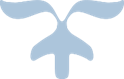


**快递物流系统**

**软件详细设计描述文档**



**V1.3 正式版**

**2015-11-07**

**噔海马小王子Team**

目录

[文档修改历史： 2](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982450)

[小组成员： 2](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982451)

[1.引言 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982452)

[1.1.编制目的 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982453)

[1.2.词汇表 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982454)

[1.3.参考资料 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982455)

[2.产品概述 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982456)

[3.体系结构设计概述 3](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982457)

[4.中层设计 4](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982458)

[4.1.客户端展示层模块的静态结构和动态职责 5](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982459)

[4.1.1.客户端展示层模块局部模块的职责 5](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982460)

[4.1.2.客户端展示层模块局部模块的接口规范 6](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982461)

[4.1.2.1.mainui模块的接口规范 6](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982462)

[4.1.2.2.userui模块的接口规范 6](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982463)

[4.1.2.3.nonuserui模块的接口规范 6](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982464)

[4.1.2.4.insititutionui模块的接口规范 6](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982465)

[4.1.2.5.commodityui模块的接口规范 7](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982466)

[4.1.2.6.logui模块的接口规范 7](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982467)

[4.1.2.7.sheetui模块的接口规范 7](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982468)

[4.1.2.8.begininfoui模块的接口规范 8](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982469)

[4.1.2.9.accountui模块的接口规范 8](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982470)

[4.1.2.10.staffmanageui模块的接口规范 8](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982471)

4.1.2.11. loginui模块的接口规范................................................................................ 8

[4.1.3.客户端展示层模块局部模块的设计原理 9](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982473)

[4.2.客户端业务逻辑层模块的静态结构和动态职责 9](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982474)

[4.2.1.客户端业务逻辑层模块局部模块的职责 9](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982475)

[4.2.2.客户端业务逻辑层模块局部模块的接口规范 10](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982476)

[4.2.2.1.userbl模块 10](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982477)

[4.2.2.2.nonuserbl模块 11](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982478)

[4.2.2.3.insititutionbl模块 13](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982479)

[4.2.2.4.commoditybl模块 14](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982480)

[4.2.2.5.logbl模块 16](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982481)

[4.2.2.6.sheetbl模块 18](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982482)

[4.2.2.7.begininfobl模块 20](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982483)

[4.2.2.8.accountbl模块 21](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982484)

[4.2.2.9.staffmanagebl模块 23](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982485)

4.2.2.10.. loginbl模块..................................................................................................24

4.2.2.11. positionbl模块..............................................................................................25

[4.2.3.客户端业务逻辑层模块局部的行为 27](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982487)

[4.2.3.1.userbl模块 27](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982488)

[4.2.3.2.nonuserbl模块 28](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982489)

[4.2.3.3.insititutionbl模块 29](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982490)

[4.2.3.4.commoditybl模块 30](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982491)

[4.2.3.5.logbl模块 32](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982492)

[4.2.3.6.sheetbl模块 35](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982493)

[4.2.3.7.begininfobl模块 36](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982494)

[4.2.3.8.accountbl模块 37](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982495)

[4.2.3.9.staffmanagebl模块 39](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982496)

4.2.3.9. loginbl模块.....................................................................................................40

4.2.3.10.positionbl模块...............................................................................................41

[4.2.4.客户端业务逻辑层模块的设计原理 42](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982498)

[4.3.客户端网络模块的静态结构和动态职责 42](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982499)

[4.3.1.客户端网络模块局部模块的职责 42](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982500)

[4.4.服务器端网络模块的静态结构和动态职责 43](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982501)

[4.4.1.服务器端网络模块局部模块的职责 43](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982502)

[4.4.2.服务器端网络模块局部模块的接口规范 43](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982503)

[4.5.服务器端数据层模块的静态结构和动态职责 44](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982504)

[4.5.1.服务器端数据层模块局部模块的职责 44](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982505)

[4.5.2.服务器端数据层模块局部模块的接口规范 45](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982506)

[4.5.3.服务器端数据层模块的设计原理 46](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982507)

[5.依赖关系 46](file:///C:\\Users\\Administrator\\Desktop\\大作业\\DDL6-11.05\\详细设计文档.doc" \l "_Toc402982508)

# 

**文档修改历史：**

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 修改原因 | 版本号 |
| 全体小组成员 | 2015/11/07 | 最初粗略草稿 | V1.0 草稿 |
| 全体小组成员 | 2015/11/16 | 界面层接口规范完善 | V1.1 正式版 |
| 曹姝玥 | 2016/1/3 | 修改部分接口规范 | V1.2正式版 |
| 李莹 | 2016/1/4 | 修改部分接口规范 | V1.3正式版 |

# 小组成员：

|  |  |
| --- | --- |
| **姓名** | **学号** |
| 曹姝玥 | 141250005 |
| 雷婷 | 141250061 |
| 李莹 | 141250065 |
| 徐海溦 | 141250158 |

# 引言

# 编制目的

本报告详细完成对快递物流系统的详细设计，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。

# 词汇表

|  |  |  |
| --- | --- | --- |
| **词汇名称** | **词汇含义** | **备注** |
| BL | 业务逻辑（business logic） | 无 |

# 参考资料

1. 软件工程与计算（卷二）——软件开发的技术基础。
2. 快递物流系统用例文档。
3. 快递物流系统软件需求规格说明文档。
4. 快递物流系统体系结构设计文档。

# 产品概述

参考进销存系统用例文档和进销存系统软件需求规格说明中对产品的概括描述。

# 体系结构设计概述

参考快递物流系统体系结构设计文档中对体系结构设计的概述。

# 中层设计

中层设计的基本思路依托了软件体系的结构视角。

在快递物流系统中，可以分别在客户端和服务器端都添加系统启动模块专门负责系统的初始化启动工作，并使它横跨系统的分层结构，可以直接访问到各层。客户端则具体分为展示层、业务逻辑层和网络模块，而服务器端则包含网络模块和数据层。具体的客户端和服务器端模块视图分别如图1和图2所示。客户端各层和服务器端各层的职责分别如表1和表2所示。



图1客户端模块视图



图2服务器端模块视图

**表1 客户端各层的职责**

|  |  |
| --- | --- |
| **层** | **职责** |
| 启动模块 | 负责初始化网络通信机制，启动用户界面。 |
| 用户界面层 | 基于窗口的进销存客户端用户界面。 |
| 业务逻辑层 | 对于用户界面的输入响应和业务处理逻辑。 |
| 客户端网络模块 | 利用Java RMI机制查找RMI服务 |

**表2 服务器端各层的职责**

|  |  |
| --- | --- |
| **层** | **职责** |
| 启动模块 | 负责初始化网络通信机制，启动用户界面。 |
| 用户界面层 | 基于窗口的进销存服务器用户界面。 |
| 业务逻辑层 | 对于用户界面的输入响应和业务处理逻辑。 |

每一层只是使用下方直接接触的层。层与层之间仅仅是通过接口的调用来完成的。

下面，根据以上对客户端和服务器的分解对各个模块进行分析与设计。

# 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10个模块组成,分别是mainui,userui,nonuserui, insititutionui,logui,commodityui,sheetui,begininfoui,accountui,staffmanageui。mainui模块负责界面的跳转；userui模块包含用户登录和修改密码界面。insititutionui模块包含账户的增删改查、查看经营情况表界面。adjustui模块包含库存报溢报损界面。customerui模块包含客户管理界面。commodityui模块包含商品管理、商品分类管理、库存盘点、库存报警界面。logui模块包含操作日志界面。sheetui模块包含所有单据界面。begininfoui模块包含初始化信息界面。accountui模块包含账户管理界面。staffmanageui模块包含包含员工信息管理界面。

# 客户端展示层模块局部模块的职责

客户端展示层模块的职责如下表所示。

**表3 客户端展示层模块的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Mainui | 负责界面的跳转 |
| Userui | 包含用户信息和修改密码界面 |
| NonUserui | 包含非用户信息管理界面 |
| Insititutionui | 包含机构管理界面 |
| Commodityui | 包含库存查看、库存分区、库存盘点、库存报警，新建出入库单界面 |
| Logui | 包含操作日志界面 |
| Sheetui | 包含所有单据和常量制定界面 |
| Begininfoui | 包含初始化信息界面 |
| Accountui | 包含账户管理界面 |
| Staffmanageui | 包含员工信息管理界面 |
| LogInui | 包含登录界面 |

# 客户端展示层模块局部模块的接口规范

# mainui模块的接口规范

mainui模块的接口规范如下表所示。

**表4 mainui模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| Mainui.init | 语法 | Public void init(String[] args) |
| 前置条件 | 无 |
| 后置条件 | 根据逻辑进行界面跳转 |

# userui模块的接口规范

userui模块的接口规范如下表所示。

**表5 userui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| UserController.getPersonalInfo | 查看个人帐号信息对象 |
| UserController.changePassword | 修改密码 |
| UserController.initial | 初始化 |
| UserController.changeInfo | 修改个人帐号信息 |
| UserController. endManagement | 确认管理 |

# insititutionui模块的接口规范

insititutionui模块的接口规范如下表所示。

**表6 insititutionui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| InstitutionController.getInstitutionInfo | 查看单一机构对象 |
| InstitutionController.addInstitution | 增加机构 |
| InstitutionController. changeInstitutionInfo | 修改机构信息 |
| InstitutionController. deleteInstitution | 删除机构 |
| InstitutionController.endManagement | 确认管理 |
| InstitutionController.getAllInstitution | 查看所有机构对象 |

# nonuserui模块的接口规范

nonuserui模块的接口规范如下表所示。

**表7 nonuserui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| NonUserController.getTruckInfo | 得到单个车辆信息 |
| NonUserController.getTruckByBusinessHall | 得到营业厅所属车辆信息 |
| NonUserController.getAllTruckInfo | 得到所有车辆信息 |
| NonUserController.addTruckInfo | 增加一个车辆 |
| NonUserController.deleteTruckInfo | 删除一个车辆 |
| NonUserController.changeTruckInfo | 修改车辆信息 |
| NonUserController.getDriverInfo | 得到单个司机信息 |
| NonUserController.getDriverByBusinessHall | 得到营业厅所属司机信息 |
| NonUserController.getAllDriverInfo | 得到所有司机信息 |
| NonUserController.addDriverInfo | 增加一个司机 |
| NonUserController.deleteDriverInfo | 删除一个司机 |
| NonUserController.changeDriverInfo | 修改司机信息 |
| NonUserController.getBussinessHallInfo | 得到单个营业厅信息 |
| NonUserController.getBussinessHallInfoByCity | 得到城市所属营业厅信息 |
| NonUserController.getAllBussinessHallInfo | 得到所有营业厅信息 |
| NonUserController.addBussinessHallInfo | 增加一个营业厅 |
| NonUserController.deleteBussinessHallInfo | 删除一个营业厅 |
| NonUserController.getIntermediateCenterInfo | 得到单个中转中心信息 |
| NonUserController.getAllIntermediateCenterInfo | 得到所有中转中心信息 |
| NonUserController.addIntermediateCenterInfo | 增加一个中转中心 |
| NonUserController.deleteIntermediateCenterInfo | 删除一个中转中心 |
|  |  |

# commodity模块的接口规范

commodity模块的接口规范如下表所示。

**表8 commodityui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| CommodityController.stockTaking | 库存盘点 |
| CommodityController.exportExcel | 导出Excel |
| CommodityController.changeAlarmScale | 修改警戒比例 |
| CommodityController.getAlarmScale | 获得警戒比例 |
| CommodityController.checkStorageMessage | 查看库存信息 |
| CommodityController. getStorageNum | 得到库存数量 |
| CommodityController.alarm | 库存报警 |
| CommodityController.divide | 库存分区 |
| CommodityController.endCommodityManagement | 结束库存操作 |
| CommodityController.changeStorageInInfo | 修改库存信息 |
| CommodityController. initStorageInfo | 初始化库存信息 |
|  |  |

# logui模块的接口规范

logui模块的接口规范如下表所示。

**表9 logui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| LogController.checkLog(String date) | 查看日志对象 |
| LogController.generateLog(String position, String name, String operation) | 生成日志对象 |

# sheetui模块的接口规范

sheetui模块的接口规范如下表所示。

**表10 sheetui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| SheetController.checkSheet(long ID) | 查看表格对象 |
| SheetController.examineSheet(long ID,boolean passed) | 审批表格 |
| SheetController.confirmSheet(SheetVO sheet,SheetType type) | 确认提交表格 |
| SheetController.formulateConstant(ArrayList<Double> distant, double price) | 确认制定常量 |

# begininfoui模块的接口规范

begininfoui模块的接口规范如下表所示。

**表11 begininfoui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| BeginInfoController.fillInfo(BeginInfoVO vo) | 创建期初信息对象 |
| BeginInfoController. getInfo() | 获得一个期初信息对象 |

