**物流管理系统（ELMS）**

**软件详细设计**

**描述文档**

****

**南京大学软件学院**

**GGS.DDU**

张海涛郑闻昊张文玘周颖婷

**2015年11月3日**

[1.引言 3](#_Toc434622053)

[1.1编制目的 3](#_Toc434622054)

[1.2词汇表 3](#_Toc434622055)

[1.3参考资料 3](#_Toc434622056)

[2.产品概述 3](#_Toc434622057)

[3.体系结构设计概述 3](#_Toc434622058)

[4.结构视角 4](#_Toc434622059)

[4.1业务逻辑层的分解 4](#_Toc434622060)

[4.1.1 userbl模块 4](#_Toc434622061)

[4.1.2 dealbl模块 8](#_Toc434622062)

[4.1.3 invoicebl模块 14](#_Toc434622063)

[4.1.4 storagebl 模块 19](#_Toc434622064)

[4.1.5 financebl模块 23](#_Toc434622065)

[4.1.6 memberbl模块 38](#_Toc434622066)

[4.1.7 Logbl模块 43](#_Toc434622067)

[5.依赖视角 46](#_Toc434622068)

[6.界面 49](#_Toc434622069)

[用户模块界面 49](#_Toc434622070)

[快递员及寄件人界面 51](#_Toc434622071)

[库存界面 55](#_Toc434622072)

[系统日志界面 58](#_Toc434622073)

[经济模块界面 59](#_Toc434622074)

[单据界面 64](#_Toc434622075)

[人事界面 71](#_Toc434622076)

# 1.引言

## 1.1编制目的

本报告详细完成对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。

## 1.2词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
| ELMS | 物流管理系统 | …… |
| …… | …… | …… |

## 1.3参考资料

# 2.产品概述

物流管理系统用例文档和物流管理系统软件需求规格说明文档中对产品的概括描述。

# 3.体系结构设计概述

参考物流管理系统概要设计文档中对体系结构设计的概述。

# 4.结构视角

## 4.1业务逻辑层的分解

业务逻辑层的开发包图参见软件体系结构文档图4.1-1。

### 4.1.1 userbl模块

(1)模块概述

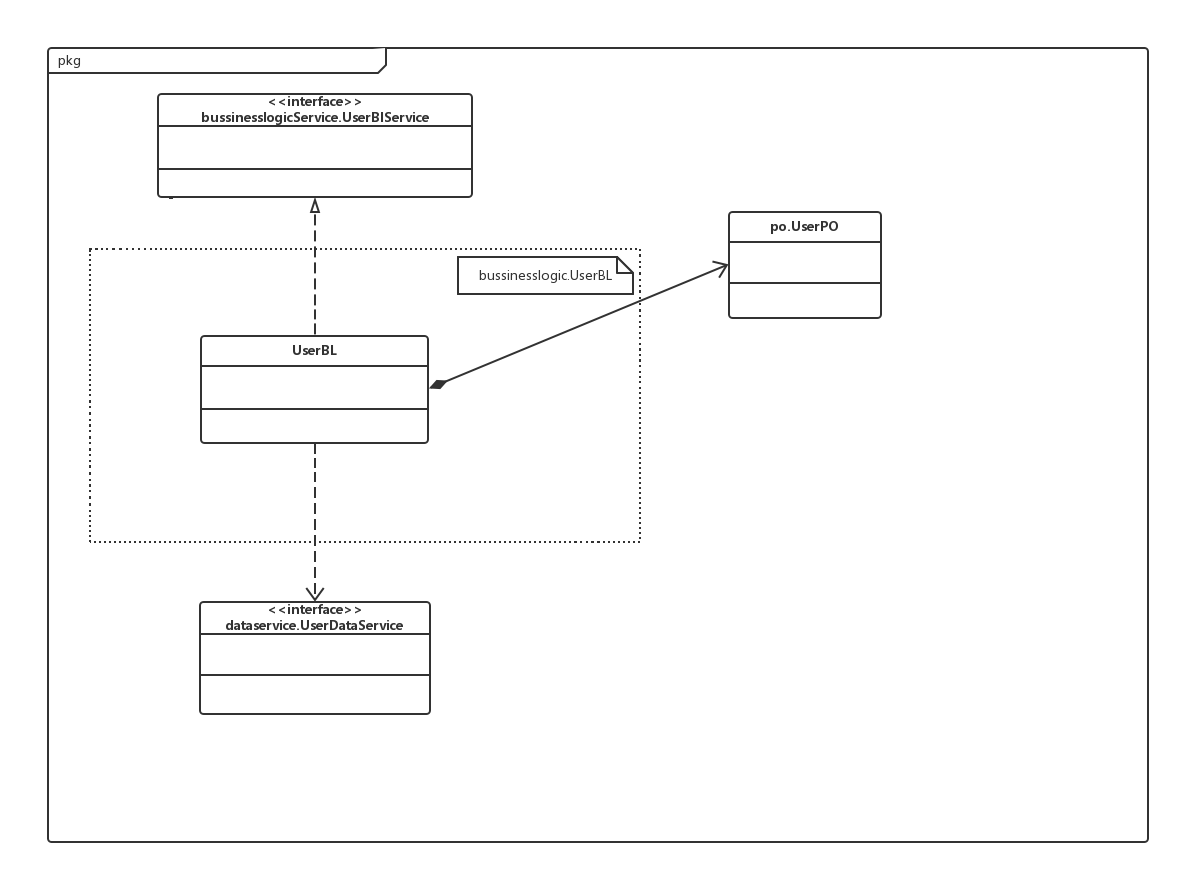
Userbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

Userbl模块的职责及接口参见软件系统结构描述文档5.3.2.1

(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，我们添加了businesslogicService.UserBLService接口。业务逻辑层和数据层之间添加了dataservice.UserDataService接口。UserPO是作为用户账户记录的持久化对象被添加到设计模型中去的。

Userbl模块的设计如下图所示。



Userbl模块各个类的职责如下表所示。

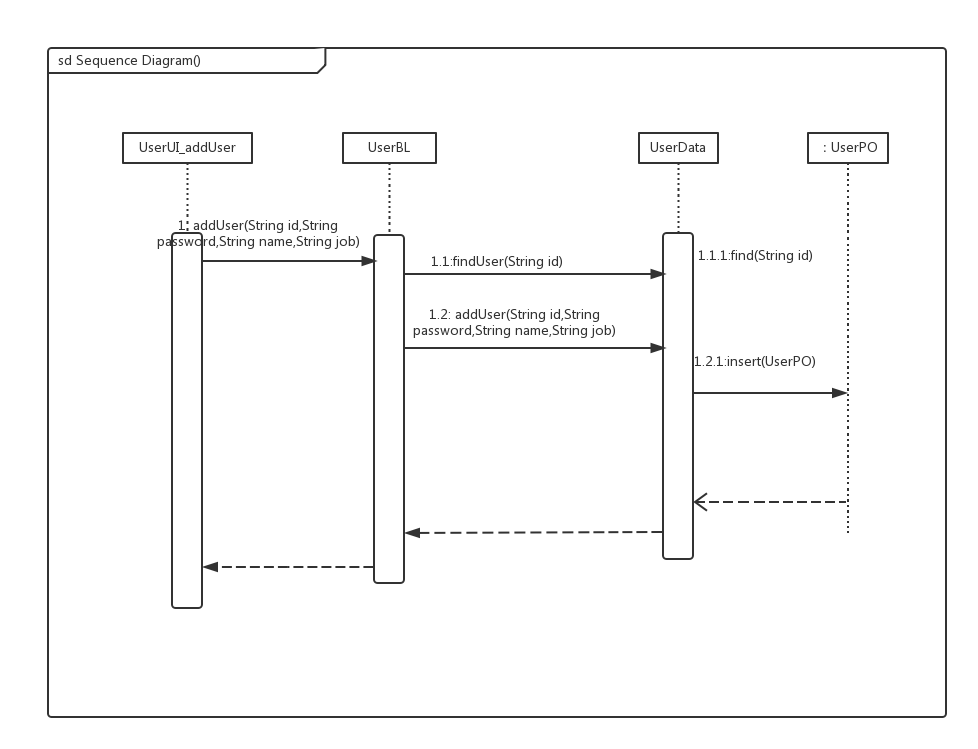
|  |  |
| --- | --- |
| 模块 | 职责 |
| Userbl | 负责用户的登录，账号的增、删、改、查。 |

(3)模块内部类的接口规范

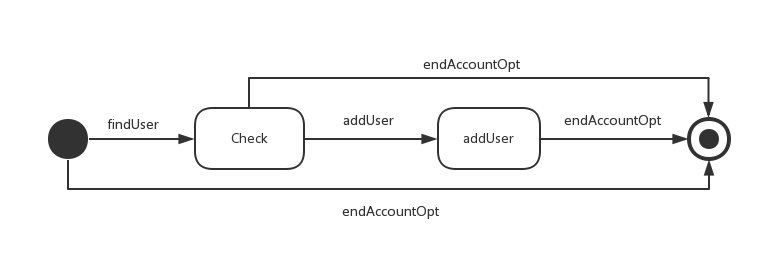
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserBL.login | 语法 | Public int login(String id,String password) throws RemoteException |
| 前置条件 | 用户登录 |
| 后置条件 | 根据输入的密码是否正确以及ID对应的职务进入相应的功能界面。 |
| UserBL.addUser | 语法 | Public boolean addUser(String id,String password,String name,String job)throws IOException |
| 前置条件 | 该ID不存在 |
| 后置条件 | 新建一个新的账户。 |
| UserBL.updateJob | 语法 | Public boolean updateJob(String id,String password,String name,String job)throws RemoteException |
| 前置条件 | 该账户存在 |
| 后置条件 | 更改对应账户的信息 |
| UserBL.deleteUser | 语法 | Public boolean(String id,String password,String name,String job) throws RemoteException |
| 前置条件 | 该ID存在 |
| 后置条件 | 删除该账户 |
| UserBL.findUser | 语法 | Public UserVO findUser(string id) throws RemoteException |
| 前置条件 | 该ID存在 |
| 后置条件 | 返回该ID对应的UserPO |
| User. endAccountOpt | 语法 | Public void endAccountOpt() |
| 前置条件 | 已结束所有对账户的操作 |
| 后置条件 | 结束此次账号管理，持久化更新涉及的领域对象的数据 |
| 需要的服务（需接口） | | |
| UserDataService. insert(UserPO po) | | 在数据库中插入UserPO对象 |
| UserDataService. delete(UserPO po) | | 在数据库中删除UserPO对象 |
| UserDataService. update(UserPO po) | | 更新数据库中的UserPO对象 |
| UserDataService. find( long id) | | 根据ID进行查找UserPO对象 |
| DatabaseFactory. geUserDatabase | | 得到User数据库的服务的引用 |

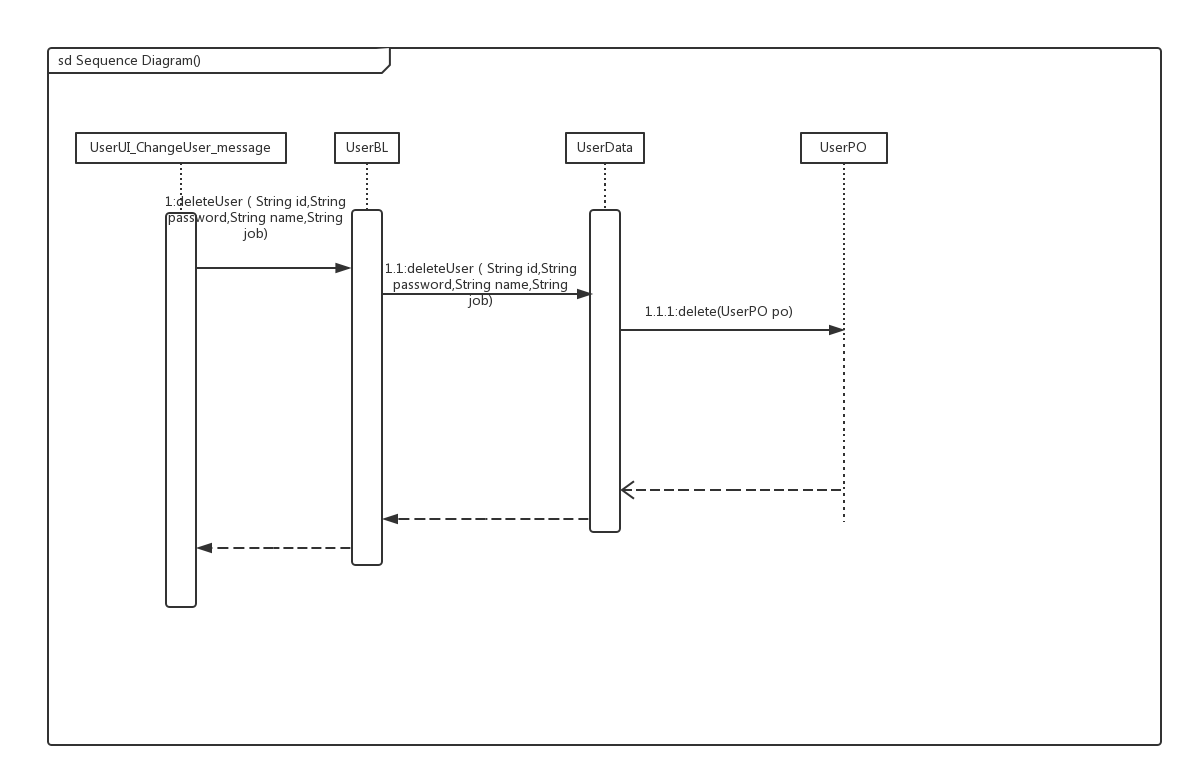
（4）业务逻辑层的动态模型

下图表明了在快递物流管理系统中，当系统管理员输入新增账户的ID、password、name以及Job后，UserBL处理的相关对象之间的协作。

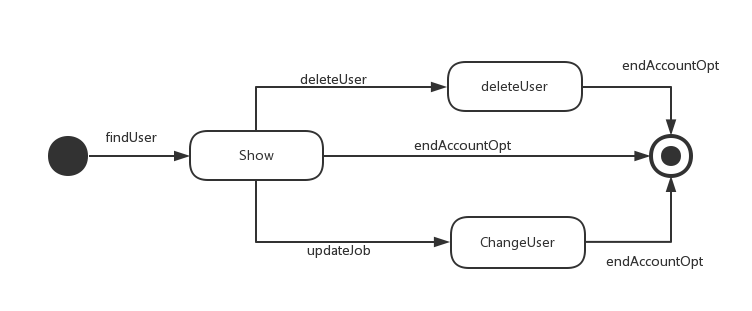


状态图如下：

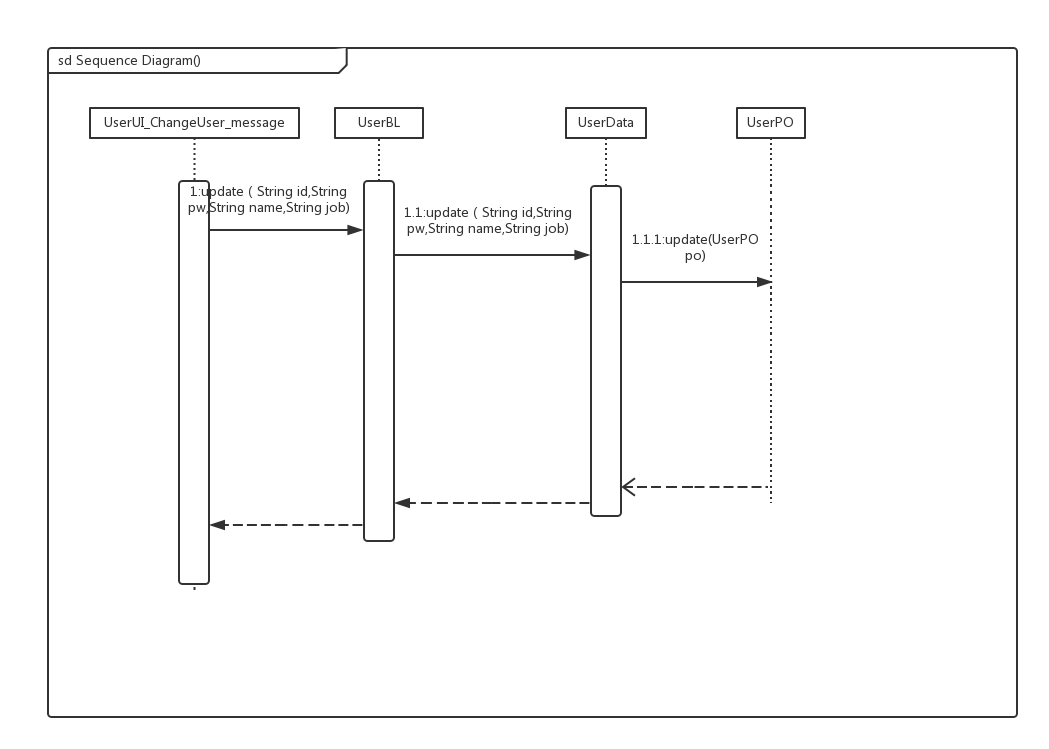


下图表示了系统管理员删除账户时的UserBL处理的相关对象之间的协作。

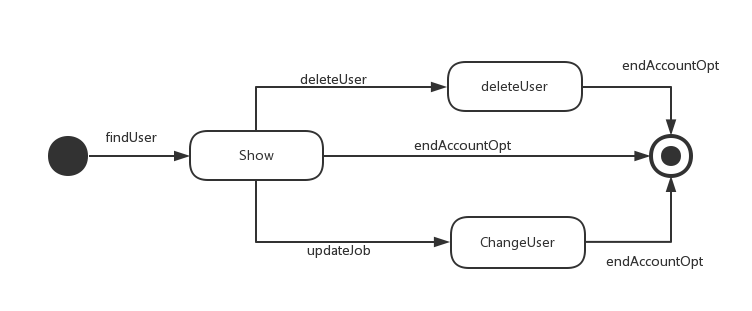
状态图如下：



下图表示了系统管理员更改账户时的UserBL处理的相关对象之间的协作。



状态图如下：



(5)业务逻辑层的设计原理

利用集中式控制风格，每个界面需要访问的业务逻辑直接调用相应的业务逻辑类。

### 4.1.2 dealbl模块

(1)模块概述

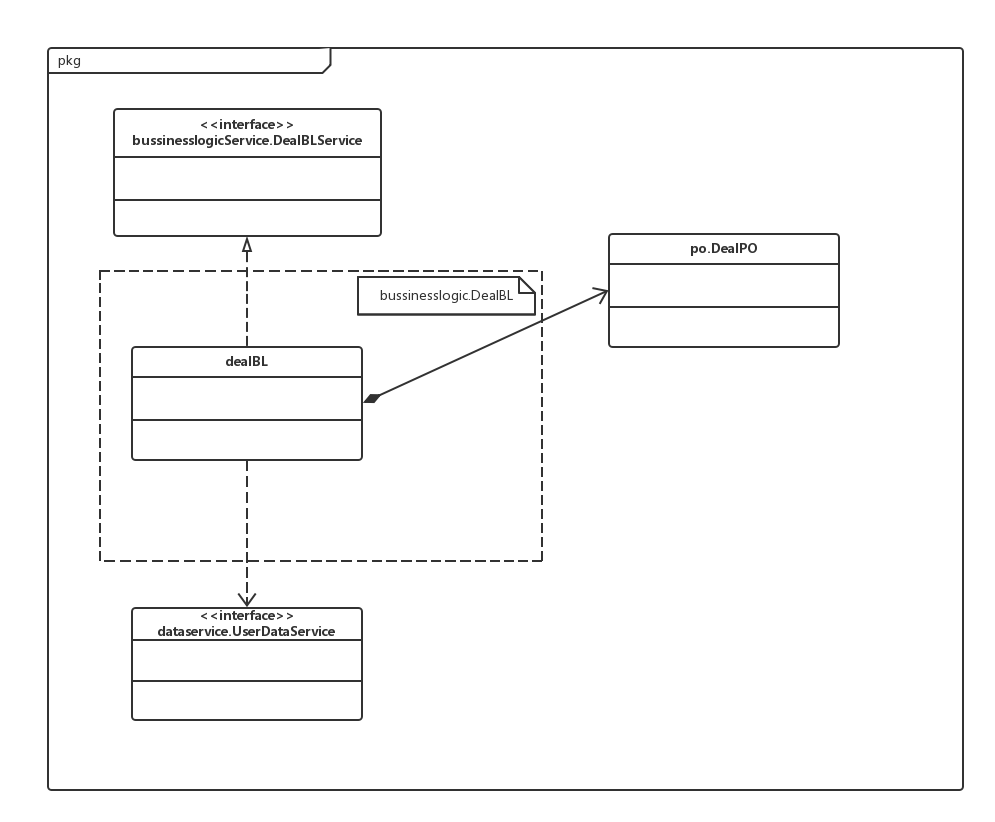
dealbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

