详细设计描述文档

物流信息管理系统

更新历史：

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 修改原因 | 版本号 |
| 栾志远 | 2015/10/26 | 完成文档 | V1.0 |

# 1.引言

## 1.1编制目的

本报告详细完成对物流信息管理系统的概要设计，达到指导详细设计和开发的目的，同

时实现和开发人员及用户的沟通。

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。

# 2.产品概述

参考物流信息管理系统用例文档和物流信息管理系统软件需求规格说明中对产品的概

括描述。

# 3.体系结构设计概述

参考物流信息管理系统概要设计文档中对体系结构设计的概述。

# 4.结构视角

## 4.1业务逻辑层的分解



### 4.1.1 informationbl模块

（1）模块概述

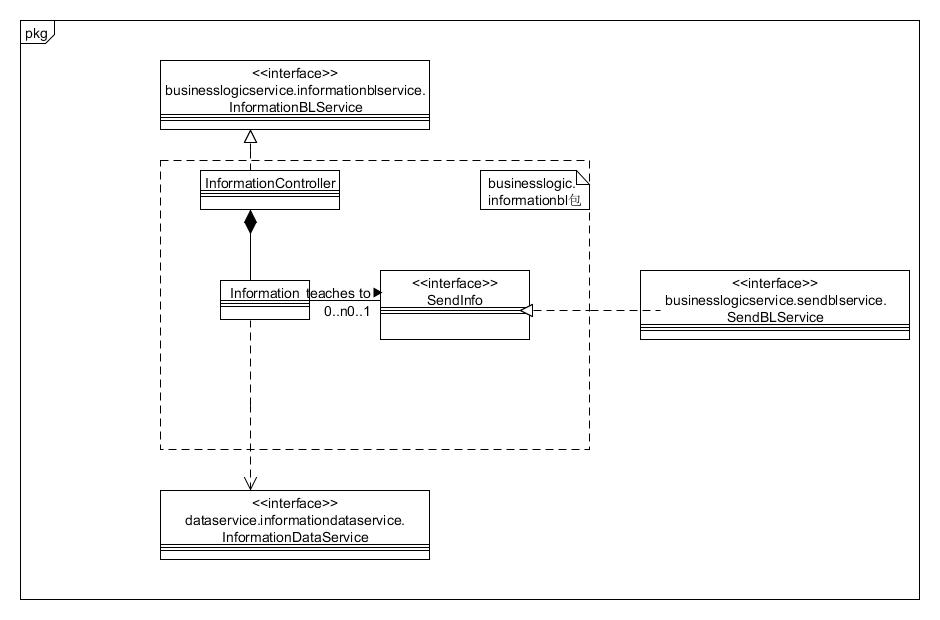
Informationbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Informationbl模块的职责及接口参见软件系统结构描述文档。

（2）整体结构

根据体系结构的设计，我们将系统分成展示层，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。比如在展示层与业务逻辑层之间，我们添加了businesslogicservice.informationblservice.InformationBLService接口。业务逻辑层与数据层之间，我们添加了dataservice.informationdataservice.InformationDataService接口。为了隔离业务逻辑职责与逻辑控制职责，我们增加了InformationController，这样对物流信息查询的业务逻辑处理委托给information对象。

**Informationbl模块的设计如图所示**



**Informationbl模块各个类的职责如表所示**

|  |  |
| --- | --- |
| **模块** | **职责** |
| InformationController | 负责实现查询界面需要的服务 |
| Information | 物流信息查询的领域模型对象，帮助实现查询界面需要的服务 |

（3）模块内部类的接口规范

InformationController和information的接口规范如表所示

**InformationController的接口规范**

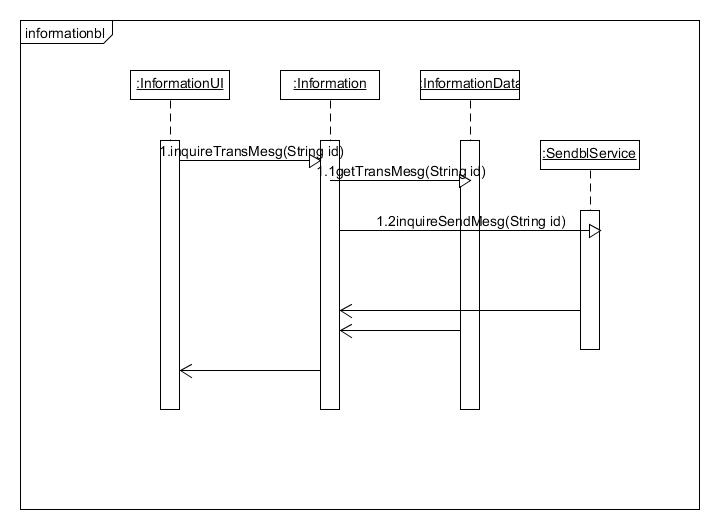
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| InformationController  .InquireTransMesg | 语法 | Public resultmessage inquireTransMesg(String id) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 调用information对象的inquireTransMesg方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Information.InquireTransMesg | 查询快递信息 | |

**Information的接口规范**

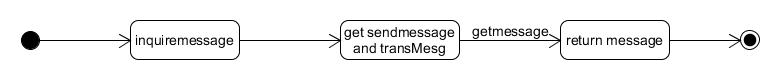
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Information.inquireTransMesg | 语法 | Public resultmessage inquireTransMesg(String id) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 返回快递物流信息 |
| Information.refreshMesg | 语法 | Public resultmessage refreshMesg(String id,String position,String time) |
| 前置条件 | 输入符合规则 |
| 后置条件 | 更新物流信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| InformationDataService  .getTransMesg(String id) | 返回物流信息 | |
| InformationDataService  .refreshTransMesg(String id,String position,String time) | 更新物流信息 | |
| Sendbl.inquireSendMesg(String id) | 查询物流信息 | |

（4）业务逻辑层的动态模型

下图表明了快递物流系统中，当用户输入查询的单号之后，查询业务逻辑处理的相关对象之间的协作



下图显示information对象生存期间的状态序列。开始时进入inquiremessage状态，随后进入getmessage状态，得到信息之后，进入返回信息状态，最后退出。



