

灯具进销存系统（JXCS）

灯具进销存系统（JXCS）软件详细设计描述文档



2014-11-3

南京大学仙林校区软件学院ROOM404

小队成员：金鑫

李斐

梁思宇

周宇宸

目录

[0.更新历史 2](#_Toc407649419)

[1.引言 2](#_Toc407649420)

[2.产品概述 2](#_Toc407649421)

[3.体系结构设计概述 2](#_Toc407649422)

[4.结构视角 3](#_Toc407649423)

[4.1 客户端展示层的分解 3](#_Toc407649424)

[4.1.1客户端展示层模块的静态结构和动态职责 3](#_Toc407649425)

[4.1.2客户端展示层模块局部模块的接口规范 3](#_Toc407649426)

[4.1.3客户端展示层模块局部模块的设计原理 5](#_Toc407649427)

[4.2业务逻辑层的分解 5](#_Toc407649428)

[4.2.1 commoditybl模块 6](#_Toc407649429)

[4.2.2 accountbl模块 16](#_Toc407649430)

[4.2.3 financialbl模块 26](#_Toc407649431)

[4.2.4 initializationbl 模块 31](#_Toc407649432)

[4.1.5 invoice 模块 35](#_Toc407649433)

[4.2.6 promotion 模块 39](#_Toc407649434)

[4.2.7 salesbl模块 44](#_Toc407649435)

[4.2.8 systemlogbl模块 61](#_Toc407649436)

[4.2.9 userbl模块 64](#_Toc407649437)

[4.3服务器端数据层分解 69](#_Toc407649438)

[4.3.1 数据层模块的描述 69](#_Toc407649439)

[4.3.2数据层模块的分解 69](#_Toc407649440)

[5.依赖视角 89](#_Toc407649441)

# 0.更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| **更新日期** | **更新说明** | **更新人员** | **版本号** |
| 2014-11-03 | 编写第一版软件详细设计描述文档 | 金鑫 | V1.0 |
| 2014-11-15 | 更新模块接口信息 | 周宇宸 | V2.0 |
| 2014-11-16 | 修改一些类的职能 | 周宇宸 | V2.1 |

# 1.引言

1.1编制目的

本报告详细完成对灯具公司进销存系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
| JXCS | 进销存系统 |  |

1.3参考资料

# 2.产品概述

参考灯具公司进销存系统用例文档和灯具公司进销存系统软件需求规格说明文档中对产品的概括描述。

# 3.体系结构设计概述

参考灯具公司进销存系统概要设计文档中对体系结构设计的概述。

# 4.结构视角

## 4.1 客户端展示层的分解

### 4.1.1客户端展示层模块的静态结构和动态职责

客户端展示层由六个模块组成，分别是ui, ui.commodity, ui.admin, ui.account, ui.manager, ui.sales。ui模块负责另外五个模块的界面的总跳转；ui.admin模块包含用户登录和修改密码界面,其中AdminAllUIController负责本模块模块内界面跳转。Ui.account模块包括财务人员职能相关界面，包括收付款单界面，单据界面等，其中AccountAllUIController负责本模块内界面跳转。Ui.manager模块包括总经理职能相关界面，包括促销策略的指定，审批单据等，其中ManagerAllUIController负责控制本模块内界面的跳转。Ui.sales模块包括销售人员职能相关的界面，包括进货退货，销售等，其中SalesUIControler负责本模块内界面的跳转。

客户端展示层模块的职责如下表所示。

**客户端展示层模块的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Ui | 负责界面模块的跳转 |
| Ui.admin | 包含管理员职能界面 |
| Ui.account | 包含财务人员职能界面 |
| Ui.sales | 包含销售人员界面 |
| Ui.commodity | 包含库存人员界面 |
| Ui.manager | 包含总经理界面 |

### 4.1.2客户端展示层模块局部模块的接口规范

#### 4.1.2.1 ui

负责commodity，sales，account，manager，admin五个模块的控制跳转

#### 4.1.2.2 ui.commodity

**Ui.commodity模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| businesslogicservice.commodityblservice | 用于商品管理，分类管理，库存管理业务逻辑接口 |
| businesslogicservice.invoiceblservice | 用于单据审批业务逻辑接口（报溢报损单） |

#### 4.1.2.3 ui.sales

**Ui.sales模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| businesslogicservice.commodityblservice | 用于获得商品，更改商品库存的的业务逻辑接口 |
| businesslogicservice.invoiceblservice | 用于单据审批业务逻辑接口（销售单，销售退货单，进货单，进货退货单） |
| businesslogicservice.salesblservice | 用于进货，销售，进货退货，销售退货，客户管理的业务逻辑接口 |
| businesslogicservice.systemlogblservice | 用于添加系统日志的业务逻辑接口 |
| businesslogicservice.promotionblservice | 用于得到促销策略的业务逻辑接口 |
| businesslogicservice.financialblservice | 用于添加经营历程表，销售明细表，经营情况表的业务逻辑接口 |

#### 4.1.2.4 ui.account

**Ui.account模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| businesslogicservice.invoiceblservice | 用于单据审批业务逻辑接口（收款单，付款单） |
| businesslogicservice.salesblservice | 用于获得客户的业务逻辑接口 |
| businesslogicservice.systemlogblservice | 用于获得和添加系统日志的业务逻辑接口 |
| businesslogicservice.accountblservice | 用于账户管理的业务逻辑接口（仅显最高权限） |
| businesslogicservice.financialblservice | 用于获得经营历程表，销售明细表，经营情况表的业务逻辑接口 |

#### 4.1.2.5 ui.manager

**Ui.manager模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| businesslogicservice.invoiceblservice | 用于单据审批业务逻辑接口 |
| businesslogicservice.systemlogblservice | 用于获得和添加系统日志的业务逻辑接口 |
| businesslogicservice.promotionblservice | 用于指定促销策略的业务逻辑接口 |
| businesslogicservice.financialblservice | 用于获得经营历程表，销售明细表，经营情况表的业务逻辑接口 |
| businesslogicservice.accountblservice | 用于账户管理的业务逻辑接口 |

#### 4.1.2.6 ui.admin

**Ui.admin模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| businesslogicservice.systemlogblservice | 用于添加系统日志的业务逻辑接口 |
| businesslogicservice.adminblservice | 用于用户管理的业务逻辑接口 |

### 4.1.3客户端展示层模块局部模块的设计原理

客户端展示层模块使用了集中式的控制风格。每个界面模块都通过ui模块实现跳转。

在每一个模块的内部都有各自的controller对模块内部的界面进行集中控制跳转.

## 4.2业务逻辑层的分解

业务逻辑层的开发包图参见软件体系结构文档图3。

业务逻辑层层模块的职责如下表所示。

**表12 客户端业务逻辑层模块的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| commoditybl | 负责实现库存管理需要的服务 |
| salesbl | 负责实现销售管理所需要的服务 |
| promotionbl | 负责实现促销所需要的服务 |
| accountbl | 负责实现账户管理所需要的服务 |
| financialbl | 负责实现经营历程表，销售明细表，经营情况表所需要的服务 |
| systemlogbl | 负责实现所需要的服务 |
| adminbl | 负责实现用户管理所需要的服务 |
| Invoicebl | 负责实现审批所需要的服务 |
| Initializationbl | 负责实现期初建账所需要的服务 |

### 4.2.1 commoditybl模块

(1)模块概述

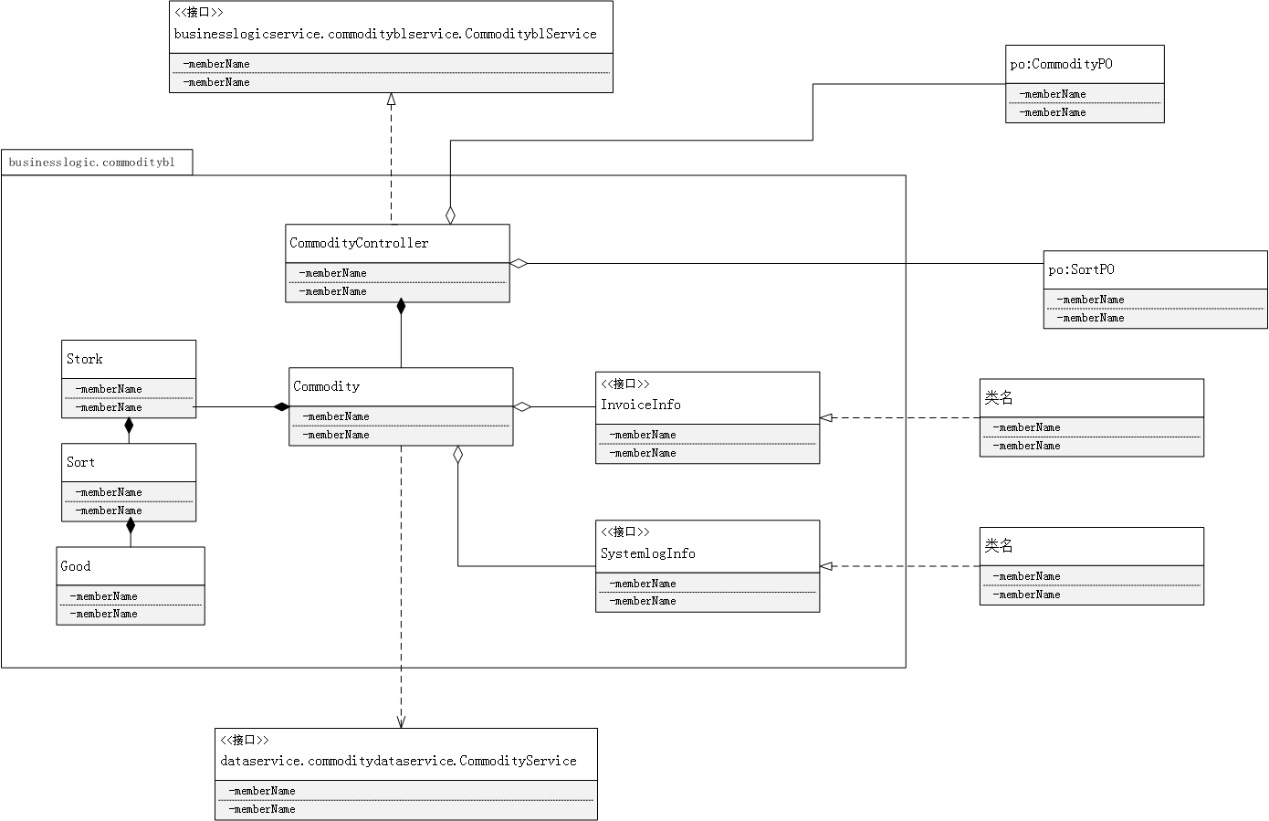
Commoditybl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Commoditybl模块的职责及借款参见软件体系结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。Commodity模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.commodityblservice.CommodityblService接口与业务逻辑层与数据层之间的dataservice.commoditydataservice.CommodityblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommodityController，这样CommodityController会将对商品与库存的业务逻辑处理委托给Commodity对象。CommodityPO是作为商品的持久化对象被添加到设计模型中去的。SortPO是作为商品种类的持久化对象被添加到设计模型中去的。而Stork保存有简单的库存信息，完成搜索与判断重复的逻辑操作，管理Sort对象。Sort保存有简单的商品信息，负责商品的查找。Good是简单的商品对象，拥有基本的商品信息。SystemlogInfo、InvoiceInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Commoditybl模块的设计如图



Commoditybl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| CommodityController | 负责实现商品界面所需要的服务 |
| Commodity | 库存领域模型对象，拥有库存与商品的信息，可以完成库存管理与商品管理操作 |
| Stork | 临时管理库存中商品与分类，保存最基本的商品与分类信息 |
| Sort | 商品分类，临时保存最基本的商品信息 |
| Good | 商品，保存有商品的名称与型号 |

(3)模块内部类的接口规范

CommodityController接口规范:

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| CommodityController.addCommodity | 语法 | Public int addCommodity\_up  (CommodityVOvo1,SortVO vo2); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的addCommodity方法 |
| CommodityController.delCommodity | 语法 | Public int delCommodity\_up  (CommodityVO vo); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的delCommodity方法 |
| CommodityController.updateCommodity | 语法 | public int updateCommodity\_up  (CommodityVO vo1,CommodityVO vo2);//更新商品信息，用vo2的信息替换vo1  public int updateCommodity\_up  (CommodityVO vo1,SortVO vo2);//修改商品所属的分类信息 |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的updateCommodity方法 |
| CommodityController.searchCommodity | 语法 | //模糊查找商品  Public ArrayList<CommodityVO> searchFuzzyCommodity\_up(String word);  //精确查找商品  Public CommodityVO  searchAccurateCommodity\_up(String name,String type); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的searchCommodity方法 |
| CommodityController.getCommodityNote | 语法 | public String getCommodityNote\_up(SortVO vo1) |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getCommodityNote方法 |
| CommodityController.addSort | 语法 | public int addSort\_up(SortVO vo1, SortVO vo2);//vo1为子类，vo2为父类 |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的addSort方法 |
| CommodityController.delSort | 语法 | public int delSort\_up(SortVO vo); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的delSort方法 |
| CommodityController.updateSort | 语法 | //修改分类信息，用vo2的信息替代vo1  Public int updateSort\_up\_Inf(SortVO vo1, SortVO vo2);  //将分类移至另一分类下，将vo1的信息移到vo2下  public int  updateSort\_up\_Mov(SortVO vo1,SortVO vo2); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的updateSort方法 |
| CommodityController.getSortNote | 语法 | public String getSortNote\_up(SortVO vo1); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getSortNote方法 |
| CommodityController.Examine | 语法 | public ArrayList<ExamineVO> Examine\_up(String time1, String time2); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的Examine方法 |
| CommodityController.patch | 语法 | public int patch\_up(PatchVO vo) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的patch方法 |
| CommodityController.patchDraft | 语法 | public int patchDraft\_up(PatchVO vo); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的patchDraft方法 |
| CommodityController.getAllDraftPatch | 语法 | public ArrayList<PatchVO> getAllDraftPatch\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getAllDraftPatch方法 |
| CommodityController.getAllPatch | 语法 | public ArrayList<PatchVO> getAllPatch\_up() ; |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getAllPatch方法 |
| CommodityController.getPatchNote | 语法 | public String getPatchNote(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getPatchNote方法 |
| CommodityController.searchDraftPatch | 语法 | public PatchVO searchDraftPatch\_up(String note); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的searchDraftPatch方法 |
| CommodityController.searchPatch | 语法 | public PatchVO searchPatch\_up(String note); |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的searchPatch方法 |
| CommodityController.getAllWarnGood | 语法 | public ArrayList<CommodityVO> getAllWarnGood\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getAllWarnGood方法 |
| CommodityController.getAllSort | 语法 | public ArrayList<SortVO> getAllSort\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getAllSort方法 |
| CommodityController.getComSort | 语法 | public ArrayList<SortVO> getComSort\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getComSort方法 |
| CommodityController.getSortSort | 语法 | public ArrayList<SortVO> getSortSort\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getSortSort方法 |
| Commodity.getAllCommodity | 语法 | public ArrayList<CommodityVO> getAllCommodity\_up(); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的getAllCommodity方法 |
| Commodity.warn | 语法 | public int warn\_up(WarnVO vo); |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的warn方法 |

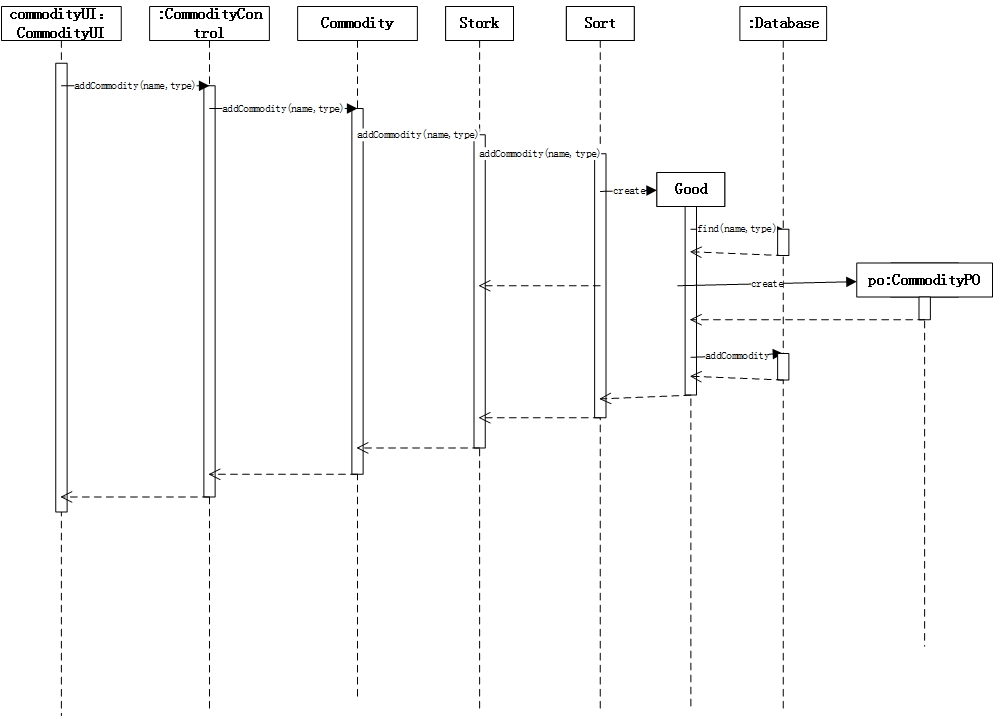
|  |  |
| --- | --- |
| 需要的服务 | |
| Commodity. addCommodity  (CommodityVOvo1,SortVO vo2) | 添加商品 |
| Commodity. delCommodity  (CommodityVO vo); | 删除商品 |
| Commodity. updateCommodity  (CommodityVO vo1,CommodityVO vo2) | 更新商品信息 |
| Commodity. updateCommodity  (CommodityVO vo1,SortVO vo2) | 更新商品所属分类 |
| Commodity. searchFuzzyCommodity(String word) | 模糊查找商品 |
| Commodity.searchAccurateCommodity(String name,String type) | 精确查找商品 |
| Commodity. String getCommodityNote(SortVO vo1) | 获得商品编号 |
| Commodity. addSort(SortVO vo1, SortVO vo2); | 添加分类 |
| Commodity. delSort(SortVO vo) | 删除分类 |
| Commodity. getSortNote(SortVO vo1) | 获得分类编号 |
| Commodity. Examine(String time1, String time2) | 库存查看 |
| Commodity. patch(PatchVO vo) | 添加库存报溢报损单 |
| Commodity. patchDraft(PatchVO vo); | 添加库存报溢报损草稿单 |
| Commodity. getAllDraftPatch(); | 获得所有库存报溢报损草稿单 |
| Commodity. getAllPatch() | 获得所有库存报溢报损单 |
| Commodity. getPatchNote() | 获得报溢报损单编号 |
| Commodity. searchDraftPatch(String note) | 根据报溢报损草稿单编号查询报溢报损草稿单 |
| Commodity. searchPatch(String note); | 根据报溢报损单编号查询报溢报损单 |
| Commodity. getAllWarnGood(); | 获得所有报警商品信息 |
| Commodity. getAllSort() | 获得所有商品分类信息 |
| Commodity. getComSort() | 获得所有可以添加商品的分类 |
| Commodity. getSortSort(); | 获得所有可以添加分类的分类 |
| Commodity. getAllCommodity(); | 获得所有的商品 |
| Commodity. warn(WarnVO vo); | 增加一个库存报警单 |

