**软件系统详细设计文档**

**快递物流管理系统**

**第45组**

**陈之威**

**刘璇琳**

**王子安**

**张诗琦**

# 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 张诗琦 | 2015-10-30 | 创建文档框架 | V0.1 |
| 陈之威 | 2015-10-30 | 部分模块完成 | V0.2 |
| 张诗琦 | 2015-10-31 | 部分模块完成 | V0.3 |
| 王子安 | 2015-10-31 | 部分模块完成 | V0.4 |
| 刘璇琳 | 2015-11-1 | 全部模块完成 | V1.0 |
|  |  |  |  |
|  |  |  |  |

目录

[更新历史 2](#_Toc434182077)

[1. 引言 4](#_Toc434182078)

[1.1. 编制目的 4](#_Toc434182079)

[2. 产品概述 4](#_Toc434182080)

[3. 体系结构设计概述 4](#_Toc434182081)

[4. 结构视角 5](#_Toc434182082)

[4.1. 业务逻辑层的分解 5](#_Toc434182083)

[4.1.1. Orderbl模块 5](#_Toc434182084)

[4.1.2. Facilitybl模块 9](#_Toc434182085)

[4.1.3. Branchbl模块 15](#_Toc434182086)

[4.1.4. Basebl模块 22](#_Toc434182087)

[4.1.5. Receiptbl模块 27](#_Toc434182088)

[4.1.6. Userbl模块 32](#_Toc434182089)

[4.1.7. Organizationbl模块 37](#_Toc434182090)

[4.1.8. Transferbl模块 43](#_Toc434182091)

[4.1.9. OpeningStockbl模块 48](#_Toc434182092)

[4.1.10. Fundbl模块 51](#_Toc434182093)

[4.1.11. Inventorybl模块 69](#_Toc434182094)

[4.1.12. Accountbl模块 70](#_Toc434182095)

[4.1.13. Recordbl模块 70](#_Toc434182096)

[**4.2.** **数据层的分解** 74](#_Toc434182097)

[4.2.1. OrderDataService 74](#_Toc434182098)

[4.2.2. FacilityDataService 75](#_Toc434182099)

[4.2.3. BranchDataService 76](#_Toc434182100)

[4.2.4. BaseDataService 77](#_Toc434182101)

[4.2.5. ReceiptDataService 77](#_Toc434182102)

[4.2.6. UserDataService 78](#_Toc434182103)

[4.2.7. OrganizationDataService 79](#_Toc434182104)

[4.2.8. TransferDataService 79](#_Toc434182105)

[4.2.9. OpeningStockDataService 80](#_Toc434182106)

[4.2.10. DebitAndPayBillDataService 81](#_Toc434182107)

[4.2.11. DebitAndPayBillShowDataService 82](#_Toc434182108)

[4.2.12. BankAccountDataService 82](#_Toc434182109)

[4.2.13. ExpenseDataservice 83](#_Toc434182110)

[4.2.14. RecordDataService 84](#_Toc434182111)

[4.2.15. AccountDataService 84](#_Toc434182112)

[4.2.16. InventoryDataService 85](#_Toc434182113)

[5.依赖视角 88](#_Toc434182114)

# 引言

## 编制目的

本报告详细完成快递物流行业快递物流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。

# 产品概述

参考快递物流行业快递物流管理系统用例文档和快递物流行业快递物流管理系统软件规格说明中对产品的概括描述。

# 体系结构设计概述

请参考快递物流管理系统需求文档对体系结构设计的概述。

# 结构视角

## 业务逻辑层的分解

业务逻辑层的开发包图详见体系结构文档图3。

### Orderbl模块

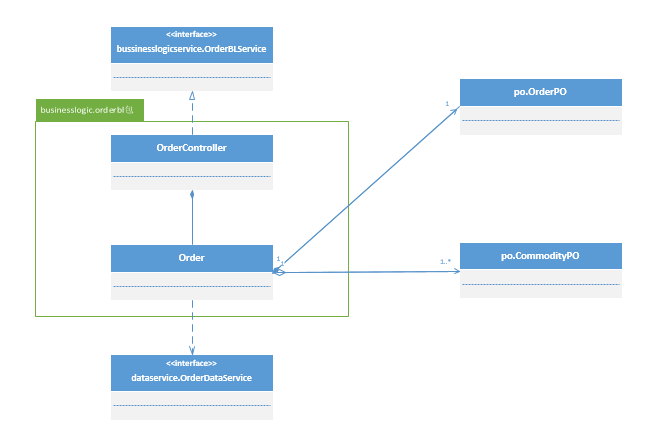
1. 模块概述

orderbl模块负责实现订单管理界面所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.OrderBLService接口。业务逻辑层和数据层之间添加dataservice.OrderDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OrderController，这样OrderController会将对订单的逻辑业务处理委托给Order对象。OrderPO是作为订单的持久化对象被添加到设计模型中去的。

Orderbl模块的设计如图



Orderbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| OrderController | 负责实现订单管理界面所需要的服务 |
| Order | 订单的领域模型对象，拥有订单的信息，可以帮助完成订单管理界面所需要的服务 |
| OrderInfo | 负责实现为其他包提供订单信息、更改订单信息的服务 |

1. 模块内部类的接口规范

**OrderController的接口规范**

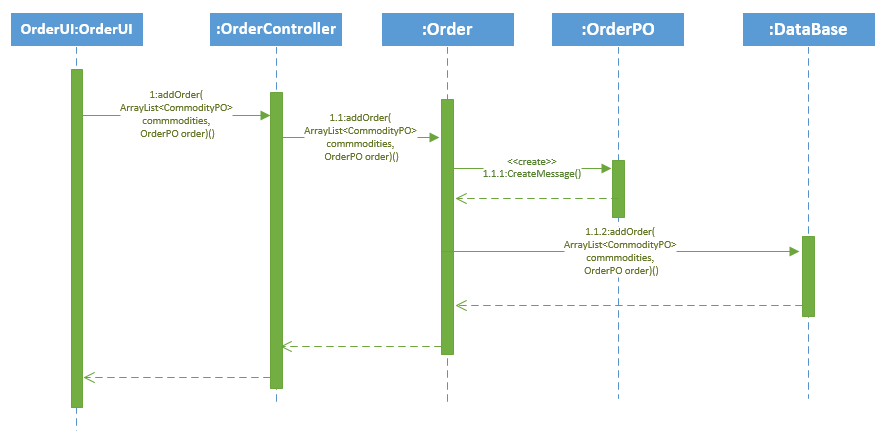
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderController.addOrder | 语法 | publicResultMessage addOrder(ArrayList<CommodityPO>commmodities, OrderPO order ); |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 添加订单，持久化更新涉及的对象的数据 |
| OrderController.inquireOrder | 语法 | Public OrderVO inquireOrder (String orderNumber, String senderName) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 查询订单，返回订单的可视化对象 |
| OrderController.confirmOrder | 语法 | public ConfirmState confirmOrder(String orderNumber) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 订单状态更新，持久化更新涉及的对象的数据 |
| OrderController.costAndTime | 语法 | public ResultMessage costAndTime(OrderVO order) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 |  |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Order.addOrder r(ArrayList<CommodityPO>commmodities, OrderPO order ) | 加入一个订单 | |
| Order.inquireOrder (String orderNumber, String senderName) | 查询一个订单 | |
| Order.confirmOrder(String orderNumber) | 签收一个订单 | |
| Order.costAndTime(OrderVO order) | 计算一个订单的费用和时间 | |

**Order的接口规范**

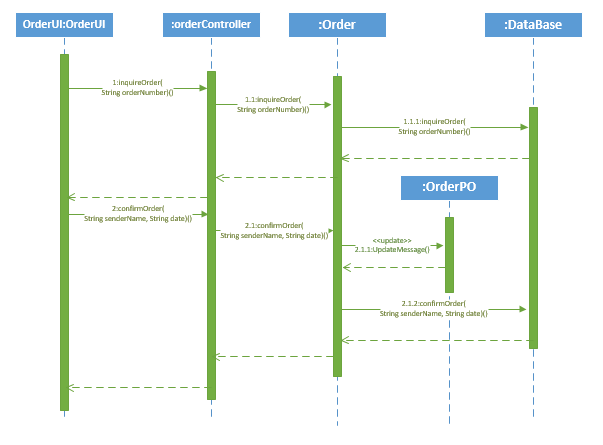
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Order.addOrder | 语法 | publicResultMessage addOrder(ArrayList<CommodityPO>commmodities, OrderPO order ); |
| 前置条件 | 待寄货物信息、收件人信息、寄件人信息符合输入规则 |
| 后置条件 |  |
| Order.inquireOrder | 语法 | Public OrderVO inquireOrder (String orderNumber, String senderName) |
| 前置条件 | 无 |
| 后置条件 | 根据输入的orderNumber查找相应的订单物流信息并返回。 |
| Order.confirmOrder | 语法 | public ConfirmState confirmOrder(String orderNumber) |
| 前置条件 | 订单号、收件人姓名符合输入规则 |
| 后置条件 | 查找相应的Order，更新订单的状态为已签收，生成收件单号。 |
| Order.costAndTime | 语法 | public ResultMessage costAndTime(OrderVO order) |
| 前置条件 | 被查询的订单已经存在。 |
| 后置条件 | 查找相应的Order，根据输入的订单信息，显示该订单的运费和预估到达时间。 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFactory.getOrderDataBase | 得到Order的服务的引用 | |
| OrderDateService.find  (OrderPO po) | 查找单一持久化对象 | |
| OrderDataService.insert  (OrderPO po) | 插入单一持久化对象 | |
| OrderDataService.update  (OrderPO po) | 更新单一持久化对象 | |

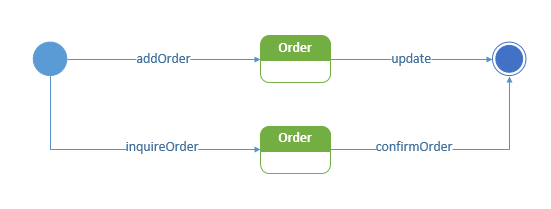
1. 业务逻辑层的动态模型

如下图表示了在快递物流系统中，当快递员想要添加一个订单的时候，订单管理业务逻辑处理的相关对象之间的协作



如下图表示了在快递物流系统中，当快递员想要确认接收一个订单的时候，订单管理业务逻辑处理的相关对象之间的协作





1. 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。

### Facilitybl模块

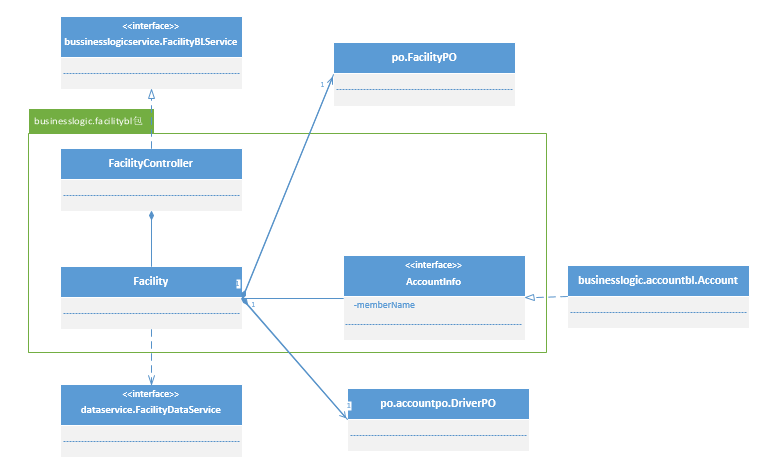
1. 模块概述

facilitybl模块负责实现营业厅运输工具管理界面所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.FacilityBLService接口。业务逻辑层和数据层之间添加dataservice.FacilityDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了FacilityController，这样FacilityController会将对营业厅运输工具的逻辑业务处理委托给Facility对象。FacilityPO是作为营业厅运输工具的持久化对象被添加到设计模型中去的。

Facilitybl模块的设计如图



Facilitybl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| FacilityController | 负责实现营业厅运输工具管理界面所需要的服务 |
| Facility | 营业厅运输工具的领域模型对象，拥有营业厅运输工具的信息，可以帮助完成营业厅运输工具管理界面所需要的服务 |

1. 模块内部类的接口规范

**FacilityController的接口规范**

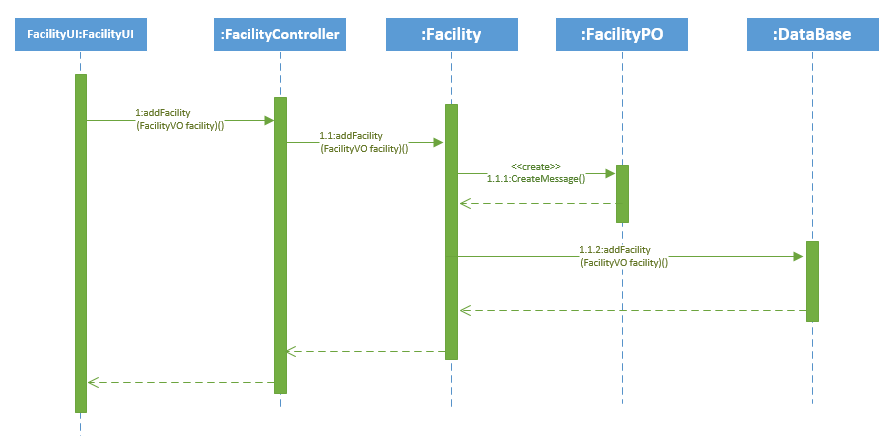
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| FacilityController.addFacility | 语法 | public ResultMessage addFacility  (FacilityVO facility) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 添加新的车辆信息，持久化更新涉及的对象的数据 |
| FacilityController.deleteFacility | 语法 | public ResultMessage deleteFacility (FacilityPO facility) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 删除车辆信息，持久化更新涉及的对象的数据 |
| FacilityController.modifyFacility | 语法 | public ResultMessage modifyFacility  (FacilityVO facility) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 更改车辆信息，持久化更新涉及的对象的数据 |
| FacilityController.findFacility | 语法 | public FacilityVO findFacility  (String Id) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 查找到该车辆信息 |
| FacilityController.addDriver | 语法 | PublicResultMessage addDriver  (DriverVO driver) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 添加新的司机信息，持久化更新涉及的对象的数据 |
| FacilityController.deleteDriver | 语法 | public ResultMessage deleteDriver (DriverVO driver) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 删除司机信息，持久化更新涉及的对象的数据 |
| FacilityController.modifyDriver | 语法 | public ResultMessage modifyDriver  (DriverVO driver) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 更改司机信息，持久化更新涉及的对象的数据 |
| FacilityController.findDriver | 语法 | public AccountVO findDriver  (int Id) |
| 前置条件 | 已经创建一个Facility领域对象，并且输入符合输入规则 |
| 后置条件 | 查找到该司机信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Facility.addFacility(FacilityVO facility) | 加入一辆运输车 | |
| Facility.deleteFacility(FacilityPO facility) | 删除一辆运输车 | |
| Facility.modifyFacility(FacilityVO facility) | 修改一辆运输车 | |
| Facility.findFacility(String id) | 查找一辆运输车 | |
| Facility.addDriver(DriverVO driver) | 加入一个司机 | |
| Facility.deleteDriver(DriverVO driver) | 删除一个司机 | |
| Facility.modifyDriver(DriverVO driver) | 修改一个司机 | |
| Facility.findDriver(String id) | 查找一个司机 | |

**Facility的接口规范**

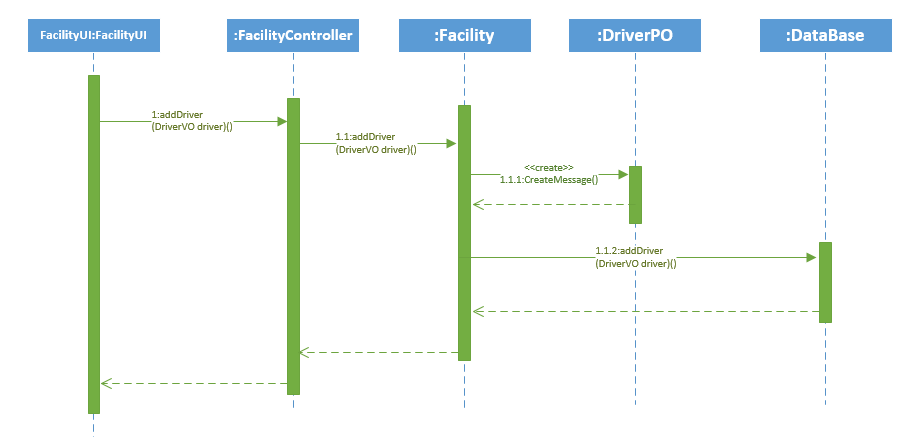
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Facility.addFacility | 语法 | public ResultMessage addFacility  (FacilityVO facility) |
| 前置条件 | 车辆信息输入合法 |
| 后置条件 | 添加新的车辆信息，持久化更新涉及的对象的数据 |
| Facility.deleteFacility | 语法 | public ResultMessage deleteFacility (FacilityPO facility) |
| 前置条件 | 车辆信息、ID输入合法 |
| 后置条件 | 删除车辆信息，持久化更新涉及的对象的数据 |
| Facility.modifyFacility | 语法 | public ResultMessage modifyFacility  (FacilityVO facility) |
| 前置条件 | 车辆信息输入合法 |
| 后置条件 | 更改车辆信息，持久化更新涉及的对象的数据 |
| Facility.findFacility | 语法 | public FacilityVO findFacility  (String id) |
| 前置条件 | 车辆信息、ID输入合法 |
| 后置条件 | 查找到该车辆信息 |
| Facility.addDriver | 语法 | PublicResultMessage addDriver  (DriverVO driver) |
| 前置条件 | 司机信息输入合法 |
| 后置条件 | 添加新的司机信息，持久化更新涉及的对象的数据 |
| Facility.deleteDriver | 语法 | public ResultMessage deleteDriver (DriverVO driver) |
| 前置条件 | 司机信息、ID输入合法 |
| 后置条件 | 删除司机信息，持久化更新涉及的对象的数据 |
| Facility.modifyDriver | 语法 | public ResultMessage modifyDriver  (DriverVO driver) |
| 前置条件 | 司机信息输入合法 |
| 后置条件 | 更改司机信息，持久化更新涉及的对象的数据 |
| Facility.findDriver | 语法 | public AccountVO findDriver  (String id) |
| 前置条件 | 司机信息、ID输入合法 |
| 后置条件 | 查找到该司机信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| FacilityDataService.  findFacility (String id) | 根据ID进行查找单一持久化对象 | |
| FacilityDataService.  findDriver (String id) | 根据ID进行查找单一持久化对象 | |
| FacilityDataService.  addFacility (FacilityPO po) | 添加单一化持久对象 | |
| FacilityDataService.  deleteFacility (FacilityPO po o) | 删除单一化持久对象 | |
| FacilityDataService.  modifyFacility (FacilityPO po) | 更改单一化持久对象 | |
| FacilityDataService.  getFacilityData() | 得到Facility数据的服务引用 | |

