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。

 **订单查询用例各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| ChecklistControler | 负责实现订单查询界面所需要的服务 |
| Check | 订单查询的领域模型对象，可以帮助完成订单查询界面所需要的服务 |

###### 模块内部类的接口规范

ChecklistController接口规范

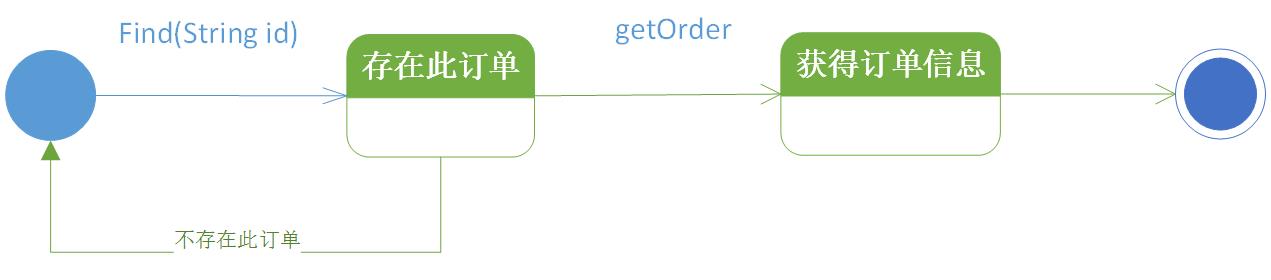
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| ChecklistController.getOrder | | 语法 | public OrderVO getOrder(String id) |
| 前置条件 | 已创建一个Check对象，并且输入符合输入规则 |
| 后置条件 | 调用Check领域对象的getOrder方法 |
| **需要的服务（需接口）** | | | |
| 服务名 | 服务 | | |
| Check.getOrder(String id) | 根据ID进行查找单一持久化对象 | | |

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Check.getOrder | 语法 | public OrderVO 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 | |
| Listinbl.getOrder (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



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。

### Listinbl模块

#### 4.1.2.1listin用例模块

###### 模块描述

Listin用例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，

Listin用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice.ListinService接口。业务逻辑层和数据层之间添加dataservice.listindataservice.ListinDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ListinController，这样ListinController会将订单输入的业务逻辑委托给listin对象。OrderPO是作为创建新订单持久化对象被添加到设计模型中去的。



**ListinBL用例模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Listinbl | 订单输入的领域模型对象，可以帮助完成订单输入界面所需要的服务 |

###### 模块内部类的接口规范

Listinbl的接口规范如下

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Listin.addOrder | 语法 | public void addOrder(OrderVO vo) |
| 前置条件 | 启动一个新建订单任务 |
| 后置条件 | 界面显示一份新建的订单 |
| Listin.getTotalMoney | 语法 | Public ResultMessage getTotalMoney（String address1, String address2, String weight,String length, String width, String height, ExpressType express,PackageType pack） |
| 前置条件 | 地址，体积，快递种类和包装种类的选择，均符合输入规范 |
| 后置条件 | 返回此订单输入中需要支付的金额 |
| Listin. getTotalTime | 语法 | Public ResultMessage getTotalTime（String address1，String address2， int Express） |
| 前置条件 | 地址，快递种类的选择，均符合输入规范 |
| 后置条件 | 返回此次货运预计到达的时间 |
| Listin. getOrder | 语法 | public OrderVO getOrder(String id) |
| 前置条件 | id有效 |
| 后置条件 | 根据订单号返回一个订单的持久化对象 |
| Listin. update | 语法 | public void update(String id, String message) |
| 前置条件 | id有效 |
| 后置条件 | 根据订单号更新对应的订单的信息 |
| Listin. getWeight | 语法 | public double getWeight(String id) |
| 前置条件 | id有效 |
| 后置条件 | 根据订单号得到对应订单的种类 |
| Listin. JudgeNull | 语法 | public boolean JudgeNull(OrderVO vo) |
| 前置条件 | 界面层点击确认新建订单 |
| 后置条件 | 返回是否有信息未填写 |
| Listin. getAllOrders | 语法 | public ArrayList<OrderPO> getAllOrders() |
| 前置条件 | 审批单据 |
| 后置条件 | 返回所有未审批的订单 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ListinDataService.insert(OrderPO po) | 插入单一持久化对象 | |
| ListinDataService. update(String id, String message) | 更新订单物流信息 | |
| ListinDataService. getOrder(String id) | 根据订单号查找单一持久化对象 | |
| ListinDataService. getWeight(String id) | 根据订单号得到订单的重量 | |
| ListinDataServcie. getAllOrders() | 返回所有未审批的订单 | |
| ConstantPolicyDataServcie. GetDistance(address1, address2) | 得到两地之间的距离 | |

###### 业务逻辑层的动态模型

下图表明了快递物流管理系统中，当用户输入地址，快递种类，包装种类，实际体积和重量后，自动计算获得付款额



下图表名了快递物流管理系统中，当用户输入地址，快递种类后，自动获得并计算时间



下图表明了快递物流管理系统中，当用户输入所有信息并点击确认后，订单输入业务逻辑相关对象之间的协作。



下图为状态图，描述了Listin对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作，随着getTotalTime的调用，Listin进入显示预计时间状态，之后getTotalMoney进入可以确认订单状态。



###### 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。

#### 4.1.2.2receiver用例模块

###### 模块描述

Receiver用例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，

Receiver用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice.ReceiverService接口。业务逻辑层和数据层之间添加dataservice.listindataservice.ReceiverDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiverController，这样ReceiverController会将收件人信息输入的业务逻辑委托给Receiver对象。ReceiverPO是作为创建新订单持久化对象被添加到设计模型中去的。



**ReceiverBL用例模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Receiverinbl | 收件人信息输入的领域模型对象，可以帮助完成收件人信息输入界面所需要的服务 |

###### 模块内部类的接口规范

Receiver的接口规范如下

|  |  |  |
| --- | --- | --- |
| Receiverinbl. addReceiver | 语法 | public void addReceiver(ReceiverVO vo) |
| 前置条件 | 已创建一个Receiverinbl对象，且输入符合输入规范 |
| 后置条件 | 调用Receiver领域对象的addReceiver方法 |
| Receiverinbl. searchOrder | 语法 | public boolean searchOrder(String id) |
| 前置条件 | Id输入符合规范 |
| 后置条件 | 返回是否存在本条订单 |
| Receiverinbl. JudgeNull | 语法 | public boolean JudgeNull(ReceiverVO vo) |
| 前置条件 | 界面层点击确定按钮 |
| 后置条件 | 返回界面层的订单输入是否完善 |
| **需要的服务（需接口）** | | |
| ReceiverDataService. insertReceiver(ReceiverPO po) | 插入单一持久化对象 | |
| ReceiverDataService.find(String number) | 根据订单号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。