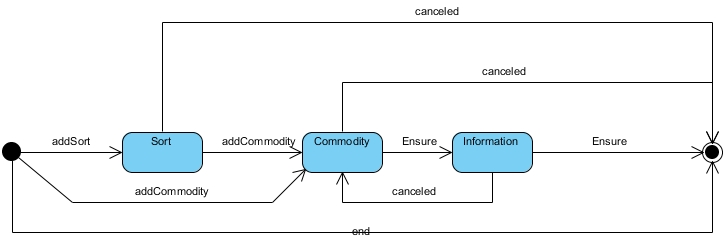
‘

**Commodity的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Commodity.addCommodity | 语法 | Public int addCommodity\_up  (CommodityVOvo1,SortVO vo2); |
| 前置条件 | 系统启动添加商品，且输入合法。 |
| 后置条件 | 添加商品，返回是否成功，返回0为成功，否则为失败。 |
| Commodity.delCommodity | 语法 | Public int delCommodity\_up  (CommodityVO vo); |
| 前置条件 | 系统启动删除商品，且输入合法。 |
| 后置条件 | 删除商品，返回是否成功，返回0为成功，否则为失败。 |
| Commodity.updateCommodity | 语法 | public int updateCommodity\_up  (CommodityVO vo1,CommodityVO vo2);//更新商品信息，用vo2的信息替换vo1  public int updateCommodity\_up  (CommodityVO vo1,SortVO vo2);//修改商品所属的分类信息 |
| 前置条件 | 系统启动修改商品信息，且输入合法。 |
| 后置条件 | 更新商品，返回是否更新成功，返回0为成功，否则为失败。 |
| Commodity.searchCommodity | 语法 | //模糊查找商品  Public ArrayList<CommodityVO> searchFuzzyCommodity\_up(String word);  //精确查找商品  Public CommodityVO  searchAccurateCommodity\_up(String name,String type); |
| 前置条件 | 无 |
| 后置条件 | 返回搜索结果，模糊查找返回一个CommodityVO类型的arrayList，精确查找返回查找的商品的CommodityVO，若未查找到相关信息，返回空的arrayList或commodityVO |
| Commodity.getCommodityNote | 语法 | public String getCommodityNote\_up(SortVO vo1) |
| 前置条件 | 系统启动添加商品，并且输入合法 |
| 后置条件 | 系统根据分类返回单据编号 |
| Commodity.addSort | 语法 | public int addSort\_up(SortVO vo1, SortVO vo2);//vo1为子类，vo2为父类 |
| 前置条件 | 系统启动添加分类，并且输入合法 |
| 后置条件 | 添加分类，返回是否添加成功，返回0为成功，否则为失败。 |
| Commodity.delSort | 语法 | public int delSort\_up(SortVO vo); |
| 前置条件 | 系统启动删除分类，且输入合法 |
| 后置条件 | 删除分类，返回是否删除成功，返回0为成功，否则为失败。 |
| Commodity.updateSort | 语法 | //修改分类信息，用vo2的信息替代vo1  Public int updateSort\_up\_Inf(SortVO vo1, SortVO vo2);  //将分类移至另一分类下，将vo1的信息移到vo2下  public int  updateSort\_up\_Mov(SortVO vo1,SortVO vo2); |
| 前置条件 | 系统启动更新分类信息，且输入合法 |
| 后置条件 | 修改分类，返回是否修改成功，返回0为成功，否则为失败 |
| Commodity.getSortNote | 语法 | public String getSortNote\_up(SortVO vo1); |
| 前置条件 | 系统启动添加分类 |
| 后置条件 | 系统根据分类的父类返回一个分类编号 |
| Commodity.Examine | 语法 | public ArrayList<ExamineVO> Examine\_up(String time1, String time2); |
| 前置条件 | 系统启动库存查看，且输入的时间格式正确 |
| 后置条件 | 返回查看结果 |
| Commodity.patch | 语法 | public int patch\_up(PatchVO vo) |
| 前置条件 | 系统启动添加报溢报损单，且输入合法 |
| 后置条件 | 生成报溢报损单，返回是否成功，返回0为成功，否则失败 |
| Commodity.patchDraft | 语法 | public int patchDraft\_up(PatchVO vo); |
| 前置条件 | 系统启动添加报溢报损单，且输入合法，用户选择保存单据 |
| 后置条件 | 生成草稿报溢报损单，返回是否成功，返回0为成功，否则失败 |
| Commodity.getAllDraftPatch | 语法 | public ArrayList<PatchVO> getAllDraftPatch\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有草稿报溢报损单 |
| Commodity.getAllPatch | 语法 | public ArrayList<PatchVO> getAllPatch\_up() ; |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有报溢报损单 |
| Commodity.getPatchNote | 语法 | public String getPatchNote(); |
| 前置条件 | 系统启动添加报溢报损单 |
| 后置条件 | 系统返回一个报溢报损单编号 |
| Commodity.searchDraftPatch | 语法 | public PatchVO searchDraftPatch\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 根据编号查询并返回一个草稿报溢报损单 |
| Commodity.searchPatch | 语法 | public PatchVO searchPatch\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 根据编号查询并返回一个报溢报损单 |
| Commodity.getAllWarnGood | 语法 | public ArrayList<CommodityVO> getAllWarnGood\_up(); |
| 前置条件 | 无 |
| 后置条件 | 获得所有报警商品信息 |
| Commodity.getAllSort | 语法 | public ArrayList<SortVO> getAllSort\_up(); |
| 前置条件 | 无 |
| 后置条件 | 获得所有分类信息 |
| Commodity.getComSort | 语法 | public ArrayList<SortVO> getComSort\_up(); |
| 前置条件 | 添加商品 |
| 后置条件 | 系统返回所有可以添加商品的分类 |
| Commodity.getSortSort | 语法 | public ArrayList<SortVO> getSortSort\_up(); |
| 前置条件 | 添加分类 |
| 后置条件 | 系统返回所有可以添加分类的分类 |
| Commodity.getAllCommodity | 语法 | public ArrayList<CommodityVO> getAllCommodity\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回所有商品信息 |
| Commodity.warn | 语法 | public int warn\_up(WarnVO vo); |
| 前置条件 | 库存存在报警 |
| 后置条件 | 系统自动生成一个报溢报损单，并返回是否成功，回0为成功，否则为失败 |
| 需要的服务 | | |
| 服务名 | 服务 | |
| CommodityDataService.findGood(String name, String type) | 根据name和type查找精确商品对象 | |
| CommodityDataService.addGood(CommodityPO po1, SortPO po2) | 在分类中添加商品对象 | |
| CommodityDataService.delGood(CommodityPO po) | 删除商品对象 | |
| ComodityDataService.getGoodNote(SortPO po1) | 根据商品所属分类获得商品对象的编号 | |
| CommodityDataService.updateGood(CommodityPO po1, CommodityPO po2) | 更新商品对象信息 | |
| CommodityDataService.updateGood(CommodityPO po1,SortPO po2) | 更新商品对象所属分类 | |
| CommodityDataService.addSort(SortPO po1，SortPO po2) | 添加分类对象 | |
| CommodityDataService.delSort(SortPO po) | 删除分类对象 | |
| CommodityDataService.updateSort(SortPO po1, SortPO po2) | 更新分类对象信息 | |
| CommodityDataService.updateSort\_MOV(SortPO po1,SortPO po2) | 更新分类对象所属分类 | |
| CommodityDataService.getSortNote(SortPO po1) | 根据分类所属父类获得分类编号 | |
| CommodityDataService.addSendGift(SendGiftPO po) | 添加赠品对象 | |
| CommodityDataService.findSendGift(SendGiftPO po) | 查找赠品对象 | |
| CommodityDataService.findSendGift(String note) |  | |
| CommodityDataService.delSendGift(SendGiftPO po) | 删除赠品对象 | |
| CommodityDataService.delSendGift(String note) |  | |
| CommodityDataService.getSendGiftNote() | 获得赠送商品的编号 | |
| CommodityDataService.getAllSendGift() | 获得当前所有赠送商品的信息 | |
| CommodityDataService.getAll() | 获得所有商品对象 | |
| CommodityDataService.getAllSort() | 获得所有商品分类对象 | |
| CommodityDataService.addWarn(WarnPO po) | 添加库存报警单对象 | |
| CommodityDataService.addPatch(PatchPO po) | 添加报溢报损单对象 | |
| CommodityDataService.delPatch(String note) | 根据报溢报损单编号查询到相应单据并删除报溢报损单对象 | |
| CommodityDataService.getPatch(String note) | 根据报溢报损单编号查询到单一报溢报损单对象 | |
| CommodityDataService.getAllPatch() | 获得当前所有报溢报损单对象 | |
| CommodityDataService.getPatchNote() | 获得报溢报损单编号 | |
| CommodityDataService.passPatch(PatchPO po) | 系统根据审批通过的报溢报损单信息修改库存信息 | |
| CommodityDataService.refusePatch(PatchPO po) | 系统根据审批拒绝的报溢报损单信息修改单据信息 | |
| CommodityDataService.passSendGift(SendGiftPO po) | 系统根据审批通过的赠送单信息修改库存信息 | |
| CommodityDataService.refuseSendGift(SendGift po) | 系统根据审批拒绝的报溢报损单信息修改单据信息 | |
| CommodityDataService.passImport(Import po) | 系统根据审批通过的进货单信息修改库存信息 | |
| CommodityDataService.passImport\_Return(Import\_Return po) | 系统根据审批通过的今后退货单修改库存信息 | |
| CommodityDateService.passExport\_Return(Export\_Return po) | 系统根据审批通过的销售退货单修改库存信息 | |
| CommodityDataService.passExport(Export po) | 系统根据审批通过的销售单修改库存信息 | |
| CommodityDataService.addDraftPatch(PatchPO po) | 添加一个报溢报损草稿单 | |
| CommodityDataService.delDraftPatch(String note) | 根据单据编号查找并删除一个报溢报损草稿单 | |
| CommodityDataService.getDraftPatch(String note) | 根据单据编号查找到一个报溢报损单 | |
| CommodityDataService.getAllDraftPatch() | 系统返回所有报溢报损单 | |
| CommodityDataService.addGoodNumber(CommodityPO po,int number) |  | |
| CommodityDataService.clear() | 清空所有对象 | |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.2 accountbl模块

(1)模块概述

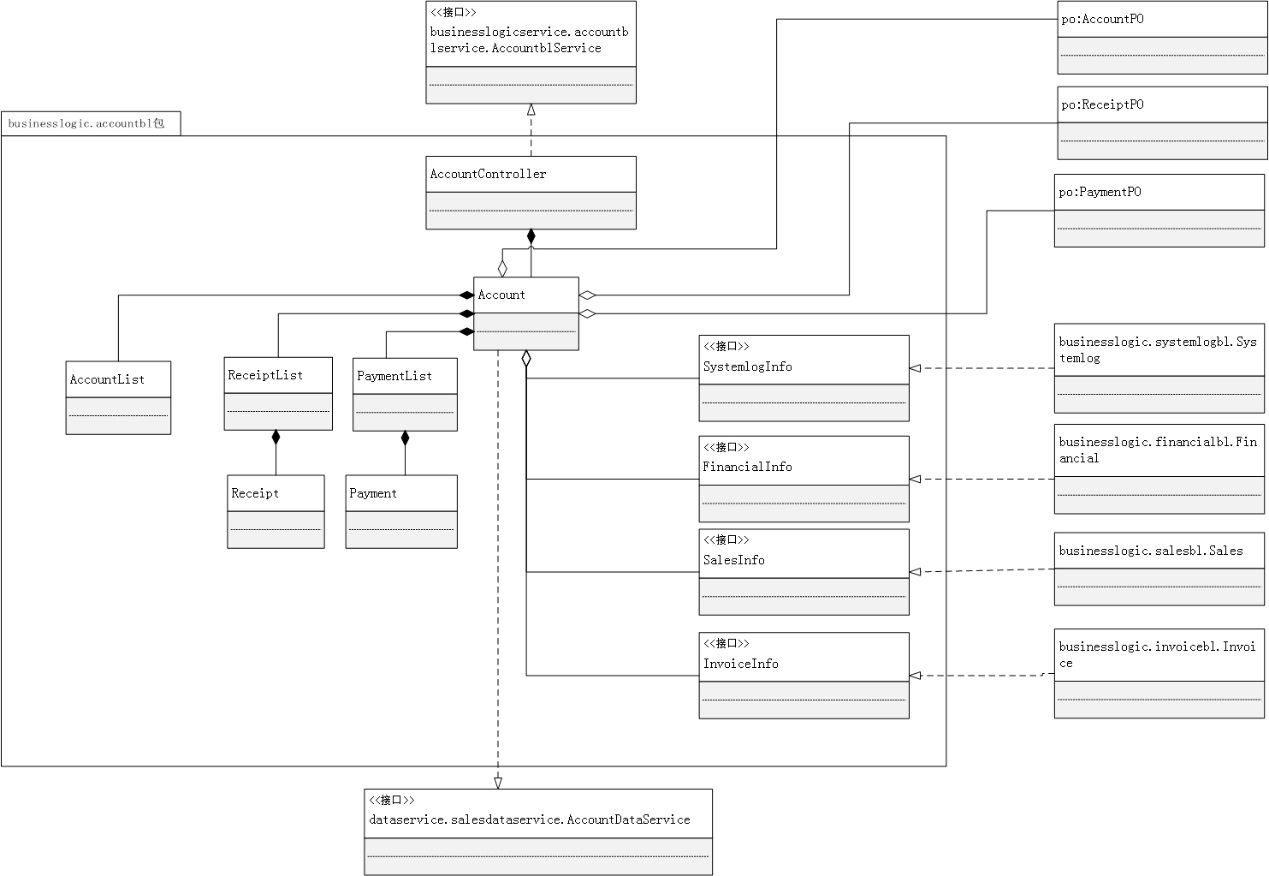
Accountbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Accountbl模块的职责及借款参见软件体系结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。Account模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.accountblservice.AccountblService接口与业务逻辑层与数据层之间的dataservice.accountdataservice.AccountblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountController，这样AccountController会将对账户的业务逻辑处理委托给Account对象。AccountPO是作为银行账户的持久化对象被添加到设计模型中去的。ReceiptPO是作为收款单的持久化对象被添加到设计模型中去的。PaymentPO是作为付款单的持久化对象被添加到设计模型中去的。而Account保存简单的账户信息，Receipt为收款的转账列表中的一项，ReceiptList负责管理Receipt，并计算收款总额。Payment为付款的条目清单中的一项。PaymentList负责管理Payment，并计算付款总额。SystemlogInfo、FinancialInfo、SalesInfo、InvoiceInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Accountbl模块的设计如图



Accountbl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountController | 负责实现账户界面所需要的服务 |
| Account | 收付款以账户管理领域模型对象，拥有一次收款、付款以及账户等信息，可以帮助完成账户界面所需要的服务 |
| ReceiptList | 管理此次操作中的收款的转账列表，完成计算收款总额操作 |
| Receipt | 收款操作中的转账列表的一项 |
| PaymentList | 管理此次操作中的付款的条目清单，完成计算付款总额的操作 |
| Payment | 付款操作中的条目清单的一项 |

(3)模块内部类的接口规范

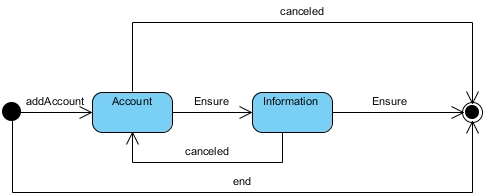
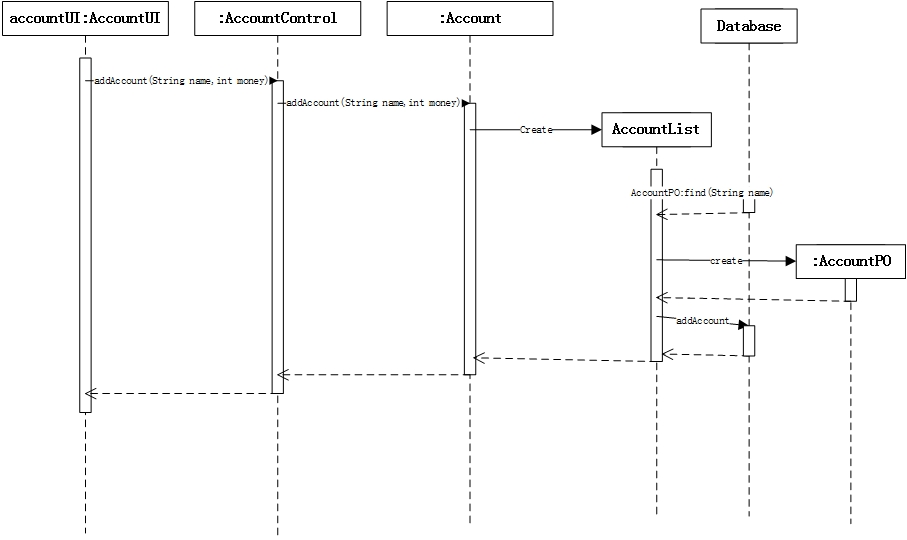
**AccountController接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| AccountController.checkAccount() | 语法 | | Public AccountVO checkAccount\_up() |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的checkAccount方法 |
| AccountController.addAccount | 语法 | | Public int addAccount\_up(AccountVO vo) |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的addAccount方法 |
| AccountController.delAccount | 语法 | | Public int delAccount\_up(AccountVO vo) |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的delAccount方法 |
| AccountController.updateAccount | 语法 | | Public int updateAccount\_up(AccountVO vo1,AccountVO vo2) |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的updateAccount方法 |
| AccountController.searchAccount | 语法 | | //模糊查找账户  public ArrayList<AccountVO> searchFuzzyAccount\_up(String word);    //精确查找账户  public AccountVO searchAccurateAccount\_up(String name); |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的searchAccount方法 |
| AccountController.addReceipt | 语法 | | public int addReceipt\_up(GetVO vo); |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的addReceipt方法 |
| AccountController.addPayment | 语法 | | public int addPayment\_up(PayVO vo); |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的addPayment方法 |
| AccountController.getAllReceipt | 语法 | | Public ArrayList<GetVO> getAllReceipt\_up(); |
| 前置条件 | | 已创建一个Account领域对象 |
| 后置条件 | | 调用Account领域对象的getAllReceipt方法 |
| AccountController.getAllPayment | 语法 | | public ArrayList<PayVO> getAllPayment\_up(); |
| 前置条件 | | 已创建一个Account领域对象 |
| 后置条件 | | 调用Account领域对象的getAllPayment方法 |
| AccountController.searchReceipt | 语法 | | public GetVO searchReceipt\_up(String note); |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的searchReceipt方法 |
| AccountController.searchPayment | 语法 | | public PayVO searchPayment\_up(String note); |
| 前置条件 | | 已创建一个Account领域对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的searchPayment方法 |
| AccountController.getReceiptNote | 语法 | | public String getReceiptNote\_up(); |
| 前置条件 | | 已创建一个Account领域对象，需要添加收款单 |
| 后置条件 | | 调用Account领域对象的getReceiptNote方法 |
| AccountController.getPaymentNote | 语法 | | public String getPaymentNote\_up(); |
| 前置条件 | | 已创建一个Account领域对象，需要添加付款单 |
| 后置条件 | | 调用Account领域对象的getPaymentNote方法 |
| AccountController.getOperator | 语法 | | public String getOperator\_up(); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的getOperator方法 |
| AccountController.getAllCustomer | 语法 | | Public ArrayList<CustomerVO> getAllCustomer\_up(); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的getAllCustomer  方法 |
| AccountController.getAllAccount | 语法 | | public ArrayList<AccountVO> getAllAccount\_up(); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的getAllAccount方法 |
| AccountController.addDraftReceipt | 语法 | | public int addDraftReceipt\_up(GetVO vo); |
| 前置条件 | | 已创建一个Account对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象的addDraftReceipt方法 |
| AccountController.addDraftPayment | 语法 | | public int addDraftPayment\_up(PayVO vo); |
| 前置条件 | | 已创建一个Account对象，并且输入符合规则 |
| 后置条件 | | 调用Account领域对象addDraftPayment的方法 |
| AccountController.getAllDraftReceipt | 语法 | | Public ArrayList<GetVO> getAllDraftReceipt\_up(); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的getAllDraftReceipt方法 |
| AccountController.getAllDraftPayment | 语法 | | public ArrayList<PayVO> getAllDraftPayment\_up(); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的getAlDraftPayment方法 |
| AccountController.searchDraftReceipt | 语法 | | public GetVO searchDraftReceipt\_up(String note); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的searchDraftReceipt方法 |
| AccountController.searchDraftPayment | 语法 | | public PayVO searchDraftPayment\_up(String note); |
| 前置条件 | | 已创建一个Account对象 |
| 后置条件 | | 调用Account领域对象的searchDraftPayment方法 |
| AccountController.calTotalMoney | 语法 | | public double calTotalMoney\_up(GetVO vo);  public double calTotalMoney\_up(PayVO vo); |
| 前置条件 | | 已创建一个Account对象，创建一个收款单或付款单，已添加收付款项目 |
| 后置条件 | | 调用Account领域对象的calTotalMoney方法 |
| 需要的服务 | | | |
| Account.checkAccount() | | 获得当前账户信息 | |
| Account.addAccount (String name, int money) | | 添加账户 | |
| Account.delAccount(String name) | | 删除账户 | |
| Account.updateAccount(String name1, String name2) | | 更新账户 | |
| Account.searchFuzzyAccount(String word) | | 根据word模糊查找账户 | |
| Account.searchAccurateAccount(String name) | | 根据名称精确查找账户 | |
| Account.addReceipt(GetVO vo) | | 添加收款单 | |
| Account.addPayment(PayVO vo) | | 添加付款单 | |
| Account.getAllReceipt() | | 获得所有收款单 | |
| Account.getAllPayment() | | 获得所有付款单 | |
| Account.searchReceipt(String note) | | 根据单据编号查找并获得一个收款单 | |
| Account.searchPayment(String note) | | 根据单据编号查找并获得一个付款单 | |
| Account.getReceiptNote() | | 获得收款单单据编号 | |
| Account.getPaymentNote() | | 获得付款单单据编号 | |
| Account.getOperator() | | 获得操作员 | |
| Account.getAllCustomer() | | 获得当前所有客户信息 | |
| Account.getAllAccount() | | 获得当前所有账户信息 | |
| Account.addDraftReceipt()  Account.addDraftPayment() | | 添加一个草稿收款单 | |
| 添加一个草稿付款单 | |
| Account.getAllDraftReceipt() | | 获得所有草稿收款单 | |
| Account.getAllDraftPayment() | | 获得所有草稿付款单 | |
| Account.searchDraftReceipt(String note) | | 根据单据编号查找并获得一个草稿收款单 | |
| Account.searchDraftPayment(String note) | | 根据单据编号查找并获得一个草稿付款单 | |
| Account.calTotalMoney(GetVO vo) | | 计算收款单中的总额 | |
| Account.calTotalMoney(PayVO vo) | | 计算付款单中的总额 | |