# accountui模块的接口规范

accountui模块的接口规范如下表所示。

**表12 accountui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| AccountController.addAccount(String name,long num,double amount ) | 加入一个账户对象 |
| AccountController.delAccount(long num) | 删除一个账户对象 |
| AccountController.getAccountList() | 获得账户对象列表 |

# staffmanageui模块的接口规范

staffmanageui模块的接口规范如下表所示。

**表13 staffmanageui模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| StaffManageController.getStaffInfo | 查看单一员工信息对象 |
| StaffManageController.getAllStaff | 查看所有员工信息对象 |
| StaffManageController.addStaffInfo | 增加员工帐号 |
| StaffManageController.changeAuthority | 修改职位权限 |
| StaffManageController.deleteStaff | 删除员工帐号 |
| StaffManageController.endManagement | 确认管理 |

# LogInui模块的接口规范

LogInui模块的接口规范如下表所示。

**表13** LogInui**模块的接口规范**

|  |  |
| --- | --- |
| **需要的接口（需接口）** | |
| **服务名** | **服务** |
| LogInController.logIn | 登录 |
| LogInController. getCurrentUser | 获得当前登陆者信息 |

# 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。每个界面都通过mainui模块实现跳转。

客户端展示层模块使用了集中式的控制风格。每一个界面的职责从mainui之中分离并通过mainui模块实现跳转。

# 客户端业务逻辑层模块的静态结构和动态职责

客户端业务逻辑层由9个模块组成，分别是userbl,nonuserbl, insititutionbl,logbl,commoditybl,sheetbl,begininfobl,accountbl,staffmanagebl。userbl模块负责实现对应登录界面和密码修改界面的服务；nonuserbl模块负责实现对应司机车辆营业厅信息管理的服务；insititutionbl模块负责实现对应机构管理的服务；commoditybl模块负责实现对应库存查看、库存分区、库存盘点和库存报警的服务；logbl模块负责实现对应日志操作的服务；sheetbl模块负责实现对应所有单据操作的服务；begininfobl模块负责实现对应初始化信息的服务；accountbl模块负责实现对应账户管理的服务；staffmanagebl模块负责实现对应员工信息管理的服务。

# 客户端业务逻辑层模块局部模块的职责

客户端业务逻辑层层模块的职责如下表所示。

**表14 客户端业务逻辑层模块的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Userbl | 负责实现对应用户信息和密码修改界面的服务 |
| NonUserbl | 负责实现对应司机车辆营业厅信息管理的服务 |
| Insititutionbl | 负责实现对应机构管理的服务 |
| Commoditybl | 负责实现对应库存查看、库存分区、库存盘点和库存报警的服务 |
| Logbl | 负责实现对应日志操作的服务 |
| Sheetbl | 负责实现对应所有单据操作的服务 |
| Begininfobl | 负责实现对应初始化信息的服务 |
| Accountbl | 负责实现对应账户管理的服务 |
| Staffmanagebl | 负责实现对应员工信息管理的服务 |
| LogInbl | 负责实现对应登陆界面 |

# 客户端业务逻辑层模块局部模块的接口规范

# 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13

（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接口。业务逻辑层和数据层之间添加dataservice.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。UserPO是作为个人帐号信息的持久化对象被添加到设计模型中去的。

**表15 个人帐号管理业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：快递物流系统用例文档V3.3 “用例三十一 账号信息管理”  快递物流系统需求规格说明文档V1.6中的3.2及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface UserBLService {    public UserVO getPersonalInfo(long UserId);  public boolean changePassword(String password,long UserId);  public boolean initial(ArrayList<InitUserVO> User);  public boolean endManagement();  }  //调用DataService层的接口  public interface UserDataService extends Remote{  public void insert(ArrayList<UserPO> User) throws RemoteException;  public UserPO find(long UserId) throws RemoteException;  public void update(UserPO po) throws RemoteException;  public void finish() throws RemoteException;  public void InitialInsert(ArrayList<UserPO> User) throws RemoteException ;  public void delete(UserPO po) throws RemoteException;  public ArrayList<UserPO> finds() throws RemoteException;  } |
| 输出 | 类图 |  |

（3）模块内部类的接口规范

**表16 UserController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| UserController. getPersonalInfo | 语法 | Public StaffInfoVO getPersonalInfo(long UserId) |
| 前置条件 | 个人帐号拥有者启动账号信息管理 |
| 后置条件 | 在一次个人账号信息管理中获得个人帐号信息 |
| UserController. changePassword | 语法 | Public boolean changePassword(String password,long userId) |
| 前置条件 | 个人帐号拥有者启动账号信息管理 |
| 后置条件 | 员工在一次帐号信息管理中更改密码 |
| UserController. endManagement | 语法 | Public boolean endManagement() |
| 前置条件 | 已确认所有个人信息的管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| UserController. initial | 语法 | Public boolean initial(ArrayList<InitUserVO> User) |
| 前置条件 | 期初建账 |
| 后置条件 | 在期初建账中初始化信息 |
| UserController.changeInfo | 语法 | Public boolean changeInfo(UserVO userVO) |
| 前置条件 | 总经理修改员工 |
| 后置条件 | 更新员工个人信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| User. getPersonalInfo(long UserId) | 根据ID查找单一个人信息 | |
| User. changePassword(String password,long userId) | 更改密码 | |
| User. initial(ArrayList<InitUserVO> User) | 期初建账时初始化 | |
| User. changeInfo(UserVO userVO) | 更改信息 | |
| User. endManagement() | 结束管理 | |

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| User. getPersonalInfo | 语法 | Public StaffInfoVO getPersonalInfo(long UserId) |
| 前置条件 | 个人帐号拥有者启动账号信息管理 |
| 后置条件 | 在一次个人账号信息管理中获得个人帐号信息 |
| User. changePassword | 语法 | Public boolean changePassword(String password,long userId) |
| 前置条件 | 个人帐号拥有者启动账号信息管理 |
| 后置条件 | 员工在一次帐号信息管理中更改密码 |
| User. endManagement | 语法 | Public boolean endManagement() |
| 前置条件 | 已确认所有个人信息的管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| User. initial | 语法 | Public boolean initial(ArrayList<InitUserVO> User) |
| 前置条件 | 期初建账 |
| 后置条件 | 在期初建账中初始化信息 |
| User.changeInfo | 语法 | Public boolean changeInfo(UserVO userVO) |
| 前置条件 | 总经理修改员工 |
| 后置条件 | 更新员工个人信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| UserDataService.find (long UserId) | 根据ID进行查找单一持久化对象 | |
| UserDataService. update (String password) | 更新单一持久化对象 | |
| UserDataService. InitialInsert(ArrayList<UserPO> User) | 期初建账时初始化 | |
| UserDataService. Insert(UserPO po) | 增加单一持久化对象 | |
| UserDataService. delete(UserPO po) | 删除单一持久化对象 | |
| UserDataService.finds() | 获得所有持久化对象 | |
| UserDataService.finish() | 结束操作 | |

User的接口规范

# nonuserbl模块

（1）模块概述

NonUser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。

NonUser模块的职责及接口参见软件体系结构描述文档表16。

（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bussinesslogicservice. nonuserblservice. NonUserBLService接口。业务逻辑层和数据层之间添加dataservice. nonuserdataservice. NonUserService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了NonUserComtroller，将对库存的业务逻辑委托给NonUser对象。GoodsPO是作为商品信息的持久化对象被添加到设计模型中去的。

**表17 非用户信息业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：软件用例描述文档中的用例7至用例8  软件需求规格文档中的3.2节及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface NonUserBLService {    /\*\*  \* 非用户信息查询界面得到某个营业厅信息  \* @param bussinessHallId  \* @return  \*/  public BussinessHallVO getBussinessHallInfo(String bussinessHallId);    /\*\*  \* 非用户信息查询界面得到所有营业厅信息  \* @return  \*/  public ArrayList<BussinessHallVO> getAllBussinessHallInfo();    /\*\*  \* 增加一个营业厅  \* @param bussinessHall  \*/  public void addBussinessHallInfo(BussinessHallVO bussinessHall);    /\*\*  \* 删除一个营业厅  \* @param bussinessHallId  \*/  public void deleteBussinessHallInfo(String bussinessHallId);  /\*\*  \* 非用户信息查询界面得到某个司机信息  \* @param driverId  \*/  public DriverInfoVO getDriverInfo(String driverId);    /\*\*  \* 非用户信息查询界面得到某营业厅所有司机信息  \* @param id  \* @return  \*/  public ArrayList<DriverInfoVO> getDriverByBusinesshall(String id);    /\*\*  \* 非用户信息查询界面得到所有司机信息  \* @return  \*/  public ArrayList<DriverInfoVO> getAllDriverInfo();    /\*\*  \* 增加一个司机  \* @param driver  \*/  public void addDriverInfo(DriverInfoVO driver);    /\*\*  \* 删除一个司机  \* @param driverId  \*/  public void deleteDriverInfo(String driverId);    /\*\*  \* 修改司机信息  \* @param driverId  \* @param dvo  \*/  public void changeDriverInfo(String driverId,DriverInfoVO dvo);    /\*\*  \* 供接口，提供司机信息初始化  \* @param dvolist  \*/  public void initDriver(ArrayList<DriverInfoVO> dvolist);  /\*\*  \* 非用户信息查询界面得到某个车辆信息  \* @param truckId  \*/  public TruckInfoVO getTruckInfo(String truckId);    /\*\*  \* 非用户信息查询界面得到某营业厅所有车辆信息  \* @param id  \* @return  \*/  public ArrayList<TruckInfoVO> getTruckByBusinesshall(String id);    /\*\*  \* 非用户信息查询界面得到所有车辆信息  \* @return  \*/  public ArrayList<TruckInfoVO> getAllTruckInfo();    /\*\*  \* 增加一个车辆  \* @param truck  \*/  public void addTruckInfo(TruckInfoVO truck);    /\*\*  \* 删除一个车辆  \* @param truckId  \*/  public void deleteTruckInfo(String truckId);    /\*\*  \* 修改车辆信息  \* @param truckId  \* @param tvo  \*/  public void changeTruckInfo(String truckId,TruckInfoVO tvo);    /\*\*  \* 供接口，提供车辆信息初始化  \* @param tvo  \*/  public void initTruck(ArrayList<TruckInfoVO> tvolist);  /\*\*  \* 非用户信息查询界面得到某个中转中心信息  \* @param id  \* @return  \*/  public IntermediateCenterVO getIntermediateCenterInfo(String id);    /\*\*  \* 非用户信息查询界面得到所有中转中心信息  \* @return  \*/  public ArrayList<IntermediateCenterVO> getAllIntermediateCenterInfo();    /\*\*  \* 增加一个中转中心  \* @param intermediateCenter  \*/  public void addIntermediateCenter(IntermediateCenterVO intermediateCenter);    /\*\*  \* 删除一个中转中心  \* @param id  \*/  public void deleteIntermediateCenter(String id);    /\*\*  \* 供接口，返回中转中心所在城市  \* @return  \*/  public ArrayList<String> getcity();  }//调用DataService层的接口  public interface NonUserDataService extends Remote{    /\*\*  \* 根据id返回BussinessHallPO  \* @param id  \* @return  \* @throws RemoteException  \*/  public BussinessHallPO findBussinessHallPO(String id) throws RemoteException;    /\*\*  \* 返回一个城市的所有营业厅  \* @param city  \* @return  \* @throws RemoteException  \*/  public ArrayList<BussinessHallPO> findBussinessHallPOByCity(String city) throws RemoteException;    /\*\*  \* 返回所有BussinessHallPO  \* @return  \* @throws RemoteException  \*/  public ArrayList<BussinessHallPO> findsBussinessHallPO() throws RemoteException;    /\*\*  \* 在数据库中增加一个BussinessHallPO记录  \* @param tpo  \* @throws RemoteException  \*/  public void insert(BussinessHallPO bpo) throws RemoteException;    /\*\*  \* 更新一个BussinessHallPO  \* @param id  \* @param bpo  \* @throws RemoteException  \*/  public void update(String id,BussinessHallPO bpo) throws RemoteException;    /\*\*  \* 删除一个BussinessHallPO  \* @param id  \* @throws RemoteException  \*/  public void deleteBussinessHallPO(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  /\*\*  \* 根据id返回DriverPO  \* @param id  \* @return  \* @throws RemoteException  \*/  public DriverPO findDriverPO(String id) throws RemoteException;    /\*\*  \* 根据营业厅编号返回其所属司机  \* @param id  \* @return  \* @throws RemoteException  \*/  public ArrayList<DriverPO> findsDriverPO(String id) throws RemoteException;    /\*\*  \* 返回所有DriverPO  \* @return  \* @throws RemoteException  \*/  public ArrayList<DriverPO> findsDriverPO() throws RemoteException;    /\*\*  \* 在数据库中增加一个DriverPO记录  \* @param dpo  \* @throws RemoteException  \*/  public void insert(DriverPO dpo) throws RemoteException;    /\*\*  \* 更新一个DriverPO  \* @param id  \* @param dpo  \* @throws RemoteException  \*/  public void update(String id,DriverPO dpo) throws RemoteException;    /\*\*  \* 删除一个DriverPO  \* @param id  \* @throws RemoteException  \*/  public void deleteDriverPO(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  /\*\*  \* 车辆数据操作的接口  \* @author Potter  \*  \*/  public interface TruckDataService extends Remote{  /\*\*  \* 根据id返回TruckPO  \* @param id  \* @return  \* @throws RemoteException  \*/  public TruckPO findTruckPO(String id) throws RemoteException;    /\*\*  \* 根据营业厅编号返回其所属车辆  \* @param id  \* @return  \* @throws RemoteException  \*/  public ArrayList<TruckPO> findsTruckPO(String id) throws RemoteException;    /\*\*  \* 返回所有TruckPO  \* @return  \* @throws RemoteException  \*/  public ArrayList<TruckPO> findsTruckPO() throws RemoteException;    /\*\*  \* 在数据库中增加一个TruckPO记录  \* @param tpo  \* @throws RemoteException  \*/  public void insert(TruckPO tpo) throws RemoteException;    /\*\*  \* 更新一个TruckPO  \* @param id  \* @param tpo  \* @throws RemoteException  \*/  public void update(String id,TruckPO tpo) throws RemoteException;    /\*\*  \* 删除一个TruckPO  \* @param id  \* @throws RemoteException  \*/  public void deleteTruckPO(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  /\*\*  \* 根据id返回IntermediateCenterPO  \* @param id  \* @return  \* @throws RemoteException  \*/  public IntermediateCenterPO findIntermediateCenterPO(String id) throws RemoteException;    /\*\*  \* 返回所有IntermediateCenterPO  \* @return  \* @throws RemoteException  \*/  public ArrayList<IntermediateCenterPO> findsIntermediateCenterPO() throws RemoteException;    /\*\*  \* 在数据库中增加一个IntermediateCenterPO记录  \* @param tpo  \* @throws RemoteException  \*/  public void insert(IntermediateCenterPO ipo) throws RemoteException;    /\*\*  \* 更新一个IntermediateCenterPO  \* @param id  \* @param ipo  \* @throws RemoteException  \*/  public void update(String id,IntermediateCenterPO ipo) throws RemoteException;    /\*\*  \* 删除一个IntermediateCenterPO  \* @param id  \* @throws RemoteException  \*/  public void deleteIntermediateCenterPO(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  } |
| 输出 | 类图 | 非用户信息详细设计类图 |