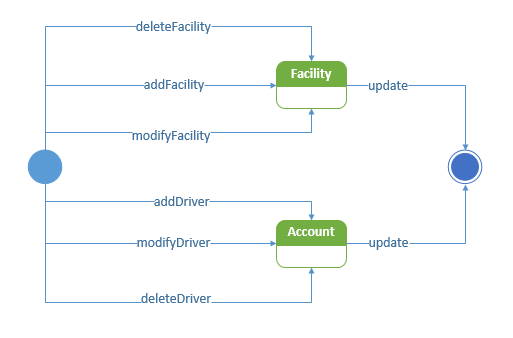
1. 业务逻辑层的动态模型

如下图表示了在快递物流系统中，当营业厅业务员想要新增一个车辆的时候，营业厅运输工具管理业务逻辑处理的相关对象之间的协作



如下图表示了在快递物流系统中，当营业厅业务员想要新增一个司机的时候，营业厅运输工具管理业务逻辑处理的相关对象之间的协作





1. 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。

### Branchbl模块

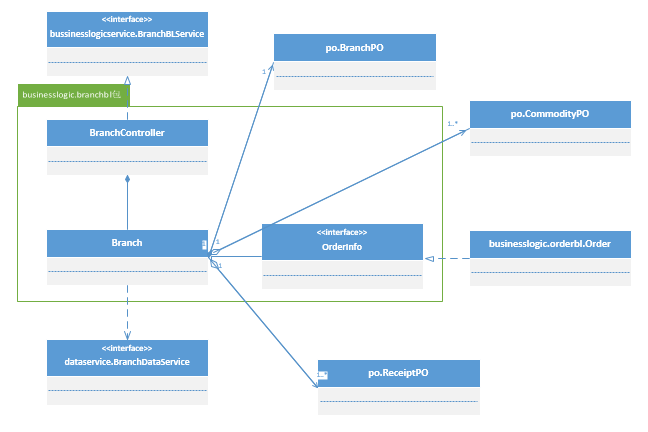
1. 模块概述

branchbl模块负责实现营业厅运输管理界面所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.BranchBLService接口。业务逻辑层和数据层之间添加dataservice.BranchDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了BranchController，这样BranchController会将对营业厅运输的逻辑业务处理委托给Branch对象。BranchPO是作为营业厅运输的持久化对象被添加到设计模型中去的。

Branchbl模块的设计如图



Branchbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| BranchController | 负责实现营业厅运输管理界面所需要的服务 |
| Branch | 营业厅运输的领域模型对象，拥有营业厅运输的信息，可以帮助完成营业厅运输管理界面所需要的服务 |

1. 模块内部类的接口规范

**BranchController的接口规范**

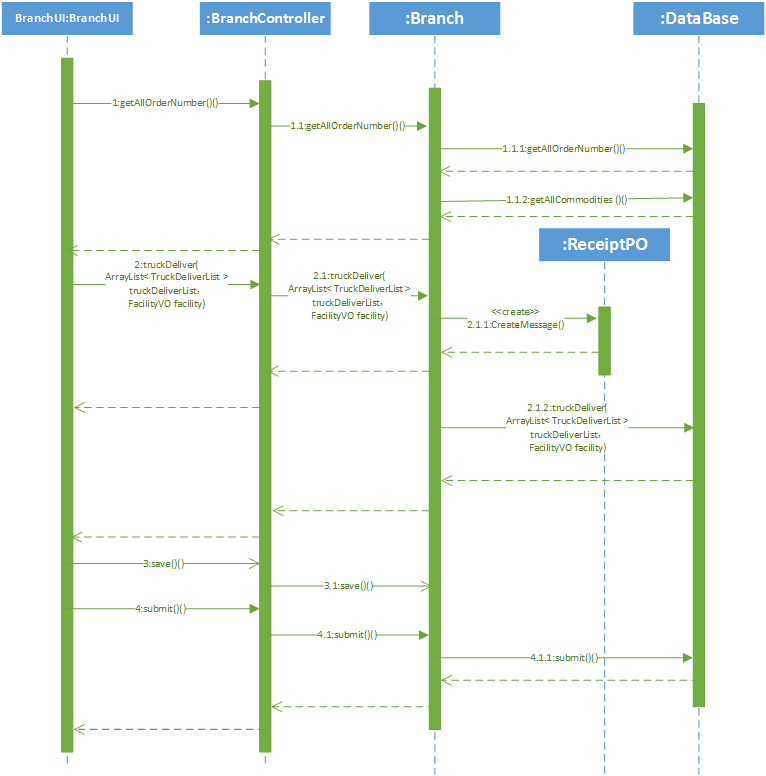
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BranchController.getDeliverID | 语法 | public String getDeliverID() |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 返回新建的装车单的ID |
| BranchController.getReceiptID | 语法 | public String getReceiptID() |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 返回新建的营业厅到达单的ID |
| BranchController.getExpressID | 语法 | public String getExpressID() |
| 前置条件 | 已经创建一个Branch领域对象 |
| 后置条件 | 返回新建的派件单的ID |
| BranchController.getAllCommidities | 语法 | public ArrayList<CommodityPO> getAllCommodities () |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 返回所有的待送的商品列表 |
| BranchController.getAllOrderNumber | 语法 | public ArrayList<String> getAllOrderNumber() |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 返回所有的到达商品或者等待派送商品的订单号 |
| BranchController.branchReceipt | 语法 | public BranchReceiptList branchReceipt(CommodityVO comoditiies,ArrayList<BranchList>branchReceiptList) |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 系统生成并显示货物到达营业厅的到达单。 |
| BranchController.branchDeliver | 语法 | public BranchDeliveryList branchDeliver(ArrayList<BranchList>branchReceiptList,AccountVO account) |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 系统生成派件单且更新快递物流信息。 |
| BranchController.submit | 语法 | public ResultMessage submit(BranchList branchList) |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| BranchController.save | 语法 | public BranchList save() |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 单据保存为草稿 |
| BranchController.truckDeliver | 语法 | public TruckDeliverList truckDeliver(ArrayList< TruckDeliverList > truckDeliverList，FacilityVO facility); |
| 前置条件 | 已经创建一个Branch领域对象，并且输入符合输入规则 |
| 后置条件 | 系统自动生成运费，并生成装车单，更新营业厅车辆信息。 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Branch.getDeliverID() | 得到装车单编号 | |
| Branch.getReceiptID() | 得到营业厅到达单编号 | |
| Branch.getExpressID() | 得到派件单编号 | |
| Branch.getAllCommidities() | 得到送达货物列表 | |
| Branch.getAllOrderNumber() | 得到所有订单编号列表 | |
| Branch.branchReceipt(CommodityVO comoditiies,ArrayList<BranchList>branchReceiptList) | 生成货物到达营业厅的到达单 | |
| Branch.branchDeliver(ArrayList<BranchList>branchReceiptList,AccountVO account) | 生成派件单 | |
| Branch.submit(BranchList branchList) | 提交单据等待审批 | |
| Branch.save() | 单据保存为草稿 | |
| Branch.truckDeliver(ArrayList< TruckDeliverList > truckDeliverList，FacilityVO facility) | 生成装车单 | |

**Branch的接口规范**

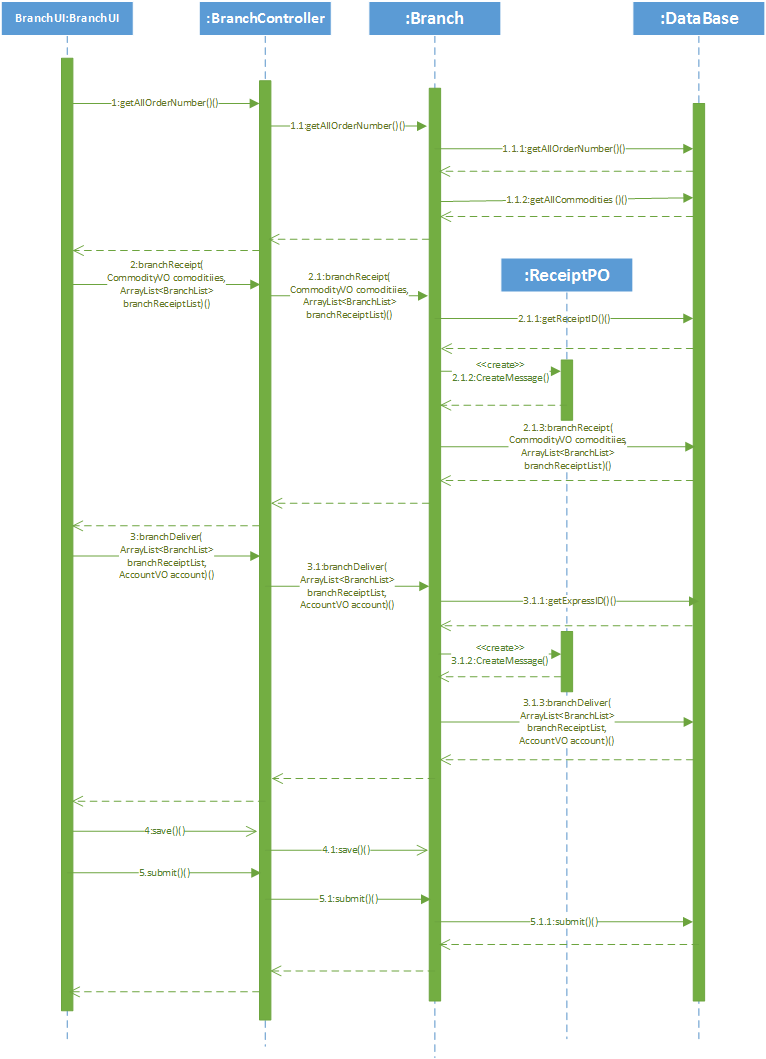
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Branch.getDeliverID | 语法 | | public String getDeliverID() |
| 前置条件 | | 得到数据，启动新建一个装车单 |
| 后置条件 | | 返回新建的装车单的ID |
| Branch.getReceiptID | 语法 | | public String getReceiptID() |
| 前置条件 | | 得到数据，启动新建一个营业厅到达单 |
| 后置条件 | | 返回新建的营业厅到达单的ID |
| Branch.getExpressID | 语法 | | public String getExpressID() |
| 前置条件 | | 得到数据，启动新建一个快递员派件单。 |
| 后置条件 | | 返回新建的派件单的ID |
| Branch.getAllCommidities | 语法 | | public ArrayList<CommodityPO> getAllCommodities () |
| 前置条件 | | 得到数据，需要显示等待派送的商品列表 |
| 后置条件 | | 返回所有的待送的商品列表 |
| Branch.getAllOrderNumber | 语法 | | public ArrayList<String> getAllOrderNumber() |
| 前置条件 | | 得到数据，需要显示到达商品或者等待派送商品的订单号 |
| 后置条件 | | 返回所有的到达商品或者等待派送商品的订单号 |
| Branch.branchReceipt | 语法 | | public BranchReceiptList branchReceipt(CommodityVO comoditiies,ArrayList<BranchList>branchReceiptList) |
| 前置条件 | | 有新的一批货物送达营业厅 |
| 后置条件 | | 系统生成并显示货物到达营业厅的到达单。 |
| Branch.branchDeliver | 语法 | | public BranchDeliveryList branchDeliver(ArrayList<BranchList>branchReceiptList,AccountVO account) |
| 前置条件 | | 营业厅生成了新的到达单据且有空闲的快递员 |
| 后置条件 | | 系统生成派件单且更新快递物流信息。 |
| Branch.submit | 语法 | | public ResultMessage submit(BranchList branchList) |
| 前置条件 | | 单据已经被制定出 |
| 后置条件 | | 单据被提交发送给经理进行审批 |
| Branch.save | 语法 | | public BranchList save() |
| 前置条件 | | 单据已经被制定出 |
| 后置条件 | | 单据保存为草稿 |
| Branch.truckDeliver | 语法 | | public TruckDeliverList truckDeliver(ArrayList< TruckDeliverList > truckDeliverList，FacilityVO facility); |
| 前置条件 | truckDeliverList符合输入规则 | |
| 后置条件 | | 系统自动生成运费，并生成装车单，更新营业厅车辆信息。 |
| 需要的服务（需接口） | | | |
| 服务名 | 服务 | | |
| DatabaseFactory.getBranchDataBase | 得到Branch的服务的引用 | | |
| BranchDataService.  find(String id) | 根据查找单一持久化对象 | | |
| BranchDataService.  insert(BranchPO po) | 插入单一持久化对象 | | |
| BranchDataService.  delete(BranchPO po) | 删除单一持久化对象 | | |
| BranchDataService.  update(BranchPO po) | 更新单一持久化对象 | | |
| BranchDataService.init | 初始化单一持久化对象 | | |

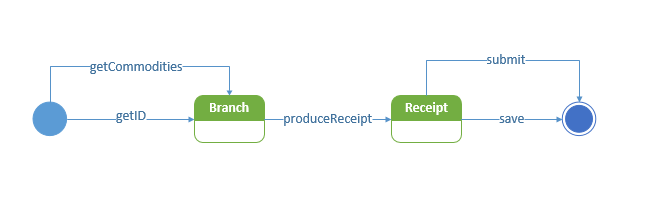
1. 业务逻辑层的动态模型

如下图表示了在快递物流系统中，当营业厅业务员进行装车信息管理的时候，营业厅运输管理业务逻辑处理的相关对象之间的协作



如下图表示了在快递物流系统中，当营业厅业务员进行接收派件货物的时候，营业厅运输管理业务逻辑处理的相关对象之间的协作





1. 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。

### Basebl模块

（1）模块概述

Basebl负责实现单据审批的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（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。比如在展示曾和业务逻辑层之间添加businesslogicservice.base blservice.BaseBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了BaseController，这样BaseConroller会将对用户管理的业务逻辑处理委托给Base对象。BasePO是作为的成本常量的持久化对象被添加到设计模型中去的。

Basebl的设计模式如图：



Basebl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| BaseController | 负责实现公司运营成本管理界面所需要的服务 |
| Base | 成本常量的领域模型对象，拥有成本常量的信息，可以帮助完成成本常量管理界面所需要的服务 |

（3）模块内类接口规范

**BaseController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BaseController.addBase | 语法 | public ResultMessage addBase(BaseVO vo) |
| 前置条件 | 已创建一个Base领域对象，添加的Base不重复 |
| 后置条件 | 调用Base领域的addBase方法 |
| BaseController.deletBase | 语法 | public ResultMessage deleteBase(String ID) |
| 前置条件 | 已创建一个Base领域对象，id符合输入规则 |
| 后置条件 | 调用Base领域的deleteBase方法 |
| BaseController.updateBase | 语法 | public ResultMessage updateBase (BaseVO base) |
| 前置条件 | 已创建一个Base领域对象 |
| 后置条件 | 调用Base领域的updateBase方法 |
| BaseController.getID() | 语法 | public String getID() |
| 前置条件 | 已经创建一个Base领域对象，界面提出添加base请求 |
| 后置条件 | 调用Base领域的getID方法 |
| BaseController.show(FindTypeBase baseType) | 语法 | public ArrayList<BaseVO vo>show(FindTypeBase baseType) |
| 前置条件 | 已经创建一个Base领域对象 |
| 后置条件 | 显示运营常量列表 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| BaseController.addBase | 添加一个常量 | |
| BaseController.deletBase | 删除一个常量 | |
| BaseController.updateBase | 更新一个常量 | |
| BaseController.show | 显示常量列表 | |
| BaseController.getID() | 获得一个常量编号 | |

**Base接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Base.addBase | 语法 | public ResultMessage addBase(BaseVO vo) |
| 前置条件 | 添加的Base不重复 |
| 后置条件 | 添加Base |
| Base.deleteBase | 语法 | public ResultMessage deleteBase  (String id) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的Base，根据输入的id删除该常量 |
| Base.getID | 语法 | public String getID() |
| 前置条件 | 界面提出添加Base申请 |
| 后置条件 | 返回一个新增Base的ID |
| Base.show | 语法 | public ArrayList<BaseVO>show(FindTypeBase baseType) |
| 前置条件 | 有可以显示在的常量 |
| 后置条件 | 显示当前类型的常量列表 |
| Base.updateBase | 语法 | public ResultMessage updateBase(BaseVObase) |
| 前置条件 | Base信息符合输入规则 |
| 后置条件 | 查找相应的Base，更改Base的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getBaseDataBase | 得到Base的服务的引用 | |
| BaseDateService.find  (BasePO po) | 查找单一持久化对象 | |
| BaseDataService.insert  (BasePO po) | 插入单一持久化对象 | |
| BaseDataService.delete  (BasePO po) | 删除单一持久化对象 | |
| BaseDataService.update  (UserPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，添加、删除、修改一个成本常量时时，常量业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。

（5）公司运营常量制定逻辑层的设计原理

利用委托式的控制风格，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。

### Receiptbl模块

（1）模块概述

Receiptbl负责实现单据审批的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（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。比如在展示曾和业务逻辑层之间添加businesslogicservice.receiptblservice.ReceiptBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReceiptController，这样ReceiptConroller会将对用户管理的业务逻辑处理委托给User对象。ReceiptPO是作为用户的持久化对象被添加到设计模型中去的。

Receiptbl的设计模式如图：



Receiptbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| ReceiptblController | 负责实现审批管理界面所需要的服务 |
| Receiptbl | 审批的领域模型对象，拥有审批单子的信息，可以帮助完成审批管理界面所需要的服务 |

1. 模块内部类的接口规范

**ReceiptblController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ReceiptblController.checkReceipt | 语法 | public ResultMessage checkReceipt (ReceiptVO vos) |
| 前置条件 | 已经创建一个Receipt领域对象,有需要审核的单据 |
| 后置条件 | 调用Receipt领域对象的checkReceipt方法 |
| ReceiptblController.updateReceipt | 语法 | public ResultMessage updateReceipt  (ReceiptVO receiptVO,ReceiptType receiptType) |
| 前置条件 | 已经创建一个Receipt领域对象,已经查看到单据内容 |
| 后置条件 | 调用Receipt领域对象的updateReceipt方法 |
| ReceiptBLController.passReceipt | 语法 | public ResultMessage passReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经创建一个Receipt领域对象,确认单据内容没有错误 |
| 后置条件 | 调用Receipt领域对象的passReceipt方法 |
| ReceiptBLController.dontPassReceipt | 语法 | public ResultMessage dontPassReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经创建一个Receipt领域对象,确认单据内容有错误 |
| 后置条件 | 调用Receipt领域对象的dontPassReceipt方法 |
| ReceiptBLController.showReceipt | 语法 | public ArrayList<ReceiptVO> showReceipt(ReceiptState receiptStates); |
| 前置条件 | 已经创建一个Receipt领域对象,有可以显示的单据 |
| 后置条件 | 调用Receipt领域对象的showReceipt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Receipt. checkReceipt | 查看单据内容 | |
| Receipt.updateReceipt | 修改单据内容 | |
| Receipt.passReceipt | 确认单据内容没有错误，通过单据审批 | |
| Receipt.dontPassReceipt | 确认单据内容有错误，不通过单据审批 | |
| Receipt.showReceipt | 显示单据列表 | |

