李凯 朴圣哲 狄逸枫 傅林华

物流快递系统

软件详细设计描述文档

物流快递系统软件详细设计描述文档

**1.引言**

**1.1 编制目的**

本报告详细完成对物流快递系统的详细设计,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 参考资料**

**2.产品概述**

参考物流快递系统用例文档和物流快递系统软件需求规格说明中对产品的概括描述。

**3.体系结构设计概述**

参考物流快递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3。

4.1.1 reviewbl模块

（1）模块描述

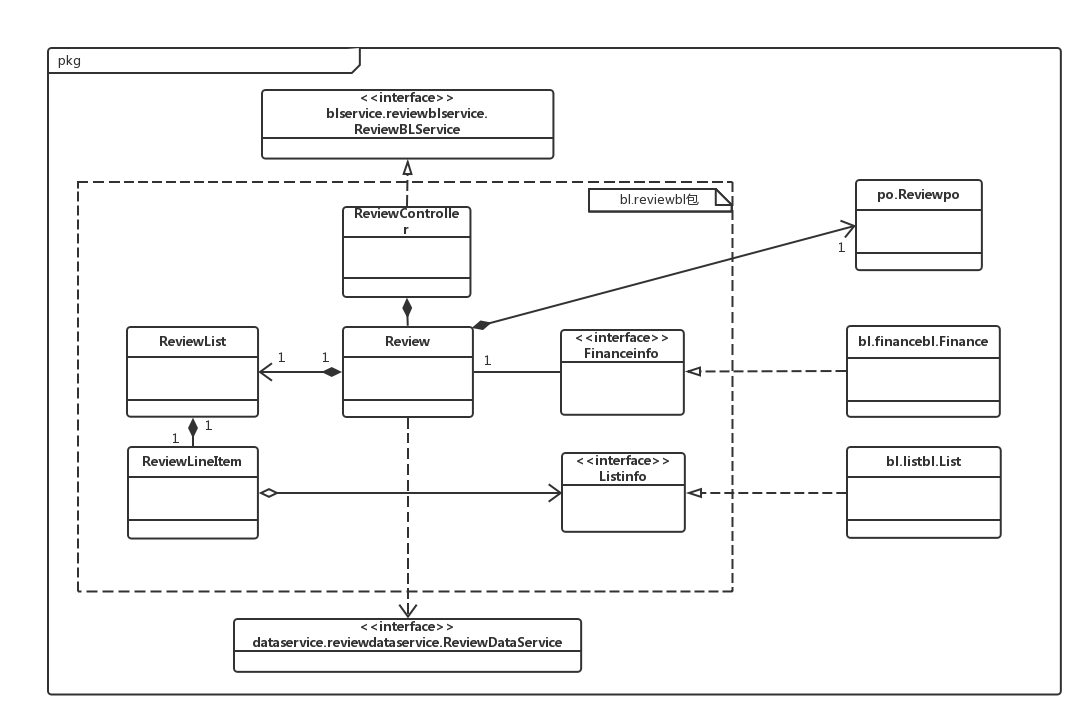
reviewbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

reviewbl模块的职责及接口参见软件系统结构描述文档表16。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice.reviewblservice. ReviewBLService接口。业务逻辑层和数据层之间添加 dataservice.reviewdataservice. ReviewDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 ReviewController,这样ReviewController会将对审查的业务逻辑处理委托给Review对象。ReviewPO是作为审查记录的持久化对象被添加到设计模型中去的。而ReviewList和ReviewLineItem的添加是List的容器类。ReviewLineItem保有职员、机构和的审查单据的数据,及相应的制定薪水的职责。而 ReviewList封装了关于 ReviewLineItem的数据集合的数据结构的秘密和制定薪水的职责。FinanceInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

reviewbl模块的设计如图7所示。

图7 reviewbl模块各个类的设计

reviewbl模块各个类的职责如表7所示。

**表7 reviewbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现日志所需要的服务 |
| ReviewController | 负责实现审查界面所需要的服务 |
| Log | 日志的领域模型对象,可以解决日志问题。 |
| Review | 审查的领域模型对象,拥有一次审查所需的职员、机构、单据等信息,可以帮助完成审查界面所需要的服务。 |

（3）模块内部类的结构规范

ReviewController和Review的接口规范如表8和表9所示。

**表8 ReviewController的接口规范**

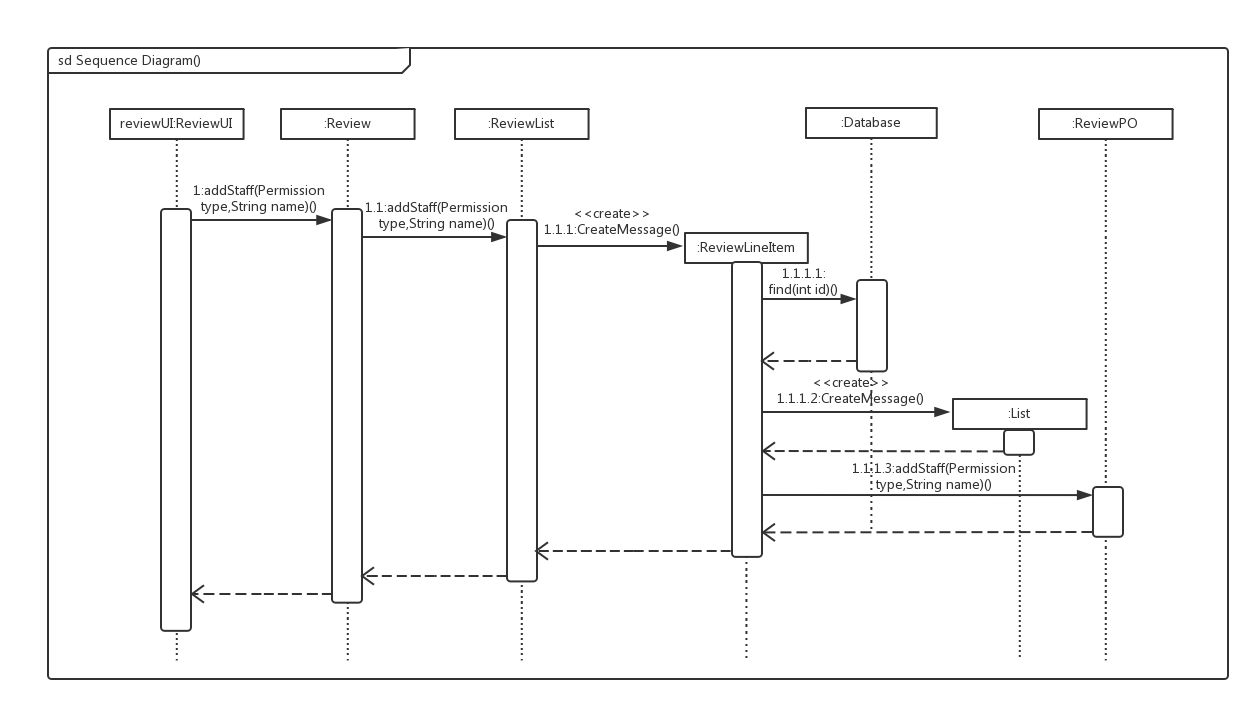
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ReviewController.addStaff | 语法 | Public ResultMessage addStaff(int type,String name) |
| 前置条件 | 已创建一个Review领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Review领域对象的addStaff方法 |
| ReviewController.removeStaff | 语法 | Public ResultMessage removeStaff(int type,String name) |
| 前置条件 | 已创建一个Review领域对象，存在职员 |
| 后置条件 | 调用Review领域对象的removeStaff方法 |
| ReviewController.addOrg | 语法 | Public ResultMessage addOrg(int type,String name) |
| 前置条件 | 已创建一个Review领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Review领域对象的addOrg方法 |
| ReviewController.removeOrg | 语法 | Public ResultMessage removeOrg(int type,String name) |
| 前置条件 | 已创建一个Review领域对象，存在机构 |
| 后置条件 | 调用Review领域对象的removeOrg方法 |
| ReviewController.setBAccount | 语法 | Public ResultMessage setBAccount(string num) |
| 前置条件 | 已创建一个Review领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Review领域对象的setBAccount方法 |
| ReviewController.setSalary | 语法 | Public ResultMessage setSalary(int type,double salary) |
| 前置条件 | 已创建一个Review领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Review领域对象的setSalary方法 |
| ReviewController.modList | 语法 | Public ResultMessage modList(ListPO list) |
| 前置条件 | 已创建一个Review领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Review领域对象的modList方法 |
| ReviewController.endReview | 语法 | Public ResultMessage endReview() |
| 前置条件 | 已创建一个Review领域对象 |
| 后置条件 | 调用Review领域对象的endReview方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Review.addStaff | 加入一个职员对象 | |
| Review.removeStaff | 删除一个职员对象 | |
| Review.addOrg | 加入一个机构对象 | |
| Review.removeOrg | 删除一个机构对象 | |
| Review.setBAccount | 设置一个账户 | |
| Review.setSalary | 设置薪水政策策略 | |
| Review.modList | 修改一张单据 | |
| Review.endReview | 结束审查 | |

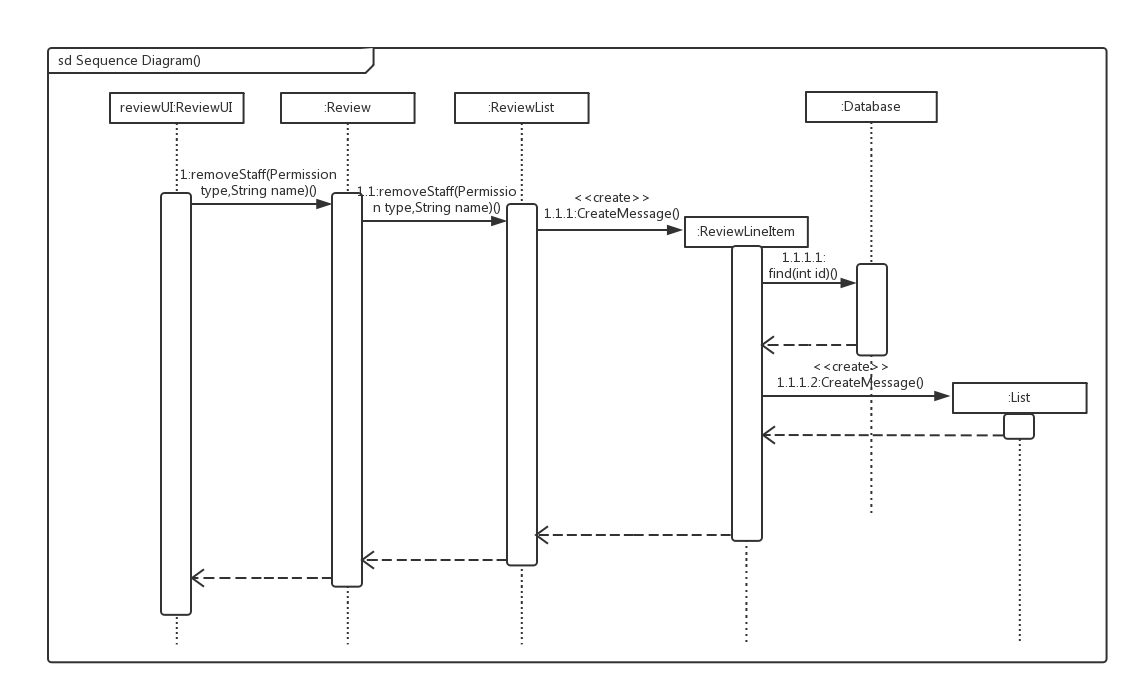
**表9 Review的接口规范**

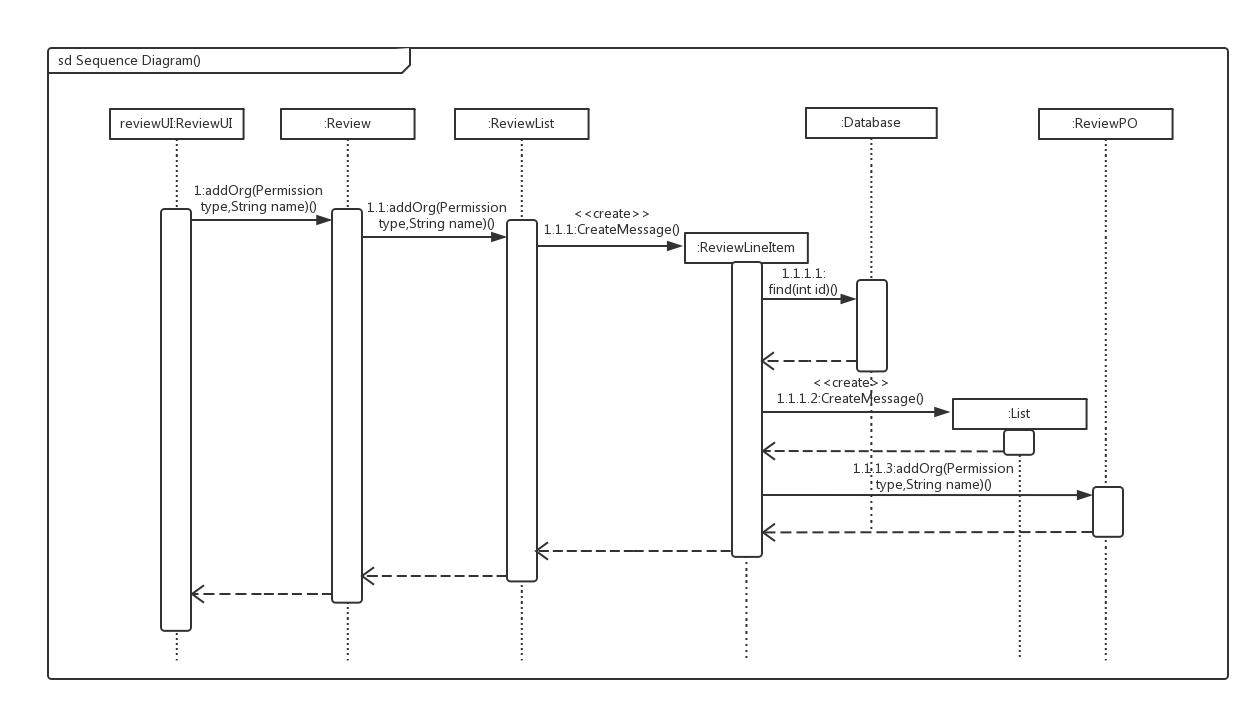
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Review.addStaff | 语法 | Public ResultMessage addStaff(int type,String name) |
| 前置条件 | 启动审查中的人员和机构管理 |
| 后置条件 | 在人员和机构管理中，增加人员信息 |
| Review.removeStaff | 语法 | Public ResultMessage removeStaff(int type,String name) |
| 前置条件 | 启动审查中的人员和机构管理 |
| 后置条件 | 在人员和机构管理中，增加机构信息 |
| Review.addOrg | 语法 | Public ResultMessage addOrg(int type,String name) |
| 前置条件 | 启动审查中的人员和机构管理 |
| 后置条件 | 在人员和机构管理中，增加机构信息 |
| Review.removeOrg | 语法 | Public ResultMessage removeOrg(int type,String name) |
| 前置条件 | 启动审查中的人员和机构管理 |
| 后置条件 | 在人员和机构管理中，删除机构信息 |
| Review.setBAccount | 语法 | Public ResultMessage setBAccount(string num) |
| 前置条件 | 启动审查中的查看账户 |
| 后置条件 | 在查看账户中，修改账户信息 |
| Review.setSalary | 语法 | Public ResultMessage setSalary(int type,double salary) |
| 前置条件 | 启动审查中的制定薪水策略 |
| 后置条件 | 在制定薪水策略中，制定薪水策略 |
| Review.modList | 语法 | Public ResultMessage modList(ListPO list) |
| 前置条件 | 启动审查中的审判单据 |
| 后置条件 | 在审判单据中，修改单据 |
| Review.endReview | 语法 | Public ResultMessage endReview() |
| 前置条件 | 已审查 |
| 后置条件 | 结束审查，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 | |
| ReviewDataService.insert(ReviewPO po) | 插入单一持久化对象 | |
| ReviewDataService.delete(ReviewPO po) | 删除单一持久化队形 | |
| ReviewDataService.update(ReviewPO po) | 更新单一持久化对象 | |
| ReviewDataService.modify(ReviewPO po) | 修改单一持久化对象 | |
| ReviewDataService.updateList (ListPO po) | 更新单一持久化List对象 | |
| ReviewDataService.find(int id) | 根据ID进行查找单一持久化对象 | |
| ReviewDataService.finds(String field,int value) | 根据字段名和值进行查找多个持久化对象 | |
| DatabaseFactory.getListDatabase | 得到List数据库的服务的引用 | |