dealbl模块的职责及接口参见软件系统结构描述文档5.3.2.2

(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，我们添加了businesslogicService.DealBLService接口。业务逻辑层和数据层之间添加了dataservice.dealDataService接口。DealPO是作为订单的持久化对象被添加到设计模型中去的。

dealbl模块的设计如下图所示。



Dealbl模块各个类的职责如下表所示。

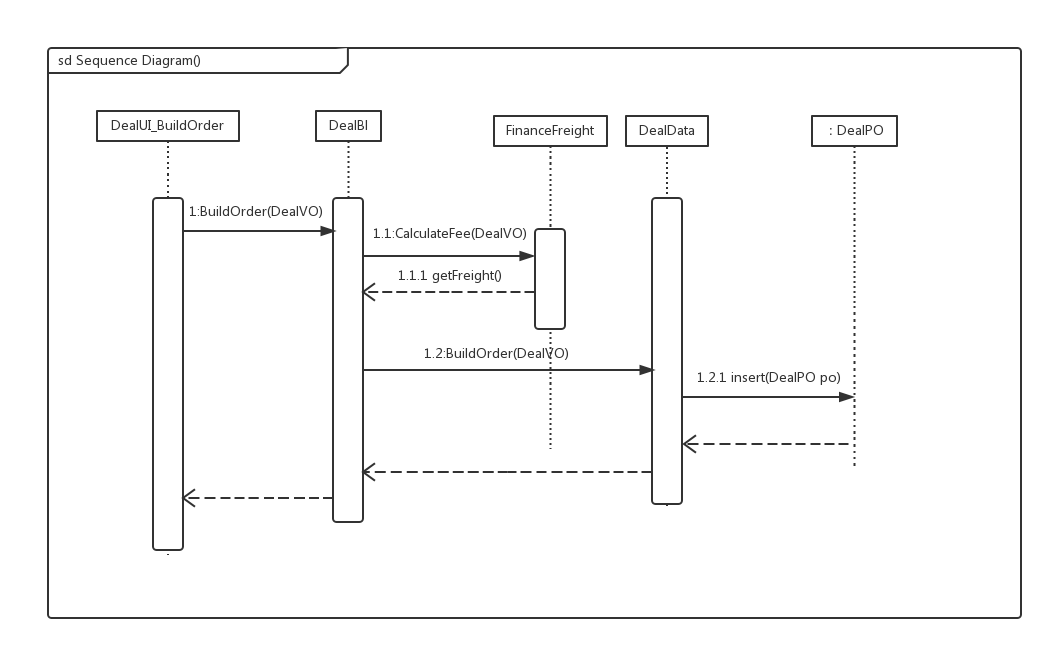
|  |  |
| --- | --- |
| 模块 | 职责 |
| Dealbl | 负责实现快递员交易界面所需要的服务 |

(3)模块内部类的接口规范

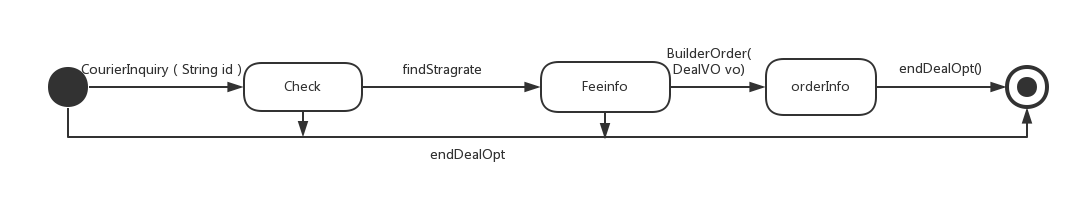
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Deal. BuildOrder | 语法 | Public boolean BuildOrder(DealVO vo) |
| 前置条件 | 该订单号原先不存在于数据库(依赖于find方法，若返回null，不存在，该方法返回true；若返回了一个dealPO,则存在，该方法返回false) |
| 后置条件 | 生成该订单，持久化更新涉及的领域对象的数据 |
| Deal. CourierInquiry | 语法 | Public DealVO CourierInquiry(string orderID) |
| 前置条件 | 输入订单号进行查询(在数据层找不到返回Null) |
| 后置条件 | 根据输入的订单号返回订单信息 |
| Deal. Recipients | 语法 | Public DealVO recipients(String orderid,string realReciever, string time) |
| 前置条件 | 该订单存在于数据库中依赖于find方法，若不存在，返回null) |
| 后置条件 | 返回一个更新收件信息后的订单信息，并且持久化更新涉及的领域对象的数据。 |
| Deal. endDealOpt | 语法 | Public void endDealOpt() |
| 前置条件 | 已结束本次操作 |
| 后置条件 | 结束此次交易处理/查询操作，持久化更新涉及的领域对象的数据 |
| Deal.findbyCourier(String courierName) | 语法 | Public ArrayList<UserVO>(String name) |
| 前置条件 | 无 |
| 后置条件 | 根据快递员姓名查找当天该快递员收到的订单。 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFactory. getDealDatabase | 得到deal数据库的服务的引用 | |
| DealDataService.find(long id) | 根据ID查找DealPO对象(找不到返回NULL) | |
| DealDataService.findbyCourier(string courierName) | 按照快递员姓名查找当天所属的DealPO结果 | |
| DealDataService.insert(DealPO po) | 在数据库中插入DealPO对象 | |
| DealDataService.delete(DealPO po) | 在数据库中删除DealPO对象 | |
| DealDataService. Update(DealPO po) | 在数据库中更新这个DealPO对象 | |

（4）业务逻辑层的动态模型

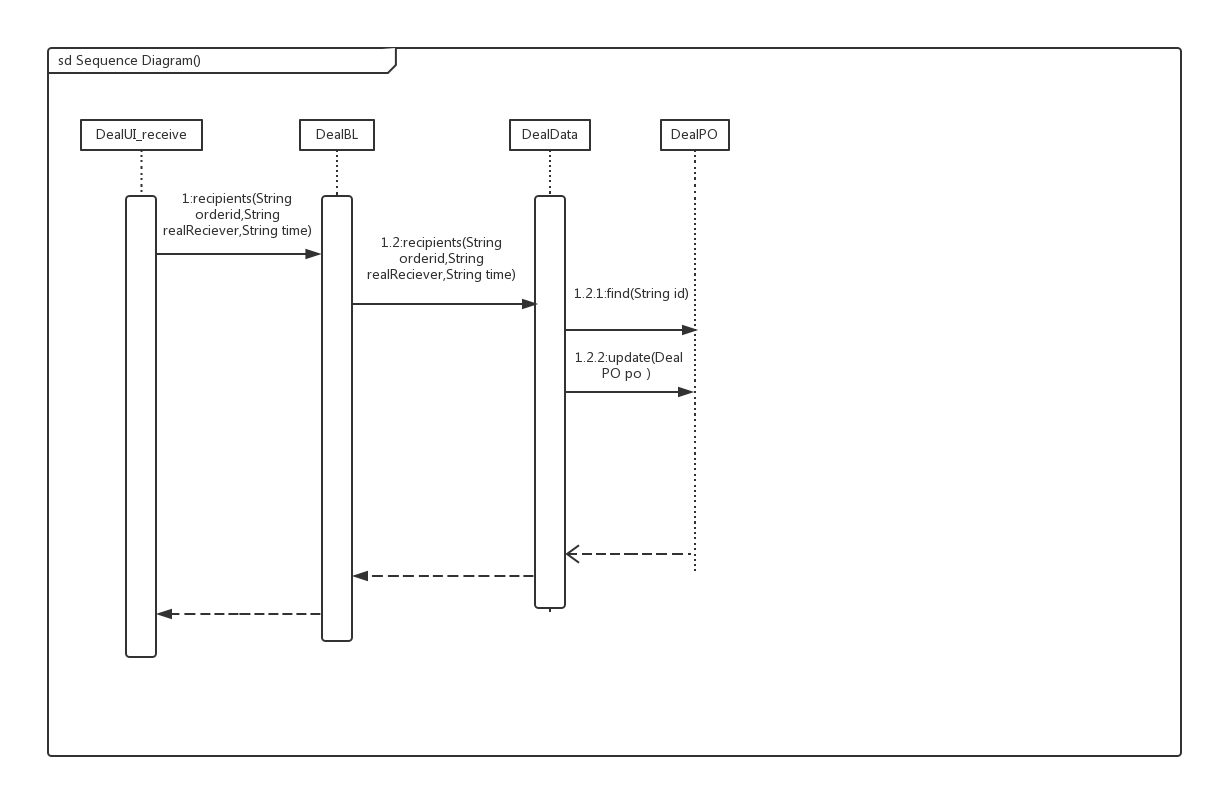
下图表明了在快递物流管理系统中，当快递员输入新增订单的信息后，DealBL处理的相关对象之间的协作。



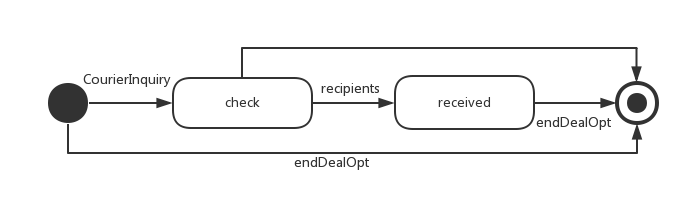
状态图如下



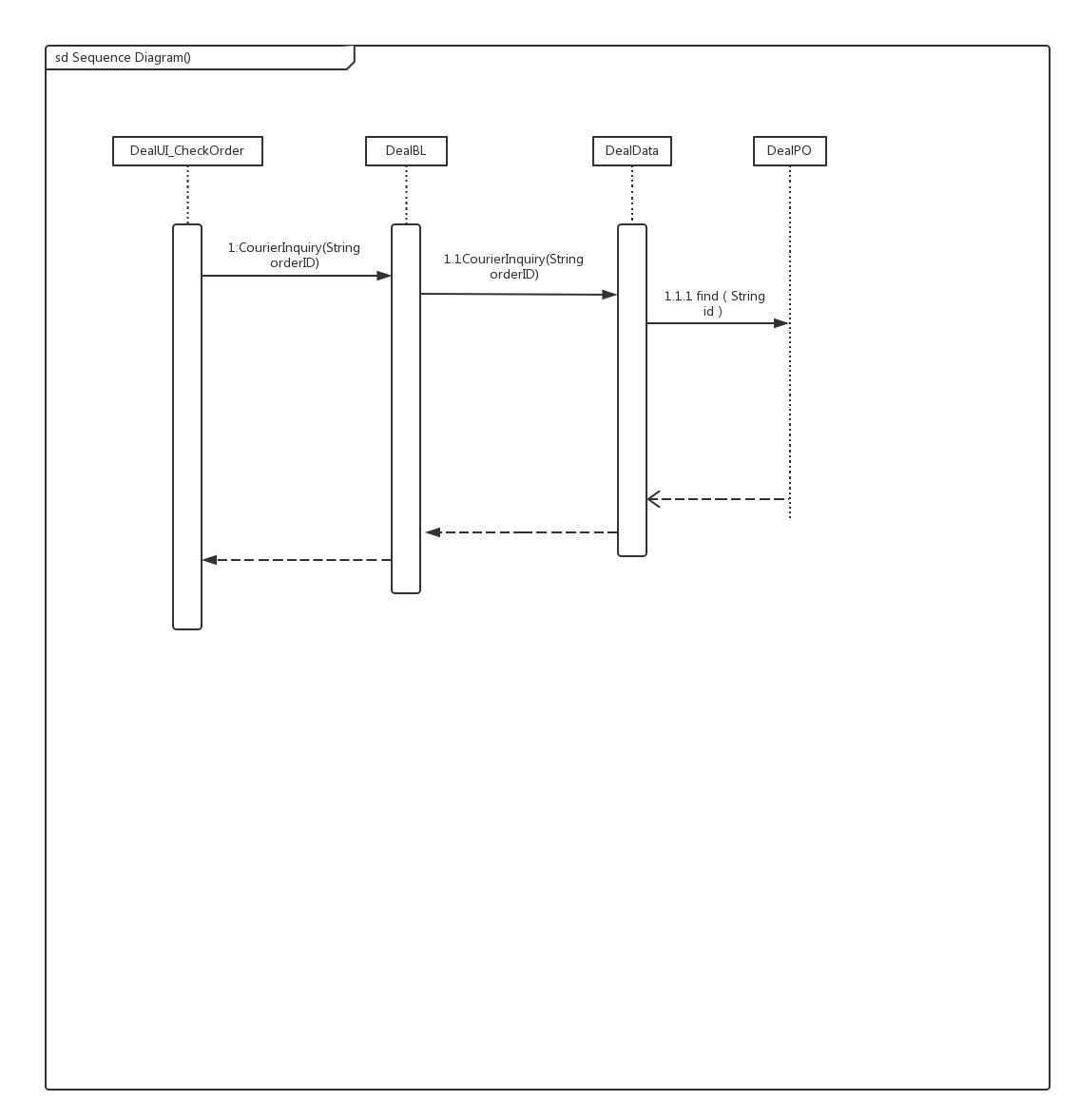
下图表明了在快递物流管理系统中，当快递员输入收件信息后，DealBL处理的相关对象之间的协作。



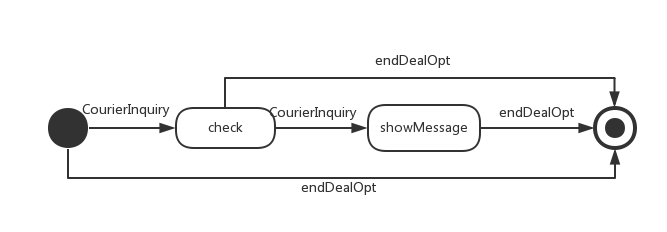
状态图如下：



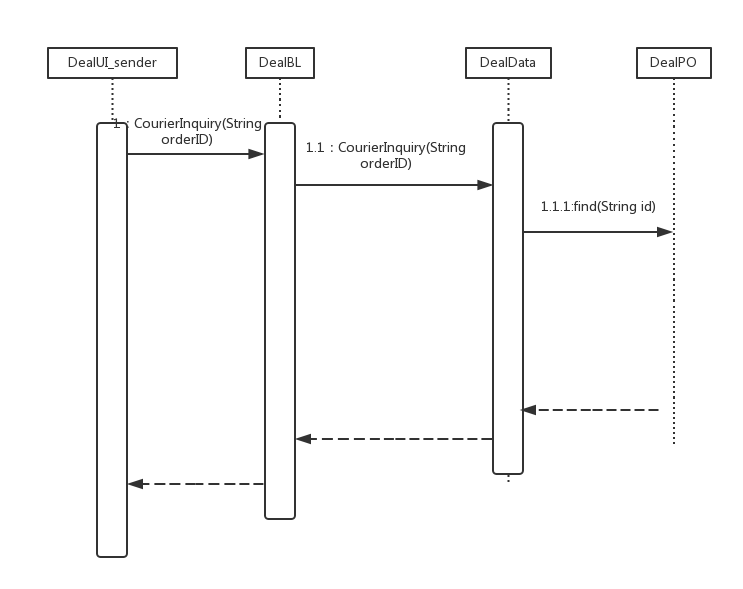
下图表明了在快递物流管理系统中，当快递员输入订单号查询订单信息后，DealBL处理的相关对象之间的协作。