**Receipt的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Receipt.checkReceipt | | 语法 | public ResultMessage checkReceipt (PersistentObject record) |
| 前置条件 | 有需要审核的单据 |
| 后置条件 | 查看单据内容 |
| Receipt. updateReceipt | | 语法 | public ResultMessage updateReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | 已经查看到单据内容 |
| 后置条件 | 修改单据内容 |
| Receipt. passReceipt | | 语法 | public ResultMessage passReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 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 |
| Receipt.dontPassReceipt | 语法 | | public ResultMessage dontPassReceipt  (ArrayList<ReceiptVO>VOs,ReceiptType receiptTypes) |
| 前置条件 | | 单据内容有错误 |
| 后置条件 | | 不通过单据，持久化更新涉及的对象的数据 |
| Receipt.showReceipt | 语法 | | public ArrayList<ReceiptVO> showReceipt(ReceiptState receiptStates); |
| 前置条件 | | 有可以进行显示的单据 |
| 后置条件 | | 显示单据列表 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| ReceiptDataService.  Insert(ReceiptPO po) | | 插入单一持久化对象 | |
| ReceiptDataService.update(ReceiptPO po) | | 修改单一持久化对象 | |
| ReceiptDataService.find(String id) | | 查找单一持久化对象 | |
| ReceiptDataService.show(ReceiptState receiptState) | | 显示单一持久化对象 | |
| ReceiptDataService.delete(String id) | | 删除单一持久化对象 | |
| ReceiptDataService.init() | | 初始化单一持久对象 | |

1. 业务逻辑层的动态模型

下图为单据处理的顺序图和状态图。



（5）公司运营常量制定逻辑层的设计原理

利用委托式的控制风格，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。

### Userbl模块

（1）模块概述

Userbl负责实现用户管理的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（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。比如在展示曾和业务逻辑层之间添加businesslogicservice.userblservice.UserBLService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UserConroller会将对用户管理的业务逻辑处理委托给User对象。UserPO是作为用户的持久化对象被添加到设计模型中去的。

Usebl的设计模式如图：



Userbl模块各个类的职责

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

（3）模块内类接口规范

**UserController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | 已创建一个User领域对象，password符合输入规则 |
| 后置条件 | 调用User领域的login方法 |
| UserController.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 已创建一个User领域对象，添加的User不重复 |
| 后置条件 | 调用User领域的addUser方法 |
| UserController.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | 已创建一个User领域对象，id符合输入规则 |
| 后置条件 | 调用User领域的deleteUser方法 |
| UserController.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 已创建一个User领域对象，用户名、密码、职位符合输入规则 |
| 后置条件 | 调用User领域的updateUser方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserController.login | 用户登录 | |
| UserController.addUser | 添加一个用户 | |
| UserController.deletUser | 删除一个用户 | |
| UserController.updateUser | 更新一个用户 | |

**User接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证的结果 |
| User.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 添加的User不重复 |
| 后置条件 | 添加User |
| User.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id删除该用户 |
| User.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 用户名、密码、职位符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id，更改该用户的权限 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getUserDataBase | 得到User的服务的引用 | |
| UserDateService.find  (UserPO po) | 查找单一持久化对象 | |
| UserDataService.insert  (UserPO po) | 插入单一持久化对象 | |
| UserDataService.delete  (UserPO po) | 删除单一持久化对象 | |
| userDataService.update  (UserPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，用户登录以及添加、删除、修改一个用户时，用户业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。





（5）用户逻辑层的设计原理

利用委托式的控制风格，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。

### Organizationbl模块

（1）模块概述

Organizationbl负责实现机构管理的功能性需求和非功能性需求，可参见快递物流系统需求规格说明文档和软件体系结构设计文档。

（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。比如在展示层和业务逻辑层之间添加businesslogicservice.organizationblservice.OrganizationBLService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OrganizationController，这样OrganizationConroller会将对机构管理的业务逻辑处理委托给Organization对象。OrganizationPO是作为机构的持久化对象被添加到设计模型中去的。

Organizationbl的设计模式如图：



Organizationbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| OrganizationController | 负责实现机构管理界面所需要的服务 |
| Organization | 机构的领域模型对象，拥有机构的信息，可以帮助完成机构管理界面所需要的服务 |

（3）模块内类接口规范

**OrganizationController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrganizationController.addOrganization | 语法 | public ResultMessage addOrganization (OrganizationVO vo) |
| 前置条件 | 已创建一个Organization领域对象，添加的Organization不重复 |
| 后置条件 | 调用Organization领域的addOrganization方法 |
| OrganizationController.deletOrganization | 语法 | public ResultMessage deleteOrganization (String ID) |
| 前置条件 | 已创建一个Organization领域对象，id符合输入规则 |
| 后置条件 | 调用Organization领域的deleteOrganization方法 |
| OrganizationController.updateOrganization | 语法 | public ResultMessage updateOrganization (OrganizationVO Organization) |
| 前置条件 | 已创建一个Organization领域对象 |
| 后置条件 | 调用Organization领域的updateOrganization方法 |
| OrganizationController.getID() | 语法 | public String getID() |
| 前置条件 | 已经创建一个Organization领域对象，界面提出添加Organization请求 |
| 后置条件 | 调用Organization领域的getID方法 |
| OrganizationController.show() | 语法 | public ArrayList<OrganizationVO vo>show() |
| 前置条件 | 已经创建一个Organization领域对象 |
| 后置条件 | 显示公司机构列表 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| OrganizationController.addOrganization | 添加一个机构 | |
| OrganizationController.deleteOrganization | 删除一个机构 | |
| OrganizationController.updateOrganization | 更新一个机构 | |
| OrganizationController.show | 显示机构列表 | |
| OrganizationController.getID | 获得一个机构编号 | |

**Organization接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Organization.addOrganization | 语法 | public ResultMessage addOrganization (OrganizationVO vo) |
| 前置条件 | 添加的Organization不重复 |
| 后置条件 | 添加Organization |
| Organization.deletOrganization | 语法 | public ResultMessage deleteOrganization (String id) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的Organization，根据输入的id删除该机构 |
| Organization.getID | 语法 | public String getID() |
| 前置条件 | 界面提出添加Organization申请 |
| 后置条件 | 返回一个新增Organization的ID |
| Organization.show | 语法 | public ArrayList<OrganizationVO>show() |
| 前置条件 | 有可以显示的机构 |
| 后置条件 | 显示机构列表 |
| Organization.updateOrganization | 语法 | public ResultMessage updateBase(OrganizationVO Organization) |
| 前置条件 | Organization信息符合输入规则 |
| 后置条件 | 查找相应的Organization，更改Organization的信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getOrganizationDataBase | 得到Organization的服务的引用 | |
| OrganizationDateService.find  (OrganizationPO po) | 查找单一持久化对象 | |
| OrganizationDataService.insert(OrganizationPO po) | 插入单一持久化对象 | |
| OrganizationDataService.delete(OrganizationPO po) | 删除单一持久化对象 | |
| OrganizationDataService.update(OrganizationPO po) | 更新单一持久化对象 | |

（4）业务逻辑层的动态模型

下列图片表明了快递物流管理系统中，添加、删除、修改一个机构时，机构管理业务逻辑处理的相关对象之间的协作。然后是它们各自的状态图。





（5）用户逻辑层的设计原理

利用委托式的控制风格，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

Transferbl模块

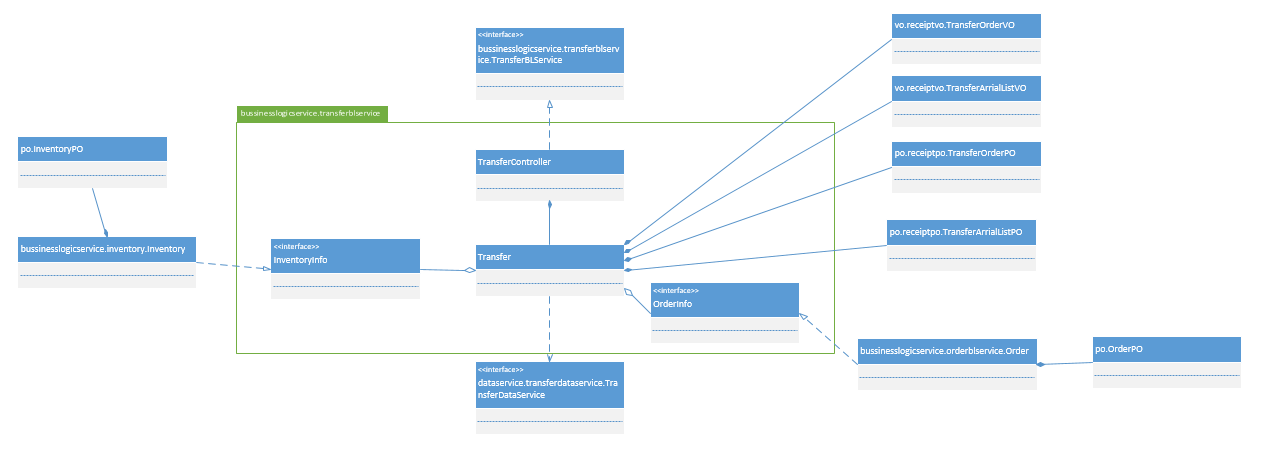
(1)模块概述

Transferbl模块负责实现中转中心机构的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

(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.transferblservice.TransferBLService接口。业务逻辑层和数据层之间添加dataservice.transferdataservice.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会将对中转中心的逻辑业务处理委托给Transfer对象。TransferOrderPO和TransferArrialListPO是作为中转中心装运单和中转中心到达单的持久化对象被分别添加到设计模型中去的。

Transferbl模块的设计如图



Transferbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| TransferController | 负责实现中转中心界面所需要的服务 |
| Transfer | 中转中心的领域模型对象，拥有中转中心装运单、到达单、货物的信息，可以帮助完成中转中心界面所需的服务 |

(3)Transferbl模块内部类的接口规范

**TransferController的接口规范**

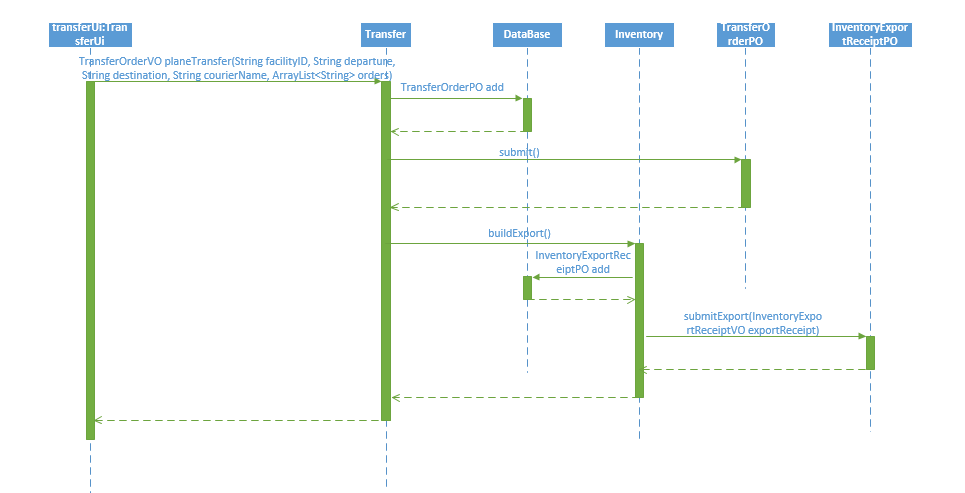
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| TransferController.getAllCommodities | 语法 | public ArrayList<CommodityVO> getAllCommodities() |
| 前置条件 | 得到数据，需要显示所有的待运货物。 |
| 后置条件 | 返回所有的待运货物列表 |
| TransferController.planeTransfer | 语法 | public TransferList planeTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | planeTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的飞机转运中转单，系统更新中转信息，提出出库申请 |
| TransferController.truckTransfer | 语法 | public TransferList truckTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | truckTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的火车转运中转单，系统更新中转信息，提出出库申请 |
| TransferController.trainTransfer | 语法 | public TransferListVO trainTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | trainTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的火车转运中转单，系统更新中转信息，提出出库申请 |
| TransferController.submit | 语法 | public TransferListVO submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| TransferController.save | 语法 | public TransferListVO save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| TransferController.receiptList | 语法 | PublicTranferListVO receiptList(  String transferID, String departure,  CommodityState state,  ArrayList<String> orders) |
| 前置条件 | 一批货物到达中转中心 |
| 后置条件 | 生成中转中心到达单，提出入库申请。 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Transfer.getAllCommodities() | 得到待运送商品列表 | |
| Transfer.planeTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) | 生成飞机转运中转单 | |
| Transfer.truckTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) | 生成汽车转运中转单 | |
| Transfer.trainTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) | 生成火车转运中转单 | |
| Transfer.submit() | 提交单据给总经理进行审批 | |
| Transfer.save() | 保存单据 | |
| Transfer.receiptList(  String transferID, String departure,  CommodityState state,  ArrayList<String> orders) | 生成中转中心到达单 | |

**Transferbl模块的接口规范**

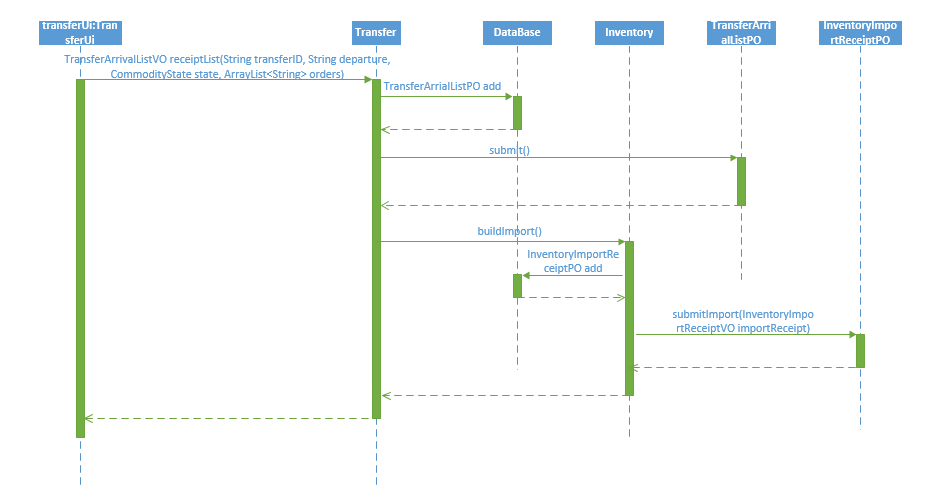
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Transfer.getAllCommodities | 语法 | public ArrayList<CommodityVO> getAllCommodities() |
| 前置条件 | 得到数据，需要显示所有的待运货物。 |
| 后置条件 | 返回所有的待运货物列表 |
| Transfer.planeTransfer | 语法 | public TransferList planeTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | planeTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的飞机转运中转单，系统更新中转信息，提出出库申请 |
| Transfer.truckTransfer | 语法 | public TransferList truckTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | truckTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的火车转运中转单，系统更新中转信息，提出出库申请 |
| Transfer.trainTransfer | 语法 | public TransferListVO trainTransfer(String facilityID, String departure, String destination, String courierName, ArrayList<String> orders) |
| 前置条件 | trainTransferList的内容输入符合规则 |
| 后置条件 | 系统自动计算此次转运费用，生成新的火车转运中转单，系统更新中转信息，提出出库申请 |
| Transfer.submit | 语法 | public TransferListVO submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| Transfer.save | 语法 | public TransferListVO save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| Transfer.receiptList | 语法 | PublicTranferListVO receiptList(  String transferID, String departure,  CommodityState state,  ArrayList<String> orders) |
| 前置条件 | 一批货物到达中转中心 |
| 后置条件 | 生成中转中心到达单，提出入库申请。 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFactory.getTransferDataBase | 得到Transfer的服务的引用 | |
| TransferDataService.  find(String id) | 根据查找单一持久化对象 | |
| TransferDataService.  insert(TransferPO po) | 插入单一持久化对象 | |
| TransferDataService.  delete(TransferPO po) | 删除单一持久化对象 | |
| TransferDataService.  update(TransferPO po) | 更新单一持久化对象 | |
| TransferDataService.init | 初始化单一持久化对象 | |

(4)业务逻辑层的动态模型

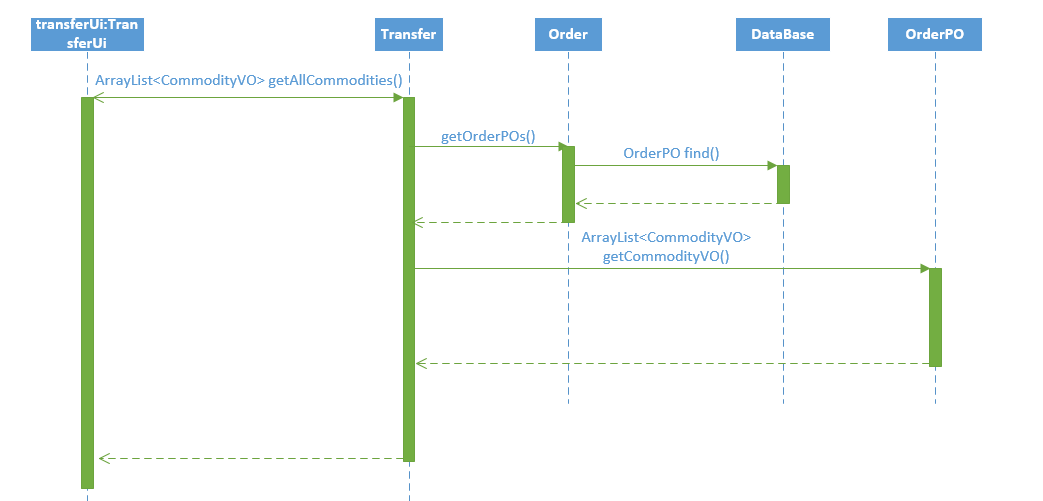
下图表明了快递物流系统中，得到生成飞机转运中转单时，用户业务逻辑处理的相关对象之间的协作。