### Transferbl模块

#### 4.1.3.1中转接收

###### 模块描述

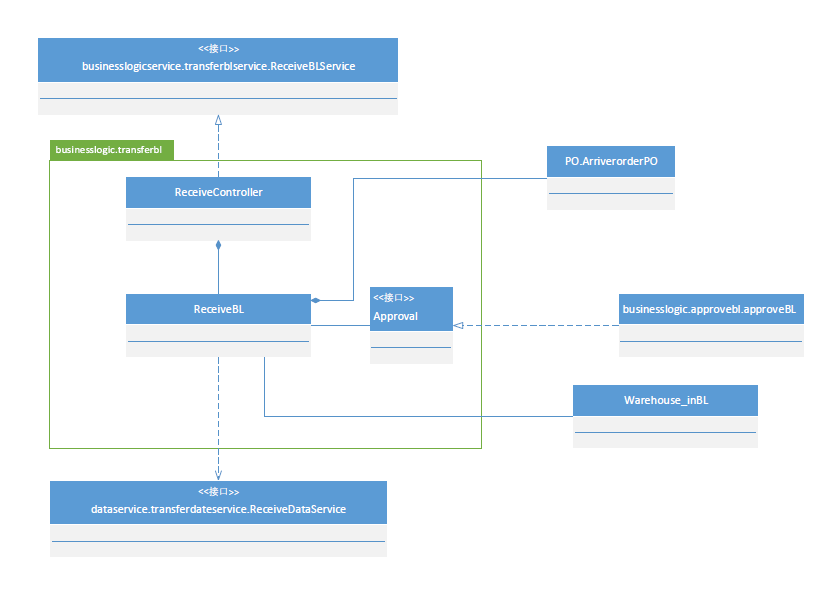
中转接收模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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 |
| 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

**ReceiveBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Receive.build | 语法 | public void build(ArriverorderVO av) |
| 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. get | 语法 | public ArrayList<ArriverorderPO> get() |
| 前置条件 | 总经理审批到达单 |
| 后置条件 | 返回所有未审批的到达单 |
| Receive. save | 语法 | public void save(ArriverorderPO PO) |
| 前置条件 | 总经理审批通过 |
| 后置条件 | 存储审批过的到达单 |
| Receive. checkUnreceive\_loadorderPO | 语法 | public LoadorderVO checkUnreceive\_loadorderPO(String s) |
| 前置条件 | 界面层查看消息提醒 |
| 后置条件 | 将对应消息提醒删除 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ReceiveDataService.insert(ArriverorderPO ap) | 插入一个持久化对象ArriverorderPO | |
| ReceiveDataService. checkUnreceive\_loadorderPO(String s) | 查看未接收的装运单 | |
| ReceiveDataService. get() | 得到所有未审批的到达单 | |

###### 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

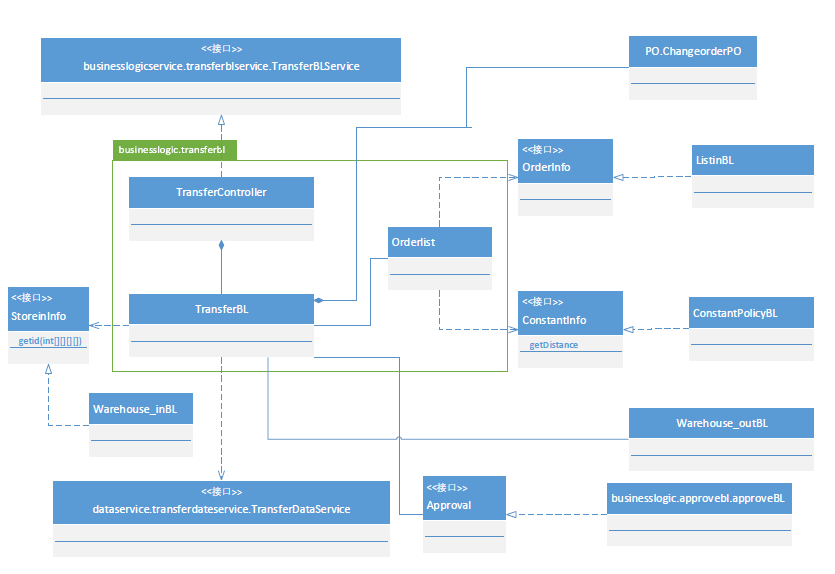
录入中转单模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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

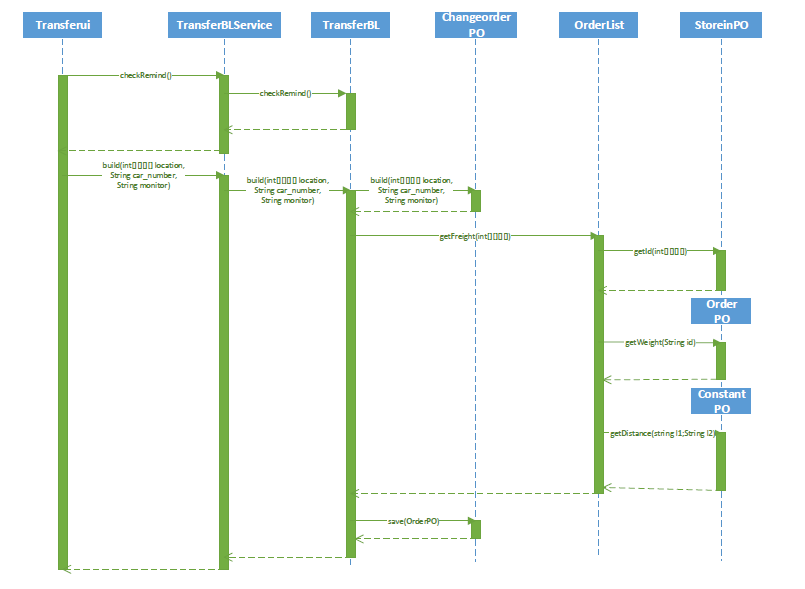
|  |  |
| --- | --- |
| 模块 | 职责 |
| TransferBL | 具有新建中转单、设置消息提醒、查看消息提醒、查看库存信息、存储新建的中转单等功能，能够帮助完成录入中转单界面所需要的服务。 |

###### 模块内部类的接口规范

**TransferBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Transfer.build | 语法 | public void build(ChangeorderVO cv) |
| 前置条件 | 启动一个新建中转单回合 |
| 后置条件 | 系统将新建的中转单提交给总经理审批 |
| Transfer. checkStore | 语法 | public String[][] checkStore() |
| 前置条件 | 查看库存 |
| 后置条件 | 返回各个库区库存信息 |
| Transfer. checkRemind | 语法 | public String checkRemind() |
| 前置条件 | 界面层查看消息提醒 |
| 后置条件 | 无 |
| Transfer. get | 语法 | public ArrayList<ChangeorderPO> get() |
| 前置条件 | 总经理审批中转单 |
| 后置条件 | 返回所有未审批的中转单 |
| Transfer. save | 语法 | public void save(ChangeorderPO cp) |
| 前置条件 | 中转单已通过总经理审核 |
| 后置条件 | 将中转单存储 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| TransferDataService. get() | 得到所有未审批的中转单 | |
| StoreinInfo. getStoreRatio() | 查看此时中转中心仓库各个库区比例 | |
| TransferDataService.insert(ChangeorderPO cp) | 插入一个持久化对象ChangeorderPO | |

###### 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。

#### 4.1.3.3录入接收单

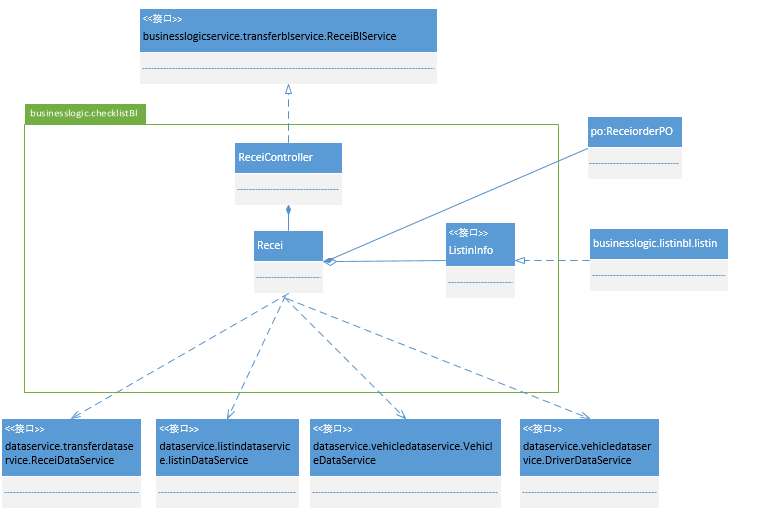
###### 模块描述

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 |
| ReceiFormBL | 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