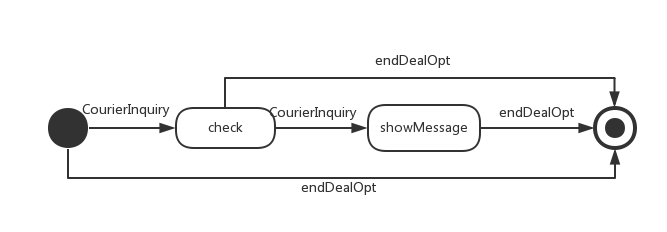
状态图如下：



下图表明了在快递物流管理系统中，当寄件人输入订单号查询订单信息后，DealBL处理的相关对象之间的协作。



状态图如下：



(5)业务逻辑层的设计原理

利用集中式控制风格，每个界面需要访问的业务逻辑直接调用与界面相关的类。。

### 4.1.3 invoicebl模块

（1）模块描述

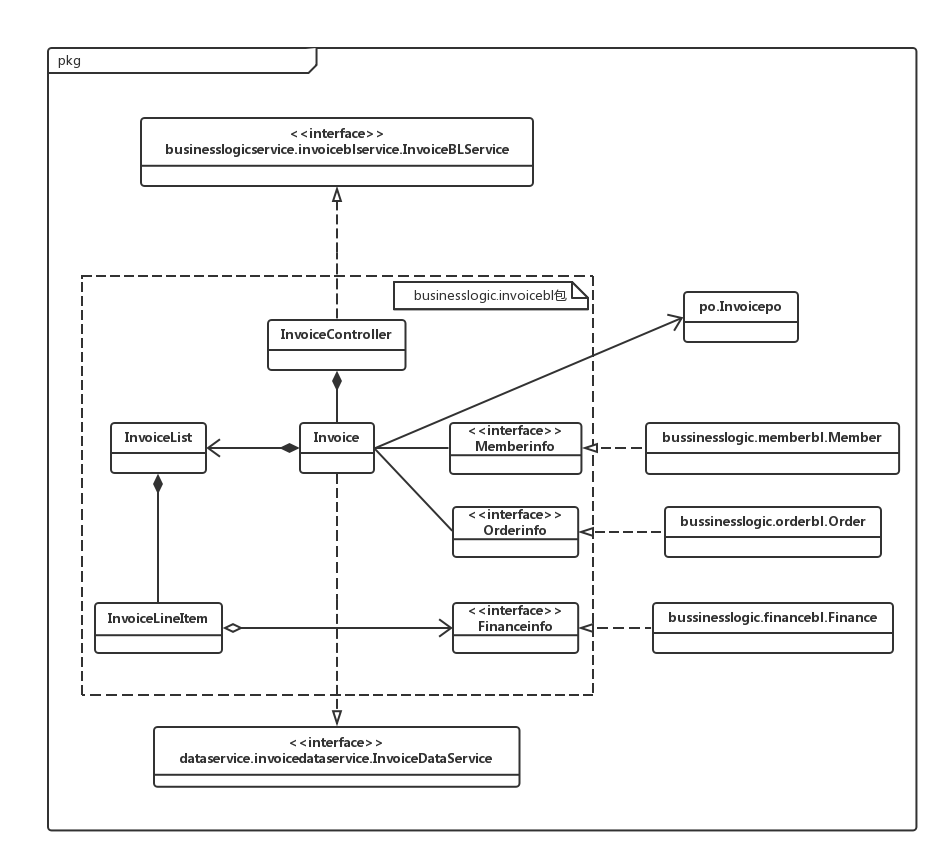
invoicebl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Invoicebl模块的职责及接口参见软件系统结构描述文档5.3.2.3。

（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.invoiceblservice.InvoiceBLService接口。业务逻辑层和数据层之间添加dataservice.invoicedataservice.InvoiceDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InvoiceController，这样InvoiceController会将对单据的业务逻辑处理委托给Invoice对象。InvoicePO是作为单据信息的持久化对象被添加到设计模型中去的。InvoiceLineItem保有计算有运费的单据中运费的计算的职责。而InvoiceList封装了关于InvoiceLineItem的数据集合的数据结构的秘密和列出所有信息的职责。Memberinfo、Financeinfo、Orderinfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

invoicebl模块的设计如图4.1.3-1所示



**4.1.3-1 invoice模块各个类的设计**

invoicebl模块各个类的职责如表4.1.3-2所示。

**表4.1.3-2 invoicebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| InvoiceController | 负责实现单据界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名和密码 |
| Invoice | 单据的领域模型对象，拥有一次生成单据所持有的各种信息，可以帮助完成单据界面所需要的服务 |

（3）模块内部类的接口规范

InvoiceController和Invoice的接口规范如表4.1.3-3和表4.1.3-4所示。

**表4.1.3-3 InvoiceController的接口规范**

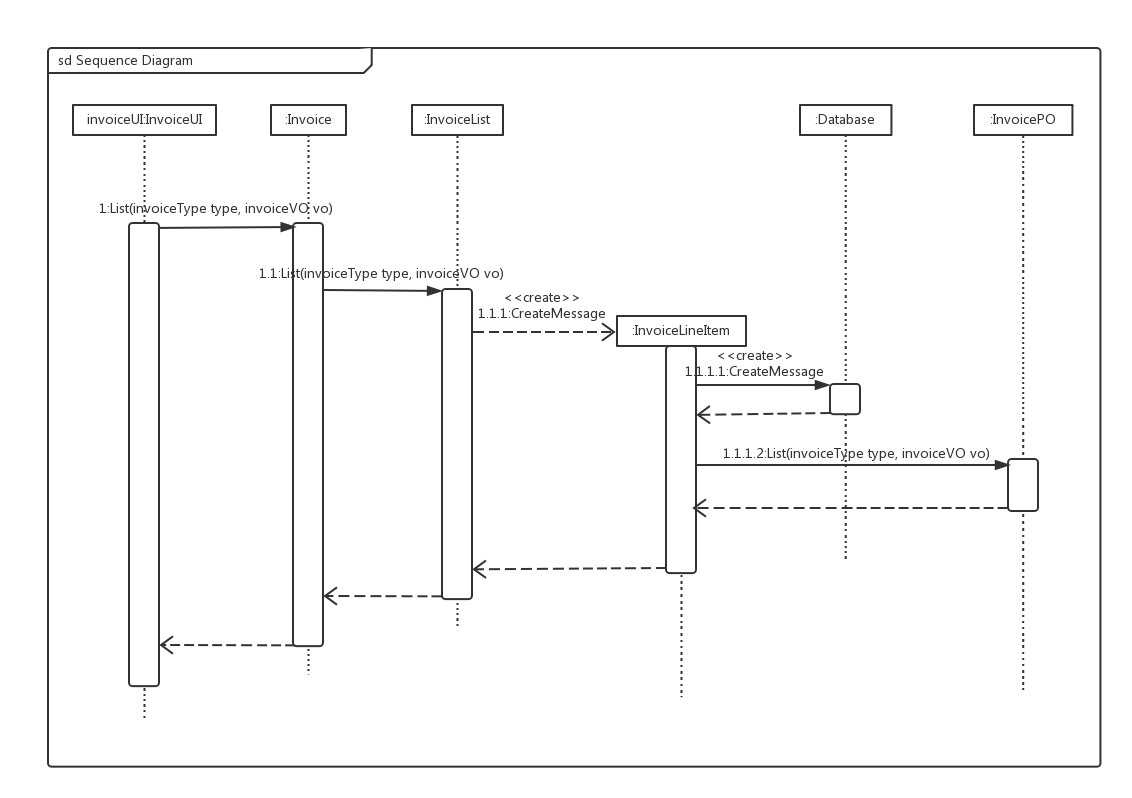
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InvoiceController. List | 语法 | Public resultMessage List(invoiceType type, invoiceVO vo) |
| 前置条件 | 已创建一个Invoice领域对象，并且已经选择单据的类别并输入单据的各项数据 |
| 后置条件 | 调用Invoice领域对象的List方法 |
| InvoiceController. inquiry | 语法 | Public invoiceVO inquiry(string invoiceID) |
| 前置条件 | 已创建一个Invoice领域对象，并且已经启动一个单据查看操作 |
| 后置条件 | 调用Invoice领域对象的inquiry方法 |
| InvoiceController. change | 语法 | Public invoicePO change(string invoiceID) |
| 前置条件 | 已创建一个Invoice领域对象，并且已经启动一个单据修改操作 |
| 后置条件 | 调用Invoice领域对象的change方法 |
| InvoiceController. delete | 语法 | Public void delete(invoicePO po) |
| 前置条件 | 已创建一个Invoice领域对象，并且已经启动一个单据删除操作 |
| 后置条件 | 调用Invoice领域对象的delete方法 |
| InvoiceController. approval | 语法 | Public ResultMessage approval(string invoiceID) |
| 前置条件 | 已创建一个Invoice领域对象，并且单据信息正确且合理 |
| 后置条件 | 调用Invoice领域对象的approval方法 |
| InvoiceController. endOpt | 语法 | Public void endOpt() |
| 前置条件 | 已创建一个Invoice领域对象，并且对单据的操作已经结束 |
| 后置条件 | 调用Invoice领域对象的endOpt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Invoice. List（invoiceType type, invoiceVO vo） | 加入一个信息列表对象 | |
| …… | …… | |

**表4.1.3-4 Invoice的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Invoice. List | 语法 | Public resultMessage List(invoiceType type, invoiceVO vo) |
| 前置条件 | 选择单据的类别并输入单据的各项数据 |
| 后置条件 | 检验数据是否正确且合理，并根据检查结果返回对应的单据生成界面 |
| Invoice. inquiry | 语法 | Public invoiceVO inquiry(string invoiceID) |
| 前置条件 | 启动一个单据查看操作 |
| 后置条件 | 返回该单据的具体信息 |
| Invoice. change | 语法 | Public invoicePO change(string invoiceID) |
| 前置条件 | 启动一个单据修改操作 |
| 后置条件 | 将该单据状态修改为草稿状态，并退回给对应提交单位进行修改 |
| Invoice. delete | 语法 | Public void delete(invoicePO po) |
| 前置条件 | 启动一个单据删除操作 |
| 后置条件 | 将该单据信息删除 |
| Invoice. approval | 语法 | Public ResultMessage approval(string invoiceID) |
| 前置条件 | 单据信息正确且合理 |
| 后置条件 | 将该单据状态修改为审批后状态 |
| Invoice. endOpt | 语法 | Public void endOpt() |
| 前置条件 | 对单据的操作已经结束 |
| 后置条件 | 结束此次单据审批操作，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 | |
| DatabaseFactory. getInvoiceDatabase | 得到invoice数据库的引用 | |
| invoiceDataService. Find(string id) | 根据ID查找invoicePO对象 | |
| invoiceDataService. Insert(invoicePO po) | 在数据库中插入invoicePO对象 | |
| invoiceDataService.delete(invoicePO po) | 在数据库中删除invoicePO对象 | |
| …… | …… | |