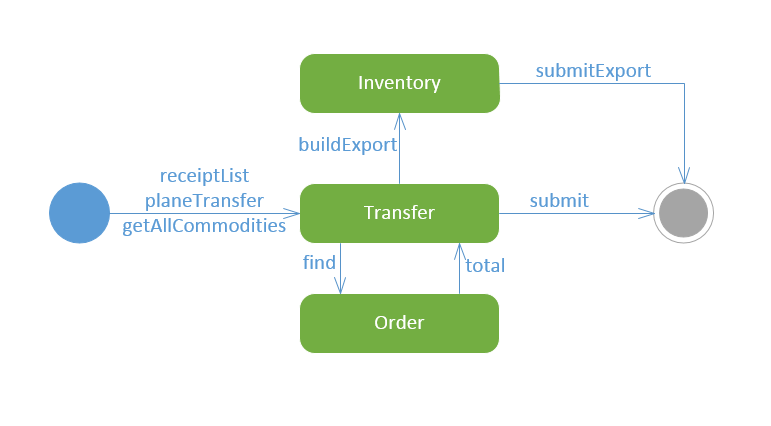
下图表明生成中转中心到达单的用户业务逻辑处理的相关对象之间的协作



下图表明得到待运送商品列表的用户业务逻辑处理的相关对象之间的协作



然后是生成飞机转运中转单、生成中转中心到达单、得到待运送商品列表的状态图



### OpeningStockbl模块

OpeningStockbl模块

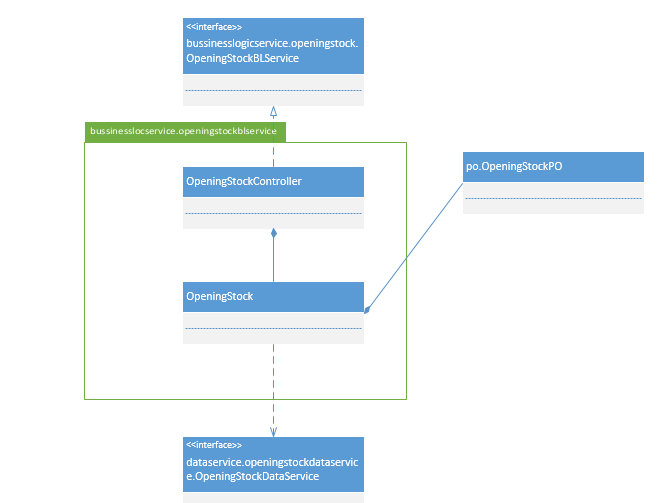
(1)模块概述

OpeningStockbl模块负责实现期初账目的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

(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.openingstockblservice.OpeningStockBLService接口。业务逻辑层和数据层之间添加dataservice.openingstockdataservice.OpeningStockDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OpeningStockController，这样OpeningStockController会将对期初账目的逻辑业务处理委托给OpeningStock对象。OpeningStockPO是作为期初账目的持久化对象被添加到设计模型中去的。

OpeningStockbl模块的设计如图



OpeningStockbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| OpeningStockController | 负责实现车辆和司机管理界面所需要的服务 |
| OpeningStock | 车辆和司机管理的领域模型对象，拥有车辆和司机的信息，可以帮助完成车辆和司机管理界面所需的服务 |

(3)模块内部类的接口规范

**OpeningStockController的接口规范**

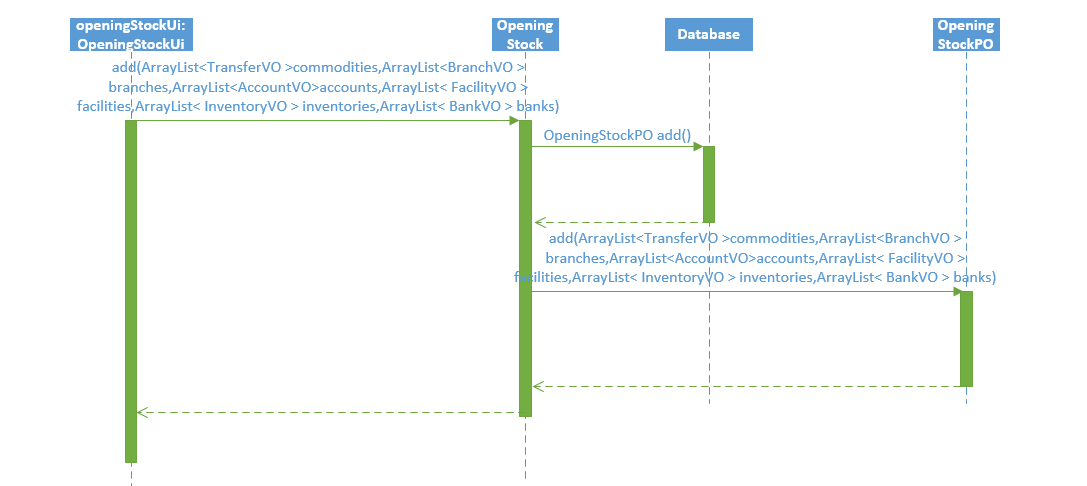
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **OpeningStockController.add** | 语法 | public ResultMessage add(  ArrayList<TransferVO >commodities,  ArrayList<BranchVO > branches,  ArrayList< AccountVO>accounts,  ArrayList< FacilityVO > facilities,  ArrayList< InventoryVO > inventories,  ArrayList< BankVO > banks); |
| 前置条件 | 机构、人员、车辆、库存、银行账户符合输入规范 |
| 后置条件 | 建立期初账目，持久化更新涉及的对象的数据 |
| **OpeningStockController.find** | 语法 | public ResultMessage find  (int id) |
| 前置条件 | 期初账目ID输入合法 |
| 后置条件 | 查找到该期初账目 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **OpeningStock.add(**  **ArrayList<TransferVO >commodities,ArrayList<BranchVO >branches,ArrayList<ccountVO>accounts,ArrayList<FacilityVO > facilities,ArrayList< InventoryVO >inventories,ArrayList<BankVO >banks)** | 建立期初账目 | |
| **OpeningStock.find(String id)** | 查找期初账目 | |

**OpeningStockbl模块的接口规范**

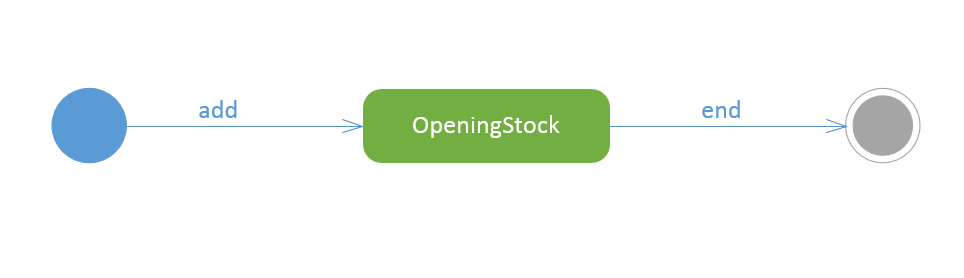
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **OpeningStock.add** | 语法 | public ResultMessage add(  ArrayList<TransferVO >commodities,  ArrayList<BranchVO > branches,  ArrayList< AccountVO>accounts,  ArrayList< FacilityVO > facilities,  ArrayList< InventoryVO > inventories,  ArrayList< BankVO > banks); |
| 前置条件 | 机构、人员、车辆、库存、银行账户符合输入规范 |
| 后置条件 | 建立期初账目，持久化更新涉及的对象的数据 |
| **OpeningStock.find** | 语法 | public ResultMessage find  (int id) |
| 前置条件 | 期初账目ID输入合法 |
| 后置条件 | 查找到该期初账目 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **OpeningStockDataService.**  **add(OpeningStockPO po)** | 添加单一持久化对象 | |
| **OpeningStockDataService.**  **find(int id)** | 根据ID进行查找单一持久化对象 | |
| **OpeningStockDataService.**  **init** | 初始化持久数据 | |

(4)业务逻辑层的动态模型

下图表明了快递物流系统中，添加期初账目时，用户业务逻辑处理的相关对象之间的协作。



然后是添加期初账目的状态图



### Fundbl模块

Fundbl模块

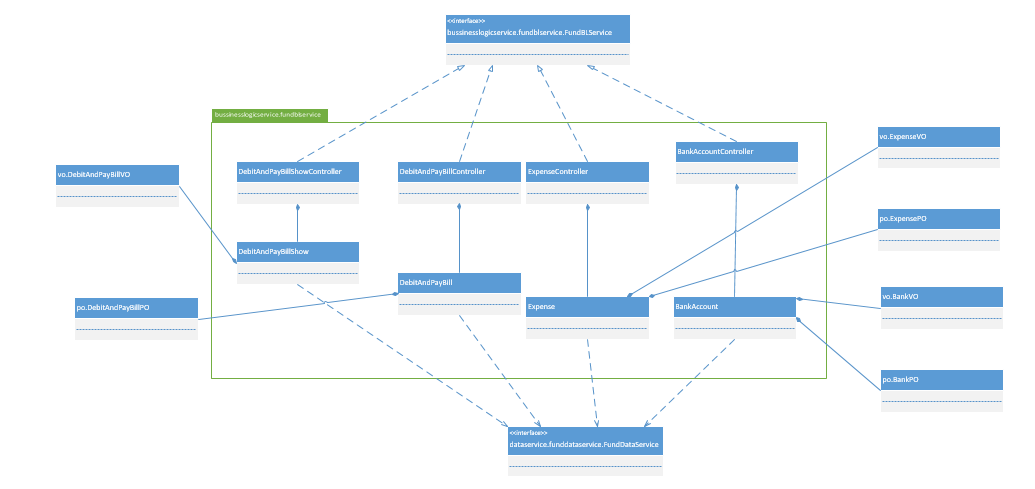
(1)模块概述

Fundbl模块负责实现营业厅收款单管理、结算管理、成本管理、银行账户管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

(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.fundblservice.FundBLService接口。业务逻辑层和数据层之间添加dataservice.fundblservice.FundDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了DebitAndPayBillController、BankController、ExpenseController和DebitAndPayBillShowController，这样DebitAndPayBillController会将对结算管理、成本管理的逻辑业务处理委托给DebitAndPayBill对象，DebitAndPayBillShowController会将对结算管理、成本管理显示的逻辑业务处理委托给DebitAndPayBillShow对象，BankController会将对银行账户管理的逻辑业务处理委托给Bank对象，ExpenseController会将对营业厅收款单管理的逻辑业务处理委托给Expense对象。DebitAndPayBillPO、BankPO、ExpensePO是作为入款收款单、银行账户、营业厅收款单的持久化对象被添加到设计模型中去的。

Fundbl模块的设计如图



Fundbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| DebitAndPayBillController | 负责实现结算管理界面和成本管理界面所需要的服务 |
| DebitAndPayBill | 入款单和收款单的领域模型对象，拥有入款单和收款单的信息，可以帮助完成入款单和收款的添加、删除、查询、修改服务。 |
| DebitAndPayBillShowController | 负责实现结算管理界面和成本管理界面显示所需要的服务 |
| DebitAndPayBillShow | 入款单和收款单显示的领域模型对象，拥有入款单和收款单的信息，可以帮助完成入款单和收款单显示的服务。 |
| BankController | 负责实现银行账户界面所需要的服务 |
| Bank | 银行账户的领域模型对象，拥有账户的信息，可以帮助完成银行账户管理界面所需的服务 |
| ExpenseController | 负责实现建立营业厅收款单界面所需要的服务 |
| Expense | 收款单管理的领域模型对象，拥有营业厅收款单的信息，可以帮助完成营业厅收款单的添加、删除、查询、修改服务服务。 |

(3)模块内部类的接口规范

**DebitAndPayBillController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBillController.find(String ID)** | 语法 | public DebitAndPayBillVO find(String ID) |
| 前置条件 | 单据ID输入合法 |
| 后置条件 | 查找到该单据 |
| **DebitAndPayBillController.addDebitAndPayBill** | 语法 | public ResultMessage addDebitAndPayBill(String operatorID , String operatorName ) |
| 前置条件 | 单据ID、收款人（付款人）姓名输入合法 |
| 后置条件 | 添加新的单据，持久化更新涉及的对象的数据 |
| **DebitAndPayBillController. updateDraft** | 语法 | public ResultMessage updateDraft  (DebitAndPayBillVO vo) |
| 前置条件 | 单据内容输入合法 |
| 后置条件 | 更新单据信息，持久化更新涉及的对象的数据 |
| **DebitAndPayBillController.findOpeningStock** | 语法 | public ResultMessage submitDraft(String ID) |
| 前置条件 | 单据ID输入合法 |
| 后置条件 | 提交信息，持久化更新涉及的对象的数据 |
| **DebitAndPayBillController.submit** | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| **DebitAndPayBillController.save** | 语法 | public ResultMessage save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **DebitAndPayBill. getPayID()** | 得到新建付款单的ID | |
| **DebitAndPayBill.getExpenseID()** | 得到新建收款单的ID | |
| **DebitAndPayBill.find(String ID)** | 根据ID查找到一条单据 | |
| **DebitAndPayBill.addDebitAndPayBill** | 添加新的单据 | |
| **DebitAndPayBill. updateDraft** | 修改草稿单据信息 | |
| **DebitAndPayBill. submitDraft** | 提交草稿状态的单据 | |
| **DebitAndPayBill.submit** | 提交单据进行审批 | |
| **DebitAndPayBill.save** | 将单据保存成草稿状态 | |

**DebitAndPayBill模块的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBill. getPayID** | 语法 | public String getPayID() |
| 前置条件 | 无 |
| 后置条件 | 得到新建付款单的ID |
| **DebitAndPayBill.getExpenseID** | 语法 | Public String getExpenseID() |
| 前置条件 | 无 |
| 后置条件 | 得到新建收款单的ID |
| **DebitAndPayBill.find** | 语法 | public DebitAndPayBillVO find(String ID) |
| 前置条件 | 单据ID输入合法 |
| 后置条件 | 查找到该单据 |
| **DebitAndPayBill.addDebitAndPayBill** | 语法 | public ResultMessage addDebitAndPayBill(String operatorID , String operatorName ) |
| 前置条件 | 单据ID、收款人（付款人）姓名输入合法 |
| 后置条件 | 添加新的单据，持久化更新涉及的对象的数据 |
| **DebitAndPayBill. updateDraft** | 语法 | public ResultMessage updateDraft  (DebitAndPayBillVO vo) |
| 前置条件 | 单据内容输入合法 |
| 后置条件 | 更新单据信息，持久化更新涉及的对象的数据 |
| **DebitAndPayBill. submitDraft** | 语法 | public ResultMessage submitDraft(String ID) |
| 前置条件 | 单据ID输入合法 |
| 后置条件 | 提交信息，持久化更新涉及的对象的数据 |
| **DebitAndPayBill.submit** | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| **DebitAndPayBill.save** | 语法 | public ResultMessage save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **DebitAndPayBillDataService.**  **getID()** | 获得单据的ID编号 | |
| **DebitAndPayBillDataService.**  **addDebitAndPayBill (addDebitAndPayBillPO po)** | 添加单一化持久对象 | |
| **DebitAndPayBillDataService.**  **findDebitAndPayBill(String ID)** | 根据ID查找单一化持久对象 | |
| **DebitAndPayBillDataService.**  **updateDebitAndPayBill (DebitAndPayBillPO po)** | 更改单一化持久对象 | |
| **DebitAndPayBillDataService.**  **getDebitAndPayBillData** | 得到DebitAndPayBill数据的服务引用 | |

**DebitAndPayBillShowController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBillShowController. showPay** | 语法 | public ArrayList<DebitAndPayBillVO> showPay() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController. showExpense** | 语法 | PublicArrayList<DebitAndPayBillVO>  showExpense() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController. showPayApproving** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayApproving() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController. showExpenseApproving** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseApproving() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showPayPass** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayPass() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showExpensePass** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpensePass() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showPayFailure** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayFailure() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showExpenseFailure** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseFailure() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showPayDraft** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayDraft() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **DebitAndPayBillShowController.**  **showExpenseDraft** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseDraft() |
| 前置条件 | 已创建一个DebitAndPayBillShow对象 |
| 后置条件 | 调用DebitAndPayBillShow的方法 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **DebitAndPayBillShow. showPay()** | 显示全部的付款单 | |
| **DebitAndPayBillShow. showExpense()** | 显示全部的收款单 | |
| **DebitAndPayBillShow. showPayApproving()** | 显示全部在审核的付款单 | |
| **DebitAndPayBillShow. showExpenseApproving()** | 显示全部在审核的收款单 | |
| **DebitAndPayBillShow. showPayPass()** | 显示全部通过审核的付款单 | |
| **DebitAndPayBillShow. showExpensePass()** | 显示全部通过审核的收款单 | |
| **DebitAndPayBillShow. showExpenseFailure()** | 显示全部没有通过审批的付款单 | |
| **DebitAndPayBillShow. showExpenseFailure()** | 显示全部没有通过审批的收款单 | |
| **DebitAndPayBillShow. showPayDraft()** | 显示显示全部草稿状态的付款单 | |
| **DebitAndPayBillShow. showExpenseDraft()** | 显示显示全部草稿状态的收款单 | |

**DebitAndPayBillShow的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBillShow. showPay** | 语法 | public ArrayList<DebitAndPayBillVO> showPay() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部的付款单 |
| **DebitAndPayBillShow. showExpense** | 语法 | Public ArrayList<DebitAndPayBillVO>  showExpense() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部的收款单 |
| **DebitAndPayBillShow. showPayApproving** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayApproving() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部在审核的付款单 |
| **DebitAndPayBillShow. showExpenseApproving** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseApproving() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部在审核的收款单 |
| **DebitAndPayBillShow. showPayPass** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayPass() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部通过审核的付款单 |
| **DebitAndPayBillShow. showExpensePass** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpensePass() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部通过审核的收款单 |
| **DebitAndPayBillShow. showPayFailure** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayFailure() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部没有通过审批的付款单 |
| **DebitAndPayBillShow. showExpenseFailure** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseFailure() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示全部没有通过审批的收款单 |
| **DebitAndPayBillShow. showPayDraft** | 语法 | public ArrayList<DebitAndPayBillVO>  showPayDraft() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示显示全部草稿状态的付款单 |
| **DebitAndPayBillShow. showExpenseDraft** | 语法 | public ArrayList<DebitAndPayBillVO>  showExpenseDraft() |
| 前置条件 | 进入成本管理/结算管理界面 |
| 后置条件 | 显示显示全部草稿状态的收款单 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **DebitAndPayBillDataService.show(ReceiptType type)** | 得到所有单据列表 | |

**BankAccountController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **BankAccountController.show** | 语法 | public ArrayList<BankAccoutVO> show(); |
| 前置条件 | 进入银行账户管理界面 |
| 后置条件 | 显示当前公司的全部银行账户 |
| **BankAccountController.add** | 语法 | public ResultMessage add(BankAccoutVO vo); |
| 前置条件 | 银行账户信息输入正确 |
| 后置条件 | 添加一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccountController.delete** | 语法 | public ResultMessage delete(String ID); |
| 前置条件 | 账户ID输入正确 |
| 后置条件 | 删除一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccountController.update** | 语法 | public ResultMessage update(BankAccoutVO vo); |
| 前置条件 | 银行账户信息输入正确 |
| 后置条件 | 修改一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccountController.find** | 语法 | public ArrayList<BankAccoutVO> find(  String keywords,FindTypeAccount type); |
| 前置条件 | 关键字输入正确，搜索类型输入正确 |
| 后置条件 | 显示找到的银行账户信息 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **BankAccount.getID()** | 获得一个银行账户ID | |
| **BankAccount.show()** | 显示当前公司的全部银行账户 | |
| **BankAccount.add(BankAccoutVO vo)** | 增加一个持久化对象 | |
| **BankAccount.delete(String ID)** | 删除一个持久化对象 | |
| **BankAccount.update(BankAccoutVO vo)** | 修改一个持久化对象 | |
| **BankAccount.find(**  **String keywords,FindTypeAccount type)** | 根据关键字、搜索类型查找银行账户信息 | |

