快递管理系统 MSE(Management System of Expressage)

软件详细设描述文档

V1.0

完整版

一组

成员：吕德超、李昊朔、阙俊杰、罗铉斌

2015-11-5

# 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| **修改人员** | **日期** | **变更原因** | **版本号** |
| 全体人员 | 2015-10-21 | 最初草稿（不完整） | V1.0 草稿 |

# 引言

## 编制目的

本报告详细完成对快递管理系统的详细设计，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 |
| MSE | 快递管理系统 |  |
| checklist | 订单查询 | 订单查询 |
| listin | 单据输入 | 订单输入、收件人信息输入 |
| transfer | 中转接收 | 接收与派件、中转接收、录入中转单 |
| load | 装运管理 | 车辆装车管理、装运管理 |
| vehicle | 车辆管理 | 司机信息管理、车辆信息管理 |
| finance | 财务管理 | 收款单管理，结算管理，成本管理，查看成本收益表，统计报表，查看经营情况 |
| store | 仓库管理 | 入库管理，出库管理，库存管理 |
| bill | 期初建账 | 期初建账 |
| account | 账户管理 | 账户管理，帐户查询 |
| staff | 员工管理 | 人员机构管理，用户信息管理 |
| policy | 策略管理 | 策略管理 |
| approve | 审批单据 | 审批单据 |
| user | 用户 |  |
| order | 订单 |  |
| receiver | 收件人信息 |  |
| loadorder | 装车单 |  |
| driver | 司机信息 |  |
| arriveorder | 到达单 | 中转中心到达单 |
| receiveorder | 接收单 | 营业厅接收单 |
| sendorder | 派件单 |  |
| payeeorder | 收款单 |  |
| changeorder | 中转单 |  |
| storeinorder | 入库单 |  |
| storeoutorder | 出库单 |  |
| payorder | 付款单 |  |
| startinfo | 期初信息 |  |
| salary | 薪水信息 |  |
| constant | 常量信息 |  |

## 参考资料

1. IEEE标准
2. 快递管理系统MSE用例文档
3. 快递管理系统MSE结构逻辑设计文档

# 产品概述

参考快递管理系统用例文档和快递管理系统软件需求规格说明文档中对产品的概括描述。

# 体系结构设计概述

参考快递管理系统概要设计文档中对体系结构文档的概述。

# 结构视角

## 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

### Checklistbl模块

#### 4.1.1.1订单查询

###### 模块描述

ChecklistBl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。

ChecklistBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.checklistblservice.ChecklistBlService接口。业务逻辑层和数据层之间添加dataservice.checklistdataservice.ChecklistDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ChecklistController，将对查询订单操作的业务逻辑委托给Check 对象。ListinInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。



###### 模块内部类的接口规范

ChecklistController的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| ChecklistController.getOrder | | 语法 | public ResultMessage getOrder(String id) |
| 前置条件 | 已创建一个Check对象，并且输入符合输入规则 |
| 后置条件 | 调用Check领域对象的getOrder方法 |
| **需要的服务（需接口）** | | | |
| 服务名 | 服务 | | |
| Check.getOrder(String id) | 根据ID进行查找单一持久化对象 | | |

Check的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Check.getOrder | 语法 | public ResultMessage getOrder(String id) |
| 前置条件 | 启动一个订单查询回合 |
| 后置条件 | 在一个订单查询回合中返回所查询的订单信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| checklistDataService.find(String id) | 根据ID进行查找单一持久化对象 | |

###### 业务逻辑层的动态模型



###### 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。

### Listinbl模块

#### 4.1.2.1具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

#### 4.1.2.2具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

### Transferbl模块

#### 4.1.3.1中转接收

###### 模块描述

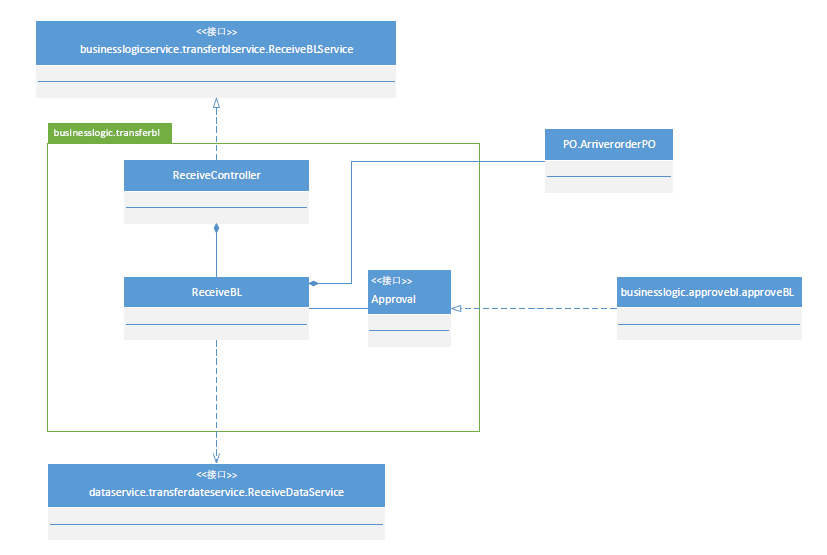
中转接收模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

中转接收模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.transferblservice.ReceiveBLService接口。业务逻辑层和数据层之间添加dataservice.transferdateservice. ReceiveDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiveController，这样ReceiveController会将中转接收的业务逻辑处理委托给ReceiveBL对象。ArriverorderPO是作为到达单的持久化对象被添加到设计模型中去的。Approval是为了将单据提交给总经理审批而产生的接口。

中转接收模块的设计如图：



中转接收模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| ReceiveController | 负责实现中转接收界面所需要的服务。 |
| ReceiveBL | 具有新建到达单、设置消息提醒、查看消息提醒、存储新建的到达单等功能，能够帮助完成中转接收界面所需要的服务。 |

###### 模块内部类的接口规范

**ReceiveController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Receive.build | 语法 | public void build(LoadorderVO lv,ArriveState as) |
| 前置条件 | 已创建一个ReceiveBL领域对象，且输入符合输入规范 |
| 后置条件 | 调用ReceiveBL领域的build方法 |
| Receive.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 已创建一个ReceiveBL领域对象 |
| 后置条件 | 调用ReceiveBL领域的checkRemind方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Receive.build | 新建一张到达单 | |
| Receive.checkRemind | 查看消息提醒 | |

**ReceiveBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Receive.build | 语法 | public void build(LoadorderVO lv,ArriveState as) |
| 前置条件 | 启动一个新建到达单回合 |
| 后置条件 | 系统将新建的到达单提交给总经理审批 |
| Receive.setRemind | 语法 | public void setRemind(LoadorderVO lv) |
| 前置条件 | 有货物到达中转中心 |
| 后置条件 | 增加一条提醒消息 |
| Receive.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 启动一个新建到达单回合 |
| 后置条件 | 删除该条提醒消息 |
| Receive.saveOrder | 语法 | public void save(ArriverorderVO av) |
| 前置条件 | 到达单已通过总经理审核 |
| 后置条件 | 将到达单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Approval.checkOrder(PO) | 提交给总经理审核单据 | |
| ReceiveDataService.insert(ArriverorderPO av) | 插入一个持久化对象ArriverorderPO | |

###### 业务逻辑层的动态模型

顺序图参照4.1.4.2装车管理

###### 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。

#### 4.1.3.2录入中转单

###### 模块描述

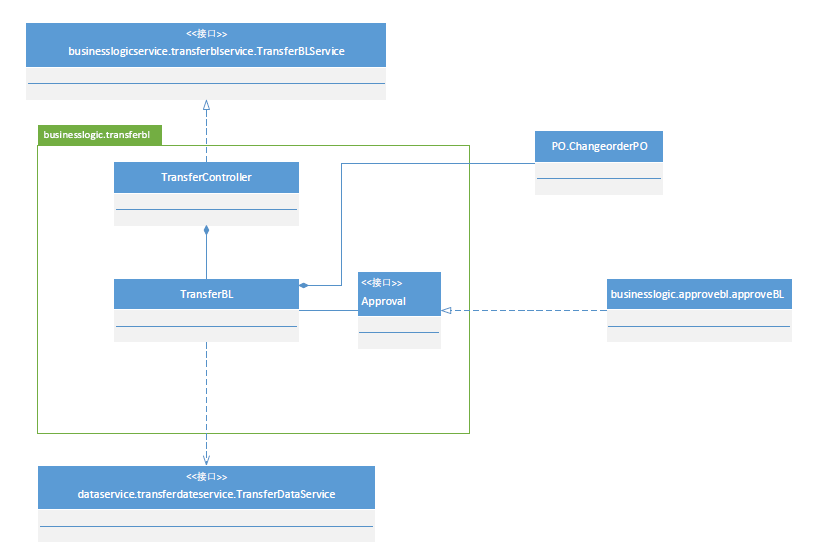
录入中转单模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

录入中转单模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.transferblservice.TransferBLService接口。业务逻辑层和数据层之间添加dataservice.transferdateservice. TransferDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了TransferController，这样TransferController会将录入中转单的业务逻辑处理委托给TransferBL对象。ChangeorderPO是作为中转单的持久化对象被添加到设计模型中去的。Approval是为了将单据提交给总经理审批而产生的接口。

录入中转单模块的设计如图：



录入中转单模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| TransferController | 负责实现录入中转单界面所需要的服务。 |
| TransferBL | 具有新建中转单、设置消息提醒、查看消息提醒、查看库存信息、存储新建的中转单等功能，能够帮助完成录入中转单界面所需要的服务。 |

###### 模块内部类的接口规范

**TransferController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Transfer.build | 语法 | public void build(int[][][][] location,String car\_number,String monitor) |
| 前置条件 | 已创建一个TransferBL领域对象，且输入信息符合输入规范 |
| 后置条件 | 调用TransferBL领域的build方法 |
| Transfer.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 已创建一个TransferBL领域对象 |
| 后置条件 | 调用TransferBL领域的checkRemind方法 |
| Transfer.checkInventory | 语法 | public ResultMessage checkInventory() |
| 前置条件 | 已创建一个TransferBL领域对象 |
| 后置条件 | 调用TransferBL领域的checkInventory方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Transfer.build | 新建一张中转单 | |
| Transfer.checkRemind | 查看消息提醒 | |
| Transfer.checkInventory | 查看此时中转中心仓库库存信息 | |

**TransferBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Transfer.build | 语法 | public void build(int[][][][] location,String car\_number,String monitor) |
| 前置条件 | 启动一个录入中转单回合 |
| 后置条件 | 系统将中转单提交总经理审批 |
| Transfer.setRemind | 语法 | public void setRemind() |
| 前置条件 | 有库区库存值达到提醒值 |
| 后置条件 | 增加一条提示消息 |
| Transfer.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 启动一个录入中转单回合 |
| 后置条件 | 删除该条消息 |
| Transfer.checkInventory | 语法 | public ResultMessage checkInventory() |
| 前置条件 | 启动一个录入中转单回合 |
| 后置条件 | 返回各个库区的库存信息 |
| Transfer.saveOrder | 语法 | public void save(ChangeorderVO cv) |
| 前置条件 | 中转单通过总经理审批 |
| 后置条件 | 将中转单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Approval.checkOrder(PO) | 提交给总经理审核单据 | |
| Inventory\_management.checkInventory\_now | 查看此时中转中心仓库库存信息 | |
| TransferDataService.insert(ChangeorderPO cp) | 插入一个持久化对象ChangeorderPO | |

###### 业务逻辑层的动态模型

顺序图参照4.1.4.2装车管理

###### 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。

#### 4.1.3.3录入接收单

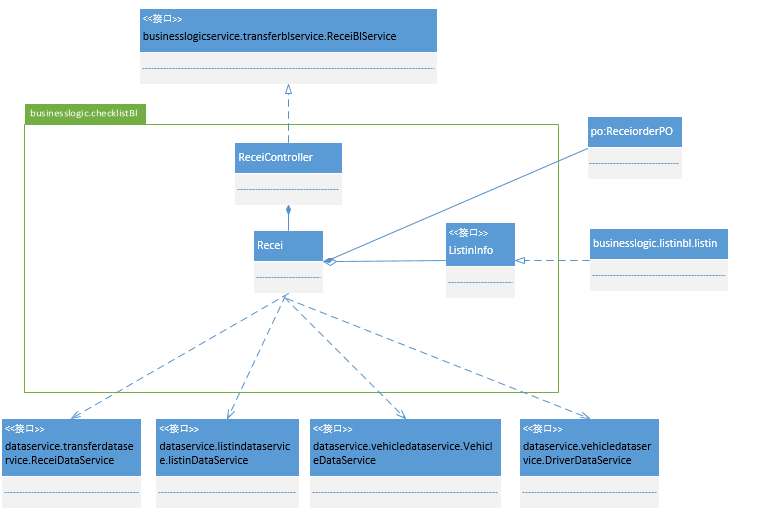
###### 模块描述

ReceiBl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。

ReceiBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.transferblservice.ReceiBlService接口。业务逻辑层和数据层之间添加dataservice.transferdataservice.ReceiDataService接口，还与dataservice.listindataservice.listinDataService,dataservice.vehicledataservice.VehicleDataService, dataservice.vehicledataservice.DriverDataService。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiController，将对新建接收单操作的业务逻辑委托给Recei 对象。ReceiorderPO是作为接收单信息的持久化被添加到设计模型里的。 ListinInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。



ReceiBl模块各个类的职责如表 所示：

表5 ReceBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| ReceiController | 负责实现新建接收单界面所需要的服务 |
| Recei | 新建接收单的领域模型对象，能够获得已有的订单信息，帮助新建接收单界面完成所需要的任务 |

###### 模块内部类的接口规范

ReceiController接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReceiController.addReceiveOrder | 语法 | public ResultMessasge addReceiveOrder (String data, String changeOrderNum, String departPlace,String arriveState, ArrayList<String> orderNum, String name ) |
| 前置条件 | 已创建一个Recei领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Recei领域对象的addReceiveOrder方法 |
| ReceiController.endRecei | 语法 | public void endRecei() |
|  | 前置条件 | 已创建一个Recei领域对象，并且确认信息后 |
|  | 后置条件 | 调用Recei领域对象的endRecei方法 |
| ReceiController.exportReceiOrder | 语法 | public ResultMessage exportReceiOrder(ReceiveorderPO po) |
|  | 前置条件 | 已创建一个Recei领域对象， 并输入完成并确认保存后 |
|  | 后置条件 | 调用Recei领域对象的exportReceiOrder方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Recei.addReceiveOrder(String data, String changeOrderNum, String departPlace,String arriveState, ArrayList<String> orderNum, String name) | 加入一个接收单 | |
| Recei.endRecei() | 结束新建接收单操作 | |
| Recei.exportReceiOrder(ReceiveorderPO po) | 导出接收单操作 | |

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Recei.addReceiveOrder | 语法 | public ResultMessasge addReceiveOrder (String data, String changeOrderNum, String departPlace,String arriveState, ArrayList<String> orderNum, String name ) |
| 前置条件 | 启动一个接收回合 |
| 后置条件 | 在一个接收回合中，新建一张接收单 |
| Recei.endRecei | 语法 | public void endRecei() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次接收，持久化更新涉及的对象数据 |
| Recei.exportReceiOrder | 语法 | public ResultMessage exportReceiOrder(ReceiveorderPO po) |
|  | 前置条件 | 填写接收单完成且确认保存后 |
|  | 后置条件 | 导出excel文件 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ReceiDataService.insert(ReceiveorderPO po) | 增加单一持久化对象 | |
| VehicleDataService.updata(String key) | 更新车辆信息持久化对象中的车辆状态及所在地 | |
| DriverDataService.updata(String key) | 更新司机信息持久化对象中的车辆状态及所在地 | |
| ListinDataService.updata(String key) | 更新订单物流状态 | |

###### 业务逻辑层的动态模型



###### 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。

#### 4.1.3.4录入派件单

###### 模块描述

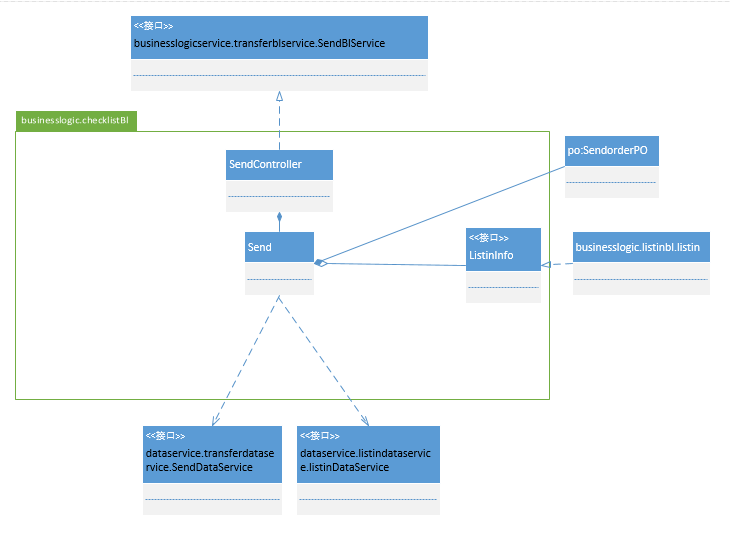
SendBl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

SendBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.transferblservice.SendBlService接口。业务逻辑层和数据层之间添加dataservice.transferdataservice.SendDataService接口，还与dataservice.listindataservice.listinDataService。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SendController，将对新建派件单操作的业务逻辑委托给Send 对象。SendorderPO是作为派件单信息的持久化被添加到设计模型里的。ListinInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

SendBl模块的设计如图 所示：



SendBl模块各个类的职责如表 所示：

表5 SendBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| SendController | 负责实现新建派件单界面所需要的服务 |
| Send | 新建派件单的领域模型对象，能够获得已有的订单信息，帮助新建派件单界面完成所需要的任务 |

###### 模块内部类的接口规范

SendController接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| SendController.addSendOrder | 语法 | public ResultMessasge addSendOrder (String data, ArrayList<String> orderNum, String sender ) |
| 前置条件 | 已创建一个Send领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Send领域对象的addSendOrder方法 |
| SendController.endSend | 语法 | public void endSend() |
|  | 前置条件 | 已创建一个Send领域对象，并且确认信息后 |
|  | 后置条件 | 调用Send领域对象的endSend方法 |
| SendController.exportSendOrder | 语法 | public ResultMessage exportSendOrder(SendorderPO po) |
|  | 前置条件 | 已创建一个Send领域对象， 并输入完成并确认保存后 |
|  | 后置条件 | 调用Send领域对象的exportSendOrder方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Send.addSendOrder(String data, ArrayList<String> orderNum, String sender) | 加入一个派件单 | |
| Send.endSend() | 结束新建派件单操作 | |
| Send.exportSendOrder(SendorderPO po) | 导出派件单操作 | |

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Send.addSendOrder | 语法 | public ResultMessasge addSendOrder (String data, ArrayList<String> orderNum, String sender ) |
|  | 前置条件 | 启动一个派件回合 |
|  | 后置条件 | 在一个派件回合中，新建一张派件单 |
| Send.endSend | 语法 | public void endSend() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次派件，持久化更新涉及的对象数据 |
| Send.exportSendOrder | 语法 | public ResultMessage exportSendOrder(SendorderPO po) |
|  | 前置条件 | 填写派件单完成且确认保存后 |
|  | 后置条件 | 导出excel文件 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| SendDataService.insert(SendorderPO po) | 增加单一持久化对象 | |
| ListinDataService.updata(String key) | 更新订单物流状态 | |

###### 业务逻辑层的动态模型



###### 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。

### Loadbl模块

#### 4.1.4.1装运管理

###### 模块描述

装运管理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。

装运管理模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.loadblservice.ShippingBLService接口。业务逻辑层和数据层之间添加dataservice.loaddateservice.ShippingDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ShippingController，这样ShippingController会将装运管理的业务逻辑处理委托给ShippingBL对象。LoadorderPO是作为装运单的持久化对象被添加到设计模型中去的。Approval是为了将单据提交给总经理审批而产生的接口。

