, y);  }  } |

主窗口控制层类（MainFrameController.java）

|  |
| --- |
| /\*\*  \* 主窗口控制类  \*/  **public** **class** MainFrameController **extends** AbstractMainFrame {  @Override  **public** **void** ledgerMng() {  }  @Override  **public** **void** sortMng() {  }  } |

### 分类管理对话框

#### 分类管理对话框相关类创建

1. 复制已编写好的GUI分类管理界面类AbstractSortMngDialog.java到view包中；
2. 在domain包中，创建实体类 Sort.java，类中成员变量参照数据库表gjp\_sort的字段，要求名称一致;
3. 复制已编写好的工具类ListTableModel.java 到 tools包中；
4. 复制jar包commons-beanutils-1.8.3.jar、commons-logging-1.1.1.jar，到lib文件夹中，通过Build Path操作，添加到classPath路径中，提供给ListTableModel使用;
5. 在controller控制层，创建类SortMngController.java，继承AbstractSortMngDialog用来创建分类管理窗口

#### 弹出分类管理对话框

1. 在MainFrameController#sortMng()方法中，创建分类管理对话框（SortMngController）对象；
2. 通过setVisible（）方法显示主窗口

#### 分类管理对话框相关类详解

GUI分类管理对话框类（AbstractSortMngDialog.java）

|  |
| --- |
| /\*\*  \* 分类管理窗口  \*/  **public** **abstract** **class** AbstractSortMngDialog **extends** JDialog {  **protected** JTable sortDataTable = **new** JTable();//账户数据列表  **private** JButton closeBtn = **new** JButton("关闭");  **private** JButton addBtn = **new** JButton("添加");  **private** JButton editBtn = **new** JButton("编辑");  **private** JButton delBtn = **new** JButton("删除");    **public** AbstractSortMngDialog(JFrame frame) {  **super**(frame, **true**);  **this**.initDialog();  }    **protected** **void** initDialog() {  **this**.init();  **this**.addComponent();  **this**.addListener();  }    **private** **void** init() {  **this**.setResizable(**false**);// 设置窗体大小不可变  **this**.setTitle("分类管理：");// 设置标题  **this**.setSize(680, 400);// 设置大小  GUITools.*center*(**this**);//设置居中  **this**.setLayout(**null**);  **this**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);// 设置关闭按钮  }      **private** **void** addComponent() {  // 设置标签标题  JLabel titleLable = **new** JLabel();  titleLable.setFont(**new** Font("宋体", Font.***ITALIC***, 18));  titleLable.setText("分类管理");  titleLable.setBounds(280, 20, 165, 20);  **this**.add(titleLable);  // 滚动面板  JScrollPane scrollPane = **new** JScrollPane();  scrollPane.setBounds(30, 100, 620, 160);    sortDataTable.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);//单选  sortDataTable.getTableHeader().setReorderingAllowed(**false**);//列不能移动  scrollPane.setViewportView(sortDataTable);  **this**.add(scrollPane);    // 按钮  addBtn.setBounds(30, 290, 140, 28);  **this**.add(addBtn);  editBtn.setBounds(270, 290, 140, 28);  **this**.add(editBtn);  delBtn.setBounds(510, 290, 140, 28);  **this**.add(delBtn);  // 关闭按钮  closeBtn.setBounds(570, 330, 80, 28);  **this**.add(closeBtn);  }  /\*\*  \* 显示账务表格  \*/  **protected** **void** setTableModel(List<Sort> sortList) {  String[] colNames = **new** String[] {"ID", "分类名称", "父分类", "说明"};  String[] propNames = **new** String[] {"sid", "sname", "parent", "sdesc"};  **if**(sortList == **null** || sortList.size() == 0) {  sortDataTable.setModel(**new** DefaultTableModel(**new** Object[][] {  {**null**, **null**, **null**, **null**}, {**null**, **null**, **null**, **null**},  {**null**, **null**, **null**, **null**}, {**null**, **null**, **null**, **null**},  {**null**, **null**, **null**, **null**}, {**null**, **null**, **null**, **null**},  {**null**, **null**, **null**, **null**}, {**null**, **null**, **null**, **null**}  },colNames));  sortDataTable.setEnabled(**false**);  **return**;  }  **try** {  sortDataTable.setModel(**new** ListTableModel<Sort>(sortList, Sort.**class**, colNames, propNames));  sortDataTable.setEnabled(**true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }    **protected** Sort getSortByTableRow(**int** row) {  **return** ((ListTableModel<Sort>)sortDataTable.getModel()).getInstance(row);  }  /\*\*  \* 给组件添加监听器  \*/  **private** **void** addListener() {  closeBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  AbstractSortMngDialog.**this**.dispose();  }  });    addBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  addSort();  }  });  editBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  editSort();  }  });  delBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  deleteSort();  }  });  sortDataTable.addMouseListener(**new** MouseAdapter() {  **public** **void** mouseClicked(MouseEvent e) {  **if**(e.getButton() == 1) {  **if**(e.getClickCount() >= 2) {  editSort();  }  }  }  });  }    **public** **abstract** **void** addSort();  **public** **abstract** **void** editSort();  **public** **abstract** **void** deleteSort();  } |

实体类（Sort.java）

|  |
| --- |
| /\*\*  \* 账务分类实体类  \*/  **public** **class** Sort {  **private** **int** sid; // id  **private** String sname;// 小分类名称  **private** String parent; // 大分类名称  **private** String sdesc; // 分类说明  **…**  } |

表格内容填充工具类（ListTableModel.java）

|  |
| --- |
| /\*\*  \* 表格填充模式  \*/  **public** **class** ListTableModel<T> **extends** AbstractTableModel {  **private** java.util.List<T> list;  **private** String[] colNames;  **private** String[] propNames;    **public** ListTableModel(java.util.List<T> list, Class<?> c, String[] colNames, String[] propNames) **throws** Exception {  **if** (list == **null**) {  **throw** **new** Exception();  }  **this**.list = list;  **this**.colNames = colNames;  **this**.propNames = propNames;  }  **public** **int** getRowCount() {  **int** size = list.size();  **return** size < 10 ? 10 : size;  }  **public** **int** getColumnCount() {  **return** colNames.length;  }  **public** String getColumnName(**int** c) {  **return** colNames[c];  }  **public** Object getValueAt(**int** r, **int** c) {  **if**(r >= list.size()) **return** **null**;  **try** {  **return** BeanUtils.*getProperty*(list.get(r), propNames[c]);  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }  }  **public** T getInstance(**int** row) {  **if**(row >= list.size()) **return** **null**;  **return** list.get(row);  }  } |

分类管理窗口控制层类（SortMngController.java）

|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  **public** SortMngController(JFrame frame) {  **super**(frame);  }  @Override  **public** **void** addSort() {  }  @Override  **public** **void** editSort() {  }  @Override  **public** **void** deleteSort() {  }  } |

### 添加分类对话框

#### 添加分类对话框相关类创建

1. 复制已编写好的GUI分类操作界面类AbstractOperationSortDialog.java到view包中；
2. 在controller控制层，创建类AddSortController.java，继承AbstractOperationSortDialog用来创建添加分类窗口；

#### 弹出添加分类对话框

1. 在SortMngController#addSort()方法中，创建添加分类窗口（AddSortController）对象；
2. 通过setVisible（）方法显示主窗口

#### 添加分类对话框相关类详解

GUI分类操作（添加、编辑）窗口类（AbstractOperationSortDialog.java）

|  |
| --- |
| /\*\*  \* 分类操作（添加、编辑）界面类  \*/  **public** **abstract** **class** AbstractOperationSortDialog **extends** JDialog {  **protected** JComboBox parentBox = **new** JComboBox(**new** DefaultComboBoxModel(**new** String[]{"=请选择=", "收入", "支出"}));//父分类  **protected** JTextField snameTxt = **new** JTextField();//分类名称  **protected** JTextArea sdescArea = **new** JTextArea();//说明文本域  **protected** JLabel titleLabel = **new** JLabel("需要子类设置");    **private** JButton cancelBtn = **new** JButton("取消");  **private** JButton submitBtn = **new** JButton("确定");    **public** AbstractOperationSortDialog(JDialog dialog) {  **super**(dialog, **true**);  **this**.initDialog();  }    **public** **void** initDialog() {  init();  addComponent();  addListener();  }    **private** **void** init() {  **this**.setResizable(**false**);  **this**.setSize(350, 320);  **this**.setLayout(**null**);  GUITools.*center*(**this**);  **this**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  }    **private** **void** addComponent() {  // 标题标签  titleLabel.setFont(**new** java.awt.Font("宋体", 0, 18)); // NOI18N  titleLabel.setBounds(140, 10, 120, 28);  **this**.add(titleLabel);  // 父分类名称  JLabel parentLabel = **new** JLabel("父分类：");  parentLabel.setBounds(30, 50, 60, 28);  parentBox.setBounds(100, 50, 150, 28);  **this**.add(parentLabel);  **this**.add(parentBox);    // 分类名称  JLabel snameLabel = **new** JLabel("分类名称：");  snameLabel.setBounds(30, 90, 60, 28);  snameTxt.setBounds(100, 90, 150, 28);  **this**.add(snameLabel);  **this**.add(snameTxt);  // 分类说明  JLabel sdescLabel = **new** JLabel("分类说明：");  sdescLabel.setBounds(30, 130, 60, 28);  sdescArea.setColumns(20);  sdescArea.setRows(5);  JScrollPane scrollPane = **new** JScrollPane();  scrollPane.setBounds(100, 130, 200, 80);  scrollPane.setViewportView(sdescArea);  **this**.add(sdescLabel);  **this**.add(scrollPane);    //取消按钮  cancelBtn.setBounds(80, 220, 90, 28);  **this**.add(cancelBtn);    //确定按钮  submitBtn.setBounds(210, 220, 90, 28);  **this**.add(submitBtn);  }  **private** **void** addListener() {  cancelBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  AbstractOperationSortDialog.**this**.dispose();  }  });    submitBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  confirm();  }  });  }  /\*\*  \* 点击确定按钮时调用  \*/  **public** **abstract** **void** confirm();  } |

添加分类窗口控制层类（AddSortController.java）

|  |
| --- |
| /\*\*  \* 添加分类窗口控制层类  \*/  **public** **class** AddSortController **extends** AbstractOperationSortDialog {  **public** AddSortController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** **void** confirm() {  }  } |

### 编辑分类对话框

#### 编辑分类对话框相关类创建

1. 在controller控制层，创建类EditSortController.java，继承AbstractOperationSortDialog用来创建编辑分类窗口；

#### 弹出编辑分类对话框

1. 在SortMngController#editSort()方法中，创建编辑分类窗口（EditSortController）对象；
2. 通过setVisible（）方法显示主窗口

#### 编辑分类对话框相关类详解

分类操作（添加、编辑）界面类（AbstractOperationSortDialog.java）

|  |
| --- |
| 该类AbstractOperationSortDialog已在添加分类界面中介绍 |

编辑分类窗口控制层类（EditSortController.java）

|  |
| --- |
| /\*\*  \* 编辑分类窗口控制层类  \*/  **public** **class** EditSortController **extends** AbstractOperationSortDialog {  **public** EditSortController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** **void** confirm() {  }  } |

### 账务管理对话框

#### 账务管理对话框相关类创建

1. 复制已编写好的GUI分类管理界面类AbstractLedgerMngDialog.java到view包中；
2. 在domain包中，创建实体类 Ledger.java，类中成员变量参照数据库表gjp\_ledger的字段，要求名称一致；表格中要体现出分类名称，所以添加一个属性sname(分类名称)
   1. 实体类要与数据库和表格对应：
   2. 与数据库对应：把实体类对象保存到数据库中，以及把数据库数据查询出来封装到实体类对象中；
   3. 与表单对应：把实体类对象显示到表单中，把表单数据封装到实体类对象中；
3. 复制已编写好的工具类DateChooser.java与DateUtils.java 到 tools包中；
4. 在controller控制层，创建类LedgerMngController.java，继承AbstractLedgerMngDialog用来创建分类管理窗口;