**BankAccount的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **BankAccount.getID** | 语法 | public String getID(); |
| 前置条件 | 无 |
| 后置条件 | 获得一个银行账户ID |
| **BankAccount.show** | 语法 | public ArrayList<BankAccoutVO> show() |
| 前置条件 | 进入银行账户管理界面 |
| 后置条件 | 显示当前公司的全部银行账户 |
| **BankAccount.add** | 语法 | public ResultMessage add(BankAccoutVO vo) |
| 前置条件 | 银行账户信息输入正确 |
| 后置条件 | 添加一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccount.delete** | 语法 | public ResultMessage delete(String ID) |
| 前置条件 | 账户ID输入正确 |
| 后置条件 | 删除一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccount.update** | 语法 | public ResultMessage update(BankAccoutVO vo) |
| 前置条件 | 银行账户信息输入正确 |
| 后置条件 | 修改一个银行账户，持久化更新涉及的对象的数据 |
| **BankAccount.find** | 语法 | public ArrayList<BankAccoutVO> find(  String keywords,FindTypeAccount type) |
| 前置条件 | 关键字输入正确，搜索类型输入正确 |
| 后置条件 | 显示找到的银行账户信息 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **BankAccountDataService insert(BankPO po)** | 增加一个持久化对象 | |
| **BankAccountDataService delete(String ID)** | 根据ID删除一个持久化对象 | |
| **BankAccountDataService update(BankPO PO)** | 更新持久化对象 | |
| **BankAccountDataService find(String keywords,FindTypeAccount bankAccoutType)** | 以keywords查看账户信息，type是查找的方式 | |

**ExpenseController的接口规范**

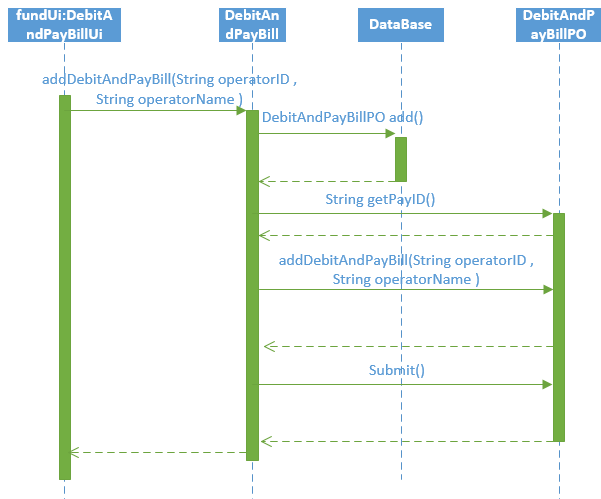
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **ExpenseController.show** | 语法 | public ArrayList<ExpenseVO> show() |
| 前置条件 | 进入营业厅收款单管理界面 |
| 后置条件 | 显示全部收款记录 |
| **ExpenseController.add** | 语法 | public ResultMessage add(ExpenseVOvo) |
| 前置条件 | 收款记录信息输入正确 |
| 后置条件 | 添加一个收款记录，持久化更新涉及的对象的数据 |
| **ExpenseController.delete** | 语法 | public ResultMessage delete(String ID) |
| 前置条件 | 收款记录ID输入正确 |
| 后置条件 | 删除一个添加收款记录，持久化更新涉及的对象的数据 |
| **ExpenseController.update** | 语法 | public ResultMessage update(ExpenseVOvo) |
| 前置条件 | 收款记录信息输入正确 |
| 后置条件 | 修改一个收款记录，持久化更新涉及的对象的数据 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **Expense.getID** | 获得一个收款记录ID | |
| **Expense.show** | 显示当前公司的全部收款记录 | |
| **Expense.add** | 增加一个持久化对象 | |
| **Expense.delete** | 删除一个持久化对象 | |
| **Expense.update** | 修改一个持久化对象 | |

**Expense的接口规范**

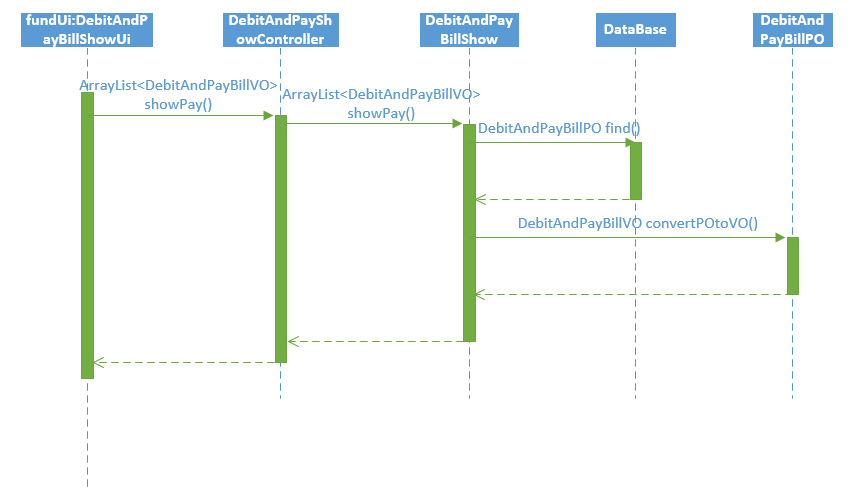
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
|  | 语法 | public String getID(); |
| 前置条件 | 提出创建/新增收款记录 |
| 后置条件 | 返回一条收款记录的ID |
|  | 语法 | public ArrayList<ExpenseVO> show() |
| 前置条件 | 进入营业厅收款单管理界面 |
| 后置条件 | 显示全部收款记录 |
|  | 语法 | public ResultMessage add(ExpenseVOvo) |
| 前置条件 | 收款记录信息输入正确 |
| 后置条件 | 添加一个收款记录，持久化更新涉及的对象的数据 |
|  | 语法 | public ResultMessage delete(String ID) |
| 前置条件 | 收款记录ID输入正确 |
| 后置条件 | 删除一个添加收款记录，持久化更新涉及的对象的数据 |
|  | 语法 | public ResultMessage update(ExpenseVOvo) |
| 前置条件 | 收款记录信息输入正确 |
| 后置条件 | 修改一个收款记录，持久化更新涉及的对象的数据 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **ExpenseDataService.insert(BankPO po)** | 增加一个持久化对象 | |
| **ExpenseDataService.delete(String ID)** | 根据ID删除一个持久化对象 | |
| **ExpenseDataService. update(BankPO PO)** | 更新持久化对象 | |
| **ExpenseDataService. show(String ID);** | 根据ID查看一个持久化对象 | |

(4)业务逻辑层的动态模型

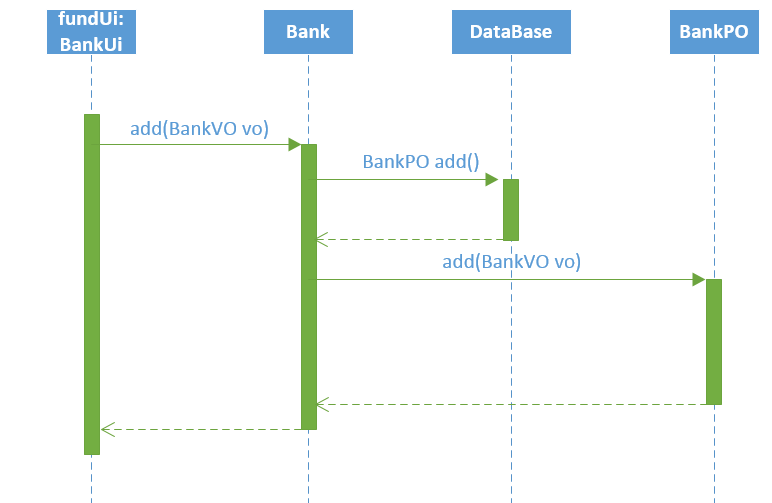
下图表明了快递物流系统中，添加收款单（付款单）时，用户业务逻辑处理的相关对象之间的协作。



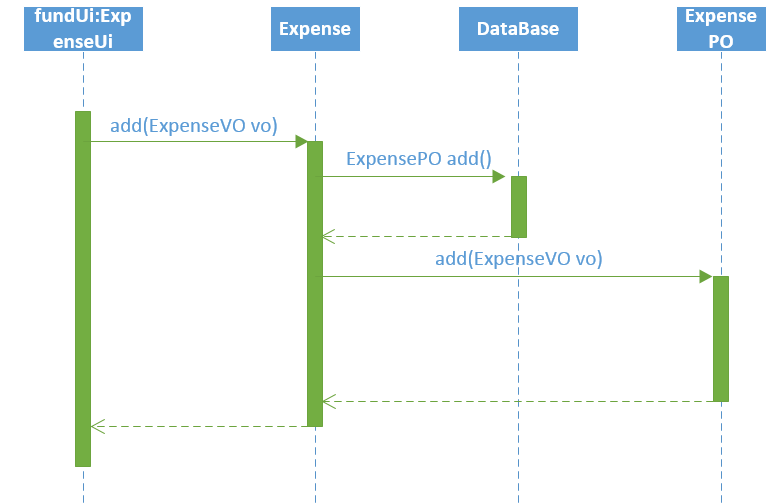
下图表明了快递物流系统中，显示全部的付款单信息时，用户业务逻辑处理的相关对象之间的协作。



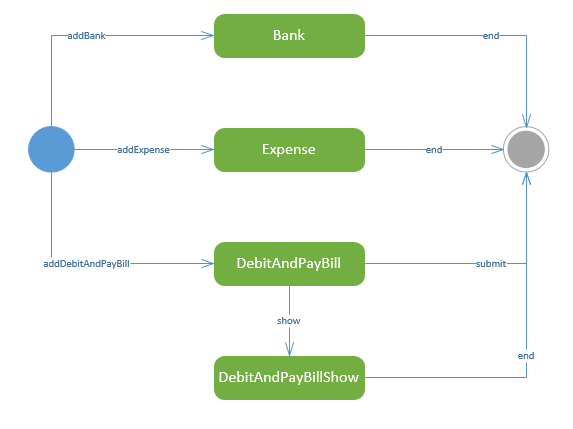
下图表明了快递物流系统中，添加一个银行账户时，用户业务逻辑处理的相关对象之间的协作。



下图表明了快递物流系统中，添加一个收款记录时，用户业务逻辑处理的相关对象之间的协作。



然后是添加银行账户、添加收款记录、添加收款单并显示部分功能的状态图



### Inventorybl模块

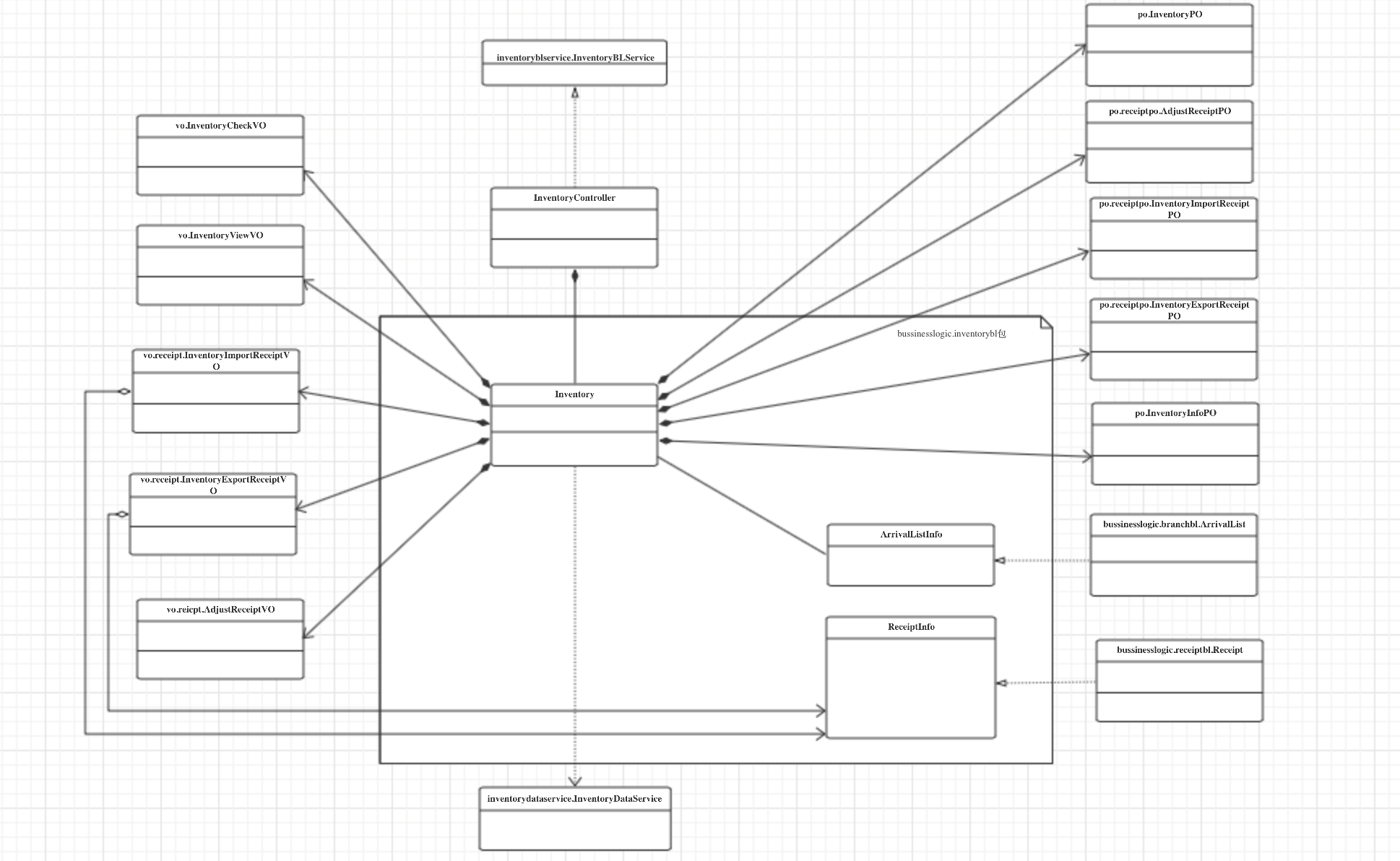
1. 模块概述

orderbl模块负责实现仓库管理所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.InventoryBLService接口。业务逻辑层和数据层之间添加dataservice.InventoryDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InventoryController，这样InventoryrController会将对订单的逻辑业务处理委托给Inventory对象。InventoryInfoPO,InventoryPO,InventoryImportReceiptPO,InventoryExportReceiptPO,AdjustReceiptPO是作为仓库相关信息的持久化对象被添加到设计模型中去的。

inventorybl模块的设计如图



Inventory模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| InventoryController | 负责实现仓库管理界面所需要的服务 |
| Inventory | 仓库的领域模型对象，拥有仓库的信息，可以帮助完成仓库管理界面所需要的服务 |