**Account的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Account.checkAccount() | 语法 | Public AccountVO checkAccount\_up() |
| 前置条件 | 管理人员为总经理或财务经理 |
| 后置条件 | 返回账户信息 |
| Account.addAccount | 语法 | Public int addAccount\_up(AccountVO vo) |
| 前置条件 | 需要添加账户时，管理人员为总经理或财务经理 |
| 后置条件 | 返回是否添加成功，返回0为成功，否则为失败 |
| Account.delAccount | 语法 | Public int delAccount\_up(AccountVO vo) |
| 前置条件 | 需要删除账户时，管理人员为总经理或财务经理 |
| 后置条件 | 返回是否删除账户成功，返回0为成功，否则为失败 |
| Account.updateAccount | 语法 | Public int updateAccount\_up(AccountVO vo1,AccountVO vo2) |
| 前置条件 | 需要更新账户信息时，管理人员为总经理或财务经理 |
| 后置条件 | 完成账户信息修改，返回是否成功修改,返回1为成功，否则为失败。 |
| Account.searchAccount | 语法 | //模糊查找账户  public ArrayList<AccountVO> searchFuzzyAccount\_up(String word);    //精确查找账户  public AccountVO searchAccurateAccount\_up(String name); |
| 前置条件 | 操作员是总经理或财务经理 |
| 后置条件 | 返回搜索结果，模糊查找返回一个AccountVO类型的arrayList，精确查找返回查找的账户的VO，若未查找到相关信息，返回空的arrayList或AccountVO |
| Account.addReceipt | 语法 | public int addReceipt\_up(GetVO vo); |
| 前置条件 | 收到货款时，系统启动添加收款单，且输入的信息合法 |
| 后置条件 | 完成收款单，系统返回是否成功，成功返回0并交由总经理审批，否则失败。 |
| Account.addPayment | 语法 | public int addPayment\_up(PayVO vo); |
| 前置条件 | 完成付款后，系统启动添加付款单，且输入的信息合法 |
| 后置条件 | 完成付款单，系统返回是否成功，成功返回0并交由总经理审批，否则失败。 |
| Account.getAllReceipt | 语法 | Public ArrayList<GetVO> getAllReceipt\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有收款单 |
| Account.getAllPayment | 语法 | public ArrayList<PayVO> getAllPayment\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有的付款单 |
| Account.searchReceipt | 语法 | public GetVO searchReceipt\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 系统根据收款单单据编号查找并返回收款单具体信息 |
| Account.searchPayment | 语法 | public PayVO searchPayment\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 系统根据付款单单据编号查找并返回付款单具体信息 |
| Account.getReceiptNote | 语法 | public String getReceiptNote\_up(); |
| 前置条件 | 需要添加收款单时，系统启动添加收款单 |
| 后置条件 | 系统返回要添加的新的收款单的编号 |
| Account.getPaymentNote | 语法 | public String getPaymentNote\_up(); |
| 前置条件 | 需要添加付款单时，系统启动添加付款单 |
| 后置条件 | 系统返回要添加的新的付款单的编号 |
| Account.getOperator | 语法 | public String getOperator\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前登录人员姓名 |
| Account.getAllCustomer | 语法 | Public ArrayList<CustomerVO> getAllCustomer\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有客户的信息 |
| Account.getAllAccount | 语法 | public ArrayList<AccountVO> getAllAccount\_up(); |
| 前置条件 | 操作员为总经理或者财务经理 |
| 后置条件 | 系统返回当前所有账户的信息 |
| Account.addDraftReceipt | 语法 | public int addDraftReceipt\_up(GetVO vo); |
| 前置条件 | 收到货款时，系统启动添加收款单，且输入的信息合法，用户信息填写不完全选择保存 |
| 后置条件 | 完成草稿收款单，系统返回是否成功，成功返回0，否则失败。 |
| Account.addDraftPayment | 语法 | public int addDraftPayment\_up(PayVO vo); |
| 前置条件 | 完成付款后，系统启动添加付款单，且输入的信息合法，用户信息填写不完全选择保存 |
| 后置条件 | 完成草稿付款单，系统返回是否成功，成功返回0，否则失败。 |
| Account.getAllDraftReceipt | 语法 | Public ArrayList<GetVO> getAllDraftReceipt\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有草稿收款单 |
| Account.getAllDraftPayment | 语法 | public ArrayList<PayVO> getAllDraftPayment\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有的草稿付款单 |
| Account.searchDraftReceipt | 语法 | public GetVO searchDraftReceipt\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 系统根据收款单单据编号查找并返回草稿收款单具体信息 |
| Account.searchDraftPayment | 语法 | public PayVO searchDraftPayment\_up(String note); |
| 前置条件 | 无 |
| 后置条件 | 系统根据付款单单据编号查找并返回草稿付款单具体信息 |
| Account.calTotalMoney | 语法 | public double calTotalMoney\_up(GetVO vo);  public double calTotalMoney\_up(PayVO vo); |
| 前置条件 | 添加收款单或付款单 |
| 后置条件 | 系统返回总额 |
| 需要的服务 | | |
| AccountDataService.getAccount() | | 获得当前账户信息 |
| AccountDataService.addAccount(AccountPO po) | | 添加账户AccountPO对象 |
| AccountDataService.delAccount(AccountPO po) | | 删除账户AccountPO对象 |
| AccountDataService.updateAccount(AccountPO po1, AccountPO po2) | | 更新账户AccountPO对象 |
| AccountDataService.addMoney(AccountPO po,double money) | | 增加账户AccountPO对象po中的余额 |
| AccountDataService.delMoney(AccountPO po,double money) | | 减少账户AccountPO对象po中的余额 |
| AccountDataService.findAccount(String name) | | 根据name查找账户对象AccountPO po |
| AccountDataService.getAllAccount() | | 获得所有的账户对象AccountPO po |
| AccountDataService.addReceipt(ReceiptPO po) | | 添加收款单ReceiptPO对象ReceiptPO po |
| AccountDataService.addPayment(PaymentPO po) | | 添加付款单PaymentPO对象PaymentPO po |
| AccountDataService.findReceipt(String note) | | 根据收款单编号查找获得单一ReceiptPO po对象 |
| AccountDataService.findPayment(String note) | | 根据付款单编号查找获得单一PaymentPO po对象 |
| AccountDataService.getAllReceipt() | | 获得所有的收款单对象ReceiptPO po |
| AccountDataService.getAllPayment() | | 获得所有的付款单对象PaymentPO po |
| AccountDataService.getReceiptNote() | | 获得要新增的收款单的单据编号 |
| AccountDataService.getPaymentNote() | | 获得要新增的付款单的单据编号 |
| AccountDataService. passReceipt(ReceiptPO po) | | 新增审批通过的收款单对象ReceiptPO po |
| AccountDataService.passPayment(PaymentPO po) | | 新增审批通过的付款单对象PaymentPO po |
| AccountDataService.refuseReceipt(ReceiptPO po) | | 新增审批拒绝的首款单对象ReceiptPO po |
| AccountDAtaService.refusePayment(PaymentPO po) | | 新增审批拒绝的付款单对象PaymentPO po |
| AccountDataService.addDraftReceipt(ReceiptPO po) | | 添加草稿收款单ReceiptPO对象ReceiptPO po |
| AccountDataService.addDraftPayment(PaymentPO po) | | 添加草稿付款单PaymentPO对象PaymentPO po |
| AccountDataService.delDraftReceipt(ReceiptPO po) | | 删除草稿收款单ReceiptPO对象ReceiptPO po |
| AccountDataService.delDraftPayment(PaymentPO po) | | 删除草稿付款单PaymentPO对象PaymentPO po |
| AccountDataService.findDraftReceipt(String note) | | 根据草稿收款单编号查找获得单一ReceiptPO po对象 |
| AccountDataService.findDraftPayment(String note) | | 根据草稿付款单编号查找获得单一PaymentPO po对象 |
| AccountDataService.getAllDraftReceipt() | | 获得所有的草稿收款单对象ReceiptPO po |
| AccountDataService.getAllDraftPayment() | | 获得所有的草稿付款单对象PaymentPO po |
| AccountDataService. passReceipt(ReceiptPO po) | | 新增审批通过的收款单对象ReceiptPO po |
| AccountDataService.passPayment(PaymentPO po) | | 新增审批通过的付款单对象PaymentPO po |
| AccountDataService.refuseReceipt(String note) | | 新增审批拒绝的首款单对象ReceiptPO po |
| AccountDAtaService.refusePayment(String note) | | 新增审批拒绝的付款单对象PaymentPO po |
| AccountDataService.clear | | 清除所有对象 |

(4)业务逻辑层的动态模型



(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.3 financialbl模块

(1)模块概述

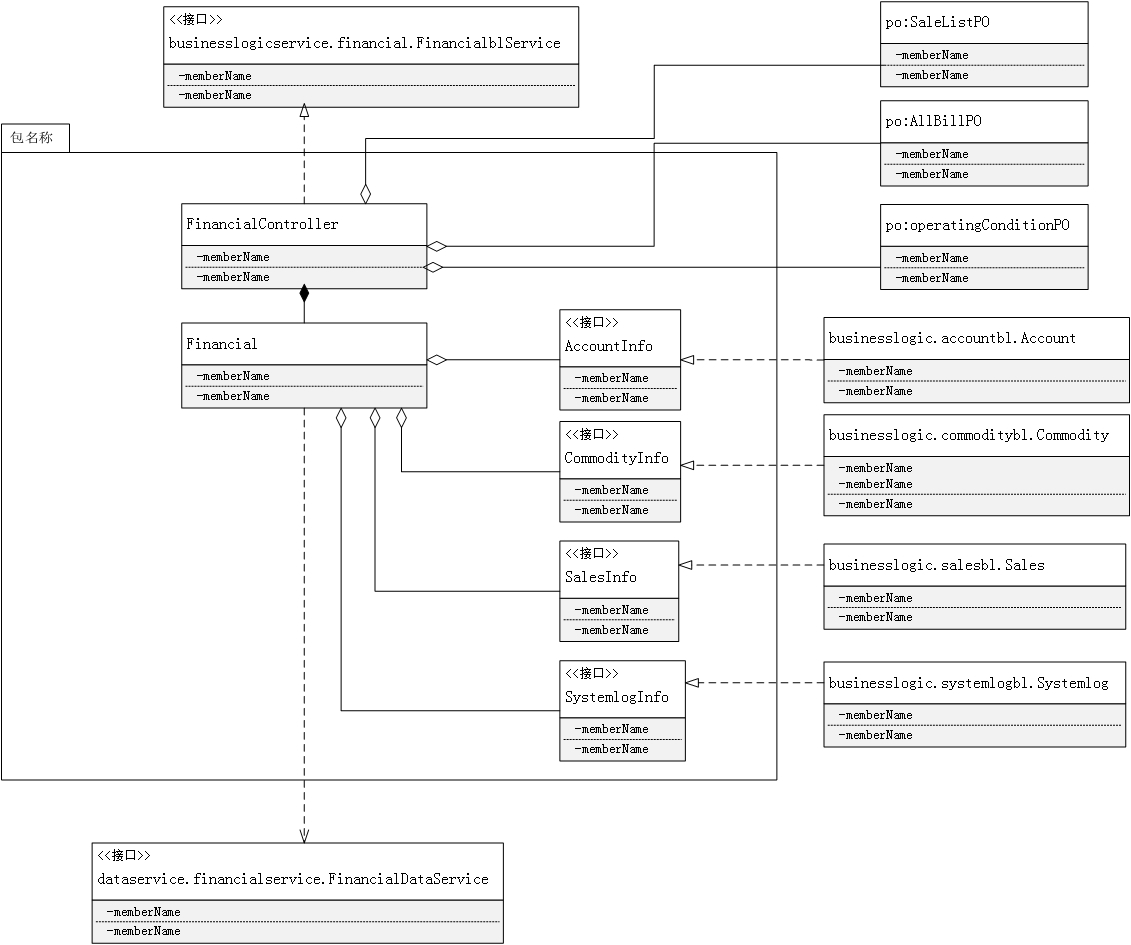
Financialbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Financialbl模块的职责及借款参见软件体系结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。Financial模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.financialblservice.FinancialblService接口与业务逻辑层与数据层之间的dataservice.financialdataservice.FinancialblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了FinancialController，这样FinancialController会将对账户的业务逻辑处理委托给Financial对象。SalesListPO是作为销售明细的持久化对象被添加到设计模型中去的。AllBillPO是作为经营历程的持久化对象被添加到设计模型中去的。OperatingConditionPO是作为经营情况的持久化对象被添加到设计模型中去的。SystemlogInfo、AccountInfo、SalesInfo、CommodityInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Financialbl模块的设计如图



Financialbl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| FinancialController | 负责实现财务表单界面所需要的服务 |
| Financial | 财务账单领域模型对象，拥有经过加工后的账单信息，可以帮助完成财务表单界面所需要的服务 |

(3)模块内部类的接口规范

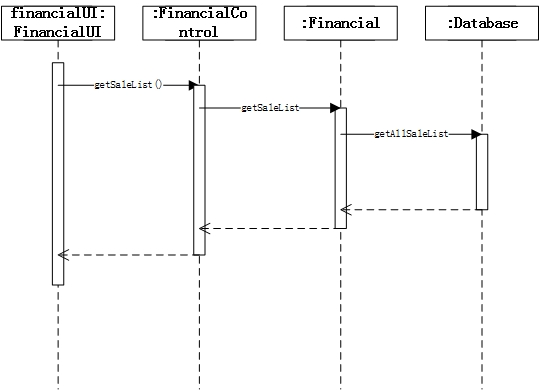
**FinancialController接口规范**

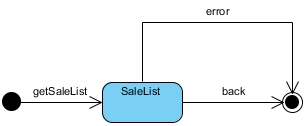
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| FinancialController.saleList | 语法 | public ArrayList<SalesDetailVO> saleList\_up(String time1, String time2, String good\_name,String good\_type, String customer\_name, String clerk, String warehouse); |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的saleList方法 |
| FinancialController.allBill | 语法 | public ArrayList<AllBillVO> allBill\_up  (String time1, String time2, String note\_type,String customer\_name, String clerk, String warehouse); |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的allBill方法 |
| FinancialController.operatingCondition | 语法 | public ArrayList<ConditionVO> operatingCondition\_up(String start\_time, String end\_time); |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的operatingCondition方法 |
| FinancialController.allBillExcel | 语法 | public int allBillExcel\_up(ArrayList<AllBillVO> allBill\_condition); |
| 前置条件 | 已创建一个Financial领域对象 |
| 后置条件 | 调用Financial领域对象的allBillExcel方法 |
| FinancialController.saleListExcel | 语法 | public int saleListExcel\_up(ArrayList<SalesDetailVO> saleList\_condition); |
| 前置条件 | 已创建一个Financial领域对象 |
| 后置条件 | 调用Financial领域对象的saleListExcel方法 |
| FinancialController.operatingConditionExcel | 语法 | public int operatingConditionExcel\_up(ArrayList<ConditionVO> oprating\_condition); |
| 前置条件 | 已创建一个Financial领域对象 |
| 后置条件 | 调用Financial领域对象的operating  ConditionExcel方法 |
| FinancialController.getOperator | 语法 | public String getOperator\_up(); |
| 前置条件 | 已创建一个Financial领域对象 |
| 后置条件 | 调用Financial领域对象的getOperator方法 |
| 需要的服务 | | |
| Financial.saleList(String time1, String time2, String good\_name, String good\_type, String customer\_name, String clerk, int warehouse) | 根据条件查询销售明细 | |
| Financial.allBill(String time1, String time2, String note\_type, String customer\_name, String clerk, int warehouse) | 根据条件查询经营历程 | |
| Financial.operatingCondition(String time1, String time2) | 根据条件查询经营情况 | |
| Finacial.saleListExcel\_up(ArrayList<SalesDetailVO> saleList\_condition) | 导出销售明细表 | |
| Financial.allBillExcel\_up(ArrayList<AllBillVO> allBill\_condition); | 导出经营历程表 | |
| Financial.operatingConditionExcel\_up(ArrayList<ConditionVO> oprating\_condition) | 到处经营情况表 | |
| Financial.GetOperator\_up() | 获得当前操作员 | |

**Financial的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Financial.saleList | 语法 | public ArrayList<SalesDetailVO> saleList\_up(String time1, String time2, String good\_name,String good\_type, String customer\_name, String clerk, String warehouse); |
| 前置条件 | 操作员是经过验证的财务人员或总经理，输入信息合法 |
| 后置条件 | 查询销售明细，返回符合条件的销售明细表 |
| Financial.allBill | 语法 | public ArrayList<AllBillVO> allBill\_up  (String time1, String time2, String note\_type,String customer\_name, String clerk, String warehouse); |
| 前置条件 | 操作员是经过验证的财务人员，输入信息合法 |
| 后置条件 | 查询经营历程表，返回符合条件的经营历程表。 |
| Financial.operatingCondition | 语法 | public ArrayList<ConditionVO> operatingCondition\_up(String start\_time, String end\_time); |
| 前置条件 | 操作员是经过验证的财务人员，输入信息合法 |
| 后置条件 | 查看经营情况表，返回符合条件的经营情况表 |
| Financial.allBillExcel | 语法 | public int allBillExcel\_up(ArrayList<AllBillVO> allBill\_condition); |
| 前置条件 | 无 |
| 后置条件 | 导出经营历程表，地址默认为D盘根目录下 |
| Financial.saleListExcel | 语法 | public int saleListExcel\_up(ArrayList<SalesDetailVO> saleList\_condition); |
| 前置条件 | 无 |
| 后置条件 | 导出销售明细表，地址默认为D盘根目录下 |
| Financial.operatingConditionExcel | 语法 | public int operatingConditionExcel\_up(ArrayList<ConditionVO> oprating\_condition); |
| 前置条件 | 无 |
| 后置条件 | 导出经营情况表，默认地址为D盘根目录下 |
| Financial.getOperator | 语法 | public String getOperator\_up(); |
| 前置条件 | 无 |
| 后置条件 | 返回当前，登陆人员的姓名 |
| 需要的服务 | | |
| FinancialDataService.getSaleList() | 获得所有销售明细对象SalesListPO | |
| FinancialDataService.getAllBill() | 获得所有经营历程对象AllBillPO | |
| FinancialDataService.getOperatingCondition() | 获得所有经营情况对象OperatingConditionPO | |
| FinancialDataService.addSaleList(SaleListPO po) | 添加销售明细的单据对象SaleListPO po | |
| FinancialDataService.addAllBill(AllBillPO po) | 添加经营历程的单据对象AllBillPO po | |
| FinancialDataService.addOperatingCondition(OperatingConditionPO po) | 添加经营情况的单据对象OperatingConditionPO po | |
| FinancialDataService.clear() | 清空所有对象 | |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.4 initializationbl 模块

(1)模块描述

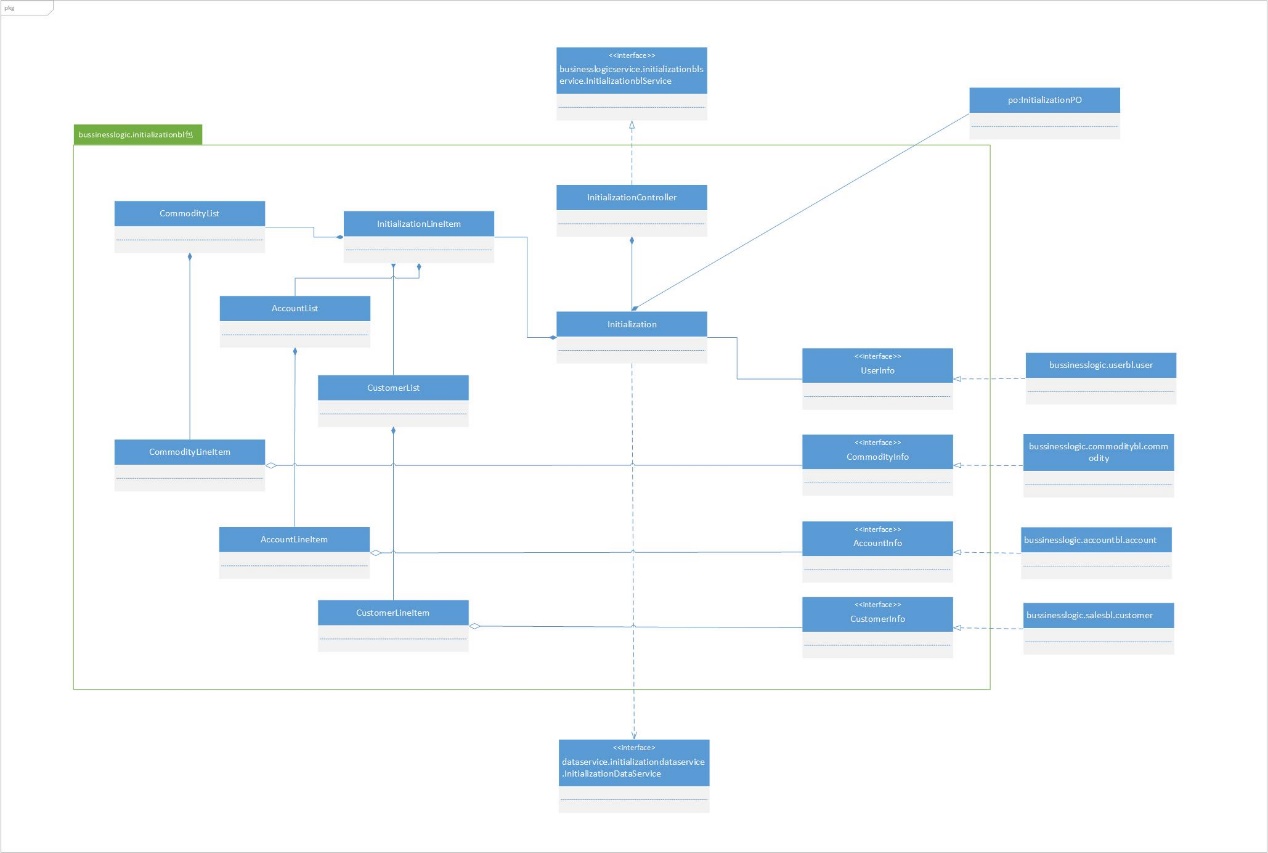
Initializationbl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Initializationbl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.initializationbl.InitializationblService接口。业务逻辑层和数据层之间添加dataservice.intializationdata.InitializtionDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InitializationController，这样InitializationController会将对期初建账的业务逻辑处理委托给Initialization对象。InitializationPO时作为期初建账记录的持久化对象被添加到设计模型中去的。而CommodityLineItem,AccountLineItem,CustomerLineItem和CommodityList，AccountList，CustomerList的添加分别是CommodityInfo，AccountInfo，CustomerInfo的容器类。CommodityLineItem，CustomerLineItem和AccountLineItem分别包有商品信息（商品分类，某一商品的名称，类别，型号，进价/售价（默认为上年平均），最近进价和最近售价（留空）），客户信息（客户分类，某一客户的名称，联系方式，应收应付），银行账户信息（账户名称，余额）。而CommodityList，CustomerList和AccountList分别封装了关于添加的CommodityLineItem，CustomerLineItem和AccountLineItem的数据集合的数据结构的秘密。InitializationLineItem封装了CommodityList，CustomerList和AccountList的数据集合的数据结构的秘密。UserInfo,CommodityInfo,AccountInfo和CustomerInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Initializationbl模块的设计如下图所示



Initializationbl模块各个类的设计

Initializationbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| InitializationController | 负责实现期初建账界面所需要的服务。 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名，ID，职位和密码，可以解决登录问题。 |
| Initialization | 期初建账的领域模型对象，拥有一次期初建账所持有的新增商品信息，客户信息和账户信息，可以帮助完成期初建账界面所需要的服务。 |