#### 弹出账务管理对话框

1. 在MainFrameController#ledgerMng()方法中，创建账务管理窗口（LedgerMngController）对象；
2. 通过setVisible（）方法显示主窗口

#### 账务管理对话框相关类详解

账务管理窗口类（AbstractLedgerController.java）

|  |
| --- |
| /\*\*  \* 账务管理窗口类  \*/  **public** **abstract** **class** AbstractLedgerMngDialog **extends** JDialog {  **protected** JTextField beginDateTxt = **new** JTextField(6);//开始查询时间  **protected** JTextField endDateTxt = **new** JTextField(6);//结束查询时间  **protected** JComboBox parentBox = **new** JComboBox();//父分类下拉列表  **protected** JComboBox sortBox = **new** JComboBox();//小分类下拉列表  **protected** JTable ledgerDataTable = **new** JTable();//账务数据列表  **protected** JLabel inMoneyTotalLabel = **new** JLabel("总收入：0.00元");  **protected** JLabel payMoneyTotalLabel = **new** JLabel("总支出：0.00元");    **private** JButton queryBtn = **new** JButton("查　询");//查询按钮  **private** JButton pieBtn = **new** JButton("收/支比重统计");  **private** JButton closeBtn = **new** JButton("关闭");    **private** JButton addBtn = **new** JButton("添加");  **private** JButton editBtn = **new** JButton("编辑");  **private** JButton delBtn = **new** JButton("删除");    **public** AbstractLedgerMngDialog(JFrame frame) {  **super**(frame, **true**);  initDialog();  }    **protected** **void** initDialog() {  **this**.init();  **this**.addComponent();  **this**.addListener();  }    **private** **void** init() {  **this**.setResizable(**false**);// 设置窗体大小不可变  **this**.setTitle("账务管理：");// 设置标题  **this**.setSize(680, 400);// 设置大小  GUITools.*center*(**this**);//设置居中  **this**.setLayout(**null**);  **this**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);// 设置关闭按钮  }      **private** **void** addComponent() {  // 设置标签标题  JLabel titleLabel = **new** JLabel();  titleLabel.setFont(**new** Font("宋体", Font.***ITALIC***, 18));  titleLabel.setText("账务管理");  titleLabel.setBounds(280, 20, 165, 20);  **this**.add(titleLabel);  // 起始日期  JLabel beginDateLabel = **new** JLabel("起始：");  beginDateLabel.setBounds(30, 70, 60, 28);  **this**.beginDateTxt.setBounds(70, 70, 80, 28);  **this**.beginDateTxt.setEditable(**false**);  beginDateTxt.setText(String.*format*("%tF", DateUtils.*getFirstDayOfMethod*()));  DateChooser.*getInstance*().register(**this**.beginDateTxt);  **this**.add(beginDateLabel);  **this**.add(**this**.beginDateTxt);  // 结束日期  JLabel endDateLabel = **new** JLabel("至：");  endDateLabel.setBounds(160, 70, 30, 28);  **this**.endDateTxt.setBounds(190, 70, 80, 28);  **this**.endDateTxt.setEditable(**false**);  endDateTxt.setText(String.*format*("%tF", **new** Date()));  DateChooser.*getInstance*().register(**this**.endDateTxt);  **this**.add(endDateLabel);  **this**.add(**this**.endDateTxt);  // 收支选择  JLabel inAndPayLabel = **new** JLabel("收/支：");  inAndPayLabel.setBounds(280, 70, 50, 28);  parentBox.setModel(**new** DefaultComboBoxModel(**new** String[] {"-请选择-", "收入/支出", "收入", "支出"}));  **this**.parentBox.setBounds(320, 70, 90, 28);  **this**.add(inAndPayLabel);  **this**.add(**this**.parentBox);  // 收支项目  JLabel sortLabel = **new** JLabel("分类：");  sortLabel.setBounds(420, 70, 50, 28);  sortBox.setModel(**new** DefaultComboBoxModel(**new** String[]{"-请选择-"}));  **this**.sortBox.setBounds(460, 70, 110, 28);  **this**.add(sortLabel);  **this**.add(**this**.sortBox);  // 查询按钮  queryBtn.setBounds(580, 70, 70, 28);  **this**.add(queryBtn);  // 滚动面板  JScrollPane scrollPane = **new** JScrollPane();  scrollPane.setBounds(30, 100, 620, 160);  setTableModel(**null**);    ledgerDataTable.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);//单选  ledgerDataTable.getTableHeader().setReorderingAllowed(**false**);//列不能移动  scrollPane.setViewportView(ledgerDataTable);  **this**.add(scrollPane);  // 总收入标签  **this**.inMoneyTotalLabel.setBounds(400, 260, 120, 28);  **this**.add(inMoneyTotalLabel);  inMoneyTotalLabel.setForeground(**new** java.awt.Color(0, 102, 0));  // 总支出标签  **this**.payMoneyTotalLabel.setBounds(530, 260, 120, 28);  **this**.add(**this**.payMoneyTotalLabel);  payMoneyTotalLabel.setForeground(**new** java.awt.Color(255, 0, 0));    // 按钮  addBtn.setBounds(30, 290, 140, 28);  **this**.add(addBtn);  editBtn.setBounds(270, 290, 140, 28);  **this**.add(editBtn);  delBtn.setBounds(510, 290, 140, 28);  **this**.add(delBtn);  // 收支统计报表按钮  pieBtn.setBounds(30, 330, 140, 28);  **this**.add(pieBtn);  // 关闭按钮  closeBtn.setBounds(570, 330, 80, 28);  **this**.add(closeBtn);  }  /\*\*  \* 通过表格行号获取账务对象  \*/  @SuppressWarnings("unchecked")  **protected** Ledger getLedgerByTableRow(**int** row) {  **return** ((ListTableModel<Ledger>)ledgerDataTable.getModel()).getInstance(row);  }  /\*\*  \* 显示账务表格  \*/  **protected** **void** setTableModel(List<Ledger> ledgerList) {  String[] colNames = **new** String[] {"ID", "收/支", "分类", "金额", "账户", "创建时间", "说明"};  String[] propNames = **new** String[] {"lid", "parent", "sname", "money", "account", "createtime", "ldesc"};  **if**(ledgerList == **null** || ledgerList.size() == 0) {  ledgerDataTable.setModel(**new** DefaultTableModel(**new** Object[][] {  { **null**, **null**, **null**, **null**, **null**, **null**, **null** }, { **null**, **null**, **null**, **null**, **null**, **null**, **null** },  { **null**, **null**, **null**, **null**, **null**, **null**, **null** }, { **null**, **null**, **null**, **null**, **null**, **null**, **null** },  { **null**, **null**, **null**, **null**, **null**, **null**, **null** }, { **null**, **null**, **null**, **null**, **null**, **null**, **null** },  { **null**, **null**, **null**, **null**, **null**, **null**, **null** }, { **null**, **null**, **null**, **null**, **null**, **null**, **null** }  },colNames));  ledgerDataTable.setEnabled(**false**);  **return**;  }  **try** {  ledgerDataTable.setModel(**new** ListTableModel<Ledger>(ledgerList, Ledger.**class**, colNames, propNames));  ledgerDataTable.setEnabled(**true**);  }**catch** (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 给组件添加监听器  \*/  **private** **void** addListener() {  queryBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  queryLedger();  }  });  pieBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  pie();  }  });  closeBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  AbstractLedgerMngDialog.**this**.dispose();  }  });    addBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  addLedger();  }  });  editBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  editLedger();  }  });  delBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  deleteLedger();  }  });  parentBox.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  parentChange();  }  });    ledgerDataTable.addMouseListener(**new** MouseAdapter() {  **public** **void** mouseClicked(MouseEvent e) {  **if**(e.getButton() == 1) {  **if**(e.getClickCount() >= 2) {  editLedger();  }  }  }  });  }  /\*\*  \* 添加账务  \*/  **public** **abstract** **void** addLedger();    /\*\*  \* 添加账务  \*/  **public** **abstract** **void** editLedger();    /\*\*  \* 删除账务  \*/  **public** **abstract** **void** deleteLedger();    /\*\*  \* 查询账务  \*/  **public** **abstract** **void** queryLedger();    /\*\*  \* 父分类变化  \*/  **public** **abstract** **void** parentChange();    /\*\*  \* 生成饼形图  \*/  **public** **abstract** **void** pie();  } |

日期工具类（DateUtils.java）

|  |
| --- |
| /\*\*  \* 日期工具类  \*/  **public** **class** DateUtils {  /\*\*  \* 获取当前年  \*/  **public** **static** **int** getYear() {  **return** Calendar.*getInstance*().get(Calendar.***YEAR***);  }    /\*\*  \* 获取今年第一天  \*/  **public** **static** String firstdayByYear() {  **int** year = Calendar.*getInstance*().get(Calendar.***YEAR***);  **return** year + "-01-01";  }    /\*\*  \* 获取今年最后一天  \*/  **public** **static** String lastdayByYear() {  **int** year = Calendar.*getInstance*().get(Calendar.***YEAR***);  **return** year + "-12-31";  }    /\*\*  \* 把字符串转换成java.sql.Date  \*/  **public** **static** java.sql.Date toSQLDate(String str) {  **return** **new** java.sql.Date(*string2Date*(str).getTime());  }    /\*\*  \* 获取当前时间表示的字符串  \*/  **public** **static** String today() {  **return** *date2String*(**new** java.util.Date());  }    /\*\*  \* 把Date类型转换成字符串类型  \*/  **public** **static** String date2String(java.util.Date date) {  **return** String.*format*("%tF", date);  }    /\*\*  \* 把字符串类型转换成Date类型  \*/  **public** **static** java.util.Date string2Date(String str) {  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");  **try** {  **return** sdf.parse(str);  } **catch** (ParseException e) {  **throw** **new** RuntimeException("日期字符格式错误：" + str);  }  }    /\*\*  \* 返回本月第一天  \*/  **public** **static** java.util.Date getFirstDayOfMethod() {  Calendar c = Calendar.*getInstance*();//获取当前日期对象  c.set(Calendar.***DAY\_OF\_MONTH***, 1);//设置日期为1  **return** c.getTime();  }    /\*\*  \* 返回本月最后一天  \*/  **public** **static** java.util.Date getLastDayOfMethod() {  Calendar c = Calendar.*getInstance*();//获取当前日期对象  c.set(Calendar.***DAY\_OF\_MONTH***, c.getActualMaximum(Calendar.***DAY\_OF\_MONTH***));//设置日期为1  **return** c.getTime();  }  } |

日期选择组件工具类（DateChooser.java）

|  |
| --- |
| /\*\*  \* 日期选择组件工具类  \*/  **public** **class** DateChooser **extends** JPanel{  //此处代码有500多行，省略…，会使用该日期选择组件即可  } |

账务实体类（Ledger.java）

|  |
| --- |
| /\*\*  \* 账务实体类  \*/  **public** **class** Ledger {  **private** **int** lid; // 账务ID  **private** String parent; // 父分类名称  **private** String sname; // 小分类名称  **private** **int** sid; // 分类ID  **private** **double** money; // 金额  **private** String account; // 账户  **private** String createtime; // 创建时间  **private** String ldesc; // 账务说明  …  } |

账务管理窗口控制层类（LedgerMngController.java）

|  |
| --- |
| /\*\*  \* 账务管理窗口控制层类  \*/  **public** **class** LedgerMngController **extends** AbstractLedgerMngDialog {  **public** LedgerMngController(JFrame frame) {  **super**(frame);  }  @Override  **public** **void** addLedger() {  }  @Override  **public** **void** editLedger() {  }  @Override  **public** **void** deleteLedger() {  }  @Override  **public** **void** queryLedger() {  }  @Override  **public** **void** parentChange() {  }  @Override  **public** **void** pie() {  }  } |

### 添加账务对话框

#### 添加账务对话框相关类创建

1. 复制已编写好的GUI账务操作界面类AbstractOperationLedgerDialog.java到view包中；
2. 在controller控制层，创建类AddLedgerController.java，继承AbstractOperationLedgerDialog用来创建添加账务窗口；