(3)模块内部类的接口规范

InventoryController的接口规范

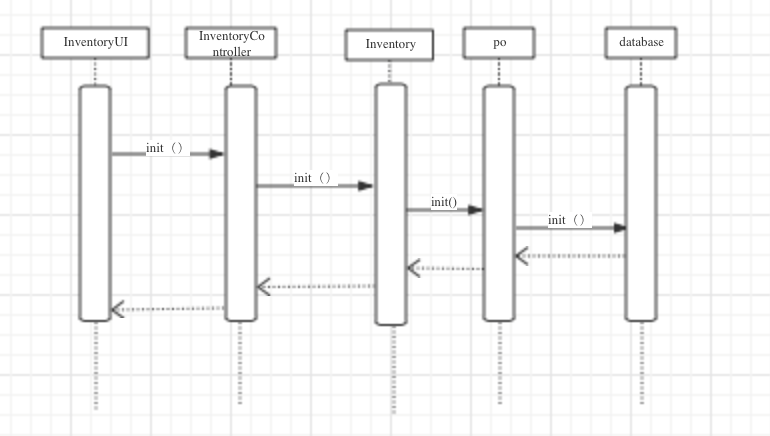
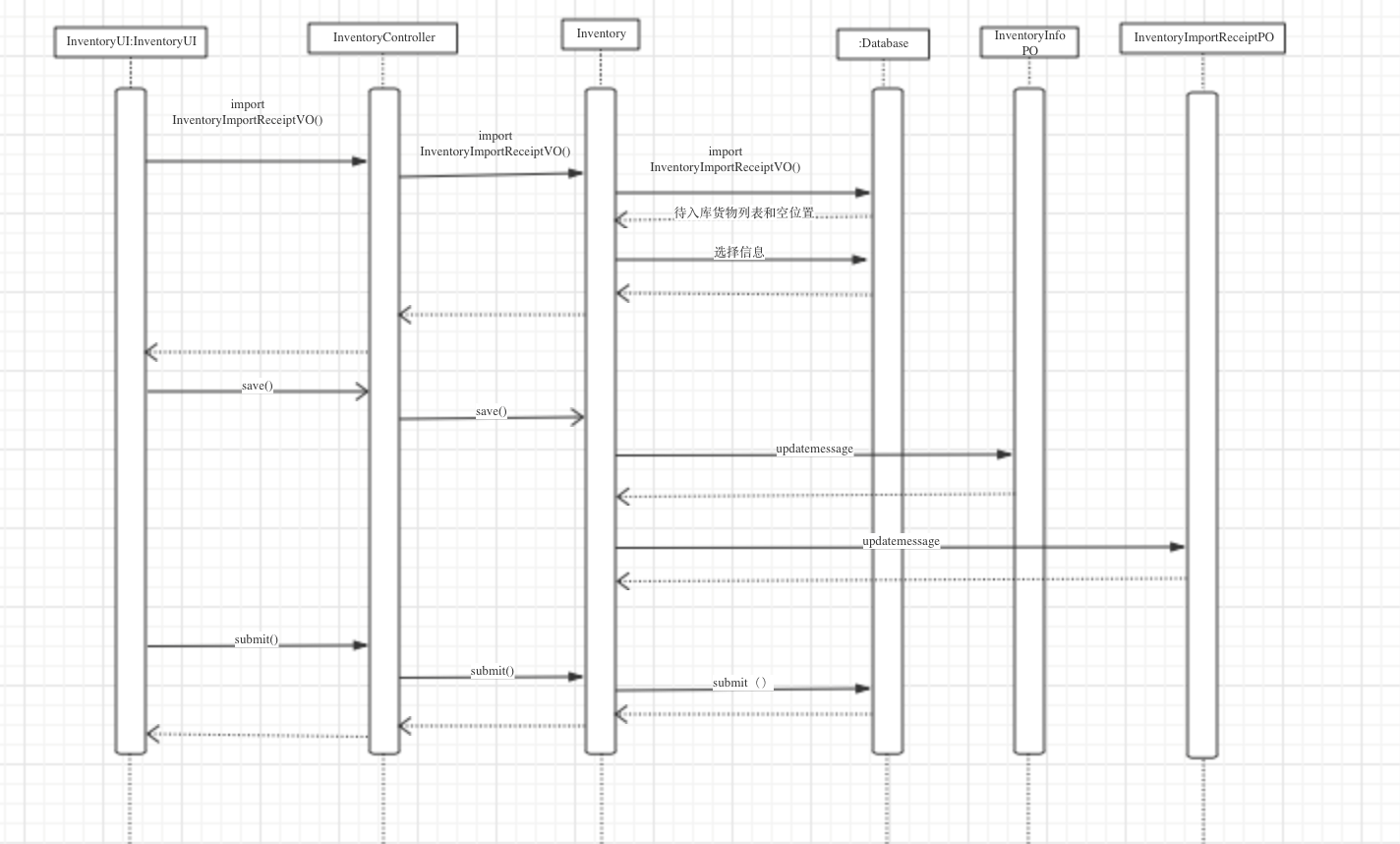
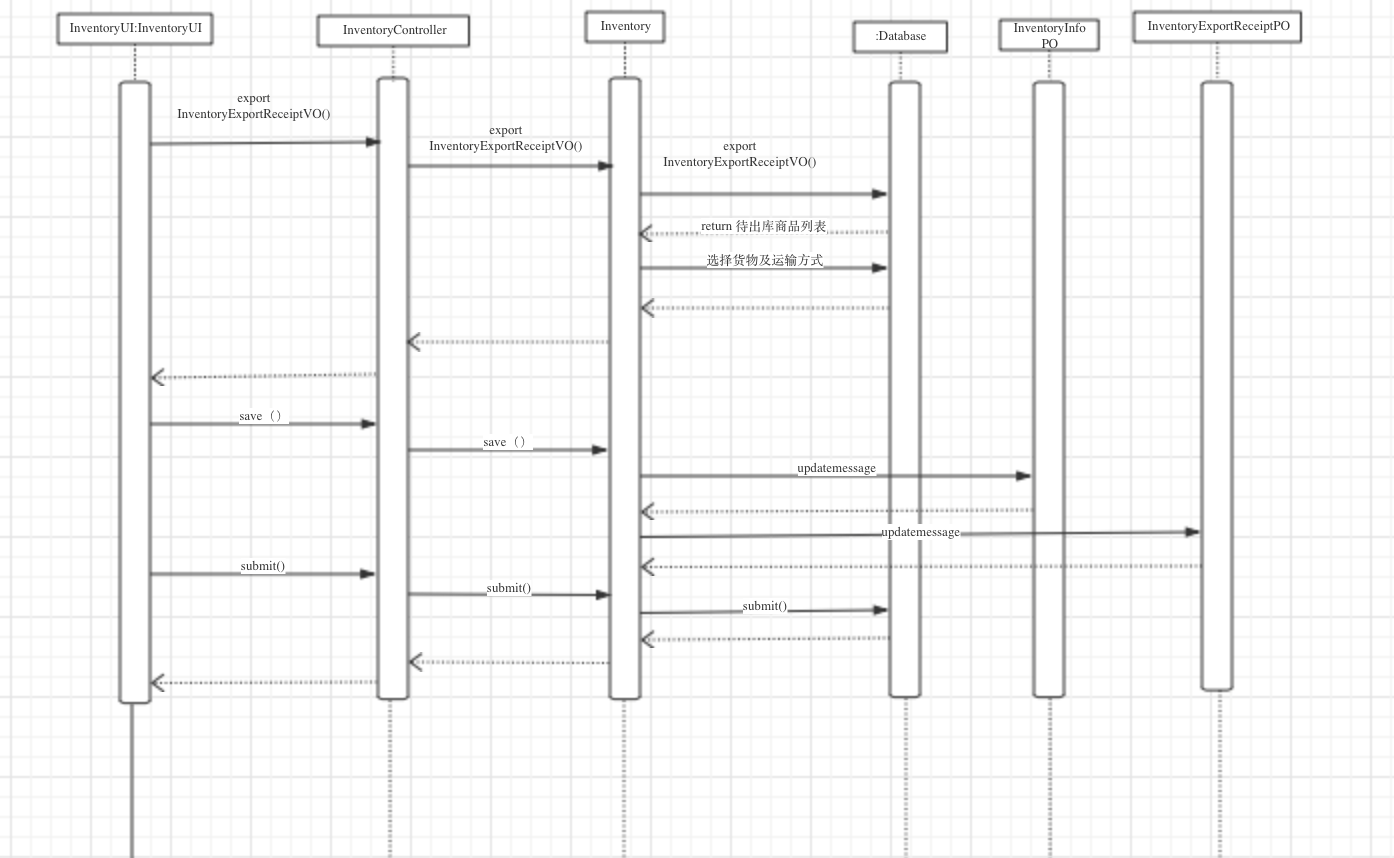
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **InventoryController.viewInventory** | 语法 | public InventoryViewVO viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 已创建一个Inventory领域对象，时间输入规范 |
| 后置条件 | 调用Inventory领域对象viewInventory方法 |
| **InventoryController.checkRecord** | 语法 | public InventoryCheckVO checkRecord( ) |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象checkRecord方法 |
| **InventoryController.getImportID** | 语法 | public String getImportID () |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象getImportID方法 |
| **InventoryController.Import** | 语法 | Public ResultMessage addCommodities(TransferArrivalListVO receipt) |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象Import方法 |
| **InventoryController.buildImport** | 语法 | public InventoryImportReceiptPO buildImport(); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象buildImport方法 |
| **InventoryController.submitImport** | 语法 | public ResultMessage submitImport(InventoryImportReceiptVO importReceipt); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象submitImport方法 |
| **InventoryController.getExportID** | 语法 | public String getExportID () |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的getExportID方法 |
| **InventoryController.Export** | 语法 | public ResultMessage minusCommodities(TransferArrivalListVO receipt); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的**.Export**方法 |
| **InventoryController.buildExport** | 语法 | public InventoryExportReceiptPO buildExport(); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的buildExport方法 |
| **InventoryController.submitExport** | 语法 | public ResultMessage submitExport(InventoryExportReceiptVO exportReceipt); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的submitExport方法 |
| **InventoryController.getAdjustID** | 语法 | public String getAdjustID () |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的**etAdjustID**方法 |
| **InventoryController.Adjust** | 语法 | public AdjustReceiptPO adjust( ); |
| 前置条件 | 已创建一个Inventory领域对象 |
| 后置条件 | 调用Inventory领域对象的Adjust方法 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **Inventory.viewInventory** | 库存盘点 | |
| **Inventory.checkRecord** | 库存查看 | |
| **Inventory.getImportID** | 得到入库单ID | |
| **Inventory.Import** | 选择入库货物 | |
| **Inventory.buildImport** | 选择入库位置 | |
| **Inventory.submitImport** | 提交入库单 | |
| **Inventory.getExportID** | 得到出库单ID | |
| **Inventory.Export** | 选择出库方式 | |
| **Inventory.buildExport** | 选择出库运输方式 | |
| **Inventory..submitExport** | 提交出库单 | |
| **Inventory.getAdjustID** | 得到调整单ID | |
| **Inventory.Adjust** | 添加信息得到调整单 | |

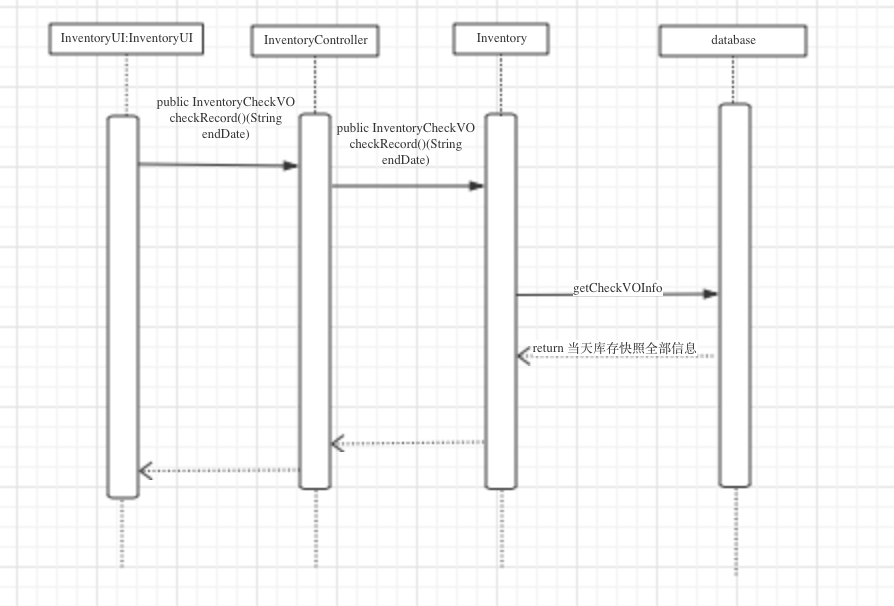
Inventory接口规范

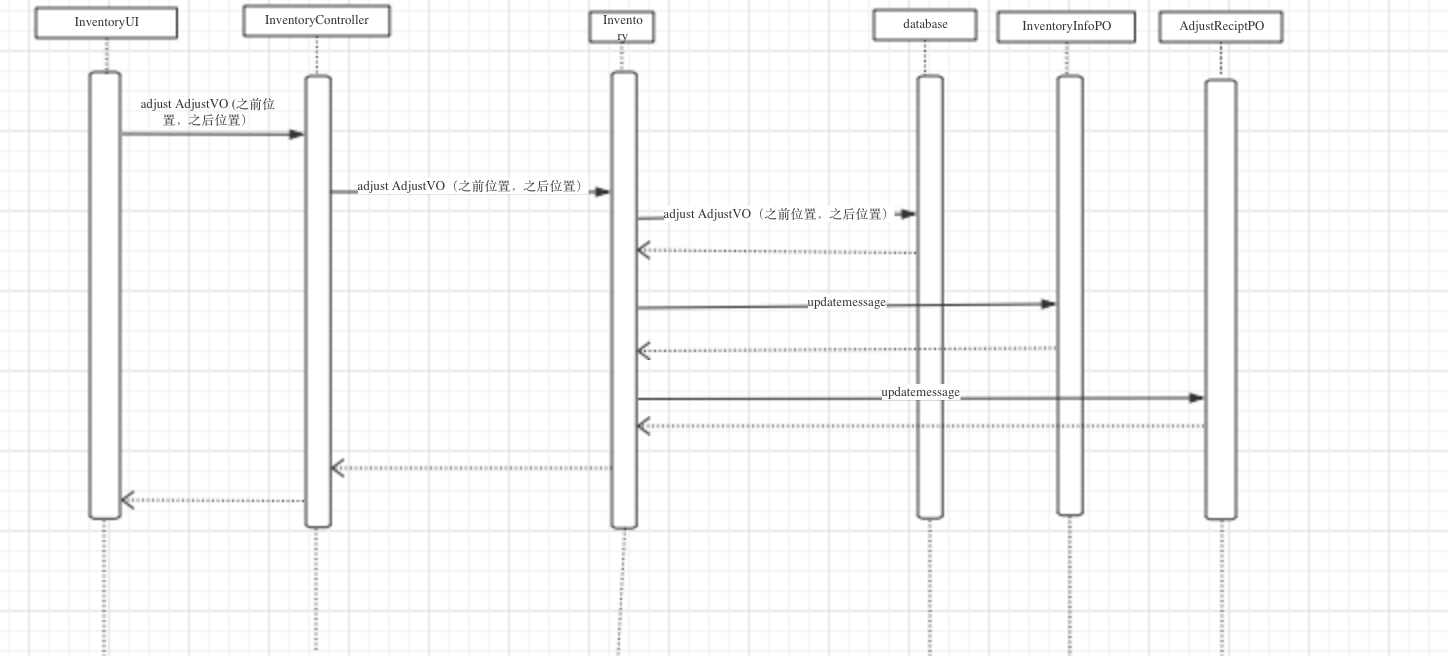
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **Inventory.viewInventory** | 语法 | public InventoryViewVO viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 日期的输入合法 |
| 后置条件 | 系统显示此时间段内的出入库数量、金额、库存位置 |
| **Inventory.checkRecord** | 语法 | public InventoryCheckVO checkRecord( ) |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 显示截至盘点时间的库存快照 |
| **Inventory.getImportID** | 语法 | public String getImportID () |
| 前置条件 | 开始创建一个入库单 |
| 后置条件 | 得到该入库单的编号 |
| **Inventory.Import** | 语法 | public ResultMessage addCommodities(TransferArrivalListVO receipt) |
| 前置条件 | 选择进行入库的货物 |
| 后置条件 | 添加信息到进货单 |
| **Inventory.buildImport** | 语法 | public InventoryImportReceiptPO buildImport(); |
| 前置条件 | 选择并添加相应的数据（订单托运编号、入库日期、目的地、区号、排号、架号、位号）到进货单中 |
| 后置条件 | 生成进货单PO |
| **Inventory.submitImport** | 语法 | public ResultMessage submitImport(InventoryImportReceiptVO importReceipt); |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| **Inventory.getExportID** | 语法 | public String getExportID () |
| 前置条件 | 开始创建一个出库单 |
| 后置条件 | 得到该出库单的编号 |
| **Inventory.Export** | 语法 | public ResultMessage minusCommodities(TransferArrivalListVO receipt); |
| 前置条件 | 选择进行出库的货物 |
| 后置条件 | 添加信息到出库单 |
| **Inventory.buildExport** | 语法 | public InventoryExportReceiptPO buildExport(); |
| 前置条件 | 选择并添加相应的数据（订单托运编号、出库日期、目的地、装运形式、中转单编号）到出库单中 |
| 后置条件 | 生成出库单PO |
| **Inventory..submitExport** | 语法 | public ResultMessage submitExport(InventoryExportReceiptVO exportReceipt); |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审 |
| **Inventory.getAdjustID** | 语法 | public String getAdjustID () |
| 前置条件 | 该区域库存达到警报值。开始创建一个调整单。 |
| 后置条件 | 得到该调整单的编号 |
| **Inventory.Adjust** | 语法 | public AdjustReceiptPO adjust( ); |
| 前置条件 | 输入被调整货物的订单托运编号、原先区号、排号、架号、位号及调整后的区号、排号、架号、位号 |
| 后置条件 | 添加信息到进货单 |
| **需要的服务（需接口）** | | |
| **服务名** | 服务 | |
| **InventoryDataService.**  **getImportID ()** | 得到入库单ID | |
| **InventoryDataService.getExportID ()** | 得到出库单ID | |
| **InventoryDataService.getAlarmID ()** | 得到调整单ID | |
| **InventoryDataService.getA()** | 得到区号 | |
| **InventoryDataService.getB()** | 得到排号 | |
| **InventoryDataService.getC()** | 得到架号 | |
| **InventoryDataService.getD()** | 得到位号 | |
| **InventoryDataService. getASize()** | 得到区容量 | |
| **InventoryDataService.getBSize()** | 得到排号容量 | |
| **InventoryDataService.getCSize()** | 得到架号容量 | |
| **InventoryDataService.getDSize()** | 得到位号容量 | |
| **InventoryDataService.getANum()** | 得到区此时容量 | |
| **InventoryDataService. getBNum()** | 得到排此时容量 | |
| **InventoryDataService. getCNum()** | 得到架此时容量 | |
| **InventoryDataService.getDNum()** | 得到位此时容量 | |
| **InventoryDataService.showImport()** | 返回所有入库单据的数据 | |
| **InventoryDataService.showExport()** | 返回所有出库单据的数据 | |
| **InventoryDataService.showAdjust()** | 返回所有调整单据的数据 | |
| **InventoryDataService.insertImport** | 增加一条po数据即生成入库单据 | |
| **InventoryDataService.insertExport** | 增加一条po数据即生成出库单据 | |
| **InventoryDataService.insertAdjust** | 增加一条po数据即生成调整单据 | |

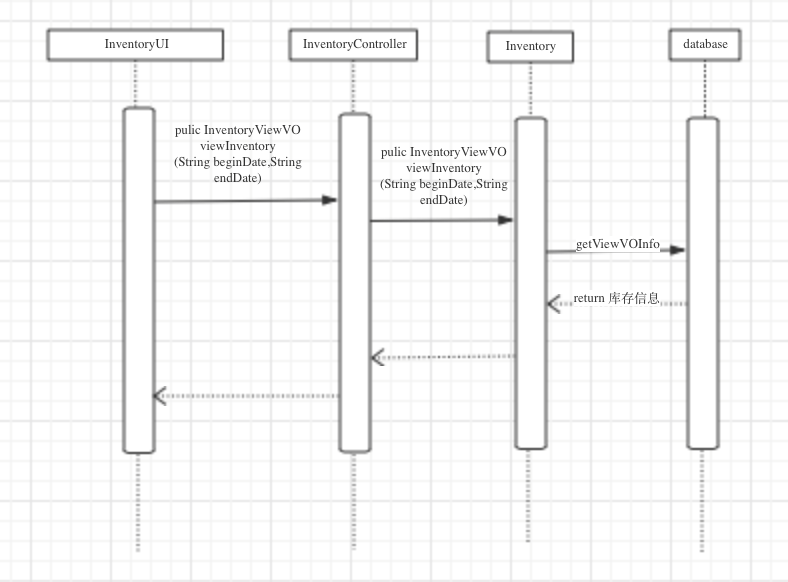
（4）业务逻辑层的动态模型

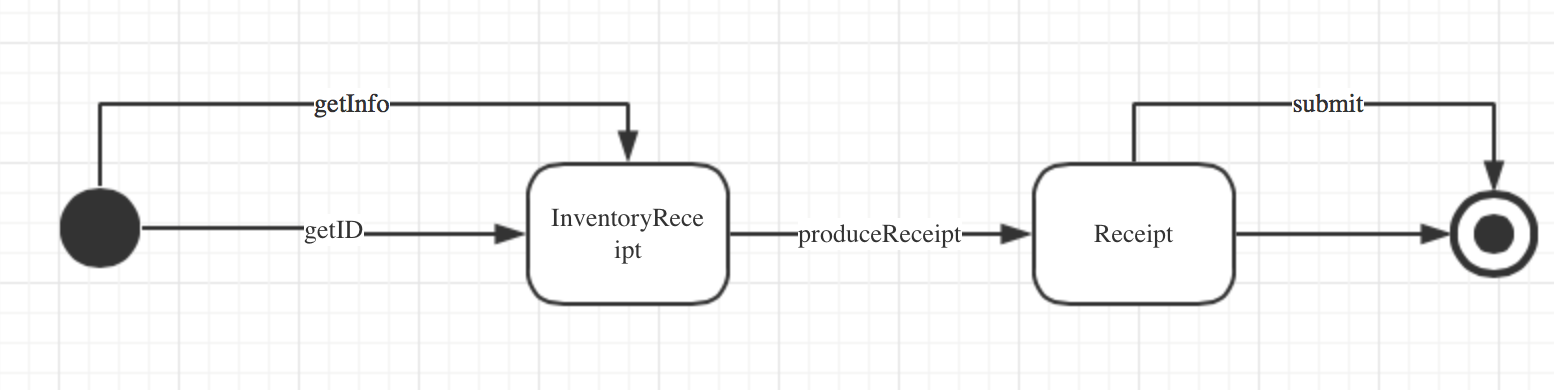
如下图表示了在快递物流系统中，当仓库管理人员想要进行仓库管理的时候，仓库管理管理业务逻辑处理的相关对象之间的协作

