### 4.1.4 shipmentbl模块

（1）模块概述

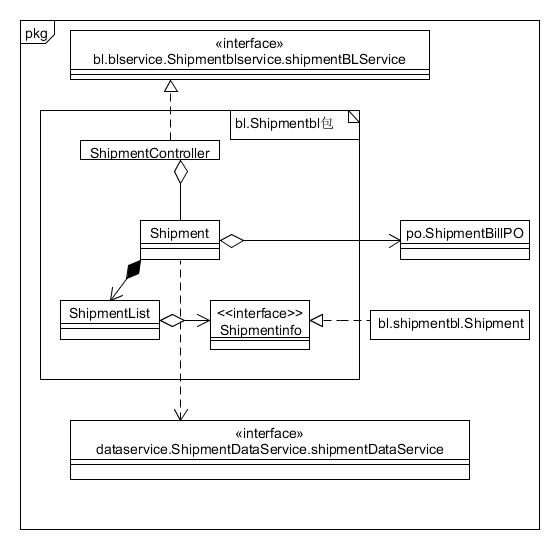
shipmentbl模块承担的需求详见需求规格说明文档功能需求及相关非功能需求。

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接口。业务逻辑层和数据层之间添加dataservice.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。而ShipmentList和的添加是Shipmentinfo的容器类。ShipmentList封装了装运信息的数据。Shipmentinfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

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**

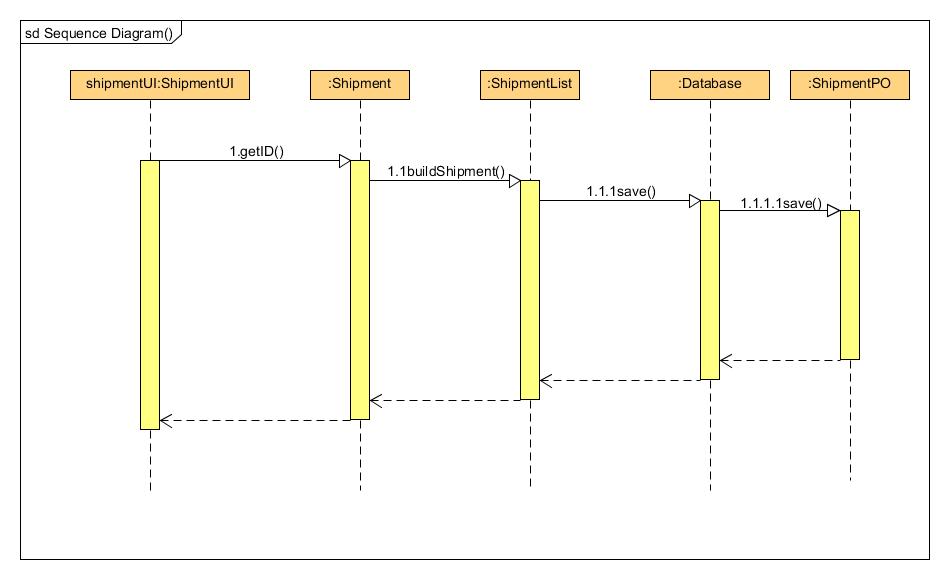
|  |  |  |  |
| --- | --- | --- | --- |
| **提供的接口（供接口）** | | | |
| **ShipmentController. getTruckID** | 语法 | | public String getTruckID () |
| 前置条件 | | 已创建一个Shipment领域对象 |
| 后置条件 | | 调用Shipment领域对象的getTruckID方法 |
| **ShipmentController. getFreightID** | 语法 | | public String getFreightID () |
| 前置条件 | | 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save方法 |
| **ShipmentController. end** | 语法 | | public void end() |
| 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end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 |
| **Shipment. end** | 语法 | | public void end() |
| 前置条件 | | 无 |
| 后置条件 | | 结束此次装运信息输入，不持久化更新涉及的领域对象的数据 |
| **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 | |
| **ShipmentDataService.init** | | 初始化持久数据 | |
| **DatabaseFacory.getShipmentDatabase** | | 得到Shipment数据库的引用 | |

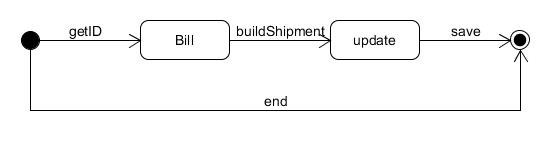
（4）业务逻辑层的动态模型

下图表明了物流信息管理系统中，当用户输入装运信息后，装运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