（3）模块内部类的接口规范

**表18 NonUserController模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| NonUser**Controller**. getTruckInfo | 语法 | public TruckVO findTruckVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getTruckInfo方法 |
| NonUser**Controller**. getTruckInfoByBusinessHall | 语法 | public TruckVO findTruckVOByBusinessHall(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getTruckInfo方法 |
| NonUser**Controller**. getAllTruckInfo | 语法 | public DriverVO findDriverVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getAllTruckInfo方法 |
| NonUser**Controller**. addTruckInfo | 语法 | public BussinessHallVO findBussinessHallVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的addTruckInfo方法 |
| NonUser**Controller**. deleteTruckInfo | 语法 | public ArrayList<TruckVO> findsTruckVO() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的deleteTruckInfo方法 |
| NonUser**Controller**. changeTruckInfo | 语法 | public ArrayList<DriverVO> findsDriverVO() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的changeTruckInfo方法 |
| NonUser**Controller**.getDriverInfo | 语法 | Public ArrayList<BussinessHallVO> findsBussinessHallVO() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的getDriverInfo方法 |
| NonUser**Controller**.getDriverInfoByBusinessHall | 语法 | Public ArrayList<BussinessHallVO> findsBussinessHallVOBusinessHall() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的getDriverInfo方法 |
| NonUser**Controller**. getAllDriverInfo | 语法 | public void insert(BussinessHallVO bVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getAllDriverInfo方法 |
| NonUser**Controller**.addDriverInfo | 语法 | public void insert(DriverVO dVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的addDriverInfo方法 |
| NonUser**Controller**. deleteDriverInfo | 语法 | public void insert(TruckVO tVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的addDriverInfo方法 |
| NonUser**Controller**. changeDriverInfo | 语法 | public void update(TruckVO tVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的changeDriverInfo方法 |
| NonUser**Controller**. getBussinessHallInfo | 语法 | public void getBussinessHallInfo(DriverVO dVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getBussinessHallInfo方法 |
| NonUser**Controller**. getBussinessHallInfoByCity | 语法 | public void getBussinessHallInfoByCity(DriverVO dVO) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getBussinessHallInfo方法 |
| NonUser**Controller**. getAllBussinessHallInfo | 语法 | public void deleteTruckVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的getAllBussinessHallInfo方法 |
| NonUser**Controller**. addBussinessHallInfo | 语法 | public void deleteDriverVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的addBussinessHallInfo方法 |
| NonUser**Controller**.deleteBussinessHallInfo | 语法 | public void deleteBussinessHallVO(long id) throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象，并且输入符合输入规则 |
| 后置条件 | 调用nonuser领域对象的deleteBussinessHallVO方法 |
| NonUser**Controller**.changeLogisticsState | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的changeLogisticsState方法 |
| NonUser**Controller**.initTruck | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的initTruck方法 |
| NonUser**Controller**.initDriver | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 已创建一个nonuser领域对象 |
| 后置条件 | 调用nonuser领域对象的initDriver方法 |
| 服务名 | 服务 | |
| NonUserDataService.find(long id) | 根据ID进行查找单一持久化对象 | |
| NonUserDataService.finds(GoodsType type) | 根据字段名和值进行查找多个持久化对象 | |
| NonUserDataService.insert(GoodsVO gVO) | 插入单一持久化对象 | |
| NonUserDataService.update(GoodsVO gVO) | 更新单一持久化对象 | |

**表19 NonUserbl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| NonUser. getTruckInfo | 语法 | public TruckInfoVO getTruckInfo(long truckId); |
| 前置条件 | 启动一个得到车辆信息的回合 |
| 后置条件 | 在一个得到车辆信息的回合中，返回车辆信息 |
| NonUser. getAllTruckInfo | 语法 | public ArrayList<TruckInfoVO> getAllTruckInfo(); |
| 前置条件 | 启动一个得到所有车辆信息的回合 |
| 后置条件 | 在一个得到所有车辆信息的回合中，返回所有车辆信息 |
| NonUser. getAllTruckInfoByBusinessHall | 语法 | public ArrayList<TruckInfoVO> getAllTruckInfoByBusinessHall(); |
| 前置条件 | 启动一个得到营业厅所属车辆信息的回合 |
| 后置条件 | 在一个得到营业厅所属车辆信息的回合中，返回所有车辆信息 |
| NonUser. addTruckInfo | 语法 | public void addTruckInfo(TruckInfoVO truck); |
| 前置条件 | 启动一个增加车辆信息的回合 |
| 后置条件 | 在一个增加车辆信息的回合中，增加车辆信息 |
| NonUser. deleteTruckInfo | 语法 | public void deleteTruckInfo(long truckId); |
| 前置条件 | 启动一个得到所有删除车辆的回合 |
| 后置条件 | 在一个得到所有删除车辆的回合中，删除车辆信息 |
| NonUser. changeTruckInfo | 语法 | public void changeTruckInfo(long truckId); |
| 前置条件 | 启动一个修改车辆信息的回合 |
| 后置条件 | 在一个修改车辆信息的回合中，修改车辆信息 |
| NonUser.getDriverInfo | 语法 | public DriverInfoVO getDriverInfo(long driverId); |
| 前置条件 | 启动一个得到司机信息的回合 |
| 后置条件 | 在一个得到司机信息的回合中，返回司机信息 |
| NonUser.getDriverInfoByBusinessHall | 语法 | public DriverInfoVO getDriverInfoByBusinessHall(long driverId); |
| 前置条件 | 启动一个得到营业厅所属司机信息的回合 |
| 后置条件 | 在一个得到营业厅所属司机信息的回合中，返回司机信息 |
| NonUser. getAllDriverInfo | 语法 | public ArrayList<DriverInfoVO> getAllDriverInfo(); |
| 前置条件 | 启动一个插入得到所有司机信息的回合 |
| 后置条件 | 在一个得到所有司机信息的回合中，得到所有司机信息 |
| NonUser.addDriverInfo | 语法 | public void addDriverInfo(DriverInfoVO driver); |
| 前置条件 | 启动一个增加司机的回合 |
| 后置条件 | 在一个增加司机的回合中增加一个司机 |
| NonUser. deleteDriverInfo | 语法 | public void deleteDriverInfo(long driverId); |
| 前置条件 | 启动一个删除司机的回合 |
| 后置条件 | 在一个删除司机的回合中，删除一个司机 |
| NonUser. changeDriverInfo | 语法 | public void changeDriverInfo(long driverId); |
| 前置条件 | 启动一个修改司机信息的回合 |
| 后置条件 | 在一个修改司机信息的回合中，修改司机信息 |
| NonUser. getBussinessHallInfo | 语法 | public BussinessHallVO getBussinessHallInfo(long bussinessHallId); |
| 前置条件 | 启动一个得到营业厅信息的回合 |
| 后置条件 | 在一个得到营业厅信息的回合中，得到营业厅信息 |
| NonUser. getBussinessHallInfoByCity | 语法 | public BussinessHallVO getBussinessHallInfoByCity(long bussinessHallId); |
| 前置条件 | 启动一个得到城市营业厅信息的回合 |
| 后置条件 | 在一个得到城市营业厅信息的回合中，得到营业厅信息 |
| NonUser. getAllBussinessHallInfo | 语法 | public ArrayList<BussinessHallVO> getAllBussinessHallInfo(); |
| 前置条件 | 启动一个得到所有营业厅信息的回合 |
| 后置条件 | 在一个得到所有营业厅信息的回合中，得到所有营业厅信息 |
| NonUser. addBussinessHallInfo | 语法 | public void addBussinessHallInfo(BussinessHallVO bussinessHall); |
| 前置条件 | 启动一个增加营业厅的回合 |
| 后置条件 | 在一个增加营业厅的回合中，增加一个营业厅 |
| NonUser.deleteBussinessHallInfo | 语法 | public void deleteBussinessHallInfo(long bussinessHallId); |
| 前置条件 | 启动一个删除营业厅信息的回合 |
| 后置条件 | 在一个删除营业厅信息的回合中，删除一个营业厅信息 |
| NonUser.changeLogisticsState | 语法 | public void changeLogisticsState(String logisticsState); |
| 前置条件 | 启动一个修改物流状态的回合 |
| 后置条件 | 在一个修改物流状态的回合中，修改物流状态 |
| NonUser.initTruck | 语法 | public void initTruck(ArrayList<TruckInfoVO> tvolist); |
| 前置条件 | 启动一个初始化车辆信息的回合 |
| 后置条件 | 在一个初始化车辆信息的回合中，初始化车辆信息 |
| NonUser.initDriver | 语法 | public void initDriver(ArrayList<DriverInfoVO> dvolist); |
| 前置条件 | 启动一个初始化司机信息的回合 |
| 后置条件 | 在一个初始化司机信息的回合中，初始化司机信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| NonUserDataService.find(long id) | 根据ID进行查找单一持久化对象 | |
| NonUserDataService.finds(GoodsType type) | 根据字段名和值进行查找多个持久化对象 | |
| NonUserDataService.insert(GoodsPO gpo) | 插入单一持久化对象 | |
| NonUserDataService.update(GoodsPO gpo) | 更新单一持久化对象 | |

# insititutionbl模块

1. 模块概述

Institutionbl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

Institutionbl模块的职责及接口参见软件系统结构描述文档表12

1. 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.institutionblservice接口。业务逻辑层和数据层之间添加dataservice.institutiondataservice接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了institutionController。UserPO和InstitutionPO是作为人员机构管理的持久化对象被添加到设计模型中去的。

**表20 人员机构管理业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：快递物流系统用例文档V3.3 “用例二十六 机构管理”和“用例二十八 人员工资调整”  快递物流系统需求规格说明文档V1.6中的3.2及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface InstitutionBLService {  public InstitutionVO getInstitutionInfo(long InstitutionId);  public ArrayList<InstitutionVO> getAllInstitution();  public boolean addInstitution(InstitutionVO Institution);  public boolean deleteInstitution(long InstitutionId);  public boolean endManagement();  public boolean changeInstitutionInfo(InstitutionVO institutionVO);  }  //调用DataService层的接口  public interface InstitutionDataService extends Remote{  public InstitutionPO find(long InstitutionId) throws RemoteException;  public ArrayList<InstitutionPO> finds() throws RemoteException;  public void insert(InstitutionPO po) throws RemoteException;  public void update(InstitutionPO po) throws RemoteException;  public void delete(InstitutionPO po) throws RemoteException;  public void finish() throws RemoteException;  } |
| 输出 | 类图 |  |