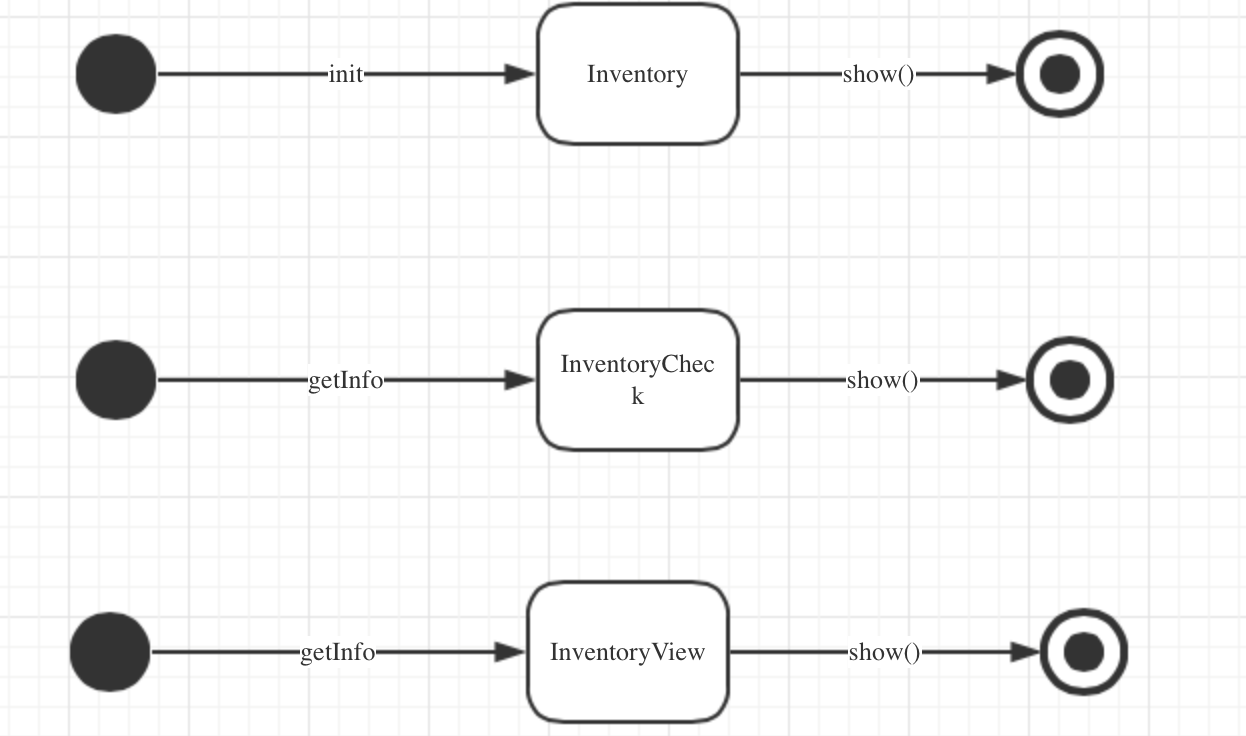












（5）业务逻辑层的设计原理

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。

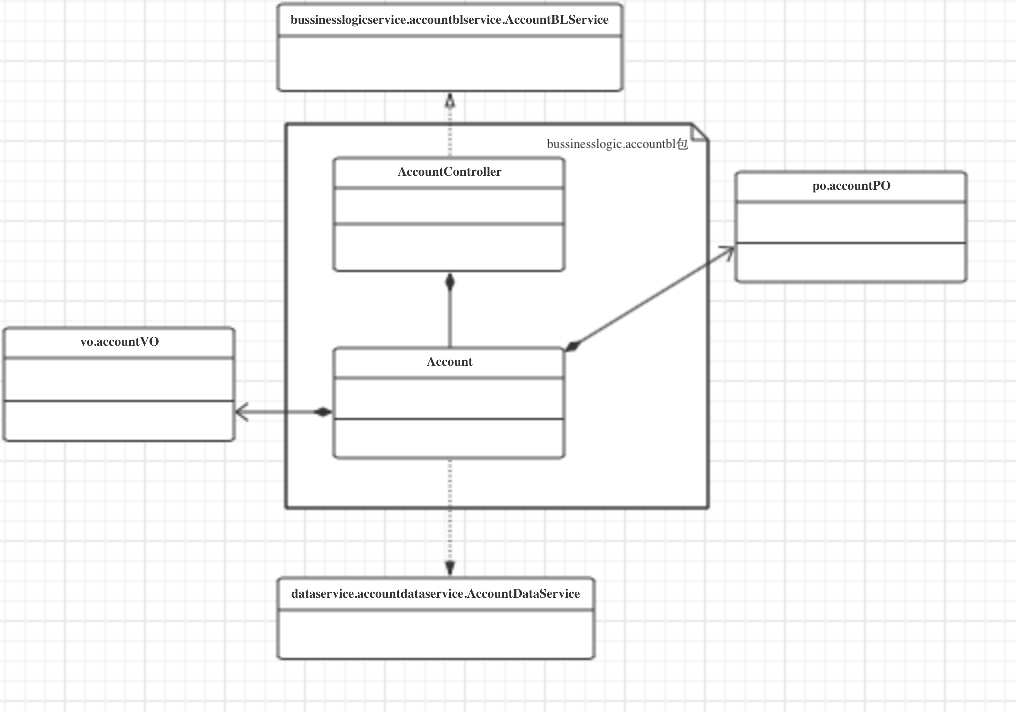
### Accountbl模块

1. 模块概述

Accountbl模块负责实现人员账户管查看所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.AccountBLService接口。业务逻辑层和数据层之间添加dataservice.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会将对订单的逻辑业务处理委托给Account对象。AccountPO是作为仓库相关信息的持久化对象被添加到设计模型中去的。

Accountbl模块的设计如图

Account模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountController | 负责实现员工账户信息管理界面所需要的服务 |
| Account | 员工信息的领域模型对象，拥有员工账户的信息，可以帮助完成员工账户信息管理界面所需要的服务 |

(3)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.show** | 语法 | public ArrayList<AccountVO> show() |
| 前置条件 | 已经创建一个Account领域对象 |
| 后置条件 | 调用account领域对象show方法 |
| **服务名** | 服务 | |
| **Account.show** | 查看所有账户信息 | |

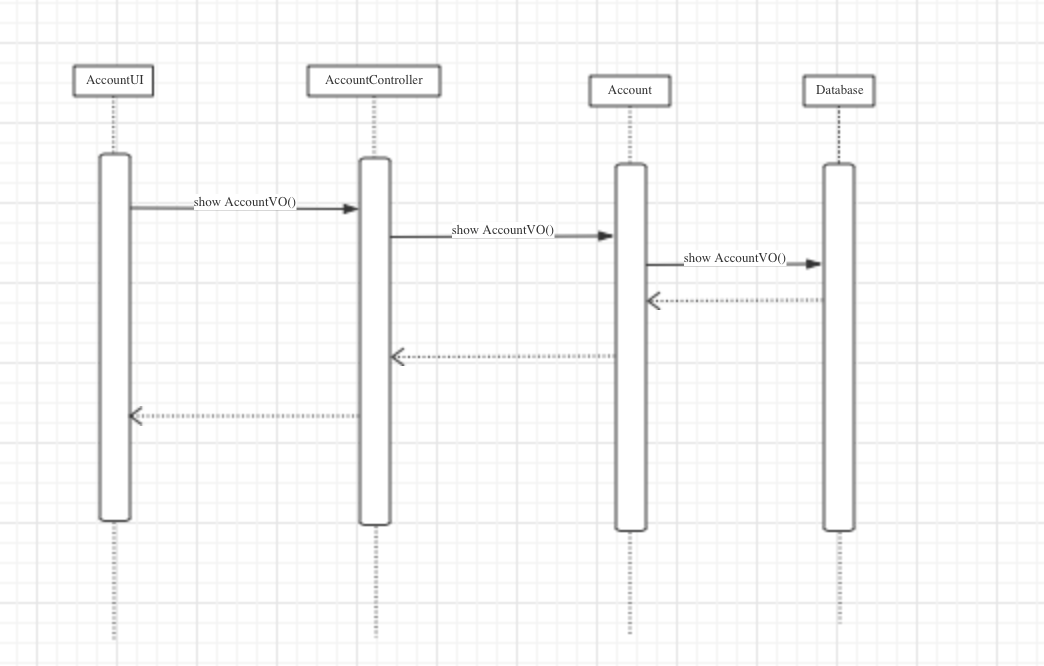
Account接口规范

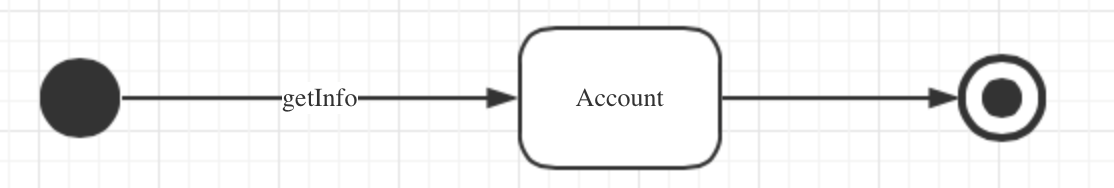
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **Account.show** | 语法 | public ArrayList<AccountVO> show() |
| 前置条件 | 已经创建一个Account对象 |
| 后置条件 | 展示所有用户ID |
| **服务名** | 服务 | |
| Account**DataService.**  **init** | 初始化持久数据 | |

（4）业务逻辑层的动态模型

如下图表示了在快递物流系统中，当总经理想要查看员工信息时候，员工账户信息管理管理业务逻

辑处理的相关对象之间的协作





（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。

### Recordbl模块

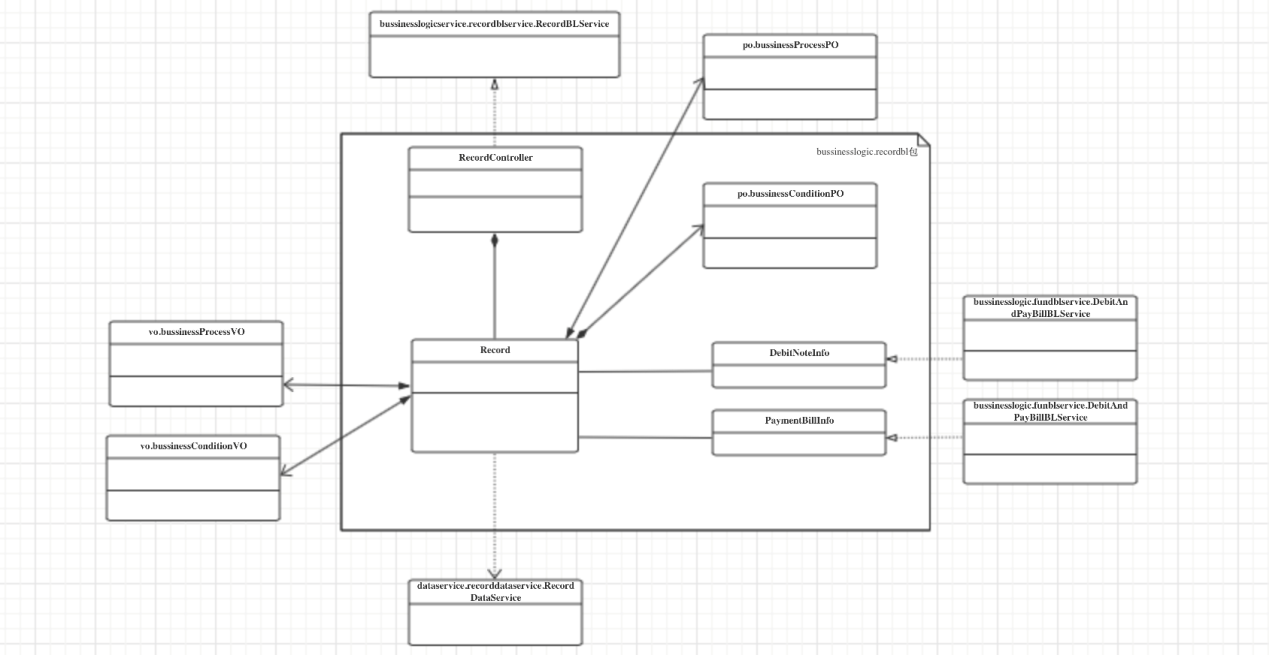
1. 模块概述

Recordbl模块负责实现统计分析所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.RecordBLService接口。业务逻辑层和数据层之间添加dataservice.RecordDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了RecordController，这样RecordController会将对订单的逻辑业务处理委托给Record对象。

Recordbl模块的设计如图



Record模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| RecordController | 负责实现统计分析管理界面所需要的服务 |
| Record | 经营情况的领域模型对象，拥有经营情况的信息，可以帮助完成统计分析界面所需要的服务 |

(3)模块内部类的接口规范

RecordController的接口规范

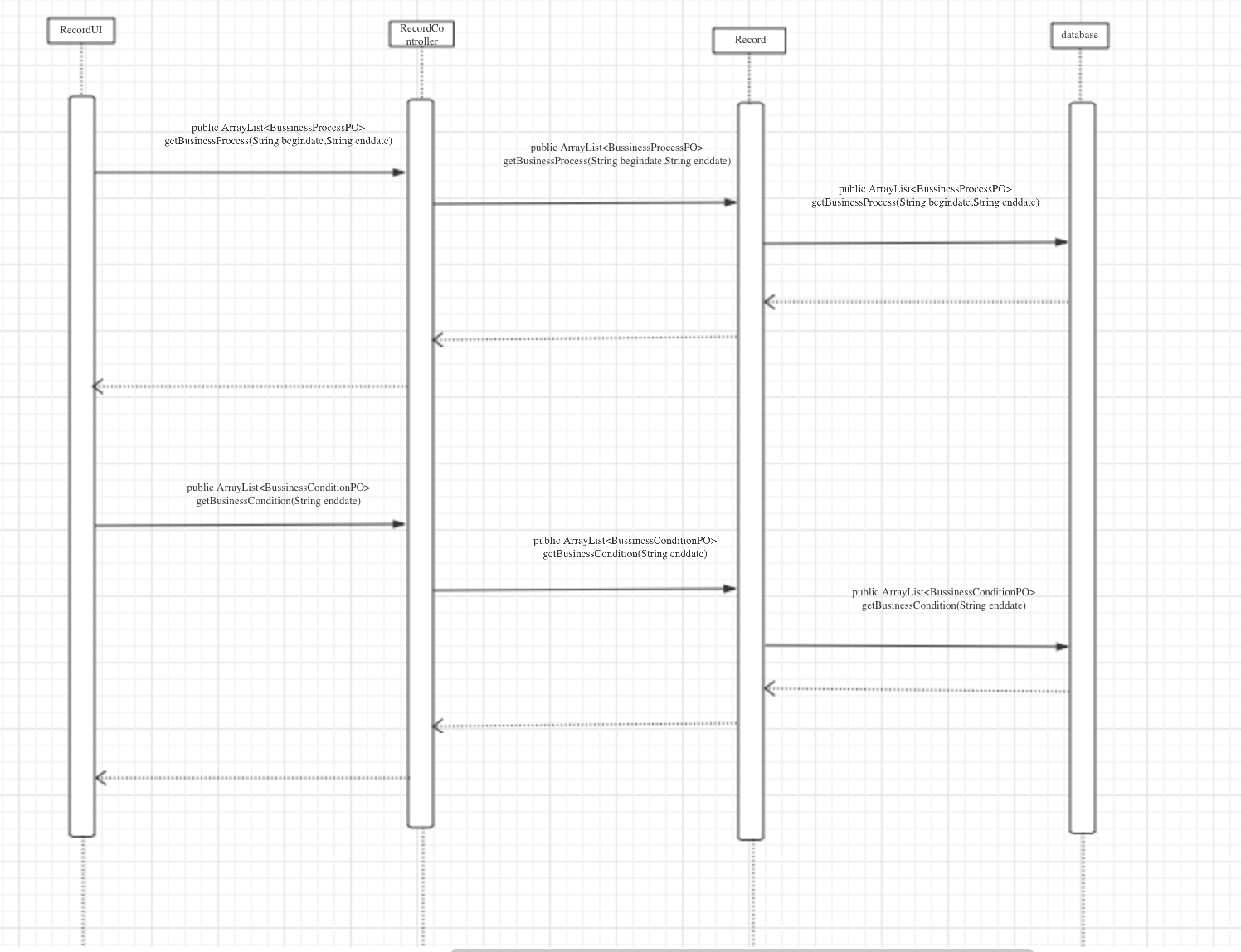
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **RecordController.getBusinessProcess** | 语法 | public BussinessProcessVO bussinessProcess(String begin, String end) |
| 前置条件 | 已经创建一个record领域对象 |
| 后置条件 | 调用Record领域对象getBusinessProcess方法 |
| **RecordController.getBusinessCondition** | 语法 | public BussinessConditionVO bussinessCondition(String end) |
| 前置条件 | 已经创建一个record领域对象 |
| 后置条件 | 调用Record领域对象getBusinessCondition方法 |
| **RecordController.income** | 语法 | public double income(String end) |
| 前置条件 | 已经创建一个record领域对象 |
| 后置条件 | 调用Record领域对象income方法 |
| **RecordController.earning** | 语法 | public double earning(String end) |
| 前置条件 | 已经创建一个record领域对象 |
| 后置条件 | 调用Record领域对象earning方法 |
| **RecordController.expense** | 语法 | public double expense(String end) |
| 前置条件 | 已经创建一个record领域对象 |
| 后置条件 | 调用Record领域对象expense方法 |
| 需要的服务（需接口） | | |
| **服务名** | 服务 | |
| **Record.getBusinessProcess** | 得到经营情况表 | |
| **RecordDataService.getBusinessCondition** | 得到经营收入表 | |
| **Record.income(String end)** | 得到收入 | |
| **Record.earning(String end)** | 得到利润 | |
| **Record.expense(String end)** | 得到花费 | |