#### 弹出添加账务对话框

1. 在LedgerMngController#addLedger()方法中，创建添加账务窗口（AddLedgerController）对象；
2. 通过setVisible（）方法显示主窗口

#### 添加账务对话框相关类详解

账务操作界面类（AbstractOperationLedgerDialog.java）

|  |
| --- |
| /\*\*  \* 账务操作界面类  \*/  **public** **abstract** **class** AbstractOperationLedgerDialog **extends** JDialog {  **protected** JComboBox parentBox = **new** JComboBox();//大分类下拉列表  **protected** JComboBox sortBox = **new** JComboBox();//小分类下拉列表  **protected** JTextField accountTxt = **new** JTextField();//账户文本框  **protected** JTextField moneyTxt = **new** JTextField("0.0");//金额文本框  **protected** JTextField createtimeTxt = **new** JTextField();//创建日期文本框  **protected** JTextArea ldescTxt = **new** JTextArea();//说明文本域  **protected** JLabel titleLabel = **new** JLabel("需要子类设置");    **private** JButton cancelBtn = **new** JButton("取消");  **private** JButton submitBtn = **new** JButton("确定");    **public** AbstractOperationLedgerDialog(JDialog dialog) {  **super**(dialog, **true**);  **this**.initDialog();  }    **public** **void** initDialog() {  init();  addComponent();  addListener();  }    **private** **void** init() {  **this**.setResizable(**false**);  **this**.setSize(400, 450);  **this**.setLayout(**null**);  GUITools.*center*(**this**);  **this**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  }    **private** **void** addComponent() {  // 标题标签  titleLabel.setFont(**new** java.awt.Font("宋体", 0, 18)); //  titleLabel.setBounds(160, 10, 120, 28);  **this**.add(titleLabel);  // "选择"下拉列表  JLabel parentLabel = **new** JLabel("收/支：");  parentLabel.setBounds(30, 50, 50, 28);  parentBox.setModel(**new** DefaultComboBoxModel(**new** String[] {"-请选择-", "收入", "支出"}));  parentBox.setBounds(70, 50, 100, 28);  **this**.add(parentLabel);  **this**.add(parentBox);    // 分类  JLabel sortLabel = **new** JLabel("分类：");  sortLabel.setBounds(30, 90, 50, 28);  sortBox.setModel(**new** DefaultComboBoxModel(**new** String[] {"-请选择-"}));  sortBox.setBounds(70, 90, 100, 28);  **this**.add(sortLabel);  **this**.add(sortBox);  // 账户  JLabel accountLabel = **new** JLabel("账户：");  accountLabel.setBounds(30, 130, 50, 28);  accountTxt.setBounds(70, 130, 200, 28);  **this**.add(accountLabel);  **this**.add(accountTxt);  // 金额  JLabel moneyLabel = **new** JLabel("金额：");  moneyLabel.setBounds(30, 170, 50, 28);  moneyTxt.setBounds(70, 170, 100, 28);  moneyTxt.setHorizontalAlignment(JTextField.***RIGHT***);// 文本右对齐  **this**.add(moneyLabel);  **this**.add(**this**.moneyTxt);    // 时间  JLabel createtimeLabel = **new** JLabel("时间：");  createtimeLabel.setBounds(30, 210, 50, 28);  createtimeTxt.setBounds(70, 210, 80, 28);  createtimeTxt.setText(String.*format*("%tF", **new** Date()));  DateChooser.*getInstance*().register(createtimeTxt);  **this**.add(createtimeLabel);  **this**.add(**this**.createtimeTxt);  // 说明  JLabel ldescLabel = **new** JLabel("说明：");  ldescLabel.setBounds(30, 250, 50, 28);  ldescTxt.setColumns(20);  ldescTxt.setRows(5);  JScrollPane scrollPane = **new** JScrollPane();  scrollPane.setBounds(70, 250, 260, 80);  scrollPane.setViewportView(ldescTxt);  **this**.add(ldescLabel);  **this**.add(scrollPane);    //取消按钮  cancelBtn.setBounds(30, 340, 90, 28);  **this**.add(cancelBtn);    //确定按钮  submitBtn.setBounds(260, 340, 90, 28);  **this**.add(submitBtn);  }  **private** **void** addListener() {  cancelBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  AbstractOperationLedgerDialog.**this**.dispose();  }  });    submitBtn.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  confirm();  }  });  parentBox.addActionListener(**new** ActionListener() {  **public** **void** actionPerformed(ActionEvent evt) {  changeParent();  }  });  }    /\*\*  \* 父分类变化时调用本方法  \*/  **public** **abstract** **void** changeParent();  /\*\*  \* 点击确定按钮时调用的方法  \*/  **public** **abstract** **void** confirm();  } |

添加掌握界面控制层类（AddLedgerController.java）

|  |
| --- |
| /\*\*  \* 添加账务界面控制层类  \*/  **public** **class** AddLedgerController **extends** AbstractOperationLedgerDialog {  **public** AddLedgerController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** **void** changeParent() {  }  @Override  **public** **void** confirm() {  }  } |

### 编辑账务对话框

#### 编辑账务对话框相关类创建

1. 在controller控制层，创建类EditLedgerController.java，继承AbstractOperationLedgerDialog用来创建添加账务窗口；

#### 弹出编辑账务对话框

1. 在LedgerMngController#editLedger()方法中，创建编辑账务窗口（EditLedgerController）对象；
2. 通过setVisible（）方法显示主窗口

#### 编辑账务对话框相关类详解

编辑账务窗口控制层类（EditLedgerController.java）

|  |
| --- |
| /\*\*  \* 编辑账务窗口控制层类  \*/  **public** **class** EditLedgerController **extends** AbstractOperationLedgerDialog {  **public** EditLedgerController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** **void** changeParent() {  }  @Override  **public** **void** confirm() {  }  } |

### 收/支比重统计对话框

#### 收/支比重统计对话框相关类创建

1. 复制已编写好的GUI账务收支比重统计窗口类AbstractShapeDialog.java到view包中；
2. 在controller控制层，创建类PieController.java，继承AbstractShapeDialog用来创建账务收支比重统计窗口

#### 弹出收/支比重统计对话框

1. 在LedgerMngController#pie()方法中，创建账务收支比重统计窗口（PieController）对象；
2. 通过setVisible（）方法显示主窗口

#### 收/支比重统计对话框相关类详解

账务收支比重统计窗口类（AbstractShapeDialog.java）

|  |
| --- |
| /\*\*  \* 账务收支比重窗口类  \*/  **public** **abstract** **class** AbstractShapeDialog **extends** JDialog {  **public** AbstractShapeDialog(JDialog dialog) {  **super**(dialog);  }    **protected** **void** initDialog() {  **this**.addComponent();  **this**.init();  }    **private** **void** init() {  **this**.pack();  GUITools.*center*(**this**);  **this**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  }    **public** **abstract** List<String> getImagePaths();    **private** **void** addComponent() {  List<String> imagePaths = getImagePaths();  **if**(imagePaths == **null**) {  **return**;  }  JPanel panel = **new** JPanel();  **this**.add(panel);  **for**(String imagePath : imagePaths) {  **try** {  Image image = ImageIO.*read*(**new** File(imagePath));  panel.add(**new** JLabel(**new** ImageIcon(image)));  } **catch**(Exception e) {  **throw** **new** RuntimeException(e);  }  }  }  } |

账务收支比重统计窗口控制层类（PieController.java）

|  |
| --- |
| /\*\*  \* 账务收支比重统计窗口控制层类  \*/  **public** **class** PieController **extends** AbstractShapeDialog {  **public** PieController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** List<String> getImagePaths() {  **return** **null**;  }  } |

## 分类模块

### 分类模块相关类创建

创建与分类模块相关的类，只是创建类而已，无需在类中添加代码。

1. 复制已编写好的工具类JDBCUtils.java 到 tools包中；
2. 复制jar包mysql-connector-java-5.1.28-bin.jar、commons-dbutils-1.4.jar、commons-dbcp-1.4.jar、commons-pool-1.3.jar，到lib文件夹中，通过Build Path操作，添加到classPath路径中，提供给JDBCUtils使用;
3. 在dao数据层，创建类SortDao.java，给SortDao类添加一个成员变量QueryRunner对象，因为我们使用dbutils来操作数据库，所以在本项目中所有的DAO类中都需要给出类型为QueryRunner的成员变量；
4. 在service业务层，创建类SortService.java，给SortService类添加一个类型为SortDao的成员变量，因为service都依赖dao。

JDBCUtils.java

|  |
| --- |
| **public** **class** JDBCUtils {  **public** **static** **final** String ***DRIVER\_CLASS\_NAME*** = "com.mysql.jdbc.Driver";  **public** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/gjp";  **public** **static** **final** String ***USERNAME*** = "root";  **public** **static** **final** String ***PASSWORD*** = "root";    **private** **static** **final** **int** ***MAX\_IDLE*** = 3;  **private** **static** **final** **long** ***MAX\_WAIT*** = 5000;  **private** **static** **final** **int** ***MAX\_ACTIVE*** = 5;  **private** **static** **final** **int** ***INITIAL\_SIZE*** = 3;    **private** **static** BasicDataSource *dataSource* = **new** BasicDataSource();  **static** {  *dataSource*.setDriverClassName(***DRIVER\_CLASS\_NAME***);  *dataSource*.setUrl(***URL***);  *dataSource*.setUsername(***USERNAME***);  *dataSource*.setPassword(***PASSWORD***);    *dataSource*.setMaxActive(***MAX\_IDLE***);  *dataSource*.setMaxWait(***MAX\_WAIT***);  *dataSource*.setMaxActive(***MAX\_ACTIVE***);  *dataSource*.setInitialSize(***INITIAL\_SIZE***);  }    **public** **static** DataSource getDataSource() {  **return** *dataSource*;  }  } |

SortDao.java

|  |
| --- |
| **package** cn.itcast.gjp.dao;  **import** org.apache.commons.dbutils.QueryRunner;  **import** cn.itcast.gjp.tools.JDBCUtils;  /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  //获取数据库连接池，得到操作表数据的对象 QueryRunner  **private** QueryRunner qr = **new** QueryRunner(JDBCUtils.*getDataSource*());  } |

SortService.java

|  |
| --- |
| **package** cn.itcast.gjp.service;  **import** cn.itcast.gjp.dao.SortDao;  /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  //service都依赖dao  **private** SortDao sortDao = **new** SortDao();  } |

### 查询所有分类

我们现在显示的分类管理界面的表格中没有数据，而我们现在要实现的功能是让分类管理这个对话框弹出时就显示出所有分类信息。



为了完成这一功能，我们需要在SortMngController类的构造器中获取所有分类数据，然后再显示到表格中，具体步骤如下：

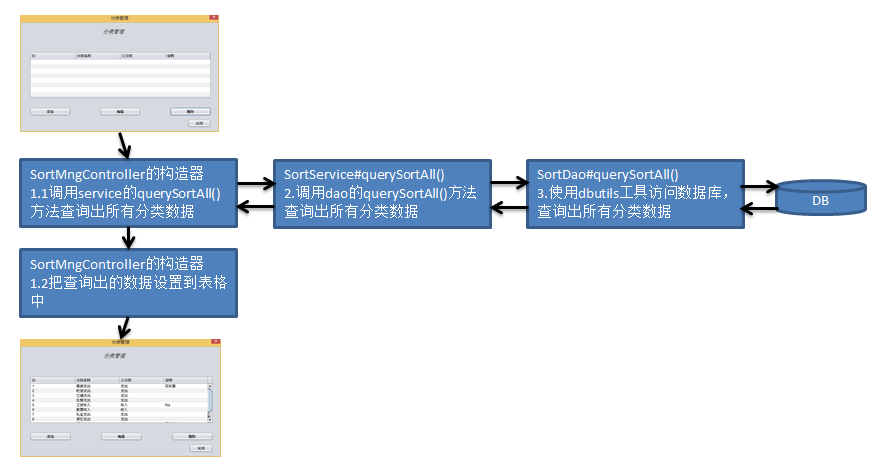
1. SortMngController()构造器：

1.1 通过SortService#querySortAll()方法得到所有分类对象，即List<Sort>类型；

1.2 把得到的List<Sort>设置到表格中，需要使用SortMngController类的setTableModel(List<Sort>)方法完成，这个方法是在SortMngController的父类（AbstractSortMngDialog）类中定义的，至于这个方法内部如何完成的，我们这里不做讨论，在这里不对GUI相关的功能做讨论。