（3）模块内部类的接口规范

**InstitutionController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| InstitutionController.getInstitutionInfo | 语法 | Public InstitutionVO getInstitutionInfo (long InstitutionId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中获得机构信息 |
| InstitutionController.addInstitution | 语法 | Public boolean addInstitution (InstitutionVO Institution) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中增加机构 |
| InstitutionController. deleteInstitution | 语法 | Public boolean deleteInstitution(long InstitutionId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中删除机构 |
| InstitutionController.endManagement | 语法 | Public boolean endManagement() |
| 前置条件 | 已确认所有管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| InstitutionController.getAllInstitution | 语法 | Public ArrayList<InstitutionVO> getAllInstitution() |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中获得所有机构信息 |
| InstitutionController. changeInstitutionInfo | 语法 | Public boolean changeInstitutionInfo(InstitutionVO institutionVO) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中更新机构信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Institution. getInstitutionInfo (long InstitutionId) | 根据ID进行查找单一对象 | |
| Institution. addInstitution (InstitutionVO Institution) | 增加机构 | |
| Institution.deleteInstitution(long InstitutionId) | 删除机构 | |
| Institution. getAllInstitution() | 获得所有机构 | |
| Institution. changeInstitutionInfo(InstitutionVO institutionVO) | 更新机构信息 | |
| Institution. endManagement() | 结束操作 | |

**表21 institution的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Institution.getInstitutionInfo | 语法 | Public InstitutionVO getInstitutionInfo (long InstitutionId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中获得机构信息 |
| Institution.addInstitution | 语法 | Public boolean addInstitution (InstitutionVO Institution) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中增加机构 |
| Institution. deleteInstitution | 语法 | Public boolean deleteInstitution(long InstitutionId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中删除机构 |
| Institution.endManagement | 语法 | Public boolean endManagement() |
| 前置条件 | 已确认所有管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| Institution.getAllInstitution | 语法 | Public ArrayList<InstitutionVO> getAllInstitution() |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中获得所有机构信息 |
| Institution. changeInstitutionInfo | 语法 | Public boolean changeInstitutionInfo(InstitutionVO institutionVO) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中更新机构信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| InstitutionDataService. find (long InstitutionId) | 根据ID进行查找单一持久化对象 | |
| InstitutionDataService. insert (InstitutionPO po) | 插入单一持久化对象 | |
| InstitutionDataService. delete (InstitutionPO po) | 删除单一持久化对象 | |
| InstitutionDataService. update (InstitutionPO po) | 更新单一持久化对象 | |
| InstitutionDataService. finds() | 查找所有持久化对象 | |
| InstitutionDataService.finish() | 结束操作 | |

# commoditybl模块

（1）模块概述

Commodity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。

Commodity模块的职责及接口参见软件体系结构描述文档表15。

（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bussinesslogicservice.commodityblservice.CommodityBLService接口。业务逻辑层和数据层之间添加dataservice.salesdataservice.SalesDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommodityComtroller，将对库存的业务逻辑委托给commodity对象。GoodsPO是作为商品信息的持久化对象被添加到设计模型中去的；StoragePO则是作为库存信息的持久化对象被添加到设计模型中去的。

**表22 库存管理业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：软件用例描述文档中的用例13至用例18  软件需求规格文档中的3.2节及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface CommodityBLService {  /\*\*  \* 盘点界面得到当天各区快递信息  \*/  public ArrayList<ExpressInfoVO> stockTaking(String storageId);    /\*\*  \* 向外部导出Excel  \*/  public void exportExcel(String realPath,ArrayList<ExpressInfoVO> users);    /\*\*  \* 修改警戒比例  \* @param alarmScale  \*/  public void changeAlarmScale(double alarmScale,String storageId);    /\*\*  \* 获得当前警戒比例  \* @param storageId  \* @return  \*/  public double getAlarmScale(String storageId);    /\*\*  \* 库存查看界面得到一段时间内的库存信息  \* @return  \*/  public ArrayList <StorageInfoVO> checkStorageMessage(String storageId,long startTime,long endTime);  public StorageNumVO getStorageNum(String storageId,long startTime,long endTime);    /\*\*  \* 商品数量超出报警值，实现库存报警  \* @return  \*/  public AlarmMessage alarm (String storageId);    /\*\*  \* 得到各区库存货物信息  \* @param areaNumber  \* @return  \*/  public ArrayList<ExpressInfoVO> getAreaGoodsPOs(String storageId,GoodsType areaNumber);    /\*\*  \* 对货物进行库存分区，调整各区的库存分配  \* @param evos  \*/  public void divide (String storageId,ArrayList<ExpressInfoVO> evos);    /\*\*  \* 结束库存操作  \*/  public void endCommodityManagement();    /\*\*  \* 供接口，提供修改库存信息  \*/  public void changeStorageInInfo(String storageId,SheetVO svo);    /\*\*  \* 供接口，提供库存信息初始化  \*/  public void initStorageInfo(ArrayList<StorageInitVO> svolist);    }  //调用DataService层的接口  public interface CommodityDataService {  /\*\*  \* 根据id返回GoodsPO  \* @param id  \* @return  \* @throws RemoteException  \*/  public GoodsPO findGoodsPO(String id) throws RemoteException;    /\*\*  \* 根据type返回一组GoodsPO  \* @param type  \* @return  \* @throws RemoteException  \*/  public ArrayList<GoodsPO> finds(GoodsType type) throws RemoteException;    /\*\*  \* 返回所有GoodsPO  \* @return  \* @throws RemoteException  \*/  public ArrayList<GoodsPO> finds() throws RemoteException;    /\*\*  \* 在数据库中增加一个GoodsPO记录  \* @param gpo  \* @throws RemoteException  \*/  public void insert(GoodsPO gpo) throws RemoteException;    /\*\*  \* 更新一个GoodsPO  \* @param gpo  \* @throws RemoteException  \*/  public void update(GoodsPO gpo) throws RemoteException;    /\*\*  \* 删除一个GoodsPO  \* @param id  \* @throws RemoteException  \*/  public void delete(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  /\*\*  \* 根据id返回StoragePO  \* @param id  \* @return  \* @throws RemoteException  \*/  public StoragePO findStoragePO(String id) throws RemoteException;    /\*\*  \* 返回所有StoragePO  \* @return  \* @throws RemoteException  \*/  public ArrayList<StoragePO> finds() throws RemoteException;    /\*\*  \* 在数据库中增加一个StoragePO记录  \* @param spo  \* @throws RemoteException  \*/  public void insert(StoragePO spo) throws RemoteException;    /\*\*  \* 更新一个StoragePO  \* @param id  \* @param spo  \* @throws RemoteException  \*/  public void update(String id,StoragePO spo) throws RemoteException;    /\*\*  \* 删除一个StoragePO  \* @param id  \* @throws RemoteException  \*/  public void delete(String id) throws RemoteException;    /\*\*  \* 结束持久化数据库的使用  \* @throws RemoteException  \*/  public void finish() throws RemoteException;  } |
| 输出 | 类图 | 库存详细设计类图 |

（3）模块内部类的接口规范

**表23 commodityController模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| CommodityController.stockTaking | 语法 | public ArrayList<ExpressInfoVO> stockTaking() |
| 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stockTaking方法 |
| CommodityController.exportExcel | 语法 | public void exportExcel () |
| 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exportExcel方法 |
| CommodityController.changeAlarmScale | 语法 | public void changeAlarmScale (double alarmScale) |
| 前置条件 | 已创建一个commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用commodity领域对象的changeAlarmScale方法 |
| CommodityController.getAlarmScale | 语法 | public void getAlarmScale (double alarmScale) |
| 前置条件 | 已创建一个commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用commodity领域对象的getAlarmScale方法 |
| CommodityController.checkStorageMessage | 语法 | Public ArrayList<StorageInfoVO> checkStorageMessage () |
| 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checkStorageMessage方法 |
| CommodityController. getStorageNum | 语法 | public StoragePO getStorageNum(long startTime,long endTime) |
| 前置条件 | 已创建一个commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用commodity领域对象的getStorageNum方法 |
| CommodityController.alarm | 语法 | public void alarm () |
| 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alarm方法 |
| CommodityController.divide | 语法 | public void divide (GoodsPO gpo，StoragePO spo) |
| 前置条件 | 已创建一个commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用commodity领域对象的divide方法 |
| CommodityController.endCommodityManagement | 语法 | public void endCommodityManagement() |
| 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endCommodityManagement方法 |
| CommodityController.changeStorageInInfo | 语法 | public void changeStorageInInfo(ArrayList<SheetVO> svolist); |
| 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changeStorageInInfo方法 |
| CommodityController. initStorageInfo | 语法 | public void initStorageInfo(StorageInitVO sivo); |
| 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initStorageInfo方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Commodity. getStorageId | 在一个得到仓库id的回合中，得到仓库id | |
| ………………. |  | |

**表24 commoditybl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Commodity.stockTaking | 语法 | public ArrayList<ExpressInfoVO> stockTaking() |
| 前置条件 | 启动一个库存盘点的回合 |
| 后置条件 | 在一个库存盘点回合中，显示货物信息，完成对相应库存信息的盘点 |
| Commodity.exportExcel | 语法 | public void exportExcel () |
| 前置条件 | 启动一个导出Excel的回合 |
| 后置条件 | 导出库存信息的Excel表格 |
| Commodity.changeAlarmScale | 语法 | public void changeAlarmScale (double alarmScale) |
| 前置条件 | 启动一个修改警戒比例的回合 |
| 后置条件 | 在一个警戒比例修改回合中完成对警戒比例的修改 |
| Commodity.getAlarmScale | 语法 | public void getAlarmScale (double alarmScale) |
| 前置条件 | 启动一个获得警戒比例的回合 |
| 后置条件 | 在一个警戒比例获得回合中完成对警戒比例的获得 |
| Commodity.checkStorageMessage | 语法 | Public ArrayList<StorageInfoVO> checkStorageMessage () |
| 前置条件 | 启动一个查看库存信息的回合 |
| 后置条件 | 在一个查看库存信息回合中显示一段时间内出入库的货物的存储位置 |
| Commodity. getStorageNum | 语法 | public StoragePO getStorageNum(long startTime,long endTime) |
| 前置条件 | 启动一个查看库存信息的回合 |
| 后置条件 | 在一个查看库存信息回合中显示一段时间内的出/入库数量/金额，合计的库存数量 |
| Commodity.alarm | 语法 | public void alarm () |
| 前置条件 | 货物数量发生变化 |
| 后置条件 | 若货物数量超出报警值则报警 |
| Commodity.divide | 语法 | public void divide (GoodsPO gpo，StoragePO spo) |
| 前置条件 | 启动一个库存分区的回合 |
| 后置条件 | 在一个库存分区回合中完成对货物的库存分区 |
| Commodity.endCommodityManagement | 语法 | public void endCommodityManagement() |
| 前置条件 | 已完成库存操作 |
| 后置条件 | 结束此次库存操作，持久化更新涉及的领域的对象的数据 |
| Commodity.changeStorageInInfo | 语法 | public void changeStorageInInfo(ArrayList<SheetVO> svolist); |
| 前置条件 | 启动一个修改库存信息的回合 |
| 后置条件 | 在一个修改库存信息回合中完成库存信息的修改 |
| Commodity. initStorageInfo | 语法 | public void initStorageInfo(StorageInitVO sivo); |
| 前置条件 | 启动一个初始化库存信息的回合 |
| 后置条件 | 在一个初始化库存信息回合中完成库存信息的修改 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| CommodityDataService.find(long id) | 根据ID进行查找单一持久化对象 | |
| CommodityDataService.finds(GoodsType type) | 根据字段名和值进行查找多个持久化对象 | |
| CommodityDataService.insert(GoodsPO gpo) | 插入单一持久化对象 | |
| CommodityDataService.update(GoodsPO gpo) | 更新单一持久化对象 | |

# logbl模块

(1)模块概述

logbl模块负责实现日志相关的功能需求，具体功能需求和非功能需求请参见需求规格说明文档和体系结构设计文档。

(2)整体结构

logbl模块实现了展示层与业务逻辑层之间的logblservice接口和业务逻辑层与数据层之间的logdataservice接口。

**表25日志业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：  用例文档中的UC2-UC22,UC25-UC31  软件需求规格说明文档中的3.2节及3.3节中有关UC2-UC22,UC25-UC31部分 |
| 体系结构 | // 被Presentation层调用的接口  public interface LogBLService {  public ArrayList<String> checkLog(String date);  public void generateLog(String position, String name, String operation);  }  //调用DataService层的接口  public interface LogDataService {  public LogPO find(long id) throws RemoteException;  public ArrayList<LogPO> findAll() throws RemoteException;  public void insert(LogPO po) throws RemoteException;  public void finish() throws RemoteException;  } |
| 输出 | 类图 | log |