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReceiFormBL.addReceiveOrder | 语法 | public void addReceiveOrder(ReceiFormVO rv) |
| 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 |
| ReceiFormBL. get | 语法 | public ArrayList<ReceiveorderPO> get() |
|  | 前置条件 | 总经理审批接收单 |
|  | 后置条件 | 返回所有未审批的接收单 |
| ReceiFormBL. save | 语法 | public void save(ReceiveorderPO rp) |
|  | 前置条件 | 总经理审批通过接收单 |
|  | 后置条件 | 存储审批过后的接收单 |
| **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 | |
| transportUpdateInfo.updata(String carNum) | 更新车辆信息持久化对象中的车辆状态及所在地 | |
| ReceiDataService.get() | 得到未审批的接收单 | |
| UpdateInfo.updata(String id,String Meessage) | 更新订单物流状态 | |

###### 业务逻辑层的动态模型



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。

#### 4.1.3.4录入派件单

###### 模块描述

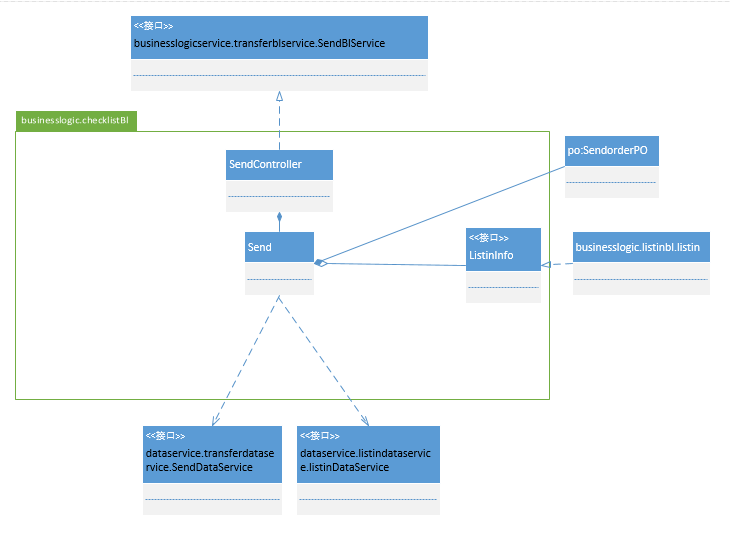
SendBl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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 |
| 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

Send接口规范

|  |  |  |
| --- | --- | --- |
| **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



状态图类似新建接收单

###### 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 |
| 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

**ShippingBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Shipping.build | 语法 | public void build(VehicleLoadorderVO vv) |
| 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. getTotal | 语法 | public double getTotal(ArrayList<String> orderlist, String offNum,  String arriveNum, String wayOfTransfer) |
| 前置条件 | 已创建一个ShippingBL领域对象 |
| 后置条件 | 调用ShippingBL领域的getTotal方法 |
| Shipping.checkRemind | 语法 | public ArrayList<ChangeorderVO> 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 |
| Shipping. get | 语法 | public ArrayList<LoadorderPO> get() |
| 前置条件 | 已创建一个ShippingBL领域对象 |
| 后置条件 | 调用ShippingBL领域的get方法 |
| Shipping. save | 语法 | public void save(LoadorderPO lp) |
| 前置条件 | 已创建一个ShippingBL领域对象 |
| 后置条件 | 调用ShippingBL领域的save方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| OrderInfo. GetDistance（String1，String2） | 达到两个地点之间的距离 | |
| ShippingDataService.insert(LoadorderPO lp) | 插入一条LoadorderPO | |
| ShippingDataService. checkUnshippingChangeorder() | 查看未装运的中转单 | |
| ShippingDataService. deleteChangeorder(String numOfTransfer) | 删除对应中转单 | |
| ShippingDataService. get() | 得到所有未审批的装运单 | |

###### 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

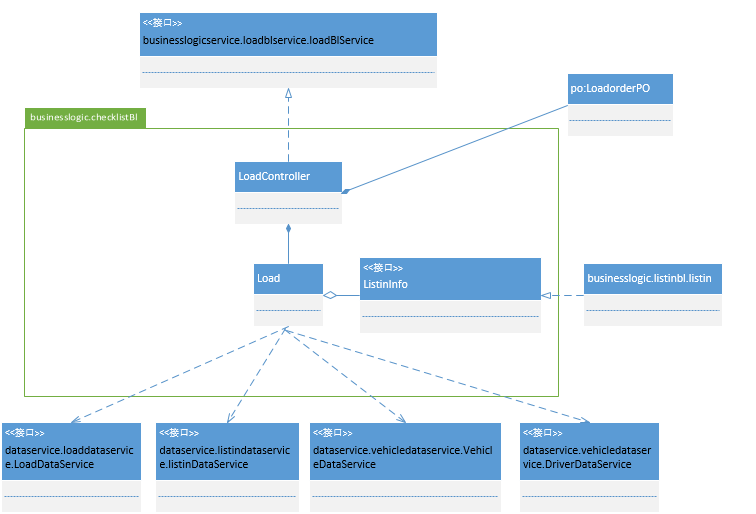
LoadBl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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 |
| LoadBL | 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

Load的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| LoadBL. addLoadOrder | 语法 | public void addLoadOrder(VehicleLoadorderVO vlv) |
| 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 |
| LoadBL. getTotal | 语法 | public double getTotal(ArrayList<String> orderlist, String offNum,  String arriveNum, String wayOfTransfer) |
|  | 前置条件 | 已创建一个ShippingBL领域对象 |
|  | 后置条件 | 调用ShippingBL领域的getTotal方法 |
| LoadBL. get | 语法 | public ArrayList<LoadorderPO> get() |
|  | 前置条件 | 已创建一个ShippingBL领域对象 |
|  | 后置条件 | 调用ShippingBL领域的get方法 |
| LoadBL. save | 语法 | public void save(LoadorderPO lp) |
| 前置条件 | 已创建一个ShippingBL领域对象 |
| 后置条件 | 调用ShippingBL领域的save方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| OrderInfo. GetDistance（String1，String2） | 达到两个地点之间的距离 | |
| ShippingDataService.insert(LoadorderPO lp) | 插入一条LoadorderPO | |
| ShippingDataService. checkUnshippingChangeorder() | 查看未装运的中转单 | |
| ShippingDataService. deleteChangeorder(String numOfTransfer) | 删除对应中转单 | |