2. SortService#querySortAll()：调用SortDao#querySortAll()方法；

3. SortDao#querySortAll()：使用dbutils工具访问数据库。



编写DAO数据层SortDao#querySortAll()

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  //获取数据库连接池，得到操作表数据的对象 QueryRunner  **private** QueryRunner qr = **new** QueryRunner(JDBCUtils.*getDataSource*());  /\*\*  \* 查询所有分类数据  \*/  **public** List<Sort> querySortAll(){  String sql = "select \* from gjp\_sort";  Object[] params = {};  **try** {  **return** qr.query(sql, **new** BeanListHandler<Sort>(Sort.**class**));  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Service业务层SortService#querySortAll()

|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  //service都依赖dao  **private** SortDao sortDao = **new** SortDao();  /\*\*  \* 查询所有分类数据  \*/  **public** List querySortAll(){  **return** sortDao.querySortAll();  }  } |

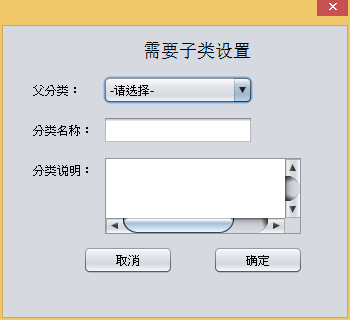
编写Controller控制层SortMngController()构造器

|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  **private** SortService sortService = **new** SortService();  **public** SortMngController(JFrame frame) {  **super**(frame);  /\*  \* 1.调用sortService#querySortAll() 获取分类数据  \* 2.显示分类数据到表格中 setTableModel(List<Sort>)  \*/  List sortList = sortService.querySortAll();  setTableModel(sortList);  }  } |

### 添加分类

#### 向数据库中添加分类记录

　　在弹出“添加分类”对话框后，用户会在对话框的表单中填写数据，然后点击“确定”按钮。



AddSortController类的comfirm()方法会在“确定”按钮被点击时调用。所以我们需要在confirm()方法中添加如下代码：

1. AddSortController#comfirm()：

1.1获取表单数据；

1.2对表单数据进行校验；

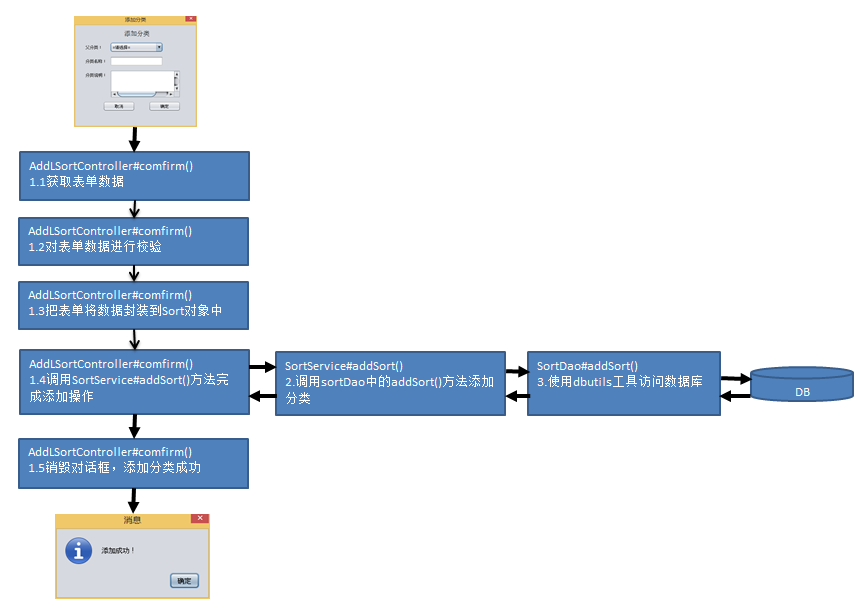
1.3把表单数据封装到Sort对象中；

1.4调用sortService#addSort()方法完成添加操作；

1.5关闭对话框。

2.SortService#addSort()：调用SortDao#addSort()方法；

3.SortDao#addSort()：使用dbutils工具访问数据库。



编写Dao数据层 SortDao#addSort()

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  …  /\*\*  \* 添加分类  \*/  **public** **void** addSort(Sort sort){  String sql = "insert into gjp\_sort(sname,parent,sdesc) values(?,?,?)";  Object[] params = {sort.getSname(), sort.getParent(), sort.getSdesc()};  **try** {  qr.update(sql, params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Service业务层 SortService#addSort()

|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  //service都依赖dao  **private** SortDao sortDao = **new** SortDao();  /\*\*  \* 添加分类  \*/  **public** **void** addSort(Sort sort) {  sortDao.addSort(sort);  }  } |

编写Controller控制层AddSortController#confirm()

|  |
| --- |
| /\*\*  \* 添加分类窗口控制层类  \*/  **public** **class** AddSortController **extends** AbstractOperationSortDialog {  **…**  @Override  **public** **void** confirm() {  /\*  \* 1获取表单数据；  \* 2对表单数据进行校验；  \* 3把表单数据封装到Sort对象中；  \* 4调用sortService#addSort()方法完成添加操作；  \* 5关闭对话框。  \*/  //1获取表单数据  String parent = **this**.parentBox.getSelectedItem().toString();  String sname = **this**.snameTxt.getText();  String sdesc = **this**.sdescArea.getText();  //2对表单数据进行校验  **if** (parent.equals("-请选择-") || sname.isEmpty() || sdesc.isEmpty()) {  JOptionPane.*showMessageDialog*(**this**, "请认真填写各项数据！");  **return**;  }  //3把表单数据封装到Sort对象中  Sort sort = **new** Sort();  sort.setParent(parent);  sort.setSname(sname);  sort.setSdesc(sdesc);  //4调用sortService#addSort()方法完成添加操作  sortService.addSort(sort);  //5关闭对话框  **this**.dispose();  JOptionPane.*showMessageDialog*(**this**, "添加成功！");  }  } |

#### 刷新分类表格

到现在添加分类功能已经实现了，但是还存在一个小问题，用户新添加的记录不会在表格显示出来。



为了在添加分类之后，表格中可以显示刚刚添加的分类数据，我们需要对表格数据进行刷新。

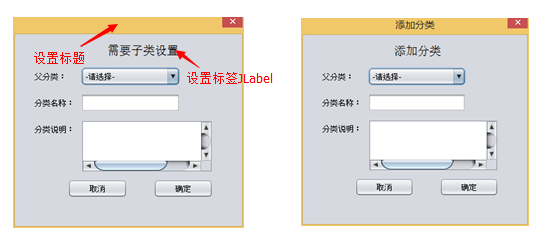
所谓刷新表格数据，其实就是重新查询所有分类，然后把新的所有分类设置给表格让其显示。

SortMngController.java

|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  **private** SortService sortService = **new** SortService();  @Override  **public** **void** addSort() {  /\*  \* 1.创建添加分类窗口（AddSortController）对象；  \* 2.通过setVisible（）方法显示主窗口  \*/  **new** AddSortController(**this**).setVisible(**true**);  /\*  \* 3.刷新分类表格数据  \*/  List sortList = sortService.querySortAll();  **this**.setTableModel(sortList);  }  } |

#### 修改对话框标题

还有些小事儿一直没做，就是对话框的标题一直没有修改。



编写Controller控制层AddSortController构造方法

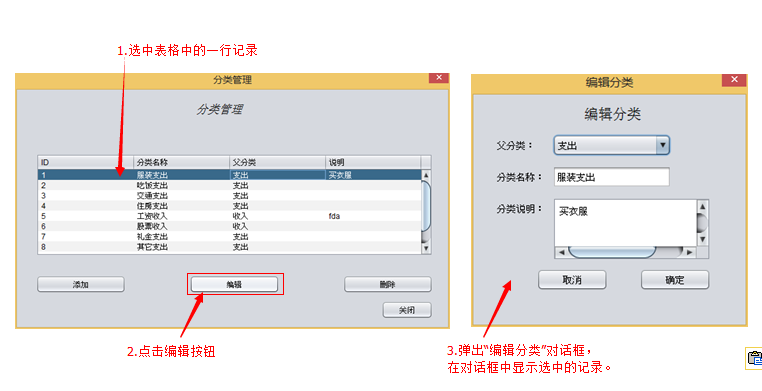
|  |
| --- |
| /\*\*  \* 添加分类窗口控制层类  \*/  **public** **class** AddSortController **extends** AbstractOperationSortDialog {  **private** SortService sortService = **new** SortService();  **public** AddSortController(JDialog dialog) {  **super**(dialog);  /\*  \* 1.设置窗口标题  \* 2.设置标题标签  \*/  **this**.setTitle("添加分类");  **this**.titleLabel.setText("添加分类");  }  } |

### 编辑分类

接下来我们要完成编辑分类功能，与添加分类相似，难度略微大一点。下面我们分析一下：

要编辑分类，首先用户需要在表格中选择要编辑的记录，然后点击“编辑”按钮。这时要弹出编辑分类对话框，在对话框表格中显示被选择的表格记录。

当用户在表格中对数据进行修改后，再点击编辑分类对话框上的“确定”按钮，我们需要对照表单数据对数据库数据进行修改，然后把修改后的数据呈现在表格上。



#### 修改SortMngController#editSort()方法

通过上面的分析我们了解到，在弹出EditSortController对话框之前，我们需要获取用户选择的表格记录。然后把表格记录传递给EditSortController对话框，这样EditSortController对话框在弹出后才能显示表格记录。具体步骤如下：

1). 获取用户选择的表格行号；

2). 判断行号是否小于0，如果小于0说明用户没有选择；

3). 通过表格行号获取表格中该行数据封装到Sort对象中（要实现这一功能需要使用JTable的一些复杂功能，本项目中已经把这一功能封装好了，我们只需要调用方法即可。）

4). 判断获取的Sort对象是否为null，如果为null说明用户选择的是空行；

5). 创建EditSortController对话框，并把Sort对象传递给EditSortController的构造器；

6). 用户编辑结束后，回到这里，我们再查询所有分类设置给JTable，作用是刷新JTable中的分类信息。

编写Controller控制层SortMngController#editSort()方法

|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  **…**  @Override  **public** **void** editSort() {  //1. 获取用户选择的表格行号  **int** row = **this**.sortDataTable.getSelectedRow();  //2. 判断行号是否小于0，如果小于0说明用户没有选择；  **if** (row < 0) {  JOptionPane.*showMessageDialog*(**this**, "请选择要编辑的分类！");  **return**;  }  //3. 通过表格行号获取表格中该行数据封装到Sort对象中  Sort sort = **this**.getSortByTableRow(row);  //4. 判断获取的Sort对象是否为null，如果为null说明用户选择的是空行；  **if** (sort == **null**) {  JOptionPane.*showMessageDialog*(**this**, "您选择的是空行");  **return**;  }  //5. 创建EditSortController对话框，并把Sort对象传递给EditSortController的构造器  **new** EditSortController(**this**, sort).setVisible(**true**);  }  } |

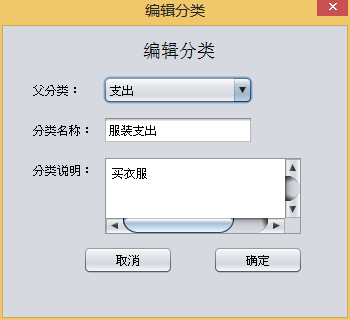
#### 修改EditSortController构造器

在修改SortMngController#editSort()方法的第5步，我们在创建EditSortController对象时，把Sort对象赋给了EditSortController构造器，但EditSortController构造器没有Sort参数，所以我们要修改它。