（5）业务逻辑层的设计原理

利用委托式风格，将不同的任务由各自的控制器交给不同的领域对象处理。

### 4.1.2 sendbl模块

（1）模块概述

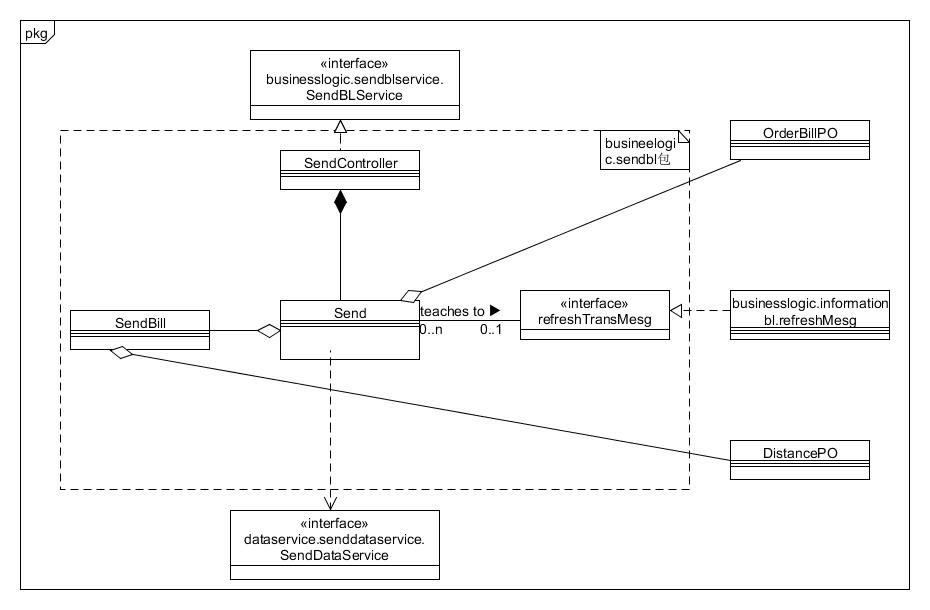
Sendbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Sendbl模块的职责及接口参见软件系统结构描述文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.sendblservice.SendBLService接口。业务逻辑层和数据层之间添加dataservice.salesdataservice.SalesDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加SendController，这样SendController把对业务逻辑处理委托给Send对象。OrderBillPO是作为订单信息的持久化对象被添加到设计模型中去的。SendBill封装了计算总价的职责，DistancePO是各地之间距离信息的持久化对象。

**Informationbl模块的设计如图所示**



**Informationbl模块各个类的职责如表所示**

|  |  |
| --- | --- |
| **模块** | **职责** |
| SendController | 负责实现订单输入界面需要的服务 |
| Send | 订单输入的领域模型对象，帮助实现订单输入需要的服务 |
| SendBill | 负责实现计算时间以及报价 |

（3）模块内部类的接口规范

**SendController的接口规范如表所示**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| SendController  .Input | 语法 | Public resultmessage input(String nameOfSender,String addressOfSender,String workplaceOfSender,String phoneOfSender,String mobileOfSender, String nameOfReceiver,String addressOfReceiver,String workplaceOfReceiver,String phoneOfReceiver,String mobileOfReceiver,int numOfItem,double weight,String nameOfItemInside,double[][][] size,double moneyForCover,String speedLevel,String kindOfSend) |
| 前置条件 | 使用者有输入订单的权限 |
| 后置条件 | 系统申请存储订单信息 |
| SendController  .inquireSendMesg | 语法 | Public resultmessage inquireSendMesg(String id) |
| 前置条件 | 使用者有查看订单信息的权限 |
| 后置条件 | 无 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Send.InquireTransMesg | 查询订单信息 | |
| Send.input | 输入订单信息 | |

**Send的接口规范如表所示**

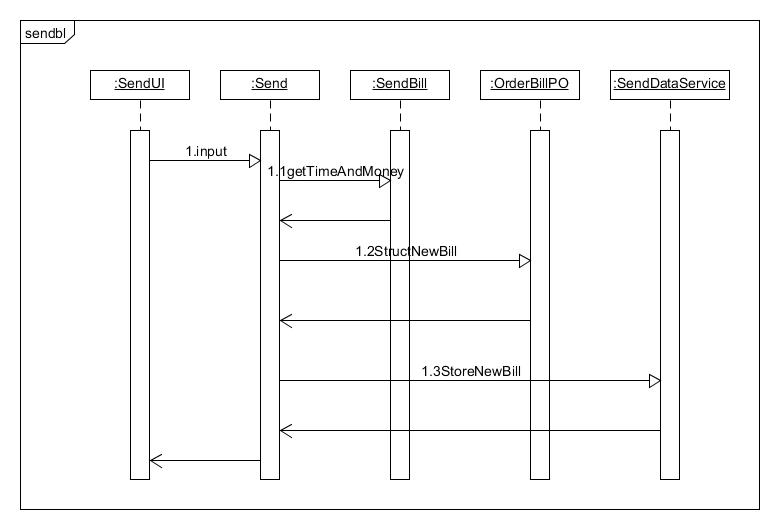
|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| Send.inquireSendMesg | 语法 | Public resultmessage inquireSendMesg(String id) |
| 前置条件 | 使用者有查看订单信息的权限 |
| 后置条件 | 无 |
| Send.input | 语法 | Public resultmessage input(String nameOfSender,String addressOfSender,String workplaceOfSender,String phoneOfSender,String mobileOfSender, String nameOfReceiver,String addressOfReceiver,String workplaceOfReceiver,String phoneOfReceiver,String mobileOfReceiver,int numOfItem,double weight,String nameOfItemInside,double[][][] size,double moneyForCover,String speedLevel,String kindOfSend) |
| 前置条件 | 使用者有输入订单的权限 |
| 后置条件 | 系统申请存储订单信息 |
| **需要的接口（需接口）** | | |
| 服务名 | 服务 | |
| Senddata.getSendMesg | 返回单一持久化对象（物流订单信息） | |
| Senddata.getInput | 写入单一持久化对象（订单信息） | |
| Imformationbl.refreshMesg | 写入单一持久对象（物流信息） | |
| SendBill.getTimeAndMoney | 得到订单时间和报价 | |

**SendBill的接口规范如表所示**

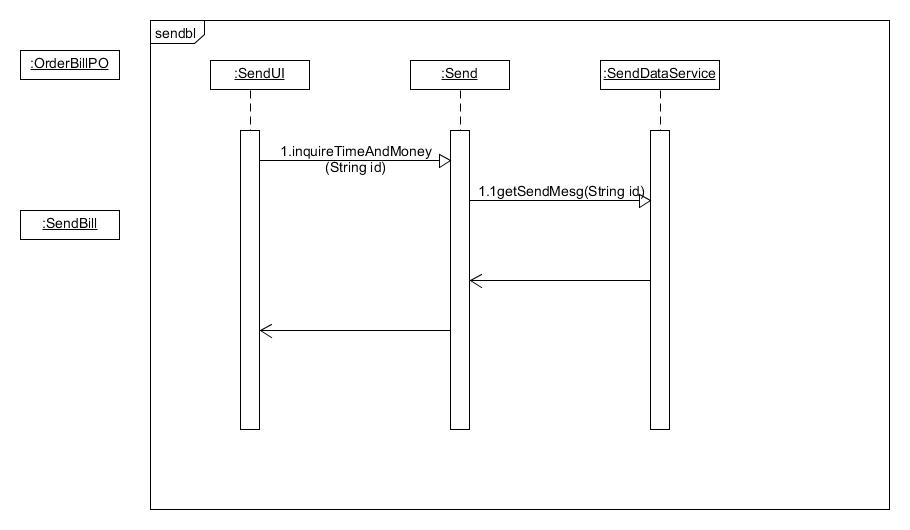
|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| SendBill.getTimeAndMoney | 语法 | Public resultmessage getTimeAndMoney (String location,destination,speedlevel) |
| 前置条件 | 使用者有查看订单信息的权限 |
| 后置条件 | 无 |