(3)模块内部类的接口规范

**表26 LogController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| LogController.checkLog | 语法 | public void checkLog(String date) | |
| 前置条件 | 已创建一个Log领域对象 | |
| 后置条件 | 调用Log领域对象的checkLog方法 | |
| LogController.generateLog | 语法 | public void generateLog(String position, String name, String operation) | |
| 前置条件 | 已创建一个Log领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用Log领域对象的generateLog方法 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| Log.checkLog(String date) | | | 查看操作日志 |
| Log.generateLog(String position, String name, String operation) | | | 生成操作日志 |

**表27 Log的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| Log.checkLog | 语法 | public ArrayList<String> checkLog(String date) | |
| 前置条件 | 启动一个查看操作日志的回合 | |
| 后置条件 | 如果date为空，查看所有操作日志，否则返回所选日期的操作日志，之后结束此次查看操作日志回合 | |
| Log.generateLog | 语法 | public void generateLog(String position, String name, String operation) | |
| 前置条件 | 操作已完成 | |
| 后置条件 | 生成操作日志，持久化更新涉及的领域对象的数据 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| DatabaseFactory.getLogDatabase | | | 得到Log数据库的服务的引用 |
| LogDataService.insert(LogPO po) | | | 插入单一持久化对象 |
| LogDataService.find(String date) | | | 根据日期进行查找多个持久化对象 |
| LogDataService.findAll() | | | 进行查找所有持久化对象 |

# sheetbl模块

(1)模块概述

sheetbl模块负责实现与表格相关的功能需求，具体功能需求和非功能需求请参见需求规格说明文档和体系结构设计文档。

(2)整体结构

sheetbl模块实现了展示层与业务逻辑层之间的sheetblservice接口和业务逻辑层与数据层之间的sheetdataservice接口。

|  |  |  |
| --- | --- | --- |
|  | 体系结构 | // 被Presentation层调用的接口  public interface SheetBLService {  public String[][] getExistedInfo();  public boolean add(SheetVO vo);  public boolean modify(long ID, SheetVO vo);  public SheetVO find(long ID);  public ArrayList<SheetVO> findVOs(FindingType findingType);  public boolean examineSheet(long ID, SheetState state);  }  //调用DataService层的接口  public interface SheetDataService {  public SheetPO find(long id) throws RemoteException;  public ArrayList<SheetPO> finds(SheetType type) throws RemoteException;  public void insert(SheetPO po) throws RemoteException;  public void update(SheetPO po) throws RemoteException;  public void finish() throws RemoteException;  } |
| 输出 | 类图 | sheetV2 |

**表28 表格业务逻辑层详细设计的上下文**

(3)模块内部类的接口规范

**表29 SheetController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| Sheet.getExistedInfo | 语法 | public String[][] getExistedInfo() | |
| 前置条件 | 已创建一个Sheet领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用Sheet领域对象的checkSheet方法 | |
| Sheet.add | 语法 | public boolean add(SheetVO vo) | |
| 前置条件 | 已创建一个Sheet领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用Sheet领域对象的examineSheet方法 | |
| Sheet.modify | 语法 | public boolean modify(long ID,SheetVO vo) | |
| 前置条件 | 已创建一个Sheet领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用Sheet领域对象的modify方法 | |
| Sheet.find | 语法 | public SheetVO find(long ID) | |
| 前置条件 | 已创建一个Sheet领域对象 | |
| 后置条件 | 调用Sheet领域对象的find方法 | |
| Sheet.findVOs | 语法 | public ArrayList<SheetVO> findVOs(FindingType type) | |
| 前置条件 | 已创建一个Sheet领域对象 | |
| 后置条件 | 调用Sheet领域对象的findVOs方法 | |
| Sheet.examineSheet | 语法 | public boolean examineSheet(long ID,SheetState state) | |
| 前置条件 | 已创建一个Sheet领域对象 | |
| 后置条件 | 调用Sheet领域对象的examineSheet方法 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| Sheet.getExistedInfo | | | 获得表格上需要显示的信息 |
| Sheet.add | | | 新建表格 |
| Sheet.modify | | | 修改表格 |
| Sheet.find | | | 查找表格 |
| Sheet.findVOs | | | 查找多个表格 |
| Sheet.examineSheet | | | 审批单据 |

**表30 Sheet的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| Sheet.getExistedInfo | 语法 | public String[][] getExistedInfo() | |
| 前置条件 | 无 | |
| 后置条件 | 返回表格所需要的信息 | |
| Sheet.add | 语法 | public boolean add(SheetVO vo) | |
| 前置条件 | 无 | |
| 后置条件 | 返回增加单据的结果，并持久化更新涉及的领域对象的数据 | |
| Sheet.modify | 语法 | public boolean modify(long ID,SheetVO vo) | |
| 前置条件 | ID符合输入规则 | |
| 后置条件 | 根据ID启动一个更改表格回合，同时返回修改结果 | |
| Sheet.find | 语法 | public SheetVO find(long ID) | |
| 前置条件 | 启动一个查找单据信息回合,ID输入符合规则 | |
| 后置条件 | 根据输入的ID返回对应单据的信息 | |
| Sheet.findVOs | 语法 | public ArrayList<SheetVO> findVOs(FindingType type) | |
| 前置条件 | type输入符合规则 | |
| 后置条件 | 根据type返回所有符合条件的单据信息 | |
| Sheet.examineSheet | 语法 | public boolean examineSheet(long ID,SheetState state) | |
| 前置条件 | 启动一个审批单据回合 | |
| 后置条件 | 结束此次审批单据回合，持续更新持久化对象 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| DatabaseFactory.getSheetDatabase | | | 得到Sheet数据库的服务的引用 |
| SheetDataService.insert(SheetPO po) | | | 插入单一持久化对象 |
| SheetDataService.update(SheetPO po) | | | 更新单一持久化对象 |
| SheetDataService.find(long ID) | | | 根据ID进行查找单一持久化对象 |
| SheetDataService.finds(SheetType type) | | | 根据type进行查找多个持久化对象 |

# begininfobl模块

(1)模块概述

beginInfobl模块负责实现与期初建账相关的功能需求，具体功能需求和非功能需求请参见需求规格说明文档和体系结构设计文档。

(2)整体结构

beginInfobl模块实现了展示层与业务逻辑层之间的beginInfoblservice接口和业务逻辑层与数据层之间的beginInfodataservice接口。

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：  用例文档中的“用例22期初建账”  软件需求规格说明文档中的3.2节和3.3节 |
| 体系结构 | //被presentation层调用的接口  public interface BeginInfoBLService {  public boolean fillInfo(BeginInfoVO vo);  public BeginInfoVO getInfo();  }  //被DataService层调用的接口  public interface BeginInfoDataService extends Remote {  public void insert(BeginInfoPO po)throws RemoteException;  public BeginInfoPO get()throws RemoteException;  } |
| 输出 | 类图 |  |

**表31 期初建账业务逻辑层详细设计的上下文**

(3)模块内部类的接口规范

**表32 BeginInfoController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| BeginInfoController. fillInfo | 语法 | public boolean fillInfo(beginInfoVO vo); | |
| 前置条件 | 已创建一个BeginInfo领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用BeginInfo领域对象的fillinfo方法 | |
| BeginInfoController. getInfo | 语法 | public BeginInfoVO getInfo(); | |
| 前置条件 | 已创建一个BeginInfo领域对象，并且输入符合输入规则 | |
| 后置条件 | 调用BeginInfo领域对象的getInfo方法 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| BeginInfo. fillInfo (beginInfoVO vo) | | | 创建期初信息对象 |
| BeginInfo. getInfo() | | | 获得一个期初信息对象 |

**表33 BeginInfo的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| beginInfo.fillInfo | 语法 | public boolean fillInfo(beginInfoVO vo); |
| 前置条件 | 启动系统 |
| 后置条件 | 系统保存期初信息并更新到持久化数据 |
| beginInfo.getInfo | 语法 | public BeginInfoVO getInfo(); |
| 前置条件 | 启动一次期初建账查看任务 |
| 后置条件 | 获取期初信息 |
| **需要的服务（需接口**） | | |
| 服务名 | | 服务 |
| BeginInfoDataService.insert(BeginInfoPO po) | | 插入单一持久化期初信息对象 |
| BeginInfoDataService.get() | | 获得单一持久化期初信息对象 |
| Account.init(ArrayList<AccountVO>) | | 初始化账户信息 |
| Commodity.init(ArrayList<StorageInitVO>) | | 初始化仓库信息 |
| Driver.init(ArrayList<DriverInfoVO>) | | 初始化司机信息 |
| Truck.init(ArrayList<TruckInfoVO>) | | 初始化车辆信息 |
| User.init(ArrayList<InitUserVO)> | | 初始化人员信息 |

# accountbl模块

(1)模块概述

accountbl模块负责实现与银行账户相关的功能需求，具体功能需求和非功能需求请参见需求规格说明文档和体系结构设计文档。

(2)整体结构

accountbl模块实现了展示层与业务逻辑层之间的accountblservice接口和业务逻辑层与数据层之间的accountdataservice接口。

**表34 银行账户管理业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：  用例文档中的“用例20银行账户管理”  软件需求规格说明文档中的3.2节和3.3节 |
| 体系结构 | //被presentation层调用的接口  public interface AccountBLService {  public boolean addAccount(String name,long num,double amount);    public void delAccount (long num);    public ArrayList<AccountVO> getAccountList();  }  //被DataService层调用的接口  public interface AccountDataService {  public void insert(AccountPO po) throws RemoteException;  public void delete(AccountPO po) throws RemoteException;    public ArrayList<AccountPO> getAll() throws RemoteException;  public void update(AccountPO po) throws RemoteException;  } |
| 输出 | 类图 |  |

(3)模块内部类的接口规范

**表35 AccountController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| AccountController.addAccount | 语法 | public boolean addAccount(String name,long num,double amount ); | |
| 前置条件 | 已创建一个Account领域对象，输入符合输入规则 | |
| 后置条件 | 调用Account领域对象的addAccount方法 | |
| AccountController.delAccount | 语法 | public void delAccount (long num); | |
| 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delAccount方法 | |
| Account.getAccountList | 语法 | public ArrayList<AccountVO> getAccountList(); | |
| 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getAccountList方法 | |
| **需要的服务（需接口**） | | | |
| 服务名 | | | 服务 |
| Account.addAccount(String name,long num,double amount ) | | | 加入一个账户对象 |
| Account.deleteAccount(long ID) | | | 删除一个账户对象 |
| Account.getAccountList() | | | 获得账户对象列表 |

**表36 Account的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| Account.addAccount | 语法 | public boolean addAccount(String name,long num,double amount ); | |
| 前置条件 | 开启一次银行账号管理任务 | |
| 后置条件 | 增加一个银行账户 | |
| Account.delAccount | 语法 | public void delAccount (long num); | |
| 前置条件 | 启动一次银行账号管理任务 | |
| 后置条件 | 删除一个银行账户 | |
| Account.getAccountList | 语法 | public ArrayList<AccountVO> getAccountList(); | |
| 前置条件 | 启动一次银行账号管理任务 | |
| 后置条件 | 获取账号列表 | |
| Account.updateAmount(给Sheet) | 语法 | public void updateAmount(long ID,double Amount); | |
| 前置条件 | 启动一次付款单填写任务 | |
| 后置条件 | 更新账户余额任务 | |
| Account.init(给BeginInfo) | 语法 | public void init(Arraylist<AccountVO> vo) | |
| 前置条件 | 启动一次期初建账任务 | |
| 后置条件 | 初始化期初信息 | |
| **需要的服务（需接口）** | | | |
| 服务名 | | | 服务 |
| AccountDataService.insert(AccountPO po) | | | 插入单一持久化账户对象 |
| AccountDataService.delete(AccountPO po) | | | 删除单一持久化账户对象 |
| AccountDataService.getAll () | | | 获得所有Account持久化账户对象 |
| AccountDataService.findl (long id) | | | 获得单一持久化账户对象 |
| AccountDataService.update(AccountPo po) | | | 更新单一持久化账户对象 |

# staffmanagebl模块

1. 模块概述

StaffManagebl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

StaffManagebl模块的职责及接口参见软件系统结构描述文档表14

1. 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.staffmanageblservice接口。业务逻辑层和数据层之间添加dataservice.staffmanagedataservice接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了StaffManageController。UserPO是作为管理用户信息的持久化对象被添加到设计模型中去的。

**表37 用户管理业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：快递物流系统用例文档V3.3 “用例三十 系统帐号管理”  快递物流系统需求规格说明文档V1.6中的3.2及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface StaffManageBLService {  public StaffInfoVO getStaffInfo(long StaffId);  public ArrayList<StaffInfoVO> getAllStaff();  public boolean changeAuthority(ArrayList<Integer> authority,String position);  public boolean addStaffInfo(StaffInfo VO StaffInfo);  public boolean deleteStaff(long StaffId);  public boolean endManagement();  }  //调用DataService层的接口  public interface StaffManageDataService extends Remote{  public UserPO find(long StaffId) throws RemoteException;  public ArrayList<UserPO> finds() throws RemoteException;  public void insert(UserPO po) throws RemoteException;  public void update(UserPO po) throws RemoteException;  public void delete(UserPO po) throws RemoteException;  public void finish() throws RemoteException;  } |
| 输出 | 类图 |  |