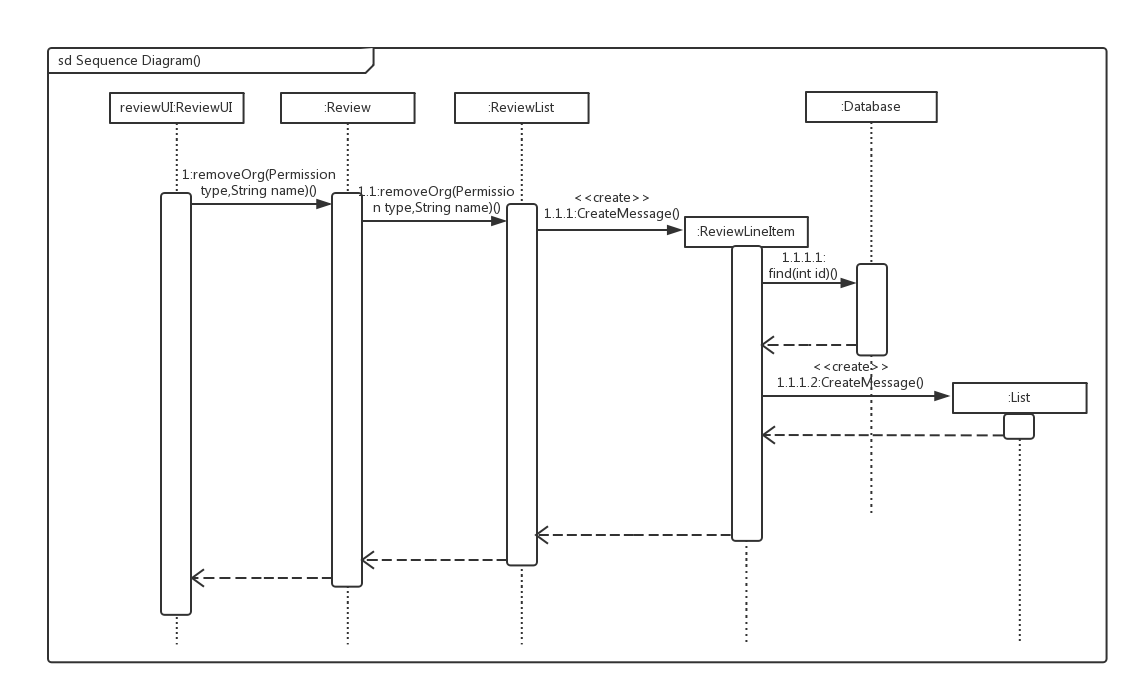
（4）业务逻辑层的动态模型

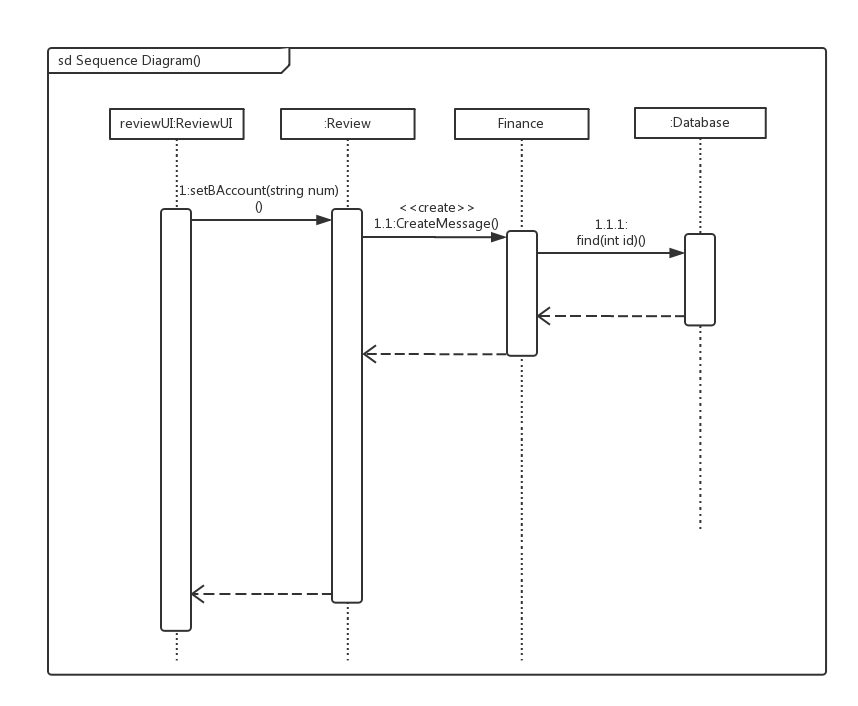
图8表明了物流快递系统中，当用户输入相关信息之后，审查业务逻辑处理的相关对象之间的协作。

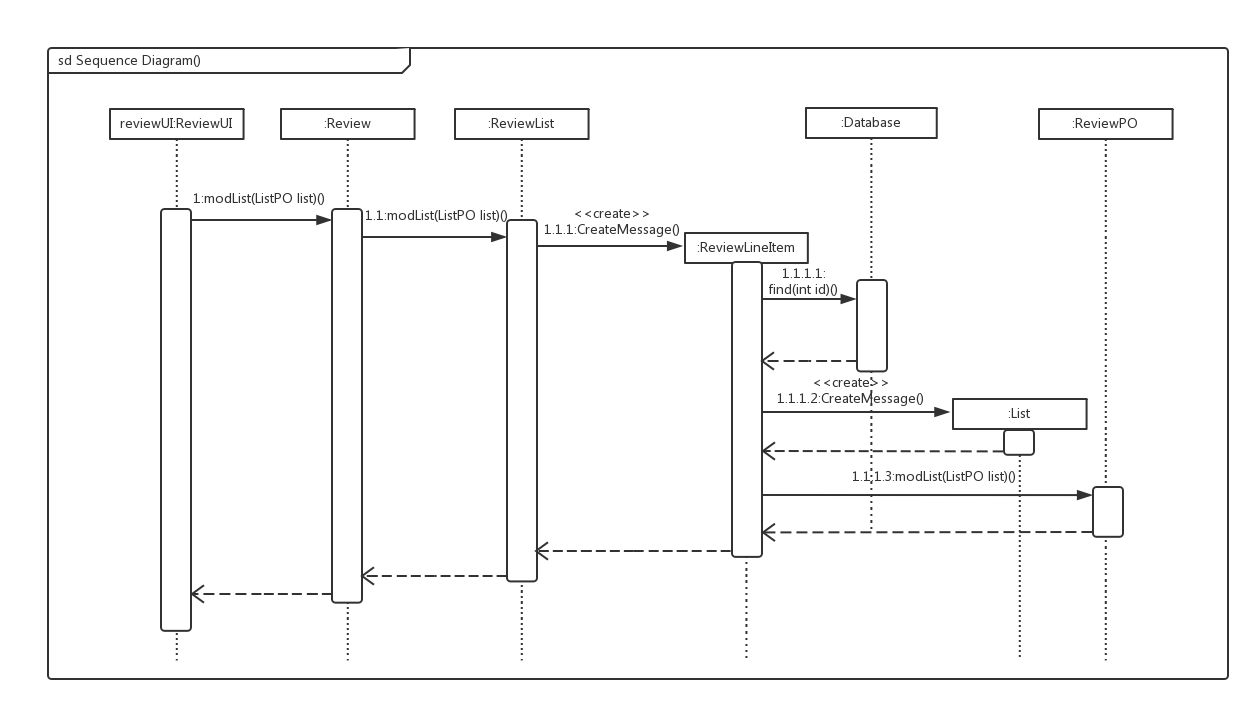
图8

添加职员的顺序图

删除职员的顺序图

添加机构的顺序图

删除机构的顺序图

设置账户的顺序图

修改单据的顺序图

如图9所示的状态图描述了Review对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着addStaff、addOrg、modList方法被UI调用，Review进入ReviewList状态；之后通过添加对象进入LineItem状态。UI也可以调用setSalary、setBAccount，使Review进入Finance状态。

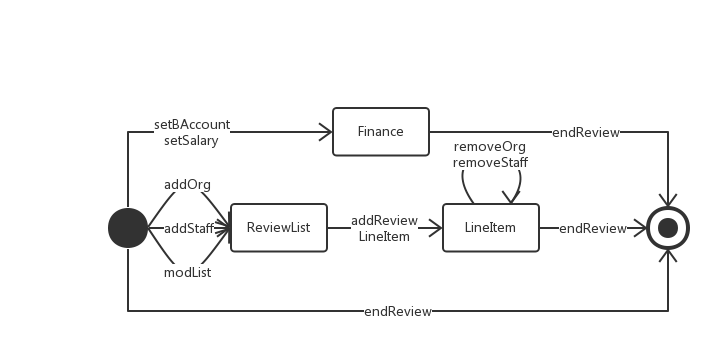


图9 Review对象状态图

（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。

其他略。

4.1.2 warehousebl模块

（1）模块描述

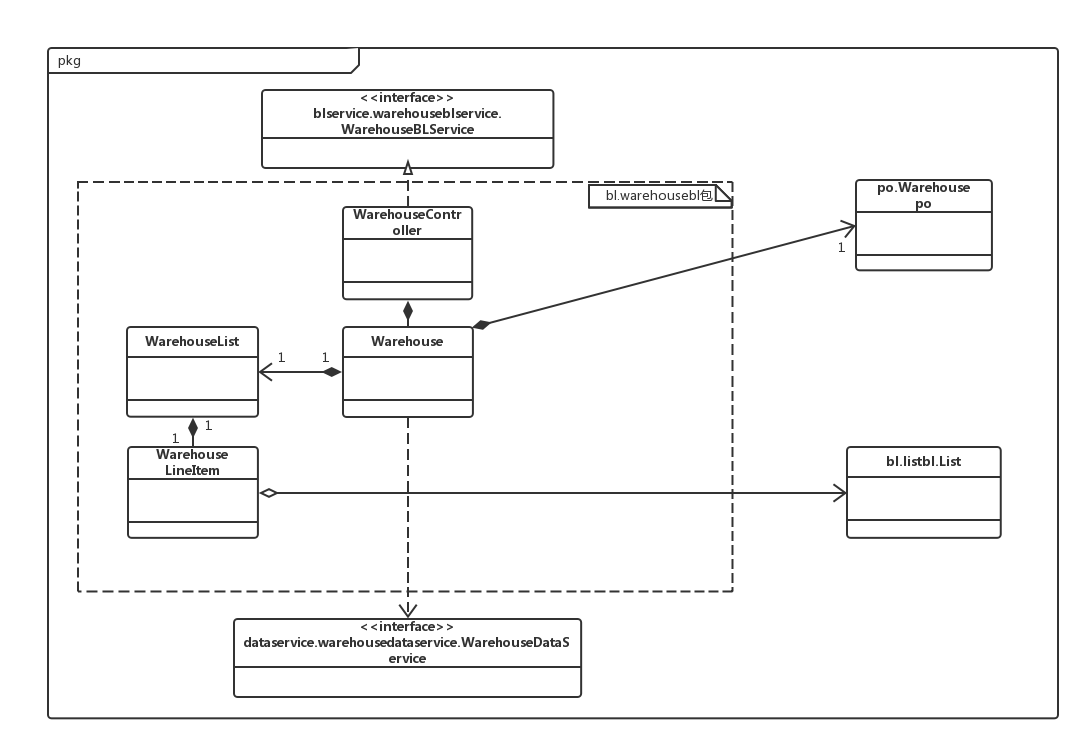
warehousebl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

warehousebl模块的职责及接口参见软件系统结构描述文档表15。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice.warehouseblservice.WarehouseBLService接口。业务逻辑层和数据层之间添加 dataservice.warehousedataservice.WarehouseDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 WarehouseController,这样WarehouseController会将对中转中心的业务逻辑处理委托给Warehouse对象。WarehousePO是作为审查记录的持久化对象被添加到设计模型中去的。而WarehouseList和WarehouseLineItem的添加是List的容器类。WarehouseLineItem保有货物的数据,及相应的出入库的职责。而 WarehouseList封装了关于 WarehouseLineItem的数据集合的数据结构的秘密和出入库的职责。

warehousebl模块的设计如图13所示。

图13 warehousebl模块各个类的设计

warehousebl模块各个类的职责如表13所示。

**表13 warehousebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| WarehouseController | 负责实现审查界面所需要的服务 |
| Warehouse | 审查的领域模型对象,拥有一次审查所需的职员、机构、单据等信息,可以帮助完成审查界面所需要的服务。 |

（3）模块内部类的结构规范

WarehouseController和Warehouse的接口规范如表14和表15所示。

**表14 WarehouseController的接口规范**

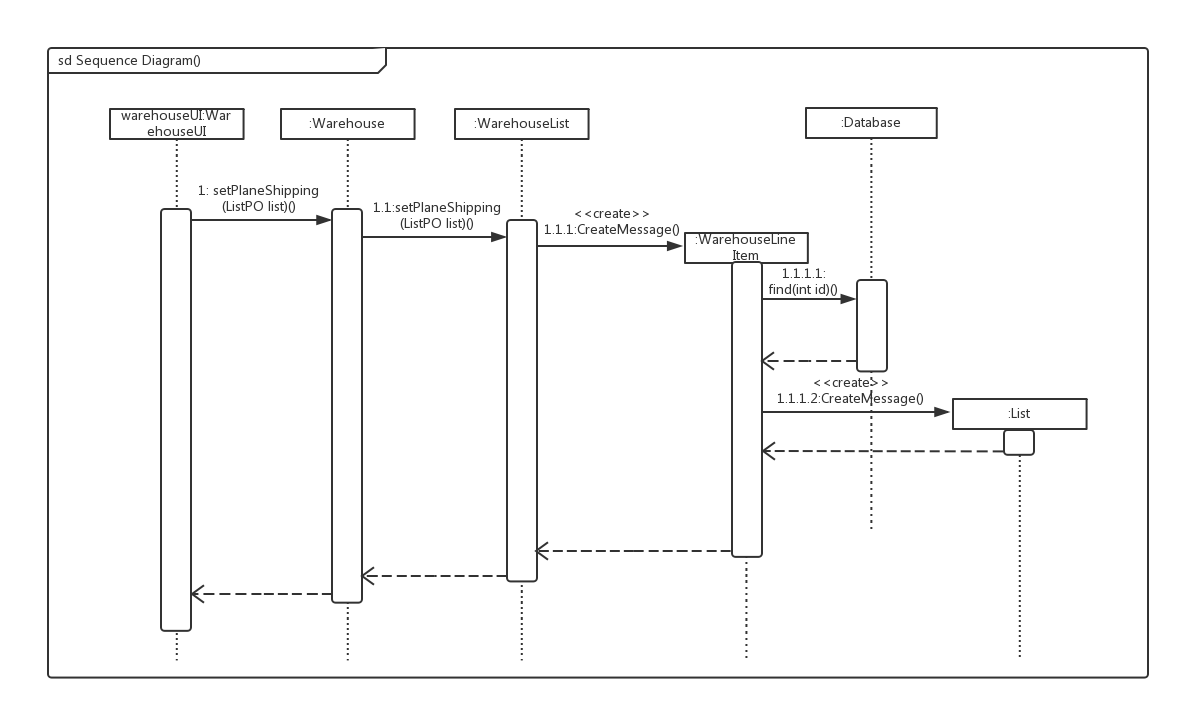
|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| WarehouseController.setPlaneShipping | 语法 | | Public ResultMessage setPlaneShipping (ListPO list) |
| 前置条件 | | 已创建一个Warehouse领域对象，并且符合飞机装运管理条件 |
| 后置条件 | | 调用Warehouse领域对象的setPlaneShipping方法 |
| WarehouseController.setTrainShipping | 语法 | | Public ResultMessage setTrainShipping (ListPO list) |
| 前置条件 | | 已创建一个Warehouse领域对象，并且符合汽车装运管理条件 |
| 后置条件 | | 调用Warehouse领域对象的setPlaneShipping方法 |
| WarehouseController.setCarShipping | 语法 | | Public ResultMessage setCarShipping (ListPO list) |
| 前置条件 | | 已创建一个Warehouse领域对象，并且符合火车装运管理条件 |
| 后置条件 | | 调用Warehouse领域对象的setCarShipping方法 |
| WarehouseController.geneCenterList | 语法 | | Public ListVO geneCenterList() |
| 前置条件 | | 已创建一个Warehouse领域对象，并且符合进行中转接收的条件 |
| 后置条件 | | 调用Warehouse领域对象的geneCenterList方法 |
| WarehouseController.wareIn | 语法 | | Public ResultMessage wareIn() |
| 前置条件 | | 已创建一个Warehouse领域对象，且快递到达，可以进行入库 |
| 后置条件 | | 调用Warehouse领域对象的wareIn方法 |
| WarehouseController.wareOut | 语法 | | Public ResultMessage wareOut() |
| 前置条件 | | 已创建一个Warehouse领域对象，且快递将要进行进行出库 |
| 后置条件 | | 调用Warehouse领域对象的wareOut方法 |
| WarehouseController.getWareInfo | 语法 | | Public WareHouseVO getWareInfo(int timeDiff) |
| 前置条件 | | 已创建一个Warehouse领域对象 |
| 后置条件 | | 调用Warehouse领域对象的getWareInfo方法 |
| WarehouseController.WareStockTake | 语法 | | Public ResultMessage WareStockTake() |
| 前置条件 | | 已创建一个Warehouse领域对象，且需要进行库存盘点 |
| 后置条件 | | 调用Warehouse领域对象的WareStockTake方法 |
| WarehouseController.WareAlarm | 语法 | | Public ResultMessage WareAlarm() |
| 前置条件 | | 已创建一个Warehouse领域对象，且商品的库存数量高于某数值 |
| 后置条件 | | 调用Warehouse领域对象的WareAlarm方法 |
| WarehouseController.ModWarePartition | 语法 | | Public ResultMessage ModWarePartition (int type, double storage) |
| 前置条件 | | 已创建一个Warehouse领域对象，且超出报警值 |
| 后置条件 | | 调用Warehouse领域对象的ModWarePartition方法 |
| WarehouseController.endWareHouse | 语法 | | Public void endWareHouse() |
| 前置条件 | | 已创建一个Warehouse领域对象，且中转中心任务已完成 |
| 后置条件 | | 调用Warehouse领域对象的endWareHouse方法 |
| **需要的服务（需接口）** | | | |
| 服务名 | | 服务 | |
| WareHouse.setPlaneShipping | | 进行飞机装运管理 | |
| WareHouse.setTrainShipping | | 进行火车装运管理 | |
| WareHouse.setCarShipping | | 进行汽车装运管理 | |
| WareHouse.geneCenterList | | 进行中转接收并生成中转中心接收单 | |
| WareHouse.wareIn | | 进行入库 | |
| WareHouse.wareOut | | 进行出库 | |
| WareHouse.getWareInfo | | 获取库存信息 | |
| WareHouse.WareStockTake | | 进行库存盘点 | |
| WareHouse.WareAlarm | | 进行库存报警提示 | |
| WareHouse.ModWarePartition | | 调整库存分区 | |
| WareHouse.endWareHouse | | 结束中转中心任务 | |