###### 业务逻辑层的动态模型

###### 

状态图类似于新建接收单

###### 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

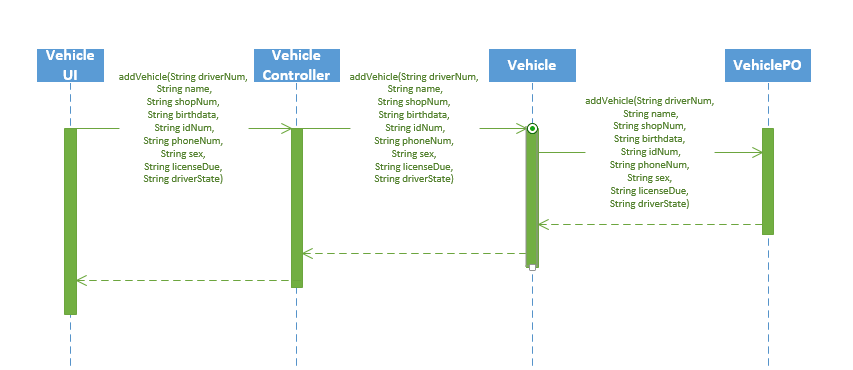
|  |  |
| --- | --- |
| 模块 | 职责 |
| VehicleController | 负责实现新建车辆信息界面所需要的服务 |
| Vehicle | 新建车辆信息的领域模型对象，能够获得已有的车辆信息，帮助新建车辆信息界面完成所需要的任务 |

###### 模块内部类的接口规范

VehicleController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Vehiclebl.addVehicle | 语法 | public void addVehicle(VehicleVO vo) |
| 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 |
| Vehiclebl.updateVehicle | 语法 | public void updateVehicle(VehicleVO vo) |
|  | 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 |
| Vehiclebl.deleteVehicle | 语法 | public void 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 |
| Vehiclebl. searchVehicle | 语法 | public boolean searchVehicle(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searchVehicle方法 |
| Vehiclebl. find | 语法 | public VehicleVO find(String carNum) |
|  | 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find方法 |
| Vehiclebl. update | 语法 | public boolean update(String carNum, TransportState state) |
|  | 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update方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| VehicleDateService. insert(VehiclePO po) | 增加一个车辆信息 | |
| VehicleDateService. update(VehiclePO po) | 更新一个车辆信息 | |
| VehicleDateService. find(String carNum) | 查找一个车辆信息 | |
| VehicleDateService. delete(VehiclePO po) | 删除一个车辆信息 | |
| VehicleDateService. isExist(String carNum) | 判断车辆是否存在 | |
| VehicleDateService.updateState(String carNum, TransportState state) | 更新车辆状态 | |

###### 业务逻辑层的动态模型



状态图类似新建接收单

###### 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

Driver的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Driver.addDriver | 语法 | public void addDriver(DriverVO vo) |
| 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 void updateDriver(DriverVO vo) |
|  | 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 void 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. searchDriver | 语法 | public boolean searchDriver(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，判断一个司机信息是否存在 |
| Driver. find | 语法 | public DriverVO 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. update | 语法 | public boolean update(String name, DriverState state) |
| 前置条件 | 车辆发出或到达 |
| 后置条件 | 更新司机的闲忙信息 |
| **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



状态图类似新建接收单

###### 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

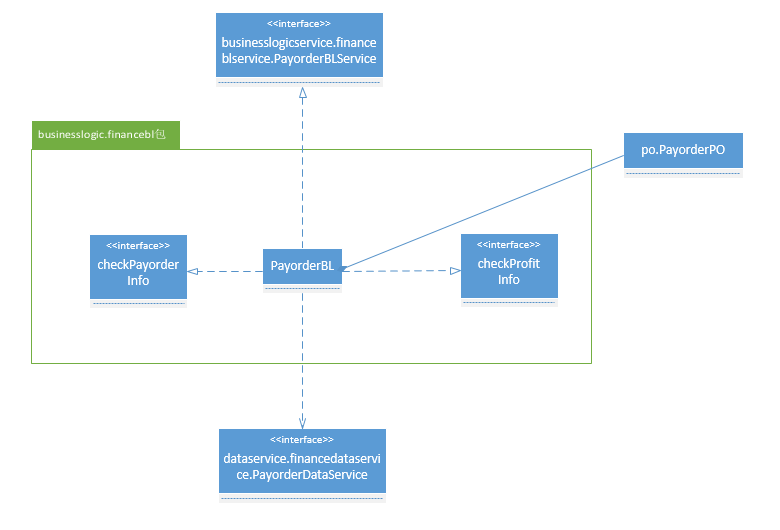
成本管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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。CheckPayorderInfo的添加是为了给其他需要查看付款单的模块提供查看的接口，具有查看付款单的功能。CheckProfitInfo的添加是为了给其他需要查看成本收益的模块提供查看的接口，具有查看成本收益的功能。checkPayorderInfo和CheckProfitInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

成本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 |
| 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。 |
| checkPayorderInfo | 用来提供给其他模块查看付款单的接口 |
| checkProfitInfo | 用来提供给其他模块查看成本收益的接口 |

###### 模块内部类的接口规范

**PayorderBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| PayorderBL. addPayorder | 语法 | public void addPayorder(double paymoney, String date, 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 |
| PayorderBL. checkPayorder | 语法 | public ArrayList<PayorderVO> checkPayorder(); |
| 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 |
| PayorderBL.profit | 语法 | public double[] profit(); |
| 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 |
| PayorderBL.excel | 语法 | public void excel(String date, 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 |
| PayorderBL.save | 语法 | public void save(PayorderPO po); |
| 前置条件 | 付款单已通过审批 |
| 后置条件 | 存储付款单为已审批状态 |
| **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

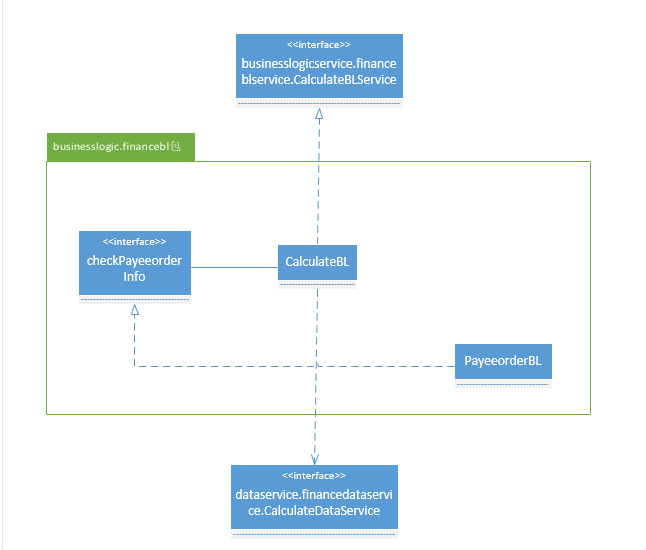
结算管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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，我们将对结算管理的业务逻辑处理委托给CalculateBL对象。checkPayeeorder的添加是Payoeerder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 |
| 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。 |
| checkPayeeorderInfo | 用来调用PayeeorderBL中查看收款单方法的接口，具有查看收款单的功能。 |

###### 模块内部类的接口规范

**Calculate的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| CalculateBL.check | 语法 | public ArrayList<PayeeorderVO> 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 |
| CalculateBL.total | 语法 | public double total(ArrayList<PayeeorderVO> volist); |
| 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

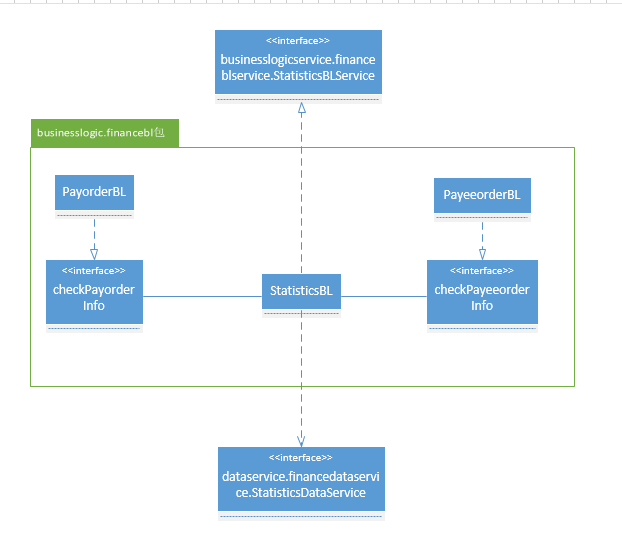
统计报表模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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。checkPayeeorder、checkPayeeorder、是根据依赖倒置原则，为了消除循环依赖而产生的接口。