装运管理模块的设计如图：



装运管理模块各个类的设计

装运管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| ShippingController | 负责实现装运管理界面所需要的服务。 |
| ShippingBL | 具有新建装运单单、设置消息提醒、查看消息提醒、存储新建的装运单等功能，能够帮助完成装运管理界面所需要的服务。 |

###### 模块内部类的接口规范

**ShippingController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Shipping.build | 语法 | public void build(ChangeorderVO cv,String car\_number,String monitor,String transfer) |
| 前置条件 | 已创建一个ShippingBL领域对象，且输入符合规范 |
| 后置条件 | 调用ShippingBL领域的build方法 |
| Shipping.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 已创建一个ShippingBL领域对象 |
| 后置条件 | 调用ShippingBL领域的checkRemind方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Shipping.build | 新建一张装运单 | |
| Shipping.checkRemind | 查看消息提醒 | |

**ShippingBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Shipping.build | 语法 | public void build(ChangeorderVO cv,String car\_number,String monitor,String transfer) |
| 前置条件 | 启动一个新建装运单回合 |
| 后置条件 | 系统将装运单提交给总经理审批 |
| Shipping.setRemind | 语法 | public void setRemind(ChangeorderVO cv) |
| 前置条件 | 需要装运的中转单上的货物已出库完毕 |
| 后置条件 | 增加一条提示消息 |
| Shipping.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 启动一个新建装运单回合 |
| 后置条件 | 删除该条消息 |
| Shipping.saveOrder | 语法 | public void save(LoadorderVO) |
| 前置条件 | 到达单已经通过总经理审核 |
| 后置条件 | 将中转单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Approval.checkOrder(PO) | 提交给总经理审核单据 | |
| ShippingDataService.insert(LoadorderPO lp) | 插入一个持久化对象LoadorderPO | |

###### 业务逻辑层的动态模型

顺序图参照4.1.4.2装车管理

###### 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。

#### 4.1.4.2装车管理

###### 模块描述

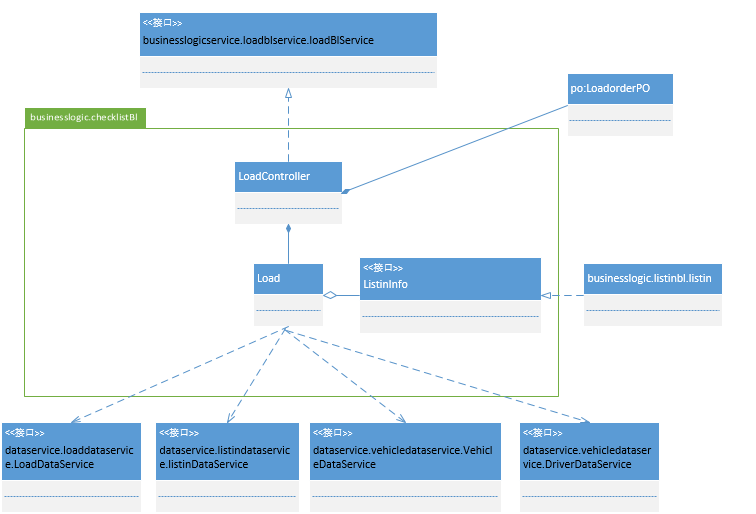
LoadBl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

LoadBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.loadblservice.loadBlService接口。业务逻辑层和数据层之间添加dataservice.loaddataservice.loadDataService接口，还与dataservice.listindataservice.listinDataService,dataservice.vehicledataservice.VehicleDataService, dataservice.vehicledataservice.DriverDataService。为了隔离业务逻辑职责和逻辑控制职责，我们增加了LoadController，将对新建接收单操作的业务逻辑委托给Send 对象LoadorderPO是作为派送单信息的持久化被添加到设计模型里的。 ListinInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

LoadBl模块的设计如图 所示：



LoadBl模块各个类的职责如表 所示：

表5 LoadFormBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoadController | 负责实现新建装车单界面所需要的服务 |
| Load | 新建装车单的领域模型对象，能够获得已有的订单信息，帮助新建装车单界面完成所需要的任务 |

###### 模块内部类的接口规范

LoadController的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| LoadController.addLoadOrder | 语法 | public ResultMessasge addLoadOrder(String data, String shop, String c  arShippingNum,String arrival, String vehicleNum, String monitor，String transfer, ArrayList<String> orderNum ) |
| 前置条件 | 已创建Load领域对象，且输入符合输入规则 |
| 后置条件 | 调用Load领域对象的addLoadOrder方法 |
| LoadController.getTotal | 语法 | public double getTotal(String arrival) |
|  | 前置条件 | 已创建Load领域对象， |
|  | 后置条件 | 调用Load领域对象的getTotal方法 |
| LoadController.endLoad | 语法 | public void endLoad() |
|  | 前置条件 | 已创建Load领域对象，并已确认信息 |
|  | 后置条件 | 调用Load领域对象的endLoad方法 |
| LoadController.export | 语法 | public ResultMessage export(LoadorderPO po) |
|  | 前置条件 | 已创建Load领域对象，并已完成输入且确定保存后 |
|  | 后置条件 | 调用Load领域对象的export方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Load.addLoadOrder(String data, String shop, String c  arShippingNum,String arrival, String vehicleNum, String monitor，String transfer, ArrayList<String> orderNum) | 新建一个装车单 | |
| Load.getTotal(String arrival) | 获得总运费 | |
| Load.endLoad() | 结束装车单的新建 | |
| Load.export(LoadorderPO) | 导出装车单 | |
|  |  | |

Load的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Load.addLoadOrder | 语法 | public ResultMessasge addLoadOrder(String data, String shop, String c  arShippingNum,String arrival, String vehicleNum, String monitor，String transfer, ArrayList<String> orderNum ) |
| 前置条件 | 启动一个车辆装车管理回合 |
| 后置条件 | 在一个车辆装车管理中，新建一张装车单 |
| Load.getTotal | 语法 | public double getTotal(String arrival) |
|  | 前置条件 | 装车单填写过程中填写完成到达地 |
|  | 后置条件 | 返回运费总价 |
| Load.endLoad | 语法 | public void endLoad() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次车辆装车管理，持久化更新涉及的对象数据 |
| Load.export | 语法 | public ResultMessage export(LoadorderPO po) |
|  | 前置条件 | 填写装车单完成且确认保存后 |
|  | 后置条件 | 导出excel文件 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| LoadDataService.add(LoadorderPO po) | 增加单一持久化对象 | |
| VehicleDataService.updata(String key) | 更新车辆信息持久化对象中的车辆状态及所在地 | |
| DriverDataService.updata(String key) | 更新司机信息持久化对象中的车辆状态及所在地 | |
| ListinDataService.updata(String key) | 更新订单物流状态 | |

###### 业务逻辑层的动态模型

###### 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。

### Vehiclebl模块

#### 4.1.5.1车辆信息管理

###### 模块描述

VehicleBl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。

VehicleBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.vehicleblservice.vehicleBlService接口。业务逻辑层和数据层之间添加dataservice.vehicledataservice.vehicleDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了VehicleController，将对新建车辆信息操作的业务逻辑委托给Vehicle 对象。VehiclePO是作为车辆信息的持久化被添加到设计模型里的。

VehicleBl模块的设计如图 所示：



VehicleBl模块各个类的职责如表 所示：

表5 VehicleBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| VehicleController | 负责实现新建车辆信息界面所需要的服务 |
| Vehicle | 新建车辆信息的领域模型对象，能够获得已有的车辆信息，帮助新建车辆信息界面完成所需要的任务 |

###### 模块内部类的接口规范

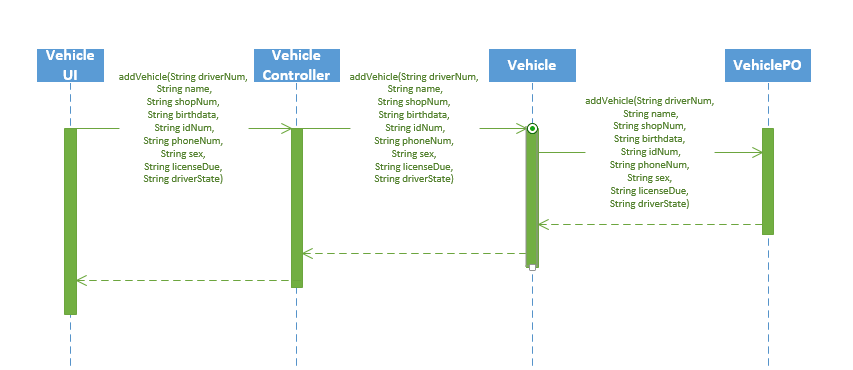
VehicleController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| VehicleController.addVehicle | 语法 | public ResultMessasge addVehicle (String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState) |
| 前置条件 | 已建立Vehicle领域对象，输入符合输入规则 |
| 后置条件 | 调用Vehicle领域对象的addVehicle方法 |
| VehicleController.updateVehicle | 语法 | public ResultMessage updataVehicle(String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState)) |
|  | 前置条件 | 已建立Vehicle领域对象，输入符合输入规则 |
|  | 后置条件 | 调用Vehicle领域对象updateVehicle方法 |
| VehicleController.deleteVehicle | 语法 | public ResultMessage deleteVehicle(String carNum) |
|  | 前置条件 | 已建立Vehicle领域对象，选择删除该车辆信息 |
|  | 后置条件 | 调用Vehicle领域对象的deleteVehicle方法 |
| VehicleController.find | 语法 | public ResultMessage find(String carNum) |
|  | 前置条件 | 已建立Vehicle领域对象，输入符合输入规则 |
|  | 后置条件 | 调用Vehicle领域模型的find方法 |
| VehicleController.end | 语法 | public void end() |
|  | 前置条件 | 已建立Vehicle领域对象，已确认输入 |
|  | 后置条件 | 调用Vehicle领域对象的end方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Vehicle.addVehicle(String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState) | 增加一个车辆信息 | |
| Vehicle.updateVehicle(String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState) | 更新一个车辆信息 | |
| Vehicle.deleteVehicle(String carNum) | 删除一个车辆信息 | |
| Vehicle.find(String carNum) | 查找一个车辆信息 | |
| Vehicle.end() | 结束车辆信息管理 | |