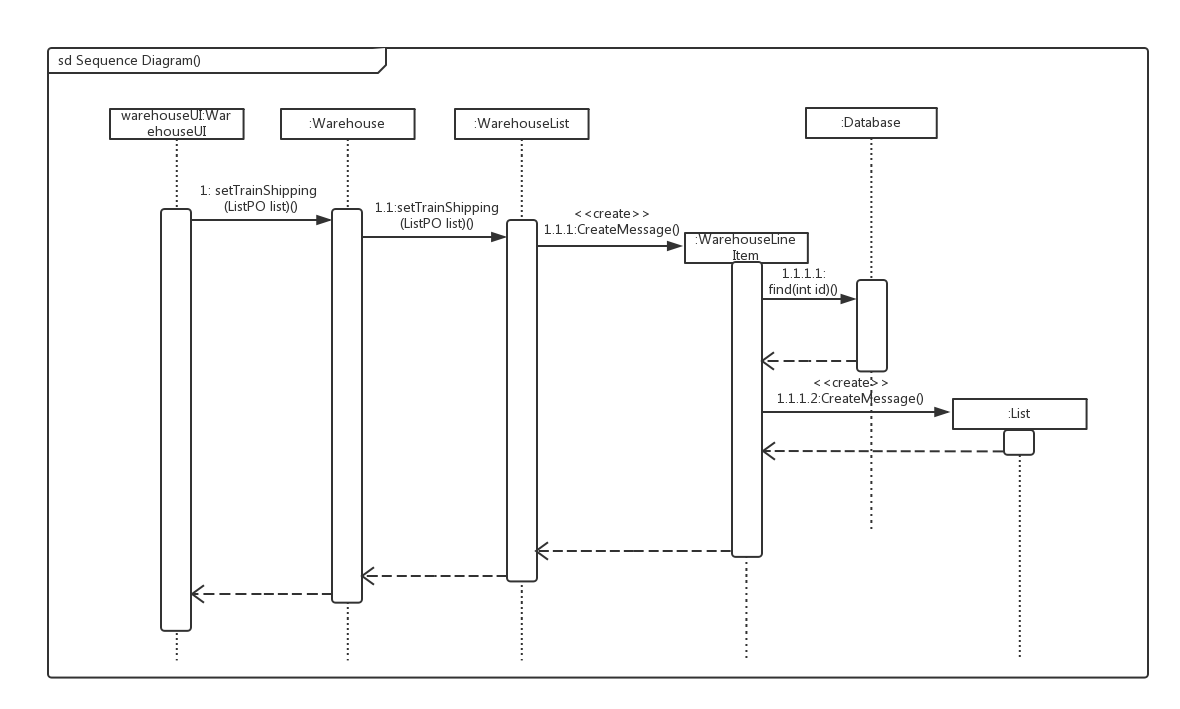
**表15 Warehouse的接口规范**

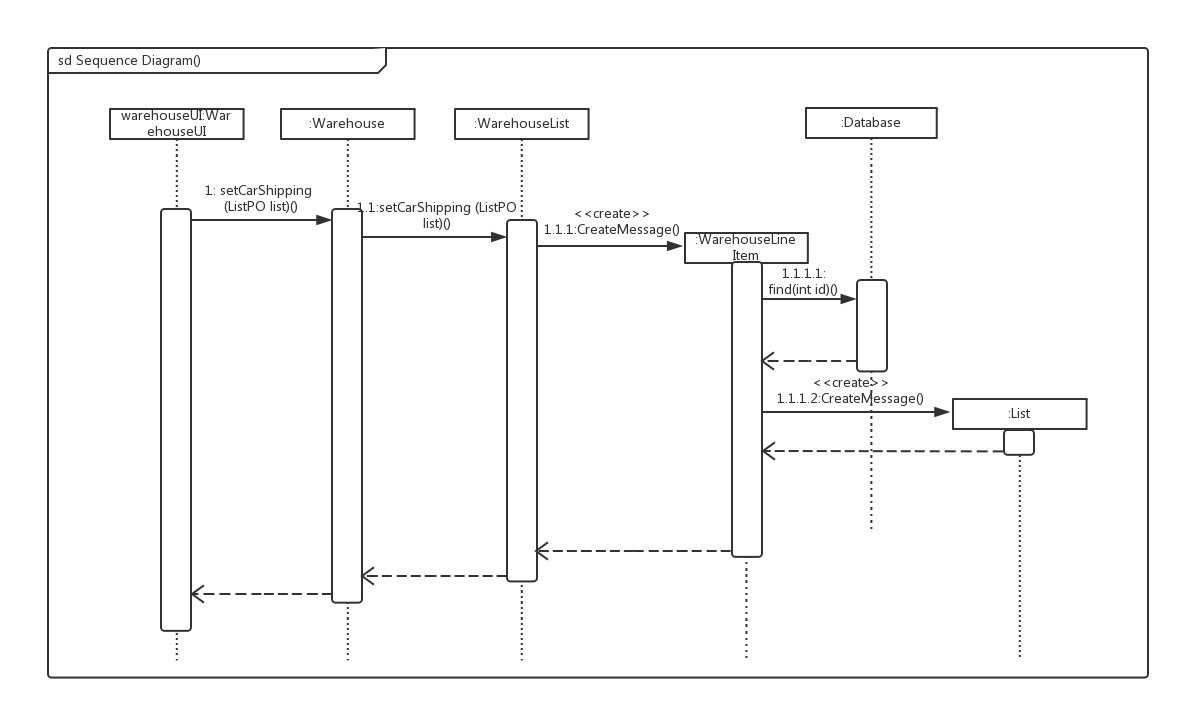
|  |  |  |  |
| --- | --- | --- | --- |
| **提供的服务（供接口）** | | | |
| WareHouse.setPlaneShipping | 语法 | | Public ResultMessage setPlaneShipping (ListPO list) |
| 前置条件 | | 启动中转中的飞机装运管理 |
| 后置条件 | | 在飞机装运管理中进行参数设置 |
| WareHouse.setTrainShipping | 语法 | | Public ResultMessage setTrainShipping (ListPO list) |
| 前置条件 | | 启动中转中的火车装运管理 |
| 后置条件 | | 在火车装运管理中进行参数设置 |
| WareHouse.setCarShipping | 语法 | | Public ResultMessage setCarShipping (ListPO list) |
| 前置条件 | | 启动中转中的汽车装运管理 |
| 后置条件 | | 在汽车装运管理中进行参数设置 |
| WareHouse.geneCenterList | 语法 | | Public ListVO geneCenterList() |
| 前置条件 | | 启动中转中的中转接收 |
| 后置条件 | | 生成中转中心接受单 |
| WareHouse.wareIn | 语法 | | Public ResultMessage wareIn() |
| 前置条件 | | 快递到达，进行入库 |
| 后置条件 | | 生成入库单 |
| WareHouse.wareOut | 语法 | | Public ResultMessage wareOut() |
| 前置条件 | | 快递将要进行进行出库 |
| 后置条件 | | 生成出库单 |
| WareHouse.getWareInfo | 语法 | | Public WareHouseVO getWareInfo(int timeDiff) |
| 前置条件 | | 需要得到库存信息 |
| 后置条件 | | 获取到库存信息 |
| WareHouse.WareStockTake | 语法 | | Public ResultMessage WareStockTake() |
| 前置条件 | | 需要进行库存盘点 |
| 后置条件 | | 完成盘点 |
| WareHouse.WareAlarm | 语法 | | Public ResultMessage WareAlarm() |
| 前置条件 | | 商品的库存数量高于某数值 |
| 后置条件 | | 进行库存报警提示 |
| WareHouse.ModWarePartition | 语法 | | Public ResultMessage ModWarePartition (int type, double storage) |
| 前置条件 | | 超出报警值 |
| 后置条件 | | 分区得到调整 |
| WareHouse.endWareHouse | 语法 | | Public void endWareHouse() |
| 前置条件 | | 中转中心任务已完成 |
| 后置条件 | | 结束中转中心任务，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 | |
| WareHouseDataService.insert(WareHousePO po) | | 插入单一持久化对象 | |
| WareHouseDataService.delete(WareHousePO po) | | 删除单一持久化队形 | |
| WareHouseDataService.update(WareHousePO po) | | 更新单一持久化对象 | |
| WareHouseDataService.updateList(ListPO po) | | 更新单一持久化List对象 | |
| WareHouseDataService.modify(WareHousePO po) | | 修改单一持久化对象 | |
| WareHouseDataService.find(int id) | | 根据ID进行查找单一持久化对象 | |
| WareHouseDataService.finds(String field,int value) | | 根据字段名和值进行查找多个持久化对象 | |
| DatabaseFactory.getListDatabase | | 得到List数据库的服务的引用 | |

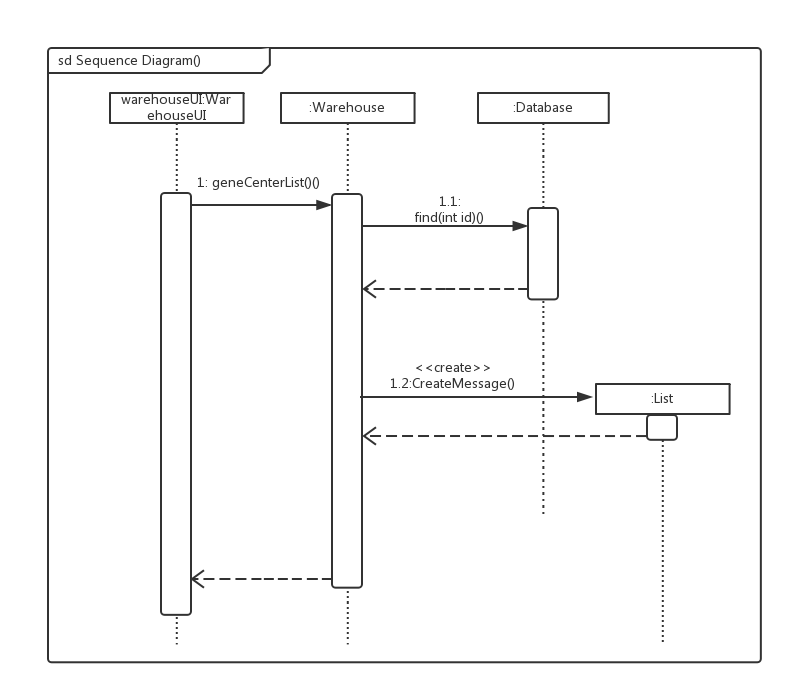
（4）业务逻辑层的动态模型

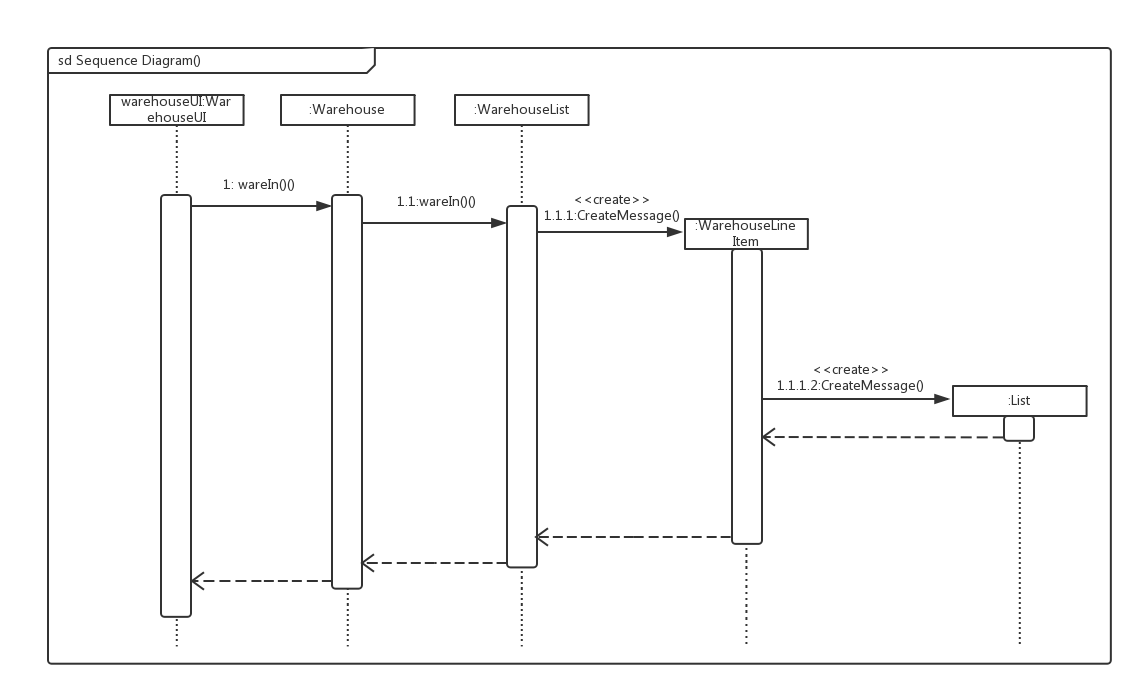
图14表明了物流快递系统中，当用户输入相关信息之后，中转中心业务逻辑处理的相关对象之间的协作。