（4）业务逻辑层的动态模型

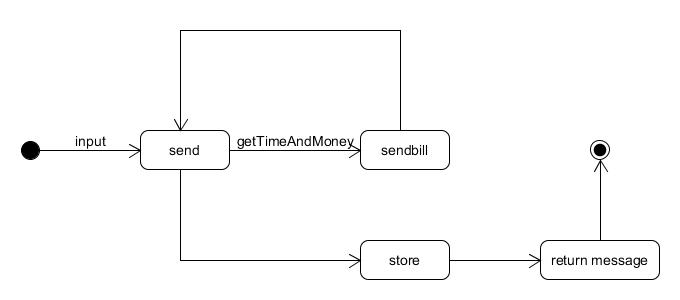
下图表明了快递物流系统中，当快递员输入订单信息之后，订单输入逻辑处理的相关对象之间的协作



下图表明了快递物流系统中，当快递员输入查询的单号之后，时间和报价查询的相关对象之间的协作



下图显示send对象生存期间的状态序列。开始时进入send状态，随后接收输入信息后进入sendbill状态，得到信息之后，返回send状态，随后进入store状态，然后进入return message状态，最后退出。



（5）业务逻辑层的设计原理

利用委托式风格，将不同的任务由各自的控制器交给不同的领域对象处理。

### 4.1.3 judgementbl模块

（1）模块概述

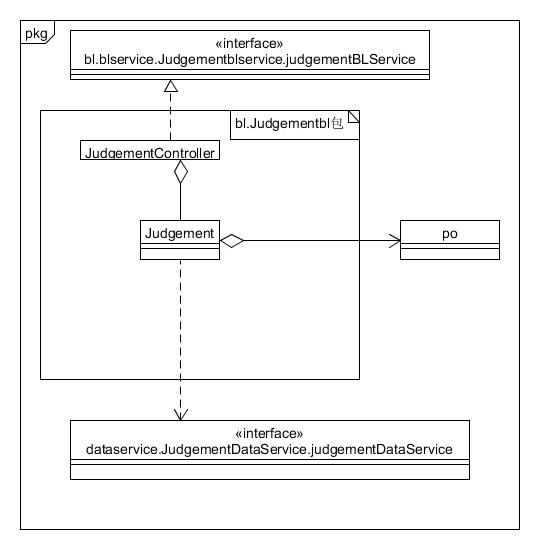
judgementbl模块承担的需求详见需求规格说明文档功能需求及相关非功能需求。

judgementbl模块的职责及接口参见软件系统结构描述文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bl.blservice.Judgementblservice.judgementBLService接口。业务逻辑层和数据层之间添加dataservice.JudgementDataService.judgementDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了JudgementController，这样JudgementController会将对销售的业务逻辑处理委托给Judgement对象。po是作为单据的持久化对象被添加到设计模型中去的。

judgementbl模块的设计如图所示。



**Judgementbl模块各个类的设计**

judgementbl模块各个类的职责如表所示。

**judgementbl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| JudgementController | 负责实现单据审判界面所需要的服务 |
| Judgement | 单据审判的领域模型对象，拥有一次单据审判所持有的订单信息，可以帮助完成单据审判界面所需要的服务 |

（3）模块内部类的接口规范

JudgementController和Judgement的接口规范如表所示。

**JudgementController的接口规范**

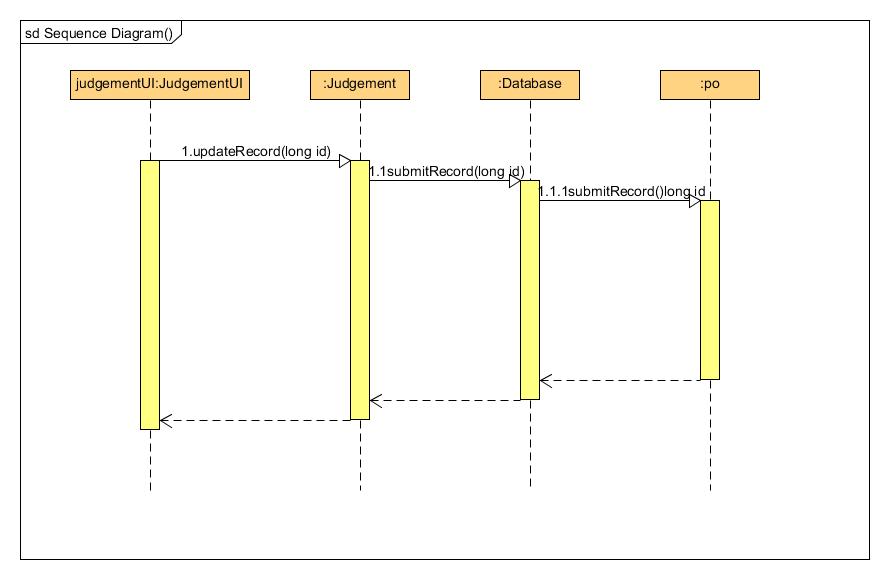
|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| **JudgementController.checkRecord** | 语法 | public ResultMessage checkRecord（long id） |
| 前置条件 | 已创建一个Judgement领域对象 |
| 后置条件 | 调用Judgement领域对象的checkRecord方法 |
| **JudgementController.updateRecord** | 语法 | Public ResultMessage updateRecord（long id） |
| 前置条件 | 已创建一个Judgement领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Judgement领域对象的updateRecord方法 |
| **JudgementController.submitRecord** | 语法 | public ResultMessage submitRecord  (long id) |
| 前置条件 | 已创建一个Judgement领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Judgement领域对象的submitRecord方法 |
| **需要的服务（需接口）** | | |
| **Judgement.checkRecord（long id）** | 查找一个单据对象 | |
| **Judgement.updateRecord（long id）** | 修改一个单据对象 | |
| **Judgement.submitRecord（long id）** | 更新一个单据对象 | |

**judgementbl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| **Judgement.checkRecord** | 语法 | public ResultMessage checkRecord（long id） |
| 前置条件 | 有需要审核的单据 |
| 后置条件 | 查看单据内容 |
| **Judgement.updateRecord** | 语法 | Public ResultMessage updateRecord（long id） |
| 前置条件 | 已经查看到单据内容 |
| 后置条件 | 修改单据内容 |
| **Judgement.submitRecord** | 语法 | public ResultMessage submitRecord  (long id) |
| 前置条件 | 确认单据内容没有错误 |
| 后置条件 | 通过单据，持久化更新涉及的对象的数据 |
| **需要的服务（需接口）** | | |
| **JudgementDataService.find** | 插入单一持久化对象 | |
| **JudgementDataService.update** | 更新单一持久化对象 | |
| **DatabaseFacory.getJudgementDatabase** | 得到Judgement数据库的引用 | |

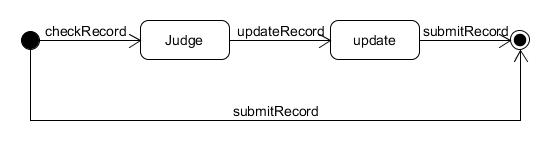
（4）业务逻辑层的动态模型

下图表明了物流信息管理系统中，当总经理点击审判单据后，单据审判业务逻辑处理的相关对象之间的协作。



**审判单据的顺序图**

如下图所示的状态图描述了Judgement对象的生存期间的状态序列、引起转移到事件，以及因状态转移而伴随的动作。随着checkRecord方法被UI调用，Judgement进入Judge状态；之后通过修改数据进入Update状态。