Vehicle的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Vehicle.addVehicle | 语法 | public ResultMessasge addVehicle (String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState) |
| 前置条件 | 启动一个车辆信息管理回合 |
| 后置条件 | 在一个车辆信息管理回合中，新建一个车辆信息 |
| Vehicle.updateVehicle | 语法 | public ResultMessage updataVehicle(String carNum, String engineNum, String car, String baseNum, String buyData, String useData, String carState)) |
|  | 前置条件 | 启动一个车辆信息管理回合 |
|  | 后置条件 | 在一个车辆信息管理回合中，更改一个车辆信息 |
| Vehicle.deleteVehicle | 语法 | public ResultMessage deleteVehicle(String carNum) |
|  | 前置条件 | 启动一个车辆信息管理回合 |
|  | 后置条件 | 在一个车辆信息管理回合中，删除一个车辆信息 |
| Vehicle.find | 语法 | public ResultMessage find(String carNum) |
|  | 前置条件 | 启动一个车辆信息管理回合 |
|  | 后置条件 | 在一个车辆信息管理回合中，查找一个车辆信息 |
| Vehicle.end | 语法 | public void end() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次车辆信息管理，持久化更新涉及的对象数据 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| VehicleDataService.find(String carNum) | 根据车辆代号查找单一持久化对象 | |
| VehicleDataService.insert(VehiclePO po) | 插入单一持久化对象 | |
| VehicleDataService.delete(VehiclePO po) | 删除单一持久化对象 | |
| VehicleDataService.updata(VehiclePO po) | 更新单一持久化对象 | |

###### 业务逻辑层的动态模型



###### 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。

#### 4.1.5.2司机信息管理

###### 模块描述

DriverBl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。

DriverBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.vehicleblservice.driverBlService接口。业务逻辑层和数据层之间添加dataservice.vehicledataservice.driverDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了DriverController，将对新建司机信息操作的业务逻辑委托给Driver 对象。DriverPO是作为司机信息的持久化被添加到设计模型里的。

DriverBl模块的设计如图 所示：



DriverBl模块各个类的职责如表 所示：

表5 DriverBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| DriverController | 负责实现新建司机信息界面所需要的服务 |
| Driver | 新建司机信息的领域模型对象，能够获得已有的司机信息，帮助新建司机信息界面完成所需要的任务 |

###### 模块内部类的接口规范

DriverController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| DriverController.addDriver | 语法 | public ResultMessasge addDriver (String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) |
| 前置条件 | 已建立Driver领域模型，输入符合输入规则 |
| 后置条件 | 调用Driver领域模型的addDriver方法 |
| DriverController.updateDriver | 语法 | public ResultMessage updataDriver(String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) |
|  | 前置条件 | 已建立Driver领域模型，输入符合输入规则 |
|  | 后置条件 | 调用Driver领域模型updateDriver方法 |
| DriverController.deleteDriver | 语法 | public ResultMessage deleteDriver(String name) |
|  | 前置条件 | 已建立Driver领域模型，选择删除该司机信息 |
|  | 后置条件 | 调用Driver领域模型的deleteDriver方法 |
| DriverController.find | 语法 | public ResultMessage find(String name) |
|  | 前置条件 | 已建立Driver领域模型，输入符合输入规则 |
|  | 后置条件 | 调用Driver领域模型的find方法 |
| DriverController.end | 语法 | public void end() |
|  | 前置条件 | 已建立Driver领域模型，已确认输入 |
|  | 后置条件 | 调用Driver领域模型的end方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Driver.addDriver(String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) | 增加一个司机信息 | |
| Driver.updateDriver(String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) | 更新一个司机信息 | |
| Driver.deleteDriver(String name) | 删除一个司机信息 | |
| Driver.find(String name) | 查找一个司机信息 | |
| Driver.end() | 结束司机信息管理 | |
|  |  | |

Driver的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Driver.addDriver | 语法 | public ResultMessasge addDriver (String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) |
| 前置条件 | 启动一个司机信息管理回合 |
| 后置条件 | 在一个司机信息管理回合中，新建一个司机信息 |
| Driver.updataDriver | 语法 | public ResultMessage updataDriver(String driverNum, String name, String shopNum, String birthdata, String idNum, String phoneNum, String sex, String licenseDue,String driverState) |
|  | 前置条件 | 启动一个司机信息管理回合 |
|  | 后置条件 | 在一个司机信息管理回合中，更改一个司机信息 |
| Driver.deleteDriver | 语法 | public ResultMessage deleteDriver(String name) |
|  | 前置条件 | 启动一个司机信息管理回合 |
|  | 后置条件 | 在一个司机信息管理回合中，删除一个司机信息 |
| Driver.find | 语法 | public ResultMessage find(String name) |
|  | 前置条件 | 启动一个司机信息管理回合 |
|  | 后置条件 | 在一个司机信息管理回合中，查找一个司机信息 |
| Driver.end | 语法 | public void end() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次司机信息管理，持久化更新涉及的对象数据 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| DriverDataService.find(String name) | 根据姓名查找单一持久化对象 | |
| DriverDataService.insert(DriverPO po) | 插入单一持久化对象 | |
| DriverDataService.delete(DriverPO po) | 删除单一持久化对象 | |
| DriverDataService.update(DriverPO po) | 更新单一持久化对象 | |

###### 业务逻辑层的动态模型

###### 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。

### Financebl模块

#### 4.1.6.1成本管理

###### 模块描述

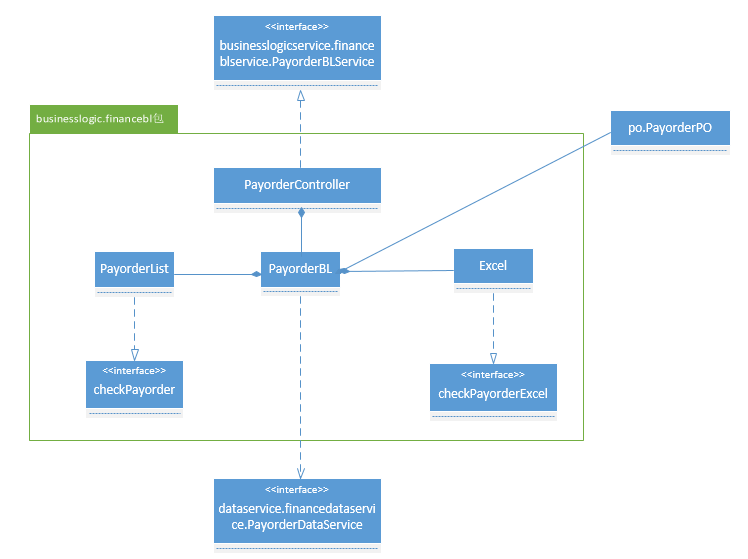
成本管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

成本管理模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.financeblservice.PayorderBLService接口。业务逻辑层和数据层之间添加dataservice.payorderdateservice.PayorderDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PayorderController，这样PayorderController会将对成本管理的业务逻辑处理委托给PayorderBL对象。PayorderPO是作为付款单的持久化对象被添加到设计模型中去的。PayorderList的添加是PayorderList的容器类，保有所有付款单的信息，及相应的查看付款单信息的职责。Excel的添加是checkPayorderExcel的容器类，保有成本收益表的信息，及相应的查看成本收益表的职责。checkPayorder和checkPayorderExcel是根据依赖倒置原则，为了消除循环依赖而产生的接口。

入库管理模块的设计如图：



成本管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| PayorderController | 负责实现成本管理界面所需要的服务。 |
| PayorderBL | 具有查看、增加付款单等功能，能够帮助完成成本管理界面所需要的服务。 |
| Excel | 成本收益表的领域模型对象，拥有截至当前日期的成本收益情况表，可以解决查看及导出成本收益表问题 |
| PayorderList | 付款单的领域模型对象，拥有所有付款单的信息，可以帮助完成查看付款单所需要的服务 |

###### 模块内部类的接口规范

**PayorderController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| PayorderController.new | 语法 | public ResultMessage new(int paymoney,String data,String payname,String payaccount,String list,String comment); |
| 前置条件 | 启动一个新建付款单的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| PayorderController.check | 语法 | public ResultMessage check(); |
| 前置条件 | 启动一个查看付款单的任务 |
| 后置条件 | 显示所有的付款信息 |
| PayorderController.profit | 语法 | public double profit(doubl[]e pay,double[] payee); |
| 前置条件 | 已计算出总收入、总支出和总利润 |
| 后置条件 | 显示总收入、总支出和总利润 |
| PayorderController.excel | 语法 | public void excel(String data,double income,double outcome,double benefit); |
| 前置条件 | 已启动查看付款信息任务，并创建成本收益表 |
| 后置条件 | 将成本收益表导出到电脑 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Payorder.new | 新建一个付款单对象 | |
| Payorder.check | 查看所有付款单信息 | |
| Pay order.profit | 生成截至当前日期的成本收益表 | |
| Payorder.excel | 导出成本收益表 | |

**PayorderBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Payorder.new | 语法 | public ResultMessage new(int paymoney,String data,String payname,String payaccount,String list,String comment); |
| 前置条件 | 启动一个新建付款单的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| Payorder.check | 语法 | public ResultMessage check(); |
| 前置条件 | 启动一个查看付款单的任务 |
| 后置条件 | 显示所有的付款信息 |
| Payorder.profit | 语法 | public double profit(doubl[]e pay,double[] payee); |
| 前置条件 | 已计算出总收入、总支出和总利润 |
| 后置条件 | 显示总收入、总支出和总利润 |
| Payorder.excel | 语法 | public void excel(String data,double income,double outcome,double benefit); |
| 前置条件 | 已启动查看付款信息任务，并创建成本收益表 |
| 后置条件 | 将成本收益表导出到电脑 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| PayorderDataService.insert(PayorderPO po) | 插入单一持久化对象 | |
| PayorderDataService.finds() | 查找涉及的领域的所有持久化对象 | |
| FinancebaseFactory.gePayorderDatabase | 得到Payorder数据库的服务的引用 | |