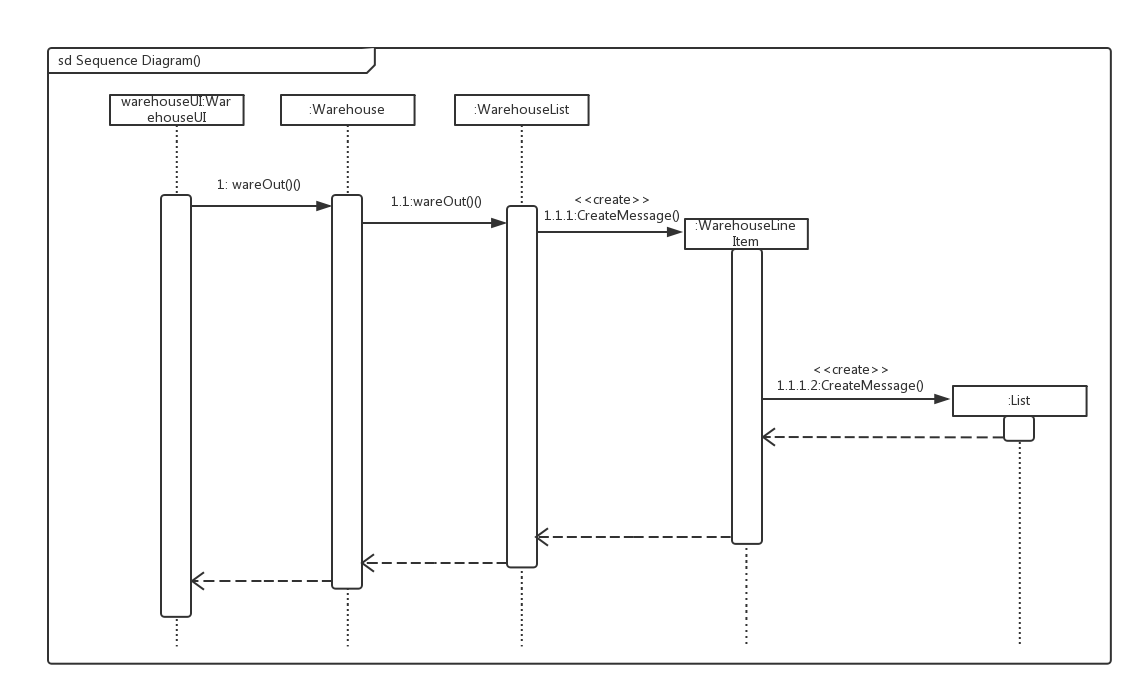
图14 

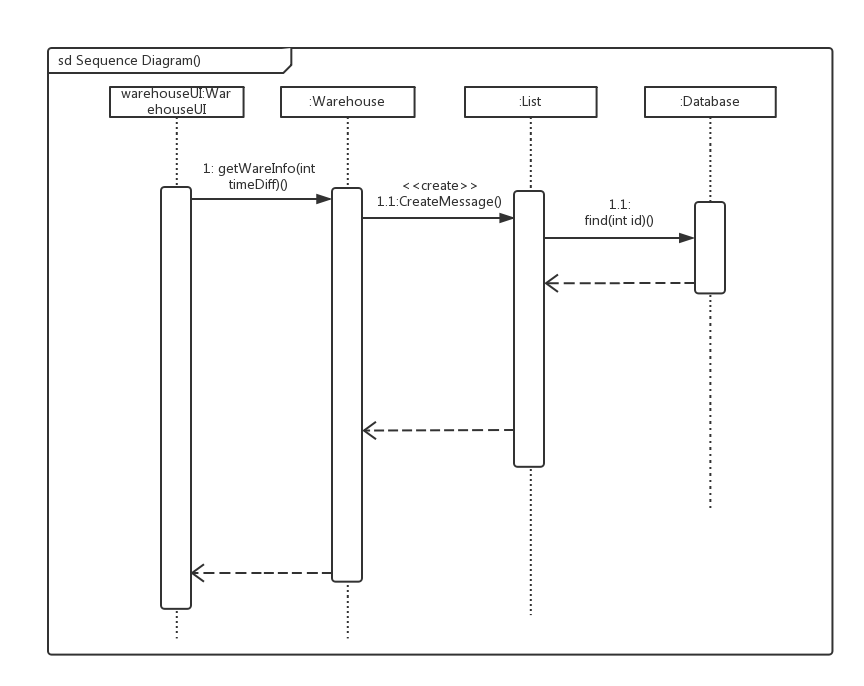
飞机装运管理的顺序图

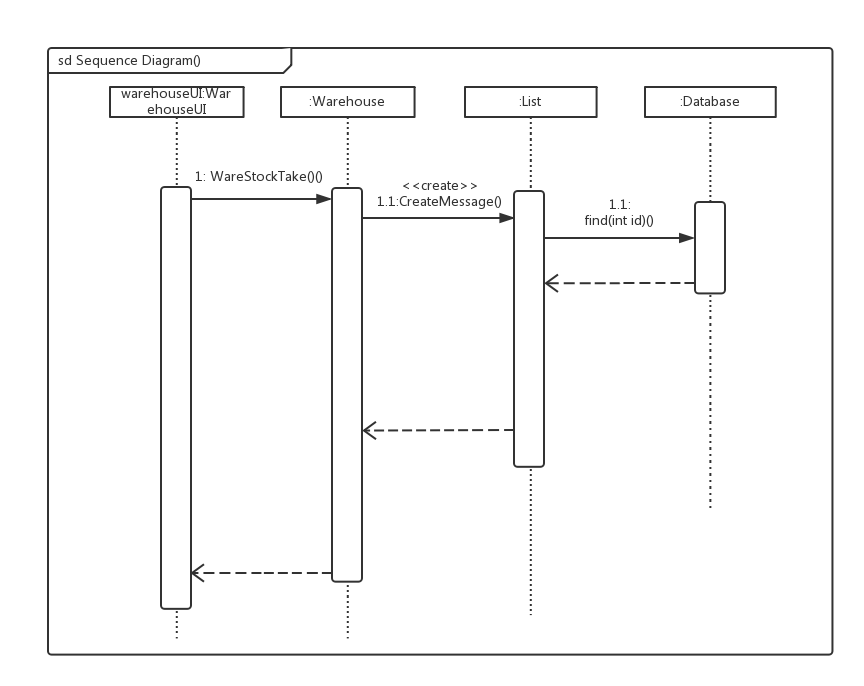
火车装运管理的顺序图

汽车装运管理的顺序图

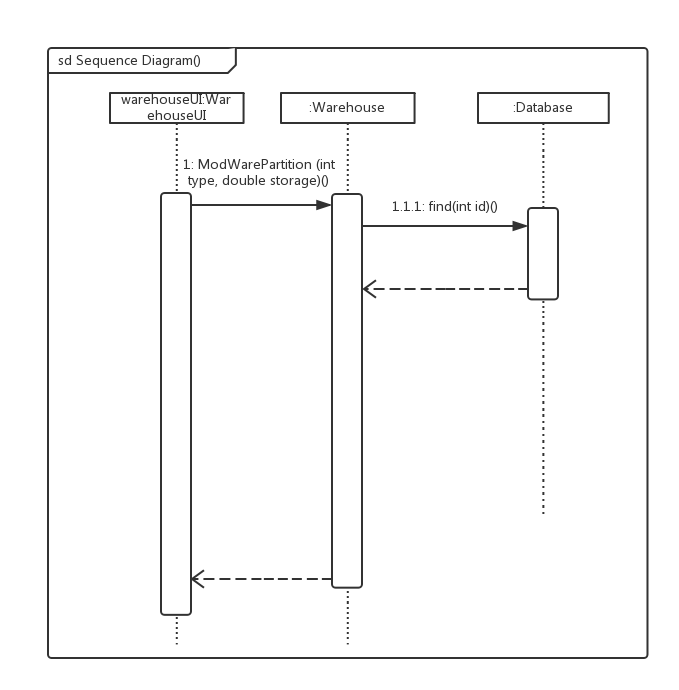
中转接收的顺序图

入库的顺序图

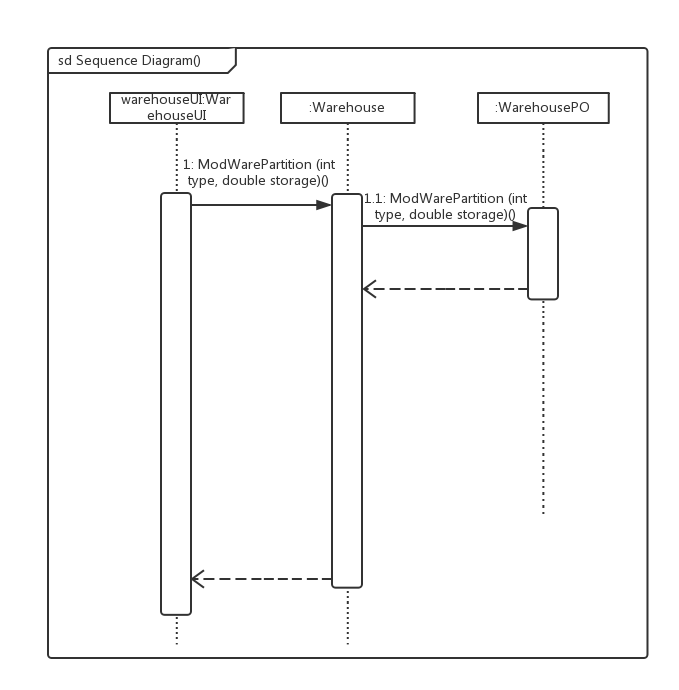
出库的顺序图

获取库存信息的顺序图

库存盘点的顺序图



库存报警的顺序图



分区调整的顺序图

如图15所示的状态图描述了Warehouse对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着setPlaneShipping、setTrainShipping、setCarShipping方法被UI调用，Warehouse进入WarehouseList状态；之后通过添加对象进入LineItem状态；最后进入WareInfo状态。UI也可以调用wareStockTake、wareAlarm、getWareInfo、modWarePartition，使Warehouse进入WareInfo状态。除此以外，UI还可以调用geneCenterList，使Warehouse进入List状态。

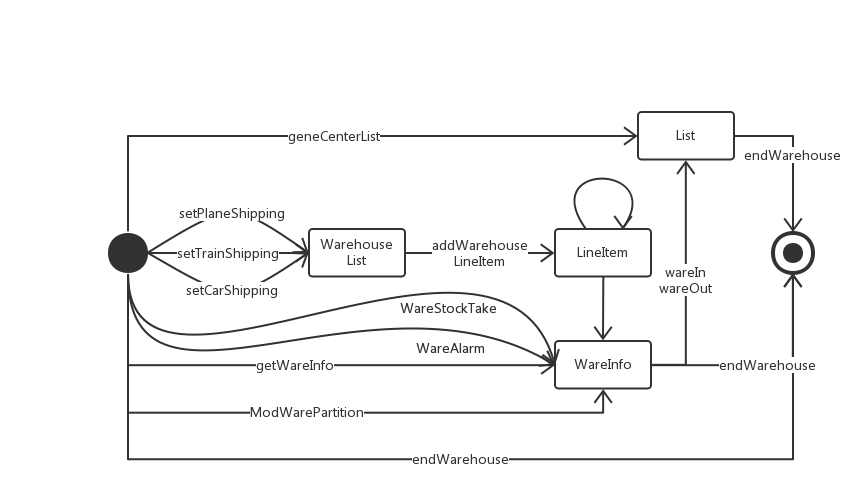


图15 Warehouse对象状态图

（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。

其他略。

4.1.3transbl模板

（1）模块描述

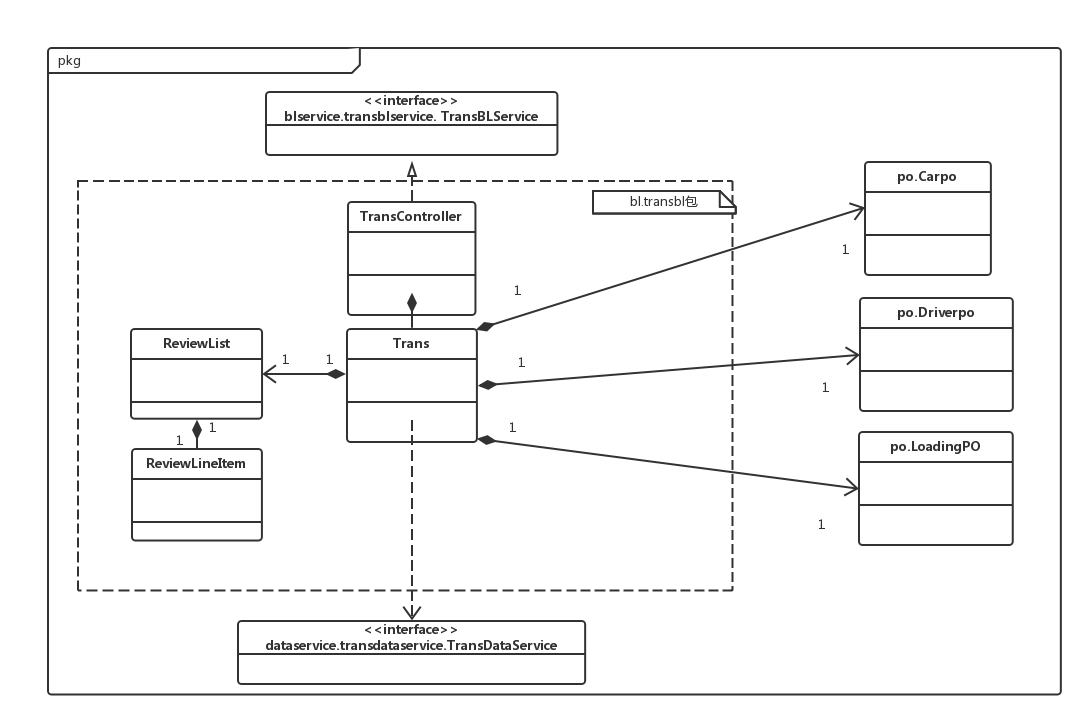
transbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

transbl模块的职责及接口参见软件系统结构描述文档表16。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice.transblservice. ReviewBLService接口。业务逻辑层和数据层之间添加 dataservice.reviewdataservice. transDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 TransController,这样TransController会将对运算的业务逻辑处理委托给Trans对象。CarPO,DriverPO,LoadingPO是作为运输任务的持久化对象被添加到设计模型中去的。而CarList和CarLineItem等的添加是List的容器类。CarLineItem,DriverLineItem,LoadingLineItem保有车辆，司机和装运的数据。而 对应的List封装了关于 对应的LineItem的数据集合的数据结构的秘密.

transbl模块的设计如图所示。



transbl模块各个类的职责如表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现日志所需要的服务 |
| TransController | 负责实现运输界面所需要的服务 |
| Log | 日志的领域模型对象,可以解决日志问题。 |
| Trans | 运输的领域模型对象,拥有一次管理所需的车辆、司机、装运等信息,也可以帮助完成订单界面所需要的服务。 |

（3）模块内部类的结构规范

TransController和Trans的接口规范如表所示。

**TransController的接口规范**

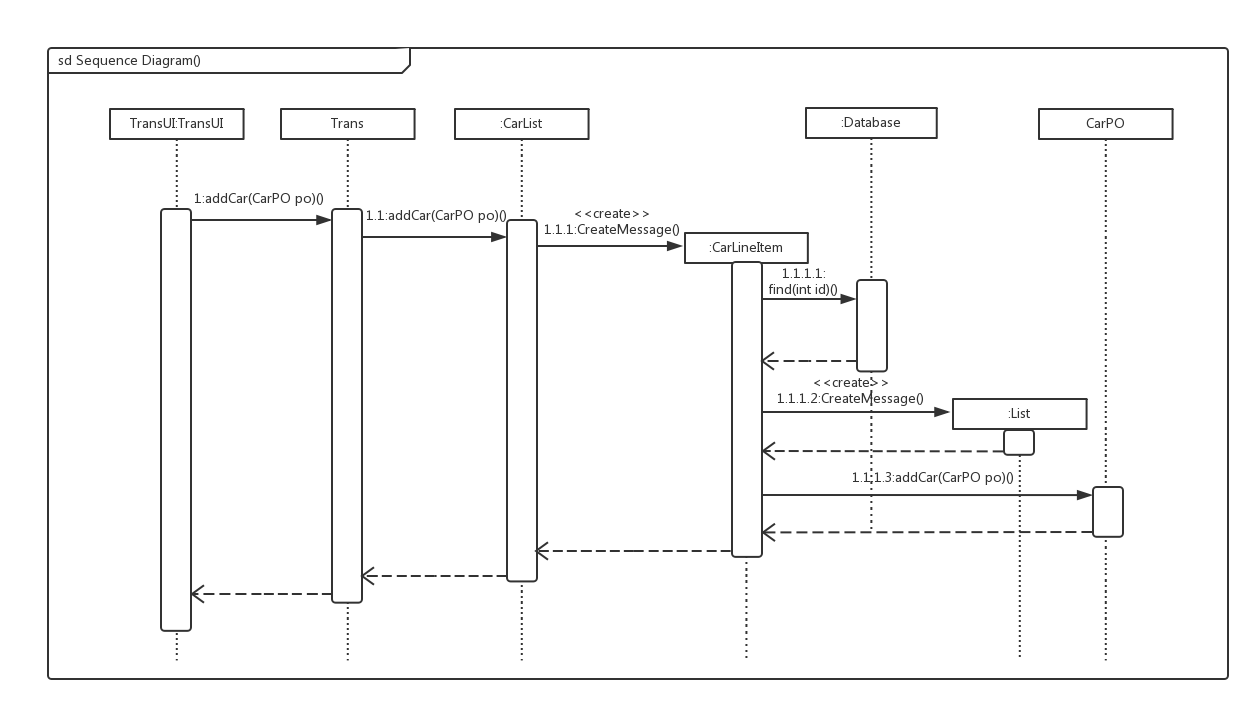
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| TransController. addLoading | 语法 | public ResultMessage addLoading(LoadingPO po); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的addLoading方法 |
| TransController. getIncome | 语法 | public double getIncome(double income,long id); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的getIncome方法 |
| TransController. endTrans | 语法 | public void endTrans(); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的endTrans方法 |
| TransController. getCost | 语法 | public double getCost(WarePO po) |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的getCost方法 |
| TransController. getTime | 语法 | public double getTime(WarePO po); |
| 前置条件 | 已创建一个Trans领域对象，存在职员 |
| 后置条件 | 调用Trans领域对象的getTime方法 |
| TransController. addCar | 语法 | public ResultMessage addCar(CarPO po); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的addCar方法 |
| TransController. deleteCar | 语法 | public ResultMessage deleteCar(long id); |
| 前置条件 | 已创建一个Trans领域对象，存在机构 |
| 后置条件 | 调用Trans领域对象的deleteCar方法 |
| TransController. getCar | 语法 | public CarVO getCar(long id); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的getCar方法 |
| TransController.getDriver | 语法 | public DriverVO getDriver(long id); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的getDriver方法 |
| TransController. addDriver | 语法 | public ResultMessage addDriver(DriverPO po,long id); |
| 前置条件 | 已创建一个Trans领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Trans领域对象的addDriver方法 |
| TransController. deleteDriver | 语法 | public ResultMessage deleteDriver(long id); |
| 前置条件 | 已创建一个Trans领域对象 |
| 后置条件 | 调用Trans领域对象的deleteDriver方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Trans. getCost | 输出报价 | |
| Trans. getTime | 输出估计时间 | |
| Trans. addCar | 增加相应车辆信息 | |
| Trans. deleteCar | 删除相应车辆信息 | |
| Trans. getCar | 得到相应车辆信息 | |
| Trans. getDriver | 得到相应司机信息 | |
| Trans. deleteDriver | 删除相应司机信息 | |
| Trans. addLoading | 增加相应装运信息 | |
| Trans. getIncome | 在相应快递员下保存输入的收款 | |
| Trans. endTrans | 结束运输任务 | |