统计报表模块的设计如图：



统计报表模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| 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。 |
| checkPayeeorderInfo | 用来调用PayeeorderBL中查看收款单方法的接口，具有查看收款单的功能。 |
| checkPayorderInfo | 用来调用PayorderBL中查看付款单方法的接口，具有查看付款单的功能。 |

###### 模块内部类的接口规范

**Statistics的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Statistics.check | 语法 | public ArrayList<PayorderVO> checkPay(int start, int end);  public ArrayList<PayeeorderVO> checkPayee(int start, int 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 void excel(ArrayList<PayorderVO> payVo,ArrayList<PayeeorderVO> payeeVo, String date); |
| 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 | |

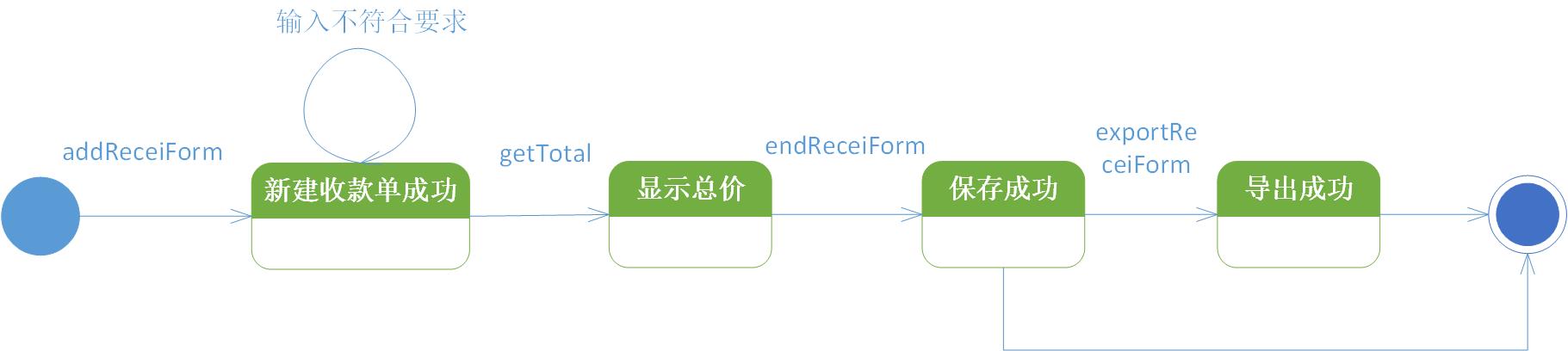
ReceiForm的接口规范

|  |  |  |
| --- | --- | --- |
| **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

顺序图类似于新建装车单

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。

### Storebl模块

#### 4.1.7.1入库管理

###### 模块描述

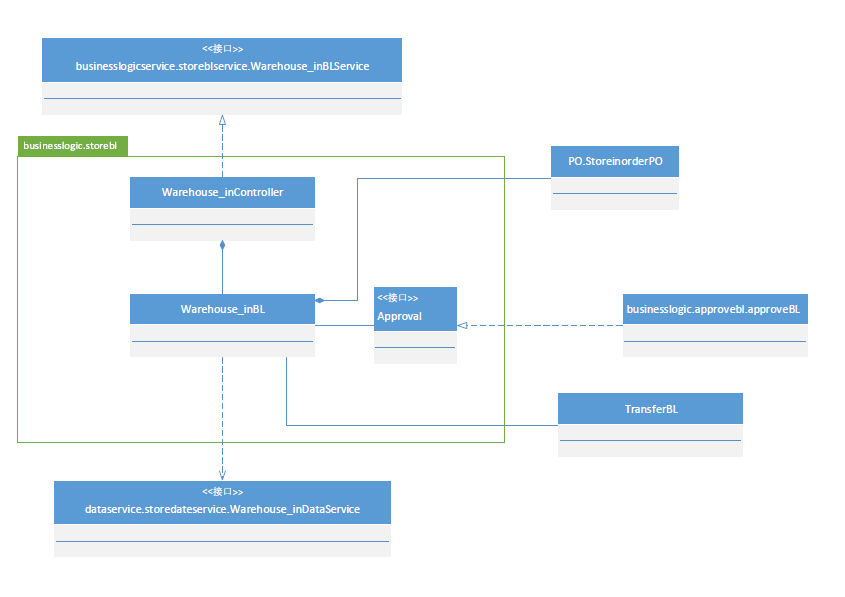
入库管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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\_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\_inBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse\_inBL.build | 语法 | public void build(StoreinVO sv) |
| 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\_inBL. checkRemind | 语法 | public String checkRemind() |
| 前置条件 | 启动一个入库管理回合 |
| 后置条件 | 查看未入库的到达单 |
| Warehouse\_inBL. deleteRemind | 语法 | public void deleteRemind() |
| 前置条件 | 启动一个入库管理回合 |
| 后置条件 | 删除消息提醒 |
| Warehouse\_inBL. get | 语法 | public ArrayList<StoreinorderPO> get() |
| 前置条件 | 总经理审批入库单 |
| 后置条件 | 返回所有未审批的入库单 |
| Warehouse\_inBL. save | 语法 | public void save(StoreinorderPO sp) |
| 前置条件 | 入库单通过总经理审批 |
| 后置条件 | 将入库单存储 |
| Warehouse\_inBL. getLocation | 语法 | public StoreLocationVO getLocation(String orderNum) |
| 前置条件 | 启动一个入库管理回合 |
| 后置条件 | 返回对应订单库存位置 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Warehouse\_inDataService. insert(StoreinorderPO sp) | 插入一个持久化对象StoreinorderPO | |
| Warehouse\_inDataService. checkUnstoreinArriveorder() | 查看未入库的到达单 | |
| Warehouse\_inDataService.deleteRemind() | 删除提醒 | |
| Warehouse\_inDataService. get() | 得到未审批的入库单 | |
| UpdateInfo.update | 更新订单物流信息 | |
| StoreinUpdateInfo.update | 更新库存信息 | |