我们要让EditSortController构造器接收Sort参数，并且还要把Sort的属性值显示在EditSortController表单组件上。这样，在EditSortController被弹出时就显示了表格数据。

编写Controller控制层EditSortController构造器

|  |
| --- |
| /\*\*  \* 编辑分类窗口控制层类  \*/  **public** **class** EditSortController **extends** AbstractOperationSortDialog {  **private** Sort sort;  **private** SortService sortService = **new** SortService();  **public** EditSortController(JDialog dialog, Sort sort) {  **super**(dialog);  **this**.sort = sort;  /\*  \* 设置窗口标题  \* 设置标题标签  \*/  **this**.setTitle("编辑分类");  **this**.titleLabel.setText("编辑分类");  //把Sort的属性值显示在EditSortController表单组件上  **this**.parentBox.setSelectedItem(sort.getParent());  **this**.snameTxt.setText(sort.getSname());  **this**.sdescArea.setText(sdescArea.getText());  }  } |



#### 完成修改分类操作

当用户点击“编辑分类”对话框中的“确定”按钮时，会调用confirm()方法。在　confirm()方法中我们需要获取表单中的数据，对原有Sort对象进行修改，然后把Sort对象传递给SortService#editSort()方法，完成对数据库的修改。具体步骤如下：

1. EditSortCotroller#confirm()：

1.1.获取用户修改后的表单数据；

1.2.对数据进行前端校验；

1.3.把用户修改后的数据同步到Sort对象中；

1.4.调用SortService#editSort()方法对数据库进行修改；

1.5.关闭对话框；

1. SortService#editSort()：调用SortDao#editSort()编辑分类
2. SortDao#editSort()：使用dbutils工具访问数据库

编写Dao数据层SortDao#editSort()

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  …  /\*\*  \* 编辑分类  \*/  **public** **void** editSort(Sort sort) {  String sql = "update gjp\_sort set sname=?,parent=?,sdesc=? where sid=?";  Object[] params = {sort.getSname(),sort.getParent(),sort.getSdesc(), sort.getSid()};  **try** {  qr.update(sql, params);  } **catch**(SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Service业务层SortService#editSort()

|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  …  /\*\*  \* 编辑分类  \*/  **public** **void** editSort(Sort sort) {  sortDao.editSort(sort);  }  } |

编写Controller控制层EditSortController#confirm()

|  |
| --- |
| /\*\*  \* 编辑分类窗口控制层类  \*/  **public** **class** EditSortController **extends** AbstractOperationSortDialog {  **…**  @Override  **public** **void** confirm() {  //1.获取用户修改后的表单数据  String parent = **this**.parentBox.getSelectedItem().toString();  String sname = **this**.snameTxt.getText();  String sdesc = **this**.sdescArea.getText();  //2.对数据进行前端校验  **if** (parent.equals("-请选择-") || sname.isEmpty() || sdesc.isEmpty()) {  JOptionPane.*showMessageDialog*(**this**, "请认真填写各项数据！");  **return**;  }  //3.把用户修改后的数据同步到Sort对象中；  sort.setParent(parent);  sort.setSname(sname);  sort.setSdesc(sdesc);  //4.调用sortService#addSort()方法完成添加操作  sortService .editSort(sort);  //5.关闭对话框  **this**.dispose();  JOptionPane.*showMessageDialog*(**this**, "编辑成功！");  }  } |

#### 刷新分类表格

与添加分类一样，在编辑分类后也需要对表格进行刷新，这样才能让用户看到修改后的分类数据。

编写Controller控制层SortMngController#editSort()

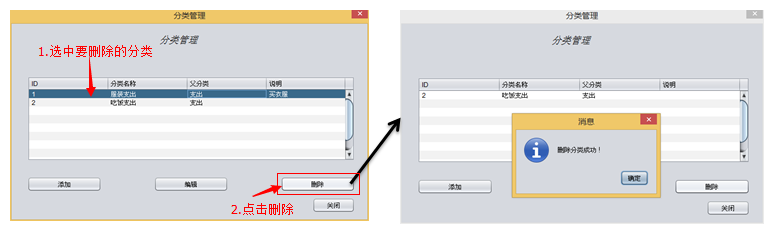
|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  …  @Override  **public** **void** editSort() {  …  //6. 用户编辑结束后，回到这里，我们再查询所有分类设置给JTable，作用是刷新JTable中的分类信息。  List sortList = sortService.querySortAll();  **this**.setTableModel(sortList);  }  } |

### 删除分类

接下来我们做删除分类功能，删除分类功能与编辑分类功能比较相似，但要简单一些，因为删除不需要弹出对话框。下面我们来分析一下：

首先用户在表格中选中要删除的一行记录，然后点击“删除”按钮。这时我们要获取用户选中的这行记录，然后把记录从数据库中删除。

最后我们需要刷新表格，让删除的记录在表格中消失。即让表格数据与数据库数据同步。这一点已经在添加、编辑中做过了。具体步骤如下：



1. SortMngController#deleteSort()
   1. 判断用户是否选择要删除的表格记录
   2. 获取用户选择的表格记录
   3. 判断用户选择的是否为空行
   4. 判断用户是否真的要删除记录
   5. 调用SortService#deleteSort()方法删除记录
   6. 刷新表格数据
2. SortService#deleteSort()：调用SortDao#deleteSort()删除分类
3. SortDao#deleteSort()：使用dbutils工具访问数据库

编写Dao数据层SortDao#deleteSort()

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  …  /\*\*  \* 删除分类  \*/  **public** **void** deletetSort(Sort sort) {  String sql = "delete from gjp\_sort where sid=?";  Object[] params = {sort.getSid()};  **try** {  qr.update(sql, params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Service业务层SortService#deleteSort()

|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  //service都依赖dao  **private** SortDao sortDao = **new** SortDao();    /\*\*  \* 查询所有分类数据  \*/  **public** List querySortAll(){  **return** sortDao.querySortAll();  }    /\*\*  \* 添加分类  \*/  **public** **void** addSort(Sort sort) {  sortDao.addSort(sort);  }  /\*\*  \* 编辑分类  \*/  **public** **void** editSort(Sort sort) {  sortDao.editSort(sort);    }    /\*\*  \* 删除分类  \*/  **public** **void** deleteSort(Sort sort) {  sortDao.deleteSort(sort);  }  } |

编写Controller控制层SortMngController#deleteSort()

|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  …  /\*\*  \* 删除分类  \*/  **public** **void** deleteSort(Sort sort) {  sortDao.deleteSort(sort);  }  } |

编写Controller控制层SortMngController#deleteSort()

|  |
| --- |
| /\*\*  \* 分类管理窗口控制层类  \*/  **public** **class** SortMngController **extends** AbstractSortMngDialog {  …  @Override  **public** **void** deleteSort() {  //1判断用户是否选择要删除的表格记录  **int** row = **this**.sortDataTable.getSelectedRow();  **if** (row < 0) {  JOptionPane.*showMessageDialog*(**this**, "请选择要删除的分类！");  **return**;  }  //2获取用户选择的表格记录  Sort sort = **this**.getSortByTableRow(row);  //3判断用户选择的是否为空行  **if** (sort == **null**) {  JOptionPane.*showMessageDialog*(**this**, "您选择的是空行");  **return**;  }  //4判断用户是否真的要删除记录  **int** m = JOptionPane.*showConfirmDialog*(**this**, "您是否确定要删除该行记录？");  **if** (m == JOptionPane.***OK\_OPTION***) {  //5调用SortService#deleteSort()方法删除记录  sortService.deleteSort(sort);  }  //6刷新表格数据  List sortList = sortService.querySortAll();  **this**.setTableModel(sortList);  }  } |

### 分类模块总结

恭喜大家，我们已经完成了分类模块所有功能。通过分类模块功能的实现，大家可能会发现如下几个问题：

* 难度大的地方：自行功能分析的能力欠缺，需要讲师来分析；
* 代码实现：
* Controller层：代码量最大；
* Service层：代码量最小；
* DAO层：相似度高，难度不大；

到现在为止我们是第一次写这么复杂的项目，所以大家之间也没有练习过功能分析的能力，所以第一次需要讲师来分析是正常的，当大家不断练习后，有了正确的思维模式，自然就能分析了。

Controller层的作用是“调度”，调度的是表现层和业务层，主要功能分为：一是把表现层的数据交给业务层处理；二是把业务层返回的数据交给表现层显示。至于为什么这一层的代码量大，这里我要说明一下，其实不是Controller层的代码量大，而是其他层代码量少，所以你会感觉Controller层代码量大。

Service层的作用是“业务”，我们也可以把“业务”当成是“功能”。那为什么Service层代码量少呢？原因是现在我们写的项目很小，没有什么复杂的业务，而今后大家要写的大型项目代码量最大的就是Service层。

DAO层是操作数据库，现在我们使用的是commons-dbutils工具来简化JDBC，所以我们发现代码不多，比较简单。最后我们还会学习其他DAO层的工具，例如：hibernate和mybatis，他们都是JDBC的封装，用来简化JDBC。

思考：

1. *Service层是否必须存在？*
2. *是否存在冗余代码？如果存在那么在哪里存在？如何优化？*

## 账务模块

### 账务模块相关类创建

下面要创建与账务模块相关的类。可参考分类模块相关类的创建，重复的注意事项这里就不再赘述了。

1. 复制已编写好的工具类JFreeChartUtils.java 到 tools包中；
2. 复制jar包jfreechart-1.0.9.jar、jcommon-1.0.12.jar，到lib文件夹中，通过Build Path操作，添加到classPath路径中，提供给JDBCUtils使用;
3. 在dao数据层，创建类LedgerDao.java，给LedgerDao类添加一个成员变量QueryRunner对象，因为我们使用dbutils来操作数据库，所以在本项目中所有的DAO类中都需要给出类型为QueryRunner的成员变量；
4. 在service业务层，创建类LedgerService.java，给LedgerService类添加一个类型为LedgerDao的成员变量，因为service都依赖dao。

工具类JFreeChartUtils.java

|  |
| --- |
| **public** **class** JFreeChartUtils {  /\*\*  \* 使用jfreechart生成图片  \* **@param** title 标题  \* **@param** totalMap 数据（key:分类名称, value：该分类的汇总数据，例如分类名称为"工资收入"，汇总数据为所有工资收入的和）  \* **@param** sum 汇总数据和（例如"工资收入汇总 + 股票收入汇总 + ...."）  \* **@param** path 生成图片保存路径  \*/  **public** **static** **void** pie(String title, Map<String,Double> totalMap, **double** sum, String path) {  DefaultPieDataset pieDataset = **new** DefaultPieDataset();  **for**(String dataName : totalMap.keySet()) {  **double** dataValue = totalMap.get(dataName);  String bf = String.*format*("%.2f%%", dataValue / sum \* 100);  dataName = dataName + ":" + dataValue + "元(" + bf + ")";  pieDataset.setValue(dataName, dataValue);  }  JFreeChart chart = ChartFactory.*createPieChart3D*(title, pieDataset, **true**, **true**, **false**);    **try** {  ChartUtilities.*saveChartAsJPEG*(**new** File(path), chart, 500, 300);  } **catch**(Exception e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Dao数据层LedgerDao.java

|  |
| --- |
| /\*\*  \* 账务数据层类  \*/  **public** **class** LedgerDao {  //获取数据库连接池，得到操作表数据的对象 QueryRunner  QueryRunner qr = **new** QueryRunner(JDBCUtils.*getDataSource*());  } |

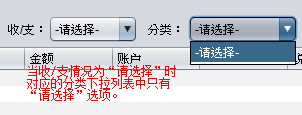
编写Service业务层，LedgerService.java

|  |
| --- |
| /\*\*  \* 账务业务层类  \*/  **public** **class** LedgerService {  //service都依赖dao  **private** LedgerDao ledgerDao = **new** LedgerDao();  } |