**Trans的接口规范**

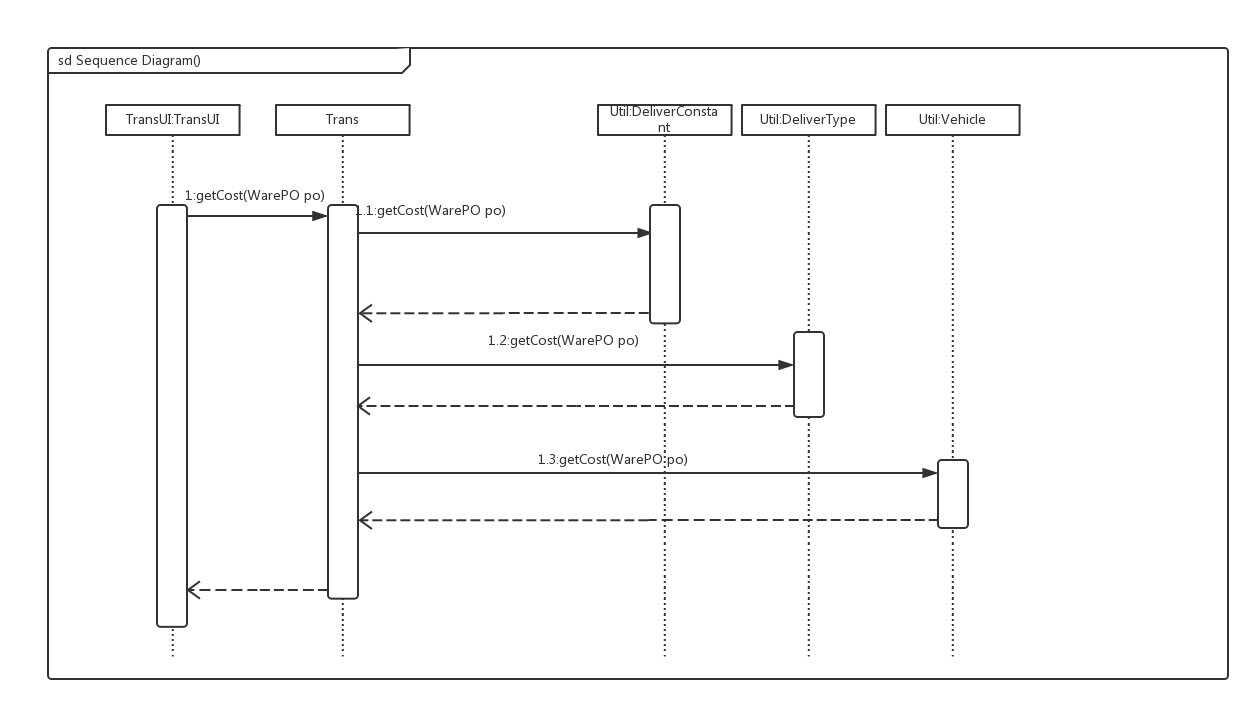
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Trans. addLoading | 语法 | public ResultMessage addLoading(LoadingPO po); |
| 前置条件 | 输入的装载信息合法且完整 |
| 后置条件 | 保存相应的装载信息 |
| Trans. getIncome | 语法 | public double getIncome(double income,long id); |
| 前置条件 | 输入的id正确且输入的收入数值合法 |
| 后置条件 | 在相应快递员下保存输入的收款 |
| Trans. endTrans | 语法 | public void endTrans(); |
| 前置条件 | 已完成运输 |
| 后置条件 | 结束任务，持久化更新设计的领域对象的数据 |
| Trans. getCost | 语法 | public double getCost(WarePO po) |
| 前置条件 | 输入的货物信息合法且完整 |
| 后置条件 | 显示相应报价 |
| Trans. getTime | 语法 | public double getTime(WarePO po); |
| 前置条件 | 输入的货物信息合法且完整 |
| 后置条件 | 显示相应预估时间 |
| Trans. addCar | 语法 | public ResultMessage addCar(CarPO po); |
| 前置条件 | 输入的车辆信息合法且完整 |
| 后置条件 | 保存相应的车辆信息 |
| Trans. deleteCar | 语法 | public ResultMessage deleteCar(long id); |
| 前置条件 | 输入车辆id合法且存在 |
| 后置条件 | 去除该条车辆信息 |
| Trans. getCar | 语法 | public CarVO getCar(long id); |
| 前置条件 | 输入车辆id合法且存在 |
| 后置条件 | 显示相应车辆信息 |
| Trans.getDriver | 语法 | public DriverVO getDriver(long id); |
| 前置条件 | 输入司机id合法且存在 |
| 后置条件 | 显示相应司机信息 |
| Trans. addDriver | 语法 | public ResultMessage addDriver(DriverPO po,long id); |
| 前置条件 | 输入的司机信息合法且完整 |
| 后置条件 | 保存相应的司机信息 |
| Trans. deleteDriver | 语法 | public ResultMessage deleteDriver(long id); |
| 前置条件 | 输入司机id合法且存在 |
| 后置条件 | 去除该条司机信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| TransDataService. findWare(long id) | 按ID进行查找单一WarePO持久化对象 | |
| TransDataService. insertWare(WarePO po) | 插入单一WarePO持久化对象 | |
| TransDataService. deleteWare(WarePO po) | 删除单一WarePO持久化对象 | |
| TransDataService. findDriver(long id) | 按ID进行查找单一DriverPO持久化对象 | |
| TransDataService. insertDriver(DriverPO po) | 插入单一DriverPO持久化对象 | |
| TransDataService. deleteDriver(DriverPO po) | 删除单一DriverPO持久化对象 | |
| TransDataService.FindCar (long id) | 按ID进行查找单一CarPO持久化对象 | |
| TransDataService. insertCar (CarPO po) | 插入单一CarPO持久化对象 | |
| TransDataService. deleteCar (CarPO po) | 删除单一CarPO持久化对象 | |

（4）业务逻辑层的动态模型

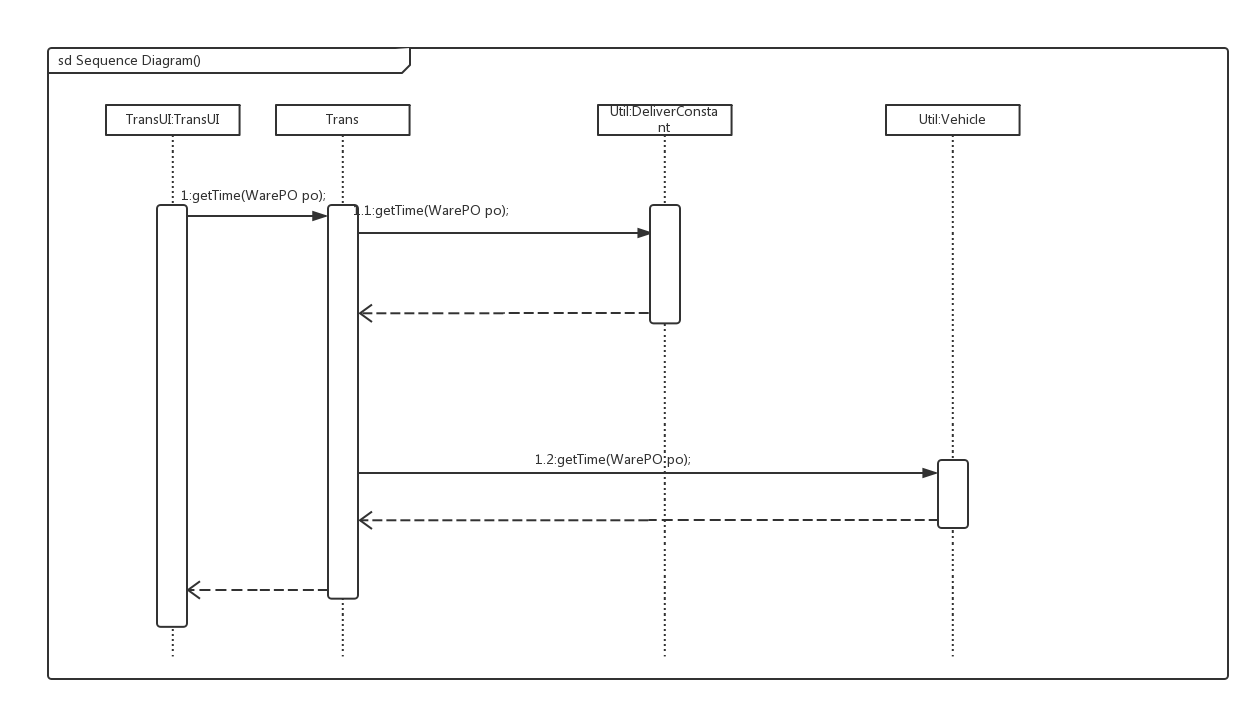
如图表明了物流快递系统中，当用户输入相关信息之后，运输逻辑处理的相关对象之间的协作。