###### 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

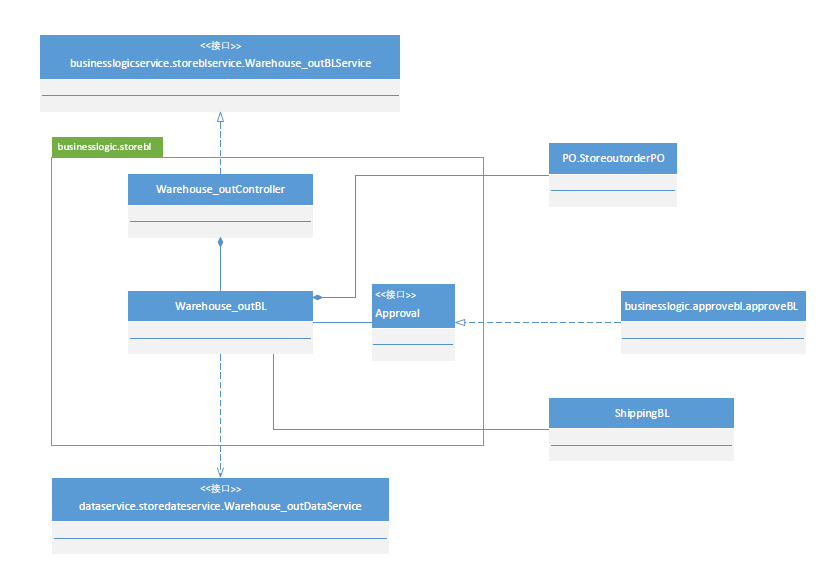
出库管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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\_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\_outBL的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Warehouse\_outBL.build | 语法 | public void build(ChangeorderVO cv, String date) |
| 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\_outBL. checkRemind | 语法 | public String checkRemind() |
| 前置条件 | 启动一个出库管理回合 |
| 后置条件 | 查看未出库的到达单 |
| Warehouse\_outBL. get | 语法 | public ArrayList<StoreoutorderPO> get() |
| 前置条件 | 总经理审批出库单 |
| 后置条件 | 返回所有未审批的出库单 |
| Warehouse\_outBL. save | 语法 | public void save(StoreoutorderPO sp) |
| 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 | |
| Warehouse\_outDataService. insert(StoreoutorderPO sp) | 插入一个持久化对象StoreoutorderPO | |
| Warehouse\_outDataService. checkUnstoreoutChangeorder() | 查看未出库的中转单 | |
| Warehouse\_outDataService. get() | 得到未审批的出库单 | |
| UpdateInfo.update | 更新订单物流信息 | |
| GetlocationInfo.getLocation(String orderNum) | 根据订单号得到对应的库存位置 | |
| StoreinUpdateInfo.storein\_update(String qu,String pai,String jia,String wei,String id) | 根据库存位置更新订单号 | |

###### 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

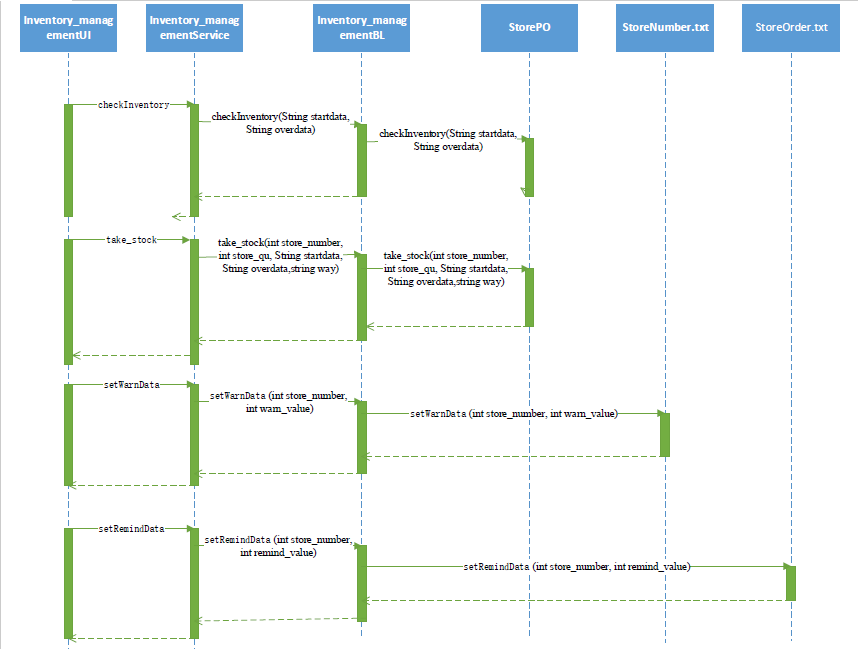
|  |  |
| --- | --- |
| 模块 | 职责 |
| Inventory\_managemenController | 负责实现库存管理界面所需要的服务。 |
| Inventory\_managemenBL | 具有查看库存、盘点库存、设置警戒值、设置提醒值等功能，能够帮助完成库存管理界面所需要的服务。 |

###### 模块内部类的接口规范

**Inventory\_managemenBL的接口规范**

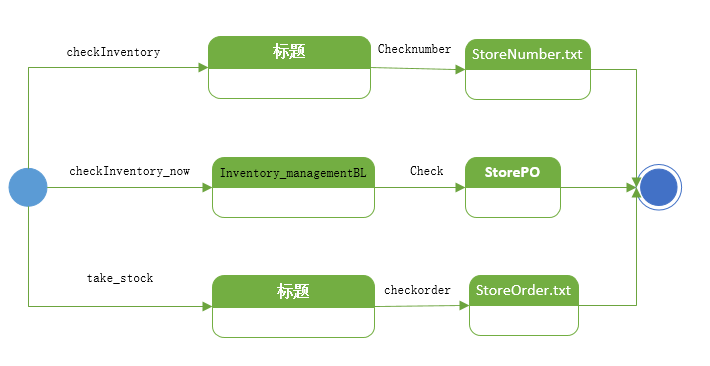
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Inventory\_management.checkInventory | 语法 | public String[][] checkInventory(String startdate, String overdate) |
| 前置条件 | 启动一个查看库存回合 |
| 后置条件 | 系统显示仓库在指定时间段内的出入库数量 |
| Inventory\_management. checkInventoryOrder | 语法 | public String[][] checkInventoryOrder(String qu) |
| 前置条件 | 启动一个库存盘点回合，且选择界面显示 |
| 后置条件 | 系统显示指定区域所有库存商品的订单信息 |
| Inventory\_management. InventoryOrderToExcel | 语法 | public void InventoryOrderToExcel(String qu) |
| 前置条件 | 启动一个库存盘点回合，且excel导出 |
| 后置条件 | 通过excel指定区域所有库存商品的订单信息 |
| Inventory\_management. getWarnData | 语法 | public String getWarnData() |
| 前置条件 | 启动一个管理警戒值回合 |
| 后置条件 | 返回当前警戒值 |
| Inventory\_management.setWarnData | 语法 | public void setWarnData(String warnData) |
| 前置条件 | 启动一个管理警戒值回合 |
| 后置条件 | 更新当前警戒值 |
| Inventory\_management. getRemindData |  | public String getRemindData(int area) |
|  | 启动一个管理提醒值回合 |
|  | 返回指定区域的提醒值 |
| Inventory\_management. setRemindData |  | public void setRemindData(int area, String RemindData) |
|  | 启动一个管理提醒值回合 |
|  | 更新指定区域的提醒值 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Inventory\_managementDataService.checkStoreinNumber(  String startdate,String overdate) | 查看指定时间段内各个区的入库数量 | |
| Inventory\_managementDataService.checkStoreoutNumber(  String startdate,String overdate) | 查看指定时间段内各个区的出库数量 | |
| Inventory\_managementInfo.getOrderList(String qu); | 得到指定区域当前在库中的所有订单号 | |
| Inventory\_managementDataService(ArrayList<String> orderlist) | 根据订单号列表得到订单信息列表 | |
| Inventory\_managementInfo.getWarnData() | 得到当前警戒值比例 | |
| Inventory\_managementInfo.setWarnData(String warnData) | 更新当前警戒值比例 | |
| Inventory\_managementInfo(int area) | 得到指定区域提醒值比例 | |
| Inventory\_managementInfo.setRemindData(int area,String RemindData) | 更新指定区域提醒值比例 | |