**Judgement对象状态图**

（5）业务逻辑层的设计原理

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

### 4.1.4 shipmentbl模块

（1）模块概述

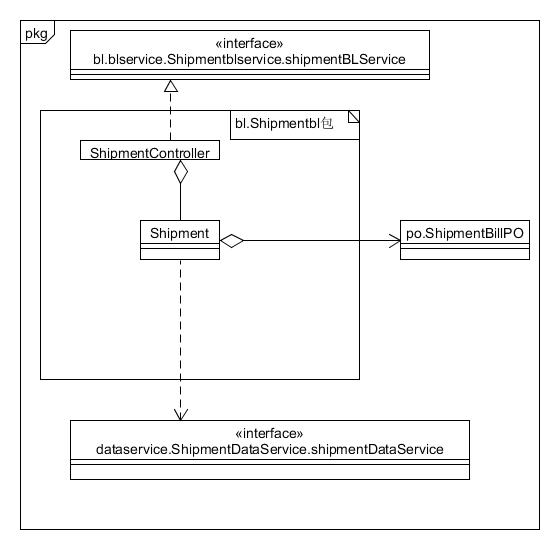
shipmentbl模块承担的需求详见需求规格说明文档功能需求及相关非功能需求。

shipmentbl模块的职责及接口参见软件系统结构描述文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bl.blservice.Shipmentblservice.shipmentBLService接口。业务逻辑层和数据层之间添加dataservice.ShipmentDataService.shipmentDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ShipmentController，这样ShipmentController会将对销售的业务逻辑处理委托给Shipment对象。ShipmentBillPO是作为单据的持久化对象被添加到设计模型中去的。

shipmentbl模块的设计如图所示。



**Shipmentbl模块各个类的设计**

shipmentbl模块各个类的职责如下表所示。

**shipmentbl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| ShipmentController | 负责实现装运界面所需要的服务 |
| Shipment | 装运的领域模型对象，拥有一次装运所持有的单据信息，可以帮助完成装运界面所需要的服务 |

（3）模块内部类的接口规范

ShipmentController和Shipment的接口规范如表所示。

ShipmentController**的接口规范**

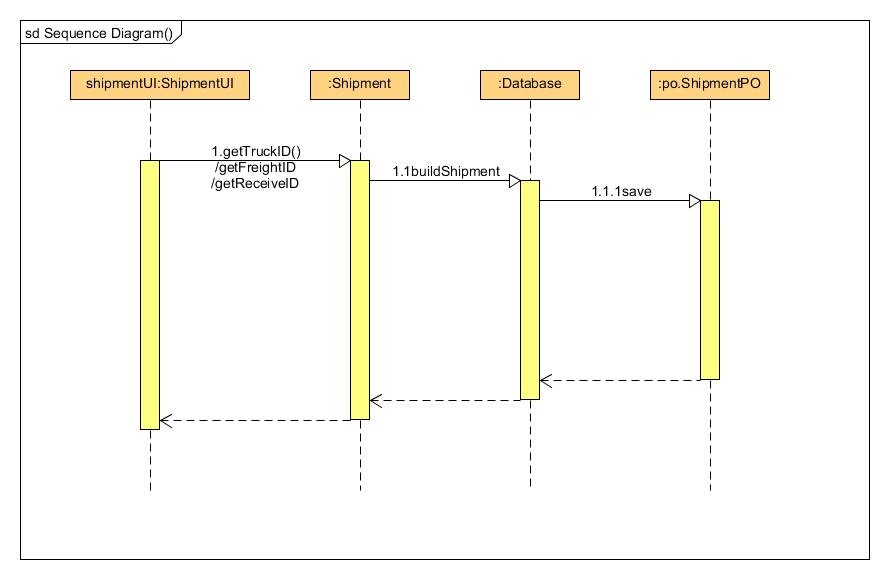
|  |  |  |  |
| --- | --- | --- | --- |
| **提供的接口（供接口）** | | | |
| **ShipmentController. getTruckID** | 语法 | | public String getTruckID () |
| 前置条件 | | 已创建一个Shipment领域对象 |
| 后置条件 | | 调用Shipment领域对象的getTruckID方法 |
| **ShipmentController. getFreightID** | 语法 | | public String getFreightID () |
| 前置条件 | | 已创建一个Shipment领域对象 |
| 后置条件 | | 调用Shipment领域对象的getFreightID方法 |
| **ShipmentController. getReceiveID** | 语法 | | public String getReceiveID () |
| 前置条件 | | 已创建一个Shipment领域对象 |
| 后置条件 | | 调用Shipment领域对象的getReceiceID方法 |
| **ShipmentController. buildShipment** | 语法 | | public ShipmentBillPO buildShipment() |
| 前置条件 | | 已创建一个Shipment领域对象，并且输入符合输入规则 |
| 后置条件 | | 调用Shipment领域对象的buildShipment方法 |
| **ShipmentController. save** | 语法 | | public ShipmentVO save(ShipmentInputInfo info) |
| 前置条件 | | 已创建一个Shipment领域对象 |
| 后置条件 | | 调用Shipment领域对象的sava方法 |
| **需要的接口（需接口）** | | | |
| **服务名** | | **服务** | |
| **Shipment. getTruckID** | | 加入一个装车单对象 | |
| **Shipment. getFreightID** | | 加入一个中转单对象 | |
| **Shipment. getReceiveID** | | 加入一个中转中心到达单对象 | |
| **Shipment. buildShipment** | | 添加数据到单据对象中 | |
| **Shipment. save** | | 保存一个单据对象 | |

**Shipment的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的接口（供接口）** | | | |
| **Shipment. getTruckID** | 语法 | | public String getTruckID () |
| 前置条件 | | 开始创建一个装车单 |
| 后置条件 | | 得到该装车单的编号 |
| **Shipment. getFreightID** | 语法 | | public String getFreightID () |
| 前置条件 | | 开始创建一个中转单 |
| 后置条件 | | 得到该中转单的编号 |
| **Shipment. getReceiveID** | 语法 | | public String getReceiveID () |
| 前置条件 | | 开始创建一个中转到达单 |
| 后置条件 | | 得到该中转到达单的编号 |
| **Shipment. buildShipment** | 语法 | | public ShipmentBillPO buildShipment() |
| 前置条件 | | 添加相应的数据到装运信息单据中 |
| 后置条件 | | 生成单据PO |
| **Shipment. save** | 语法 | | public ShipmentVO save(ShipmentInputInfo info) |
| 前置条件 | | 单据已经被制定出 |
| 后置条件 | | 单据被保存 |
| **需要的接口（需接口）** | | | |
| **服务名** | | **服务** | |
| **ShipmentDataService. findTruck（int id）** | | 根据ID进行查找单一持久化对象 | |
| **ShipmentDataService.insertTruck（ShipmentBillPO po）** | | 插入单一持久化对象 | |
| **ShipmentDataService.deleteTruck (ShipmentBillPO)** | | 删除单一持久化对象 | |
| **ShipmentDataService.updateTruck（ShipmentBillPO po）** | | 更新单一持久化对象 | |
| **ShipmentDataService.findFreight (int id)** | | 根据ID进行查找单一持久化对象 | |
| **ShipmentDataService.insertFreight（ShipmentBillPO po）** | | 插入单一持久化对象 | |
| **ShipmentDataService.deleteFreight（ShipmentBillPO po）** | | 删除单一持久化对象 | |
| **ShipmentDataService.updateFreight（ShipmentBillPO po）** | | 更新单一持久化对象 | |
| **DatabaseFacory.getShipmentDatabase** | | 得到Shipment数据库的引用 | |