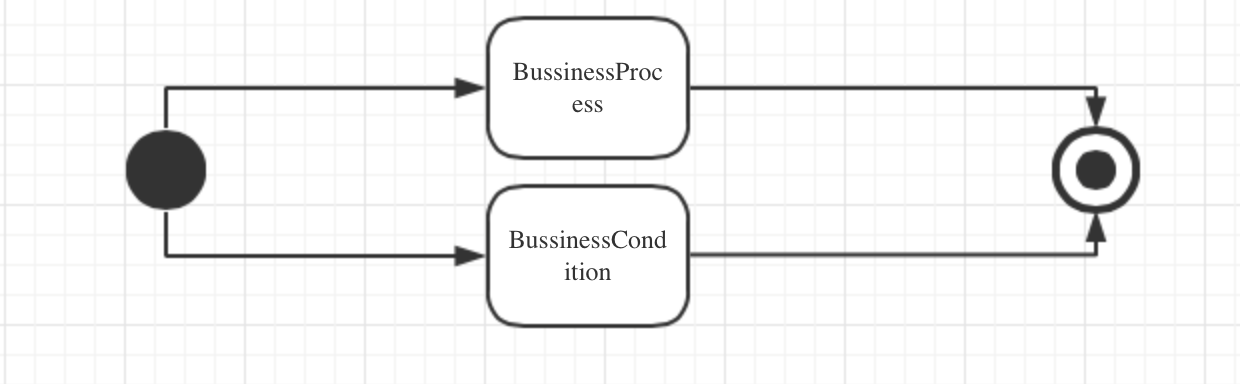
Record接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **Record.getBusinessProcess** | 语法 | public BussinessProcessVO bussinessProcess(String begin, String end) |
| 前置条件 | 符合输入规则 |
| 后置条件 | 返回符合条件的所有入款单和收款单 |
| **Record.getBusinessCondition** | 语法 | public BussinessConditionVO bussinessCondition(String end) |
| 前置条件 | 符合输入规则 |
| 后置条件 | 返回收入利润支出数据 |
| **Record.income** | 语法 | public double income(String end) |
| 前置条件 | 时间区间符合输入规则 |
| 后置条件 | 返回计算得到的收入 |
| **Record.earning** | 语法 | public double earning(String end) |
| 前置条件 | 时间区间符合输入规则 |
| 后置条件 | 返回计算得到的利润 |
| **Record.expense** | 语法 | public double expense(String end) |
| 前置条件 | 时间区间符合输入规则 |
| 后置条件 | 返回计算得到的支出 |
| 需要的服务（需接口） | | |
| **服务名** | 服务 | |
| Record**DataService.** getBusinessProcess(String begindate,String enddate) | 返回1. 收款单2.付款单 | |
| Record**DataService.**getBusinessCondition(String enddate);  } | 返回信息： 1. 收入2. 支出。3. 利润：总收入-总支出 | |

（4）业务逻辑层的动态模型

如下图表示了在快递物流系统中，当财务人员想要进行统计分析的时候，统计分析业务逻辑处理的相关对象之间的协作





（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。

* 1. **数据层的分解**

数据层的开发包图请参见体系结构描述文档图4。

### OrderDataService

1. 模块概述

OrderDataService模块提供订单数据的增加、查找、修改服务

1. 模块内类接口规范

OrderDataService接口规范如下表

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderDataService.insert | 语法 | public ResultMessage insert(OrderPO 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数据 |
| OrderDataService.find | 语法 | public UserPO find(String orderNumber) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| OrderDataService.update | 语法 | public ResultMessageupdate (OrderPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| OrderDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### FacilityDataService

1. 模块概述

FacilityDataService模块提供营业厅运输工具数据的增加、查找、修改、删除服务

1. 模块内类接口规范

FacilityDataService接口规范如下表

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| FacilityDataService.add | 语法 | public ResultMessage add  (FacilityPO 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数据 |
| FacilityDataService.find | 语法 | public FacilityPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| FacilityDataService.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 |
| FacilityDataService.modify | 语法 | public ResultMessage modify  (FacilityPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 修改po |
| FacilityDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| FacilityDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| FacilityDataService. show | 语法 | public ArrayList< FacilityPO >  show () |
| 前置条件 | 无 |
| 后置条件 | 返回所有车辆信息 |

### BranchDataService

1. 模块概述

BranchDataService模块提供营业厅运输事件数据的增加、查找、修改、删除服务模块内类接口规范

BranchDataService接口规范如下表

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BranchDataService.insert | 语法 | public ResultMessageinsert(BranchPO 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数据 |
| BranchDataService.find | 语法 | public BranchPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| BranchDataService.update | 语法 | public ResultMessageupdate (BranchPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| BranchDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### BaseDataService

（1）模块概述

BaseDataService模块提供公司运营成本常量数据的增加、查找、修改、删除服务模块内类接口规范

BaseDataService接口规范如下表

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| BaseDataService.insert | 语法 | public ResultMessageinsert(BasePO 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数据 |
| BaseDataService.find | 语法 | public BasePO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| BaseDataService.update | 语法 | public ResultMessageupdate BasePO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| BaseDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### ReceiptDataService

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ReceiptDataService.insert | 语法 | public ResultMessageinsert(ReceiptPO 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数据 |
| ReceiptDataService.find | 语法 | public ReceiptPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| ReceiptDataService.update | 语法 | public ResultMessageupdate (ReceiptPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| ReceiptDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### UserDataService

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserDataService.insert | 语法 | public ResultMessageinsert(UserPO 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数据 |
| UserDataService.find | 语法 | public ReceiptPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| UserDataService.update | 语法 | public ResultMessageupdate (UserPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| UserDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### OrganizationDataService

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrganizationDataService.insert | 语法 | public ResultMessageinsert(OrganizationPO 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数据 |
| OrganizationDataService.find | 语法 | public ReceiptPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| OrganizationDataService.update | 语法 | public ResultMessageupdate (OrganizationPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| OrganizationDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### TransferDataService

（1）模块概述

TransferDataService模块提供中转中心数据的初始化、增加、查找、修改服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **TransferDataService.insert** | 语法 | public ResultMessageinsert(TransferPO 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数据 |
| **TransferDataService.find** | 语法 | public TransferPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| **TransferDataService.update** | 语法 | public ResultMessageupdate (TransferPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| **TransferDataService.init** | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### OpeningStockDataService

（1）模块概述

OpeningStockDataService模块提供期初账目的初始化、增加、查找服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **OpeningStockDataService.add** | 语法 | public ResultMessage add(OpeningStockPO 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数据 |
| **OpeningStockDataService.init** | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| **OpeningStockDataService.find** | 语法 | public OpeningStockPOfind(String id) |
| 前置条件 | 无 |
| 后置条件 | 返回OpeningStockPO持久数据 |

### DebitAndPayBillDataService

（1）模块概述

DebitAndPayBillDataService模块提供收款单和付款单的获取编号、增加、查找、删除、修改服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBillDataService.insert** | 语法 | public ResultMessage  insert(DebitAndPayBillPO 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数据 |
| **DebitAndPayBillDataService.find** | 语法 | public ArrayList<DebitAndPayBillPO>  find(String id) |
| 前置条件 | 单据Id输入正确 |
| 后置条件 | 查找返回相应的结果 |
| **DebitAndPayBillDataService.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 |
| **DebitAndPayBillDataService.update** | 语法 | public ResultMessage update  (DebitAndPayBillPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| **DebitAndPayBillDataService.getId** | 语法 | public void getId() |
| 前置条件 | 无 |
| 后置条件 | 获得一个单据Id |
| **DebitAndPayBillDataService.getPayId** | 语法 | public void getPayId() |
| 前置条件 | 无 |
| 后置条件 | 获得一个付款单Id |
| **DebitAndPayBillDataService.getExpense** | 语法 | public void getExpenseId() |
| 前置条件 | 无 |
| 后置条件 | 获得一个收款单Id |

### DebitAndPayBillShowDataService

（1）模块概述

DebitAndPayBillShowDataService模块提供收款单和付款单的展示服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **DebitAndPayBillshowDataService.show** | 语法 | public ArrayList<DebitAndPayBillPO>  show(ReceiptType type) |
| 前置条件 | 单据类型输入正确 |
| 后置条件 | 查找返回相应的结果 |

### BankAccountDataService

（1）模块概述

BankAccountDataService模块提供银行账户的增加、查找、删除、修改服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| **BankAccountDataService.insert** | 语法 | public ResultMessage  insert(BankPO 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数据 |
| **BankAccountDataService.find** | 语法 | public ArrayList<BankPO> find(  String keywords,  FindTypeAccount bankAccoutType) |
| 前置条件 | 关键字和搜索类型输入正确 |
| 后置条件 | 查找返回相应的结果 |
| **BankAccountDataService.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 |
| **BankAccountDataService.update** | 语法 | public ResultMessage update (BankPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |

### ExpenseDataservice

（1）模块概述

ExpenseDataservice模块提供快递员收款单数据的增加、查找、删除、修改服务

（2）模块内类接口规范

|  |  |  |
| --- | --- | --- |
| **ExpenseDataservice.insert** | 语法 | public ResultMessage  insert(ExpensePO 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数据 |
| **ExpenseDataservice.show** | 语法 | public ArrayList<ExpensePO> show(String ID) |
| 前置条件 | 收款记录Id输入正确 |
| 后置条件 | 查找返回相应的结果 |
| **ExpenseDataservice.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 |
| **ExpenseDataservice.update** | 语法 | public ResultMessage update (ExpensePO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |

### RecordDataService

（1）模块概述

RecordDataService模块提供订单数据的增加、查找、修改服务

（2）模块内类接口规范

RecordDataService接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **RecordDataService.getBusinessProcess** | 语法 | publicArrayList<BussinessProcessPO> getBusinessProcess(String begindate,String enddate) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 返回1. 收款单2.付款单 |
| **RecordDataService.getBusinessCondition** | 语法 | publicArrayList<BussinessConditionPO> getBusinessCondition(String enddate) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 返回信息： 1. 收入2. 支出。3. 利润：总收入-总支出 |

### AccountDataService

（1）模块概述

AccountDataService模块提供订单数据的增加、查找、修改服务

（2）模块内类接口规范

AccountDataService接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| **AccountDataService.delete** | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### InventoryDataService

（1）模块概述

InventoryDataService模块提供订单数据的增加、查找、修改服务

（2）模块内类接口规范

InventoryDataService接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| **InventoryDataService.init** | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| **InventoryDataService.**  **getImportID** | 语法 | public String getImportID () |
| 前置条件 | 无 |
| 后置条件 | 得到入库单ID |
| **InventoryDataService.**  **getExportID** | 语法 | public String getExportID () |
| 前置条件 | 无 |
| 后置条件 | 得到出库单ID |
| **InventoryDataService.**  **getAlarmID** | 语法 | public String getAlarmID () |
| 前置条件 | 无 |
| 后置条件 | 得到调整单ID |
| **InventoryDataService.getA()** | 语法 | public String getA() |
| 前置条件 | 无 |
| 后置条件 | 得到区号 |
| **InventoryDataService.getB()** | 语法 | public String getB() |
| 前置条件 | 无 |
| 后置条件 | 得到排号 |
| **InventoryDataService.getC()** | 语法 | public String getC() |
| 前置条件 | 无 |
| 后置条件 | 得到架号 |
| **InventoryDataService.getD()** | 语法 | public String getD() |
| 前置条件 | 无 |
| 后置条件 | 得到位号 |
| **InventoryDataService.getASize()** | 语法 | public String getASize() |
| 前置条件 | 无 |
| 后置条件 | 得到区容量 |
| **InventoryDataService.getBSize()** | 语法 | public String getBSize() |
| 前置条件 | 无 |
| 后置条件 | 得到排号容量 |
| **InventoryDataService.getCSize()** | 语法 | public String getCSize() |
| 前置条件 | 无 |
| 后置条件 | 得到架号容量 |
| **InventoryDataService.getDSize()** | 语法 | public String getDSize() |
| 前置条件 | 无 |
| 后置条件 | 得到位号容量 |
| **InventoryDataService.getANum()** | 语法 | public String getANum() |
| 前置条件 | 无 |
| 后置条件 | 得到区此时容量 |
| **InventoryDataService.getBNum()** | 语法 | public String getBNum() |
| 前置条件 | 无 |
| 后置条件 | 得到排此时容量 |
| **InventoryDataService.getCNum()** | 语法 | public String getCNum() |
| 前置条件 | 无 |
| 后置条件 | 得到架此时容量 |
| **InventoryDataService.getDNum()** | 语法 | public String getDNum() |
| 前置条件 | 无 |
| 后置条件 | 得到位此时容量 |
| **InventoryDataService.showImport()** | 语法 | public ArrayList<InventoryImportReceiptPO> showImport() |
| 前置条件 | 无 |
| 后置条件 | 返回所有入库单据的数据 |
| **InventoryDataService.showExport()** | 语法 | Public ArrayList<InventoryExportReceiptPO> showExport() |
| 前置条件 | 无 |
| 后置条件 | 返回所有出库单据的数据 |
| **InventoryDataService.showAdjust()** | 语法 | public ArrayList<AdjustReceiptPO> showAdjust() |
| 前置条件 | 无 |
| 后置条件 | 返回所有调整单据的数据 |
| **InventoryDataService.insertImport(InventoryImportReceiptPO po)** | 语法 | public ResultMessage insertImport(InventoryImportReceiptPO po) |
| 前置条件 | 无 |
| 后置条件 | 增加一条po数据即生成入库单据 |
| **InventoryDataService.insertExport(InventoryExportReceiptPO po)** | 语法 | public ResultMessage insertExport(InventoryExportReceiptPO po) |
| 前置条件 | 无 |
| 后置条件 | 增加一条po数据即生成出库单据 |
| **InventoryDataService.insertAdjust(AdjustReceiptPO po)** | 语法 | public ResultMessage insertAdjust(AdjustReceiptPO po) |
| 前置条件 | 无 |
| 后置条件 | 增加一条po数据即生成调整单据 |

# 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。

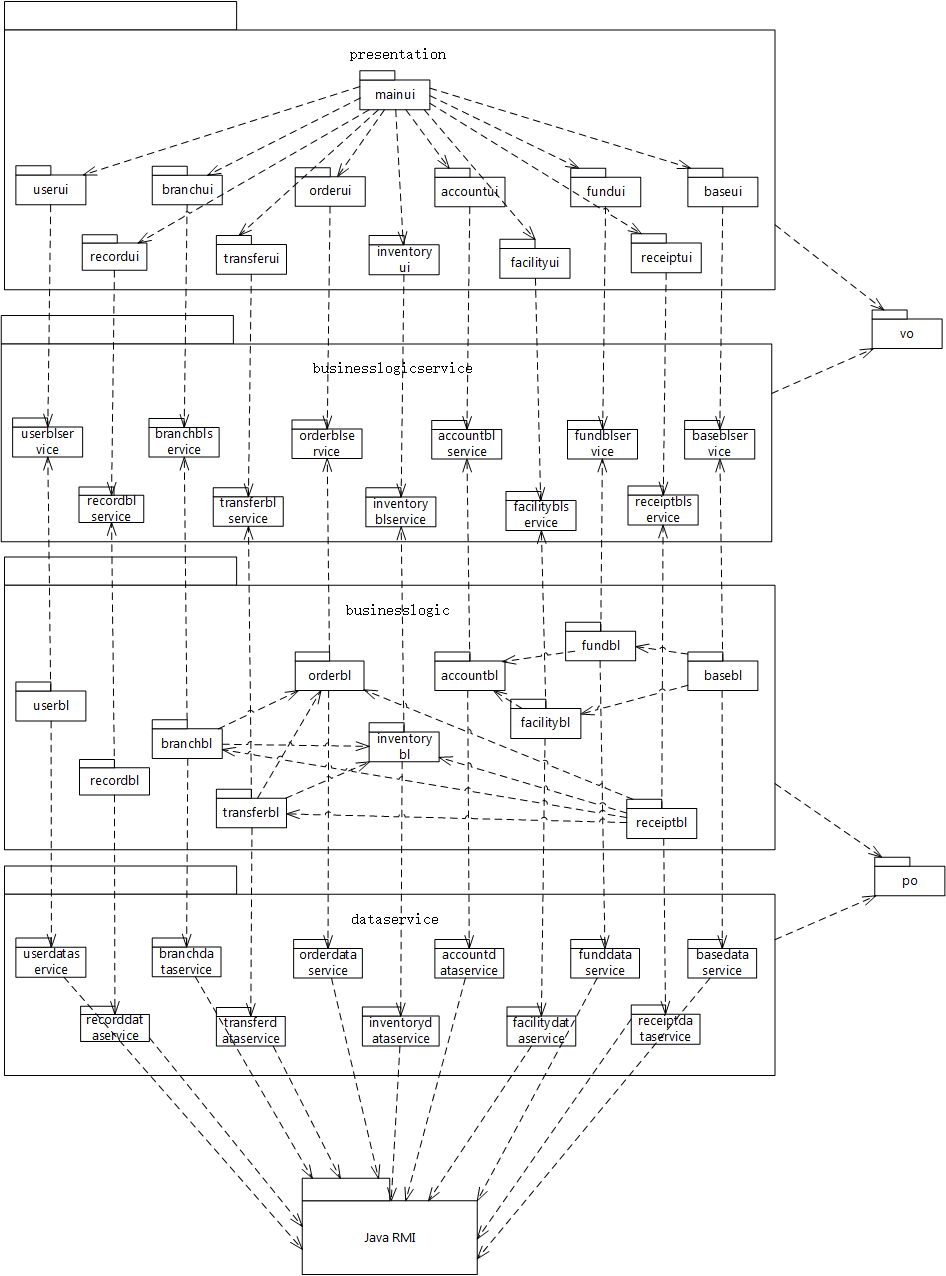
****

图1客户端包图

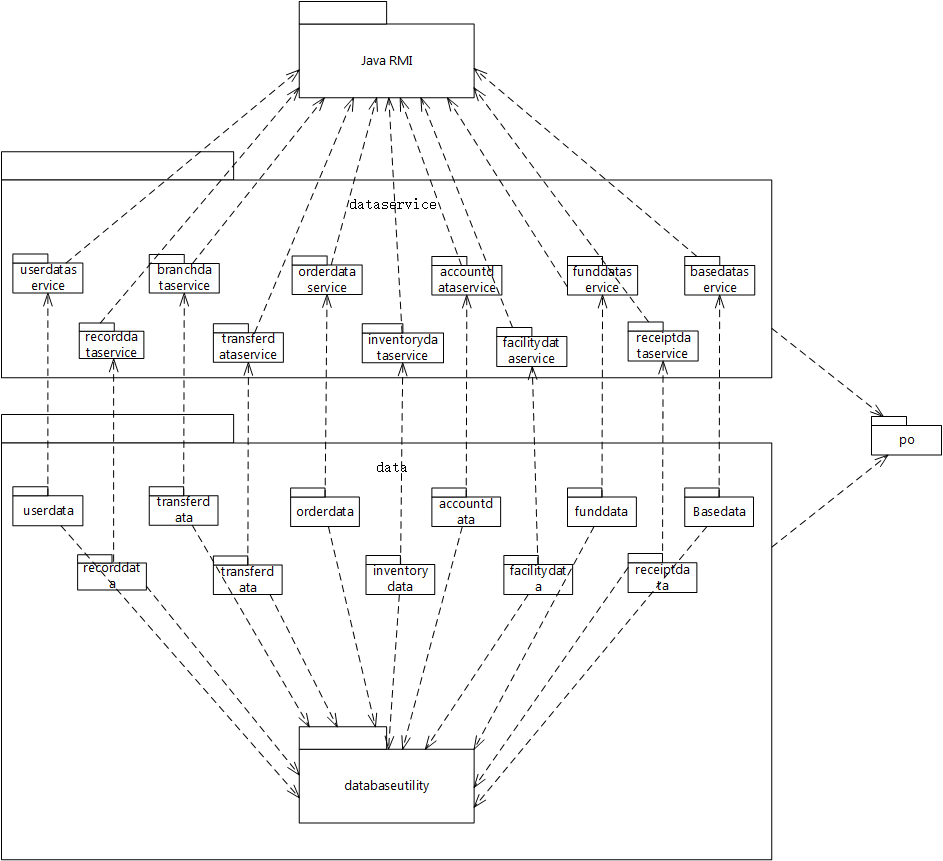
****

图2服务器端包图