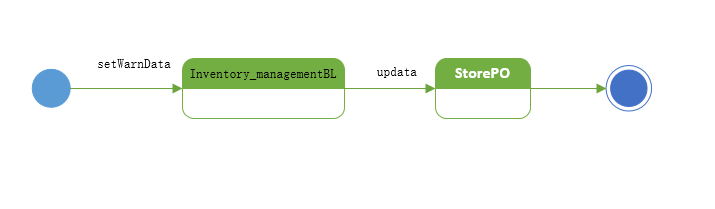
###### 业务逻辑层的动态模型



###### 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

**Bill的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| BillBL. addBill | 语法 | public void addBill(String name, String account, String organization,  String staff, String vehicle, String store); |
| 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 |
| BillBL.checkBill | 语法 | public ArrayList<StartinfoVO> checkBill(); |
| 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

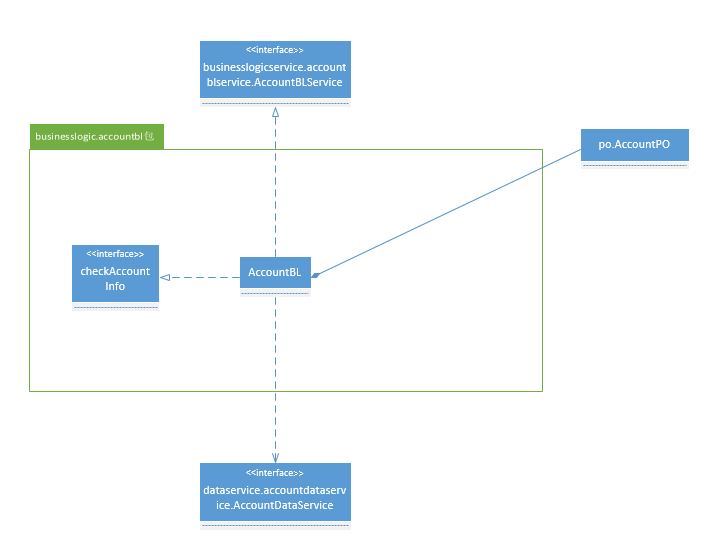
账户管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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。checkAccountInfo是用来为其他模块查看账户提供接口，具有查看账户的功能。checkAccountInfo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 |
| 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。 |
| checkAccountInfo | 用来为其他模块查看账户提供接口，具有查看账户功能 |

###### 模块内部类的接口规范

**Account的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Account.addAccount | 语法 | public void addAccount(String name, String money, String creator,  String date); |
| 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.updateAccount | 语法 | public ArrayList<AccountVO> updateAccount(int pos,String name, String creator, String date); |
| 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.deleteAccount | 语法 | public void deleteAccount(String name); |
| 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.check | 语法 | public ArrayList<AccountVO> check(); |
| 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

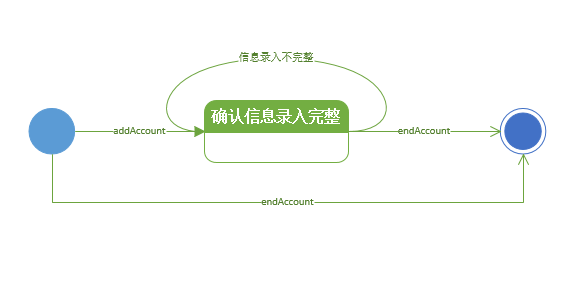
AccountBL模块的顺序图表明了当用户新建、查看、删除账户，及修改账户属性时，账户管理业务逻辑处理的相关对象之间的协作。

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

#### 4.1.10.1staffbl用例模块

###### 模块描述

Staffbl用例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，

Staffbl用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice.StaffBLService接口。业务逻辑层和数据层之间添加dataservice.StaffDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了staffblController，这样StaffController会将收件人信息输入的业务逻辑委托给staffbl对象。Stafflineitem持有staffPO的部分数据。Stafflist封装stafflineitem的数据集合的数据结构的秘密和相关操作。



**StaffBL用例模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Staffbl | 人员机构管理的领域模型对象，可以帮助完成人员机构管理界面所需要的服务 |

###### 模块内部类的接口规范

**表21 staffbl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Staffbl.showStaff | 语法 | public ArrayList<StaffVO> showStaff() |
| 前置条件 | 启动一个人员管理任务 |
| 后置条件 | 界面显示多份人员信息 |
| Staffbl.checkStaff | 语法 | public StaffVO checkStaff(String name) |
| 前置条件 | 点击人员信息的其中一项 |
| 后置条件 | 界面显示人员具体信息 |
| Staffbl.addStaff | 语法 | public void addStaff(StaffVO vo) |
| 前置条件 | 启动一个新建人员信息表任务 |
| 后置条件 | 界面显示新建的一份人员信息表 |
| Staffbl. getStaffPO | 语法 | public StaffPO getStaffPO(String name) |
| 前置条件 | 启动一个查询人员信息任务 |
| 后置条件 | 查找是否存在相应姓名的人员信息，返回查询验证的结果 |
| Staffbl.editStaff | 语法 | public void editStaff(StaffVO vo) |
| 前置条件 | 启动一个修改人员信息表任务 |
| 后置条件 | 持续化更新涉及的领域对象的数据 |
| Staffbl.deleteStaff | 语法 | public void deleteStaff(String id) |
| 前置条件 | 启动一个删除人员信息表任务 |
| 后置条件 | 持续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 | |
| StaffDataService.gets() | 获得多个持久化对象 | |
| StaffDataService.get(String name) | 获得单一持久化对象 | |
| StaffDataService.insert(StaffPO po) | 插入单一持久化对象 | |
| StaffDataService. delete(StaffPO po) | 删除单一持久化对象 | |
| StaffDataService. updata(StaffPO po) | 修改单一持久化对象 | |

###### 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。

#### 4.1.10.2用户管理

###### (一) 模块描述

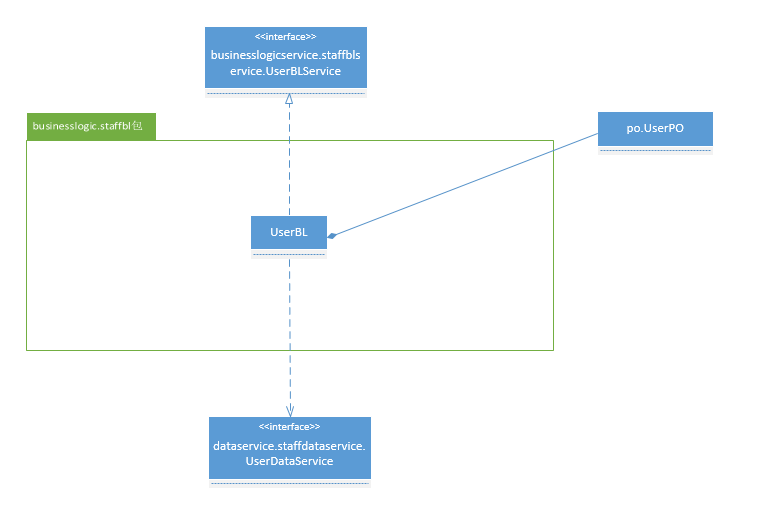
用户管理模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

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 |
| 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

**User的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| User.addUser | 语法 | public void addUser(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.updataUser | 语法 | public ArrayList<UserVO> updateUser(int pos, String oldname, String name,  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.deleteUser | 语法 | public void deleteUser(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.checkUsers | 语法 | public ArrayList<UserVO> checkUsers(); |
| 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

#### 4.1.11.1SalaryPolicy

###### 模块描述

SalaryPolicy用例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，

SalaryPolicy用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice. SalaryPolicyBLService接口。业务逻辑层和数据层之间添加dataservice. SalaryPolicyDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SalaryPolicyController，这样SalaryPolicyController会将薪水管理的业务逻辑委托给SalaryPolicy对象。