###### 业务逻辑层的动态模型

PayorderBL模块的顺序图表明了当用户新建、查看付款单，及导出成本收益表时，成本管理业务逻辑处理的相关对象之间的协作。

如下图所示：



PayorderBL模块的顺序图

新建付款单的状态图如下：



addPayorder的状态图

###### 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。

#### 4.1.6.2结算管理

###### 模块描述

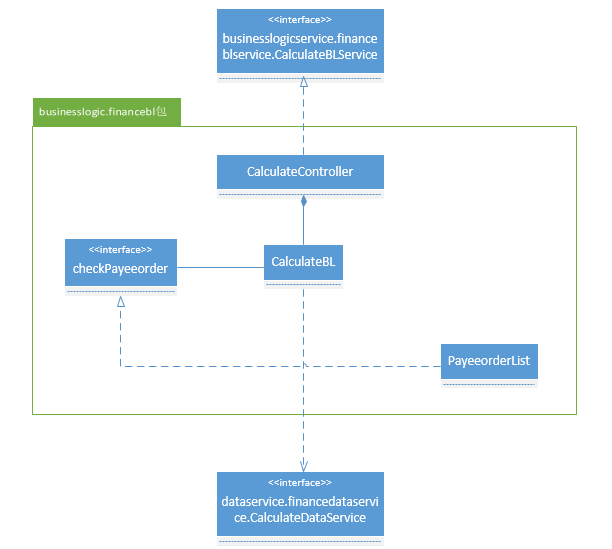
结算管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

结算管理模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.financeblservice.CalculateBLService接口。业务逻辑层和数据层之间添加dataservice.financedateservice.CalculateDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CalculateController，这样CalculateController会将对结算管理的业务逻辑处理委托给CalculateBL对象。checkPayeeorder的添加是PayoeerderList的容器类，保有所有收款单的信息，及相应的查看收款单信息的职责。checkPayeeorder是根据依赖倒置原则，为了消除循环依赖而产生的接口。

结算管理模块的设计如图：



结算管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| CalculateController | 负责实现结算管理界面所需要的服务。 |
| CalculateBL | 具有查看收款单等功能，能够帮助完成结算管理界面所需要的服务。 |
| PayeeorderList | 收款单的领域模型对象，拥有所有收款单的信息，可以帮助完成查看收款单所需要的服务 |

###### 模块内部类的接口规范

**CalculateController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| CalculateController.check | 语法 | public ResultMessage check(String data,String shop); |
| 前置条件 | 启动一个查看结算信息任务，并选定日期与营业厅 |
| 后置条件 | 显示选定日期选定营业厅的收款信息 |
| CalculateController.total | 语法 | public double total(double[] payee); |
| 前置条件 | 已启动查看结算信息任务，并已计算总收款金额 |
| 后置条件 | 显示总收款金额 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Calculate.check | 查看结算信息 | |
| Calculate.total | 合计收款金额 | |

**Calculate的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Calculate.check | 语法 | public ResultMessage check(String data,String shop); |
| 前置条件 | 启动一个查看结算信息任务，并选定日期与营业厅 |
| 后置条件 | 显示选定日期选定营业厅的收款信息 |
| Calculate.total | 语法 | public double total(double[] payee); |
| 前置条件 | 已启动查看结算信息任务，并已计算总收款金额 |
| 后置条件 | 显示总收款金额 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| DatabaseFactory.getCalculateDatabase | 得到Calculatee数据库的服务的引用 | |
| CalculateDataService.finds(String data,String shop) | 根据字段名和值进行查找多个持久化对象 | |

###### 业务逻辑层的动态模型

CalculateBL模块的顺序图表明了当用户查看结算信息时，结算管理业务逻辑处理的相关对象之间的协作。

如下图所示：



CalculateBL模块的顺序图

###### 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。

#### 4.1.6.3统计报表

###### 模块描述

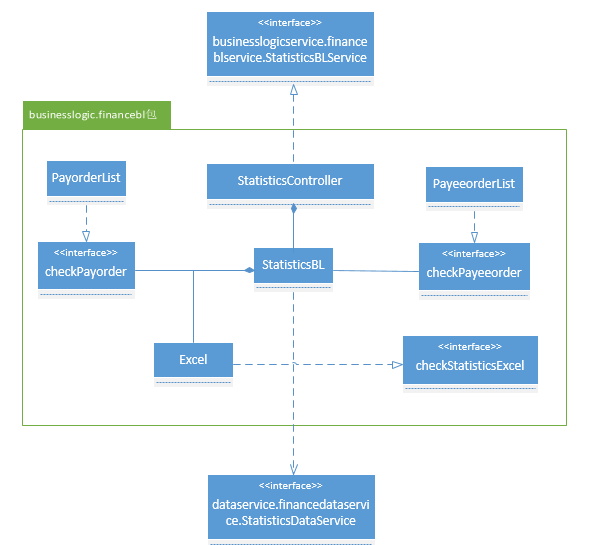
统计报表模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

统计报表模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.financeblservice.StatisticsBLService接口。业务逻辑层和数据层之间添加dataservice.financedateservice.StatisticsDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了StatisticsController，这样StatisticsController会将对统计报表的业务逻辑处理委托给StatisticsBL对象。checkPayeeorder的添加是PayoeerderList的容器类，保有所有收款单的信息，及相应的查看收款单信息的职责。checkPayorder的添加是PayorderList的容器类，保有所有付款单的信息，及相应的查看付款单信息的职责。Excel的添加是checkStatisticsExcel的容器类，保有某段日期内所有的收款单信息和付款单信息，及相应的查看统计报表的职责。checkPayeeorder、checkPayeeorder和checkStatisticsExcel是根据依赖倒置原则，为了消除循环依赖而产生的接口。

结算管理模块的设计如图：



统计报表模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| StatisticsController | 负责实现统计报表界面所需要的服务。 |
| StatisticsBL | 具有查看某段时期内经营情况等功能，能够帮助完成统计报表界面所需要的服务。 |
| PayeeorderList | 收款单的领域模型对象，拥有所有收款单的信息，可以帮助完成查看收款单所需要的服务 |
| PayorderList | 付款单的领域模型对象，拥有所有付款单的信息，可以帮助完成查看付款单所需要的服务 |
| Excel | 经营情况表的领域模型对象，拥有某段时期内的经营情况表，可以解决查看及导出经营情况表问题 |

###### 模块内部类的接口规范

**StatisticsController用例的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| StatisticsController.check | 语法 | public ResultMessage check(String start,String end); |
| 前置条件 | 启动一个查看统计报表任务，并输入起始和结束日期 |
| 后置条件 | 显示选定日期内所有的入款单和收款单信息 |
| StatisticsController.excel | 语法 | public ResultMessage excel(String start,String end); |
| 前置条件 | 已启动查看统计报表任务，并已创建经营情况统计表 |
| 后置条件 | 将经营情况表导出到电脑 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Statistics.check | 查看经营情况 | |
| Statistics.excel | 导出经营情况表 | |

**Statistics的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Statistics.check | 语法 | public ResultMessage check(String start,String end); |
| 前置条件 | 启动一个查看统计报表任务，并输入起始和结束日期 |
| 后置条件 | 显示选定日期内所有的入款单和收款单信息 |
| Statistics.excel | 语法 | public ResultMessage excel(String start,String end); |
| 前置条件 | 已启动查看统计报表任务，并已创建经营情况统计表 |
| 后置条件 | 将经营情况表导出到电脑 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| DatabaseFactory.ge StatisticsDatabase | 得到Statistics数据库的服务的引用 | |
| StatisticsDataService.finds(String start,String end) | 根据字段名和值进行查找多个持久化对象 | |

###### 业务逻辑层的动态模型

StatisticsBL模块的顺序图表明了查看经营情况，及导出某段时期内经营情况表时，统计报表业务逻辑处理的相关对象之间的协作。

如下图所示：



###### 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。

#### 4.1.6.4收款单管理

###### 模块描述

ReceiFormBl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。

ReceiFormBl模块的职责及接口参见软件系统结构描述文档表3。

###### 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层和数据层。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.financeblservice.ReceiFormBlService接口。业务逻辑层和数据层之间添加dataservice.financedataservice.ReceiFormDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiFormController，将对新建收款单操作的业务逻辑委托给ReceiForm 对象。PayeeorderPO是作为收款单信息的持久化被添加到设计模型里的。OrderList和OrderItem的添加是ListinInfo的容器类。ListinInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口

ReceiFormBl模块的设计如图 所示：



ReceiFormBl模块各个类的职责如表 所示：

表5 ReceiFormBl

|  |  |
| --- | --- |
| 模块 | 职责 |
| ReceiFormController | 负责实现新建收款单界面所需要的服务 |
| ReceiForm | 新建收款单的领域模型对象，能够获得已有的收款单信息，帮助新建收款单信息界面完成所需要的任务 |
| OrderList | 存放OrderItem，计算订单总金额 |
| OrderItem | 保有订单数据 |

###### 模块内部类的接口规范

ReceiFormController 的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReceiFormController.addReceiForm | 语法 | public ResultMessasge addReceiveOrder (String data, ArrayList<Integer> price, ArrayList<String> orderNum, String carrierName, String recorder ) |
| 前置条件 | 已创建ReceiForm领域对象 |
| 后置条件 | 调用ReceiForm领域对象的addReceiForm方法 |
| ReceiFormController.getTotal | 语法 | public double getTotal(ArrayList<String> orderNum) |
|  | 前置条件 | 已创建ReceiForm领域对象，获取订单完成 |
|  | 后置条件 | 调用ReceiForm领域对象的getTotal方法 |
| ReceiFormController.endReceiForm | 语法 | public void endReceiForm() |
|  | 前置条件 | 已创建ReceiForm领域对象，已确认信息 |
|  | 后置条件 | 调用ReceiForm领域对象的endReceiForm方法 |
| ReceiFormController.exportPayeeorder | 语法 | public ResultMessage exportPayeeorder (PayeeorderPO po) |
|  | 前置条件 | 已创建ReceiForm领域对象，保存完成 |
|  | 后置条件 | 调用ReceiForm领域对象的exportPayeeorder方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ReceiForm.addReceiForm(String data, ArrayList<Integer> price, ArrayList<String> orderNum, String carrierName, String recorder) | 新增一个收款单 | |
| ReceiForm.getTotal(ArrayList<String> orderNum) | 获得总价格 | |
| ReceiForm.endReceiForm() | 结束管理收款单 | |
| ReceiForm.exportPayeeorder(PayeeorderPO po) | 导出收款单 | |