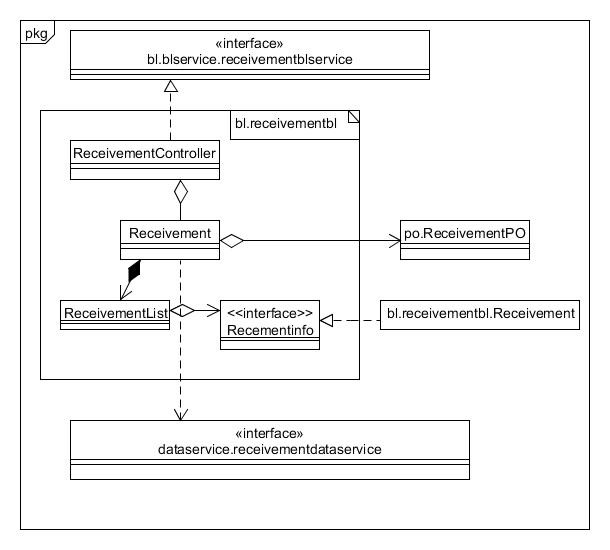
receivementbl模块承担的需求参见需求规格说明文档功能需求和相关非功能需求

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bl.blservice.receivementblservice接口。业务逻辑层和数据层之间添加dataservice.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Receivementinfo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



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** |
| ReceivementController | 负责收件信息界面所需要的服务 |
| 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.add | 语法 | public ResultMessage add (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add方法 |
| ReceivementContoller.update | 语法 | public ResultMessage update(ReceivementPO po) |
| 前置条件 | 已创建一个Receivement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 |
| ReceivementController.delete | 语法 | public ResultMessage delete(String id) |
| 前置条件 | 已创建一个Receiement领域对象，输入符合输入规则 |
| 后置条件 | 调用Receivement领域对象的delete方法 |
| ReceivementController.end | 语法 | public void end() |
| 前置条件 | 已创建一个Receiement领域对象，输入符合输入规则 |
| 后置条件 | 调用Receivement领域对象的end方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Receivement.add | 加入一个收件对象信息 | |
| Receivement.update | 更新一个收件对象信息 | |
| Receivement.delete | 删除一个收件对象信息 | |

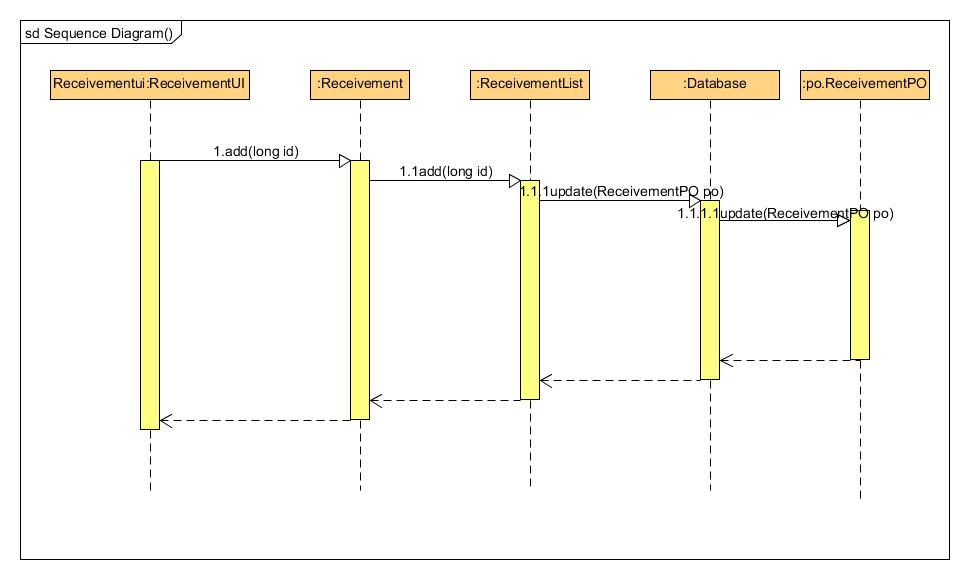
**Receivement的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Receivement.add | 语法 | public ResultMessage add (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 |
| Receivement.end | 语法 | Publicvoid end() |
| 前置条件 | 无 |
| 后置条件 | 结束此次接收信息输入，不持久化更新涉及的领域对象的数据 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Receivementdataservice.insert(ReceivementPO po) | 插入单一持久化对象 | |
| receivementdataservice.init() | 初始化持久化数据 | |
| receivementdataservice.delete(long id) | 删除单一持久化对象 | |
| receivementdataservice.update(ReceivementPO po) | 更新单一持久化对象 | |
| receivementdataservice.find(log id) | 查找单一持久化对象 | |

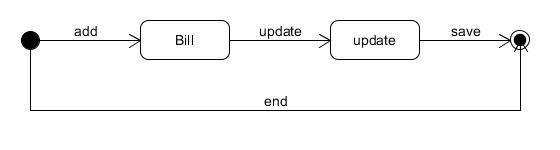
（4）业务逻辑层的动态模型

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



如下图所示的状态图描述了Receivement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add方法被UI调用，Receivement进入Bill状态；之后通过修改数据进入update状态。