(3)模块内部类的接口规范

InitializationController 和 Initialization的接口规范如下表

InitializationController的接口规范

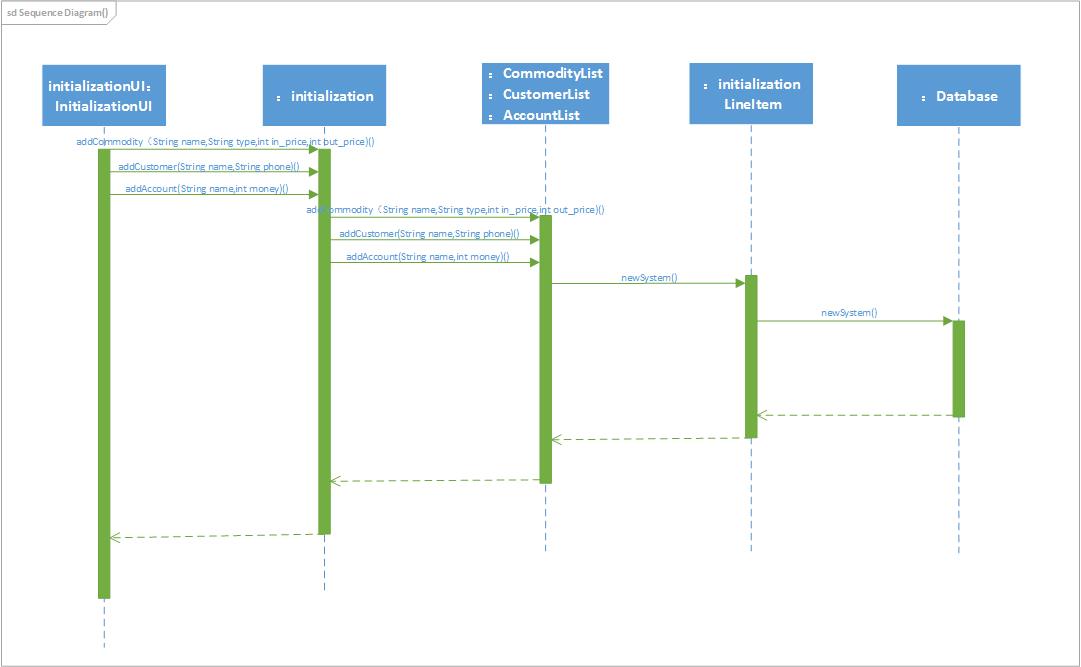
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| InitializationController.newSystem | 语法 | | Public ResultMessage newSystem() |
| 前置条件 | | 已创建一个Initialization领域对象，并且操作人员输入完成期初建账的信息，包括商品，客户，账户。 |
| 后置条件 | | 调用Initialization领域对象的newSystem方法 |
| InitializationController.addCommodity | 语法 | | Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addCommodity方法 |
| InitializationController.addCustomer | 语法 | | Public ResultMessage addCustomer(String name, String phone) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addCustomer方法 |
| InitializationController.addAccount | 语法 | | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addAccount方法 |
| InitializationController.showInformation | 语法 | | Public ResultMessage showInformation() |
| 前置条件 | | 已创建一个Initialization领域对象。 |
| 后置条件 | | 调用Initialization领域对象的showInformation方法 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| Initialization.newSystem | 创建一个新的系统，并替换旧系统。 | | |
| Initialization.addCommodity | 记录此商品信息，等待替换库存内商品 | | |
| Initialization.addCustomer | 记录此客户信息，等待替换客户信息 | | |
| Initialization.addAccount | 记录此账户信息，等待替换账户信息 | | |
| Initialization.showInformation | 返回显示期初信息 | | |

Initialization的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Initialization.newSystem | 语法 | Public ResultMessage newSystem() |
| 前置条件 | 操作人员输入完成期初建账的信息，包括商品，客户，账户 |
| 后置条件 | 旧系统内容将被删除，新系统替换旧系统 |
| Initialization.addCommodity | 语法 | Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此商品信息，等待替换库存内商品 |
| Initialization.addCustomer | 语法 | Public ResultMessage addCustomer(String name, String phone) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此客户信息，等待替换客户信息 |
| Initialization.addAccount | 语法 | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此账户信息，等待替换账户信息 |
| Initialization.showInformation | 语法 | Public ResultMessage showInformation() |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 返回期初信息 |
| 需要的服务 | | |
| InitializationDataService.getInformation | 查看InitializationPO对象的信息 | |

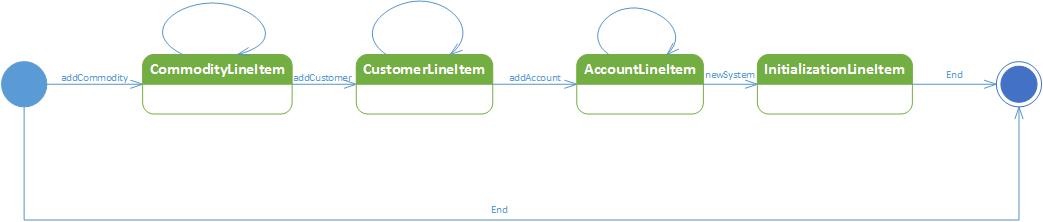
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户输入要添加的商品信息、客户信息和账户信息之后，期初建账逻辑处理的相关对象之间协作。



期初建账的顺序图

下图所示的状态图描述了Initialization对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着addCommodity方法被UI调用，Initialization进入CommodityLineItem状态，接着addCustomer方法被 UI调用，Initialization进入CustomerLineItem状态，接着addAccount方法被UI调用，Initialization进入AccountLineItem状态。之后通过添加新帐进入InitializationLineItem状态。



Initialization对象状态图

(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.1.5 invoice 模块

(1)模块描述

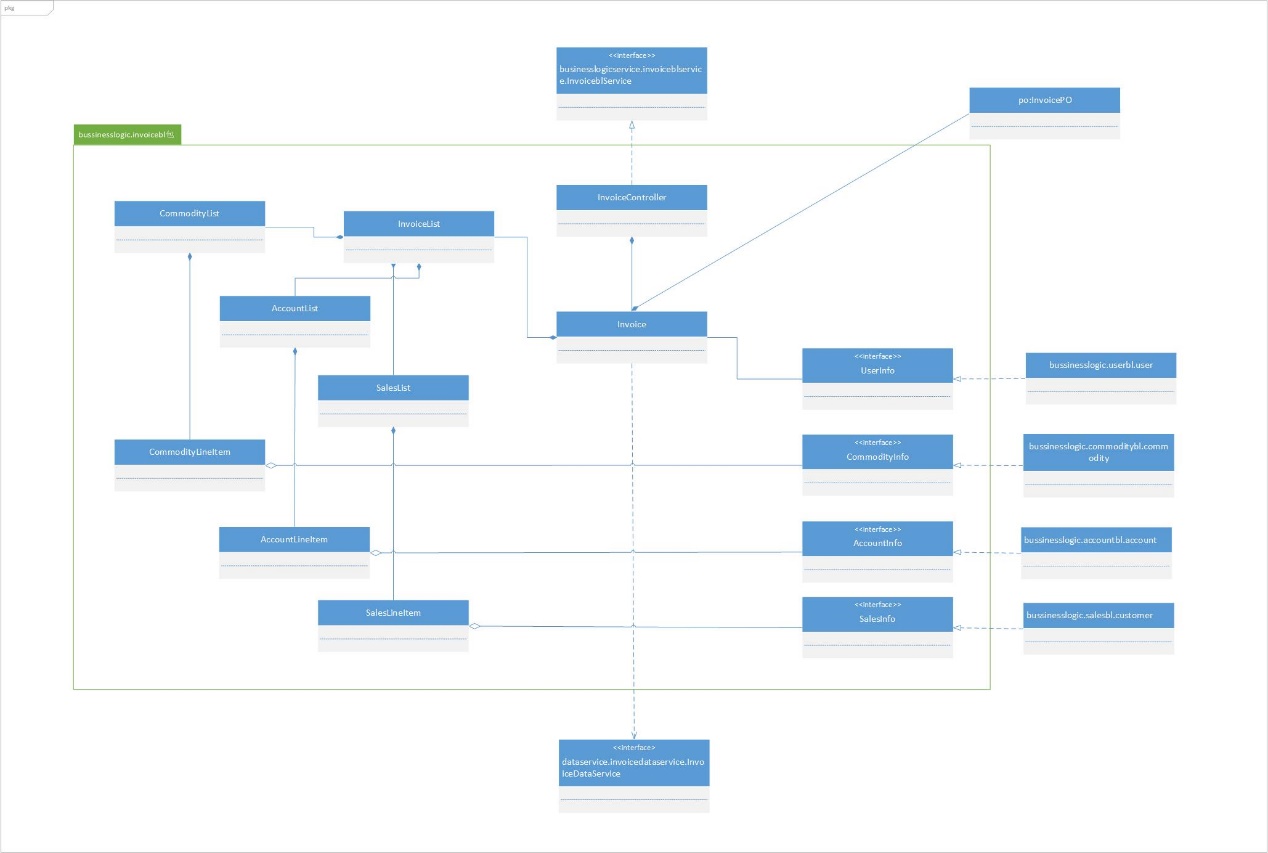
Invoicebl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Invoicebl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.invoicebl.InvoiceblService接口。业务逻辑层和数据层之间添加dataservice.invoicedata.InvoiceDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InvoiceController，这样InvoiceController会将审批单据的业务逻辑处理委托给Invoice对象。InvoicePO是作为审批单据记录的持久化对象被添加到设计模型中去的。而CommodvityLineItem,AccountLineItem,SalesLineItem和CommodityList，AccountList，SalesList的添加分别是CommodityInfo，AccountInfo，SalesInfo的容器类。CommodityLineItem，SalesLineItem和AccountLineItem分别包有库存进货单信息，进货单、销售单、进货退货单、销售退货单信息，收款单、付款单信息。而CommodityList，SalesList和AccountList分别封装了关于添加的CommodityLineItem，SalesLineItem和AccountLineItem的数据集合的数据结构的秘密。InvoiceList封装了CommodityList，SalesList和AccountList的数据集合的数据结构的秘密。UserInfo,CommodityInfo,AccountInfo和SalesInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Invoicebl模块的设计如下图所示：



Invoicebl模块各个类的设计

Invoicebl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| InvoiceController | 负责实现审批单据界面所需要的服务。 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名，ID，职位和密码，可以解决登录问题。 |
| Invoice | 审批单据的领域模型对象，拥有一次审批单据所持有的新增的单据信息，可以帮助完成审批单据界面所需要的服务。 |

(3)模块内部类的接口规范

**InvoiceController的接口规范**

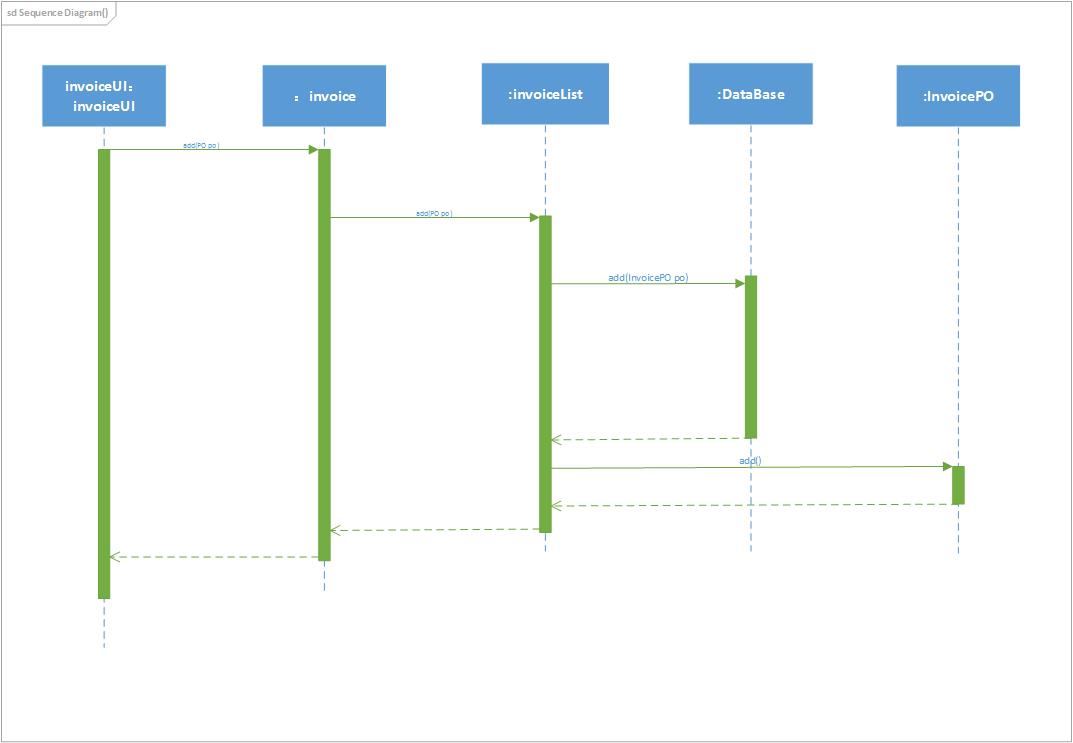
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| InvoiceController.show | | 语法 | public ArrayList<InvoiceVO> show\_up(); |
| 前置条件 | 已创建一个Invoice领域对象 |
| 后置条件 | 调用Invoice领域对象的show方法 |
| InvoiceController.pass | | 语法 | public int pass\_up(InvoiceVO vo); |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的pass方法 |
| InvoiceController.refuse | | 语法 | public int refuse\_up(String note); |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的refuse方法 |
| InvoiceController.search | | 语法 | public InvoiceVO searchNote\_up(String note); |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的search方法 |
| Invoice.show\_pass | | 语法 | public ArrayList<InvoiceVO> show\_pass(); |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的show\_pass方法 |
| Invoice.show\_refuse | | 语法 | Public ArrayList<InvoiceVO>  show\_refuse(); |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的show\_refuse方法 |
| 需要的服务（需接口） | | | |
| 服务名 | 服务 | | |
| Invoice.show() | 显示待审核单据 | | |
| Invoice.pass(InvoiceVO vo) | 通过单据审批 | | |
| Invoice.refuse(InvoiceVO vo) | 拒绝单据审批 | | |
| Invoice.search(String note) | 根据单据编号查找单据 | | |
| Invoice.show\_pass() | 显示审批通过单据 | | |
| Invoice.show\_refuse() | 显示审批拒绝单据 | | |

**Invoice的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| Invoice.show | 语法 | | public ArrayList<InvoiceVO> show\_up(); |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 返回所有待审批单据信息 |
| Invoice.show\_pass | 语法 | | public ArrayList<InvoiceVO> show\_pass(); |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 返回所有审批通过的单据信息 |
| Invoice.show\_refuse | 语法 | | Public ArrayList<InvoiceVO>  show\_refuse(); |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 返回所有审批拒绝单据的信息 |
| Invoice.pass | 语法 | | public int pass\_up(InvoiceVO vo); |
| 前置条件 | | 操作人员为总经理，库存管理人员，销售人员、财务人员填写好单据 |
| 后置条件 | | 总经理审批通过单据 |
| Invoice.refuse | 语法 | | public int refuse\_up(String note); |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 系统通过单据编号查找到具体单据信息，总经理审批拒绝单据 |
| Invoice.search | 语法 | | public InvoiceVO searchNote\_up(String note); |
| 前置条件 | | 无 |
| 后置条件 | | 系统通过单据编号查找并返回单据具体信息 |
| 需要的服务 | | | |
| InvoiceDataServoice.getInvoice(String note) | | 根据单据编号查找并获得一个单据InvoicePO对象 | |
| InvoiceDataService.addInvoice(InvoicePO po) | | 添加一个待审批单据InvoicePO对象 | |
| InvoiceDataService.passInvoice(InvoicePO po) | | 审批通过一个单据InvoicePO对象 | |
| InvoiceDataService.refuseInvoice(InvoicePO po) | | 审批拒绝一个单据InvoicePO对象 | |
| InvoiceDataService.getAllInvoice() | | 获得所有单据InvoicePO对象 | |
| InvoiceDataService.getInvNote() | | 获得一个审批单据的编号 | |
| InvoiceDataService.clear() | | 清空所有单据 | |

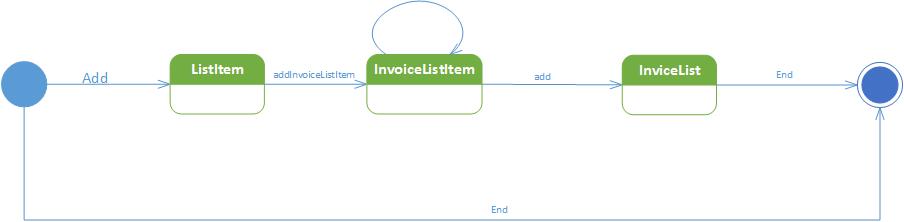
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户创建新的进货单、销售单、进货退货单、销售退货单、收款单、付款单或库存赠送单之后，总经理审批单据逻辑处理的相关对象之间协作。



审批单据的顺序图

下图所示的状态图描述了Invoice对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着add方法被UI调用，Invoice进入ListItem状态，接着addInvoiceListItem方法调用，Invoice进入InvoiceLineItem状态。之后通过向InvoiceList中添加新的待审批的单据要进入InvoiceList状态。



Invoice对象状态图

(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.6 promotion 模块

(1)模块描述

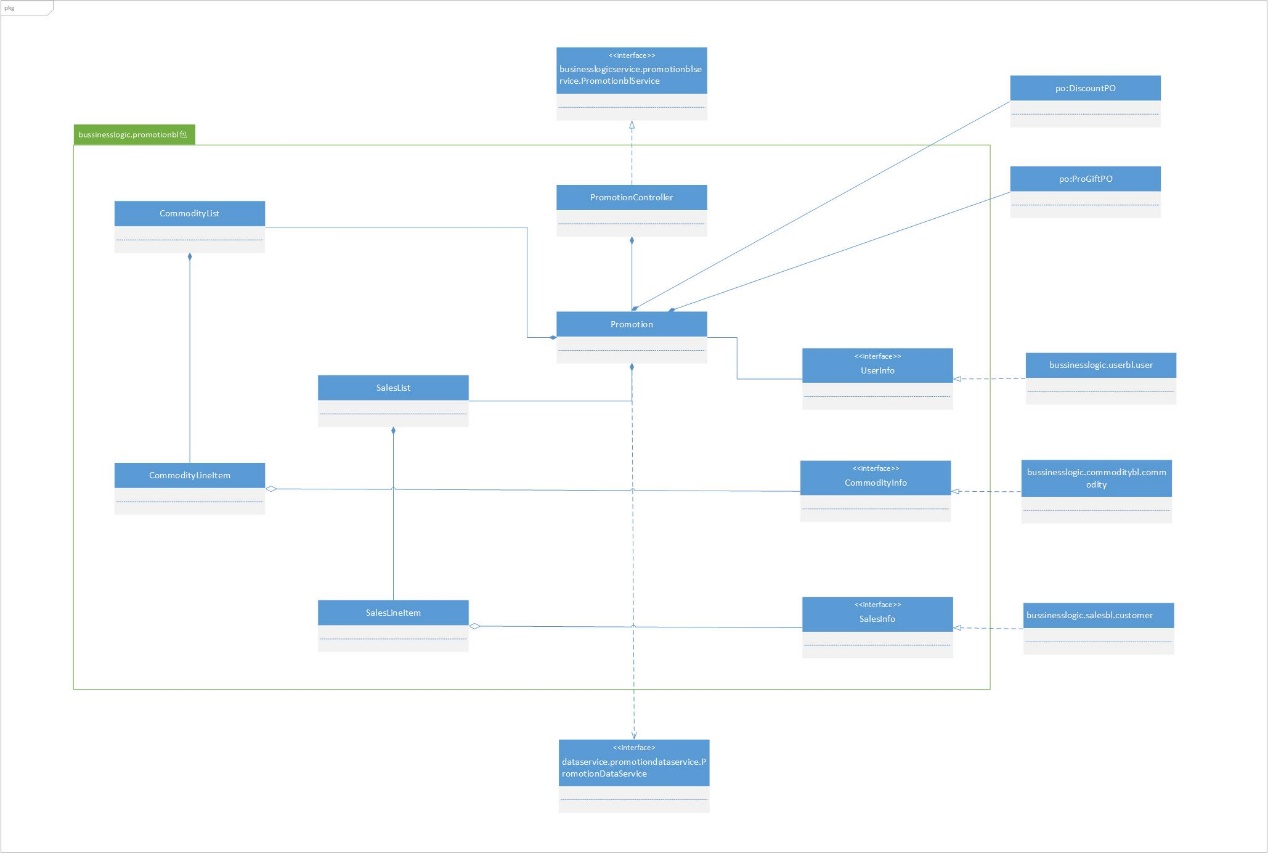
Promotionbl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Promotionbl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.promotionbl.PromotionblService接口。业务逻辑层和数据层之间添加dataservice.promotiondata.InvoiceDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PromotionController，这样PromotionController会将制定促销策略的业务逻辑处理委托给Promotion对象。DiscountPO，ProGiftPO是作为促销策略记录的持久化对象被添加到设计模型中去的。而CommodvityLineItem,SalesLineItem和CommodityList，SalesList的添加分别是CommodityInfo，SalesInfo的容器类。CommodityLineItem，SalesLineItem分别包有要赠送的商品信息和销售中的折扣信息，并根据该信息对销售做相应改变。而CommodityList，SalesList分别封装了关于添加的CommodityLineItem和SalesLineItem的数据集合的数据结构的秘密。PromotionList封装了CommodityList和SalesList的数据集合的数据结构的秘密。UserInfo,CommodityInfo和SalesInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Promotionbl模块的设计如下图所示：



Promotionbl模块各个类的设计

Promotionbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| PromotionController | 负责实现制定销售策略界面所需要的服务。 |
| Promotion | 制定销售策略的领域模型对象，拥有一次制定销售策略所持有的新增的销售策略信息，可以帮助完成制定销售策略界面所需要的服务。 |