ReceiForm的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReceiForm.addReceiForm | 语法 | public ResultMessasge addReceiveOrder (String data, ArrayList<Integer> price, ArrayList<String> orderNum, String carrierName, String recorder ) |
| 前置条件 | 启动一个管理收款单回合 |
| 后置条件 | 在一个管理收款单回合中，新建一张收款单 |
| ReceiForm.getTotal | 语法 | public double getTotal(ArrayList<String> orderNum) |
|  | 前置条件 | 收款单填写过程中填写完成订单 |
|  | 后置条件 | 返回运费总价 |
| ReceiForm.endReceiForm | 语法 | public void endReceiForm() |
|  | 前置条件 | 已确认信息 |
|  | 后置条件 | 结束此次收款单管理，持久化更新涉及的对象数据 |
| ReceiForm.exportPayeeorder | 语法 | public ResultMessage exportPayeeorder (PayeeorderPO po) |
|  | 前置条件 | 填写管理收款单完成且确认保存后 |
|  | 后置条件 | 导出excel文件 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ReceiFormDataService.insert(PayeeorderPO po) | 增加单一持久化对象 | |

###### 业务逻辑层的动态模型

###### 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。

### Storebl模块

#### 4.1.7.1入库管理

###### 模块描述

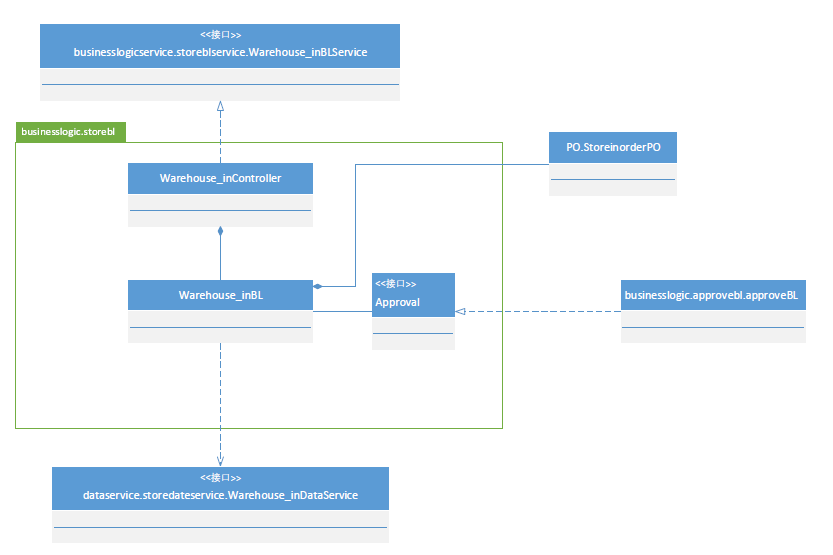
入库管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

入库管理模块的职责及接口参见软件系统结构描述文档。

###### 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。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.storeblservice.Warehouse\_inBLService接口。业务逻辑层和数据层之间添加dataservice.storedateservice.Warehouse\_inDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了Warehouse\_inController，这样Warehouse\_inController会将对入库管理的业务逻辑处理委托给Warehouse\_inBL对象。StoreinorderPO是作为入库单的持久化对象被添加到设计模型中去的。Approval是为了将单据提交给总经理审批而产生的接口。

入库管理模块的设计如图：



入库管理模块各个类的设计

入库管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| Warehouse\_inController | 负责实现入库管理界面所需要的服务。 |
| Warehouse\_inBL | 具有新建入库单、设置消息提醒、查看消息提醒、存储新建的入库单等功能，能够帮助完成入库管理界面所需要的服务。 |

###### 模块内部类的接口规范

**Warehouse\_inController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse-in.build | 语法 | public void build(ArriverorderVO av,int jia,int wei) |
| 前置条件 | 已创建一个Warehouse\_inBL领域对象，并输入符合输入规则 |
| 后置条件 | 调用Warehouse\_inBL领域对象的build方法 |
| Warehouse-in.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 已创建一个Warehouse\_inBL领域对象， |
| 后置条件 | 调用Warehouse\_inBL领域对象的checkRemind方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Warehouse-in.build | 创建一张入库单 | |
| Warehouse-in.checkRemind | 查看消息提醒 | |

**Warehouse\_inBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse-in.build | 语法 | public void build(ArriverorderVO av,int jia,int wei) |
| 前置条件 | 启动一个入库管理回合 |
| 后置条件 | 系统将入库单提交总经理审批 |
| Warehouse-in.setRemind | 语法 | public void setRemind(ArriverorderVO av) |
| 前置条件 | 货物录入到达单 |
| 后置条件 | 增加一条消息提示 |
| Warehouse-in.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 启动一个入库管理回合 |
| 后置条件 | 删除该条消息 |
| Warehouse-in.saveOrder | 语法 | public void save(StoreinorderVO sv) |
| 前置条件 | 入库单通过总经理审批 |
| 后置条件 | 将入库单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Approval.checkOrder(PO) | 提交给总经理审核单据 | |
| Warehouse-in DataService.insert(StoreinorderPO sp) | 插入一个持久化对象StoreinorderPO | |

###### 业务逻辑层的动态模型

顺序图参照4.1.4.2装车管理

###### 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。

#### 4.1.7.2出库管理

###### 模块描述

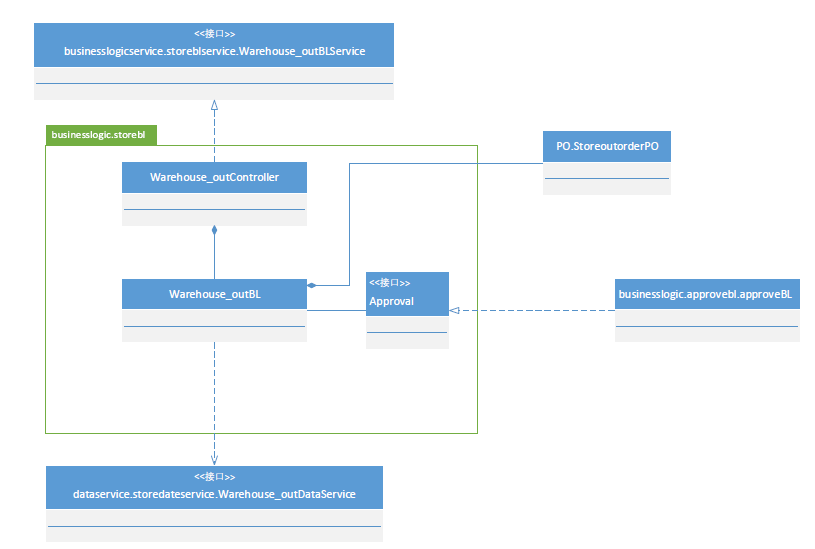
出库管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

出库管理模块的职责及接口参见软件系统结构描述文档。

###### 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。每一层之间为增加灵活性，我们会添加接口。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.storeblservice.Warehouse\_outBLService接口。业务逻辑层和数据层之间添加dataservice.storedateservice.Warehouse\_outDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了Warehouse\_outController，这样Warehouse\_outController会将对出库管理的业务逻辑处理委托给Warehouse\_outBL对象。StoreoutorderPO是作为出库单的持久化对象被添加到设计模型中去的。Approval是为了将单据提交给总经理审批而产生的接口。

出库管理模块的设计如图：



出库管理模块各个类的设计

入库管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| Warehouse\_outController | 负责实现出库管理界面所需要的服务。 |
| Warehouse\_outBL | 具有新建出库单、设置消息提醒、查看消息提醒、存储新建的出库单等功能，能够帮助完成出库管理界面所需要的服务。 |

###### 模块内部类的接口规范

**Warehouse\_outController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse-out.build | 语法 | public void build(ChangeorderVO,String number) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象，并输入符合输入规则 |
| 后置条件 | 调用Warehouse\_outBL领域对象的build方法 |
| Warehouse-out.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象， |
| 后置条件 | 调用Warehouse\_outBL领域对象的checkRemind方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Warehouse-out.build | 创建一张入库单 | |
| Warehouse-out.checkRemind | 查看消息提醒 | |

**Warehouse\_outBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse-out.build | 语法 | public void build(ChangeorderVO,String number) |
| 前置条件 | 启动一个出库管理回合 |
| 后置条件 | 系统将出库单提交总经理审批 |
| Warehouse-out.setRemind | 语法 | public void setRemind(ChangeorderVO cv) |
| 前置条件 | 货物录入中转单 |
| 后置条件 | 增加一条消息提示 |
| Warehouse-out.checkRemind | 语法 | public ResultMessage checkRemind() |
| 前置条件 | 启动一个出库管理回合 |
| 后置条件 | 删除该条消息 |
| Warehouse-out.saveOrder | 语法 | public void save(StoreoutorderVO sv) |
| 前置条件 | 出库单通过总经理审批 |
| 后置条件 | 将出库单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Approval.checkOrder(PO) | 提交给总经理审核单据 | |
| Warehouse-out rDataService.insert(StoreoutorderPO sp) | 插入一个持久化对象StoreoutorderPO | |

###### 业务逻辑层的动态模型

顺序图参照4.1.4.2装车管理

###### 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。

#### 4.1.7.3库存管理

###### 模块描述

库存管理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。

库存管理模块的职责及接口参见软件系统结构描述文档。

###### 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。每一层之间为增加灵活性，我们会添加接口。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.storeblservice.Inventory\_managementBLService接口。业务逻辑层和数据层之间添加dataservice.storedateservice. Inventory\_managemenDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了Inventory\_managemenController，这样Inventory\_managemenController会将对库存管理的业务逻辑处理委托给Inventory\_managemenBL对象。StorePO是作为库存信息的持久化对象被添加到设计模型中去的。