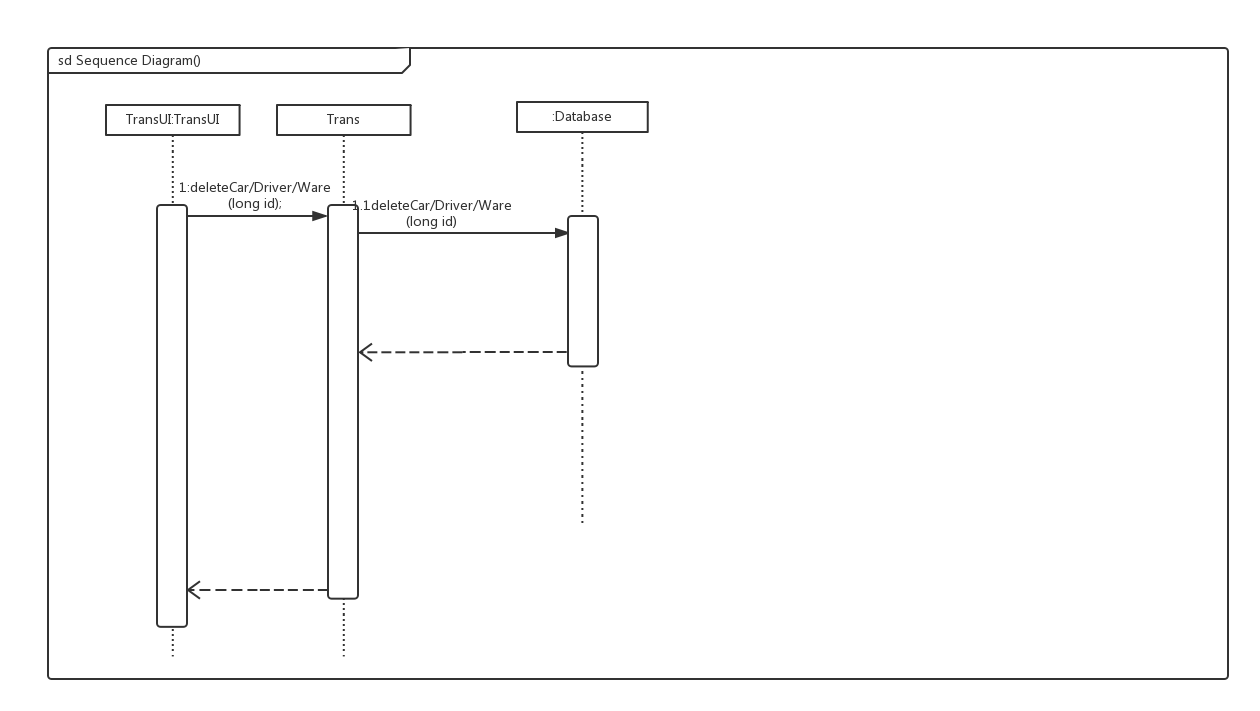
**图 添加车辆的顺序图**



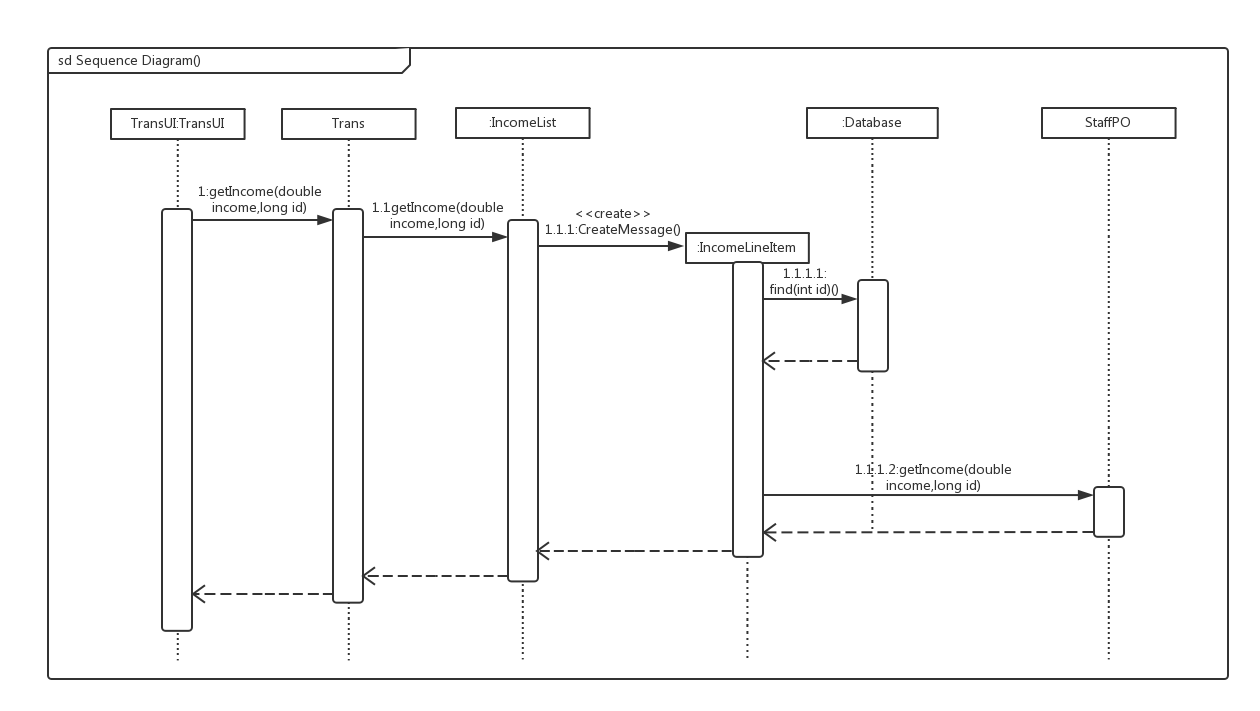
**图 得到报价的顺序图**



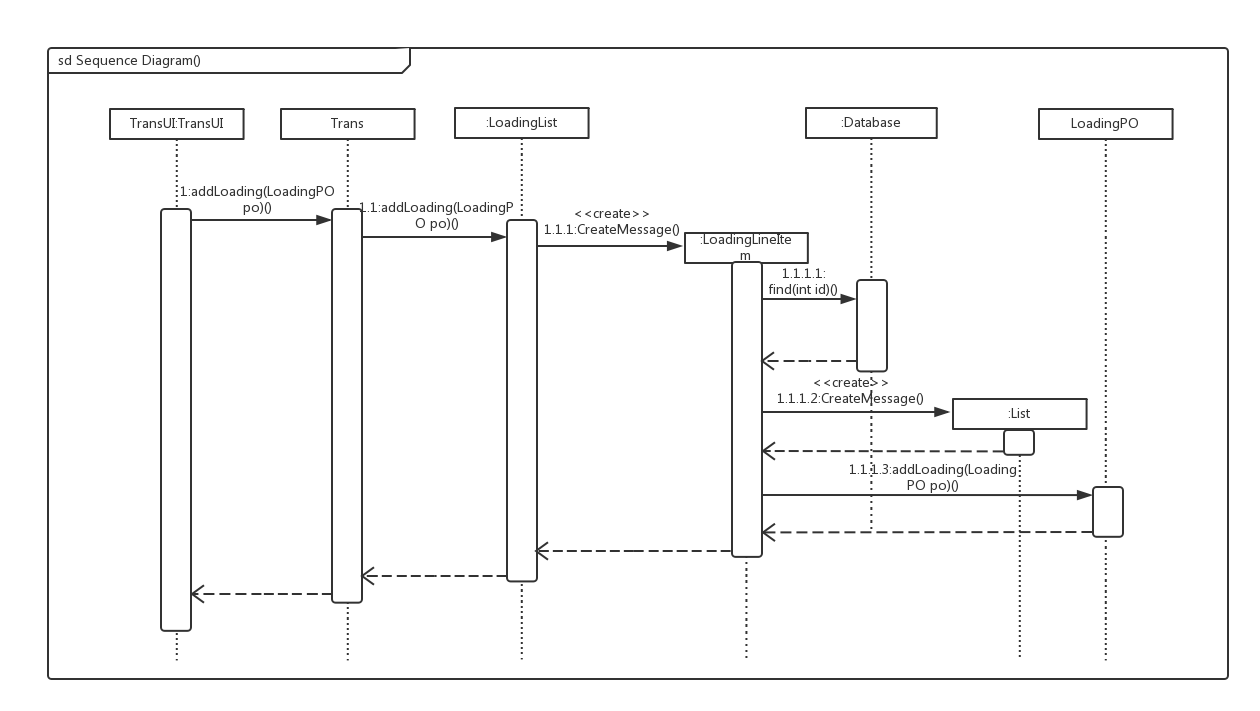
**图 得到预计时间的顺序图**



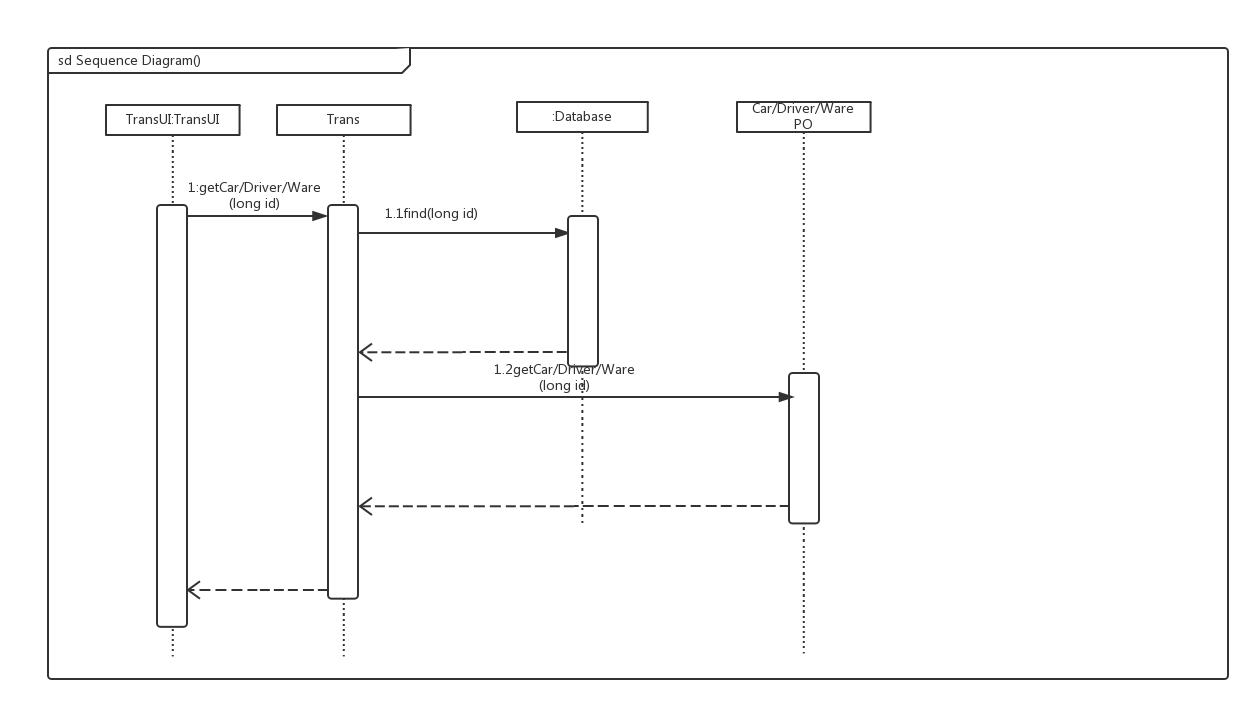
**图 删除车辆（司机，货物）的顺序图**



**图 收款单的顺序图**

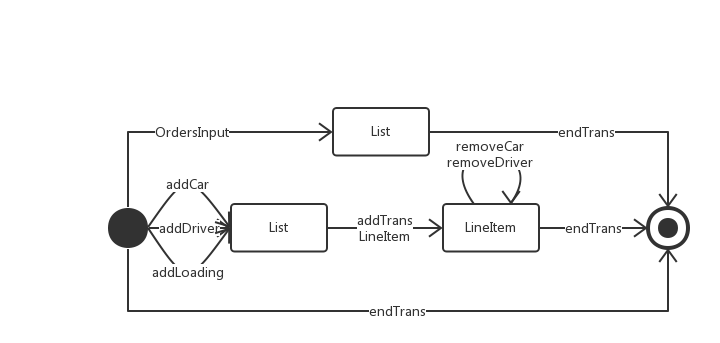


**图 添加装运信息的顺序图**



**图 得到车辆（司机，货物）信息的顺序图**

如图所示的状态图描述了List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。



（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 listbl 模板

（1）模块描述

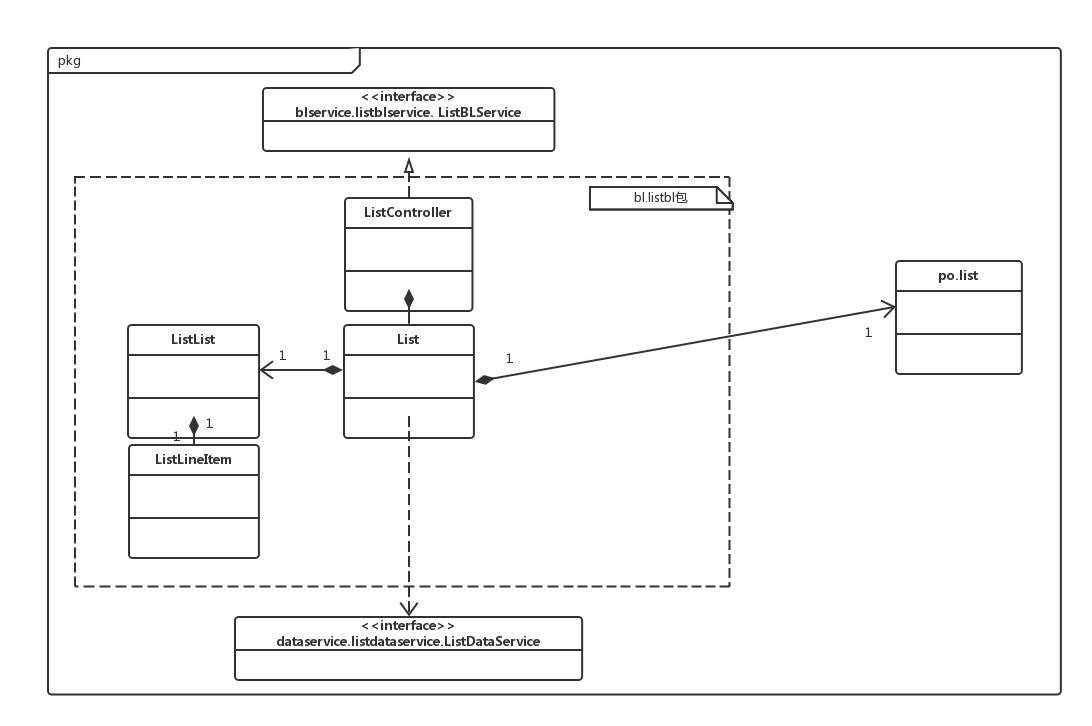
listbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

listbl模块的职责及接口参见软件系统结构描述文档表16。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice. listblservice. ListBLService接口。业务逻辑层和数据层之间添加 dataservice.listdataservice. ListDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 ListController,这样ListController会将对单据的业务逻辑处理委托给List对象。listPO是作为单据记录的持久化对象被添加到设计模型中去的。而ListList和ListLineItem的添加是List的容器类。ListLineItem保有单据的数据。而 ListList封装了关于 ,ListLineItem的数据集合的数据结构的秘密。

listbl模块的设计如图1所示。



listbl模块各个类的职责如表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现日志所需要的服务 |
| listController | 负责实现单据界面所需要的服务 |
| Log | 日志的领域模型对象,可以解决日志问题。 |
| List | 运输的领域模型对象,拥有一次管理所需的单据信息 |

（3）模块内部类的结构规范

ListController和List的接口规范如表2和表3所示。

**表2 ListController的接口规范**

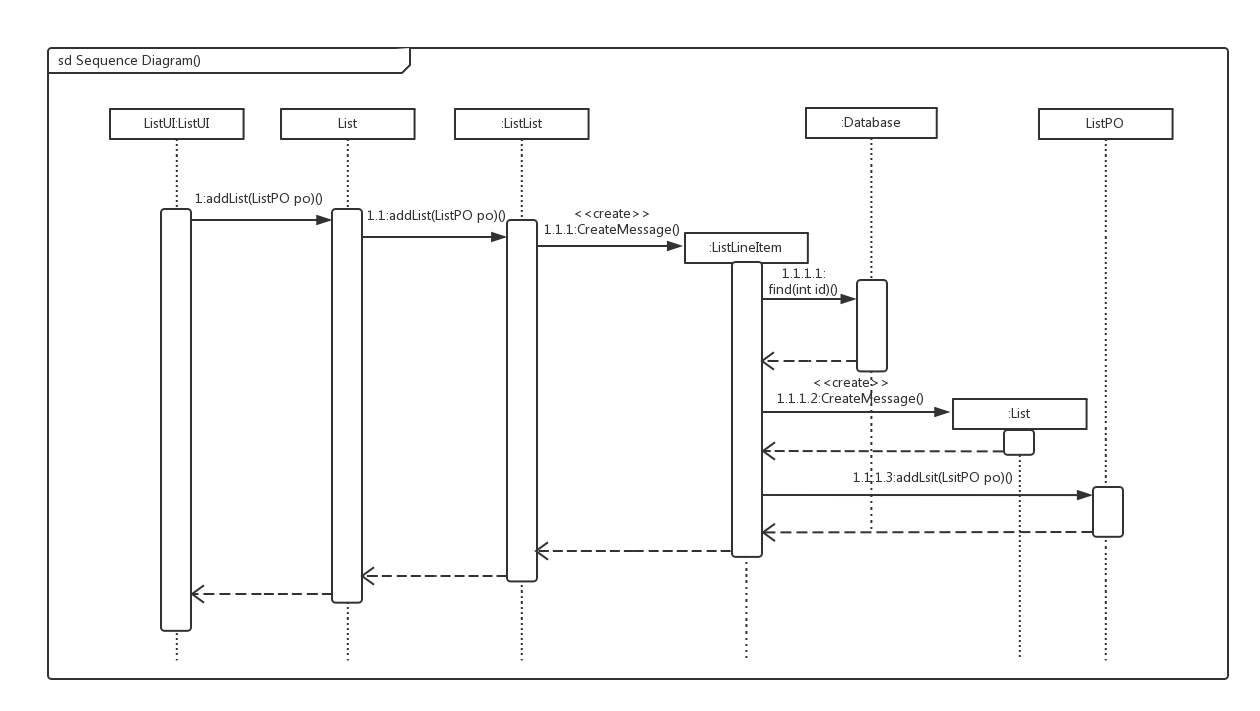
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| ListController. modifyList | 语法 | public ResultMessage modifyList(long id,ListPO po); |
| 前置条件 | 已创建一个List领域对象，并且输入符合输入规则 |
| 后置条件 | 调用List领域对象的modifyList方法 |
| ListController. delrteList | 语法 | public ResultMessage delrteList(long id); |
| 前置条件 | 已创建一个List领域对象，存在单据 |
| 后置条件 | 调用List领域对象的delrteList方法 |
| ListController.addList | 语法 | public ResultMessage addList(long id, ListPO po); |
| 前置条件 | 已创建一个List领域对象，并且输入符合输入规则 |
| 后置条件 | 调用List领域对象的addList方法 |
| ListController.exportList | 语法 | public ResultMessage exportList(long id); |
| 前置条件 | 已创建一个List领域对象，存在机构 |
| 后置条件 | 调用List领域对象的exportList方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| List. modifyList | 修改一个单据对象 | |
| List. delrteList | 删除一个单据对象 | |
| List. addList | 加入一个单据对象 | |
| List. exportList | 导出一个单据对象 | |

**表3 List的接口规范**

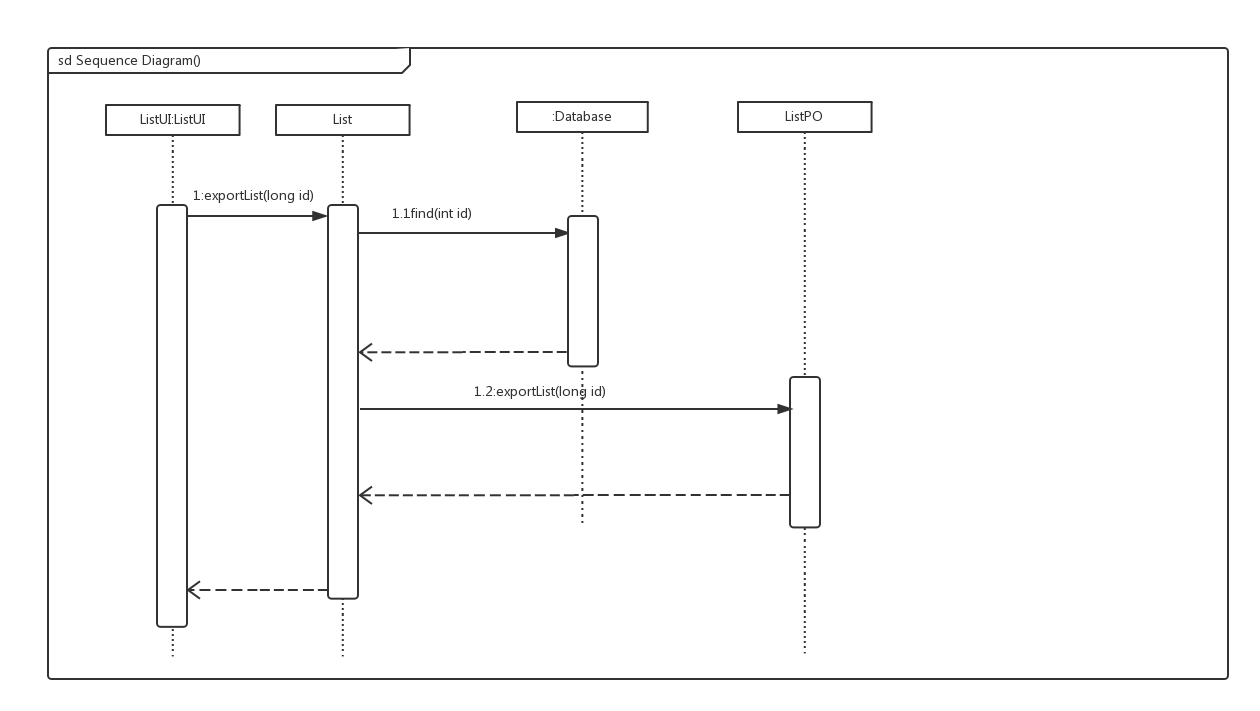
|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| List. modifyList | 语法 | public ResultMessage modifyList(long id,ListPO po); |
| 前置条件 | 输入的id合法且存在 |
| 后置条件 | 修改相应单据信息 |
| List. delrteList | 语法 | public ResultMessage delrteList(long id); |
| 前置条件 | 输入的id合法且存在 |
| 后置条件 | 删除相单据信息 |
| List. addList | 语法 | public ResultMessage addList(long id, ListPO po); |
| 前置条件 | 输入的id合法 |
| 后置条件 | 增加相应单据信息 |
| List. exportList | 语法 | public ResultMessage exportList(long id); |
| 前置条件 | 输入的id合法且存在 |
| 后置条件 | 导出相应单据信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| ListDataService.insert(ListPO po) | 插入单一持久化对象 | |
| ListDataService.delete(ListPO po) | 删除单一持久化队形 | |
| ListDataService.update(ListPO po) | 更新单一持久化对象 | |
| ListDataService.updateList (ListPO po) | 更新单一持久化List对象 | |
| ListDataService.find(int id) | 根据ID进行查找单一持久化对象 | |
| ListDataService.init() | 初始化持久化数据库 | |
| DatabaseFactory.finish() | 结束持久化数据库 | |

（4）业务逻辑层的动态模型

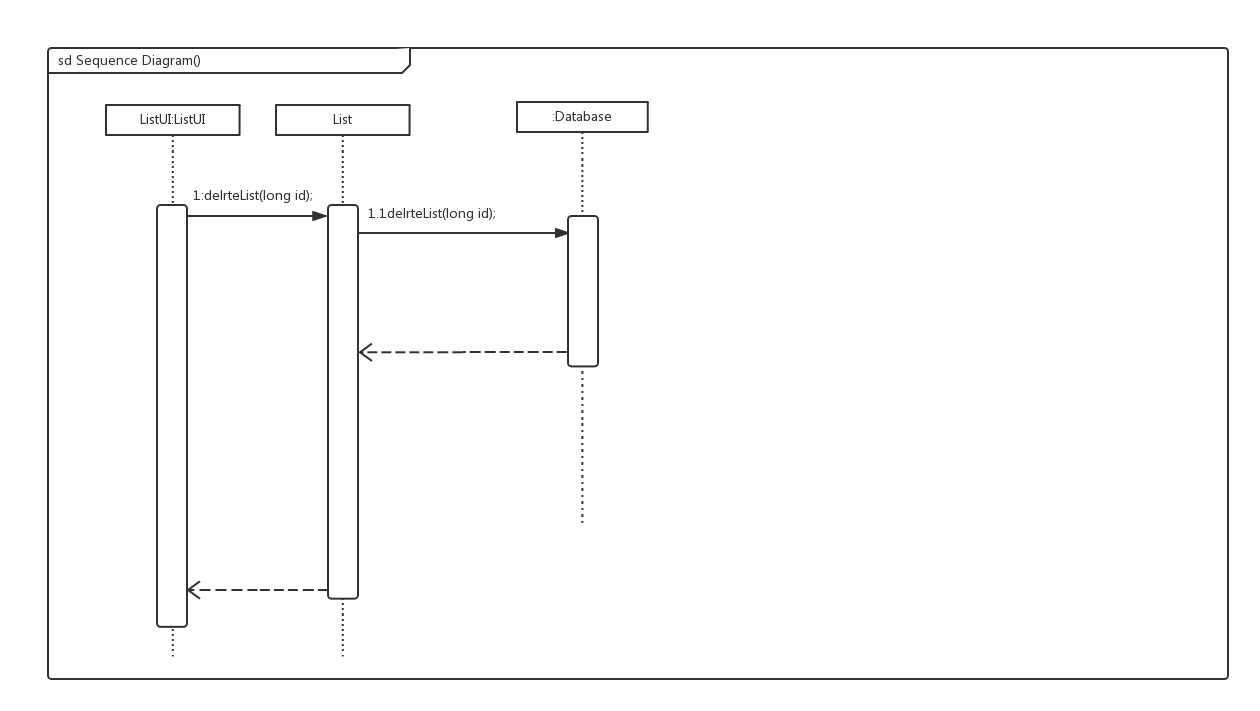
如图表明了物流快递系统中，当用户输入相关信息之后，单据业务逻辑处理的相关对象之间的协作。