(3)模块内部类的接口规范

PromotionController的接口规范

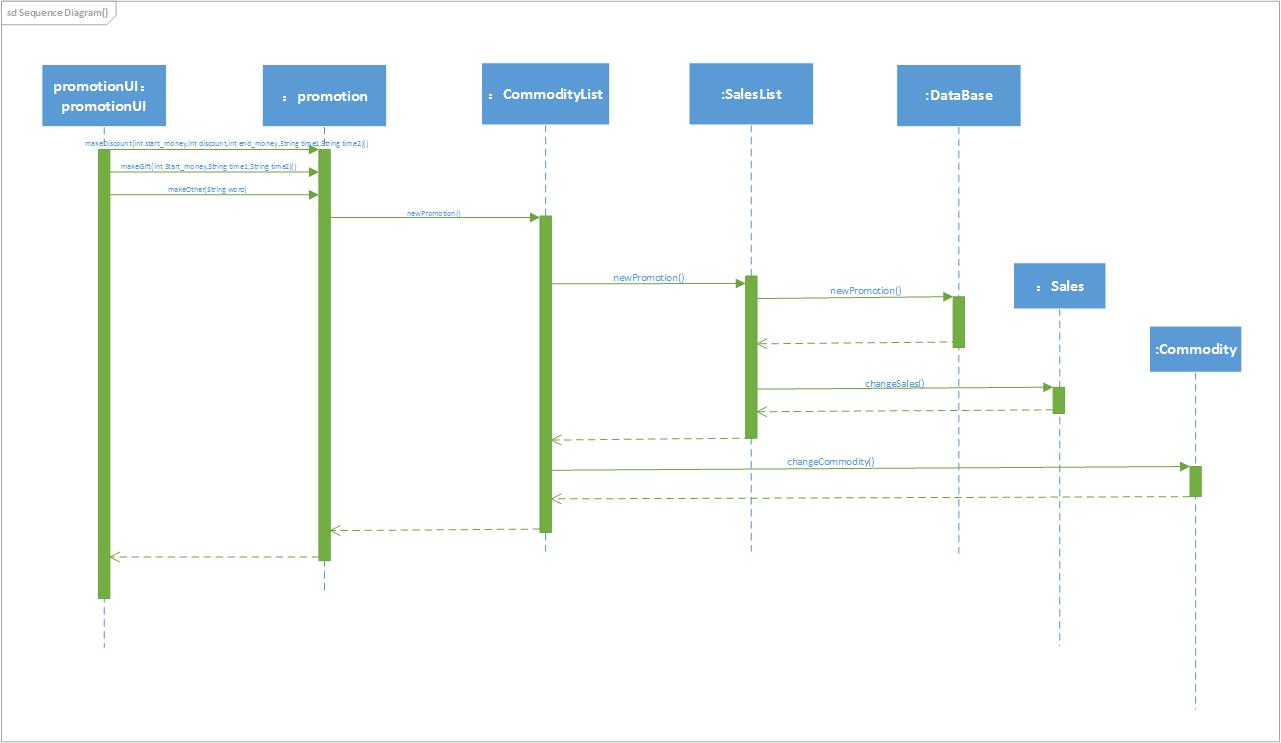
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| PromotionController.makeDiscount | 语法 | | public int makeDiscount\_up(DiscountVO vo); |
| 前置条件 | | 操作用户为总经理,系统启动新增折扣促销策略 |
| 后置条件 | | 制作好折价优惠之后，立即更新至销售 |
| PromotionController.makeVoucher | 语法 | | public int makeVoucher\_up(VoucherVO vo); |
| 前置条件 | | 操作用户为总经理,系统启动新增代金券促销策略 |
| 后置条件 | | 制作好代金券优惠之后，立即更新至销售 |
| PromotionController.makeGift | 语法 | | public int makeGift\_up(ProGiftVO vo); |
| 前置条件 | | 操作用户为总经理，系统启动新增赠品促销策略 |
| 后置条件 | | 制作好赠品优惠之后，立即更新至销售 |
| PromotionController.delDiscount | 语法 | | public int delDiscount\_up(int level); |
| 前置条件 | | 操作用户为总经理,系统启动删除折扣促销策略 |
| 后置条件 | | 系统删除折扣优惠策略，立即更新至销售 |
| PromotionController.delVoucher | 语法 | | public int delVoucher\_up(int level); |
| 前置条件 | | 操作用户为总经理,系统启动删除代金券促销策略 |
| 后置条件 | | 系统删除代金券优惠，立即更新至销售 |
| PromotionController.delGift | 语法 | | public int delGift\_up(int level); |
| 前置条件 | | 操作用户为总经理，系统启动删除赠品促销策略 |
| 后置条件 | | 系统删除赠品优惠，立即更新至销售 |
| PromotionController.getDiscount | 语法 | | Public ArrayList<DiscountVO> getDiscount\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有折扣促销策略信息 |
| PromotionController.getProGift | 语法 | | public ArrayList<ProGiftVO> getProGift\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有赠品促销信息 |
| PromotionController.getVoucher | 语法 | | public ArrayList<VoucherVO> getVoucher\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有代金券促销信息 |
| 需要的服务 | | | |
| Promotion.addDiscount(DiscountPO po) | | 添加折价促销DiscountPO方案对象 | |
| Promotion.addGift(ProGiftPO po) | | 添加赠品促销ProGiftPO方案对象 | |
| Promotion.addVoucher(VoucherPO po) | | 添加代金券促销VoucherPO方案对象 | |
| Promotion.delDiscount(int level) | | 根据用户等级查找到折扣促销策略DiscountPO方案对象后删除 | |
| Promotion.delGift(int level) | | 根据用户等级查找到赠品促销策略  ProGiftPO方案对象后删除 | |
| Promation.delVoucher(int level) | | 根据用户等级查找到代金券促销策略  VoucherPO方案对象后删除 | |
| Promotion.getDiscount(int level) | | 根据用户等级获得折扣促销策略DiscountPO | |
| Promotion.getGift(int level) | | 根据用户等级获得赠品促销策略ProGiftPO | |
| Promotion.getVoucher(int level) | | 根据用户等级获得代金券促销策略VoucherPO | |

Promotion的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| Promotion.makeDiscount | 语法 | | public int makeDiscount\_up(DiscountVO vo); |
| 前置条件 | | 操作用户为总经理,系统启动新增折扣促销策略 |
| 后置条件 | | 制作好折价优惠之后，立即更新至销售 |
| Promotion.makeVoucher | 语法 | | public int makeVoucher\_up(VoucherVO vo); |
| 前置条件 | | 操作用户为总经理,系统启动新增代金券促销策略 |
| 后置条件 | | 制作好代金券优惠之后，立即更新至销售 |
| Promotion.makeGift | 语法 | | public int makeGift\_up(ProGiftVO vo); |
| 前置条件 | | 操作用户为总经理，系统启动新增赠品促销策略 |
| 后置条件 | | 制作好赠品优惠之后，立即更新至销售 |
| Promotion.delDiscount | 语法 | | public int delDiscount\_up(int level); |
| 前置条件 | | 操作用户为总经理,系统启动删除折扣促销策略 |
| 后置条件 | | 系统删除折扣优惠策略，立即更新至销售 |
| Promotion.delVoucher | 语法 | | public int delVoucher\_up(int level); |
| 前置条件 | | 操作用户为总经理,系统启动删除代金券促销策略 |
| 后置条件 | | 系统删除代金券优惠，立即更新至销售 |
| Promotion.delGift | 语法 | | public int delGift\_up(int level); |
| 前置条件 | | 操作用户为总经理，系统启动删除赠品促销策略 |
| 后置条件 | | 系统删除赠品优惠，立即更新至销售 |
| Promotion.getDiscount | 语法 | | Public ArrayList<DiscountVO> getDiscount\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有折扣促销策略信息 |
| Promotion.getProGift | 语法 | | public ArrayList<ProGiftVO> getProGift\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有赠品促销信息 |
| Promotion.getVoucher | 语法 | | public ArrayList<VoucherVO> getVoucher\_up(); |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回所有代金券促销信息 |
| 需要的服务 | | | |
| PromotionDataService.addDiscount(DiscountPO po) | | 添加折价促销DiscountPO方案对象 | |
| PromotionDataService.addGift(ProGiftPO po) | | 添加赠品促销ProGiftPO方案对象 | |
| PromotionDataService.addVoucher(VoucherPO po) | | 添加代金券促销VoucherPO方案对象 | |
| PromotionDataService.delDiscount(int level) | | 根据用户等级查找到折扣促销策略DiscountPO方案对象后删除 | |
| PromotionDataService.delGift(int level) | | 根据用户等级查找到赠品促销策略  ProGiftPO方案对象后删除 | |
| PromationDataService.delVoucher(int level) | | 根据用户等级查找到代金券促销策略  VoucherPO方案对象后删除 | |
| PromotionDataService.getDiscount(int level) | | 根据用户等级获得折扣促销策略DiscountPO | |
| PromotionDataService.getGift(int level) | | 根据用户等级获得赠品促销策略ProGiftPO | |
| PromotionDataService.getVoucher(int level) | | 根据用户等级获得代金券促销策略VoucherPO | |
| PromotionDataservice.showDiscount() | | 获得当前折价促销方案对象 | |
| PromotionDataService.showProGift() | | 获得当前赠品促销方案对象 | |
| PromotionDataService.showVoucher() | | 获得当前代金券促销方案对象 | |
| PromotionDataService.clear() | | 清空所有对象 | |

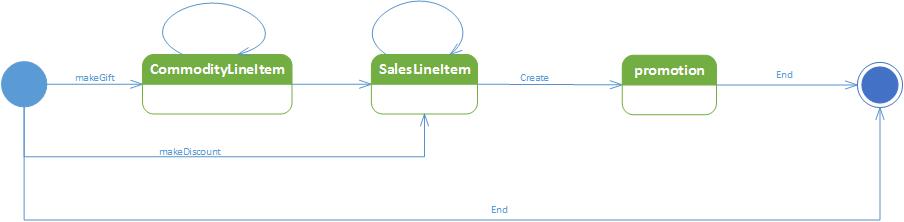
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户创建新的进货单、销售单、进货退货单、销售退货单、收款单、付款单或库存赠送单之后，总经理审批单据逻辑处理的相关对象之间的协作。



创建新的销售策略的顺序图

下图所示的状态图描述了Promotion对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着makeGift方法被UI调用，Promotion进入CommodityLineItem状态，接着进入SalesLineItem状态。UI可以通过直接调用makeDiscount方法直接进入SalesLineItem状态。之后通过创建新的销售策略进入promotion状态。



Promotion对象状态图

(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.7 salesbl模块

(1)模块概述

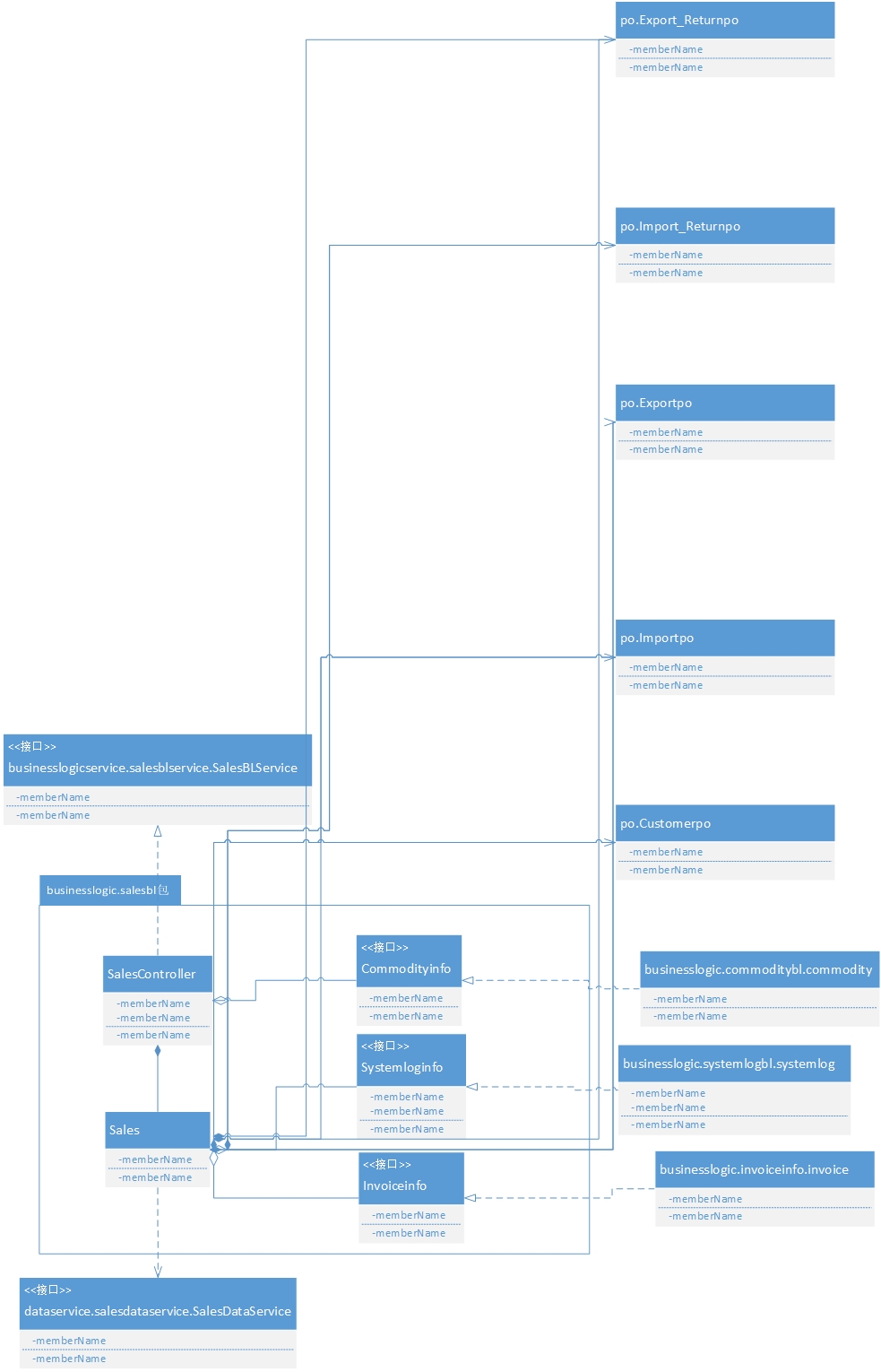
salesbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

salesbl模块的职责及接口参见软件系统结构描述文档表9中的salesbl的接口规范。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。sales模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.salesblservice.SalesblService接口与业务逻辑层与数据层之间的dataservice.salesdataservice.SalesblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SalesController，这样SalesController会将对销售的业务逻辑处理委托给Sales对象。ImportPO是作为进货的持久化对象被添加到设计模型中去的。Import\_ReturnPO是作为进货退货的持久化对象被添加到设计模型中去的。CustomerPO是作为客户的持久化对象被添加到设计模型中去的。ExportPO是作为销售的持久化对象被添加到设计模型中去的。Export\_ReturnPO是作为销售退货的持久化对象被添加到设计模型中去的。SystemlogInfo、InvoiceInfo、CustomerInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Salesbl模块的设计如图



Salesbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| SalesController | 负责实现销售界面所需要的服务。 |
| Sales | 销售的领域模型对象，拥有销售所持有的新增的商品、客户、单据信息，可以帮助完成销售界面所需要的服务。 |

(3)模块内部类的接口规范

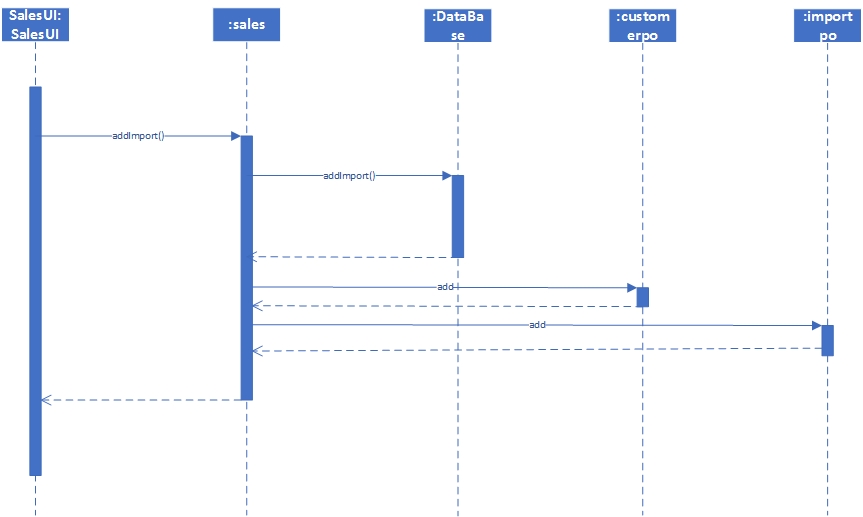
SalesController的接口规范

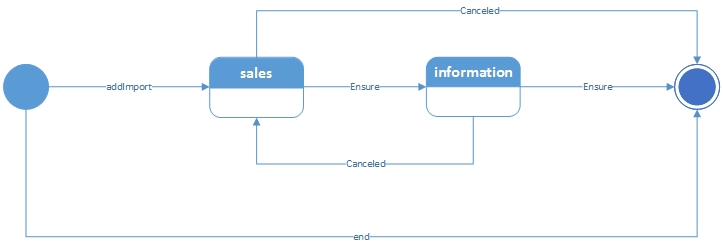
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SalesController.addCustomer | 语法 | Public int addCustomer\_up (Customer customerVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的addCustomer\_up方法 |
| SalesController.delCustomer | 语法 | Public int delCustomer\_up(CustomerVO customerVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的delCustomer\_up方法 |
| SalesController.updateCustomer | 语法 | Public int updateCustomer\_up(CustomerVO vo1,CustomerVO vo2) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的updateCustomer\_up方法 |
| SalesController.searchCustomer | 语法 | //模糊查找客户  Public ArrayList<CustomerVO> searchFuzzyCustomer\_up(String word)  //根据客户姓名精确查找客户  Public CustomerVO searchExactCustomer\_up(String name) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的searchFuzzyCustomer\_up方法  //调用Sales领域对象的searchExactCustomer\_up方法 |
| SalesController.addImport | 语法 | Public int addImport\_up  (ImportMenuVO importMenuVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的addImport\_up方法 |
| SalesController.addImport\_Return | 语法 | Public int addImport\_Return\_up  (ImportMenuVO importMenuVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的addImport\_Return\_up方法 |
| SalesController.addExport | 语法 | Public int addExport\_up  (ExportMenuVO exportMenuVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的addExport\_up方法 |
| SalesController.addExport\_Return | 语法 | public int addExport\_Return\_up  (ExportMenuVO exportMenuVO); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的addExport\_Return\_up方法 |
| SalesController.getImport\_Note | 语法 | public String getImportNote\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getImportNote\_up方法 |
| SalesController.getImport\_Return\_Note | 语法 | public String  getImport\_ReturnNote\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getImport\_ReturnNote\_up方法 |
| SalesController.getExport\_Return\_Note | 语法 | public String  getExport\_ReturnNote\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getExport\_ReturnNote\_up方法 |
| Sales.getExport\_Note | 语法 | public String getExportNote\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getExportNote\_up方法 |
| SalesController.searchNote | 语法 | // 根据编号查找进货单  public ImportMenuVO searchImportNote\_up(String note);  // 根据编号查找销售单  public ExportMenuVO searchExportNote\_up(String note);  // 根据编号查找进货退货单  public ImportMenuVO searchImport\_ReturnNote\_up(String note);    // 根据编号查找销售退货单  public ExportMenuVO searchExport\_ReturnNote\_up(String note); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的searchImportNote\_up方法  //调用Sales领域对象的searchExportNote\_up方法  //调用Sales领域对象的searchImport\_ReturnNote\_up方法  //调用Sales领域对象的searchExport\_ReturnNote\_up方法 |
| SalesController.getCustomer | 语法 | // 获得所有供货商信息  public ArrayList<CustomerVO> getAllImportCustomer\_up();  // 获得所有销售客户信息  public ArrayList<CustomerVO> getAllExportCustomer\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的getAllImportCustomer\_up方法  //调用Sales领域对象的getAllExportCustomer\_up方法 |
| SalesController.getReturnMaxNumber | 语法 | // 根据进货编号查找进货可退货最大数量  public int getImport\_ReturnMaxNumber\_up(String old\_note);  // 根据销售编号查找销售可退货最大数量  public int getExport\_ReturnMaxNumber\_up(String old\_note); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的getImport\_ReturnMaxNumber\_up方法  //调用Sales领域对象的getExport\_ReturnMaxNumber\_up方法 |
| SalesController.getAllCommodity | 语法 | public ArrayList<CommodityVO> getAllCommodity\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getAllCommodity\_up方法 |
| SalesController.getCommodity | 语法 | public CommodityVO getCommodity\_up(String name, String type); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getCommodity\_up方法 |
| SalesController.getClerkDiscount | 语法 | public int getClerkDiscount\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getClerkDiscount\_up方法 |
| SalesController.getDiscount | 语法 | public int getDiscount\_up(int money, int level); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getDiscount\_up方法 |
| SalesController.getVoucher | 语法 | public int getVoucher\_up(int money, int level); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getVoucher\_up方法 |
| SalesController.getProGift | 语法 | public CommodityVO getProGift\_up(int money, int level); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getProGift\_up方法 |
| SalesController.getCustomerNote | 语法 | Public String getCustomerNote\_up() |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Sales领域对象的getCustomerNote\_up方法 |
| SalesController.addDraftBill | 语法 | //添加草稿进货单  Public int addDraftImport\_up (ImportMenuVO importMenuVO)  //添加草稿进货退货单  Public int addDraftImport\_Return\_up (ImportMenuVO importMenuVO)  //添加草稿销售单  Public int addDraftExport\_up (ExportMenuVO exportMenuVO)  //添加草稿销售退货单  Public int addDraftExport\_Return\_up (ExportMenuVO exportMenuVO) |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的addDraftImport\_up方法  //调用Sales领域对象的addDraftImport\_Return\_up方法  //调用Sales领域对象的addDraftExport\_up方法  //调用Sales领域对象的addDraftExport\_Return\_up方法 |
| SalesController.getDraftBill | 语法 | //获得所有的草稿进货单  Public ArrayList<ImportMenuVO> getAllDraftImport\_up()  //获得所有的草稿进货退货单  public ArrayList<ImportMenuVO> getAllDraftImport\_Return\_up();  // 获得所有的草稿销售单  public ArrayList<ExportMenuVO> getAllDraftExport\_up ();  // 获得所有的草稿销售退货单  public ArrayList<ExportMenuVO> getAllDraftExport\_Return\_up(); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的getAllDraftImport\_up方法  //调用Sales领域对象的getAllDraftImport\_Return\_up方法  //调用Sales领域对象的getAllDraftExport\_up方法  //调用Sales领域对象的getAllDraftExport\_Return\_up方法 |
| SalesController.searchDraftBill | 语法 | // 根据编号查找草稿进货单  public ImportMenuVO searchDraftImport\_up(String note);  // 根据编号查找草稿进货退货单  public ImportMenuVO searchDraftImport\_Return\_up(String note);  // 根据编号查找草稿销售退货单  public ExportMenuVO searchDraftExport\_Return\_up(String note);  // 根据编号查找草稿销售单  public ExportMenuVO searchDraftExport\_up(String note); |
| 前置条件 | 已创建一个Sales领域对象，并且输入符合输入规则 |
| 后置条件 | //调用Sales领域对象的searchDraftImport\_up方法  //调用Sales领域对象的searchDraftImport\_Return\_up方法  //调用Sales领域对象的searchDraftExport\_Return\_up方法  //调用Sales领域对象的searchDraftExport\_up方法 |
| 需要的服务 | | |
| Sales.addCustomer(CustomerPO po) | 添加客户对象 | |
| Sales.delCustomer(CustomerPO po) | 删除客户对象 | |
| Sales.updateCustomer(CustomerPO po1, CustomerPO po2) | 更新客户对象 | |
| Sales.findCustomer(String name) | 根据name查找客户对象 | |
| Sales.getCustomerNote() | 获得要新增客户编号对象 | |
| Sales.addImport(ImportPO po) | 添加进货单对象 | |
| Sales.addImport\_Return(Import\_ReturnPO po) | 添加进货退货单对象 | |
| Sales.addExport(ExportPO po) | 添加销售单对象 | |
| Sales.addExport\_Return(Export\_ReturnPO po) | 添加销售退货单对象 | |
| Sales.getAllCustomer() | 获得所有的客户对象 | |
| Sales.getAllImport() | 获得所有的进货单I对象 | |
| Sales.getAllImport\_Return() | 获得所有的进货退货单对象 | |
| Sales.getAllExport() | 获得所有的销售单对象 | |
| Sales.getAllExport\_Return() | 获得所有的销售退货单O对象 | |
| Sales.getImportNote() | 获得要新增的进货单的单据编号 | |
| Sales.getImport\_ReturnNote() | 获得要新增的进货退货单的单据编号 | |
| Sales.getExportNote() | 获得要新增的销售单的单据编号 | |
| Sales.getExport\_ReturnNote() | 获得要新增的销售退货单的单据编号 | |
| Sales.getImport(String note) | 根据单据编号进行查找单一的进货单对象 | |
| Sales.getExport(String note) | 根据单据编号进行查找单一的销售单对象 | |
| Sales.getImport\_Return(String note) | 根据单据编号进行查找单一的进货退货单对象 | |
| Sales.getExport\_Return(String note) | 根据单据编号进行查找单一的销售退货单对象 | |
| Sale.getAllImportCustomer() | 获得所有进货商 | |
| Sale.getAllExportCustomer() | 获得所有销售商 | |
| Sale.addDraftImport(ImportPO po) | 添加草稿进货单 | |
| Sale.addDraftImport\_Return(Import\_ReturnPO po) | 添加草稿进货退货单 | |
| Sale. addDraftExport(ExportPO po) | 添加草稿销售单 | |
| Sale.addDraftExport\_Return(Export\_ReturnPO po) | 添加草稿销售退货单 | |
| Sale.delDraftImport(String note) | 删除草稿进货单 | |
| Sale.delDraftImport\_Return(String note) | 删除草稿进货退货单 | |
| Sale.delDraftExport(String note) | 删除草稿销售单 | |
| Sale.delDraftExport\_Reutrn(String note) | 删除草稿销售退货单 | |
| Sale. getAllDraftImport() | 获得草稿进货单 | |
| Sale.getAllDraftImport\_Return() | 获得草稿进货退货单 | |
| Sale. getAllDraftExport() | 获得草稿销售单 | |
| Sale.getAllDraftExport\_Return() | 获得草稿销售退货单 | |
| Sale.getDraftImport(String note) | 根据单据编号查找草稿进货单 | |
| Sale.getDraftImport\_Return(String note) | 根据单据编号查找草稿进货退货单 | |
| Sale.getDraftExport(String note) | 根据单据编号查找草稿销售单 | |
| Sale.getDraftExport\_Return(String note) | 根据单据编号查找草稿销售退货单 | |
| Sale.getImport\_ReturnMaxNumber(String old\_note) | 根据单据编号获得最大进货退货数量 | |
| Sales.getExport\_ReturnMaxNumber(String old\_note) | 根据单据编号获得最大销售退货数量 | |
| Sales.clear() | 清除所有对象 | |