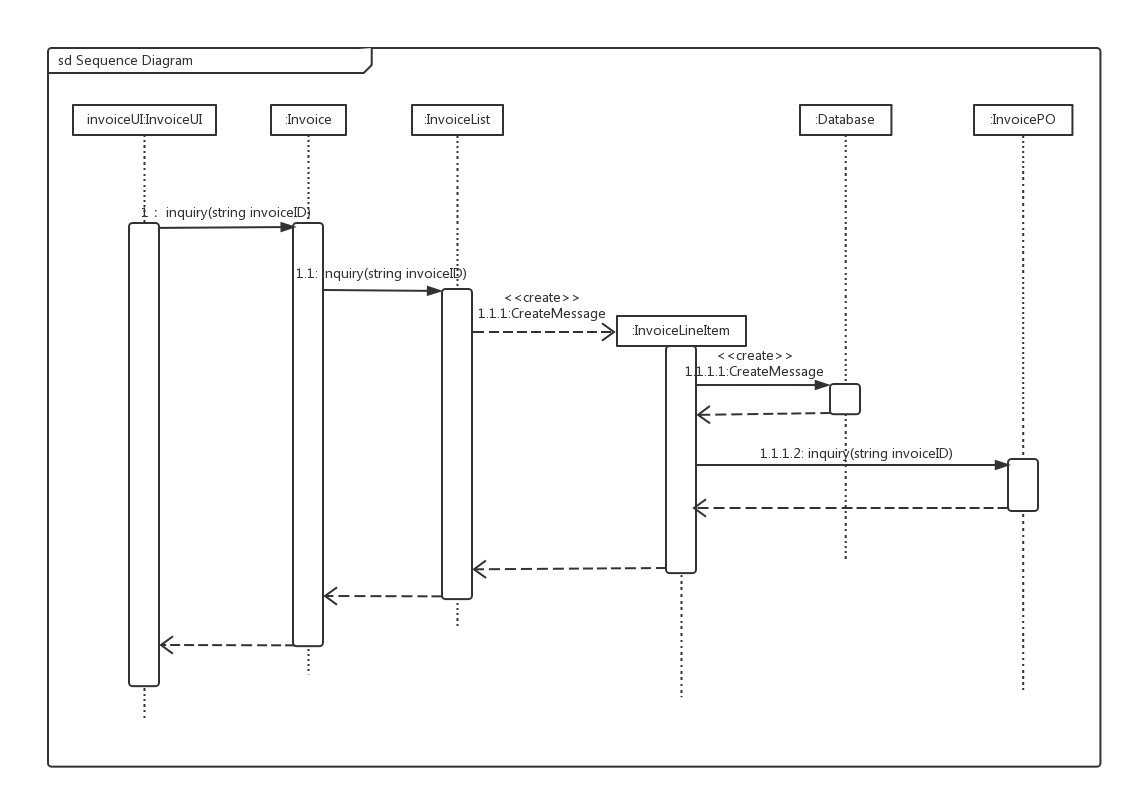
（4）业务逻辑的动态模型

图4.1.3-5.1表明了物流管理系统中，当业务员输入单据信息之后，单据业务逻辑处理的相关对象之间的协作。



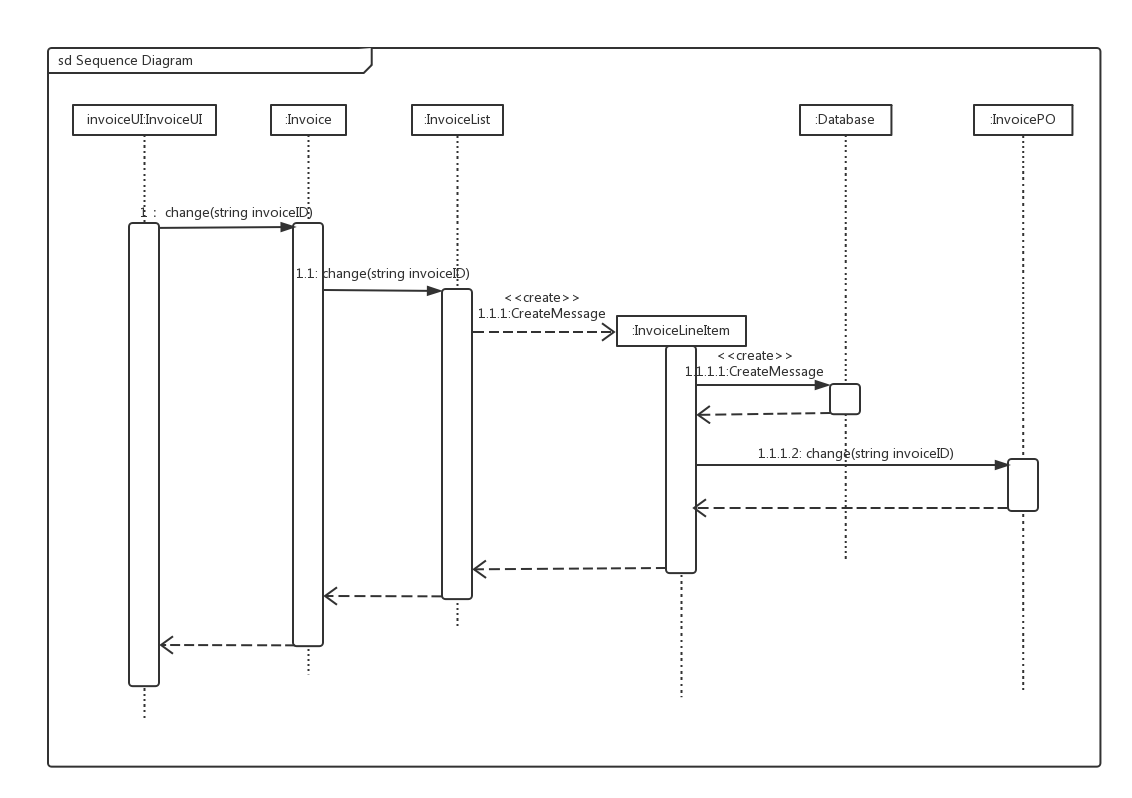
**图4.1.3-5.1 生成单据的顺序图**

图4.1.3-5.2表明了物流管理系统中，当业务员查看单据信息之后，单据业务逻辑处理的相关对象之间的协作。



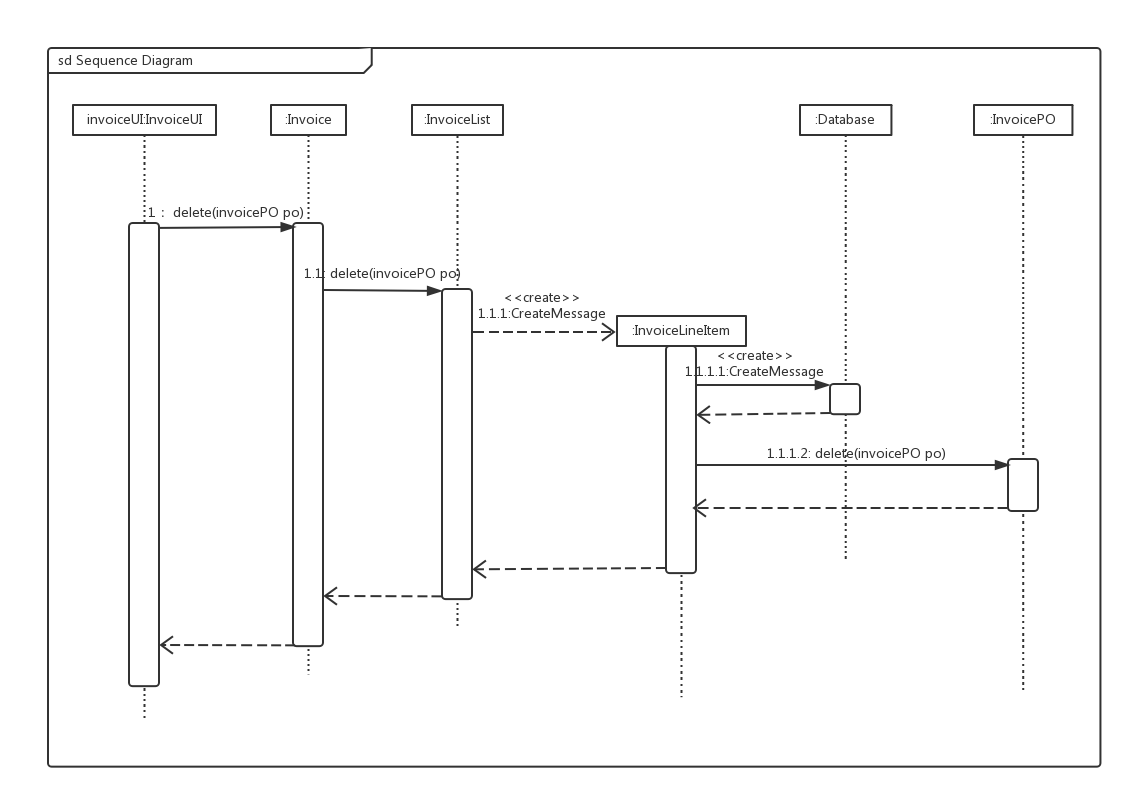
**图4.1.3-5.2 查看单据的顺序图**

图4.1.3-5.3表明了物流管理系统中，当业务员修改单据信息之后，单据业务逻辑处理的相关对象之间的协作。



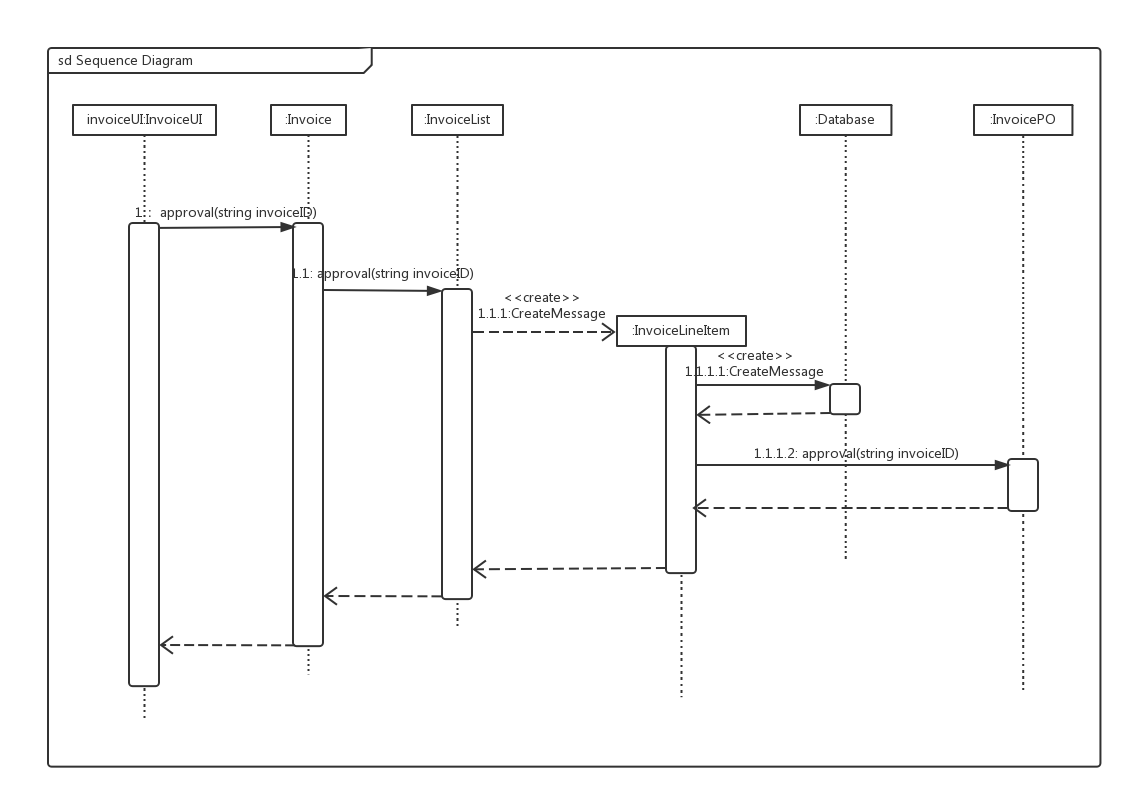
**图4.1.3-5.3 修改单据的顺序图**

图4.1.3-5.4表明了物流管理系统中，当业务员删除单据信息之后，单据业务逻辑处理的相关对象之间的协作。



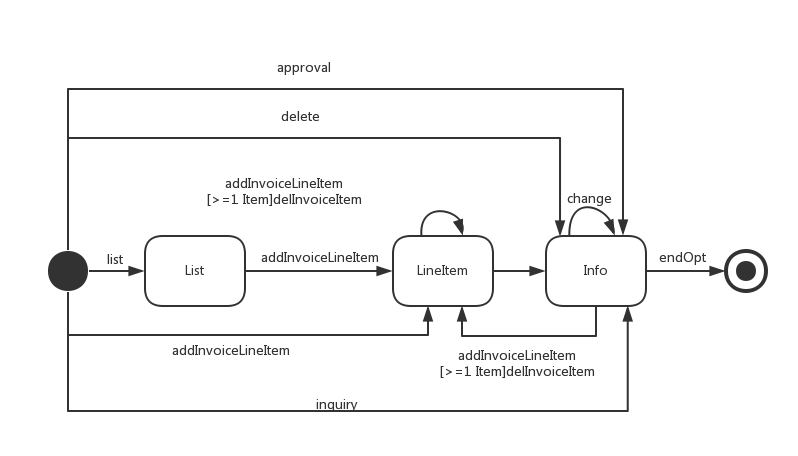
**图4.1.3-5.4 删除单据的顺序图**

图4.1.3-5.5表明了物流管理系统中，当业务员审批单据信息之后，单据业务逻辑处理的相关对象之间的协作。



**图4.1.3-5.5 删除单据的顺序图**

如图4.1.3-6所示的状态图描述了Invoice对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



**图4.1.3-6 Invoice对象状态图**

（5）业务逻辑层的设计原理

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

### 4.1.4 storagebl 模块

1. 模块概述

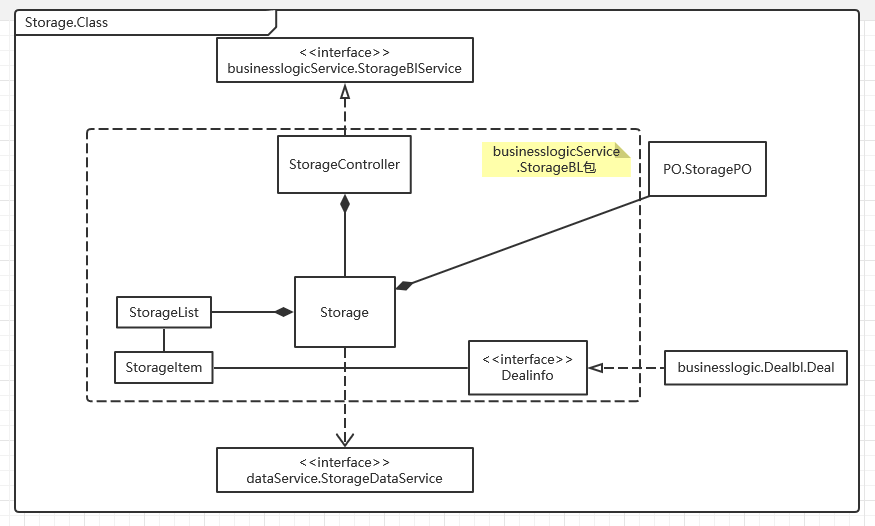
storagebl模块承担的需求参见需求规格说明文档的功能需求及相关的非功能需求

Storagebl模块的职责及接口参见软件系统结构描述文档

1. 整体结构

根据体系结构设计，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，我们会增加接口，比如businesslogicservice.StorageblService 、dataservice.StorageDateService。为了隔离业务逻辑职责和逻辑控制职责，我们增加了StorageController，这样StorageController会将对库存业务逻辑的处理委托给Storage对象。StoragePO是作为库存记录的持久化对象被添加到设计模型中的。StorageList封装了关于Storage的数据集合的数据结构的秘密和显示当前库存的职责。DealInfo是根据依赖导致原则，为了消除循环依赖产生的接口。

Storagebl模块的设计如下



Storage模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面需要的服务 |
| StorageController | 负责实现对应于库存界面需要的服务 |
| Storage | 库存的领域模型对象，拥有一次对库存操作所持有的库存信息，可以帮助库存界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户的信息，可以解决登陆问题 |

1. 模块的内部类的接口规范

#### StorageController的接口规范

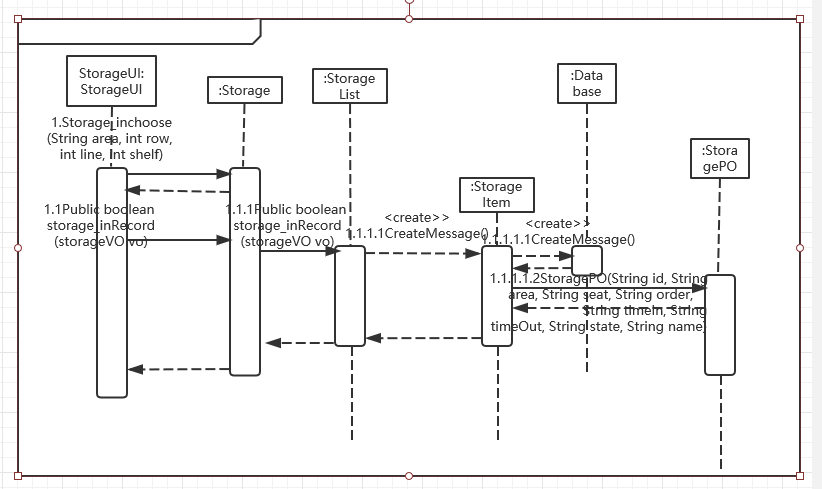
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| StorageController. Init | 语法 | PublicArrayList<StoragePO> init(ArrayList<StorageVO vo>) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的Init方法 |
| StorageController. Storage\_inChoose | 语法 | Public Boolean storage\_inChoose(string area, int row, int line, int shelf) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的Storage\_inChoose方法 |
| StorageController. storage\_inRecord | 语法 | Public boolean storage\_inRecord(storageVO vo) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的Storage\_inRecord方法 |
| StorageController Storage\_out | 语法 | Public void storage\_out(string orderID) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的Storage\_out方法 |
| StorageController. inquiryByTime | 语法 | Public ArrayList<StorageVO>inquiryByTime(string time1, string time2, string center) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的inquiryByTime方法 |
| StorageController.inquiryAll | 语法 | Public ArrayList<StorageVO> inquiryAll(Sring center) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的inquiryAll方法 |
| StorageController. inquiry | 语法 | Public StorageVO inquiry(string id) |
| 前置条件 | 已创建一个Storage领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Storage领域的inquiry方法 |
| StorageController. endStoreOpt | 语法 | Public void endStoreOpt() |
| 前置条件 | 已创建一个Storage领域对象 |
| 后置条件 | 调用Storage领域的endStoreOpt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Storage. Init | 初始化库存 | |
| Storage. Storage\_inChoose | 判断能否入库 | |
| Storage.storage\_inRecord | 存入新的库存信息 | |
| Storage. Storage\_out | 执行出库操作 | |
| Storage. inquiryByTime | 执行按时间查询操作 | |
| Storage.inquiryAll | 执行当前库存查询操作 | |
| Storage. inquiry | 执行单库存查询操作 | |
| Storage. endStoreOpt | 结束库存操作 | |

#### Storage的接口规范

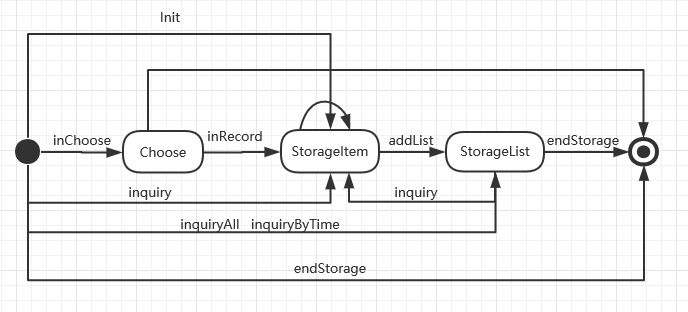
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Storage. Init | 语法 | PublicArrayList<StoragePO> init(ArrayList<StorageVO vo>) |
| 前置条件 | 启动一次库存信息初始化操作 |
| 后置条件 | 返回新输入的库存信息列表 |
| Storage. Storage\_inChoose | 语法 | Public Boolean storage\_inChoose(string area, int row, int line, int shelf) |
| 前置条件 | 选择入库位置 |
| 后置条件 | 检验该商品是否能入库 |
| Storage. storage\_inRecord | 语法 | Public boolean storage\_inRecord(storageVO vo) |
| 前置条件 | 该商品可以入库 |
| 后置条件 | 将对应商品信息及入库时间存入库存信息中 |
| Storage. Storage\_out | 语法 | Public void storage\_out(string orderID) |
| 前置条件 | 当前库存中存在商品ID对应的货物 |
| 后置条件 | 在库存信息中删除该商品，调用出库单 |
| Storage. inquiryByTime | 语法 | Public ArrayList<StorageVO>inquiryByTime(string time1, string time2, string center) |
| 前置条件 | 时间范围正确 |
| 后置条件 | 返回该中转仓库对应时间段内商品的库存信息 |
| Storage.inquiryAll | 语法 | Public ArrayList<StorageVO> inquiryAll(Sring center) |
| 前置条件 | 输入当前操作的中转中心业务员所属中转仓库 |
| 后置条件 | 返回当前中转仓库内的所有库存信息 |
| Storage. inquiry | 语法 | Public StorageVO inquiry(string id) |
| 前置条件 | 输入要查询的订单ID |
| 后置条件 | 检验库存中是否有该订单，如果有，则显示该订单的详细信息 |
| Storage. endStoreOpt | 语法 | Public void endStoreOpt() |
| 前置条件 | 所有库存操作已完成 |
| 后置条件 | 结束此次库存操作，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 | |
| DatabaseFactory. getStorageDatabase | 得到Storage数据库的服务的引用 | |
| StorageDataService. Find(string id) | 根据订单id在数据库中查找storagePO对象 | |
| StorageDataService. Findall(string center) | 在数据库中查找该中转仓库中的所有storagePO对象 | |
| StorageDataService. findByTime(string time1, string time2, string center) | 在数据库中查找该中转仓库中time1到time2时间段内的所有storagePO对象 | |
| StorageDataService. Insert(StoragePO po) | 在数据库中插入StoragePO对象 | |
| StorageDataService. Delete(StoragePO po) | 在数据库中删除storagePO对象 | |
| storageDataService. Update(storagePOpo) | 在数据库中更新storagePO对象 | |

(4)业务逻辑层的动态模型

下图表明在EMLS中，当有要对货物入库时，库存业务逻辑处理的相关对象之间的协作。



下图表明Storage对象在生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作。



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。

其他略。

### 4.1.5 financebl模块

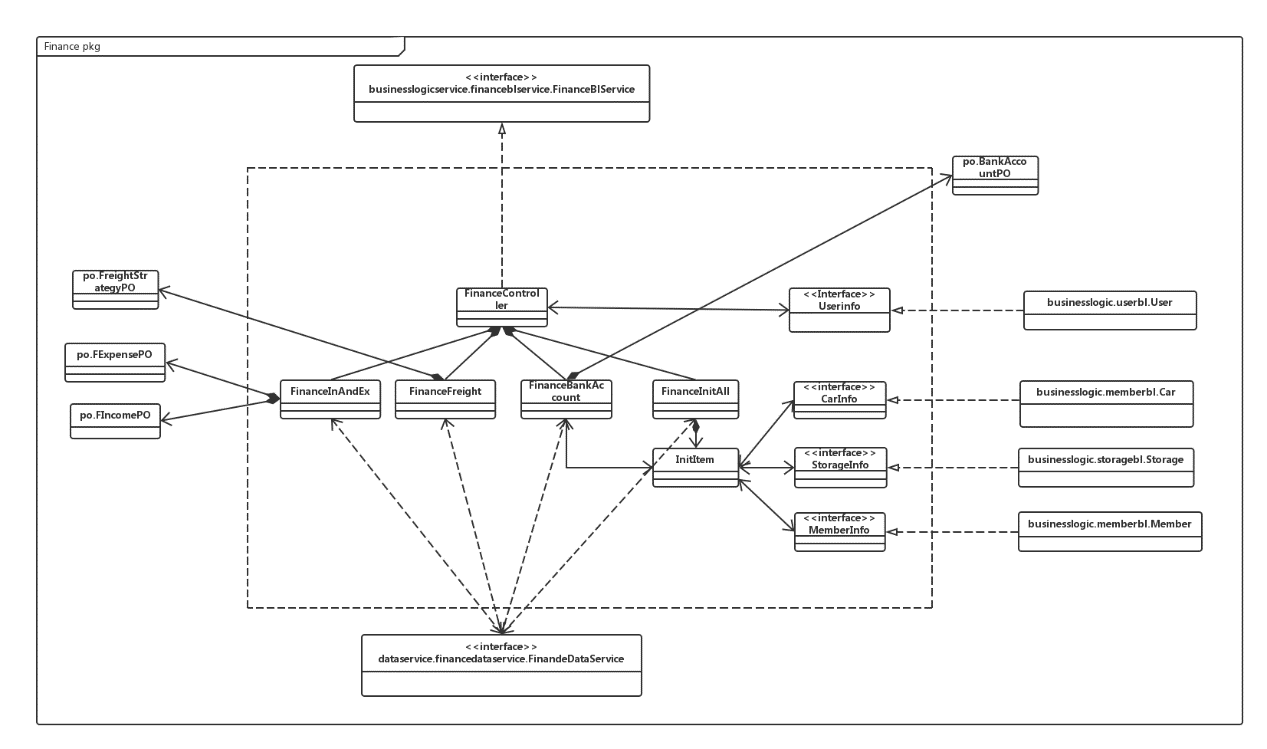
(1)模块概述

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

Financebl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

为隔离业务逻辑职责和逻辑控制职责，增加**FinanceController**，它会将对经济模块的业务逻辑处理委托给Finance对象。Financebl包中主要包括四个主要功能， **FinanceInAndEx**包含对系统收入项和支出项的查询和新建（付款单）操作以及报表工作，**FinanceBank**包含对银行账户的增删改查操作，**FinanceFreight**包含一个设置该公司收取运费的价格策略的操作，**FinanceInitAll**类用来进行期初建账操作，需要用到memberbl，FinanceBankBl，storagebl中的新建操作，**initItem**是一个期初建账时新建每个外部po的类，是memberbl，financeBankbl，storagebl的容器类。 FinanceFreightBl会影响Dealbl中运费价格的收取。各个financebl类都要使用userbl中的login功能，它也会反馈登录信息给userbl显示对应的人员界面。CarInfo,MemberInfo,StorageInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。



Financebl模块各个类的设计

Financebl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登陆界面所需要的服务 |
| FinanceController | 负责实现经济界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名和密码，可以解决登陆问题 |
| financeInAndEx | 经济模块的领域对象，可以实现对于收入项支出项的增加与各种查询操作，以及报表的生成与导出操作 |
| FinanceBankAccount | 经济模块的领域对象，可以实现对于银行账户的增删改查操作 |
| FinanceInitAll | 经济模块的领域对象，可以进行期初建账操作 |
| FinanceFreight | 经济模块的领域对象，可以制定公司收取的运费策略 |