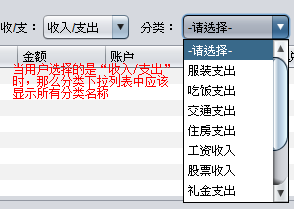
### 下拉列表之间数据级联

现在弹出的账务管理对话框中，可以在下拉列表中选择“收/支”情况，当用户在选择“收/支”情况后，对应的“分类”情况要随之变化。下面是每项“收/支”选项应该对应的“分类”情况：

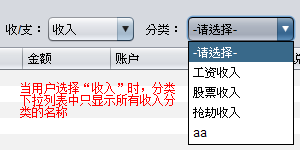
* 请选择：“分类”下拉列表中只应该包含“请选择”选项，没有其他选项；



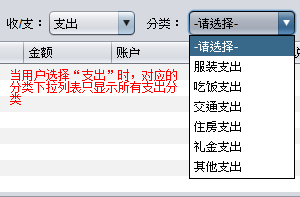
* 收入/支出：“分类”下拉列表中应该包含所有分类选项；



* 收入：“分类”下拉列表中只应该包含所有收入分类选项；



* 支出：“分类”下拉列表中只应该包含所有支出分类选项；



这说明在用户选择某项“收/支”时，我们要获取用户选择的“收/支”情况去查询数据库，得到对应的“分类”情况，放到分类下拉列表中。

具体步骤如下：

1. 获取用户选择的“收/支”情况；
2. 判断用户选择的是哪一种“收支情况”；
3. 如果是：“-请选择-”，那么使用只包含“-请选择-”字符串的Object[]数组来创建DefualtComboxModel，然后设置给下拉列表对象，这部分涉及到了GUI的内容，大家只需要了解怎么使用即可；
4. 如果是：“收入/支出”，那么使用SortService#querySortNameAll()方法查询所有分类名称，得到List<Object>对象，然后把“-请选择-”字符串插入到List<Object>的最前面，即下标为0的位置上，再把List<Object>转换成数组，再创建DefualtComboxModel，然后设置给下拉列表对象；
5. 如果是：“收入”或“支出”，那么使用SortService#querySortNameByParent()方法查询出指定父分类的子分类列表，得到List<Object>对象，然后把“-请选择-”字符串插入到List<Object>的最前面，即下标为0的位置上，再把List<Object>转换成数组，再创建DefualtComboxModel，然后设置给下拉列表对象；

编写dao数据层 SortDao#querySortNameAll() 与 SortDao#querySortNameByParent()

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  …  /\*\*  \* 查询所有分类名称  \*/  **public** List<Object> querySortNameAll(){  String sql = "select sname from gjp\_sort";  Object[] params = {};  **try** {  **return** qr.query(sql, **new** ColumnListHandler(), params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  /\*\*  \* 查询指定父分类下的子分类名称  \*/  **public** List<Object> querySortNameByParent(String parent){  String sql = "select sname from gjp\_sort where parent=?";  Object[] params = {parent};  **try** {  **return** qr.query(sql, **new** ColumnListHandler(), params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

编写Service业务层SortService#querySortNameAll() 与 SortService#querySortNameByParent()

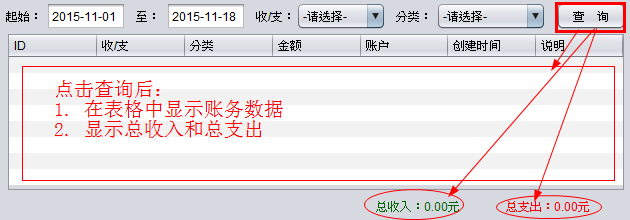
|  |
| --- |
| /\*\*  \* 账务分类业务层类  \*/  **public** **class** SortService {  …  /\*\*  \* 查询所有分类名称  \*/  **public** List<Object> querySortNameAll(){  **return** sortDao.querySortNameAll();  }  /\*\*  \* 查询指定父分类下的子分类名称  \*/  **public** List<Object> querySortNameByParent(String parent){  **return** sortDao.querySortNameByParent(parent);  }  } |

编写Controller控制层LedgerMngController#parentChange()

|  |
| --- |
| /\*\*  \* 账务管理窗口控制层类  \*/  **public** **class** LedgerMngController **extends** AbstractLedgerMngDialog {  **private** SortService sortService = **new** SortService();  …  @Override  **public** **void** parentChange() {  //1.获取用户选择的“收/支”情况；  String parent = **this**.parentBox.getSelectedItem().toString();  //2.判断用户选择的是哪一种“收支情况”  **if** (parent.equals("-请选择-")) {  // 情况一, “-请选择-”  Object[] sortItems = {"-请选择-"};  sortBox.setModel(**new** DefaultComboBoxModel(sortItems));  } **else** **if** (parent.equals("收入/支出")) {  // 情况二, “收入/支出”  List<Object> sortItems = sortService.querySortNameAll();  sortItems.add(0, "-请选择-");  sortBox.setModel(**new** DefaultComboBoxModel(sortItems.toArray()));  } **else** {  // 情况三, “收入”或“支出”  List<Object> sortItems = sortService.querySortNameByParent(parent);  sortItems.add(0, "-请选择-");  sortBox.setModel(**new** DefaultComboBoxModel(sortItems.toArray()));  }  }  } |

### 多条件查询账务

#### 需求分析



当用户点击“查询”按钮后，我们需要从数据库查询出数据，并显示到界面上。那么界面上要显示哪些数据呢？

* 表格中显示所有符合条件的账务数据；
* 在表格下方显示总收入和总支出；

**Controller层**

为了完成需求，Controller层首先需要把用户选择的条件数据封装到一个JavaBean对象（这个JavaBean类型需要我们自己创建，就叫QueryForm吧）中，然后传递给Service，Service返回我们所需的数据（Map类型），其中包含三个键/值，分别是：

* ledgerList（List<Ledger>）：对应表格中要显示的数据；
* in（Double）：对应总收入数据；
* pay（Double）：对应总支出数据。

**Service层**

Service层需要给Controller层返回一个Map对象，其中包含了Controller层需要的3个键/值。Service层需要创建Map对象，然后使用QueryForm对象调用DAO层得到所有符合条件的账务对象，即List<Ledger>，这时第一个键/值就已经有了。

然后再循环遍历每个账务对象，统计出总支出和总收入，这时第二、第三个键/值也有了。

注意：因为DAO中查出的Ledger对象没有sname，只有sid，这是因为在数据库表gjp\_ledger中只有sid字段，没有sname字段，所以我们需要在循环遍历每个账务对象时，还需要通过sid获取对应的sname，再设置给账务对象。

**DAO层**

　　DAO层需要做多条件组合查询，就是把QueryForm中的条件转换成WHERE子句。我们需要先给出一条基础的查询语句：SELECT \* FROM gjp\_ledger，然后根据QueryForm对象中提供的条件转换成SQL的条件，在基础查询语句后追加WHERE子句。

#### 功能实现步骤

**创建QueryForm类**

QueryForm用来封装查询条件，当用户选择查询条件后，点击“查询”按钮，我们首先需要获取用户选择的：开始时间、结束时间、父分类、分类，然后把这些数据封装到QueryForm对象中：

QueryForm.java

|  |
| --- |
| /\*\*  \* 查询条件类  \*/  **public** **class** QueryForm {  **private** String begin;  **private** String end;  **private** String parent;  **private** String sname;  …  } |

**实现LedgerDao#queryLedgerByQueryForm()方法**

1. 给出基本查询语句，以及参数值列表；

* 给出基本的查询语句：“SELECT \* FROM gjp\_ledger”；
* 创建参数值列表：List<Object> params；

1. 判断QueryForm中每个属性是否存在，确定是否转换成条件子句；

* 开始时间和结束时间是否存在，如果存在就添加这个条件子句；
* 因为开始时间与结束时间

1. 当确定添加条件子句后，要追加对应的参数值到参数值列表中；

* SQL语句中有几个“?”，就要提供几个值，所以我们需要在向params中添加对应的参数值。

1. 执行SQL语句。

　　注意：我们知道，多个条件中只有第一个条件前缀是“WHERE”，其他都使用“AND”。为了让所有条件前缀是“AND”，我们在基本查询语句后添加一个无意义的条件“1=1”把“WHERE”占用，那么后面的条件前缀就都是“AND”了。

LedgerDao.java

|  |
| --- |
| /\*\*  \* 账务数据层类  \*/  **public** **class** LedgerDao {  //获取数据库连接池，得到操作表数据的对象 QueryRunner  **private** QueryRunner qr = **new** QueryRunner(JDBCUtils.*getDataSource*());  **private** SortDao sortDao = **new** SortDao();    /\*\*  \* 多条件查询账务  \*/  **public** List<Ledger> queryLedgerByQueryForm(QueryForm queryForm){  /\*  \* 1.给出基本查询语句，以及参数值列表；   给出基本的查询语句：“SELECT \* FROM gjp\_ledger”；   创建参数值列表：List<Object> params；  \*/  StringBuilder sql = **new** StringBuilder("SELECT \* FROM gjp\_ledger WHERE 1=1");  List<Object> params = **new** ArrayList<Object>();  /\*  \* 2.判断QueryForm中每个属性是否存在，确定是否转换成条件子句；   开始时间和结束时间是否存在，如果存在就添加这个条件子句；   参数值添加开始时间与结束时间  3.当确定添加条件子句后，要追加对应的参数值到参数值列表中；   SQL语句中有几个“?”，就要提供几个值，所以我们需要在向params中添加对应的参数值。  \*/  **if** (queryForm.getBegin() != **null** && !queryForm.getBegin().isEmpty() && queryForm.getEnd() != **null** && !queryForm.getEnd().isEmpty()) {  sql.append("AND createtime BETWEEN ? AND ? ");  params.add(queryForm.getBegin());  params.add(queryForm.getEnd());  }  **if** (queryForm.getParent() != **null**) {  **if** (queryForm.getParent().equals("收入") || queryForm.getParent().equals("支出")) {  sql.append("AND parent=? ");  params.add(queryForm.getParent());  }  }  **if** (queryForm.getSname() != **null** && !queryForm.getSname().equals("-请选择-")) {  **int** sid = sortDao.getSidBySname(queryForm.getSname());  sql.append("AND sid=?");  params.add(sid);  }  /\*  \* 4.执行SQL语句  \*/  **try** {  **return** qr.query(sql.toString(), **new** BeanListHandler<Ledger>(Ledger.**class**), params.toArray());  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

**实现SortDao#getSidBySname()方法、SortDao#getSnameBySid()方法**

获取指定分类名称所对应分类ID

|  |
| --- |
| /\*\*  \* 账务分类数据层类  \*/  **public** **class** SortDao {  …  /\*\*  \* 获取指定分类ID对应的分类名称  \*/  **public** String getSnameBySid(**int** sid) {  String sql = "select sname from gjp\_sort where sid=?";  Object[] params = {sid};  **try** {  **return** (String) qr.query(sql, **new** ScalarHandler(), params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  /\*\*  \* 获取指定分类名称对应的分类ID  \*/  **public** **int** getSidBySname(String sname) {  String sql = "select sid from gjp\_sort where sname=?";  Object[] params = {sname};  **try** {  **return** (Integer) qr.query(sql, **new** ScalarHandler(), params);  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }} |

**实现LedgerService#query()方法**

1. 使用QueryForm对象调用LedgerDao#queryLedgerByQueryForm()方法，得到所有符合条件的Ledger对象，即List<Ledger>；
2. 循环List<Ledger>对象：

* 设计总收入和总支出；
* 获取每个Ledger对象的sid，然后调用SortDao#getSnameBySid()方法获取对应sanme，并设置给Ledger对象的sname属性；