(5)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

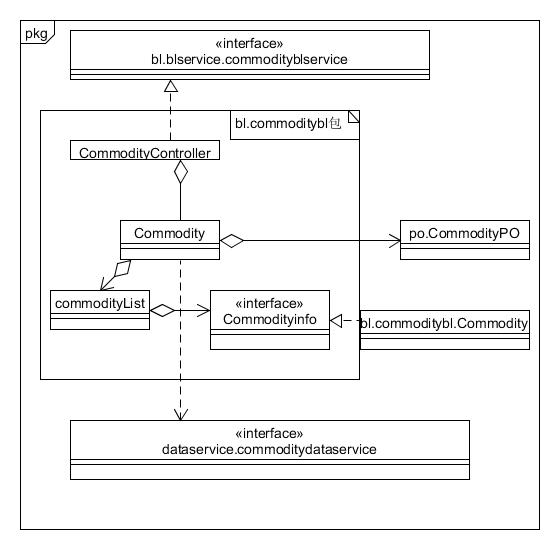
commoditybl模块承担的需求参见需求规格说明文档功能需求和相关非功能需求

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。



commoditybl模块各个类的职责如表

**commoditybl模块各个类的职责**

|  |  |
| --- | --- |
| **模块** | **职责** |
| CommodityController | 负责库存信息界面所需要的服务 |
| 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**

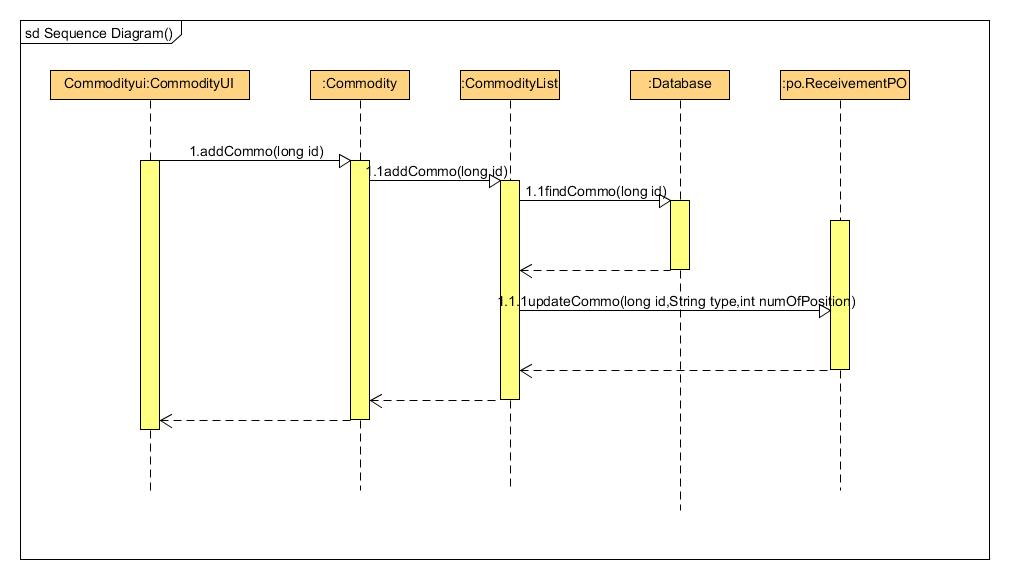
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| CommodityController.addCommodity | 语法 | public ResultMessage addCommodity(long id) |
| 前置条件 | 已创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity领域对象的addCommodity方法 |
| CommodityContoller.update | 语法 | public ResultMessage update(CommodityPO po) |
| 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 |
| CommodityContoller.delete | 语法 | public ResultMessage delete(long id) |
| 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deletecommo方法 |
| CommodityContoller.find | 语法 | public ResultMessage find(long id) |
| 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findCommo方法 |
| CommodityContoller.end | 语法 | public void end() |
| 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endCommo方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Commodity.addCommodity | 加入一个仓库对象信息 | |
| Commodity.updateCommo | 更新库存货物信息 | |
| Commodity.deleteCommo | 删除一个仓库对象信息 | |
| Commodity.findCommo | 查找一个仓库对象信息 | |

**Commodity 的接口规范**

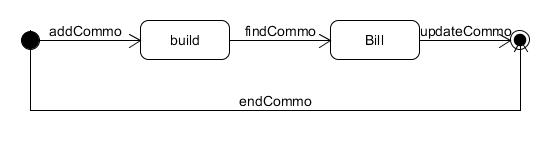
|  |  |  |  |
| --- | --- | --- | --- |
| **提供的接口（供接口）** | | | |
| commodity.addCommo | 语法 | | public ResultMessage addCommo(long id,String type,int numOfPosition) |
| 前置条件 | | 货物编号、货物类型、库区位置输入合法 |
| 后置条件 | | 将货物添加到仓库，持久化更新涉及的对象的信息 |
| commodity.deleteCommo | 语法 | | public ResultMessage deleteCommo(long 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(long 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(long 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 |
| commodity.endCommo | 语法 | | public void endCommo() |
| 前置条件 | | 无 |
| 后置条件 | | 结束此次仓库信息输入，不持久化更新涉及的领域对象的数据 |
| **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 | |
| Commoditydataservice.init() | | 初始化持久数据 | |

(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。



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



(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。

### 4.2.4 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 |
| **ShipmentDataService**.init | 语法 | Public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### 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 |