（3）模块内部类的接口规范

**FinanceController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| FinanceController.inquiryIncome | 语法 | public FIncomeVO inquiryIncome(String id) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryIncome方法 |
| FinanceController.inquiryExpense | 语法 | public FExpenseVO inquiryExpense(String id) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryExpense方法 |
| FinanceController.  inquiryIncomeByHall | 语法 | public ArrayList<FIncomeVO> inquiryIncomeByHall(String center) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryIncome方法 |
| FinanceController.  inquiryIncomeByTime | 语法 | public ArrayList<FIncomeVO> inquiryIncomeByTime(String time1, String time2) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryIncomeByTime方法 |
| FinanceController.  inquiryInByTimeHall | 语法 | public ArrayList<FIncomeVO> inquiryInByTimeHall(String time1, String time2,String center) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryInByTimeHall方法 |
| FinanceController.  inquiryExpenseByTime | 语法 | public ArrayList<FExpenseVO> inquiryExpenseByTime(String time1, String time2) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的inquiryExpenseByTime方法 |
| FinanceController.  addExpense | 语法 | public boolean addExpense(FExpenseVO vo) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的addExpense方法 |
| FinanceController.  deleteExpense | 语法 | public boolean deleteExpense(FExpensePO po) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的deleteExpense方法 |
| FinanceController.  changeExpense | 语法 | public FExpenseVO changeExpense(String expenseID) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的changeExpense方法 |
| FinanceController.  createForm | 语法 | public boolean createForm(String time1, String time2) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInAndEx领域对象的createForm方法 |
| FinanceController.  deriveForm | 语法 | public boolean deriveForm(String operator) |
| 前置条件 | 已创建一个FinanceInAndEx领域对象，已生成一个报表 |
| 后置条件 | 调用FinanceInAndEx领域对象的deriveForm方法 |
| FinanceController.  initAll | 语法 | public boolean initAll(String operator) |
| 前置条件 | 已创建一个FinanceInitAll领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceInitAll领域对象的initAll方法 |
| FinanceController.  initFreight | 语法 | public boolean initFreight() |
| 前置条件 | 已创建一个FinanceFreight领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceFreight领域对象的initFreight方法 |
| FinanceController.  setFreight | 语法 | public FreightStrategyVO setFreight(FreightStrategyVO vo) |
| 前置条件 | 已创建一个FinanceFreight领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceFreight领域对象的setFreight方法 |
| FinanceController.  addAccount | 语法 | public boolean addAccount(BankAccountVO vo) |
| 前置条件 | 已创建一个FinanceBankAccount领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceBankAccount领域对象的addAccount方法 |
| FinanceController.  deleteAccount | 语法 | public boolean deleteAccount(String bankAccount) |
| 前置条件 | 已创建一个FinanceBankAccount领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceBankAccount领域对象的deleteAccount方法 |
| FinanceController.  changeAccount | 语法 | public BankAccountVO changeAccount(String accountName) |
| 前置条件 | 已创建一个FinanceBankAccount领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceBankAccount领域对象的changeAccount方法 |
| FinanceController.  inquiryAccount | 语法 | public BankAccountVO inquiryAccount(String accountName) |
| 前置条件 | 已创建一个FinanceBankAccount领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceBankAccount领域对象的inquiryAccount方法 |
| FinanceController.  inquiryAccountByBank | 语法 | public ArrayList<BankAccountVO> inquiryAccountByBank(String BankName) |
| 前置条件 | 已创建一个FinanceBankAccount领域对象，并且输入符合输入规则 |
| 后置条件 | 调用FinanceBankAccount领域对象的inquiryAccountByBank方法 |
| FinanceController.  endFinanceOpt | 语法 | public void endFinanceOpt() |
| 前置条件 | 已创建一个对应领域对象 |
| 后置条件 | 调用对应领域对象的endFinanceOpt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| FinanceInAndEx.  inquiryIncome | 查询收入项 | |
| FinanceInAndEx.  inquiryExpense | 查询支出项 | |
| FinanceInAndEx.  inquiryIncomeByHall | 按营业厅查找收入项 | |
| …… | ……（见下面的几个表格） | |

**FinanceInAndEx的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| FinanceInAndEx.  inquiryIncome | 语法 | | public FIncomeVO inquiryIncome(String id) |
| 前置条件 | | 输入要查询的收入项编号 |
| 后置条件 | | 查找对应收入项并显示相应信息 |
| FinanceInAndEx.  inquiryExpense | 语法 | | public FExpenseVO inquiryExpense(String id) |
| 前置条件 | | 输入要查询的支出项编号 |
| 后置条件 | | 查找对应支出项并显示相应信息 |
| FinanceInAndEx.  inquiryIncomeByHall | 语法 | | public ArrayList<FIncomeVO> inquiryIncomeByHall(String center) |
| 前置条件 | | 输入要查询的营业厅编号 |
| 后置条件 | | 返回该营业厅的所有收入项（IncomePO）列表 |
| FinanceInAndEx.  inquiryIncomeByTime | 语法 | | public ArrayList<FIncomeVO> inquiryIncomeByTime(String time1, String time2) |
| 前置条件 | | 输入时间段信息 |
| 后置条件 | | 返回该段时间内的所有收入项列表 |
| FinanceInAndEx.  inquiryInByTimeHall | 语法 | | public ArrayList<FIncomeVO> inquiryInByTimeHall(String time1, String time2,String center) |
| 前置条件 | | 输入时间段信息与营业厅编号 |
| 后置条件 | | 返回该营业厅在该段时间内的所有收入项列表 |
| FinanceInAndEx.  inquiryExpenseByTime | 语法 | | public ArrayList<FExpenseVO> inquiryExpenseByTime(String time1, String time2) |
| 前置条件 | | 选择查询支出项并输入相应时间信息 |
| 后置条件 | | 返回该时间段内的支出项信息列表 |
| FinanceInAndEx.  addExpense | 语法 | | public boolean addExpense(FExpenseVO vo) |
| 前置条件 | | 选择新增一个支出项并输入相应信息 |
| 后置条件 | | 新建一个支出项信息PO |
| FinanceInAndEx.  deleteExpense | 语法 | | public boolean deleteExpense(FExpensePO po) |
| 前置条件 | | 选择对应支出项并选择删除 |
| 后置条件 | | 删除一个支出项信息PO |
| FinanceInAndEx.  changeExpense | 语法 | | public FExpenseVO changeExpense(FExpenseVOvo) |
| 前置条件 | | 输入更改后的支出项信息 |
| 后置条件 | | 对该支出项进行更改 |
| FinanceInAndEx.  createForm | 语法 | | public boolean createForm(String time1, String time2) |
| 前置条件 | | 输入对应的时间段信息 |
| 后置条件 | | 新建一个报表 |
| FinanceInAndEx.  deriveForm | 语法 | | public boolean deriveForm(String operator) |
| 前置条件 | | 输入操作者id，且已生成一个报表 |
| 后置条件 | | 将该报表导出为excel表格 |
| FinanceInAndEx.  getTotalIn | 语法 | | public double getTotalIn(ArrayList<FIncomeVO> vo) |
| 前置条件 | | 已获得一个收入项列表 |
| 后置条件 | | 返回列表中总的收入金额 |
| FinanceInAndEx.  getTotalEx | 语法 | | public double getTotalEx(ArrayList<FExpenseVO> vo) |
| 前置条件 | | 已获得一个支出项列表 |
| 后置条件 | | 返回列表中总的支出金额 |
| FinanceInAndEx.  endFinanceOpt | 语法 | | public void endFinanceOpt() |
| 前置条件 | | 已结束收入支出项操作 |
| 后置条件 | | 结束此次经济操作回合，持久化更新设计的领域对象的数据 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| FinanceDataService.  findIncome(String id) | | 根据ID在finance数据库中查找IncomePO对象 | |
| FinanceDataService.  findExpense(String id) | | 根据ID在finance数据库中查找ExpensePO对象 | |
| FinanceDataService.  findbyHall(String hall) | | 根据营业厅编号在finance数据库中查找IncomePO对象列表 | |
| FinanceDataService.  findIncome(String time1,String time2) | | 根据时间段信息在finance数据库中查找IncomePO对象列表 | |
| FinanceDataService.  findHallIncome(String time1,String time2,String hall) | | 根据时间和营业厅信息在finance数据库中查找IncomePO对象列表 | |
| FinanceDataService.  findExpense(String time1,String time2) | | 根据时间段信息在finance数据库中查找ExpensePO对象 | |
| FinanceDataService.  insertExpense(FExpensePO po) | | 在数据库中插入ExpensePO对象 | |
| FinanceDataService.  deleteExpense(FExpensePO po) | | 在数据库中删除ExpensePO对象 | |
| FinanceDataService.  updateExpense(FExpensePO po) | | 在数据库中更新ExpensePO对象 | |

**FinanceInitAll的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| FinanceInitAll.  initAll | 语法 | | public boolean initAll(String operator) |
| 前置条件 | | 启动一个期初建账回合 |
| 后置条件 | | 保存旧账并新建一个新帐，保存该帐的初始状态，并将该帐作为日后操作的总账 |
| FinanceInitAll.  save | 语法 | | public void save(String operator) |
| 前置条件 | | 启动一个期初建账 |
| 后置条件 | | 保存账目信息 |
| FinanceInitAll.  endFinanceOpt | 语法 | | Public void endFinanceOpt() |
| 前置条件 | | 结束期初建账操作 |
| 后置条件 | | 结束此次操作回合，并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 | |
| FinanceDataService.  initAll(String operator) | | 在数据库中建立一个新的空帐目 | |
| FinanceDataService.  save(String operator) | | 在数据库中保存该账目 | |

**FinanceFreight的接口规范**

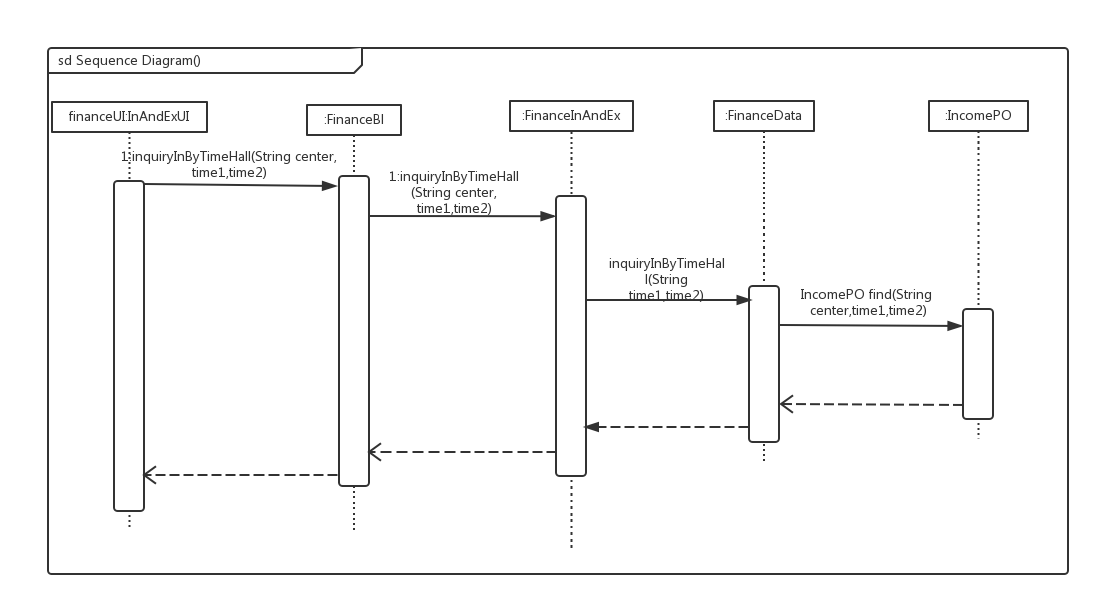
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| FinanceFreight.  initFreight | 语法 | | public FreightStrategyVO initFreight() |
| 前置条件 | | 启动一次更改运费策略操作 |
| 后置条件 | | 获得系统运费策略 |
| FinanceFreight.  setFreight | 语法 | | public FreightStrategyVO setFreight(FreightStrategyVO vo) |
| 前置条件 | | 输入新的运费策略vo |
| 后置条件 | | 对运费策略进行更改 |
| FinanceFreight.  endFinanceOpt | 语法 | | Public void endFinanceOpt() |
| 前置条件 | | 运费策略更改结束 |
| 后置条件 | | 结束此次运费策略更改操作，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 | |
| FinanceDataService.  find() | | 在Finance数据库中查找freightPO对象 | |
| FinanceDataService.  initFreight() | | 在Finance数据库中新建一个空的freightPO对象 | |
| FinanceDataService.  add(FreightStrategyVO vo) | | 在Finance数据库中新建一个FreightPO对象 | |
| FinanceDataService.  delete(FreightStrategyPO po) | | 在Finance数据库中删除一个FreightPO对象 | |
| FinanceDataService.  update(FreightStrategyPO po) | | 在Finance数据库中更新FreightPO对象 | |

**FinanceBankAccount的接口规范**

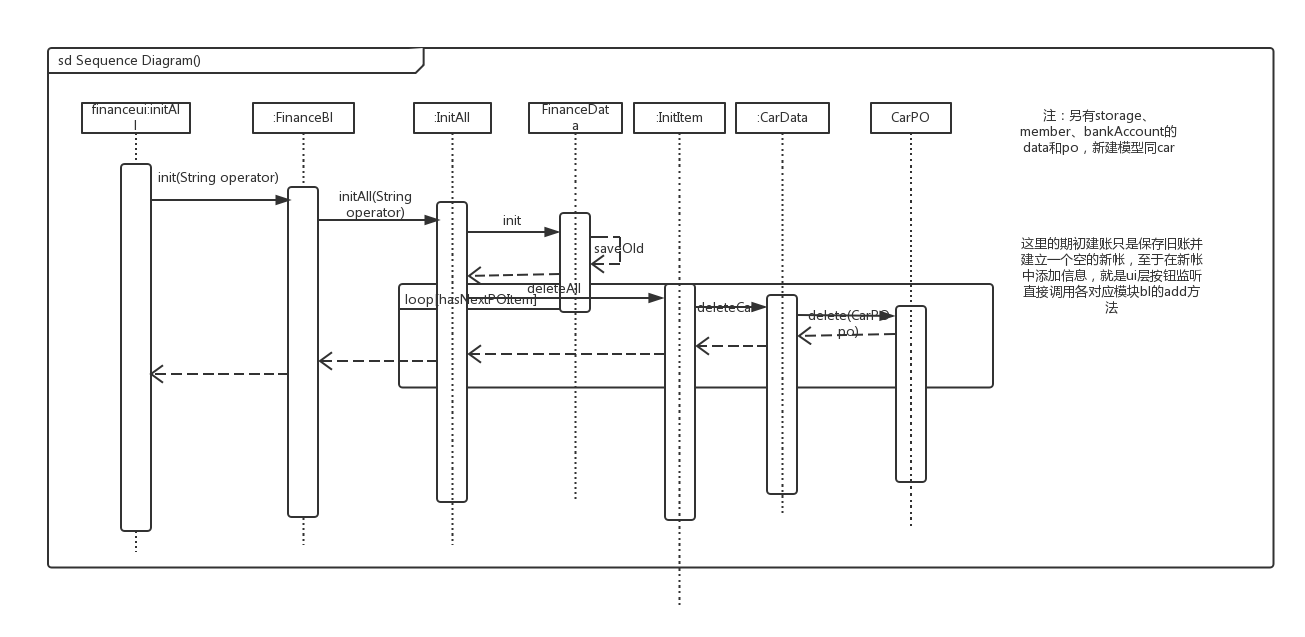
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| FinanceBankAccount.  addAccount | 语法 | | public boolean addAccount(BankAccountVO vo) |
| 前置条件 | | 选择新建一个bankAccount，并输入对应信息 |
| 后置条件 | | 新建一个银行账户 |
| FinanceBankAccount.  deleteAccount | 语法 | | public boolean deleteAccount(String bankAccount) |
| 前置条件 | | 输入银行账户名，并选择删除 |
| 后置条件 | | 删除一个银行账户 |
| FinanceBankAccount.  changeAccount | 语法 | | public BankAccountVO changeAccount(String accountName) |
| 前置条件 | | 输入银行账户名，并选择更改 |
| 后置条件 | | 对该银行账户进行更改并返回对应vo |
| FinanceBankAccount.  inquiryAccount | 语法 | | public BankAccountVO inquiryAccount(String accountName) |
| 前置条件 | | 输入银行账户名进行查询 |
| 后置条件 | | 查询对应银行账户，返回对应详细信息 |
| FinanceBankAccount.  inquiryAccountByBank | 语法 | | public ArrayList<BankAccountVO> inquiryAccountByBank(String BankName) |
| 前置条件 | | 输入银行名称进行查询 |
| 后置条件 | | 返回该银行下的所有银行账户列表 |
| FinanceBankAccount.  endFinanceOpt | 语法 | | public void endFinanceOpt() |
| 前置条件 | | 结束银行账户操作 |
| 后置条件 | | 结束此次操作回合，并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 | |
| FinanceDataService.  find(String id) | | 根据id在finance数据库中查找BankAccountPO对象 | |
| FinanceDataService.  findByBank(String bankName) | | 根据银行名称在finance数据库中查找BankAccountPo对象列表 | |
| FinanceDataService.  insertAccount(BankAccountPO po) | | 在finance数据库中插入新的BankAccountPo对象 | |
| FinanceDataService.  deleteAccount(BankAccountPO po) | | 在finance数据库中删除一个BankAccountPo对象 | |
| FinanceDataService.  updateAccount(BankAccountPO po) | | 在finance数据库中更新一个BankAccountPo对象 | |
| FinanceDataService.  initAccount() | | 在finance数据库中新建一个空的bankAccount对象 | |