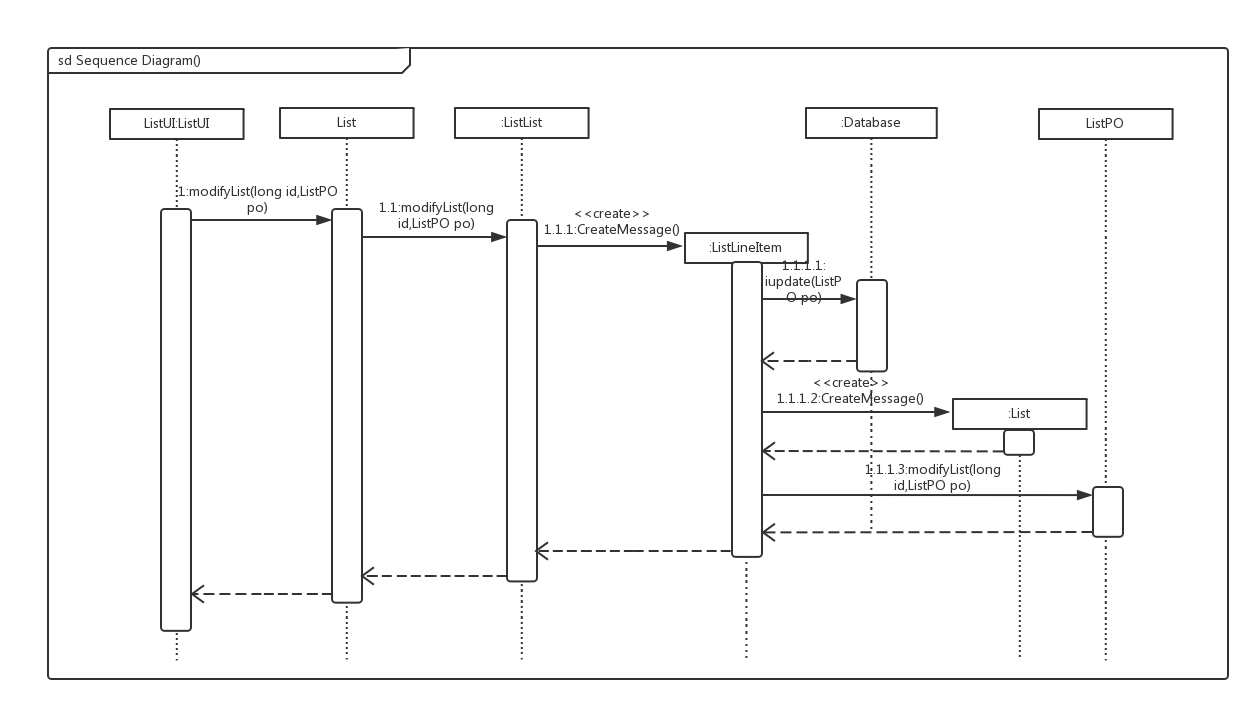
**图 添加单据顺序图**



**图 导出单据顺序图**

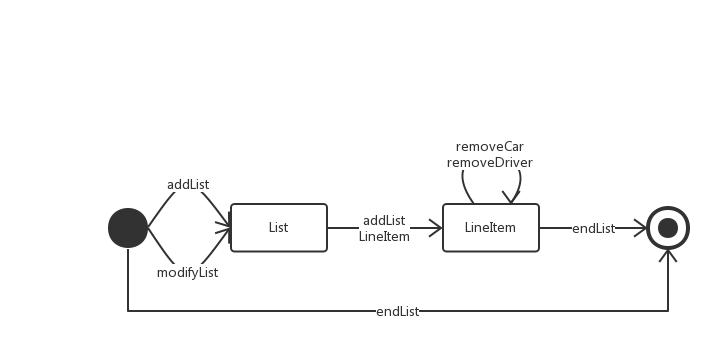


**图 删除单据顺序图**



**图 修改单据顺序图**

如图所示的状态图描述了List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。

****

（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。

4.1.5 inquirebl模块

（1）模块描述

inquirebl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。

inquirebl模块的职责及接口参见软件系统结构描述文档表13。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice. inquireblservice.InquireBLService接口。业务逻辑层和数据层之间添加 dataservice. inquiredataservice.InquireDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 InquireController,这样InquireController会将对查询逻辑处理委托给Inquire对象。InquirePO是作为审查记录的持久化对象被添加到设计模型中去的。而InquireList和InquireLineItem的添加是List的容器类。InquireLineItem保有职员、机构和的审查单据的数据的职责。而 InquireList封装了关于 InquireLineItem的数据集合的数据结构的秘密和制定薪水的职责。

inquirebl模块的设计如图1所示。

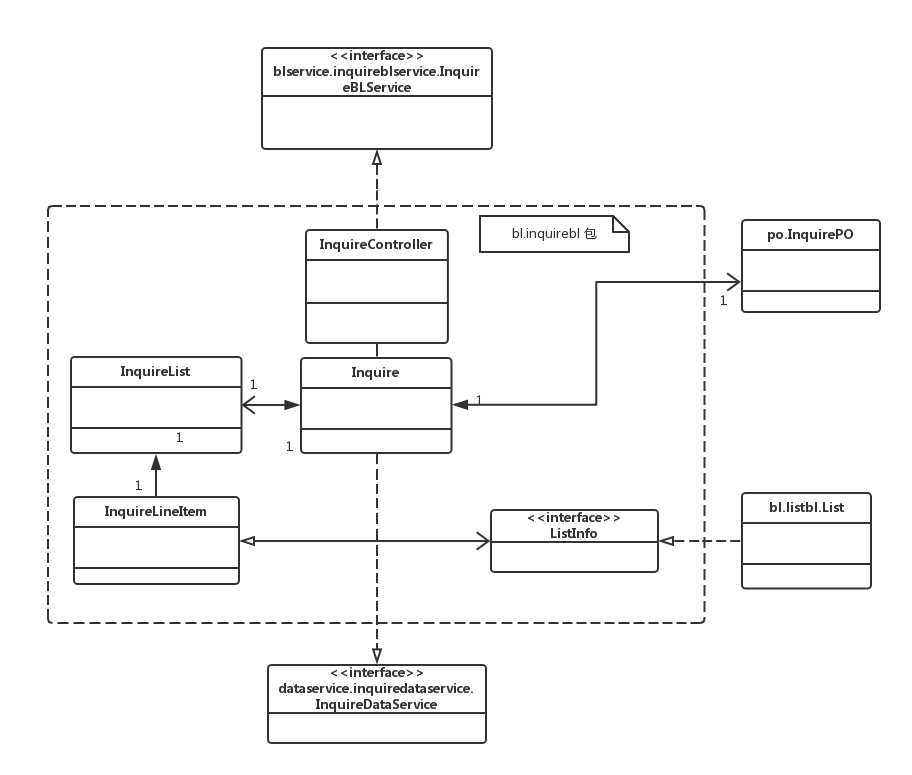


图1 inquirebl模块各个类的设计

inquirebl模块各个类的职责如表1所示。

**表1 Inquirebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现日志所需要的服务 |
| InquireController | 负责实现查询界面所需要的服务 |
| Log | 日志的领域模型对象,可以解决日志问题。 |
| Inquire | 查询的领域模型对象,可以帮助完成查询界面所需要的服务。 |

（3）模块内部类的结构规范

InquireController和Inquire的接口规范如表2和表3所示。

**表2 InquireController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| InquireController.inquire | 语法 | Public ResultMessage inquire (long id) |
| 前置条件 | 输入一个快件id，并且输入符合输入规则 |
| 后置条件 | 调用Inquire领域对象的inquire方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Inquire.inquire | 查询一次快件物流信息 | |

**表3 Inquire的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Inquire.inquire | 语法 | Public ResultMessage inquire (long id) |
| 前置条件 | 输入的快件id合法且存在 |
| 后置条件 | 显示物流信息界面 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| DatabaseFactory.getInquireDatabase | 得到物流信息数据库服务的引用 | |
| DatabaseFactory.getListDatabase | 得到快件单据信息数据库服务的引用 | |
| Inquiredataservice.find(long id) | 查找单一持久化对象 | |

（4）业务逻辑层的动态模型

图2表明了物流快递系统中，当用户输入相关信息之后，查询业务逻辑处理的相关对象之间的协作。

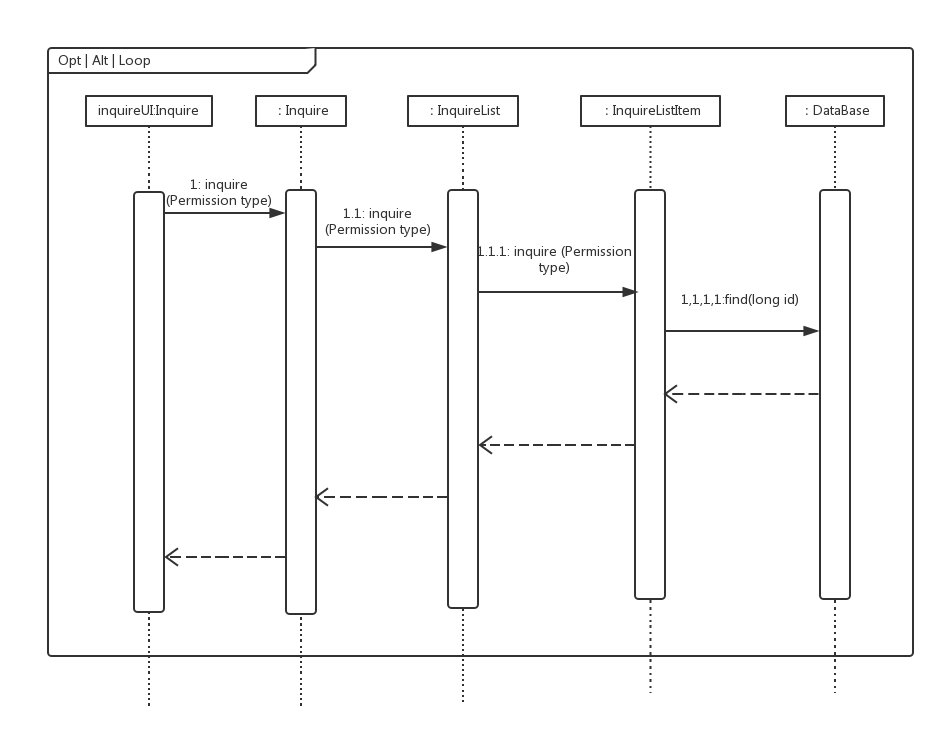


图2 查询快件的流程图

如图3所示的状态图描述了Inquire对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着inquire方法被UI调用，Inquire进入List状态；之后通过添加对象进入LineItem状态。可以由图看出是一个相当简单的过程。

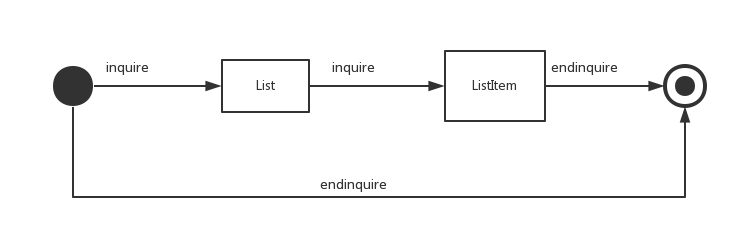


图3 Inquire对象状态图

（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。

其他略。

4.1.6 financebl模块

（1）模块描述

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表14。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice. financeblservice.FinanceBLService接口。业务逻辑层和数据层之间添加 dataservice. financedataservice.FinanceDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 FinanceController,这样FinanceController会将对查询逻辑处理委托给Finance对象。FinancePO是作为审查记录的持久化对象被添加到设计模型中去的。而FinanceList和FinanceLineItem的添加是List的容器类。FinanceLineItem保有的职责。而 InquireList封装了关于 InquireLineItem的数据集合的数据结构的秘密和制定薪水的职责。

inquirebl模块的设计如图1所示。

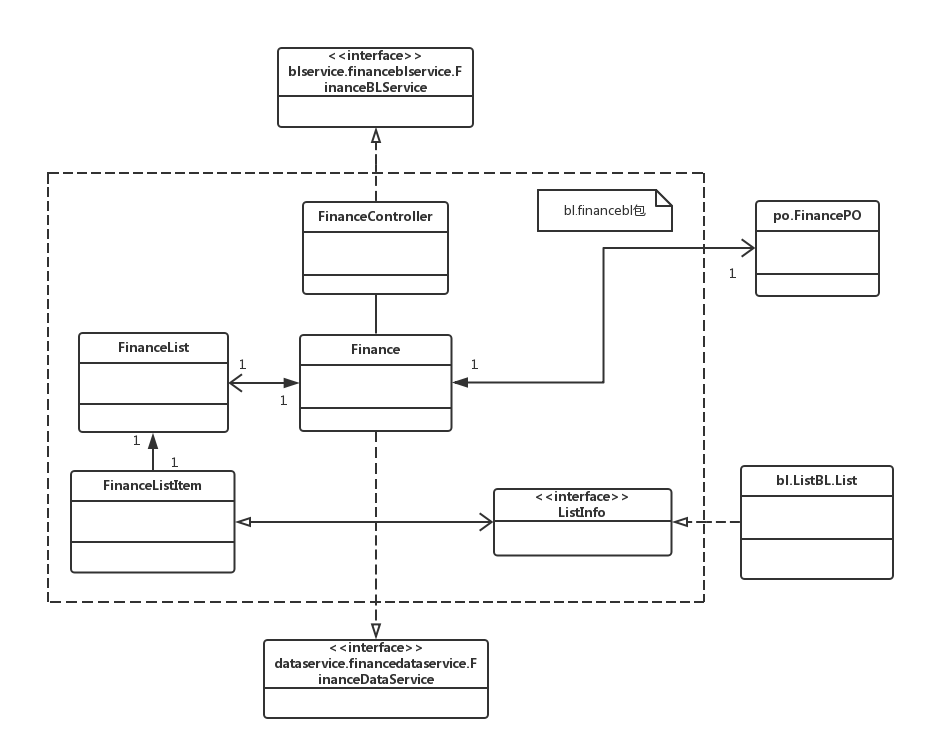


图1 financebl模块各个类的设计

financebl模块各个类的职责如表1所示。

**表1 Financebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现日志所需要的服务 |
| FinanceController | 负责实现财务界面所需要的服务 |
| Log | 日志的领域模型对象,可以解决日志问题。 |
| Finance | 财务的领域模型对象,可以帮助完成财务界面所需要的服务。 |

（3）模块内部类的结构规范

FinanceController和Finance的接口规范如表2和表3所示。

**表2 FinanceController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Finance.BalanceManagement | 语法 | Public ResultMessage BalanceMan (FinancePO po) |
| 前置条件 | 结算管理合法并经过允许 |
| 后置条件 | 调用Finance领域对象的BalanceManagement方法 |
| Finance.CostManagement | 语法 | Public ResultMessage CostMan (BAccount ba，costchartPO po,) |
| 前置条件 | 成本管理合法并经过允许 |
| 后置条件 | 调用Finance领域对象的CostManagement方法 |
| Finance.StatisticalChart | 语法 | Public ResultMessage StatisticalChart (statisticalchartPO po) |
| 前置条件 | 更新经营情况表合法并经过允许 |
| 后置条件 | 调用Finance领域对象的StatisticalChart方法 |
| Finance.BaccountReview | 语法 | Public ResultMessage BaccountReview(Baccount ba) |
| 前置条件 | 账户查询合法并经过允许 |
| 后置条件 | 调用Finance领域对象的BaccountReview方法 |
| Finance.BaccountInit | 语法 | Public ResultMessage BaccountInit (BaccountPO po)) |
| 前置条件 | 账户管理合法并经过允许 |
| 后置条件 | 调用Finance领域对象的BaccountInit方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| Finance.BalanceManagement | 进行一次结算管理 | |
| Finance.CostManagement | 进行一次成本管理 | |
| Finance.StatisticalChart | 更新经营情况表 | |
| Finance.BaccountReview | 检查一次账户信息 | |
| Finance.BaccountInit | 进行一次账户管理 | |

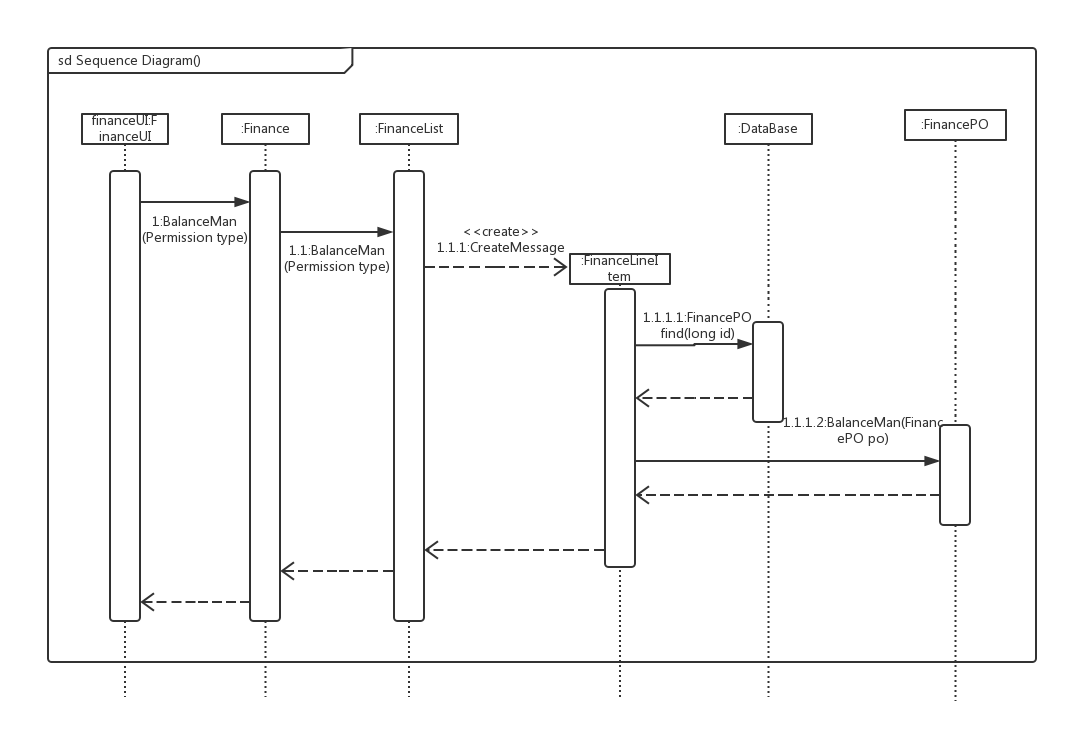
**表3 Finance的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Finance.BalanceManagement | 语法 | Public ResultMessage BalanceMan (FinancePO po) |
| 前置条件 | 输入的财政信息完整 |
| 后置条件 | 系统更新结算管理相关的信息，如收款日期等 |
| Finance.CostManagement | 语法 | Public ResultMessage CostMan (BAccount ba，costchartPO po,) |
| 前置条件 | 传入的账户正确，报表信息完整 |
| 后置条件 | 系统更新成本管理相关的账户，成本收益表及租金等数据信息 |
| Finance.StatisticalChart | 语法 | Public ResultMessage StatisticalChart (statisticalchartPO po) |
| 前置条件 | 输入的报表信息完整 |
| 后置条件 | 系统更新经营情况表 |
| Finance.BaccountReview | 语法 | Public ResultMessage BaccountReview(Baccount ba) |
| 前置条件 | 传入的账户对象正确 |
| 后置条件 | 显示账户信息 |
| Finance.BaccountInit | 语法 | Public ResultMessage BaccountInit (BaccountPO po) |
| 前置条件 | 仅限最高权限财务人员id得到确认输入账户信息完整 |
| 后置条件 | 系统初始化账户信息 |
| **需要的服务（需8接口）** | | |
| 服务名 | 服务 | |
| Financedataservice.find(long id) | 查找单一持久化对象 | |
| Financedataservice.insert(FinancePO po) | 在数据库中插入FinancePO对象 | |
| Financedataservice.updata (FinancePO po) | 在数据库中更新FinancePO对象 | |
| Financedataservice.insert(costchartPO po) | 在数据库中插入costchartPO对象 | |
| DatabaseFactory.getFinanceDatabase | 得到财务信息数据库服务的引用 | |
| Financedataservice.insert(statisticalchartPO po) | 在数据库中插入statisticalchartPO对象 | |
| Financedataservice.insert(BaccountPO po) | 在数据库中插入BaccountPO对象 | |