StaffManagebl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职 责 |
| StaffManage | 用户管理的领域模型对象，拥有一次用户管理的员工帐号信息,可以帮助完成用户管理所需要的服务 |
| StaffManageController | 负责实现用户管理界面所需要的服务 |

1. 模块内部类的接口规范

**StaffManageController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| StaffManageController.changeAuthority | 语法 | Public boolean changeAuthority (ArrayList<Integer> authority,String position) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中修改相应职位的权限 |
| StaffManageController. getStaffInfo | 语法 | Public StaffInfoVO getStaffInfo (long StaffId) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中获得员工信息 |
| StaffManageController. addStaffInfo | 语法 | Public void addStaffInfo (StaffInfoVO StaffInfo) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中增加员工帐号信息 |
| StaffManageController. deleteStaff | 语法 | Public void deleteStaff (long StaffId) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中删除员工帐号信息 |
| StaffManageController.endManagement | 语法 | Public void endManagement() |
| 前置条件 | 已确认所有管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| StaffManageController. getAllStaff | 语法 | Public ArrayList<StaffInfoVO> getAllStaff () |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中获得所有员工帐号信息 |
| SalaryController.changeSalary | 语法 | Public boolean changeSalary(long StaffId,SalaryVO salary) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中调整人员工资 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| StaffManage. changeAuthority (ArrayList<Integer> authority,String position) | 修改权限 | |
| StaffManage. getStaffInfo (long StaffId) | 获得单一对象 | |
| StaffManage. addStaffInfo (StaffInfoVO StaffInfo) | 增加员工账号 | |
| StaffManage. deleteStaff (long StaffId) | 删除员工 | |
| StaffManage. endManagement() | 结束管理 | |
| StaffManage. getAllStaff () | 获得所有员工信息 | |
| Salary.changeSalary(long StaffId,SalaryVO salary) | 修改薪水 | |

**表38 StaffManage的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| StaffManage.changeAuthority | 语法 | Public boolean changeAuthority (ArrayList<Integer> authority,String position) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中修改相应职位的权限 |
| StaffManage. getStaffInfo | 语法 | Public StaffInfoVO getStaffInfo (long StaffId) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中获得员工信息 |
| StaffManage. addStaffInfo | 语法 | Public void addStaffInfo (StaffInfoVO StaffInfo) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中增加员工帐号信息 |
| StaffManage. deleteStaff | 语法 | Public void deleteStaff (long StaffId) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中删除员工帐号信息 |
| StaffManage.endManagement | 语法 | Public void endManagement() |
| 前置条件 | 已确认所有管理 |
| 后置条件 | 保存操作日志，保存所有已更新数据 |
| StaffManage. getAllStaff | 语法 | Public ArrayList<StaffInfoVO> getAllStaff () |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中获得所有员工帐号信息 |
| Salary.changeSalary | 语法 | Public boolean changeSalary(long StaffId,SalaryVO salary) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中调整人员工资 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| StaffManageDataService. insert (StaffInfoPO po) | 插入单一持久化对象 | |
| StaffManageDataService. delete (StaffInfoPO po) | 删除单一持久化对象 | |
| StaffManageDataService. update (StaffInfoPO po) | 更新单一持久化对象 | |
| StaffManageDataService. find (long StaffId) | 根据ID进行查找单一持久化对象 | |
| StaffManageDataService. findAll () | 查找所有持久化对象 | |
| StaffManageDataService. finish() | 结束操作 | |

# LogInbl模块

（1）模块概述

logInbl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

logInbl模块的职责及接口参见软件系统结构描述文档表13

（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.logInblservice接口。业务逻辑层和数据层之间添加dataservice. logIndataservice接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了logInController。UserPO是作为个人帐号信息的持久化对象被添加到设计模型中去的。

**表15 登录业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：快递物流系统用例文档V3.3 “用例三十一 账号信息管理”  快递物流系统需求规格说明文档V1.6中的3.2及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface LogInBLService {  public ResultMessage logIn(long UserId, String password)  public UserVO getCurrentUser();  }  //调用DataService层的接口  public interface UserDataService extends Remote{  public void insert(ArrayList<UserPO> User) throws RemoteException;  public UserPO find(long UserId) throws RemoteException;  public void update(UserPO po) throws RemoteException;  public void finish() throws RemoteException;  public void InitialInsert(ArrayList<UserPO> User) throws RemoteException ;  public void delete(UserPO po) throws RemoteException;  public ArrayList<UserPO> finds() throws RemoteException;  } |
| 输出 | 类图 |  |

（3）模块内部类的接口规范

**表39 LogInController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| LogInController. logIn | 语法 | public ResultMessage logIn(long UserId, String password) |
| 前置条件 | 用户打开系统 |
| 后置条件 | 登录 |
| LogInController. getCurrentUser | 语法 | public UserVO getCurrentUser() |
| 前置条件 | 登陆成功 |
| 后置条件 | 获得当前登陆者信息 |

|  |  |
| --- | --- |
| **需要的服务（需接口）** | |
| 服务名 | 服务 |
| User. getPersonalInfo(long UserId) | 根据ID查找单一个人信息 |
| User. changePassword(String password,long userId) | 更改密码 |
| User. initial(ArrayList<InitUserVO> User) | 期初建账时初始化 |
| User. changeInfo(UserVO userVO) | 更改信息 |
| User. endManagement() | 结束管理 |

**LogIn的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| LogIn. logIn | 语法 | public ResultMessage logIn(long UserId, String password) |
| 前置条件 | 用户打开系统 |
| 后置条件 | 登录 |
| LogIn. getCurrentUser | 语法 | public UserVO getCurrentUser() |
| 前置条件 | 登陆成功 |
| 后置条件 | 获得当前登陆者信息 |

|  |  |
| --- | --- |
| **需要的服务（需接口）** | |
| 服务名 | 服务 |
| UserDataService.find (long UserId) | 根据ID进行查找单一持久化对象 |
| UserDataService. update (String password) | 更新单一持久化对象 |
| UserDataService. InitialInsert(ArrayList<UserPO> User) | 期初建账时初始化 |
| UserDataService. Insert(UserPO po) | 增加单一持久化对象 |
| UserDataService. delete(UserPO po) | 删除单一持久化对象 |
| UserDataService.finds() | 获得所有持久化对象 |
| UserDataService.finish() | 结束操作 |

# Positionbl模块

（1）模块概述

Positionbl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

Positionbl模块的职责及接口参见软件系统结构描述文档表13

（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.Positionblservice接口。业务逻辑层和数据层之间添加dataservice. Positiondataservice接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了PositionController。AuthorityPO是作为权限信息的持久化对象被添加到设计模型中去的。

**表15 职位权限业务逻辑层详细设计的上下文**

|  |  |  |
| --- | --- | --- |
| 输入 | 需求 | 参见：快递物流系统用例文档V3.3 “用例三十一 账号信息管理”  快递物流系统需求规格说明文档V1.6中的3.2及3.3节 |
| 体系结构 | //被Presentation层调用的接口  public interface PositionBLService {  public ArrayList<Integer> initialAuthority(StaffInfoVO staff)  public boolean changeStaffPosition(String position, long StaffId)  public boolean setCity(String city, long StaffId)  public boolean setBusinessHall(String businessHall, long StaffId)  public boolean changeAuthority(AuthorityVO authorityVO)  }  //调用DataService层的接口  public interface PositionDataService extends Remote{  public void insert(AuthorityPO authorityPO) throws RemoteException;  public AuthorityPO find(String position) throws RemoteException;  public ArrayList<AuthorityPO> finds() throws RemoteException;  public void delete(String position) throws RemoteException;  public void update(AuthorityPO authorityPO) throws RemoteException;  } |
| 输出 | 类图 |  |

（3）模块内部类的接口规范

**表39 PositionController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| PositionController.changeStaffPosition | 语法 | Public boolean changeStaffPosition(String position, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中更改员工职位 |
| PositionController. initialAuthority | 语法 | Public ArrayList<Integer> initialAuthority(StaffInfoVO staff) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中初始化新增员工权限 |
| PositionController. changeAuthority | 语法 | public boolean changeAuthority(AuthorityVO authorityVO) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中修改职位对应的权限 |
| PositionController. setCity | 语法 | Public boolean setCity(String city, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中设置人员所属城市 |
| PositionController. setBusinessHall | 语法 | Public boolean setBusinessHall(String businessHall, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中设置人员所属营业厅 |

|  |  |
| --- | --- |
| **需要的服务（需接口）** | |
| 服务名 | 服务 |
| Position.changeStaffPosition(String position, long StaffId) | 修改员工职位 |
| Position.initialAuthority(StaffInfoVO staff) | 初始化权限 |
| Position. changeAuthority(AuthorityVO authorityVO) | 修改职位对应的权限 |
| Position.setCity(String city, long StaffId) | 设置员工所属城市 |
| Position.setBusinessHall(String businessHall, long StaffId) | 设置员工所属营业厅 |

**Position的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Position.changeStaffPosition | 语法 | Public boolean changeStaffPosition(String position, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中更改员工职位 |
| Position. initialAuthority | 语法 | Public ArrayList<Integer> initialAuthority(StaffInfoVO staff) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中初始化新增员工权限 |
| Position. changeAuthority | 语法 | public boolean changeAuthority(AuthorityVO authorityVO) |
| 前置条件 | 管理员启动员工帐号管理 |
| 后置条件 | 在一次员工帐号管理中修改职位对应的权限 |
| Position. setCity | 语法 | Public boolean setCity(String city, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中设置人员所属城市 |
| Position. setBusinessHall | 语法 | Public boolean setBusinessHall(String businessHall, long StaffId) |
| 前置条件 | 总经理启动人员机构管理 |
| 后置条件 | 在一次人员机构管理中设置人员所属营业厅 |

|  |  |
| --- | --- |
| **需要的服务（需接口）** | |
| 服务名 | 服务 |
| PositionDataService.find(String position) | 根据职位查找权限 |
| PositionDataService.finds() | 获得所有职位的权限 |
| PositionDataService.insert(AuthorityPO authorityPO) | 增加一个职位及其权限 |
| PositionDataService.delete(String position) | 删除一个职位 |
| PositionDataService.update(AuthorityPO authorityPO) | 更改权限 |
| PositionDataService.find(String position) | 根据职位查找权限 |