(4)业务逻辑层的动态模型

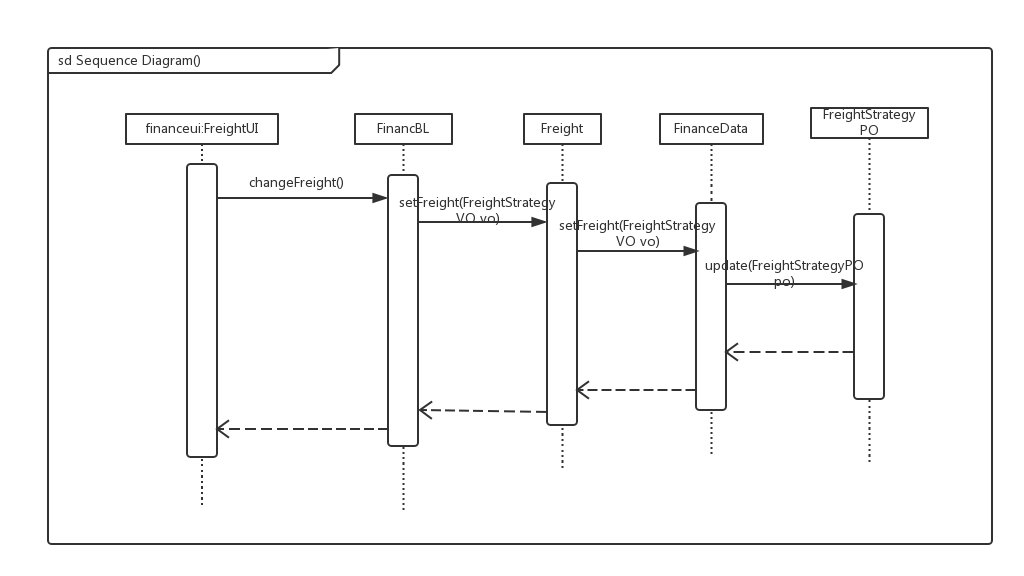
下图为用户根据时间和营业厅对收入项进行查询时的顺序图



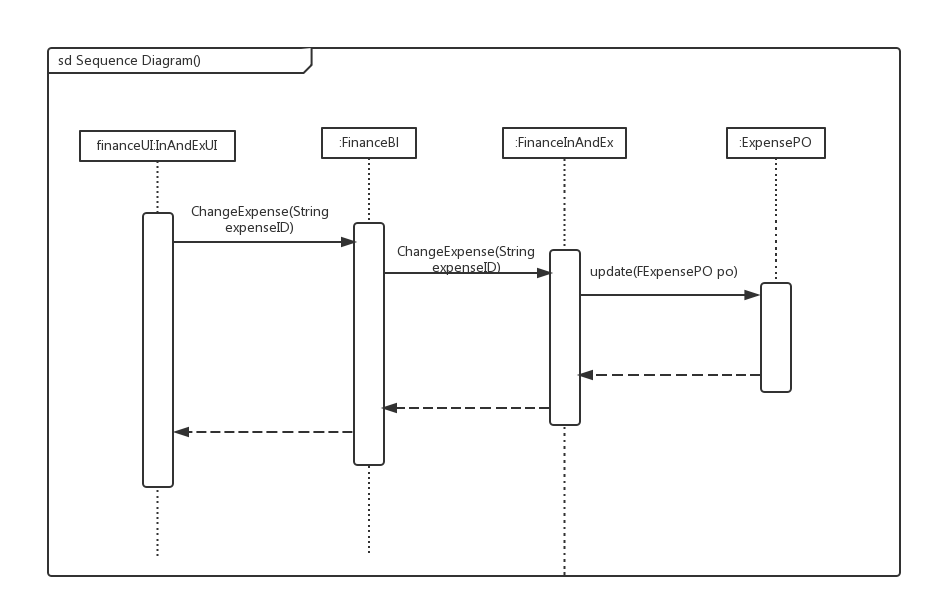
下图为用户进行期初建账时的顺序图



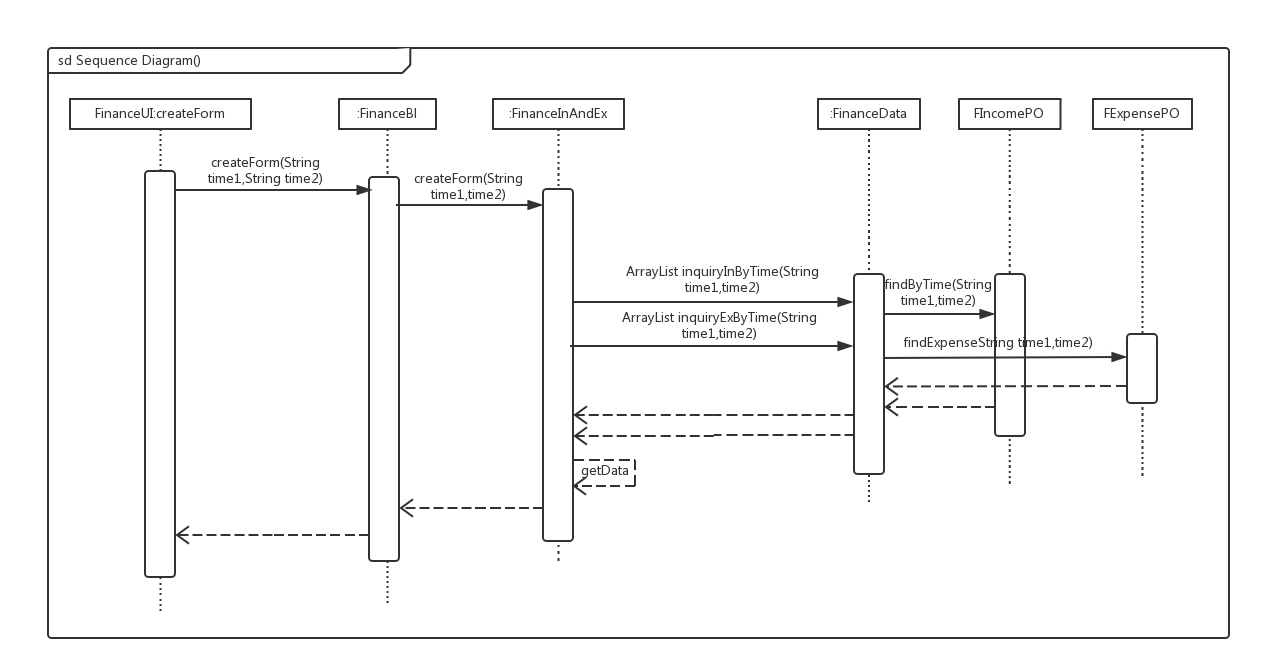
下图为用户进行运费策略制定时的顺序图



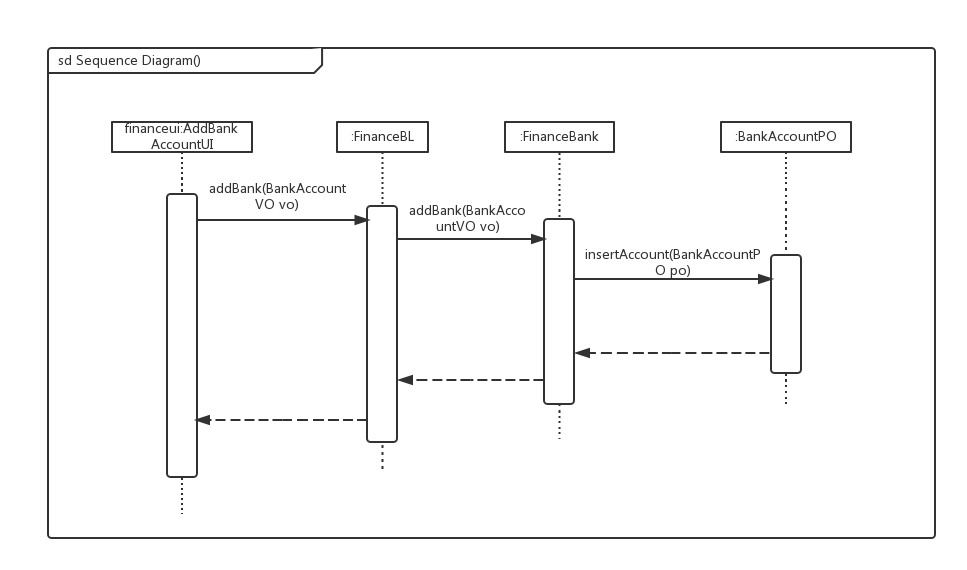
下图为用户对支出项信息进行更改时的顺序图

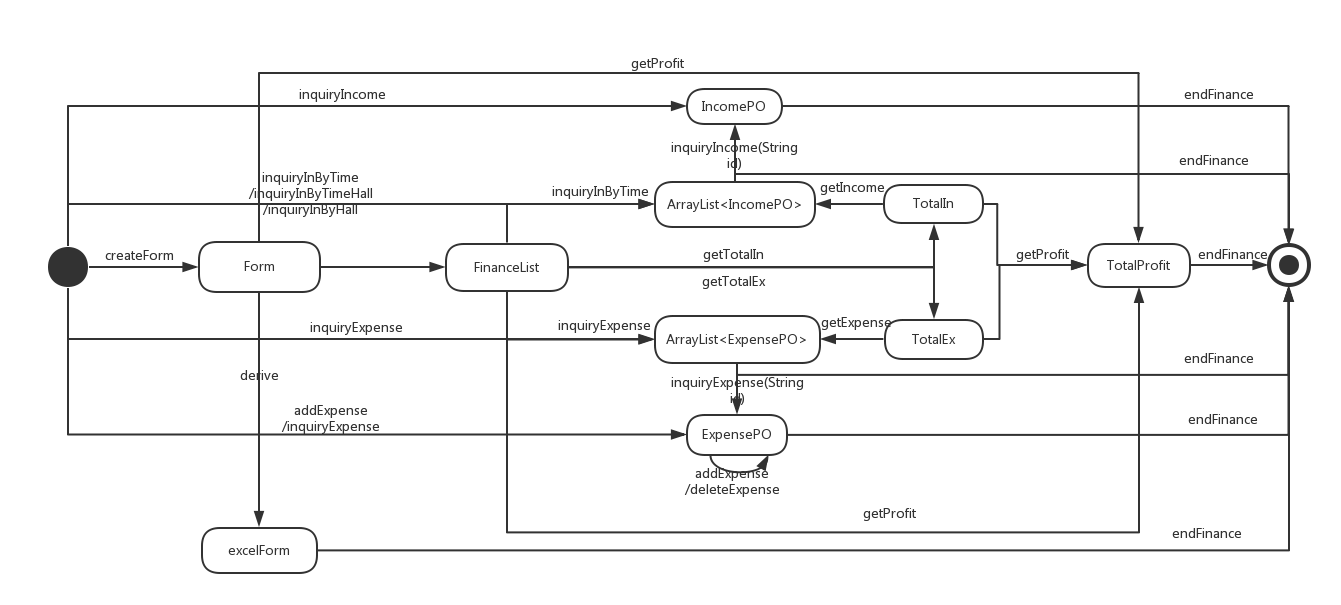


下图为用户生成报表时的顺序图

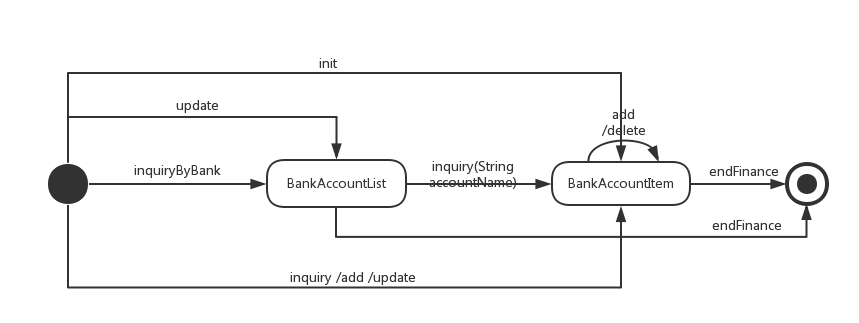


下图为用户新建银行账户时的顺序图

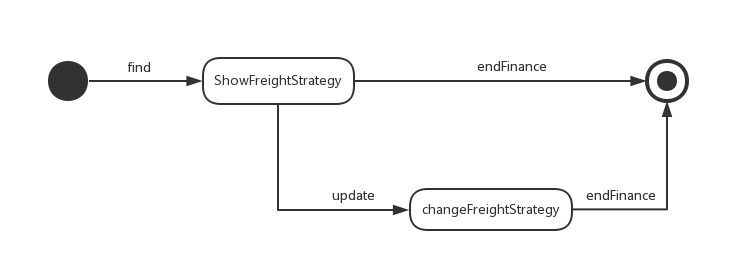


下图表示FinanceInAndEx领域对象生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作

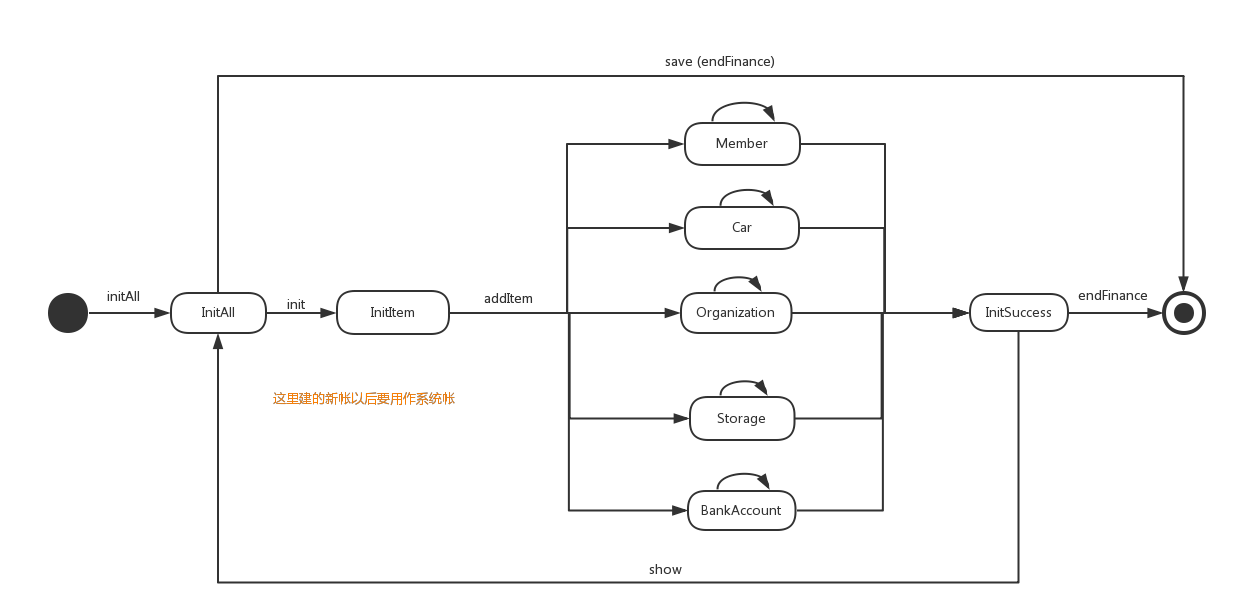
下图表示FinanceBankAccount领域对象生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作



下图表示FinanceFreight领域对象生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作



下图表示FinanceFreight领域对象生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作



(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。

### 4.1.6 memberbl模块

（1）模块描述

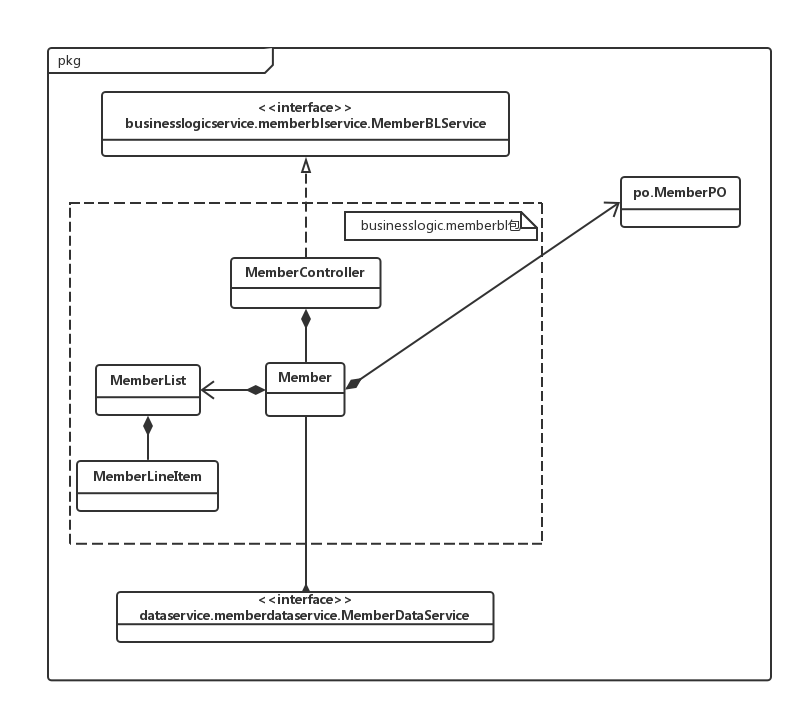
memberbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Memberbl模块的职责及接口参见软件体系结构描述文档表5.3.2.6。

（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.memberblservice.MemberBLService接口。业务逻辑层和数据层之间添加dataservice.memberdataservice.MemberDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了MemberController，这样MemberController会将对人事的业务逻辑处理委托给Member对象。MemberPO是作为人事信息的持久化对象被添加到设计模型中去的。MemberLineItem保有具体的人事信息的职责。而MemberList封装了关于MemberLineItem的数据集合的数据结构的秘密的职责。

memberbl模块的设计如图4.1.6-1所示。



**图4.1.6-1 memberbl模块各个类的设计**

memberbl模块各个类的职责如表4.1.6-2所示。

**表4.1.6-2 memberbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| MemberController | 负责实现人事管理界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名和密码，可以解决登陆问题 |
| Member | 人事管理的领域模型对象，拥有管理人员、司机、车辆的信息，可以帮助完成人事管理界面所需要的服务 |