（4）业务逻辑层的动态模型

下图表明了物流信息管理系统中，当用户后，单据审判业务逻辑处理的相关对象之间的协作。



**新建装运单的顺序图**

如下图所示的状态图描述了Shipment对象的生存期间的状态序列、引起转移到事件，以及因状态转移而伴随的动作。随着getID方法被UI调用，Shipment进入Bill状态；之后通过修改数据进入update状态。



**Shipment对象状态图**

### receivementbl模块

(1)模块概述

receivementbl模块承担的需求参见需求规格说明文档功能需求和相关非功能需求

receivementbl模块的职责及接口参见软件系统结构描述文档

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.receivementblservice.ReceivementBLService接口。业务逻辑层和数据层之间添加dataservice.receivementdataservice.ReceivementDataService接口，为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceivementController，这样ReceivementController会将对收件的业务逻辑处理委托给Receivement对象。CommodityPO是作为收件货物信息记录的持久化对象被添加到设计模型中去的。而ReceivementList的添加是CommodityInfo的容器类。ReceivementList封装了收件货物信息的数据集合的数据结构的秘密。

Receivementbl模块的设计如图

pkg

<<interface>>

businesslogicservice.receivementblservice.ReceivementBLService

po.CommodityPO

businesslogic.receivementbl包

ReceivementController

businesslogic.receivementbl.Receivement

ReceivementList

<<interface>>

CommodityInfo

receivement

<<interface>>

dataservice.receivementdataservice.ReceivementDataService

receivementbl模块各个类的职责如表所示

**receivementbl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| LoginController | 负责实现对应登录界面所需要的服务 |
| receivementController | 负责收件信息界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的名称和密码，可以解决登录问题 |
| receivement | 收件信息的领域模型对象，拥有收到的所有货物的详细信息，可以帮助收件业务所需要的服务 |

（3）模块内部类的接口规范

**ReceivementController模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReceivementController.addCommodity | 语法 | Public ResultMessage addCommodity(long id) |
| 前置条件 | 已创建一个Receivement领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Receivement领域对象的addCommodity方法 |
| ReceivementContoller.update | 语法 | Public ResultMessage update(CommodityPO po) |
| 前置条件 | 已创建一个Receivemen领域对象，输入符合输入规则 |
| 后置条件 | 调用Receivement领域对象的update方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Receivement.addCommodity | 加入一个货物对象信息 | |
| Receivement.update | 更新库存货物信息 | |

**Receivement的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Receivement.addcommodity | 语法 | Public ResultMessage addCommodity(String id) |
| 前置条件 | 启动一个收件任务 |
| 后置条件 | 在一次收件任务中，增加货物信息 |
| Receivement.update | 语法 | Public ResultMessage update(String id) |
| 前置条件 | 已输入收件货物信息 |
| 后置条件 | 系统持久化更新收件信息 |
| Receivement.delete | 语法 | Public ResultMessage delete(String id) |
| 前置条件 | 存在已经输入的Po |
| 后置条件 | 将选中的po删除 |
| **需要的服务（需接口）** | | |
| 服务名 |  | 服务 |
| Receivementdataservice.insert(ReceivementPO po) | 插入单一持久化对象 |  |

（4）业务逻辑层的动态模型

在物流信息管理系统中，当用户输入收件信息后，库存业务逻辑处理相关对象之间的协作。

得到收件信息的顺序图

1. addCommodity 1.1:addCommodity(long id) 1.1.1commodityPO

find(long id)

1.1.2addCommodity(long id)

Sd Sequence Diagram

commoditypo

DataBase

:receivementList

:receivement

Receivementui:

ReceivementUI

（5）业务逻辑设计原理

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。

### commoditybl模块

(1)模块概述

commoditybl模块承担的需求参见需求规格说明文档功能需求和相关非功能需求

commoditybl模块的职责及接口参见软件系统结构描述文档

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层为了增加灵活性，我们会添加接口，比如展示层和业务逻辑层之间，添加businesslogicservice.commodityblservice.CommodityBLService接口。为了隔离业务逻辑职责和逻辑控制职责，添加了CommodityController，这样CommodityController会将对库存管理的业务逻辑处理交给Commodity对象。CommodityPO是作为库存货物记录的持久化对象被添加到设计模型中去的。CommodityList的添加是CommodityInfo的容器类。CommodityList封装了库存货物的数据信息和数据结构的秘密。CommodityInfo是根据依赖倒置原则，为消除循环依赖而产生的接口。

commoditybl模块的设计如图。

pkg

<<interface>>

businesslogicservice.commodityblservice.CommodityBLService

po.Commoditypo

businesslogic.commoditybl包

CommodityController

<<interface>>

CommodityInfo

commodity

businesslogic.commoditybl.Commodity

CommodityList

<<interface>>

dataservice.commoditydataservice.CommodityDataService

commoditybl模块各个类的职责如表

**commoditybl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| LoginController | 负责实现对应登录界面所需要的服务 |
| CommodityController | 负责库存信息界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的名称和密码，可以解决登录问题 |
| Commodity | 库存信息的领域模型对象，拥有库存的所有货物的详细信息，可以帮助库存管理所需要的服务 |

(3)模块内部类的接口规范

**CommodityController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| CommodityController.addCommodity | 语法 | Public ResultMessage addCommodity(long id) |
| 前置条件 | 已创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity领域对象的addCommodity方法 |
| CommodityContoller.update | 语法 | Public ResultMessage update(CommodityPO po) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合输入规则 |
| 后置条件 | 调用Commodity领域对象的update方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Commodity.addCommodity | 加入一个货物对象信息 | |
| Commodity.update | 更新库存货物信息 | |

**Commodity 的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的接口（供接口）** | | | |
| commodity.addcommo | 语法 | | Public ResultMessage addcommo(String id,String type,int numOfPosition) |
| 前置条件 | | 货物编号、货物类型、库区位置输入合法 |
| 后置条件 | | 将货物添加到仓库，持久化更新涉及的对象的信息 |
| commodity.deletecommo | 语法 | | Public ResultMessage deletecommo(String id) |
| 前置条件 | | 货物编号输入合法，且货物信息没被操作过 |
| 后置条件 | | 将货物从仓库中删除，持久化更新涉及的对象的信息 |
| commodity.updatecommo | 语法 | | Public ResultMessage updatecommo(String id,String type,int numOfPosition) |
| 前置条件 | | 货物编号、货物类型、库区位置输入合法 |
| 后置条件 | | 库存更新，持久化更新涉及的对象的信息 |
| commodity.findcommo | 语法 | | Public ResultMessage findcommo(String id) |
| 前置条件 | | 货物编号输入合法 |
| 后置条件 | | 查找到该货物的信息 |
| **需要的接口（需接口）** | | | |
| 服务名 | | 服务 | |
| Commoditydataservice.insert(CommodityPO po) | | 添加单一持久化对象 | |
| Commoditydataservice.delete(CommodityPO po) | | 删除单一持久化对象 | |
| Commoditydataservice.update（CommodityPO po) | | 更新单一持久化对象 | |
| Commoditydataservice.find(String id) | | 根据编号查找相应持久化对象 | |
| Commoditydataservice.getCommoditydata | | 得到commodity数据的服务引用 | |