**ListinBL用例模块各个类的职责**

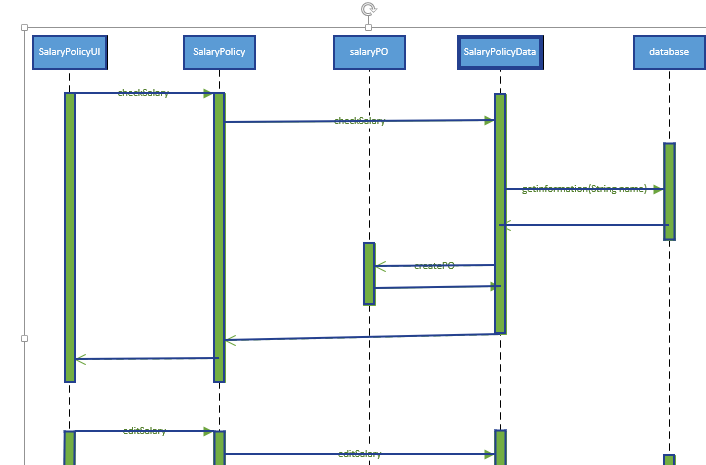
|  |  |
| --- | --- |
| **模块** | **职责** |
| SalaryPolicybl | 订单输入的领域模型对象，可以帮助完成订单输入界面所需要的服务 |

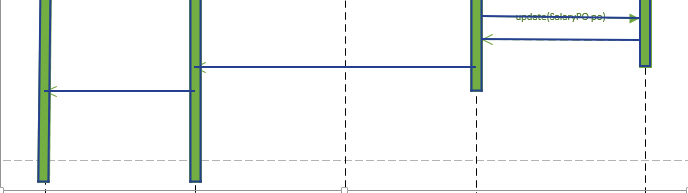
###### 模块内部类的接口规范

**表22 SalaryPolicy模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| SalaryPolicybl.checkSalary | 语法 | public SalaryVO checkSalary(String typeOfStaff) |
| 前置条件 | 点击人员薪水表的其中一项 |
| 后置条件 | 界面薪水表项对应的具体信息 |
| SalaryPolicybl.editSalary | 语法 | public void editSalary(SalaryVO vo) |
| 前置条件 | 启动一个编辑人员信息任务 |
| 后置条件 | 持续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 | |
| PolicyDataService.get (String typeOfStaff) | 获得单一持久化对象 | |
| PolicyDataService.updataSalary(SalaryPO po) | 修改单一持久化对象 | |

###### 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。

#### 4.1.11.2ConstantPolicy

###### 模块描述

ConstantPolicy用例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，

ConstantPolicy用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice. ConstantPolicyBLService接口。业务逻辑层和数据层之间添加dataservice. ConstantPolicyDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ConstantPolicyController，这样ConstantPolicyController会将薪水管理的业务逻辑委托给ConstantPolicy对象。



**ConstantPolicy用例模块各个类的职责**

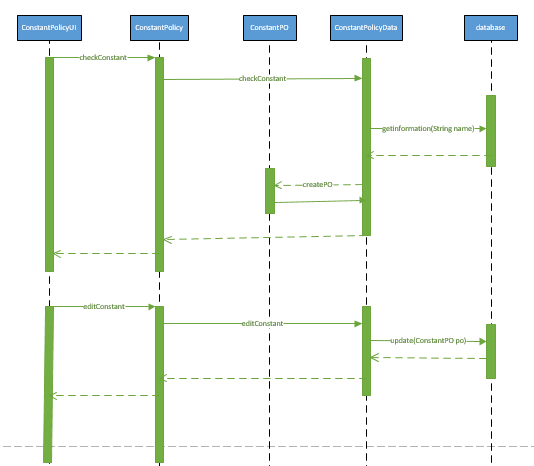
|  |  |
| --- | --- |
| **模块** | **职责** |
| ConstantPolicy | 订单输入的领域模型对象，可以帮助完成订单输入界面所需要的服务 |

###### 模块内部类的接口规范

**表22 ConstantPolicybl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ConstantPolicy.searchConstant | 语法 | public ConstantVO checkConstant(String address1, String address2) |
| 前置条件 | 启动一个查询两地价格和距离信息任务，并且两地地址输入规范 |
| 后置条件 | 查找是否存在相应两地，返回查询验证的结果 |
| ConstantPolicy.editConstant | 语法 | public void editConstant(ConstantVO vo) |
| 前置条件 | 启动一个修改常量表任务 |
| 后置条件 | 持续化更新涉及的领域对象的数据 |
| **需要的服务（需接口）** | | |
| PolicyDataService.find(String address1，String address2) | 根据String查找单一持久化对象 | |
| PolicyDataService.updataConstant(ConstantPO po) | 修改单一持久化对象 | |

###### 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。

### Approvebl模块

#### 4.1.12.1Approve

###### 模块描述

Approve用例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，

Aprrove用例模块的职责及接口参见软件系统结构描述文档表10

###### 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，我们添加businesslogicservice. ApproveBLService接口。业务逻辑层和数据层之间添加dataservice. ApproveDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ApproveController，这样ApproveController会将薪水管理的业务逻辑委托给Approve对象。



**Approve用例模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| Approve | 订单输入的领域模型对象，可以帮助完成订单输入界面所需要的服务 |

###### 模块内部类的接口规范

**表23 approvebl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| approve.showList | 语法 | public ArrayList<String> showList(String type) |
| 前置条件 | 选择单据种类和状态后，请求显示相关单据 |
| 后置条件 | 界面显示相应的所有单据 |
| approve.editList | 语法 | public void editList(int num,String type) |
| 前置条件 | 启动一个编辑单据信息任务 |
| 后置条件 | 持续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 | |
| checkPayeeOrderInfo.checkPayeeorder() | 得到所有未审批的付款单 | |
| checkPayeeOrderInfo.save(PayorderVO vo) | 存储审批通过的付款单 | |
| ApproveWarehouse\_outInfo.get(); | 得到所有未审批的出库单 | |
| ApproveWarehouse\_outInfo.save(StoreoutorderPO po) | 存储审批通过的出库单 | |
| approveTransferInfo.get() | 得到所有未审批的中转单 | |
| approveTransferInfo.save(ChangeorderPO po) | 存储审批通过的中转单 | |
| ApproveWarehouse\_inInfo.get() | 得到所有未审批的入库单 | |
| ApproveWarehouse\_inInfo.save(StoreinorderPO po) | 存储审批通过的入库单 | |
| approveReceiveInfo.get() | 得到所有未审批的中转中心到达单 | |
| approveReceiveInfo.save(ArriverorderPO po) | 存储审批通过的中转中心到达单 | |
| checkPayeeOrderInfo.checkPayeeorder() | 得到所有未审批的收款单 | |
| checkPayeeOrderInfo.save(PayeeorderVO po) | 存储审批通过的收款单 | |
| ApproveReceiFormInfo.get() | 得到所有未审批的营业厅到达单 | |
| ApproveReceiFormInfo.save(ReceiveorderPO po) | 存储审批通过的营业厅到达单 | |
| approveLoadInfo.get(); | 得到所有未审批的装车单 | |
| approveLoadInfo.save(LoadorderPO po) | 存储审批通过的装车单 | |
| ListApprove.getAllOrders() | 得到所有未审批的寄件单 | |
| ListApprove.save(OrderPO po) | 存储审批通过的寄件单 | |

###### 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。

# 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

# 6.人机交互视角

详见人机交互设计文档

# 7.重要逻辑类耦合与内聚度量

参见重要逻辑类度量数据文档