Salesbl的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Sales.addCustomer | 语法 | Public int addCustomer\_up(Customer customerVO) |
| 前置条件 | 系统启动添加客户，且输入合法。 |
| 后置条件 | 返回提示是否成功添加客户，返回为成功，否则为失败。 |
| Sales.delCustomer | 语法 | Public int delCustomer\_up(CustomerVO customerVO) |
| 前置条件 | 系统启动删除客户，切输入合法 |
| 后置条件 | 返回提示是否成功删除客户，返回0为成功，否则为失败。 |
| Sales.updateCustomer | 语法 | Public int updateCustomer\_up(CustomerVO vo1,CustomerVO vo2) |
| 前置条件 | 有客户的信息发生改变，需要更新,系统启动更新客户信息，且输入合法 |
| 后置条件 | 返回提示是否成功更新客户信息，返回0为成功，否则为失败。 |
| Sales.searchCustomer | 语法 | //模糊查找客户  Public ArrayList<CustomerVO> searchFuzzyCustomer\_up(String word)  //根据客户姓名精确查找客户  Public CustomerVO searchExactCustomer\_up(String name) |
| 前置条件 | 系统启动查找客户 |
| 后置条件 | 返回搜索结果，模糊查找返回一个CustomerVO类型的arrayList，精确查找返回查找的客户的CustomerVO，若未查找到相关信息，返回空的arrayList或customerVO |
| Sales.addImport | 语法 | Public int addImport\_up  (ImportMenuVO importMenuVO) |
| 前置条件 | 商品需要进货时，系统启动添加进货单 |
| 后置条件 | 系统返回添加进货单是否成功，成功返回0交由总经理审批，否则为失败。 |
| Sales.addImport\_Return | 语法 | Public int addImport\_Return\_up  (ImportMenuVO importMenuVO) |
| 前置条件 | 当已经进货某商品但是要进行退货时，系统启动进货退货，且输入合法 |
| 后置条件 | 系统返回添加进货退货单是否成功，成功返回0交由总经理审批，否则为失败。 |
| Sales.addExport | 语法 | Public int addExport\_up  (ExportMenuVO exportMenuVO) |
| 前置条件 | 发生销售时,系统启动添加销售单，且输入合法 |
| 后置条件 | 系统返回添加销售单是否成功，成功返回0交由总经理审批，否则为失败。 |
| Sales.addExport\_Return | 语法 | public int addExport\_Return\_up  (ExportMenuVO exportMenuVO); |
| 前置条件 | 当已经销售某商品后，但是发生销售退货时，系统启动添加销售退货单，且输入合法 |
| 后置条件 | 系统返回添加销售退货单是否成功，成功返回0交由总经理审批，否则为失败。 |
| Sales.getImport\_Note | 语法 | public String getImportNote\_up(); |
| 前置条件 | 当发生进货时，系统启动添加进货单 |
| 后置条件 | 系统返回要新添加的进货单的编号。 |
| Sales.getImport\_Return\_Note | 语法 | public String  getImport\_ReturnNote\_up(); |
| 前置条件 | 当发生进货退货时，系统启动添加进货退货单 |
| 后置条件 | 系统返回要新添加的进货退货单的编号。 |
| Sales.getExport\_Return\_Note | 语法 | public String  getExport\_ReturnNote\_up(); |
| 前置条件 | 当发生销售退货时，系统启动添加销售退货单 |
| 后置条件 | 系统返回要新添加的销售退货单的编号。 |
| Sales.getExport\_Note | 语法 | public String getExportNote\_up(); |
| 前置条件 | 当发生销售时，系统启动添加销售单 |
| 后置条件 | 系统返回要新添加的销售单的编号。 |
| Sales.searchNote | 语法 | // 根据编号查找进货单  public ImportMenuVO searchImportNote\_up(String note);  // 根据编号查找销售单  public ExportMenuVO searchExportNote\_up(String note);  // 根据编号查找进货退货单  public ImportMenuVO searchImport\_ReturnNote\_up(String note);    // 根据编号查找销售退货单  public ExportMenuVO searchExport\_ReturnNote\_up(String note); |
| 前置条件 | 需要通过单据编号查询单据具体信息 |
| 后置条件 | 系统根据编号查询到单据具体信息，并返回单据具体信息 |
| Sales.getCustomer | 语法 | // 获得所有供货商信息  public ArrayList<CustomerVO> getAllImportCustomer\_up();  // 获得所有销售客户信息  public ArrayList<CustomerVO> getAllExportCustomer\_up(); |
| 前置条件 | 无 |
| 后置条件 | 系统返回当前所有供货商或销售商的信息 |
| Sales.getReturnMaxNumber | 语法 | // 根据进货编号查找进货可退货最大数量  public int getImport\_ReturnMaxNumber\_up(String old\_note);  // 根据销售编号查找销售可退货最大数量  public int getExport\_ReturnMaxNumber\_up(String old\_note); |
| 前置条件 | 发生销售退货，或进货退货时 |
| 后置条件 | 系统根据编号查找并返回可以退货的最大数量 |
| Sales.getAllCommodity | 语法 | public ArrayList<CommodityVO> getAllCommodity\_up(); |
| 前置条件 | 发生销售，销售退货，进货，进货退货时 |
| 后置条件 | 系统返回当前所有商品的商品信息 |
| Sales.getCommodity | 语法 | public CommodityVO getCommodity\_up(String name, String type); |
| 前置条件 | 发生销售，销售退货，进货，进货退货时 |
| 后置条件 | 系统根据商品名称和型号查找并返回当前所有商品的商品信息 |
| Sales.getClerkDiscount | 语法 | public int getClerkDiscount\_up(); |
| 前置条件 | 销售人员或销售经理给客户折扣时 |
| 后置条件 | 系统返回当前操作员能修改的最大折扣 |
| Sales.getDiscount | 语法 | public int getDiscount\_up(int money, int level); |
| 前置条件 | 发生销售，且总经理制订了与本销售过程相关的折扣促销，且该销售符合该折扣促销策略。 |
| 后置条件 | 系统根据客户等级和金额查找并返回折扣促销金额金额 |
| Sales.getVoucher | 语法 | public int getVoucher\_up(int money, int level); |
| 前置条件 | 发生销售，且总经理制订了与本销售过程相关的代金券促销，且该销售符合代金券促销策略 |
| 后置条件 | 系统根据客户等级和销售金额查找并返回代金券促销策略金额 |
| Sales.getProGift | 语法 | public CommodityVO getProGift\_up(int money, int level); |
| 前置条件 | 发生销售，且总经理制订了与本销售过程相关的赠品促销，且该销售符合赠品促销策略 |
| 后置条件 | 系统根据客户等级和销售金额查找并返回赠品信息 |
| Sales.getCustomerNote | 语法 | Public String getCustomerNote\_up() |
| 前置条件 | 添加客户时，需要系统自动分配编号 |
| 后置条件 | 系统返回分配的编号 |
| Sales.addDraftBill | 语法 | //添加草稿进货单  Public int addDraftImport\_up (ImportMenuVO importMenuVO)  //添加草稿进货退货单  Public int addDraftImport\_Return\_up (ImportMenuVO importMenuVO)  //添加草稿销售单  Public int addDraftExport\_up (ExportMenuVO exportMenuVO)  //添加草稿销售退货单  Public int addDraftExport\_Return\_up (ExportMenuVO exportMenuVO) |
| 前置条件 | 系统需要添加草稿状态的进货单/销售单/进货退货单/销售退货单 |
| 后置条件 | 系统返回是否添加成功，0代表成功，否则代表不成功 |
| Sales.getDraftBill | 语法 | //获得所有的草稿进货单  Public ArrayList<ImportMenuVO> getAllDraftImport\_up()  //获得所有的草稿进货退货单  public ArrayList<ImportMenuVO> getAllDraftImport\_Return\_up();  // 获得所有的草稿销售单  public ArrayList<ExportMenuVO> getAllDraftExport\_up ();  // 获得所有的草稿销售退货单  public ArrayList<ExportMenuVO> getAllDraftExport\_Return\_up(); |
| 前置条件 | 销售过程中，销售人员要修改草稿单 |
| 后置条件 | 系统返回全部草稿单据 |
| Sales.searchDraftBill | 语法 | // 根据编号查找草稿进货单  public ImportMenuVO searchDraftImport\_up(String note);  // 根据编号查找草稿进货退货单  public ImportMenuVO searchDraftImport\_Return\_up(String note);  // 根据编号查找草稿销售退货单  public ExportMenuVO searchDraftExport\_Return\_up(String note);  // 根据编号查找草稿销售单  public ExportMenuVO searchDraftExport\_up(String note); |
| 前置条件 | 销售人员根据编号查找草稿单 |
| 后置条件 | 系统根据单据编号返回相应的草稿单据 |
| 需要的服务 | | |
| SalesDataService.addCustomer(CustomerPO po) | 添加客户CustomerPO对象 | |
| SalesDataService.delCustomer(CustomerPO po) | 删除客户CustomerPO对象 | |
| SalesDataService.updateCustomer(CustomerPO po1, CustomerPO po2) | 更新客户CustomerPO对象信息 | |
| SalesDataService.findCustomer(String name) | 根据name查找客户CustomerPO对象 | |
| SalesDataService.getCustomerNote() | 获得要新增客户编号 | |
| SalesDataService.addImport(ImportPO po) | 添加进货单ImportPO对象 | |
| SalesDataService.addImport\_Return(Import\_ReturnPO po) | 添加进货退货单Import\_ReturnPO对象 | |
| SalesDataService.addExport(ExportPO po) | 添加销售单ExportPO对象 | |
| SalesDataService.addExport\_Return(Export\_ReturnPO po) | 添加销售退货单Export\_ReturnPO对象 | |
| SalesDataService.getAllCustomer() | 获得所有的客户CustomerPO对象 | |
| SalesDataService.getAllImport() | 获得所有的进货单ImportPO对象 | |
| SalesDataService.getAllImport\_Return() | 获得所有的进货退货单Import\_ReturnPO对象 | |
| SalesDataService.getAllExport() | 获得所有的销售单ExportPO对象 | |
| SalesDataService.getAllExport\_Return() | 获得所有的销售退货单Export\_ReturnPO对象 | |
| SalesDataService.getImportNote() | 获得要新增的进货单的单据编号 | |
| SalesDataService.getImport\_ReturnNote() | 获得要新增的进货退货单的单据编号 | |
| SalesDataService.getExportNote() | 获得要新增的销售单的单据编号 | |
| SalesDataService.getExport\_ReturnNote() | 获得要新增的销售退货单的单据编号 | |
| SalesDataService.getImport(String note) | 根据单据编号进行查找单一的进货单对象 | |
| SalesDataService.getExport(String note) | 根据单据编号进行查找单一的销售单对象 | |
| SalesDataService.getImport\_Return(String note) | 根据单据编号进行查找单一的进货退货单对象 | |
| SalesDataService.getExport\_Return(String note) | 根据单据编号进行查找单一的销售退货单对象 | |
| SaleDataService.getAllImportCustomer() | 获得所有进货商 | |
| SaleDataService.getAllExportCustomer() | 获得所有销售商 | |
| SaleDataService.addDraftImport(ImportPO po) | 添加草稿进货单 | |
| SaleDataService .addDraftImport\_Return(Import\_ReturnPO po) | 添加草稿进货退货单 | |
| SaleDataService. addDraftExport(ExportPO po) | 添加草稿销售单 | |
| SaleDataService. addDraftExport\_Return(Export\_ReturnPO po) | 添加草稿销售退货单 | |
| SaleDataService.delDraftImport(String note) | 删除草稿进货单 | |
| SaleDataService.delDraftImport\_Return(String note) | 删除草稿进货退货单 | |
| SaleDataService.delDraftExport(String note) | 删除草稿销售单 | |
| SaleDataService.delDraftExport\_Reutrn(String note) | 删除草稿销售退货单 | |
| SaleDataService. getAllDraftImport() | 获得草稿进货单 | |
| SaleDataService.getAllDraftImport\_Return() | 获得草稿进货退货单 | |
| SaleDataService. getAllDraftExport() | 获得草稿销售单 | |
| SaleDataService.getAllDraftExport\_Return() | 获得草稿销售退货单 | |
| SaleDataService.getDraftImport(String note) | 根据单据编号查找草稿进货单 | |
| SaleDataService.getDraftImport\_Return(String note) | 根据单据编号查找草稿进货退货单 | |
| SaleDataService.getDraftExport(String note) | 根据单据编号查找草稿销售单 | |
| SaleDataService.getDraftExport\_Return(String note) | 根据单据编号查找草稿销售退货单 | |
| SalesDataService.clear() | 清除所有对象 | |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.8 systemlogbl模块

(1)模块概述

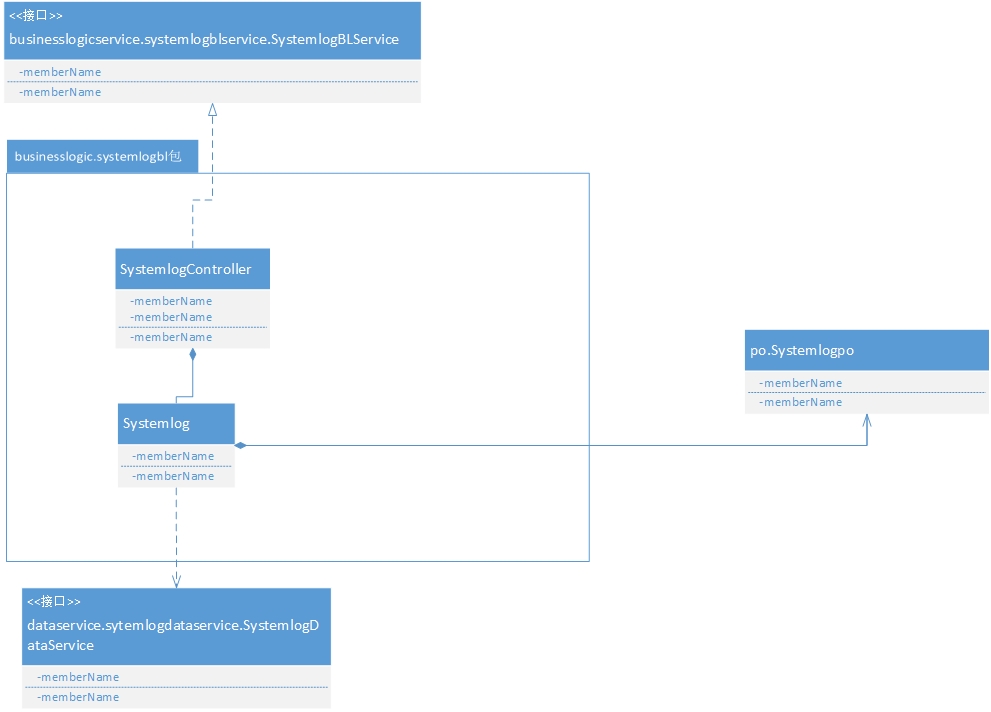
systemlogbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

systemlogbl模块的职责及接口参见软件系统结构描述文档表9中的systemlogbl的接口规范。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.systemlogblservice.SystemlogBLService接口。业务逻辑层和数据层之间添加dataservice.systemlogdataservice.SystemlogDataService接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了SystemlogController，这样SystemlogController会将对使用者的业务逻辑处理委托给Systemlog对象。SystemlogPO是作为使用者的持久化对象被添加到设计模型中去的。Systemloginfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Systemlogbl模块的设计如下图所示。



Systemlogbl模块各个类的职责如下表所示。

Systemlogbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| SystemlogController | 负责实现系统日志管理界面所需要的服务 |
| Systemlog | 系统日志管理的领域模型对象，拥有时间和操作等信息，可以帮助完成系统日志管理界面所需要的服务 |

(3)模块内部类的接口规范

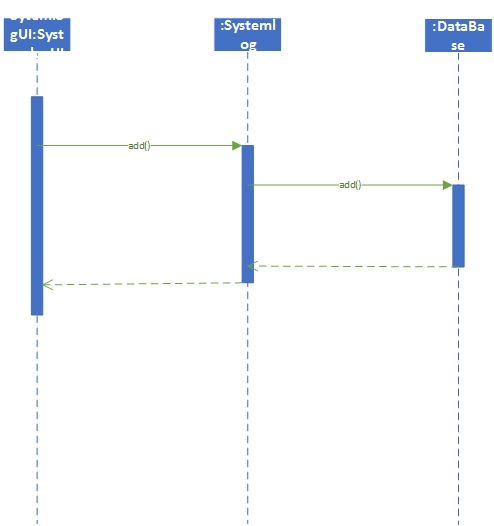
**SystemlogController的接口规格**

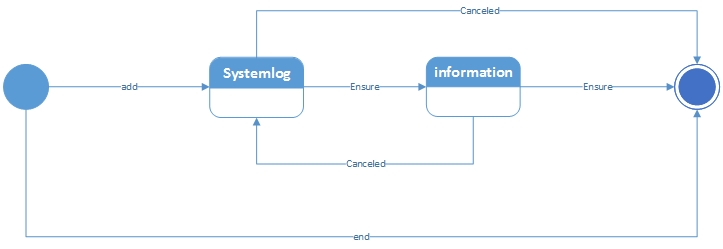
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SystemlogController.show | 语法 | public ArrayList<SystemlogVO> show\_up(); |
| 前置条件 | 已创建一个Systemlog领域对象，输入符合规则 |
| 后置条件 | 调用Systemlog领域对象的show方法 |
| SystemlogController.add | 语法 | Public int add(String word) |
| 前置条件 | 已创建一个Systemlog领域对象，输入符合规则 |
| 后置条件 | 调用Systemlog领域对象的add方法 |
| SystemlogController.exportExcel | 语法 | public int exportExcel\_up(ArrayList<SystemlogVO> sys); |
| 前置条件 | 已创建一个Systemlog领域对象，输入符合规则 |
| 后置条件 | 调用Systemlog领域对象的exportExcel方法 |
| 需要的服务 | | |
| Systemlog.show() | 显示日志内容 | |
| Systemlog.add(String word) | 添加日志内容 | |
| Systemlog. exportExcel\_up(ArrayList<SystemlogVO> sys); | 导出日志内容 | |