(4)业务逻辑层的动态模型

在物流信息管理系统中，当用户输入了货物信息后，库存业务逻辑处理相关对象之间的协作。

Sd Sequence Diagram

commodityPo

:DataBase

:commodityList

:commodity

Commodityui:

CommodityUI

1:addCommodity

1.1:addCommodity(long id

1.1.1commodityPO find(long id)

1.1.2addCommodity(long id)

(5)业务逻辑设计原理

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 moneybl模块

(1)模块概述

moneybl模块承担的需求见需求规格说明文档功能需求及相关非功能需求。

moneybl模块的职责及接口见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每层之间为了增加灵活性，我们会添加接口。比如业务逻辑层和数据层之间添加dataservice.moneydataservice.MoneyDataService接口，展示层和业务逻辑层之间添加businesslogicservice.moneyblservice,MoneyBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了MoneyController，MoneyController会将对财务管理的业务逻辑处理委托给Money对象。MoneyPO是作为财务记录的持久化对象被添加到设计模型中去的。MoneyList和MoneyItem的添加是MoneyInfo的容器类。MoneyItem保有财务出入的详细条目，以及计算财务收益成本的职责。MoneyList封装了MoneyItem的数据集合的数据结构的秘密。MoneyInfo和MemberInfo分别保有财务的账目信息和账户信息，MoneyInfo和MemberInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

moneybl模块的设计如图

pkg

<<interface>>

businesslogicservice.moneyservice.MoneyService

po.MoneyPO

businesslogic.moneybl包

moneyController

businesslogic.memberbl.Member

<<interface>>

MemberInfo

moneyList

money

businesslogic.moneybl.MOney

<<interface>>

dataservice.moneydataservice.MoneyDataService

<<interface>>

Moneyinfo

moneyItem

**moneybl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| LoginController | 负责实现对应登录界面所需要的服务 |
| MoneyController | 负责财务管理界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的名称和密码，可以解决登录问题 |
| Money | 财务管理的领域模型对象，拥有财务管理的所有账目的详细信息，可以帮助完成财务管理所需要的服务 |
| Member | 账户信息管理的领域模型对象，拥有所有账户的账号信息，可以帮助完成账户信息管理所需要的服务 |

(3)模块内部类的接口规范

**MoneyController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| MoneyController.addmember | | 语法 | public ResultMessage addMember(long id) |
| 前置条件 | 已创建一个member领域对象，并且输入符合输入规则 |
| 后置条件 | 调用member领域对象的addMember方法 |
| MoneyController.getPayment | | 语法 | public ResultMessage getPayment(MoneyPO po) |
| 前置条件 | 已创建一个payment领域对象，并且输入符合输入规则 |
| 后置条件 | 调用payment领域对象的getPayment方法 |
| MoneyController.getExpense | | 语法 | Public ResultMessage getExpense(MoneyPO po) |
| 前置条件 | 已创建一个expense领域对象，并且输入符合输入规则 |
| 后置条件 | 调用expense领域对象的getExpense方法 |
| MoneyController.getItem | | 语法 | Public ResultMessage getItem(MoneyPO po) |
| 前置条件 | 已创建一个item领域对象，并且输入符合输入规则 |
| 后置条件 | 调用item 领域对象的getItem方法 |
| **需要的服务（需接口）** | | | |
| 服务名 | 服务 | | |
| MoneyController.addMember | 加入一个账户的信息 | | |
| MoneyController.getPayment | 得到公司的付款单 | | |
| MoneyController.getExpense | 得到公司的收款单 | | |
| MoneyController.getItem | 得到公司的成本收益表 | | |

**Money的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供服务）** | | | |
| Money.getPayid | 语法 | | Public String getPayid() |
| 前置条件 | | 系统产生付款记录 |
| 后置条件 | | 返回新建的付款单id |
| Money.getExpenseid | 语法 | | Public String getExpenseid() |
| 前置条件 | | 系统产生收款记录 |
| 后置条件 | | 返回新建的收款单id |
| Money.find | 语法 | | Public MoneyVO find(String id) |
| 前置条件 | | 财务人员输入合法的账单ID |
| 后置条件 | | 返回根据ID找到的单据 |
| Money.getSumMoney | 语法 | | Public ResultMessage getSumMessage() |
| 前置条件 | | 已得到所有收款和付款数据 |
| 后置条件 | | 得到总金额 |
| Money.submit | 语法 | | Public MoneyVO submit() |
| 前置条件 | | 单据已被制定出 |
| 后置条件 | | 将单据提交审批 |
| Money.save | 语法 | | Public MoneyVO save() |
| 前置条件 | | 单据已被制定出 |
| 后置条件 | | 单据被保存在payment和expense文件中 |
| Money.getMember | 语法 | | Public ResultMessage getMember(String name，double bill) |
| 前置条件 | | 账户已在系统中注册账号 |
| 后置条件 | | 得到账户信息（名称，交易金额） |
| **需要的服务（需服务）** | | | |
| 服务名 | | 服务 | |
| Moneydataservice.insert(MoneyPO po) | | 添加单一持久化对象 | |
| Moneydataservice.delete(String id) | | 删除单一持久化对象 | |
| Moneydataservice.update(MoneyPO po) | | 更新单一持久化对象 | |
| Moneydataservice.find(String id) | | 根据编号查找相应持久化对象 | |
| Moneydataservice.init() | | 初始化单一持久化对象 | |
| Moneydataservice.insertMember（String name) | | 添加单一持久化账户对象 | |
| Moneydataservice.deleteMember（String name) | | 删除单一持久化账户对象 | |
| Moneydataservice.findMember（String name) | | 查找单一持久化账户对象 | |

(4)业务逻辑层的动态模型

在物流信息管理系统中，加入一个新账户的动态模型如下

Sd Sequence Diagram

1:addMember

1.1:addMember(long id) 1.1.1MoneyPO add 1.1.2MoneyPO find

Member(long id) (long id)