# 客户端业务逻辑层模块局部的行为

# userbl模块



图1 获得个人帐号信息的顺序图

获得仓库管理对应的城市顺序图参见userbl顺序图



图2 修改密码的顺序图



图3 User对象状态图

# nonuserbl模块

参见userbl模块顺序图和对象状态图。

# insititutionbl模块



图4 人员工资调整顺序图

增加、删除、查询机构顺序图参见user顺序图



图5 人员机构管理状态图

# commoditybl模块

图6表明了快递物流系统中，库存盘点界面想要知道快递信息时候的顺序图。

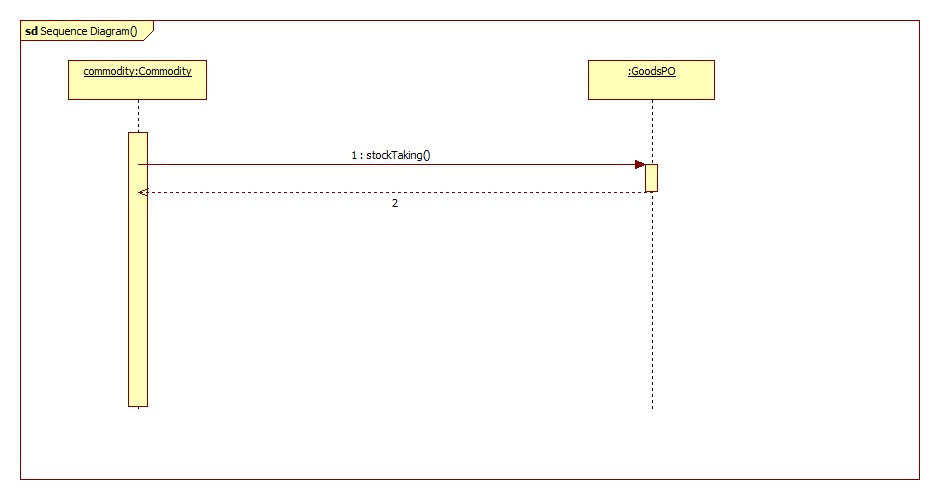


图6 得到快递信息的顺序图

图7为修改警戒比例的顺序图。

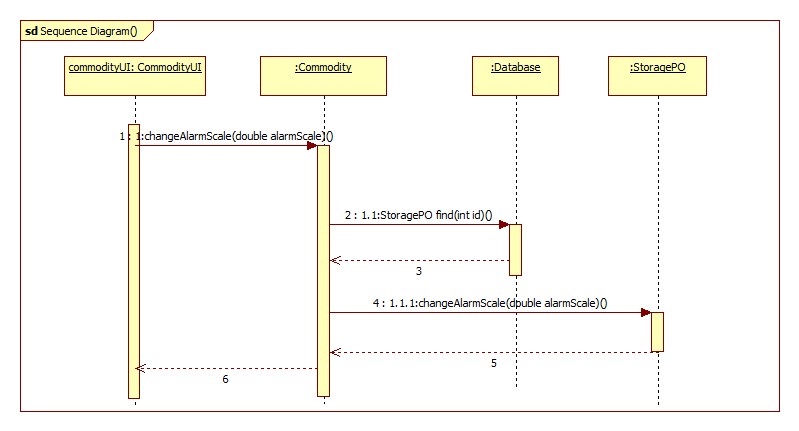


图7 修改警戒比例的顺序图

图8为库存查看界面得到库存信息的顺序图。

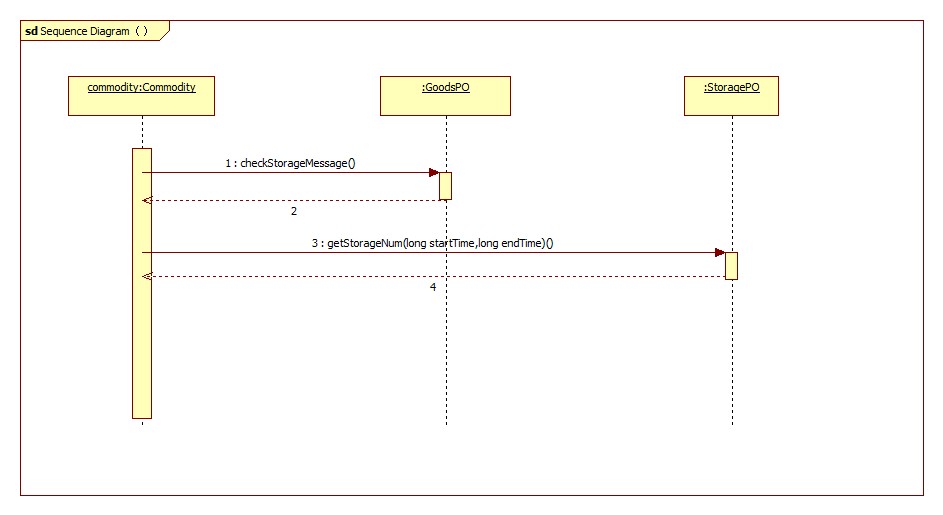


图8 得到库存信息的顺序图

图9为库存分区的顺序图

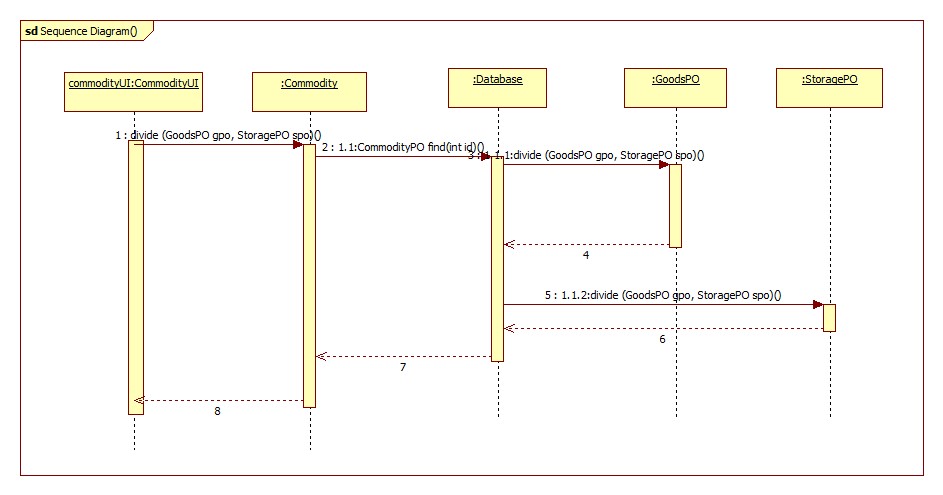


图9 库存分区的顺序图

图10所示的状态图描述了Commodity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。

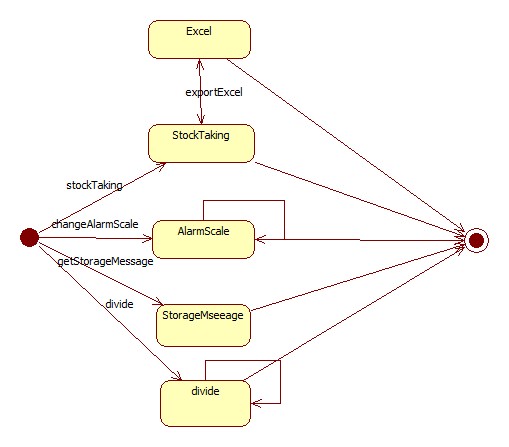


图10 Commodity对象状态图

# logbl模块

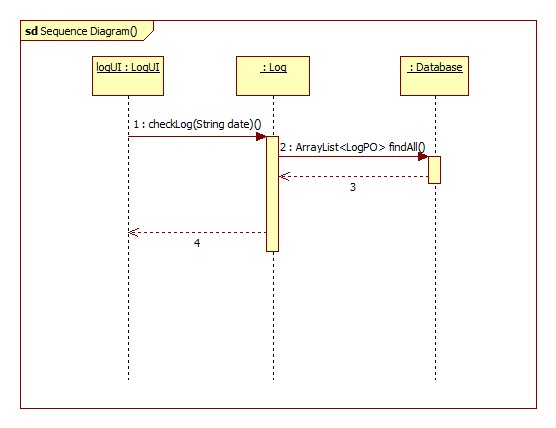
****

图11 查看日志的顺序图

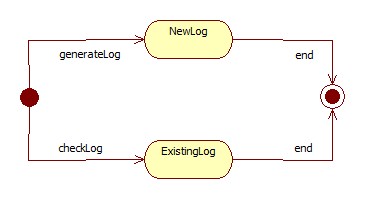
****

图12 Log对象状态图

# sheetbl模块

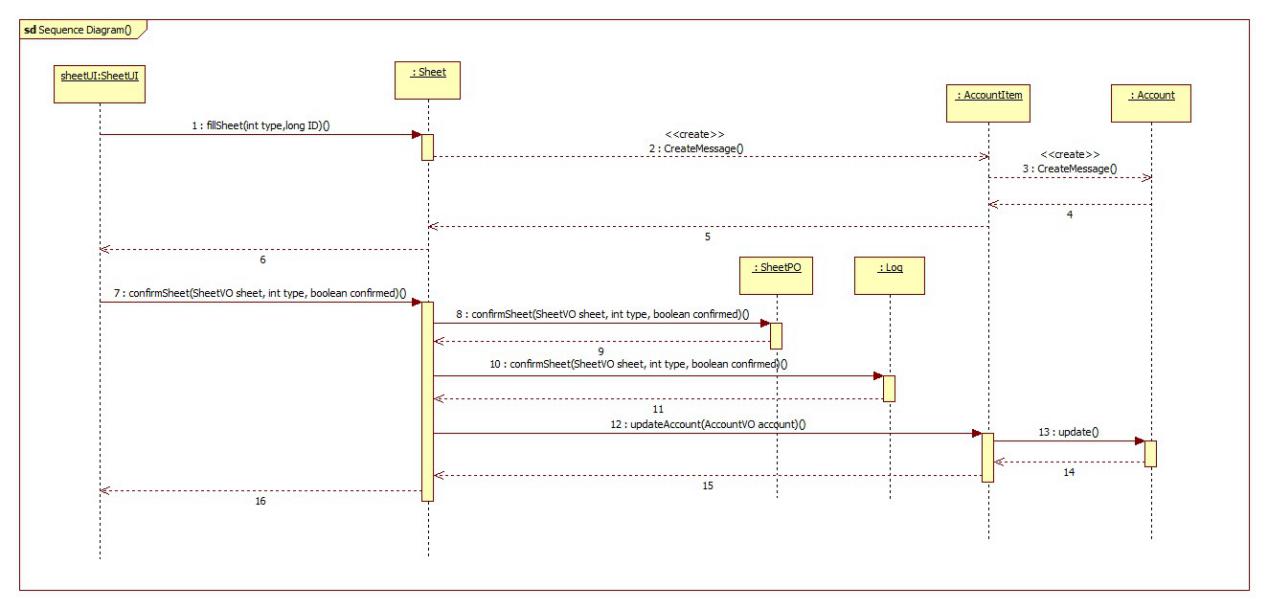
****

图13填写付款单的顺序图

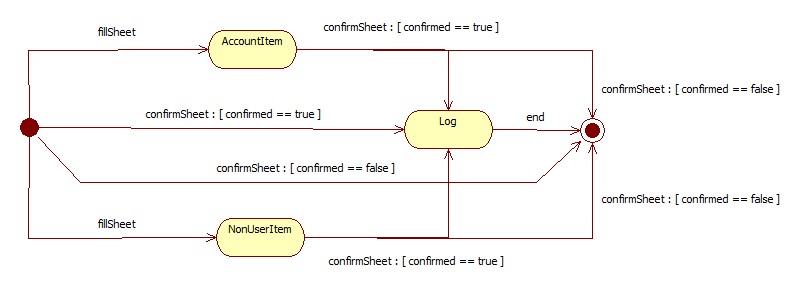


图14 Sheet对象状态图

# begininfobl模块

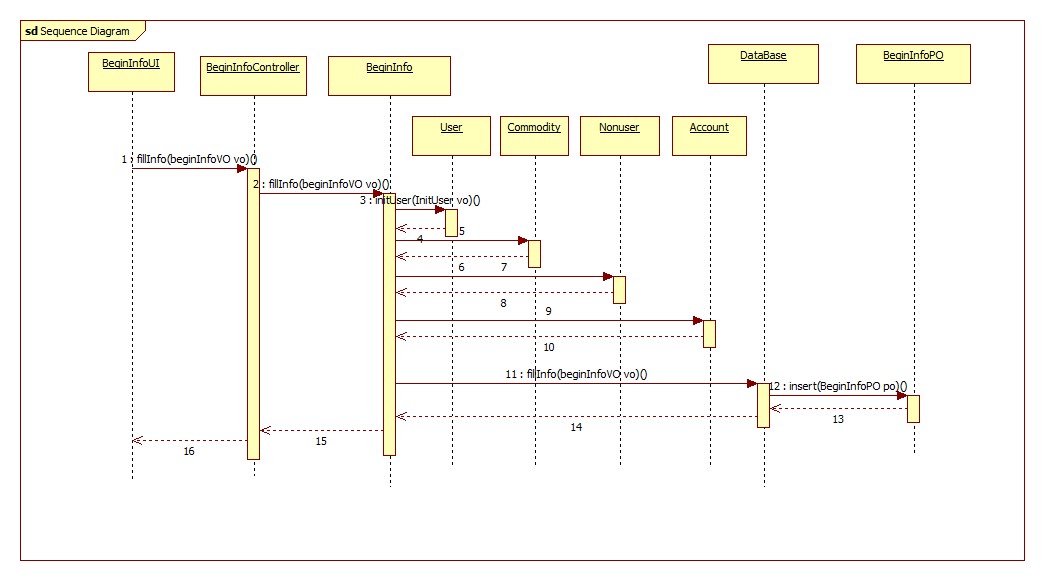
****

图15期初建账的顺序图



图16 BeginInfo对象状态图

# accountbl模块

参见userbl模块顺序图和对象状态图。

# staffmanagebl模块

增加、删除、更改、查询员工帐号信息顺序图参见userbl顺序图



图17 用户管理状态图

# LogInbl模块

参见userbl模块顺序图和对象状态图。

# 4.2.3.11.Positionbl模块

参见userbl模块顺序图和对象状态图。

# 客户端业务逻辑层模块的设计原理

利用委托式控制风格，每个模块需要访问的业务逻辑有各自的控制器委托给不同的领域对象。这些对象分别承担一定的职责，做出一定的决策，从而实现大的职责。

# 客户端网络模块的静态结构和动态职责

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。如图33所示。

具体网络通信层的实现是通过远程方法调用RMI来实现的。



图33 客户端数据通信层模块的描述

# 客户端网络模块局部模块的职责

其实网络通信层的实际运作代码是由RMI API所完成的，在客户端我们只需要部署RMI服务的Stub文件和Java RMI类库即可。如表25所示。

表25 网络通信层模块的职责

|  |  |
| --- | --- |
| **模块** | **职责** |
| IDataRemoteService\_Stub | 给业务逻辑层提供服务接口 |
| Java RMI 类库 | 实现RMI机制 |