库存管理模块的设计如图：



库存管理模块各个类的设计

入库管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| Inventory\_managemenController | 负责实现库存管理界面所需要的服务。 |
| Inventory\_managemenBL | 具有查看库存、盘点库存、设置警戒值、设置提醒值等功能，能够帮助完成库存管理界面所需要的服务。 |

###### 模块内部类的接口规范

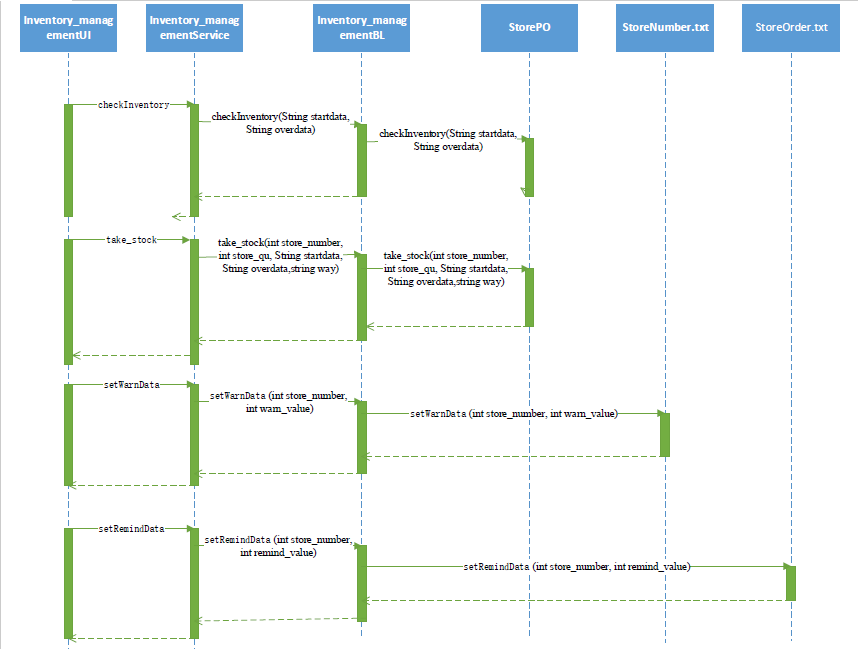
**Inventory\_managemenController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Inventory\_management.checkInventory\_now | 语法 | public ResultMessage checkInventory(int store\_qu) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象 |
| 后置条件 | 调用Inventory\_managemenBL 领域的 |
| Inventory\_management.checkInventory | 语法 | public ResultMessage checkInventory(Data startdata,Data overdata) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象，且输入时间段符合规范 |
| 后置条件 | 调用Inventory\_managemenBL 领域的checkInventory方法 |
| Inventory\_management.take\_stock | 语法 | public void take\_stock (int store\_number,int store\_qu, Data startdata,Data overdata,string way) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象，且库区，时间段和查看方式符合规范 |
| 后置条件 | 调用Inventory\_managemenBL 领域的take\_stock方法 |
| Inventory\_management.setWarnData | 语法 | public void setWarnData (int store\_number, int warn\_value) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象，且输入警戒值 |
| 后置条件 | 调用Inventory\_managemenBL 领域的setWarnData方法 |
| Inventory\_management.setRemindData | 语法 | public void setRemindData (int store\_number, int remind\_value) |
| 前置条件 | 已创建一个Warehouse\_outBL领域对象且输入提醒值 |
| 后置条件 | 调用Inventory\_managemenBL 领域的setRemindData方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Inventory\_management.checkInventory\_now | 查看当前各个库区的存储状态 | |
| Inventory\_management.checkInventory | 显示指定时间段各个库区出入库数量和金额 | |
| Inventory\_management.take\_stock | 显示或输出指定库区在指定时间段所有出入库的订单信息 | |
| Inventory\_management.setWarnData | 设置仓库警戒值比例 | |
| Inventory\_management.setRemindData | 设置不同库区提醒值比例 | |

**Inventory\_managemenBL的接口规范**

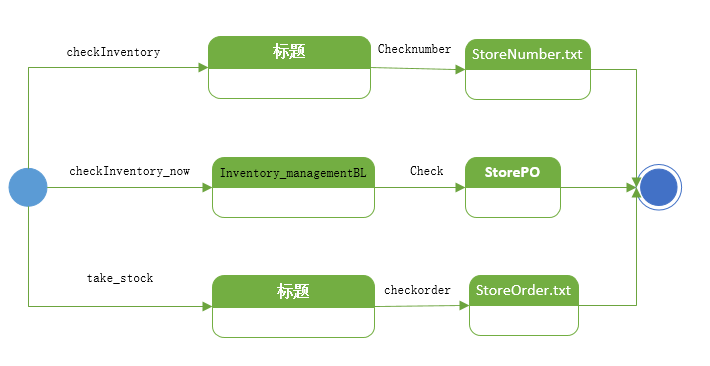
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Inventory\_management.checkInventory\_now | 语法 | public ResultMessage checkInventory(int store\_qu) |
| 前置条件 | 中转中心业务员启动一个录入中转单回合 |
| 后置条件 | 系统显示各库区库存量 |
| Inventory\_management.checkInventory | 语法 | public ResultMessage checkInventory(Data startdata,Data overdata) |
| 前置条件 | 启动一个查看库存回合 |
| 后置条件 | 系统显示各个库区出入库数量和金额 |
| Inventory\_management.take\_stock | 语法 | public void take\_stock(int store\_number,int store\_qu, Data startdata,Data overdata,string way) |
| 前置条件 | 启动一个盘点库存回合 |
| 后置条件 | 根据查看方式，显示或输出制定库区所有出入库的订单信息 |
| Inventory\_management.setWarnData | 语法 | public void setWarnData (int store\_number, int warn\_value) |
| 前置条件 | 启动一个管理警戒值回合 |
| 后置条件 | 系统更新警戒值 |
| Inventory\_management.setRemindData | 语法 | public void setRemindData (int store\_number, int remind\_value) |
| 前置条件 | 启动一个管理提醒值回合 |
| 后置条件 | 系统更新提醒值 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Inventory\_managementDataService.check(int store\_number) | 查看当前中转中心最新的StorePO | |
| Inventory\_managementDataService.  updata(StorePO sp，int store\_number) | 更新当前中转中心StorePO | |
| Inventory\_managementDataService.  checknumber(int store\_number, Data startdata,Data overdata) | 查看StoreNumber.txt | |
| Inventory\_managementDataService.  addnumber(int store\_number,Data,int storein\_number,int storeout\_number,int storein\_money,int storeout\_money) | 更新StoreNumber.txt | |
| Inventory\_managementDataService.  checkorder(int store\_number,int store\_qu, Data startdata,Data overdata) | 查看StoreOrder.txt | |
| Inventory\_managementDataService.  addorder(int store\_number,OrderPO op) | 更新StoreOrder.txt | |

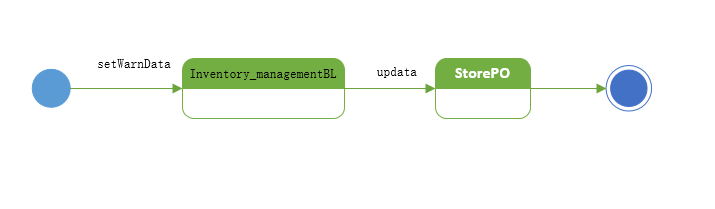
###### 业务逻辑层的动态模型



###### 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。





### Billbl模块

#### 4.1.8.1期初建账

###### 模块描述

期初建账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。

期初建账模块的职责及接口参见软件系统结构描述文档。

###### 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。每一层之间为增加灵活性，我们会添加接口。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.accountblservice.BillBLService接口。业务逻辑层和数据层之间添加dataservice.billdateservice. BillDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了BillController，这样BillController会将对期初建账的业务逻辑处理委托给BillBL对象。StartinfoPO是作为期初信息的持久化对象被添加到设计模型中去的。BillList保有所有期初信息，及相应的查看期初信息的职责。

库存管理模块的设计如图：



期初建账模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| BillController | 负责实现期初建账界面所需要的服务。 |
| BillBL | 具有查看、增加期初账本等功能，能够帮助完成期初建账界面所需要的服务。 |
| BillList | 期初信息的领域模型对象，拥有所有期初账本的信息，可以帮助完成查看期初信息所需要的服务 |

###### 模块内部类的接口规范

**BillController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| BillController.new | 语法 | public ResultMessage new(String institution,String staff,String vehicle,String store,String account,String data,String creator); |
| 前置条件 | 启动一个期初建账任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| BillController.check | 语法 | public ResultMessage check(String billname); |
| 前置条件 | 启动一个查看期初信息的任务 |
| 后置条件 | 显示选定的帐的期初信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Bill.new | 新建一个期初账本对象 | |
| Bill.check | 查看所有期初账本信息 | |

**Bill的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Bill.new | 语法 | public ResultMessage new(String institution,String staff,String vehicle,String store,String account,String data,String creator); |
| 前置条件 | 启动一个期初建账任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| Bill.check | 语法 | public ResultMessage check(String billname); |
| 前置条件 | 启动一个查看期初信息的任务 |
| 后置条件 | 显示选定的帐的期初信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| StartinfoDataService.insert(StartinfoPO po) | 插入单一持久化对象 | |
| StartinfoDataService.finds(String billname) | 根据账名（billname）查找单一持久化对象 | |
| DatabaseFactory.getStartinfoDatabase | 得到Startinfo数据库的服务的引用 | |