**Systemlog的接口规格**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Systemlog.show | 语法 | public ArrayList<SystemlogVO> show\_up(); |
| 前置条件 | 操作人员为财务人员或总经理 |
| 后置条件 | 显示日志内容，返回是否成功 |
| Systemlog.add | 语法 | Public int add(String word) |
| 前置条件 | 其他人员完成操作后，发送操作记录并存储 |
| 后置条件 | 添加日志内容，返回是否成功 |
| Systemlog.exportExcel | 语法 | public int exportExcel\_up(ArrayList<SystemlogVO> sys); |
| 前置条件 | 无 |
| 后置条件 | 导出systemLog的excel，默认地址为D盘目录下 |
| 需要的服务 | | |
| SystemlogDataService.ge() | 获得日志SystemlogPO对象 | |
| SystemlogDataService.add(SystemlogPO po) | 添加日志SystemlogPO对象 | |
| SystemlogDataService.clear() | 清空日志对象 | |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### 4.2.9 userbl模块

(1)模块概述

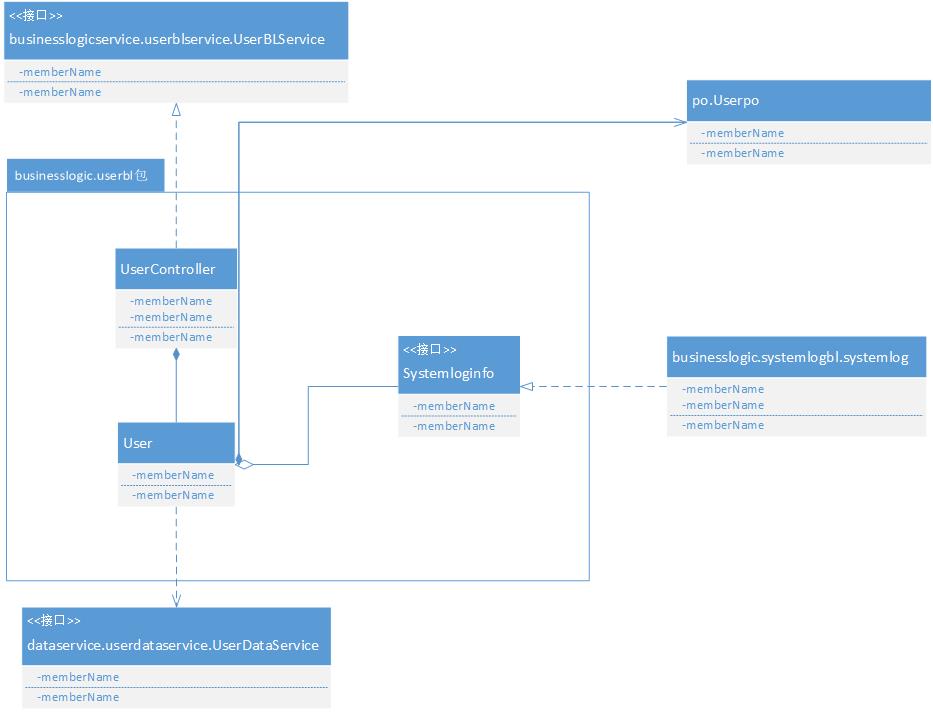
userbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

userbl模块的职责及接口参见软件系统结构描述文档表9中的userbl的接口规范。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.userblservice.UserBLService接口。业务逻辑层和数据层之间添加dataservice.userdataservice.UserDataService接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了UserController，这样UserController会将对使用者的业务逻辑处理委托给User对象。UserPO是作为使用者的持久化对象被添加到设计模型中去的。Userinfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Userbl模块的设计如下图所示。



Userbl模块各个类的职责如下表所示。

Userbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务 |
| User | 用户管理的领域模型对象，拥有用户账号、密码、职责等信息，可以帮助完成用户管理界面所需要的服务 |

(3)模块内部类的接口规范

UserController和User的借口规范如下表所示

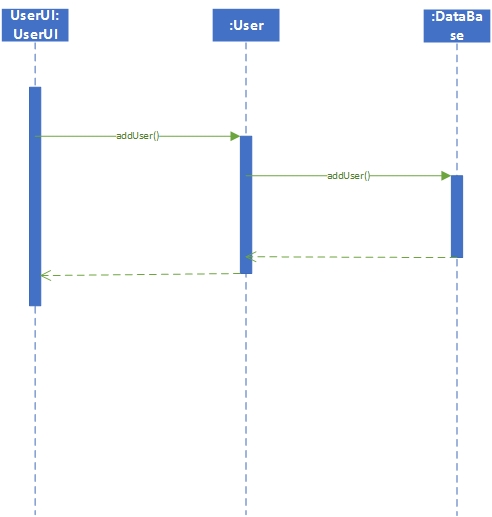
**UserController的接口规范**

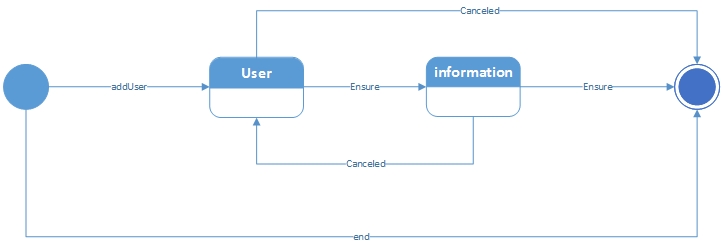
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| UserController.addUser | 语法 | public int addUser\_up(UserVO vo); |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的addUser方法 |
| UserController.delUser | 语法 | public int delUser\_up(UserVO vo); |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的delUser方法 |
| UserController.show | 语法 | public ArrayList<UserVO> show\_up(); |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的show方法 |
| UserController.login | 语法 | public int login\_up(String name, String password); |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的login方法 |
| UserController.getUserNote | 语法 | public String getUserNote(); |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的getUserNote方法 |
| UserController.searchUser | 语法 | public ArrayList<UserVO> searchUser\_up(String name); |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的searchUser方法 |
| 需要的服务 | | |
| User.addUser(UserPO po) | 添加用户 | |
| User.delUser(UserPO po) | 删除用户 | |
| User.show() | 显示用户信息 | |
| User.login() | 登陆 | |
| User.getUserNote() | 获得用户编号 | |
| User.searchUser(String name) | 根据名称查找用户 | |

**User的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| User.addUser | 语法 | public int addUser\_up(UserVO vo); |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 完成新账号的添加，可以开始使用 |
| User.delUser | 语法 | public int delUser\_up(UserVO vo); |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 删除账号信息，注销用户 |
| User.show | 语法 | public ArrayList<UserVO> show\_up(); |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 返回显示当前所有的身份信息 |
| User.login | 语法 | public int login\_up(String name, String password); |
| 前置条件 | 有用户请求登陆 |
| 后置条件 | 验证账号与密码，返回是否成功 |
| User.getUserNote | 语法 | public String getUserNote(); |
| 前置条件 | 管理员添加新用户 |
| 后置条件 | 系统返回新用户的编号 |
| User.searchUser | 语法 | public ArrayList<UserVO> searchUser\_up(String name); |
| 前置条件 | 无 |
| 后置条件 | 系统根据用户名称查找并返回用户信息 |
| 需要的服务 | | |
| UserDataService.addUser(UserPO po) | 添加一个UserPO对象 | |
| UserDataService.delUser(String name) | 根据姓名查找到一个UserPO对象并删除 | |
| UserDataService.getUser(String name) | 根据名字查找到UserPO对象 | |
| UserDataService.show() | 获得所有UserPO对象 | |
| UserDataService.clear() | 清空所有对象 | |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

## 4.3服务器端数据层分解

数据层的开发包图参见本文档末尾

### 4.3.1 数据层模块的描述

由于数据是存储在服务器上的，客户端访问数据必须通过网络通信层，具体是由客户端的业务逻辑层和服务器端的数据层通过网络通信层进行通信。

服务器端数据层主要负责数据的持久化及数据访问接口。对于持久化数据的集体载入、集体保存和增、删、改、查。这些服务由DataService接口提供。本系统程序的持久化数据的保存以序列化文件的形式实现。

### 4.3.2数据层模块的分解

#### 4.3.2.1 commoditydata模块

(1)整体结构

commoditydata模块的设计如图：

commoditydata模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**CommodityData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| CommodityDataService.findGood | 语法 | Public CommodityPO findGood(String name, String type) |
| 前置条件 | 无 |
| 后置条件 | 按名称和型号查找并返回相应的CommodityPO结果 |
| CommodityDataService.addGood | 语法 | Public void addGood(CommodityPO po1, SortPO po2) |
| 前置条件 | 序列化文件中不存在相同的CommodityPO对象，且存在一个SortPO对象可以添加CommodityPO对象 |
| 后置条件 | 添加一个CommodityPO对象 |
| CommodityDataService.delGood | 语法 | Public void delGood(CommodityPO po) |
| 前置条件 | 序列化文件中存在相同的CommodityPO对象，此商品的库存数量为0 |
| 后置条件 | 删除此CommodityPO对象 |
| CommodityDataService.updateGood | 语法 | //更新商品信息  Public void updateGood(CommodityPO po1, CommodityPO po2)  //更新商品所属分类  public boolean updateGood(CommodityPO po1,SortPO po2) throws RemoteException; |
| 前置条件 | 一个CommodityPO对象需要更新 |
| 后置条件 | 更新CommodityPO信息 |
| CommodityDataService.getGoodNote | 语法 | public String getGoodNote(SortPO po1) |
| 前置条件 | 需要添加商品时 |
| 后置条件 | 获得商品编号 |
| CommodityDataService.addSort | 语法 | public boolean addSort(SortPO po1,SortPO po2) |
| 前置条件 | 序列化文件中不存在相同名称的SortPOpo1，且存在SortPO对象po2 |
| 后置条件 | 添加一个SortPO对象 |
| CommodityDataService.delSort(SortPO po) | 语法 | public boolean delSort(SortPO po) |
| 前置条件 | 序列化文件中存在一个SortPO对象需要删除 |
| 后置条件 | 删除这个SortPO对象 |
| CommodityDataService.updateSort | 语法 | //改变分类所属分类  public boolean updateSort\_Mov(SortPO po1,SortPO po2)  //改变分类具体信息  public boolean updateSort(SortPO po1,SortPO po2) |
| 前置条件 | 序列化文件中存在一个SortPO对象需要更新 |
| 后置条件 | 更新这个SortPO对象信息 |
| CommodityDataService.getSortNote | 语法 | public String getSortNote(SortPO po1) |
| 前置条件 | 需要添加一个SortPO对象时 |
| 后置条件 | 获得一个分类的编号 |
| CommodityDataService.addSendGift | 语法 | public boolean addSendGift(SendGiftPO po) |
| 前置条件 | 序列化文件不存在相同名称与型号的GiftPO，数量小于库存内相同名称与型号CommodityPO对象的数量 |
| 后置条件 | 序列化文件添加GiftPO对象，库存中相同名称与型号的CommodityPO对象数量减少 |
| CommodityDataService.findSendGift | 语法 | public SendGiftPO findSendGift(String note) public SendGiftPO findSendGift(SendGiftPO po) |
| 前置条件 | 无 |
| 后置条件 | 根据名字和型号查找并返回SendGiftPO |
| CommodityDataService.delSendGift | 语法 | public boolean delSendGift(SendGiftPO po)  public boolean delSendGift(String note) |
| 前置条件 | 序列化文件中存在相同名称与型号的GiftPO对象 |
| 后置条件 | 删除GiftPO对象，库存中相同名称与型号的CommodityPO数量增加 |
| CommodityDataService.addGift | 语法 | public boolean addGift(CommodityPO po,int number) |
| 前置条件 | 销售人员赠送商品 |
| 后置条件 | 序列化文件中添加赠品 |
| CommodityDataService.findGift | 语法 | public CommodityPO findGift(CommodityPO po)  public CommodityPO findGift(String name,String type) |
| 前置条件 | 无 |
| 后置条件 | 序列化文件中查找销售人员的赠品CommodityPO |
| CommodityDataService.delGift | 语法 | public boolean delGift(CommodityPO po) |
| 前置条件 | 序列化文件中存在销售人员的赠品需要删除 |
| 后置条件 | 从序列化文件中删除该赠品 |
| CommodityDataService.getAllGift | 语法 | public ArrayList<CommodityPO> getAllGift() |
| 前置条件 | 无 |
| 后置条件 | 从序列化文件中获得所有销售人员的赠品CommodityPO对象 |
| CommodityDataService.getSendGiftNote | 语法 | **public** String getSendGiftNote() |
| 前置条件 | 无 |
| 后置条件 | 获得一个赠品的编号 |
| CommodityDataService.getAllSendGift | 语法 | public ArrayList<SendGiftPO> getAllSendGift() |
| 前置条件 | 无 |
| 后置条件 | 获得所有赠品SendGiftPO对象 |
| CommodityDataService.getAll | 语法 | public ArrayList<CommodityPO> getAll() |
| 前置条件 | 无 |
| 后置条件 | 获得所有的CommodityPO对象 |
| CommodityDataService.getAllSort | 语法 | public ArrayList<SortPO> getAllSort() |
| 前置条件 | 无 |
| 后置条件 | 获得所有SortPO对象 |
| CommodityDataService.addWarn | 语法 | public boolean addWarn(WarnPO po) |
| 前置条件 | 库存报警 |
| 后置条件 | 添加一个库存报警的WarnPO对象 |
| CommodityDateService.addPatch | 语法 | public boolean addPatch(PatchPO po) |
| 前置条件 | 库存存在报溢报损 |
| 后置条件 | 序列化文件中添加一个PatchPO对象 |
| CommodityDataService.delPatch | 语法 | public boolean delPatch(String note) |
| 前置条件 | 存在一个报溢报损单PatchPO对象需要删除 |
| 后置条件 | 查找并删除该报溢报损单 |
| CommodityDataService.getPatch | 语法 | public PatchPO getPatch(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据单据编号获得一个报溢报损单PatchPO对象 |
| CommodityDataService.getAllPatch | 语法 | public ArrayList<PatchPO> getAllPatch() |
| 前置条件 | 无 |
| 后置条件 | 获得所有PatchPO对象 |
| CommodityDataService.getPatchNote | 语法 | public String getPatchNote() |
| 前置条件 | 添加报溢报损单PatchPO 对象时 |
| 后置条件 | 获得一个报溢报损单编号 |
| CommofityDataService.passPatch | 语法 | public boolean passPatch(PatchPO po) |
| 前置条件 | 存在一个PatchPO对象审批通过 |
| 后置条件 | 根据审批通过的PatchPO对象修改库存信息 |
| CommunityDataService.refusePatch | 语法 | public boolean refusePatch(String note) |
| 前置条件 | 存在一个PatchPO对象审批被拒绝 |
| 后置条件 | 根据审批拒绝的PatchPO对象修改库存信息 |
| CommodityDataService.passSendGift | 语法 | public boolean passSendGift(SendGiftPO po) |
| 前置条件 | 存在一个SendGiftPO对象审批通过 |
| 后置条件 | 根据审批通过的SendGiftPO对象修改库存信息 |
| CommodiyDataService.refuseSendGift | 语法 | public boolean refuseSendGift(String note) |
| 前置条件 | 存在一个SendGiftPO对象审批被拒绝 |
| 后置条件 | 根据审批拒绝的SendGiftPO对象修改库存信息 |
| CommodityDataService.passImport | 语法 | public boolean passImport(ImportPO po) |
| 前置条件 | 存在一个ImportPO对象审批通过 |
| 后置条件 | 根据审批通过的ImportPO对象修改库存信息 |
| CommodityDataService.passImport\_Return | 语法 | public boolean passImport\_Return(Import\_ReturnPO po) |
| 前置条件 | 存在一个Import\_ReturnPO对象审批通过 |
| 后置条件 | 根据审批通过的Import\_ReturnPO对象修改库存信息 |
| CommodityDataService.passExport\_Return | 语法 | public boolean passExport\_Return(Export\_ReturnPO po) |
| 前置条件 | 存在一个Export\_ReturnPO对象审批通过 |
| 后置条件 | 根据审批通过的Export\_ReturnPO对象修改库存信息 |
| CommodityDataService.passExport | 语法 | public boolean passExport(ExportPO po) |
| 前置条件 | 存在一个ExportPO对象审批通过 |
| 后置条件 | 根据审批通过的ExportPO对象修改库存信息 |
| CommodityDataService.addDraftPatch | 语法 | public boolean addDraftPatch(PatchPO po) |
| 前置条件 | 无 |
| 后置条件 | 添加一个草稿报溢报损单对象 |
| CommodityDataService.delDraftPatch | 语法 | public boolean delDraftPatch(String note) |
| 前置条件 | 存在一个草稿报溢报损单对象PatchPO需要删除 |
| 后置条件 | 删除一个草稿报溢报损单对象PatchPO |
| CommodityDataService.getDraftPatch | 语法 | public PatchPO getDraftPatch(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找一个草稿报溢报损单对象PatchPO |
| CommodityDataService.getAllDraftPatch | 语法 | public ArrayList<PatchPO> getAllDraftPatch() |
| 前置条件 | 无 |
| 后置条件 | 获得所有草稿报溢报损单对象PatchPO |
| CommodityDataService.clear() | 语法 | Public void clear() |
| 前置条件 | 无 |
| 后置条件 | 删除所有CommodityPO与SortPO对象 |