（3）模块内部类的接口规范

MemberController和Member的接口规范如表4.1.6-3和表4.1.6-4所示。

**表4.1.6-3 MemberController的接口规范**

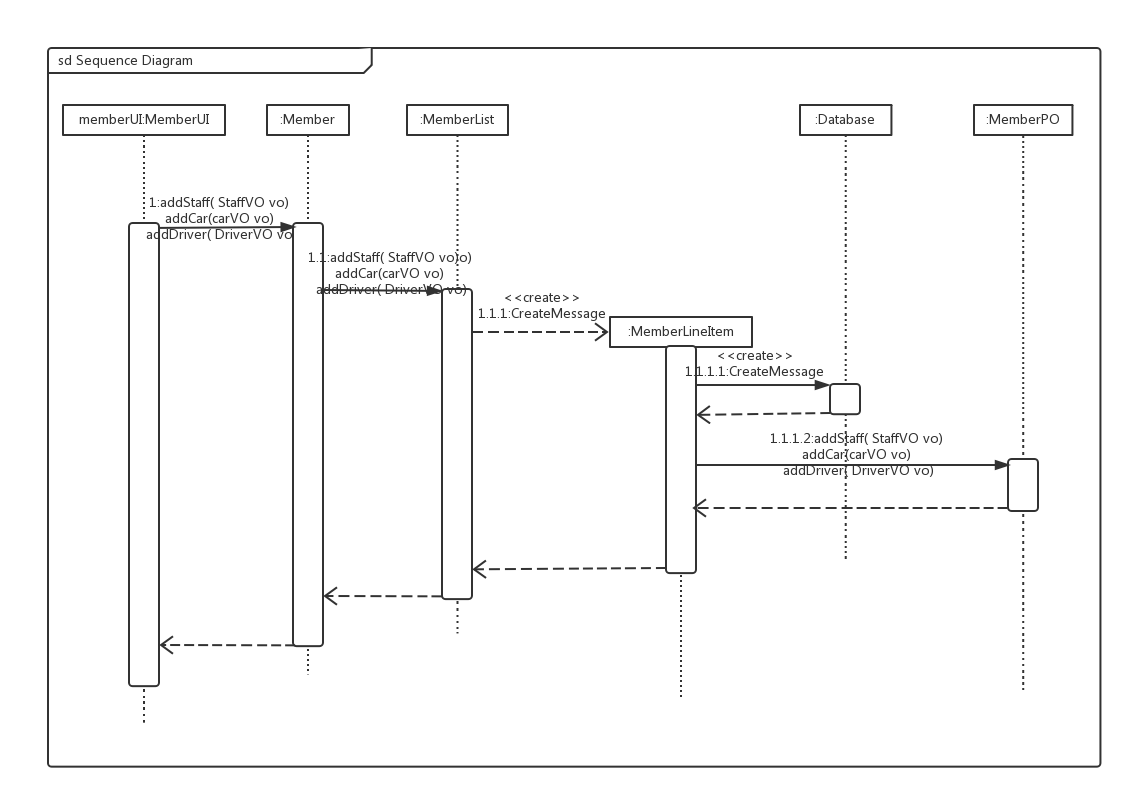
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| MemberController. find | 语法 | Public resultMessage find( stringID) |
| 前置条件 | 已创建一个Member领域对象，并且输入id |
| 后置条件 | 调用Member方法领域对象的find方法 |
| MemberController. addStaff | 语法 | Public Boolean addStaff( StaffVO vo) |
| 前置条件 | 建一个Member领域对象，并且输入员工基本信息 |
| 后置条件 | 调用Member方法领域对象的addstaff方法 |
| MemberController. deleteStaff | 语法 | Public void deleteStaff( staffPO po) |
| 前置条件 | 建一个Member领域对象，并且选择删除员工 |
| 后置条件 | 调用Member方法领域对象的deleteStaff方法 |
| MemberController. changeStaff | 语法 | Public resultMessage changeStaff( staffVo vo) |
| 前置条件 | 建一个Member领域对象，并且选择更改员工信息，输入更改后的员工vo |
| 后置条件 | 调用Member方法领域对象的changeStaff方法 |
| MemberController. addCar | 语法 | Public boolean addCar(carVO vo) |
| 前置条件 | 建一个Member领域对象，并且选择增加车辆信息，输入车辆基本信息 |
| 后置条件 | 调用Member方法领域对象的addCar方法 |
| MemberController. deleteCar | 语法 | Public void deleteCar(CarVO vo) |
| 前置条件 | 建一个Member领域对象，并且选择删除车辆信息 |
| 后置条件 | 调用Member方法领域对象的deleteCar方法 |
| MemberController. changeCar | 语法 | Public resultMessage changeCar( carVO vo) |
| 前置条件 | 建一个Member领域对象，并且选择修改车辆信息 |
| 后置条件 | 调用Member方法领域对象的changeCar方法 |
| MemberController. addDriver | 语法 | Public Boolean addDriver( DriverVO vo) |
| 前置条件 | 建一个Member领域对象，并且输入司机基本信息 |
| 后置条件 | 调用Member方法领域对象的addDriver方法 |
| MemberController. deleteDriver | 语法 | Public void deleteDriver(DriverPO po) |
| 前置条件 | 建一个Member领域对象，并且选择删除司机信息 |
| 后置条件 | 调用Member方法领域对象的deleteDriver方法 |
| MemberController. changeDriver | 语法 | Public resultMessage changeDriver(DriverVo vo) |
| 前置条件 | 建一个Member领域对象，并且选择修改司机信息 |
| 后置条件 | 调用Member方法领域对象的changeDriver方法 |
| MemberController. setWages | 语法 | Public resultMessage setWages( string job, int wages, int cycles) |
| 前置条件 | 建一个Member领域对象，并且选择修改薪水策略，输入岗位，更改薪水及结算周期 |
| 后置条件 | 调用Member方法领域对象的setWages方法 |
| MemberController. endMemberOpt | 语法 | Public void endMemberOpt() |
| 前置条件 | 建一个Member领域对象，并且已完成操作 |
| 后置条件 | 调用Member方法领域对象的endMemberOpt方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Member. find( stringID) | 查找一个人事信息 | |
| …… | …… | |

**表4.1.6-4 Member的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Member. find | 语法 | Public resultMessage find( stringID) |
| 前置条件 | 输入id |
| 后置条件 | 返回该员工/司机/车辆的详细信息（memberPO） |
| Member. addStaff | 语法 | Public Boolean addStaff( StaffVO vo) |
| 前置条件 | 输入员工基本信息 |
| 后置条件 | 系统新建一个员工 |
| Member. deleteStaff | 语法 | Public void deleteStaff( staffPO po) |
| 前置条件 | 选择删除员工 |
| 后置条件 | 系统删除该员工 |
| Member. changeStaff | 语法 | Public resultMessage changeStaff( staffVo vo) |
| 前置条件 | 选择更改员工信息，输入更改后的员工vo |
| 后置条件 | 系统对该员工的信息进行更改 |
| Member. addCar | 语法 | Public boolean addCar(carVO vo) |
| 前置条件 | 选择增加车辆信息，输入车辆基本信息 |
| 后置条件 | 系统新建一个车辆信息 |
| Member. deleteCar | 语法 | Public void deleteCar(CarVO vo) |
| 前置条件 | 选择删除车辆信息 |
| 后置条件 | 系统删除该车辆信息 |
| Member. changeCar | 语法 | Public resultMessage changeCar( carVO vo) |
| 前置条件 | 选择修改车辆信息 |
| 后置条件 | 系统修改该车辆信息 |
| Member. addDriver | 语法 | Public Boolean addDriver( DriverVO vo) |
| 前置条件 | 输入司机基本信息 |
| 后置条件 | 系统新建一个司机 |
| Member. deleteDriver | 语法 | Public void deleteDriver(DriverPO po) |
| 前置条件 | 选择删除司机信息 |
| 后置条件 | 系统删除该司机信息 |
| Member. changeDriver | 语法 | Public resultMessage changeDriver(DriverVo vo) |
| 前置条件 | 选择修改司机信息 |
| 后置条件 | 系统修改该司机信息 |
| Member. setWages | 语法 | Public resultMessage setWages( string job, int wages, int cycles) |
| 前置条件 | 选择修改薪水策略，输入岗位，更改薪水及结算周期 |
| 后置条件 | 系统修改该岗位的薪水策略 |
| Member. endMemberOpt | 语法 | Public void endMemberOpt() |
| 前置条件 | 已完成操作 |
| 后置条件 | 系统持久化更新涉及领域的对象信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFactory.getMemberDatabase | 获得member数据库的引用 | |
| memberDataService. Find( string id) | 根据输入的id在数据库中查找MemberPO对象 | |
| MemberDataService. Insert(MemberPO po) | 在数据库中插入MemberPO对象 | |
| MemberDataService. Delete( MemberPO po) | 在数据库中删除MemberPO对象 | |
| MemberDataService. Update(MemberPO po) | 在数据库中更新MemberPO对象 | |

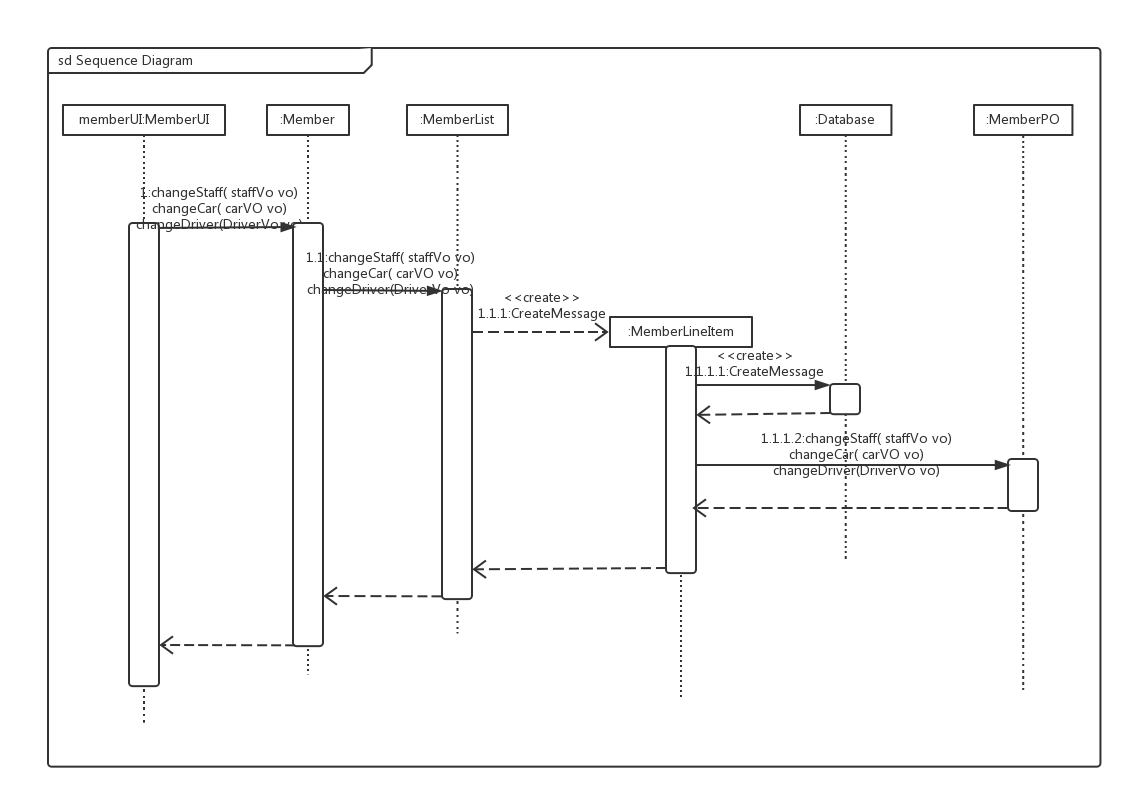
（4）业务逻辑层的动态模型

图4.1.6-5.1表明了物流管理系统中，当用户输入新增的人事信息之后，人事业务逻辑处理的相关对象之间的协作。



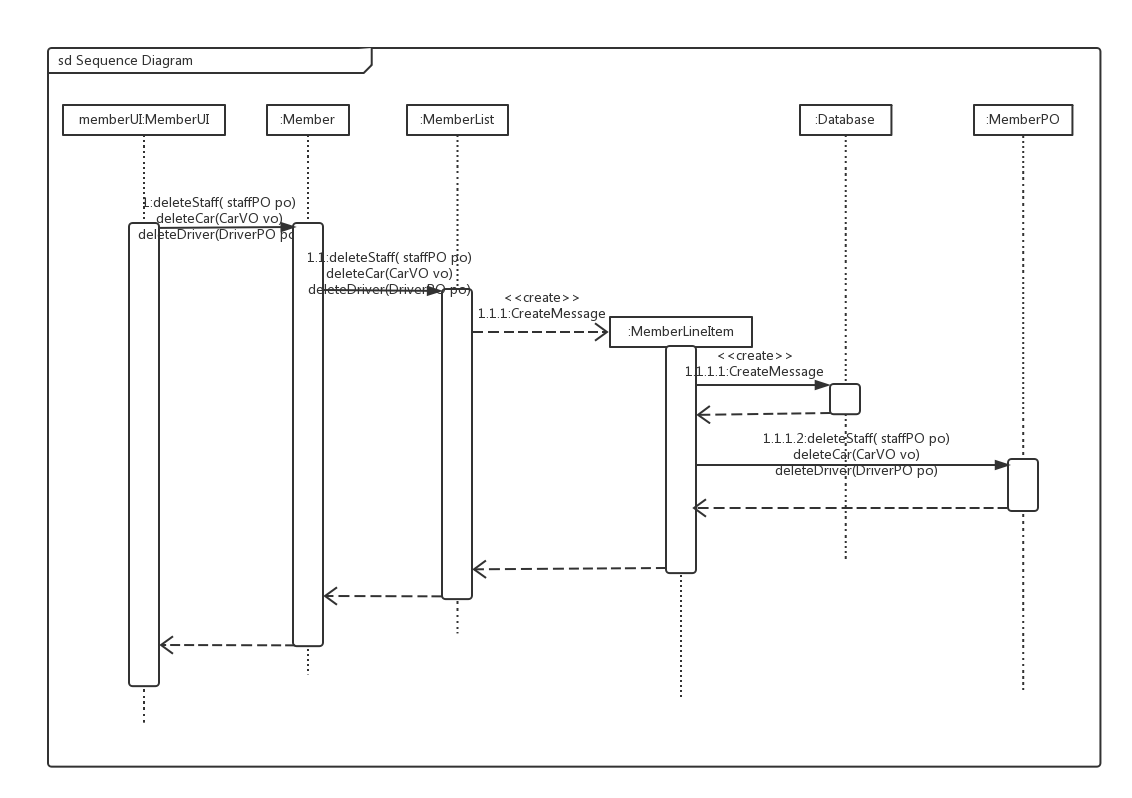
**图4.1.6-5.1 输入新增信息时的顺序图**

图4.1.6-5.2表明了物流管理系统中，当用户输入修改的人事信息之后，人事业务逻辑处理的相关对象之间的协作。



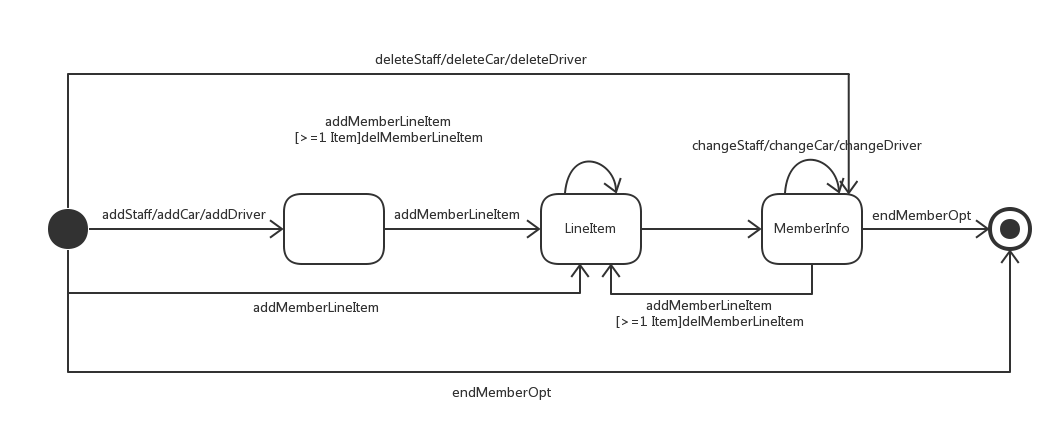
**图4.1.6-5.2 输入修改信息时的顺序图**

图4.1.6-5.3表明了物流管理系统中，当用户删除人事信息之后，人事业务逻辑处理的相关对象之间的协作。



**图4.1.6-5.3 删除信息时的顺序图**

如图4.1.6-6所示的状态图描述了Member对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



**图4.1.6-6 Member对象状态图**

（5）业务逻辑层的设计原理

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

### 4.1.7 Logbl模块

(1)模块概述

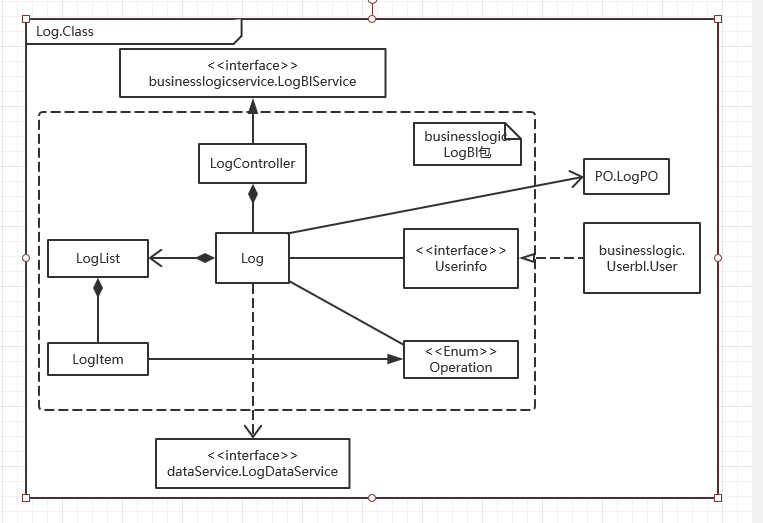
logbl模块承担的需求参见需求规格说明文档的功能需求及相关的非功能需求

Logbl模块的职责及接口参见软件系统结构描述文档

1. 整体结构

根据体系结构设计，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，我们会增加接口，比如businesslogicservice.LogBlService 、dataservice.LogDateService。为了隔离业务逻辑职责和逻辑控制职责，我们增加了LogController，这样LogController会将对系统日志逻辑的处理委托给Log对象。LogPO是作为系统日志记录的持久化对象被添加到设计模型中的。LogList封装了关于LogItem的数据集合的数据结构的秘密和显示当前库存的职责。UserInfo是根据依赖导致原则，为了消除循环依赖产生的接口

Log模块如下：



Log模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面需要的服务 |
| LogController | 负责实现对应于系统日志界面需要的服务 |
| Log | 系统日志的领域模型对象，拥有一次对系统日志操作所持有的信息，可以帮助系统日志界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户的信息，可以解决登陆问题 |

(3) 模块的内部类的接口规范

#### LogblController模块的接口规范

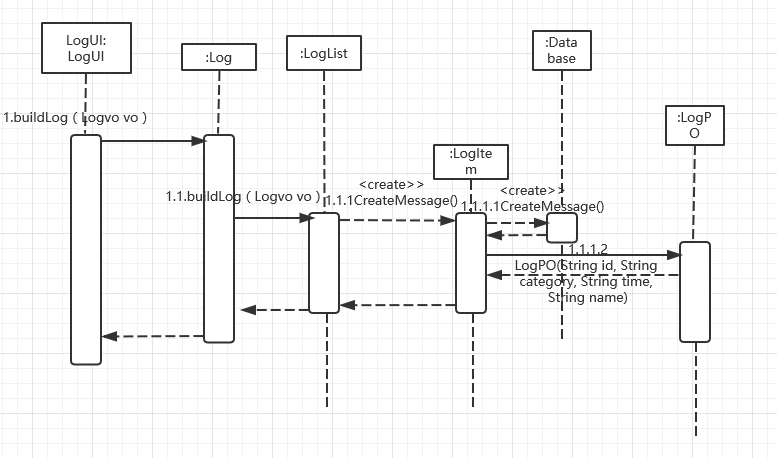
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| LogblController.buildLog | 语法 | public boolean buildLog(Logvo vo) |
| 前置条件 | 已创建一个Log领域对象，并输入合理的数据 |
| 后置条件 | 调用Log领域的buildLog方法 |
| LogblController.inquiry | 语法 | Public LogVO inquiry(String id) |
| 前置条件 | 已创建一个Log领域对象，并输入合理的数据 |
| 后置条件 | 调用Log领域的inquiry方法 |
| LogblController.inquiryAll | 语法 | Public ArrayList<LogVO> inquiryAll() |
| 前置条件 | 已创建一个Log领域对象 |
| 后置条件 | 调用Log领域的inquiryAll方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Log.buildLog | 创建新的系统日志 | |
| Log.inquiry | 查询单条记录 | |
| Log.inquiryAll | 查询所有系统日志 | |