###### 业务逻辑层的动态模型

BillBL模块的顺序图表明了当用户新建期初账本、查看期初信息时，期初建账业务逻辑处理的相关对象之间的协作。

如下图所示：



BillBL模块的顺序图

新建期初账本的状态图如下：



addBill的状态图

###### 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。

### Accountbl模块

#### 4.1.9.1账户管理

###### 模块描述

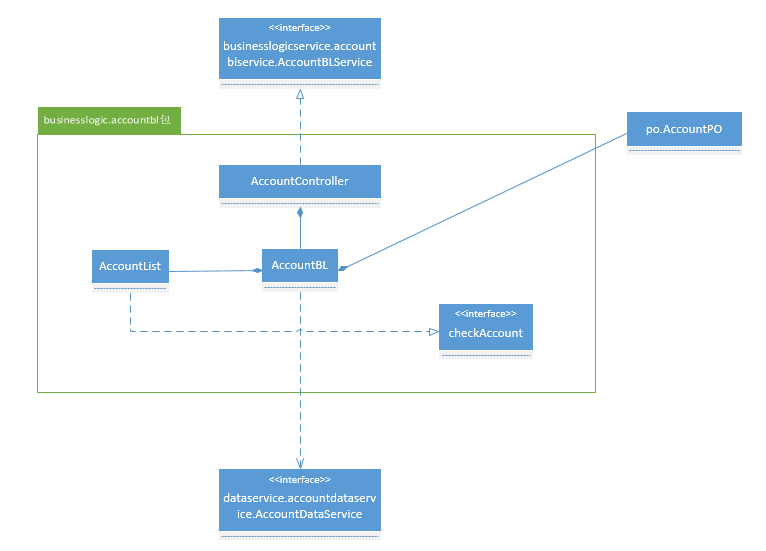
账户管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

账户管理模块的职责及接口参见软件系统结构描述文档。

###### 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。每一层之间为增加灵活性，我们会添加接口。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.accountblservice.AccountBLService接口。业务逻辑层和数据层之间添加dataservice.accountdateservice. AccountDataService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会将对账户管理的业务逻辑处理委托给AccountBL对象。AccountPO是作为账户信息的持久化对象被添加到设计模型中去的。AccountList的添加是checkAccount的容器类，保有所有账户的信息，及相应的查看账户信息的职责。checkAccount是根据依赖倒置原则，为了消除循环依赖而产生的接口。

库存管理模块的设计如图：



账户管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountController | 负责实现账户管理界面所需要的服务。 |
| AccountBL | 具有查看、增加、删除及修改账户等功能，能够帮助完成账户管理界面所需要的服务。 |
| AccountList | 银行账户的领域模型对象，拥有所有银行账户的信息，可以帮助完成查看银行账户所需要的服务 |

###### 模块内部类的接口规范

**AccountController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| AccountController.new | 语法 | public ResultMessage new(String name,double pay,double payee,String creator); |
| 前置条件 | 启动一个新建账户任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| AccountController.updata | 语法 | public ResultMessage updata(String oldname,String newname); |
| 前置条件 | 已选定目标修改的账户，并启动修改账户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| AccountController.delete | 语法 | public ResultMessage delete(String aimname); |
| 前置条件 | 已选定目标删除的账户，且账户余额为0，并启动删除账户的任务 |
| 后置条件 | 删除涉及的领域对象的数据 |
| AccountController.search | 语法 | public ResultMessage search(String key); |
| 前置条件 | 已输入关键字，并启动查询账户的任务 |
| 后置条件 | 查找是否存在与输入的关键字相关的账户，显示可能的账户的信息或提示输入更准确的关键字 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Account.new | 新建一个账户对象 | |
| Account.updata | 修改一个账户对象 | |
| Account.delete | 删除一个账户对象 | |
| Account.search | 根据账户名查找一个账户对象 | |

**Account的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Account.new | 语法 | public ResultMessage new(String name,double pay,double payee,String creator); |
| 前置条件 | 启动一个新建账户任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| Account.updata | 语法 | public ResultMessage updata(String oldname,String newname); |
| 前置条件 | 已选定目标修改的账户，并启动修改账户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| Account.delete | 语法 | public ResultMessage delete(String aimname); |
| 前置条件 | 已选定目标删除的账户，且账户余额为0，并启动删除账户的任务 |
| 后置条件 | 删除涉及的领域对象的数据 |
| Account.search | 语法 | public ResultMessage search(String key); |
| 前置条件 | 已输入关键字，并启动查询账户的任务 |
| 后置条件 | 查找是否存在与输入的关键字相关的账户，显示可能的账户的信息或提示输入更准确的关键字 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| AccountDataService.insert(AccountPO po) | 插入单一持久化对象 | |
| AccountDataService.updata(AccountPO po) | 更新单一持久化对象 | |
| AccountDataService.delete(AccountPO po) | 删除单一持久化对象 | |
| AccountDataService.find(String name) | 根据账户名（name）查找单一持久化对象 | |
| AccountDataService.finds(String key) | 根据字段名和值进行查找多个持久化对象 | |
| DatabaseFactory.getAccountDatabase | 得到Account数据库的服务的引用 | |

###### 业务逻辑层的动态模型

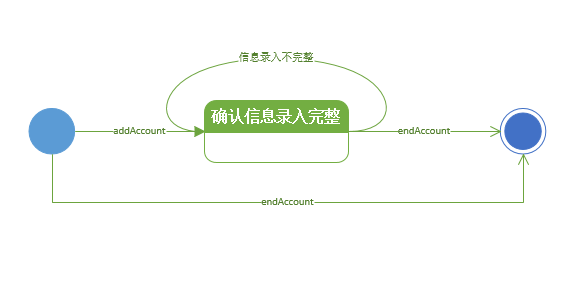
AccountBL模块的顺序图表明了当用户新建、查看、删除账户，及修改账户属性时，账户管理业务逻辑处理的相关对象之间的协作。

如下图所示：



AccountBL模块的顺序图

新建账户的状态图如下：



addAccount的状态图

###### 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。

### Staffbl模块

#### 4.1.10.1具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

#### 4.1.10.2用户管理

###### (一) 模块描述

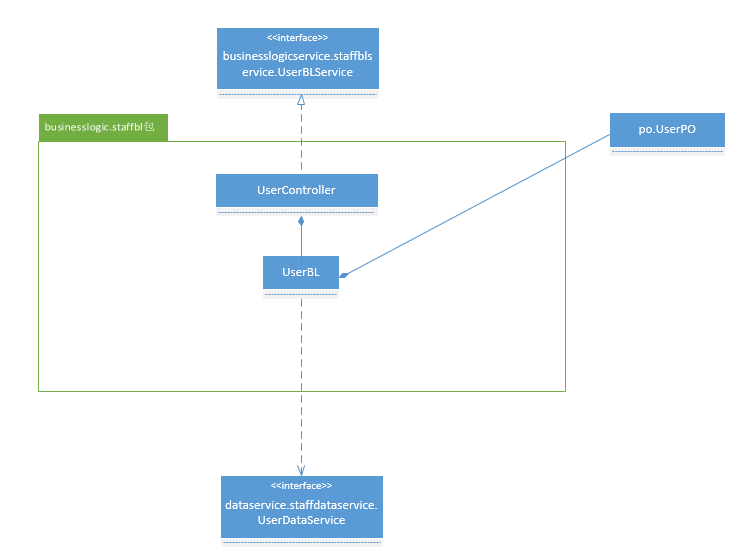
用户管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

用户管理模块的职责及接口参见软件系统结构描述文档。

###### (二) 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。每一层之间为增加灵活性，我们会添加接口。比如逻辑层和业务逻辑层之间，我们会添加businesslogicservice.staffblservice.UserBLService接口。业务逻辑层和数据层之间添加dataservice.staffdateservice. UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将对用户管理的业务逻辑处理委托给UserBL对象。UserPO是作为用户信息的持久化对象被添加到设计模型中去的。

库存管理模块的设计如图：



用户管理模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务。 |
| UserBL | 具有查看、增加、删除及修改用户属性等功能，能够帮助完成用户管理界面所需要的服务。 |

###### (三) 模块内部类的接口规范

**UserController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| UserController.new | 语法 | public ResultMessage new(String name,String key,String limit); |
| 前置条件 | 启动一个新建用户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| UserController.updata | 语法 | public ResultMessage updata(String oldname,String newname,Sting key,String limit); |
| 前置条件 | 已选定目标修改的用户，并启动修改用户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| UserController.delete | 语法 | public ResultMessage delete(String name); |
| 前置条件 | 已选定目标删除的用户，并启动删除用户的任务 |
| 后置条件 | 删除涉及的领域对象的数据 |
| UserController.check | 语法 | public ResultMessage check(UserPO po); |
| 前置条件 | 启动一个查看用户信息的任务 |
| 后置条件 | 显示选定用户的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| User.new | 新建一个用户对象 | |
| User.updata | 修改一个用户对象 | |
| User.delete | 删除一个用户对象 | |
| User.check | 查看所有用户信息 | |

**User的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| User.new | 语法 | public ResultMessage new(String name,String key,String limit); |
| 前置条件 | 启动一个新建用户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| User.updata | 语法 | public ResultMessage updata(String oldname,String newname,Sting key,String limit); |
| 前置条件 | 已选定目标修改的用户，并启动修改用户的任务 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| User.delete | 语法 | public ResultMessage delete(String name); |
| 前置条件 | 已选定目标删除的用户，并启动删除用户的任务 |
| 后置条件 | 删除涉及的领域对象的数据 |
| User.check | 语法 | public ResultMessage check(UserPO po); |
| 前置条件 | 启动一个查看用户信息的任务 |
| 后置条件 | 显示选定用户的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserDataService.insert(UserPO po) | 插入单一持久化对象 | |
| UserDataService.updata(UserPO po) | 更新单一持久化对象 | |
| UserDataService.delete(UserPO po) | 删除单一持久化对象 | |
| UserDataService.find(String name) | 根据用户名（name）查找单一持久化对象 | |
| DatabaseFactory.getUserDatabase | 得到User数据库的服务的引用 | |

###### (四) 业务逻辑层的动态模型

UserBL模块的顺序图表明了当用户新建、查看、删除用户，及修改用户属性时，用户管理业务逻辑处理的相关对象之间的协作。

如下图所示：



UserBL模块的顺序图

新建用户的状态图如下：



addUser的状态图

###### (五) 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。

### Policybl模块

#### 4.1.11.1具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

#### 4.1.11.2具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

### Approvebl模块

#### 4.1.12.1具体用例

###### 模块描述

**自己补全**

###### 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。比如逻辑层和业务逻辑层之间，**自己补全**

###### 模块内部类的接口规范

**自己补全**

###### 业务逻辑层的动态模型

**自己补全**

###### 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。

# 依赖视角



图3 快递管理系统客户端开发包图



图4 快递管理系统服务器端开发包图