1. 创建Map对象，把List<Ledger>，以及总支出和总收入保存到Map中，在Map中的key分别为：

* ledgerList：对应List<Ledger>；
* in：对应总收入；
* pay：对应总支出；

LedgerService.java

|  |
| --- |
| /\*\*  \* 账务业务层类  \*/  **public** **class** LedgerService {  //service都依赖dao  **private** LedgerDao ledgerDao = **new** LedgerDao();  **private** SortDao sortDao = **new** SortDao();    /\*\*  \* 多条件查询账务  \*/  **public** Map<String,Object> query(QueryForm queryForm) {  /\*  \* 1.使用QueryForm对象调用LedgerDao#queryLedgerByQueryForm()方法，  \* 得到所有符合条件的Ledger对象，即List<Ledger>；  \*/  List<Ledger> ledgerList = ledgerDao.queryLedgerByQueryForm(queryForm);  /\*  \* 2.循环List<Ledger>对象：   设计总收入和总支出；   获取每个Ledger对象的sid，然后调用SortDao#getSnameBySid()方法获取对应sanme，  并设置给Ledger对象的sname属性；  \*/  **double** inMoney = 0;  **double** payMoney = 0;  **for** (Ledger ledger : ledgerList) {  ledger.setSname(sortDao.getSnameBySid(ledger.getSid()));  **if** (ledger.getParent().equals("收入")) {  inMoney += ledger.getMoney();  } **else** **if** (ledger.getParent().equals("支出")) {  payMoney += ledger.getMoney();  }  }  /\*  \* 3.创建Map对象，把List<Ledger>，以及总支出和总收入保存到Map中，在Map中的key分别为：   ledgerList：对应List<Ledger>；   in：对应总收入；   pay：对应总支出；  \*/  Map<String, Object> datas = **new** HashMap<String, Object>();  datas.put("ledgerList", ledgerList);  datas.put("in", inMoney);  datas.put("pay", payMoney);    **return** datas;  }  } |

**实现LedgerMngController#queryLedger()方法**

1. 获取表单数据封装到QueryForm对象中；
2. 使用QueryForm对象调用LedgerService#query()方法，获取Map对象；
3. 把Map中数据显示到界面上：

* 把ledgerList显示到JTable中；
* 把in显示到总收入的JLabel上；
* 把pay显示到总支出的JLabel上。

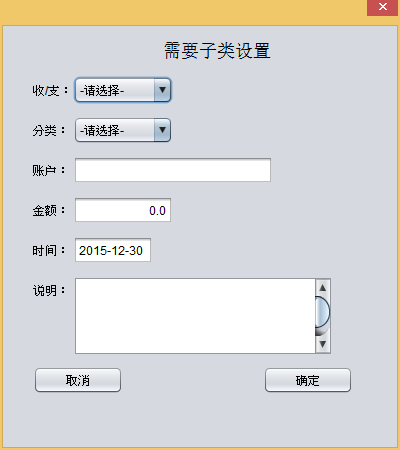
LedgerMngController.java

|  |
| --- |
| /\*\*  \* 账务管理窗口控制层类  \*/  **public** **class** LedgerMngController **extends** AbstractLedgerMngDialog {  **private** SortService sortService = **new** SortService();  **private** LedgerService ledgerService = **new** LedgerService();  …  @Override  **public** **void** queryLedger() {  //1.获取表单数据封装到QueryForm对象中；  QueryForm queryForm = **new** QueryForm();  queryForm.setBegin(beginDateTxt.getText());  queryForm.setEnd(endDateTxt.getText());  queryForm.setParent(parentBox.getSelectedItem().toString());  queryForm.setSname(sortBox.getSelectedItem().toString());  //2.使用QueryForm对象调用LedgerService#query()方法，获取Map对象；  Map<String, Object> datas = ledgerService.query(queryForm);  //3.把Map中数据显示到界面上  //把ledgerList显示到JTable中  List<Ledger> ledgerList = (List<Ledger>) datas.get("ledgerList");  **this**.setTableModel(ledgerList);  //把in显示到总收入的JLabel上  Double inMoneyTotal = (Double) datas.get("in");  inMoneyTotalLabel.setText("总收入：" + inMoneyTotal);  //把pay显示到总支出的JLabel上  Double payMoneyTotal = (Double) datas.get("pay");  payMoneyTotalLabel.setText("总支出：" + payMoneyTotal);  }  } |

### 添加账务

添加账务功能与添加分类功能比较相似，难度并不大。

当用户点击AddLedgerController对话框上的“确定”按钮时会调用AddLedgerController#confirm()方法，所以我们需要在confirm()方法中完成添加账务功能。



注意，用户在选择“收/支”后，对应的分类应该发生变化，所以我们首先需要完成这一功能。例如：当用户在“收/支”中选择“收入”，那么在分类下拉列表中显示所有收入分类。

当“收/支”下拉列表的值发生变化时，会调用AddLedgerController#changeParent()方法。所以我们需要在该方法中完成对“分类”下拉列表选项的修改。

#### 按条件查询分类，显示到下拉列表中

这一功能几乎与2.4.3中相同！

我们分析一下步骤：

1). 获取“收/支”下拉列表的值；

2). 使用父分类来调用SortService#querySortNameByParent()方法，查询出该父分类下所有子分类名称；

3). 把分类名称显示到下拉列表中。

AddLedgerController.java

|  |
| --- |
| /\*\*  \* 添加账务窗口控制层类  \*/  **public** **class** AddLedgerController **extends** AbstractOperationLedgerDialog {  **private** SortService sortService = **new** SortService();  **public** AddLedgerController(JDialog dialog) {  **super**(dialog);  }  @Override  **public** **void** changeParent() {  String parent = **this**.parentBox.getSelectedItem().toString();  **if**(parent.equals("-请选择-") ) {  Object[] sortItems = {"-请选择-"};  sortBox.setModel(**new** DefaultComboBoxModel(sortItems));  } **else** {  List<Object> nameItems = sortService.querySortNameByParent(parent);  nameItems.add(0, "-请选择-");  sortBox.setModel(**new** DefaultComboBoxModel(nameItems.toArray()));  }  }  …  } |

#### 添加账务功能

添加账务功能的实现分类：

　　　　1). 获取用户输入的数据，并校验；

　　　　2). 把数据封装到Ledger对象中，注意：用户输入的数据中只有sname而没有sid，所以我们需要使用sname来获取sid，这说明需要在DAO和Service层编写代码；

　　　　3). 调用LedgerService#addLedger()方法完成向数据库保存数据操作；

　　　　4). 关闭对话框。

AddLedgerController.java

|  |
| --- |
| **public** **class** AddLedgerController **extends** AbstractOperationLedgerDialog {    @Override  **public** **void** confirm() {  String parent = parentBox.getSelectedItem().toString();  **if**(parent.equals("-请选择-")) {  JOptionPane.*showMessageDialog*(**this**, "\"收/支\"不能为空！");  **return**;  }  String sname = sortBox.getSelectedItem().toString();  **if**(sname.equals("-请选择-")) {  JOptionPane.*showMessageDialog*(**this**, "\"分类\"不能为空！");  **return**;  }  String account = accountTxt.getText();  **if**(account.isEmpty()) {  JOptionPane.*showMessageDialog*(**this**, "\"账户\"不能为空！");  **return**;  }  **double** money = 0;  **try** {  money = Double.*parseDouble*(moneyTxt.getText());  } **catch**(NumberFormatException e) {  JOptionPane.*showMessageDialog*(**this**, "\"账户\"不能为空！");  **return**;  }  String createtime = createtimeTxt.getText();  String desc = ldescTxt.getText();  Ledger ledger = **new** Ledger();  ledger.setParent(parent);  ledger.setAccount(account);  ledger.setMoney(money);  ledger.setCreatetime(createtime);  ledger.setLdesc(desc);  **int** sid = sortService.getSidBySname(sname);  ledger.setSid(sid);  ledgerService.addLedger(ledger);  **this**.dispose();  }  } |

LedgerService.java

|  |
| --- |
| **public** **class** LedgerService {  **……**  **public** **void** addLedger(Ledger ledger) {  ledgerDao.addLedger(ledger);  }  } |

LedgerDao.java

|  |
| --- |
| **public** **class** LedgerDao {  **……**  **public** **void** addLedger(Ledger ledger) {  String sql = "INSERT INTO gjp\_ledger(parent,sid,account,money,createtime,ldesc) VALUES(?,?,?,?,?,?)";  Object[] params = {ledger.getParent(),ledger.getSid(),ledger.getAccount(),ledger.getMoney(),ledger.getCreatetime(),ledger.getLdesc()};  **try** {  qr.update(sql, params);  } **catch**(SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

#### 修改对话框标题

最后还要把对话框的标题修改一下。

这一工作可以放到构造器中完成：

* setTitle(“添加账务”);
* titleLabel.setText(“添加账务”);

AddLedgerController.java

|  |
| --- |
| **public** **class** AddLedgerController **extends** AbstractOperationLedgerDialog {  **……**  **public** AddLedgerController(JDialog dialog) {  **super**(dialog);  **this**.setTitle("添加账务");  titleLabel.setText("添加账务");  }  　　……  } |

### 编辑账务

编辑账务功能比添加账务复杂一点，首先是用户在表格中选中要编辑的账务数据，然后点击编辑按钮，这时我们需要获取用户选中的账务数据封装到Ledger对象中，并传递给对话框类，接下来对话框类要显示Ledger在表单中，再接下来用户对表单数据进行编辑，再点击确定按钮，这时我们要获取表单数据，并修改数据库中的数据，最后刷新表格数据。

#### 创建编辑账务对话框类（EditLedgerController）

因为对话框要显示用户选中的表格数据，所以要修改EditLedgerController类的构造器：

* 保存参数Ledger；
* 把参数Ledger显示到表单中；

EditLedgerController.java

|  |
| --- |
| **public** **class** EditLedgerController **extends** AbstractOperationLedgerDialog {  **private** Ledger tableLedger;    **public** EditLedgerController(JDialog dialog, Ledger ledger) {  **super**(dialog);  **this**.tableLedger = ledger;    **this**.setTitle("编辑账务");  titleLabel.setText("编辑账务");    parentBox.setSelectedItem(tableLedger.getParent());  sortBox.setSelectedItem(tableLedger.getSname());  accountTxt.setText(tableLedger.getAccount());  moneyTxt.setText(tableLedger.getMoney() + "");  createtimeTxt.setText(tableLedger.getCreatetime());  ldescTxt.setText(tableLedger.getLdesc());  }  ……  } |

#### LedgerMngController.java

1). 获取用户选中的表格数据，校验之，并封装到Ledger对象中；

* 获取用户选中的表格行号row；
* 判断row是否小于0，如果小于0说明用户没有选中表格数据，这时我们要弹出消息对话框，警告用户；
* 通过row获取对应的Ledger对象；
* 判断Ledger对象是否为null，如果为null说明用户选中的是空行，这时我们要弹出消息对话框，警告用户；

2). 创建编辑账务对话框，并传递Ledger对象；

3). 刷新表格数据，显示被编辑后的账务数据；

LedgerMngController.java

|  |
| --- |
| **public** **class** LedgerMngController **extends** AbstractLedgerMngDialog {  　　……  @Override  **public** **void** editLedger() {  **int** row = ledgerDataTable.getSelectedRow();  **if**(row < 0) {  JOptionPane.*showMessageDialog*(**null**, "请选择要编辑的账务数据！");  **return**;  }  Ledger ledger = **this**.getLedgerByTableRow(row);  **if**(ledger == **null**) {  JOptionPane.*showMessageDialog*(**null**, "您选择的是空格，请重新选择！");  **return**;  }  **new** EditLedgerController(**this**, ledger).setVisible(**true**);  **this**.queryLedger();  }  ……  } |

#### EditLedgerController.java

在EditLedgerController类中有两个方法要实现：

* changeParent ()：在父分类改变时更新子分类下拉列表，这个方法可以直接复制AddLedgerController#changeParent()方法；
* confirm()：在用户点击“确定”按钮时调用本方法，我们在该方法完成编辑账务对象功能，这个方法内容与AddLedgerController#confirm()基本相同，只有两个地方不同：
* 获取表单数据后，封装到Ledger对象中时：添加功能需要创建Ledger对象，而编辑功能无需创建，因为EditLedgerController类拥有tableLedger成员变量，所以我们只需要把表单数据封装到tableLedger对象中即可；
* 调用ledgerService方法时：添加功能调用的是LedgerService#addLedger()方法，而编辑功能调用的是LedgerService#editLedger()方法。