:memberList

MoneyPO

:member

:dataBase

Moneyui:MoneyUI

(5)业务逻辑设计原理

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

## 4.2数据层的分解

下图为数据层的开发包图。

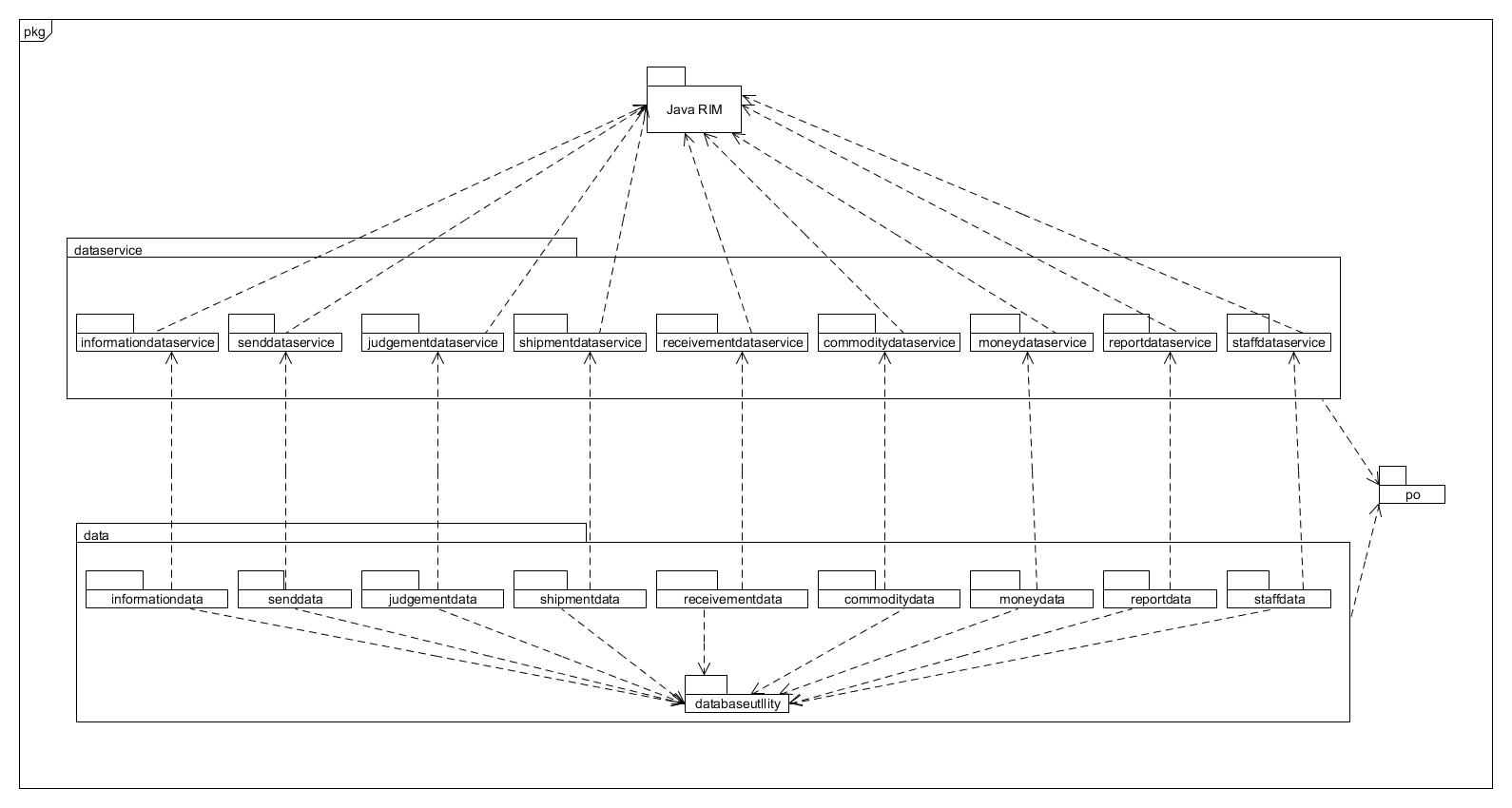


图4 物流系统服务器端开发包图

### 4.2.1 InformationData模块

（1）模块概述

InformationDataService模块提供物流信息查询的服务

（2）模块内接口规范如下

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| Imformationdata.getTransMesg | 语法 | Public resultmessage getTransMesg(String id) |
| 前置条件 | 快递信息已经返回 |
| 后置条件 | 无 |
| Imformationdata.refreshTransMesg | 语法 | Public resultmessage refreshTransMesg(String id,String position,String time) |
| 前置条件 | 货物接收成功 |
| 后置条件 | 系统保存货物物流信息 |

### 4.2.2 SendData模块

（1）模块概述

Sendblservice模块提供订单信息输入、查询、报价和时间查询的服务

（2）模块内部类的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| Senddata.getSendMesg | 语法 | Public resultmessage getSendMesg(String id) |
| 前置条件 | 无 |
| 后置条件 | 无 |
| Senddata.getInput | 语法 | Public resultmessage getInput(OrderBillVO) |
| 前置条件 | 使用者有写入的权限 |
| 后置条件 | 系统保存写入的订单信息 |

### 4.2.3 judgementdata模块

（1）模块概述

JudgementDataService模块提供单据审判数据集体载入、修改、保存

（2）模块内部类的接口规范

JudgementDataService类的接口规范如下表所示

**judgementdata模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| **JudgementDataService.find** | 语法 | public OrderBillPO find(long id) throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的OrderBillPO结果 |
| **JudgementDataService.update** | 语法 | public void update(OrderBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| **JudgementDataService.finish** | 语法 | public void finish() throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |
| **JudgementDataService.getID** | 语法 | public string getID() |
| 前置条件 | 无 |
| 后置条件 | 得到单据ID |
| **JudgementDataService.show** | 语法 | public ArrayList<OrderBill  PO> show() |
| 前置条件 | 无 |
| 后置条件 | 返回所有库存单据的数据 |

### 4.2.4 shipmentdata模块

（1）模块概述

ShipmentDataService模块提供装运单据数据集体载入、修改、保存

（2）模块内部类的接口规范

ShipmentDataService类的接口规范如下表所示

**shipmentdata模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的接口（供接口）** | | |
| **ShipmentDataService.findTruck** | 语法 | public ShipmentBillPO findTruck(long id) throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的ShipmentBillPO结果 |
| **ShipmentDataService.insertTruck** | 语法 | public void insertTruck(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 同样ID的po在数据库中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| **ShipmentDataService.deleteTruck** | 语法 | public void deleteTruck(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| **ShipmentDataService.updateTruck** | 语法 | public void updateTruck(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 删除一个po |
| **ShipmentDataService.finishTruck** | 语法 | public void finishTruck() throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |
| **ShipmentDataService.findFreight** | 语法 | public ShipmentBillPO findFreight(long id) throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的ShipmentBillPO结果 |
| **ShipmentDataService.insertFreight** | 语法 | public void insertFreight(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 同样ID的po在数据库中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| **ShipmentDataService.deleteFreight** | 语法 | public void deleteFreight(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| **ShipmentDataService.updateFreight** | 语法 | public void updateFreight(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 删除一个po |
| **ShipmentDataService.finishFreight** | 语法 | public void finishFreight() throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |
| **ShipmentDataService.findReceive** | 语法 | public ShipmentBillPO findReceive(long id) throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的ShipmentBillPO结果 |
| **ShipmentDataService.insertReceive** | 语法 | public void insertReceive(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 同样ID的po在数据库中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| **ShipmentDataService.deleteReceive** | 语法 | public void deleteReceive(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| **ShipmentDataService.updateReceive** | 语法 | public void updateReceive(ShipmentBillPO po) throws RemoteException |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 删除一个po |
| **ShipmentDataService.finishReceive** | 语法 | public void finishReceive() throws RemoteException |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |
| **ShipmentDataService.getID** | 语法 | public string getID() |
| 前置条件 | 无 |
| 后置条件 | 得到单据ID |
| **ShipmentDataService.show** | 语法 | public ArrayList<ShipmentBill  PO> show() |
| 前置条件 | 无 |
| 后置条件 | 返回所有库存单据的数据 |