#### Logbl模块的接口规范

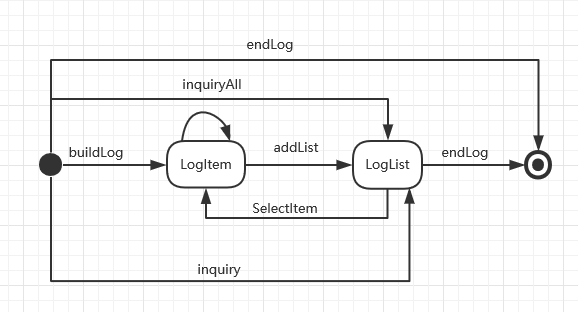
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Log.buildLog | 语法 | public boolean buildLog(Logvo vo) |
| 前置条件 | 有需要记录的系统操作发生 |
| 后置条件 | 生成本次操作的系统日志 |
| Log.inquiry | 语法 | Public LogVO inquiry(String id) |
| 前置条件 | 输入某次操作编号 |
| 后置条件 | 显示该次操作详情 |
| Log.inquiryAll | 语法 | Public ArrayList<LogVO> inquiryAll() |
| 前置条件 | 选择查看所有系统日志 |
| 后置条件 | 显示该账户所有系统日志 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFactory.getLogDatabase | 获得Log数据库的引用 | |
| LogDataService. Find( string id) | 根据输入的id在数据库中查找Log对象 | |
| LogDataService. Insert(MemberPO po) | 在数据库中插入LogPO对象 | |
| MemberDataService. Update(MemberPO po) | 在数据库中更新MemberPO对象 | |

(4)业务逻辑层的动态模型

下图表明在EMLS中，当有有新的操作被记录时，系统日志逻辑处理的相关对象之间的协作。



下图表明Log对象在生存期间的状态序列和引起转移的事件，以及因状态转移而伴随的动作。



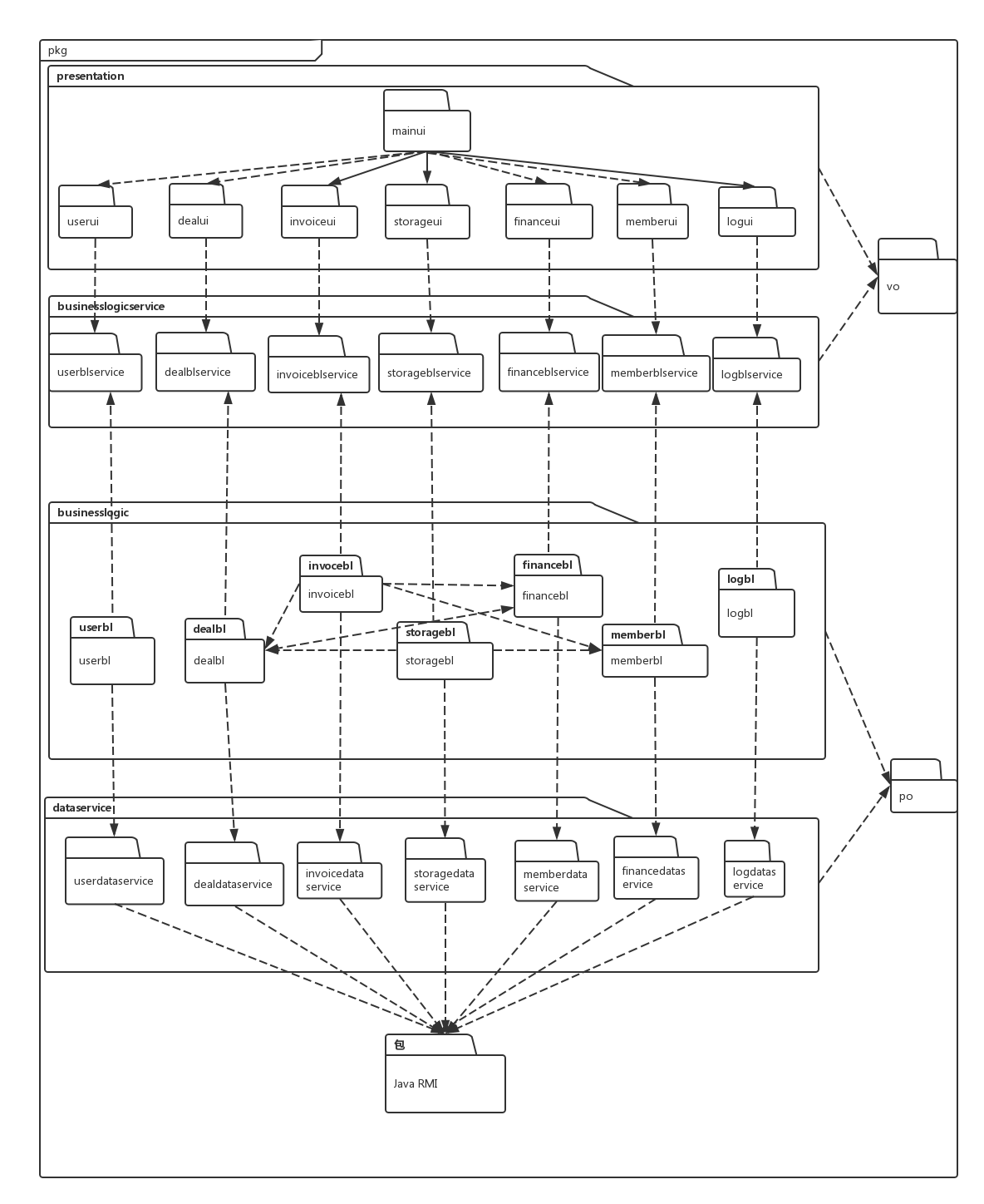
(5)业务逻辑层的设计原理

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。

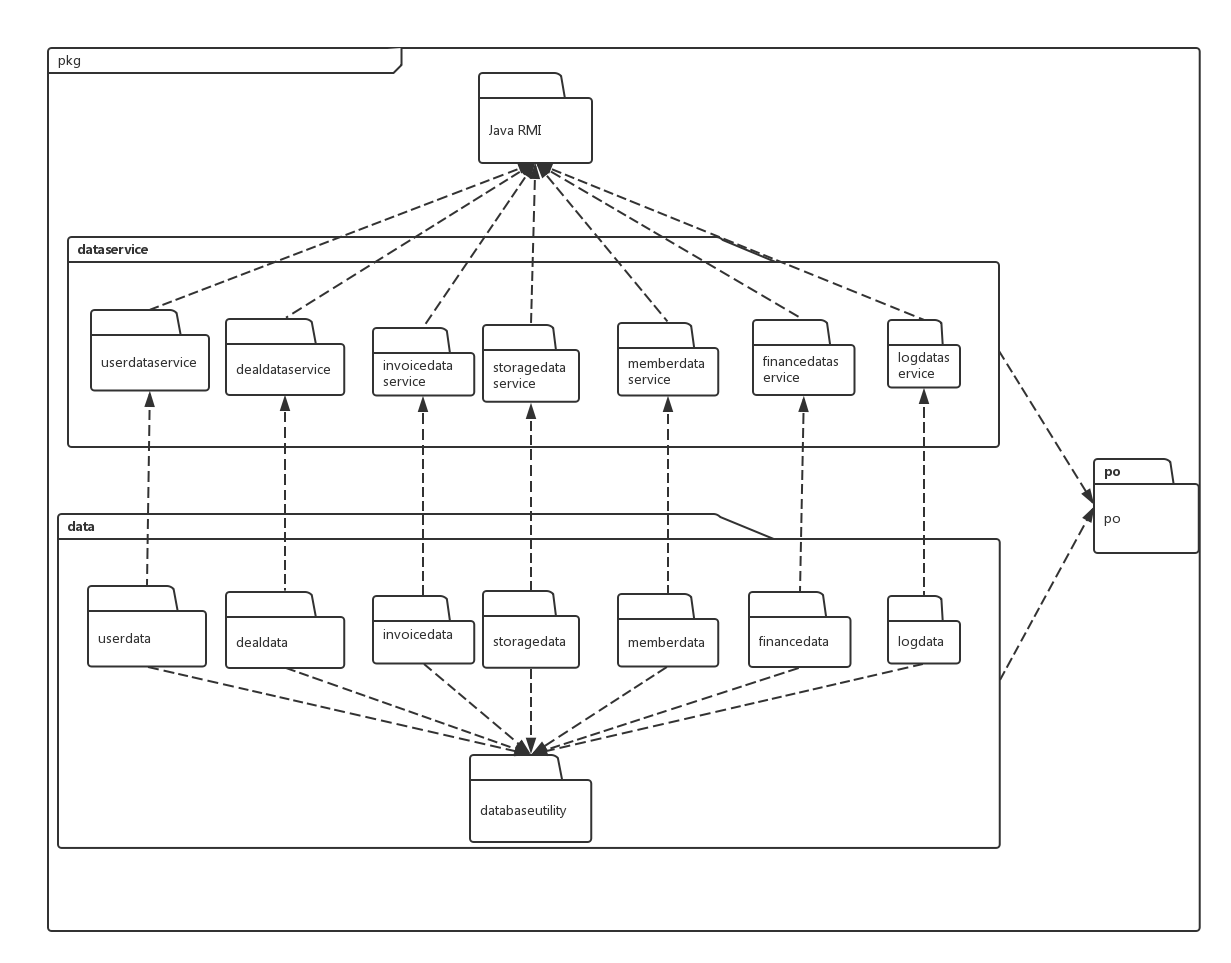
其他略。

# 5.依赖视角

图5-1和图5-2是客户端和服务器端各自的包之间的依赖关系。



**图5-1 客户端包图**

****

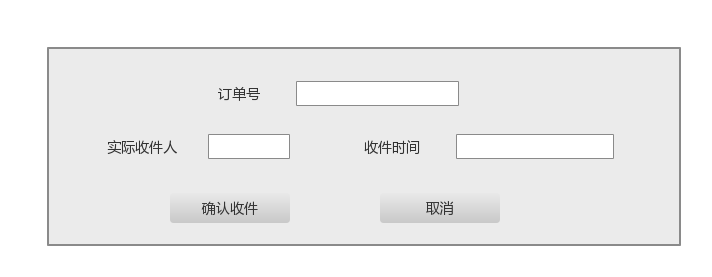
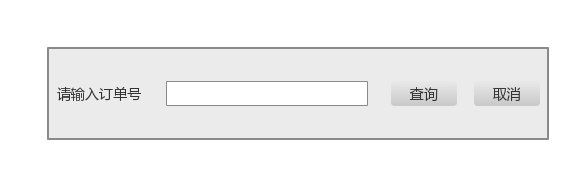
**图5-2 服务器端包图**

# 6.界面

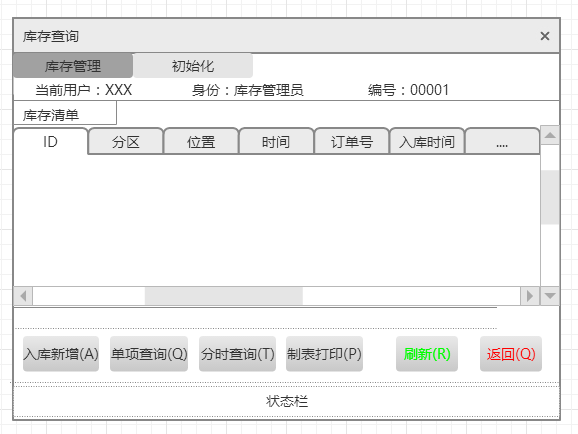
## 用户模块界面



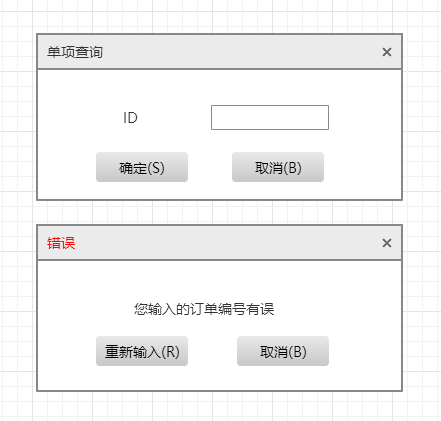
## 快递员及寄件人界面



## 库存界面













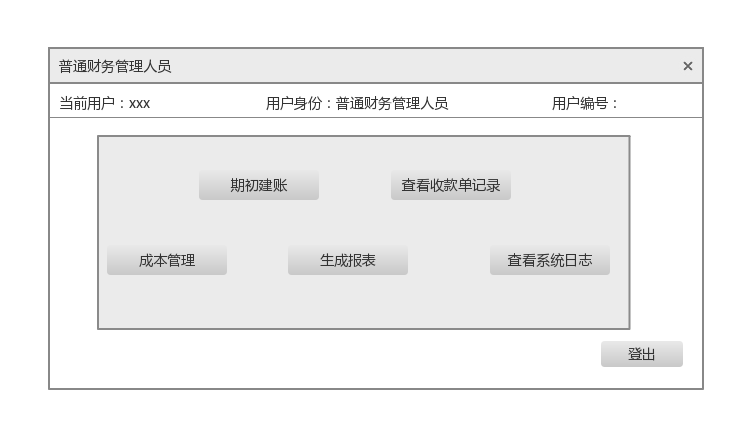


## 系统日志界面



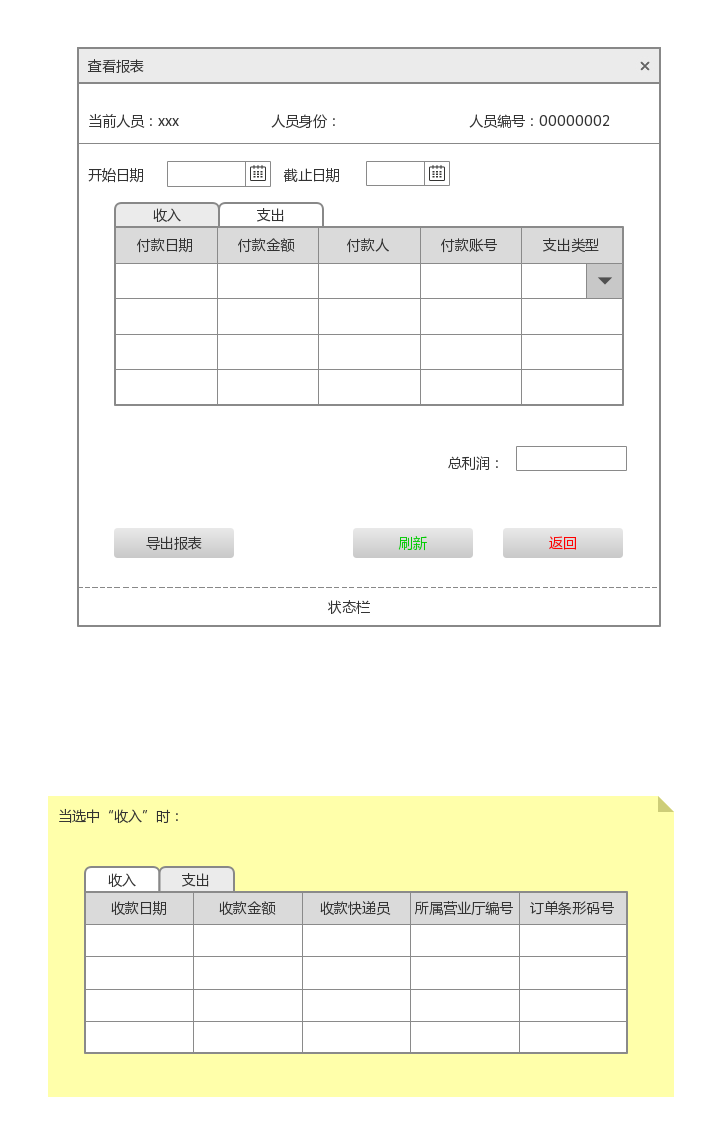


## 经济模块界面





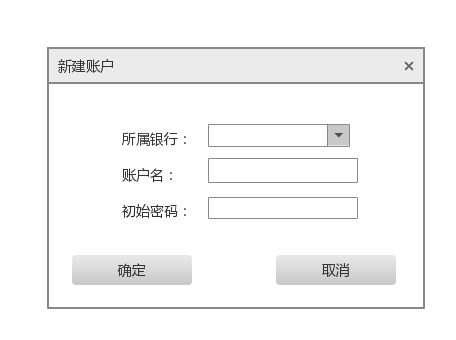












## 单据界面





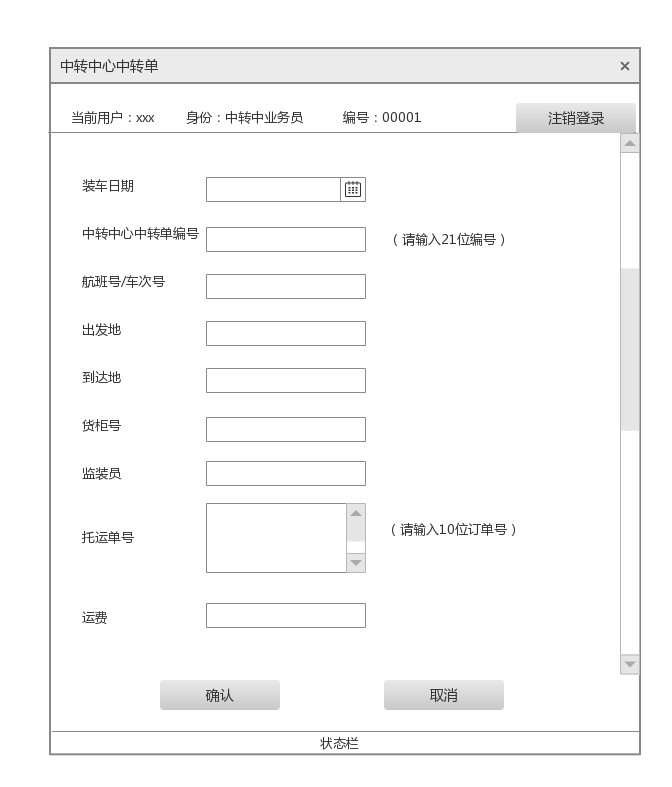














## 人事界面