#### 4.3.2.2 salesdata模块

(1)整体结构

salesdata模块的设计如图：

salesdata模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**SalesData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SalesDataService.addCustomer | 语法 | Public void addCustomer(CustomerPO po) |
| 前置条件 | 序列化文件中不存在相同名称的CustomerPO对象 |
| 后置条件 | 序列化文件添加CustomerPO对象 |
| SalesDataService.delCustomer | 语法 | Public void delCustomer(CustomerPO po) |
| 前置条件 | 序列化文件中存在相同名称的CustomerPO对象，此客户的应付或应收为0 |
| 后置条件 | 序列化文件删除Customer对象 |
| SalesDataService.updateCustomer | 语法 | Public void updateCustomer(CustomerPO po1, CustomerPO po2) |
| 前置条件 | 序列化文件中存在相同名称的CustomerPO，输入的两个CustomerPO对象名称不一致 |
| 后置条件 | 序列化文件修改Customer对象 |
| SalesDataService.findCustomer | 语法 | Public CustomerPO findCustomer(String name) |
| 前置条件 | 无 |
| 后置条件 | 根据名称查找并返回CustomerPO对象 |
| SalesDataService.getCustomerNote | 语法 | public String getCustomerNote() |
| 前置条件 | 添加客户 |
| 后置条件 | 获得一个客户编号 |
| SalesDataService.addImport | 语法 | Public void addImport(ImportPO po) |
| 前置条件 | 进货单得到总经理审批通过 |
| 后置条件 | 序列化文件中添加ImportPO对象，完成进货单 |
| SalesDataService.addImport\_Return | 语法 | Public void addImport\_Return(Import\_ReturnPO po) |
| 前置条件 | 进货退货单得到总经理审批通过，库存中相同名称与型号的CommodityPOd对象数量大于退货数量 |
| 后置条件 | 序列化文件中添加Import\_ReturnPO对象，完成进货的退货 |
| SalesDataService.addExport | 语法 | Public void addExport(ExportPO po) |
| 前置条件 | 销售单得到总经理审批通过，库存中相同名称与型号的CommodityPO对象数量大于销售数量 |
| 后置条件 | 序列化文件中添加ExportPO对象，完成销售 |
| SalesDataService.addExport\_Return | 语法 | Public void addExport\_Return(Export\_ReturnPO po) |
| 前置条件 | 销售退货单得到总经理审批通过 |
| 后置条件 | 序列化文件中添加Export\_ReturnPO对象，完成销售的退货 |
| SalesDataService.getAllCustomer | 语法 | Public ArrayList<CustomerPO> getAllCustomer() |
| 前置条件 | 无 |
| 后置条件 | 返回所有CustomerPO对象 |
| SalesDataService.getAllImportCustomer | 语法 | public ArrayList<CustomerPO> getAllImportCustomer(); |
| 前置条件 | 添加付款单 |
| 后置条件 | 返回所有进货商CustomerPO对象 |
| SalesDataService.getAllExportCustomer | 语法 | public ArrayList<CustomerPO> getAllExportCustomer(); |
| 前置条件 | 添加收款单 |
| 后置条件 | 返回所有销售商CustomerPO对象 |
| SalesDataService.getAllImport | 语法 | Public ArrayList<ImportPO> getAllImport() |
| 前置条件 | 无 |
| 后置条件 | 返回所有ImportPO对象 |
| SalesDataService.getAllImport\_Return | 语法 | public ArrayList<Import\_ReturnPO> getAllImport\_Return() |
| 前置条件 | 无 |
| 后置条件 | 返回所有Import\_ReturnPO对象 |
| SalesDataService.getAllExport | 语法 | Public ArrayList<ExportPO> getAllDraftExport() |
| 前置条件 | 无 |
| 后置条件 | 返回所有ExportPO对象 |
| SalesDataService.getAllExport\_Return | 语法 | public ArrayList<Export\_ReturnPO> getAllExport\_Return() |
| 前置条件 | 无 |
| 后置条件 | 返回所有Export\_ReturnPO对象 |
| SalesDataService.clear | 语法 | Public void clear() |
| 前置条件 | 进行期初建账之前 |
| 后置条件 | 清空所有的CustomerPO，ImportPO，Import\_ReturnPO，ExportPO，Export\_ReturnPO对象 |
| SaleDataService.addDraftImport | 语法 | public boolean addDraftImport(ImportPO po) |
| 前置条件 | 销售人员选择添加草稿进货单 |
| 后置条件 | 序列化文件中添加一个草稿进货单ImportPO对象 |
| SaleDataService.addDraftImport\_Return | 语法 | public boolean addDraftImport\_Return(Import\_ReturnPO po) |
| 前置条件 | 销售人员选择添加草稿进货退货单 |
| 后置条件 | 序列化文件中添加一个草稿进货退货单Import\_ReturnPO对象 |
| SaleDataService.addDraftExport | 语法 | public boolean addDraftExport(ExportPO po) |
| 前置条件 | 销售人员选择添加草稿销售单 |
| 后置条件 | 序列化文件中添加一个草稿销售单ExportPO对象 |
| SaleDataService.addDraftExport\_Return | 语法 | public boolean addDraftExport\_Return(Export\_ReturnPO po) |
| 前置条件 | 销售人员选择添加草稿销售退货单Export\_ReturnPO |
| 后置条件 | 序列化文件中添加一个草稿销售退货单Export\_ReturnPO对象 |
| SaleDataService.delDraftImport | 语法 | public boolean delDraftImport(String note) |
| 前置条件 | 销售人员选择删除草稿进货单 |
| 后置条件 | 序列化文件中根据编号查找并删除草稿进货单ImportPO对象 |
| SaleDataService.delDraftImport\_Return | 语法 | public boolean delDraftImport\_Return(String note) |
| 前置条件 | 销售人员选择删除草稿进货退货单 |
| 后置条件 | 序列化文件中根据编号查找并删除草稿进货退货单Import\_ReturnPO对象 |
| SaleDataService.delDraftExport | 语法 | public boolean .delDraftExport(String note) |
| 前置条件 | 销售人员选择删除草稿销售单 |
| 后置条件 | 序列化文件中根据编号查找并删除草稿销售单ExportPO对象 |
| SaleDataService.delDraftExport\_Return | 语法 | public boolean delDraftExport\_Reutrn(String note) |
| 前置条件 | 销售人员选择删除草稿销售退货单 |
| 后置条件 | 序列化文件中根据编号查找并删除草稿销售退货单Export\_ReturnPO对象 |
| SaleDataService.getAllDraftImport | 语法 | public ArrayList<ImportPO> getAllDraftImport() |
| 前置条件 | 无 |
| 后置条件 | 获得所有的草稿进货单ImportPO对象 |
| SaleDataService.getAllDraftImport\_Return | 语法 | Public ArrayList<Import\_ReturnPO> getAllDraftImport\_Return() |
| 前置条件 | 无 |
| 后置条件 | 获得所有的草稿进货退货单Import\_ReturnPO对象 |
| SaleDataService.getAllDraftExport | 语法 | public ArrayList<ExportPO> getAllDraftExport() |
| 前置条件 | 无 |
| 后置条件 | 获得所有的草稿销售单ExportPO对象 |
| SaleDataService. getAllDraftExport\_Return | 语法 | Public ArrayList<Export\_ReturnPO> getAllDraftExport\_Return() |
| 前置条件 | 无 |
| 后置条件 | 获得所有的草稿销售退货单Export\_ReturnPO对象 |
| SaleDataService.getDraftImport | 语法 | public ImportPO getDraftImport(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个草稿进货单ImportPO对象 |
| SaleDataService.getDraftImport\_Return | 语法 | public Import\_ReturnPO getDraftImport\_Return(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个草稿进货退货单Import\_ReturnPO对象 |
| SaleDataService.getDraftExport | 语法 | public ExportPO getDraftExport(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个草稿销售单ExportPO对象 |
| SaleDataService.getDraftExport\_Return | 语法 | public Export\_ReturnPO getDraftExport\_Return(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个草稿销售退货单Export\_ReturnPO对象 |
| SaleDataService.getDraftExport\_Return(String note) | 语法 | public Export\_ReturnPO getDraftExport\_Return(String note) |
| 前置条件 | 销售人员根据编号查找草稿销售退货单 |
| 后置条件 | 根据编号返回草稿销售退货单 |
| SaleDataService.getImport | 语法 | public ImportPO getImport(String note) |
| 前置条件 | 销售人员根据编号查找进货单 |
| 后置条件 | 根据编号返回进货单 |
| SaleDataService.getImport\_Return | 语法 | Public Import\_ReturnPO getImport\_Return(String note) |
| 前置条件 | 销售人员根据编号查找进货退货单 |
| 后置条件 | 根据编号返回进货退货单 |
| SaleDataService. getExport | 语法 | public ExportPO getExport(String note) |
| 前置条件 | 销售人员根据编号查找销售单 |
| 后置条件 | 根据编号返回销售单 |
| SaleDataService.getExport\_Return | 语法 | public Export\_ReturnPO getExport\_Return(String note) |
| 前置条件 | 销售人员根据编号查找销售退货单 |
| 后置条件 | 根据编号返回销售退货单 |
| SaleDataService. getImportNote | 语法 | public String getImportNote() |
| 前置条件 | 销售人员请求添加进货单时 |
| 后置条件 | 系统返回进货单编号 |
| SaleDataService.getImport\_ReturnNote | 语法 | public String getImport\_ReturnNote() |
| 前置条件 | 销售人员请求添加进货退货单时 |
| 后置条件 | 系统返回进货退货单编号 |
| SaleDataService. getExportNote | 语法 | public String getExportNote() |
| 前置条件 | 销售人员请求添加销售单时 |
| 后置条件 | 系统返回销售单编号 |
| SaleDataService.getExport\_ReturnNote | 语法 | public String getExport\_ReturnNote() |
| 前置条件 | 销售人员请求添加销售退货单时 |
| 后置条件 | 系统返回销售退货单编号 |
| SaleDataService.getImport\_ReturnMaxNumber | 语法 | public int getImport\_ReturnMaxNumber(String old\_note) |
| 前置条件 | 销售人员进行进货退货时，请求将输入的退货数量与最大可退货数量进行比较 |
| 后置条件 | 返回可最大进货退货数量 |
| SaleDataService.getExport\_ReturnMaxNumber | 语法 | public int getExport\_ReturnMaxNumber(String old\_note) |
| 前置条件 | 销售人员进行销售退货时，请求将输入的退货数量与最大可退货数量进行比较 |
| 后置条件 | 返回可最大销售退货数量 |
| SalesDataService.clear | 语法 | public boolean clear() |
| 前置条件 | 销售人员选择清空所有对象 |
| 后置条件 | 序列化文件中清空所有对象 |

#### 4.3.2.3AccountData模块

(1)整体结构

AccountData模块的设计如图：

AccountData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**AccountData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| AccountDataService.getAccount | 语法 | Public AccountPO getAccount() |
| 前置条件 | 无 |
| 后置条件 | 返回当前账户AccountPO对象 |
| AccountDataService.addAccount | 语法 | Public void addAccount(AccountPO po) |
| 前置条件 | 序列化文件中不存在相同名称的AccountPO对象 |
| 后置条件 | 添加AccountPO对象 |
| AccountDataService.delAccount | 语法 | Public void delAccount(AccountPO po) |
| 前置条件 | 序列化文件中存在相同名称的AccountPO对象，此对象中应付和应收为0 |
| 后置条件 | 删除AccountPO对象 |
| AccountDataService.updateAccount | 语法 | Public void updateAccount(AccountPO po1, AccountPO po2) |
| 前置条件 | 序列化文件中存在相同名称的AccountPO对象，输入的两个AccountPO对象的名称相同 |
| 后置条件 | 更新AccountPO对象的信息 |
| AccountDataService.findAccount | 语法 | Public AccountPO findAccount(String name) |
| 前置条件 | 无 |
| 后置条件 | 根据名称查找并返回AccountPO对象 |
| AccountDataService.getAllAccount | 语法 | public ArrayList<AccountPO> getAllAccount() |
| 前置条件 | 无 |
| 后置条件 | 返回所有AccountPO对象 |
| AccountDataService.addMoney | 语法 | public boolean addMoney(AccountPO po,double money) |
| 前置条件 | 收款单通过总经理审批 |
| 后置条件 | 将AccountPO对象中的账户余额增加 |
| AccountDataService.delMoney | 语法 | public boolean delMoney(AccountPO po,double money) |
| 前置条件 | 付款单通过总经理审批 |
| 后置条件 | 将AccountPO对象中的账户余额减少 |
| AccountDataService.addReceipt | 语法 | Public void addReceipt(ReceiptPO po) |
| 前置条件 | 收款单得到总经理审批通过 |
| 后置条件 | 序列化文件中添加ReceiptPO对象，完成收款 |
| AccountDataService.addPayment | 语法 | Public void addPayment(PaymentPO po) |
| 前置条件 | 付款单得到总经理审批通过 |
| 后置条件 | 序列化文件中添加PaymentPO对象，完成付款 |
| AccountDataService.getAllReceipt | 语法 | public ArrayList<ReceiptPO> getAllReceipt() |
| 前置条件 | 无 |
| 后置条件 | 获得当前所有ReceiptPO对象 |
| AccountDataService.getAllPayment | 语法 | public ArrayList<PaymentPO> getAllPayment() |
| 前置条件 | 无 |
| 后置条件 | 获得当前所有PaymentPO对象 |
| AccountDataService.findReceipt | 语法 | public ReceiptPO findReceipt(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个ReceiptPO对象 |
| AccountDataService.findPayment | 语法 | public PaymentPO findPayment(String note) |
| 前置条件 | 无 |
| 后置条件 | 根据编号查找并返回一个PaymentPO对象 |
| AccountDataService.addDraftReceipt | 语法 | public boolean addDraftReceipt(ReceiptPO po) |
| 前置条件 | 无 |
| 后置条件 | 序列化文件中添加一个草稿收款单ReceiptPO对象 |
| AccountDataService.addDraftPayment | 语法 | public boolean addDraftPayment(PaymentPO po) |
| 前置条件 | 无 |
| 后置条件 | 序列化文件中添加一个草稿付款单PaymentPO对象 |
| AccountDataService.delDraftReceipt | 语法 | public boolean delDraftReceipt(String note) |
| 前置条件 | 存在一个草稿收款单ReceiptPO对象需要删除 |
| 后置条件 | 根据编号从序列化文件中查找并删除一个草稿收款单ReceiptPO对象 |
| AccountDataService.delDraftPayment | 语法 | public boolean delDraftPayment(String note) |
| 前置条件 | 存在一个草稿付款单PaymentPO需要删除 |
| 后置条件 | 根据编号从序列化文件中查找并删除一个草稿付款单PaymentPO对象 |
| AccountDataService.findDraftReceipt | 语法 | public ReceiptPO findDraftReceipt(String note) |
| 前置条件 | 无 |
| 后置条件 | 从序列化文件中根据编号查找并返回一个草稿收款单ReceiptPO对象 |
| AccountDataService.findDraftPayment | 语法 | Public PaymentPO findDraftPayment(String note) |
| 前置条件 | 无 |
| 后置条件 | 从序列化文件中根据编号查找并返回一个草稿付款单PaymentPO对象 |
| AccountDataService.getAllDraftReceipt | 语法 | public ArrayList<ReceiptPO> getAllDraftReceipt() |
| 前置条件 | 无 |
| 后置条件 | 获得所有草稿收款单ReceiptPO对象 |
| AccountDataService.getAllDraftPayment | 语法 | public ArrayList<PaymentPO> getAllDraftPayment() |
| 前置条件 | 无 |
| 后置条件 | 获得所有草稿付款单PaymentPO对象 |
| AccountDataService.getReceiptNote | 语法 | public String getReceiptNote() |
| 前置条件 | 添加一个草稿收款单 |
| 后置条件 | 获得一个草稿收款单编号 |
| AccountDataService.getPaymentNote | 语法 | public String getPaymentNote() |
| 前置条件 | 添加一个草稿付款单 |
| 后置条件 | 获得一个草稿付款单编号 |
| AccountDataService.passReceipt | 语法 | public boolean passReceipt(ReceiptPO po) |
| 前置条件 | 总经理审批通过一个收款单 |
| 后置条件 | 根据该ReceiptPO对象改变账户的金额 |
| AccountDataService.passPayment | 语法 | public boolean passPayment(PaymentPO po) |
| 前置条件 | 总经理审批通过一个付款单 |
| 后置条件 | 根据该PaymentPO对象改变账户金额 |
| AccountDataService.refuseReceipt | 语法 | public boolean refuseReceipt(String note) |
| 前置条件 | 总经理审批拒绝一个收款单 |
| 后置条件 | 根据编号在序列化文件中查询到收款单，并将状态设置为拒绝 |
| AccountDataService.refusePayment | 语法 | public boolean refusePayment(String note) |
| 前置条件 | 总经理审批拒绝一个付款单 |
| 后置条件 | 根据编号在序列化文件中查询到付款单，并将状态设置为拒绝 |
| AccountDataService. | 语法 |  |
| 前置条件 |  |
| 后置条件 |  |
| AccountDataService. | 语法 |  |
| 前置条件 |  |
| 后置条件 |  |
| AccountDataService. | 语法 |  |
| 前置条件 |  |
| 后置条件 |  |
| AccountDataService. | 语法 |  |
| 前置条件 |  |
| 后置条件 |  |
| AccountDataService.clear | 语法 | Public void clear() |
| 前置条件 | 进行期初建账之前 |
| 后置条件 | 清除所有AccountPO，ReceiptPO，PaymentPO对象 |

#### 4.3.2.4FinancialData模块

(1)整体结构

FinancialData模块的设计如图：

FinancialData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**FinancialData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| FinancialDataService.getSaleList | 语法 | public ArrayList<SaleListPO> getSaleList() |
| 前置条件 | 无 |
| 后置条件 | 返回所有SaleListPO对象 |
| FinancialDataService.getAllBill | 语法 | public ArrayList<AllBillPO> getAllBill() |
| 前置条件 | 无 |
| 后置条件 | 返回所有AllBillPO对象 |
| FinancialDataService.getOperatingCondition() | 语法 | public ArrayList<OperatingConditionPO> getOperatingCondition() |
| 前置条件 | 无 |
| 后置条件 | 返回所有OperatingConditionPO对象 |
| FinancialDataService.addSaleList | 语法 | public boolean addSaleList(SaleListPO po) |
| 前置条件 | 完成一个进货或销售单据的审批 |
| 后置条件 | 序列化文件添加一个SaleListPO对象 |
| FinancialDataService.addAllBill | 语法 | public boolean addAllBill(AllBillPO po) |
| 前置条件 | 完成任何单据的审批 |
| 后置条件 | 序列化文件添加一个AllBillPO对象 |
| FinancialDataService.addOperatingCondition | 语法 | public boolean addOperatingCondition(OperatingConditionPO po) |
| 前置条件 | 完成收付款单据的审批 |
| 后置条件 | 序列化文件 添加一个OperatingConditionPO对象 |
| FinancialDataService.clear() | 语法 | Public void clear() |
| 前置条件 | 进行期初建账之前 |
| 后置条件 | 清楚所有SaleListPO，AllBillPO，OperatingConditionPO对象 |

#### 4.3.2.5 InitializationData模块

(1)整体结构

InitializationData模块的设计如图：

InitializationData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**InitializationData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| InitializationDataService.getInformation | 语法 | Public InitializationPO getInformation() |
| 前置条件 | 无 |
| 后置条件 | 返回InitializationPO对象 |

#### 4.3.2.6PromotionData模块

(1)整体结构

PromotionData模块的设计如图：

PromotionData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**PromotionData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| PromotionDataService.addDiscount | 语法 | Public boolean addDiscount(DiscountPO po) |
| 前置条件 | 无 |
| 后置条件 | 添加或替换当前的DiscountPO对象 |
| PromotionDataService.addGift | 语法 | Public boolean addGift(ProGiftPO po) |
| 前置条件 | 无 |
| 后置条件 | 添加或替换当前的ProGiftPO对象 |
| PromotionDataService.addVoucher | 语法 | Public boolean addVoucher(VoucherPO po) |
| 前置条件 | 无 |
| 后置条件 | 添加或替换当前的VoucherPO对象 |
| PromotionDataService.delDiscount(int level) | 语法 | Public boolean delDiscount(int level) |
| 前置条件 | 当前存在处于有效期的DiscountPO对象 |
| 后置条件 | 清除DiscountPO对象 |
| PromotionDataService.delGift | 语法 | Public boolean delGift(int level) |
| 前置条件 | 当前存在处于有效期的ProGiftPO对象 |
| 后置条件 | 清除ProGiftPO对象 |
| PromotionDataService.delVoucher | 语法 | Public boolen delVoucher(int level) |
| 前置条件 | 当前存在处于有效期的VoucherPO对象 |
| 后置条件 | 清除VoucherPO对象 |
| PromotionDataservice.getDiscount | 语法 | Public DiscountPO getDiscount(int level) |
| 前置条件 | 无 |
| 后置条件 | 根据客户等级返回对应的DiscountPO对象 |
| PromotionDataService.getGift | 语法 | Public ProGiftPO getGift(int level) |
| 前置条件 | 无 |
| 后置条件 | 根据客户等级返回对应的ProGiftPO对象 |
| PromotionDataService.geVoucher | 语法 | Public VoucherPO getGift(int level) |
| 前置条件 | 无 |
| 后置条件 | 根据客户等级返回相应的VoucherPO对象 |
| PromotionDataService.showDiscount | 语法 | Public ArrayList<DiscountPO> showDiscount() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有的DiscountPO对象的arrayList |
| PromotionDataService.showProGift | 语法 | Public ArrayList<ProgiftPO> showProGift() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有的ProGiftPO对象的arraylist |
| PromotionDataService.showVoucher | 语法 | Public ArrayList<VouvherPO> showProGift() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有的VoucherPO对象的arraylist |
| PromotionDataService.clear() | 语法 | Public boolean clear() |
| 前置条件 | 期初建账 |
| 后置条件 | 清空所有促销策略，并返回是否成功清空 |

#### 4.3.2.7 InvoiceData模块

(1)整体结构

InvoiceData模块的设计如图：

InvoiceData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**InvoiceData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| InvoiceDataServoice.getInvoice | 语法 | Public InvoicePO getInvoice(String note) |
| 前置条件 | 无 |
| 后置条件 | 返回最新的待审批的InvoicePO对象 |
| InvoiceDataServoice.addInvoice | 语法 | Public void addInvoice(InvoicePO po) |
| 前置条件 | 有新的待审批单据生成 |
| 后置条件 | 添加InvoicePO对象 |
| InvoiceDataServoice.delInvoice | 语法 | Public void delInvoice(InvoicePO po) |
| 前置条件 | 存在相同编号的InvoicePO对象 |
| 后置条件 | 删除InvoicePO对象 |
| InvoiceDataService.getAllInvoice | 语法 | public ArrayList<InvoicePO> getAllInvoice() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有InvoicePO对象的arrayList |
| InvoiceDataService.passInvoice | 语法 | public boolean passInvoice(InvoicePO po) |
| 前置条件 | 有新的审批通过单据 |
| 后置条件 | 添加一个审批通过的单据InvoicePO对象 |
| InvoiceDataService.refuseInvoice | 语法 | public boolean refuseInvoice(InvoicePO po) |
| 前置条件 | 有新的审批拒绝单据 |
| 后置条件 | 添加一个审批拒绝的单据InvoicePO对象 |
| InvoiceDataService.getInvNote | 语法 | public String getInvNote() |
| 前置条件 | 无 |
| 后置条件 | 获得一个审批单据的编号 |
| InvoiceDataService.clear | 语法 | Public boolean clear() |
| 前置条件 | 无 |
| 后置条件 | 清除所有InvoicePO对象 |

#### 4.3.2.8 SystemlogData模块

(1)整体结构

SystemlogData模块的设计如图：

SystemlogData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**SystemlogData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SystemlogDataService.get | 语法 | public ArrayList<SystemlogPO> get() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有SystemlogPO对象的arrayList |
| SystemlogDataService.add | 语法 | public boolean add(SystemlogPO po) |
| 前置条件 | 有新的操作，需要记录 |
| 后置条件 | 添加SystemlogPO对象 |
| SystemlogDataService.clear | 语法 | public boolean clear() |
| 前置条件 | 无 |
| 后置条件 | 清除所有SystemlogPO对象 |

#### 4.3.2.9 UserData模块

(1)整体结构

UserData模块的设计如图：

UserData模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
|  |  |

(2)模块内部类的接口规范

**UserData的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| UserDataService.addUser | 语法 | Public boolean addUser(UserPO po) |
| 前置条件 | 不存在相同名称的UserPO对象 |
| 后置条件 | 添加UserPO对象 |
| UserDataService.delUser | 语法 | public boolean delUser(String string) |
| 前置条件 | 存在相同名称的UserPO对象 |
| 后置条件 | 根据用户名查找并删除UserPO对象，若用户正在登陆，则强制退出程序 |
| UserDataService.getUser | 语法 | Public UserPO getUser(String name) |
| 前置条件 | 无 |
| 后置条件 | 根据名称查找返回UserPO对象 |
| UserDataService.getUser\_Note | 语法 | public UserPO getUser\_Note(String note) |
| 前置条件 |  |
| 后置条件 |  |
| UserDataService.getNote | 语法 | public String getNote() |
| 前置条件 | 添加用户 |
| 后置条件 | 获得一个新的用户编号 |
| UserDataService.login | 语法 | public UserPO login(UserPO po) |
| 前置条件 | 有用户请求登陆 |
| 后置条件 | 返回用户的具体信息 |
| UserDataService.show() | 语法 | Public UserPO[] show() |
| 前置条件 | 无 |
| 后置条件 | 返回一个包含所有UserPO对象的数组 |
| UserDataService.clear() | 语法 | Public void clear() |
| 前置条件 | 无 |
| 后置条件 | 清除所有UserPO对象 |

# 5.依赖视角

下图是客户端和服务器端各自的包之间的依赖关系。