### 4.2.5 receivementdata模块

（1）模块概述

ReceivementDataService提供接收单单据数据集体载入、修改、保存

（2）模块内部类的接口规范

ReceivementDataService类的接口规范如下表所示

**receivementdataservice 模块的接口规范**

|  |  |  |
| --- | --- | --- |
|  | 提供的服务 | （供接口） |
| receivementdataserv  ice.find | 语法 | Public ResultMessage find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| reseivementdataservice.storage | 语法 | Public ResultMessage storage(String id) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| receivementdataservice.delete | 语法 | Public ResultMessage delete(String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| receivementdataservice.update | 语法 | Public ResultMessage update(ReceivementPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| receivementdataservice.init | 语法 | Public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| receivementdataservice.insert | 语法 | Public void insert(ReceivementPO po) |
| 前置条件 | 同样id的po在数据中不存在 |
| 后置条件 | 在数据库中增加一条po记录 |

### 4.2.6 commoditydata模块

（1）模块概述

CommodityDataService提供库存单据数据集体载入、修改、保存

（2）模块内部类的接口规范

CommodityDataService类的接口规范如下表所示

**commoditydataservice模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| commoditydataservice.insert | 语法 | Public ResultMessage insert(CommodityPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 插入一条po数据 |
| commoditydataservice.delete | 语法 | Public ResultMessage delete(CommodityPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除一条po数据 |
| commoditydataservice.updata | 语法 | Public ResultMessage update(CommodityPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新Po数据 |
| commoditydataservice.find | 语法 | Public ResultMessage find(String id) |
| 前置条件 | 无 |
| 后置条件 | 返回查找相应的结果 |
| Commoditydataservice.init | 语法 | Public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### 4.2.7 moneydata模块

（1）模块概述

MoneyDataService提供财务单据数据集体载入、修改、保存

（2）模块内部类的接口规范

MoneyDataService类的接口规范如下表所示

**moneydataservicce模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| moneydataservice.insert | 语法 | Public ResultMessage insert(MoneyPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po记录 |
| moneydataservice.delete | 语法 | Public ResultMessage delete(String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除一条po数据 |
| moneydataservice.update | 语法 | Public ResultMessage update(MoneyPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新一条po数据 |
| moneydataservice.find | 语法 | Public MoneyPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| moneydataservice.init | 语法 | Public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| moneydataservice.getPayid | 语法 | Public String getPayid() |
| 前置条件 | 无 |
| 后置条件 | 返回付款单id |
| moneydataservie.getExpenseid | 语法 | Public String getExpenseid() |
| 前置条件 | 无 |
| 后置条件 | 返回收款单id |
| moneydataservice.getBillid | 语法 | Public String getBillid() |
| 前置条件 | 无 |
| 后置条件 | 返回成本收益单id |
| moneydataservice.getMember | 语法 | Public String getMember(String name) |
| 前置条件 | 账户已在系统中建立账号 |
| 后置条件 | 返回账户的账号名称 |
| moneydataservice.getMember  Bill | 语法 | Public String getMemberBill(double bill) |
| 前置条件 | 账户已在系统中进行交易 |
| 后置条件 | 返回查看账户的交易记录（交易金额） |

# 5.依赖视角

下图是客户端和服务器端各自的包之间的依赖关系。



图3 快递系统客户端开发包图

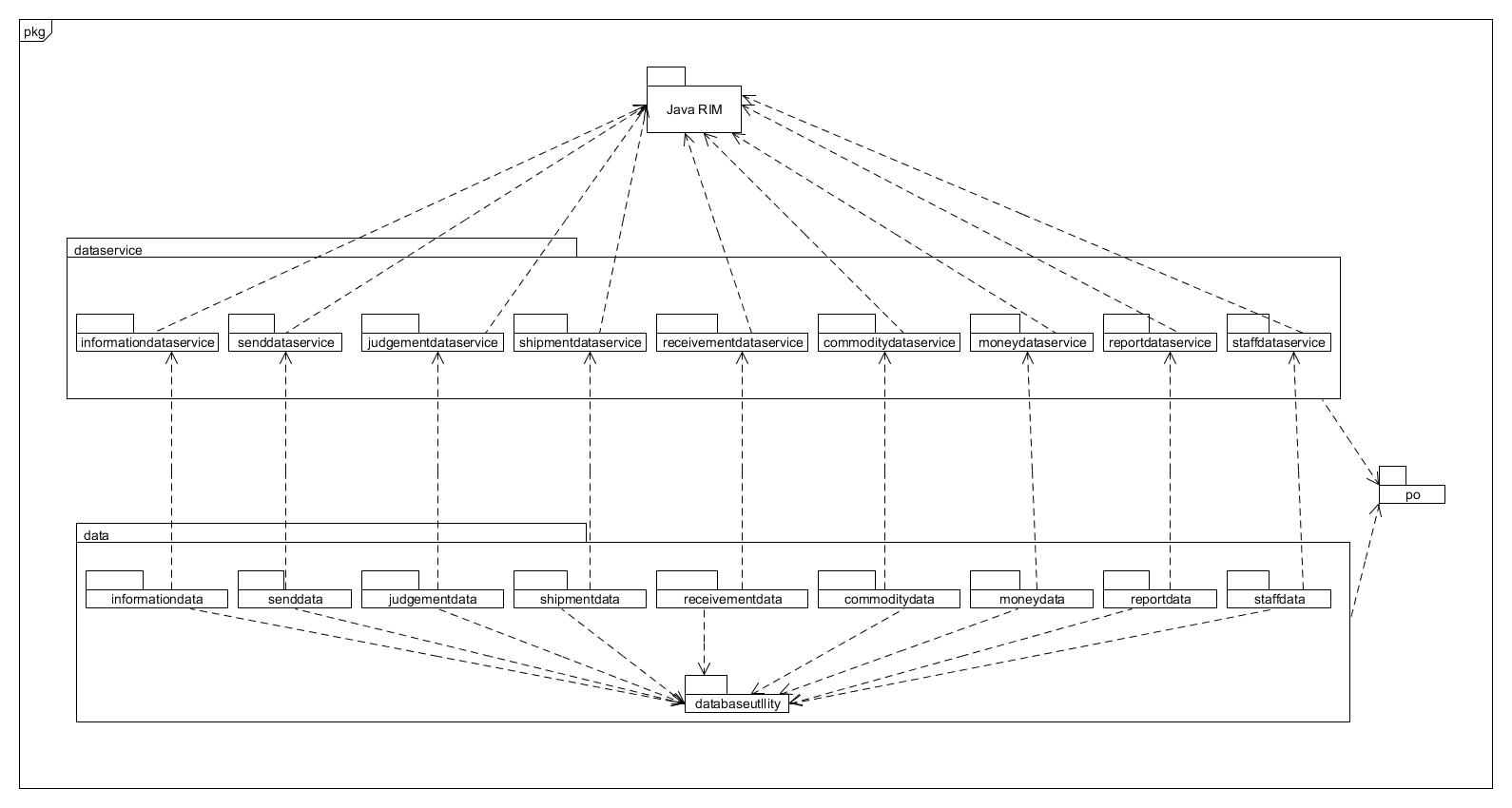


图4 物流系统服务器端开发包图