EditLedgerController.java

|  |
| --- |
| **public** **class** EditLedgerController **extends** AbstractOperationLedgerDialog {  **……**  @Override  **public** **void** confirm() {  String parent = parentBox.getSelectedItem().toString();  **if**(parent.equals("-请选择-")) {  JOptionPane.*showMessageDialog*(**this**, "\"收/支\"不能为空！");  **return**;  }  String sname = sortBox.getSelectedItem().toString();  **if**(sname.equals("-请选择-")) {  JOptionPane.*showMessageDialog*(**this**, "\"分类\"不能为空！");  **return**;  }  String account = accountTxt.getText();  **if**(account.isEmpty()) {  JOptionPane.*showMessageDialog*(**this**, "\"账户\"不能为空！");  **return**;  }  **double** money = 0;  **try** {  money = Double.*parseDouble*(moneyTxt.getText());  } **catch**(NumberFormatException e) {  JOptionPane.*showMessageDialog*(**this**, "\"账户\"不能为空！");  **return**;  }  String createtime = createtimeTxt.getText();  String desc = ldescTxt.getText();    tableLedger.setParent(parent);  tableLedger.setAccount(account);  tableLedger.setMoney(money);  tableLedger.setCreatetime(createtime);  tableLedger.setLdesc(desc);    **int** sid = sortService.getSidBySname(sname);  tableLedger.setSid(sid);  ledgerService.editLedger(tableLedger);    **this**.dispose();  }  } |

### 删除账务

用户要完成删除账务功能，首先需要在表格中选中要删除的行，点击“删除”按钮。这时我们需要获取用户点击的行号，再通过行号获取对象，最后调用LedgerService#deleteLedger()方法删除对象，最后是刷新表格。

1. 获取用户选中的表格数据，校验之，并封装到Ledger对象中；

1). 获取用户选中的表格行号row；

2). 判断row是否小于0，如果小于0说明用户没有选中表格数据，这时我们要弹出消息对话框，警告用户；

3). 通过row获取对应的Ledger对象；

4). 判断Ledger对象是否为null，如果为null说明用户选中的是空行，这时我们要弹出消息对话框，警告用户；

2. 调用LedgerService#deleteLedger()方法删除账务；

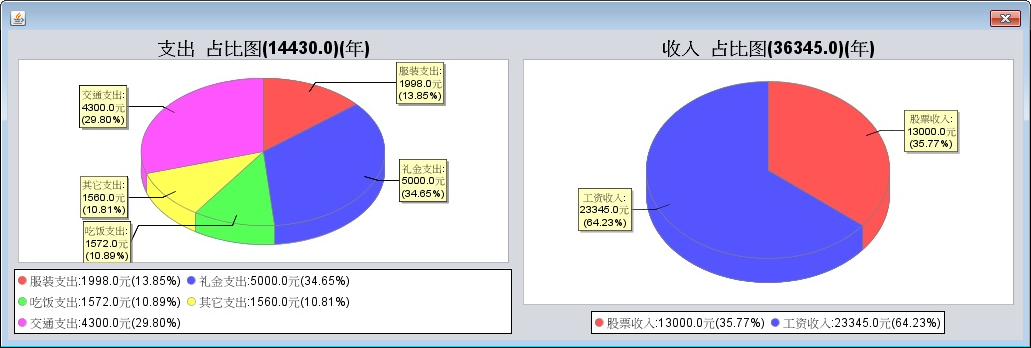
3. 刷新表格数据，显示被编辑后的账务数据；

注意，删除账务功能本应该已经算是完成了，但一般删除操作总是需要处理一下是否为误操作。就是在用户点击“删除”按钮后弹出确认对话框，询问用户是否真的要删除。

LedgerMngController.java

|  |
| --- |
| **public** **class** LedgerMngController **extends** AbstractLedgerMngDialog {  　　……  @Override  **public** **void** deleteLedger() {  **int** row = ledgerDataTable.getSelectedRow();  **if**(row < 0) {  JOptionPane.*showMessageDialog*(**null**, "请选择要编辑的账务数据！");  **return**;  }  Ledger ledger = **this**.getLedgerByTableRow(row);  **if**(ledger == **null**) {  JOptionPane.*showMessageDialog*(**null**, "您选择的是空格，请重新选择！");  **return**;  }    **int** m = JOptionPane.*showConfirmDialog*(**null**, "您是否真要删除该账务数据？");  **if**(m == JOptionPane.*OK\_OPTION*) {  /\*  \* 2.调用LedgerServic#deleteLedger()方法删除账务  \*/  ledgerService.deleteLedger(ledger);    /\*  \* 3. 对表格数据进行刷新（调用条件查询方法）  \*/  **this**.queryLedger();  }  }  　　……  } |

### 收支统计年度占比



实现这一功能需要使用JFreeChart技术，在本项目中提供了JFreeChartUtils类来简化JFreeChart的使用。在JFreeChartUtils类提供了pie()静态方法，这个方法有四个参数：

* String title：图表的标题，例如“支出 占比图（xxxx元）（年）”或“收入 占比图（xxxx元）（年）”，其中“xxxx元”表示年度总收入或总支出；
* Map<String,Double> totalMap：每种分类的汇总数据，其中key为分类名称，value为该分类的汇总金额；
* double sum：年度总支出或年度总收入，就是第1个参数String title中的金额；
* String path：生成的图片保存路径。

JFreeChartUtils. pie()方法会根据前3个参数生成图片，保存到第4个参数指定的路径下。而PieController#getImagePaths()方法返回的就是图片路径列表List<String>。

1. **PieController.java**

PieController#getImagePaths()方法返回List<String>表示多个图片的路径，这些图片会被显示到对话框中。也就是说，我们只需要在PieController#getImagePath()方法中调用JFreeChartUtils.pie()方法生成图片，并且把图片的路径返回即可。

JFreeChartUtils.pie()方法需要4个参数，其中第二个参数Map<String,Double>，以及第3个参数double需要通过LedgerService获取。

很明显，在PieController#getImagePaths()方法中需要返回两张图片的路径，一是收入总比图的图片，另一个是支出占比图的图片。

PieController.java

|  |
| --- |
| **public** **class** PieController **extends** AbstractShapeDialog {  **private** LedgerService ledgerService = **new** LedgerService();    **public** PieController(JDialog dialog) {  **super**(dialog);  initDialog();  }    @Override  **public** List<String> getImagePaths() {  Map<String, Double> zcSumMoneyMap = ledgerService.querySumMoneyBySort("支出");  **double** zcTotalMoney = ledgerService.queryTotalMoneyByParent("支出");  String zcTitle = "支出占比图(" + zcTotalMoney + "元)(年)";  String zcPath = "jfreechart\_img/zc.jpg";  JFreeChartUtils.*pie*(zcTitle, zcSumMoneyMap, zcTotalMoney, zcPath);    Map<String, Double> srSumMoneyMap = ledgerService.querySumMoneyBySort("收入");  **double** srTotalMoney = ledgerService.queryTotalMoneyByParent("收入");  String srTitle = "收入占比图(" + srTotalMoney + "元)(年)";  String srPath = "jfreechart\_img/sr.jpg";  JFreeChartUtils.*pie*(srTitle, srSumMoneyMap, srTotalMoney, srPath);    List<String> paths = **new** ArrayList<String>();  paths.add(zcPath);  paths.add(srPath);  **return** paths;  }  } |

1. **LedgerService.java**

LedgerService类中需要提供两个方法：

* Map<String,Double> querySumMoneyBySort(String parent)：该方法返回每种分类的年度汇总数据，返回Map的key为分类名称，value为对应分类的汇总数；
* double queryTotalMoneyByParent(String parent)：该方法返回年度总支出或总收入；

其中LedgerService #querySumbySort()方法需要先使用LedgerDao#querySumMoneyBySort()方法获取List<Object[]>类型数据，其中每个Object[]有两个元素：Integer sid和Double sumMoney。但LedgerService #querySumbySort()方法返回的Map类的key是String sname，而不是Integer sid，所以我们需要通过sid查询出对应的sname。

LedgerService.java

|  |
| --- |
| **public** **class** LedgerService {  **……**  **public** Map<String,Double> querySumMoneyBySort(String parent) {  List<Object[]> objArrayList =  ledgerDao.querySumMoneyBySort(parent);  Map<String,Double> sumMoneyMap = **new** HashMap<String,Double>();  **for**(Object[] objArray : objArrayList) {  **int** sid = (Integer)objArray[0];  **double** sumMoney = (Double)objArray[1];    String sname = sortDao.getSnameBySid(sid);  sumMoneyMap.put(sname, sumMoney);  }  **return** sumMoneyMap;  }    **public** **double** queryTotalMoneyByParent(String parent) {  **return** ledgerDao.getTotalMoney(parent);  }  } |

1. **LedgerDao.java**

在LedgerDao类中需要提供两个方法：querySumMoneyBySort()和queryTotalMoney()。这两个方法中前者需要使用分组查询及聚合函数；后者需要使用聚合函数。

LedgerDao.java

|  |
| --- |
| **public** **class** LedgerDao {  **……**    **public** List<Object[]> querySumMoneyBySort(String parent) {  String sql = "SELECT sid, SUM(money) total FROM gjp\_ledger" +  " WHERE createtime LIKE ? AND parent=?" +  " GROUP BY sid";  Object[] params = {DateUtils.*getYear*() + "%", parent};  **try** {  **return** qr.query(sql, **new** ArrayListHandler(), params);  } **catch**(SQLException e) {  **throw** **new** RuntimeException(e);  }  }    **public** Double getTotalMoney(String parent) {  String sql = "SELECT SUM(money) total FROM gjp\_ledger" +  " WHERE createtime LIKE ? AND parent=?";  Object[] params = {DateUtils.*getYear*() + "%", parent};  **try** {  **return** (Double)qr.query(sql, **new** ScalarHandler(), params);  } **catch**(SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

### 删除分类功能补充

删除分类功能还存在一个小小的问题！

账务表中的记录与分类表中的记录是有关联的，例如某条账务记录关联了“股票收入”分类，如果这时我们删除了“股票收入”分类，那么这条账务还继续关联“股票收入”就不合适了，通常我们称之为“数据完整性”存在问题了。

处理这一问题的办法有两种：

* 在用户删除分类时，需要先查看该分类下是否存在账务，如果存在就报错；
* 在用户删除分类时，需要先查看该分类下是否存在账务，如果存在就先删除账务，再删除分类（级联删除）；

在这里我们选择报错！

我们需要在SortMngController#deleteSort()方法中添加一些代码：

SortMngController#deleteSort()

|  |
| --- |
| @Override  **public** **void** deleteSort() {  **int** row = **this**.sortDataTable.getSelectedRow();  **if**(row < 0) {  JOptionPane.*showMessageDialog*(**this**, "请选择要编辑的分类！");  **return**;  }  Sort sort = **this**.getSortByTableRow(row);  **if**(sort == **null**) {  JOptionPane.*showMessageDialog*(**this**, "您选择的是空行！");  **return**;  }    **int** m = JOptionPane.*showConfirmDialog*(**this**, "您是否确定要删除该行记录？");  **if**(m == JOptionPane.*OK\_OPTION*) {  **int** cnt = ledgerService.getLedgerCountBySid(sort.getSid());  **if**(cnt > 0) {  JOptionPane.*showMessageDialog*(**this**, "该分类下还存在账务记录，不能删除！");  **return**;  }  sortService.deleteSort(sort);  List<Sort> sortList = sortService.querySortAll();  **this**.setTableModel(sortList);  }  } |

　　这里省略LedgerService#getLedgerCountBySid()和LedgerDao#getLedgerCountBySid()代码。因为这时你应该已经具备了自行完成的能力！