# 服务器端网络模块的静态结构和动态职责

# 服务器端网络模块局部模块的职责

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。

具体服务器端模块中只需要包含Java RMI类库即可。但是其数据层实现RMI的类需要符合以下条件：

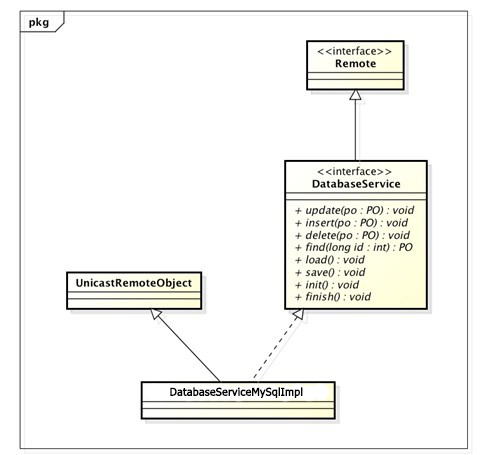


图34 服务器数据层实现RMI的类的描述

# 服务器端网络模块局部模块的接口规范

DatabaseService 是被业务逻辑层调用的接口。为了实现RMI， DatabaseService需要继承 Remote接口，而数据层中的实现这个接口的类DatabaseServiceSerializableImpl都需要继承UnicastRemoteObject类。

以下是服务器网络模块的RMI详细的规范：

**表39 服务器网络模块的RMI接口规范**

|  |  |
| --- | --- |
| **类/接口** | **规范** |
| 远程服务接口 | extends Remote，方法 throws RemoteException |
| 远程服务的实现类 | extendsUnicastRemoteObject |

**表40 服务器网络模块的RMI接口返回值的规范**

|  |  |
| --- | --- |
| **类/接口** | **规范** |
| 作为远程引用传递 | 该类需要extends Remote，方法 throws RemoteException |
| 通过数据库进行复制 | Implements MySql |

# 服务器端数据层模块的静态结构和动态职责

# 服务器端数据层模块局部模块的职责

数据层主要给业务逻辑层提供数据访问服务。包括对于持久化数据的、增、删、改、查。这些服务由DataService接口提供。由于持久化数据的保存可能存在多种形式：Txt文件、序列化文件、数据库等。而且还可能出现混搭的可能。具体如图所示。

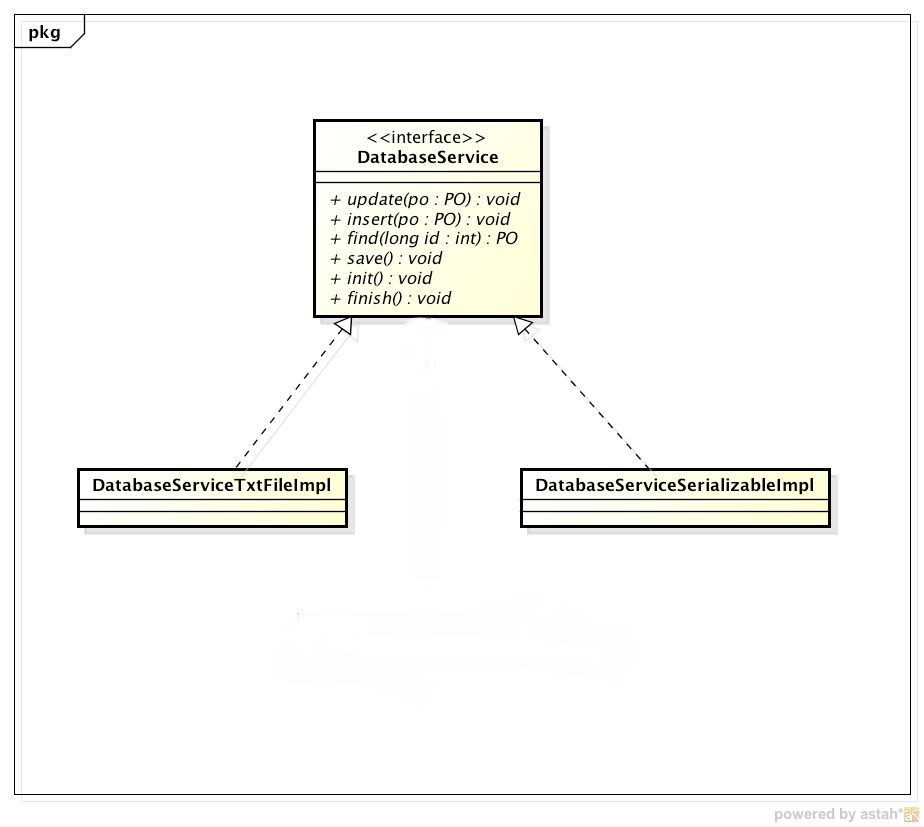


图35 数据层模块的描述

如表41所示为数据层模块的职责。

**表41 数据层模块的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| DatabaseServiceTxtFileImpl | 基于Txt文件的持久化数据库的接口，提供集体载入、集体保存、增、删、改、查服务。 |
| DatabaseServiceSerializableFileImpl | 基于序列化文件的持久化数据库的接口，提供集体载入、集体保存、增、删、改、查服务。 |

# 服务器端数据层模块局部模块的接口规范

服务器端数据层模块的接口规范如表42所示。

**表42 数据层模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| DataService.find | 语法 | public PO find(long id) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的PO结果 |
| DataService.insert | 语法 | public void insert(PO po) throws RemoteException; |
| 前置条件 | 同样ID的po在Mapper中不存在 |
| 后置条件 | 在Mapper中增加一个po记录 |
| DataService.delete | 语法 | public void delete(PO po) throws RemoteException; |
| 前置条件 | 在Mapper中存在同样ID 的po |
| 后置条件 | 删除一个po |
| DataService.update | 语法 | public void update(PO po) throws RemoteException; |
| 前置条件 | 在Mapper中存在同样ID 的po |
| 后置条件 | 更新一个po |
| DataService.init | 语法 | public void init() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 初始化持久化数据库 |
| DataService.finish | 语法 | public void finish()throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用。 |

# 服务器端数据层模块的设计原理

服务器端接受请求，处理数据后生成响应返回数据。

# 依赖关系

图34和图35是客户端和服务器端各自的包之间的依赖关系。

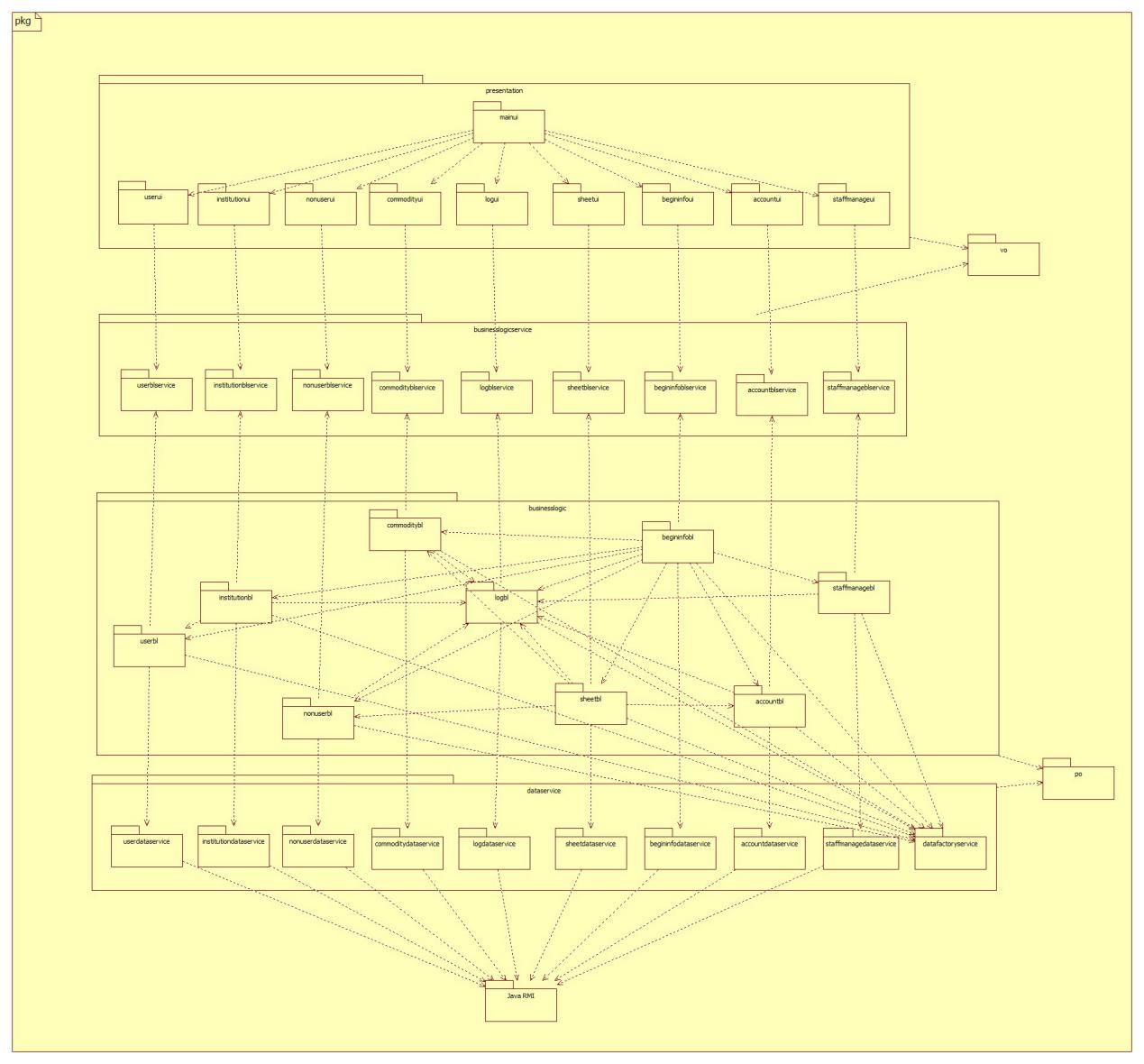


图36 客户端开发包图

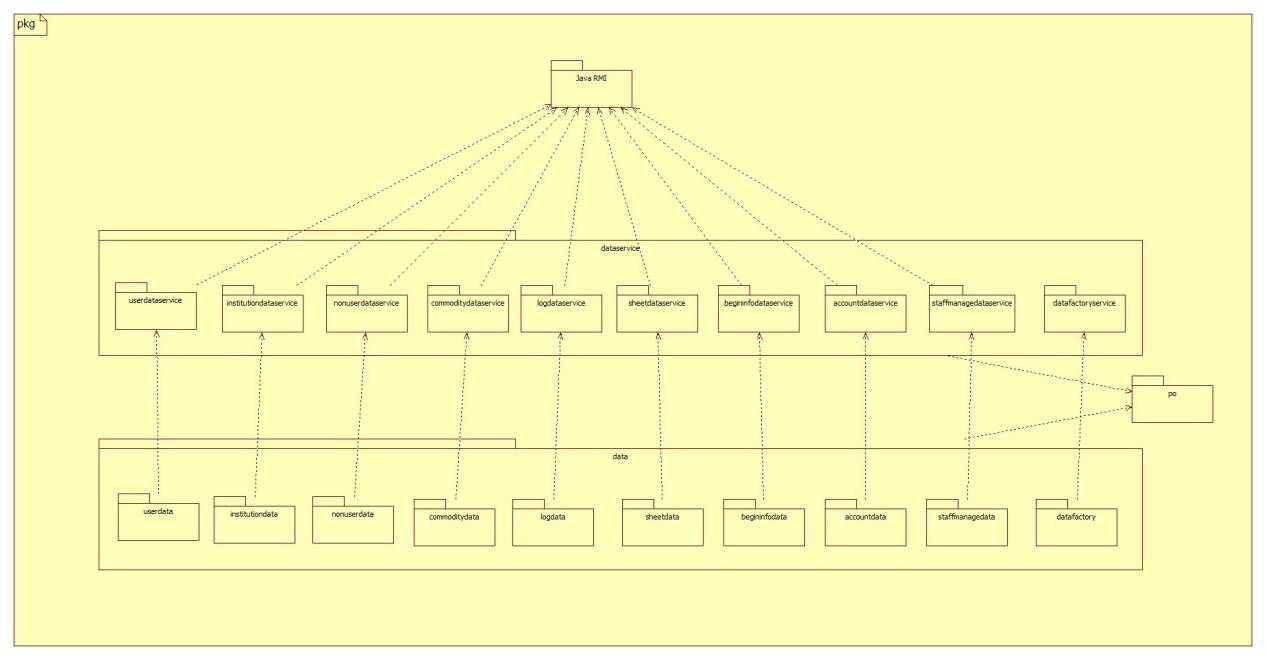


图37 服务器端开发包图