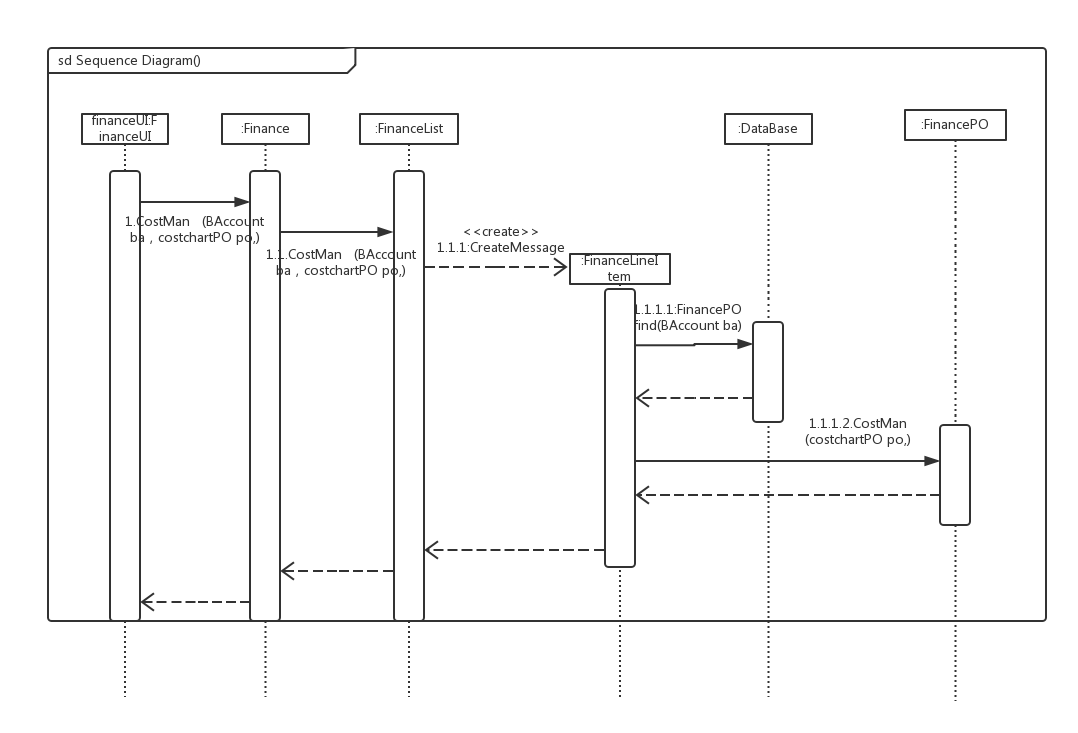
（4）业务逻辑层的动态模型

图2表明了物流快递系统中，当用户输入相关信息之后，审查业务逻辑处理的相关对象之间的协作。

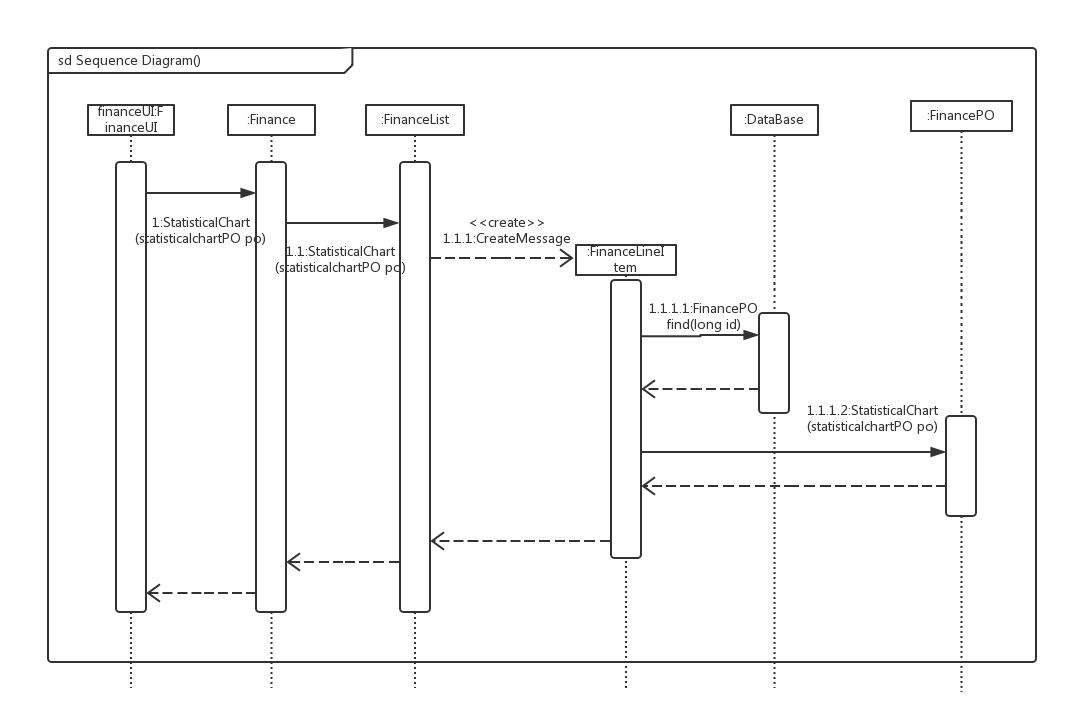
图2



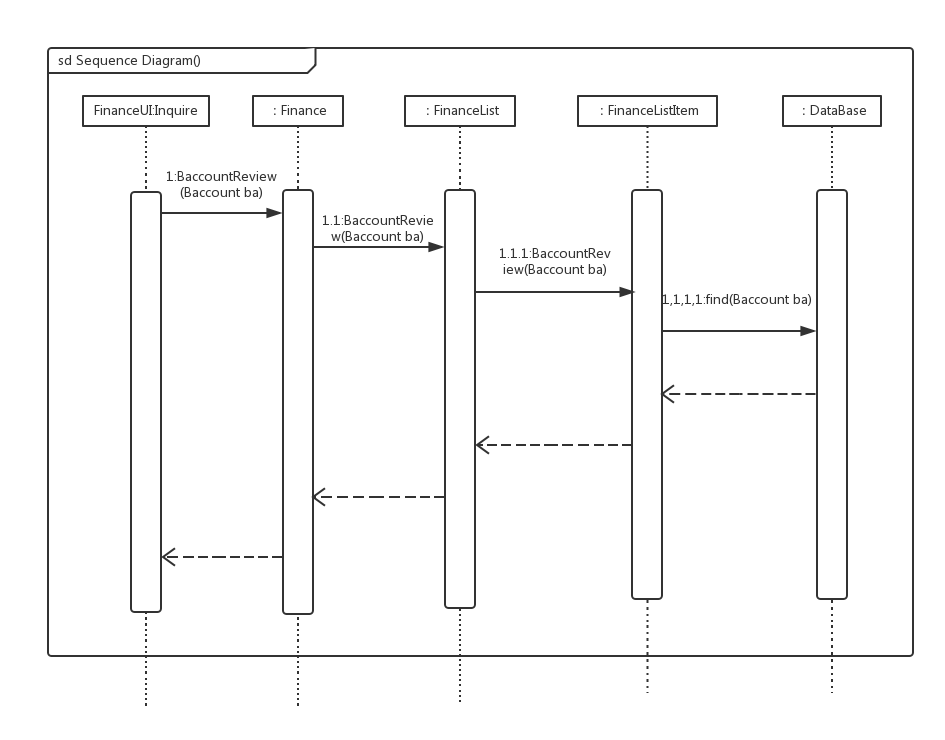
结算管理的流程图



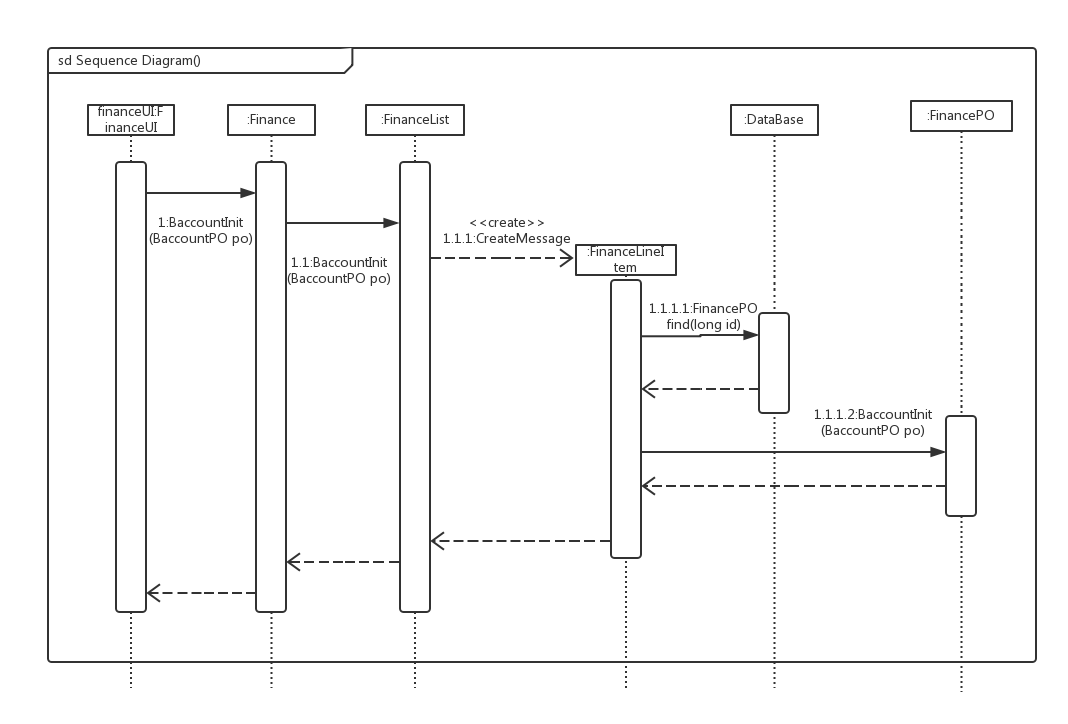
成本管理的流程图



统计报表的流程图



账户查询的顺序图



账户管理的顺序图

如图3所示的状态图描述了Inquire对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着BalanceManagement等方法被UI调用，Inquire进入List状态；之后通过添加对象进入LineItem状态。所有方法都要通过List。具体如图

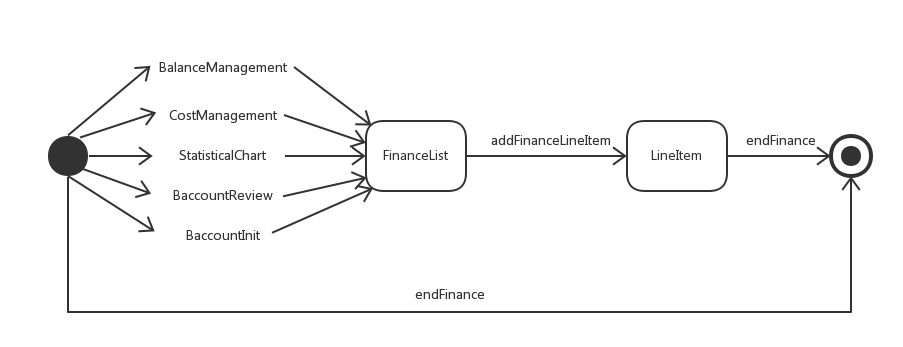


图3 Inquire对象状态图

（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。

其他略。

4.1.7 loginbl模块

（1）模块描述

loginbl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。

Loginbl模块的职责及接口参见软件系统结构描述文档表16。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice.loginblservice. LoginBLService接口。业务逻辑层和数据层之间添加 dataservice.logindataservice. LoginDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 LoginController,这样LoginController会将对销售的业务逻辑处理委托给Login对象。AccountPO是作为审查记录的持久化对象被添加到设计模型中去的。

Loginbl 模块的设计如图1所示。

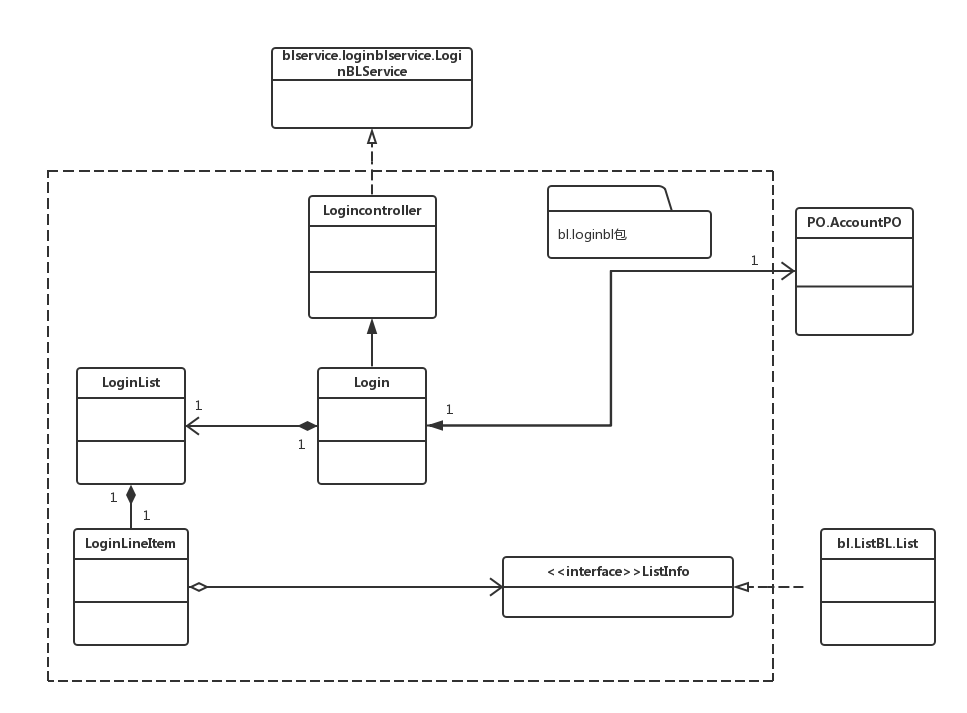


图1 loginbl模块各个类的设计

loginbl模块各个类的职责如表1所示。

**表1Loginbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现登录所需要的服务 |
| LoginController | 负责实现审查界面所需要的服务 |
| Log | 登录的领域模型对象,可以解决登录问题。 |
| Login | 登录的领域模型对象,拥有一次登录需的身份、账号、密码等信息,可以帮助完成审查界面所需要的服务。 |

（3）模块内部类的结构规范

LoginController和Login的接口规范如表2和表3所示。

**表2 LoginController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| LoginController.login | 语法 | Public typeOfLogin login(long account,String password) |
| 前置条件 | 已创建一个Login领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Login领域对象的login方法 |
| LoginController.find(AccountPO po) | 语法 | Public void find(AccountPO po) |
| 前置条件 | 已创建一个Login领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Login领域对象的find方法 |
| LoginController.insert(AccountPO po) | 语法 | Public void insert(AccountPO po) |
| 前置条件 | 已创建一个Login领域对象， |
| 后置条件 | 调用Login领域对象的insert方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| LoginController. getStaffDatabase | 得到登录人员数据库的引用 | |
| LoginController.find(AccountPO po) | 查找一个Login对象 | |
| LoginController.insert(AccountPO po) | 数据库中加入一个Login对象 | |

**表3 Login的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Login.login | 语法 | Public typeOfLogin login(long account,String password) |
| 前置条件 | 启动Login服务 |
| 后置条件 | 如果账号密码正确，跳转到相应的界面中，否则显示登录失败。 |
| Login.find(AccountPO po) | 语法 | Public void find(AccountPO po) |
| 前置条件 | 启动Login服务 |
| 后置条件 | 显示用户信息 |
| Login.insert(AccountPO po) | 语法 | Public void insert(AccountPO po)) |
| 前置条件 | 启动Login服务 |
| 后置条件 | 在人员和机构管理中，增加机构信息 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| LoginDataService.insert(AccountPO po) | 插入单一持久化对象 | |
| LoginDataService.delete(AccountPO po) | 删除单一持久化队形 | |
| LoginDataService.find(String username) | 根据用户名进行查找单一持久化对象 | |

（4）业务逻辑层的动态模型

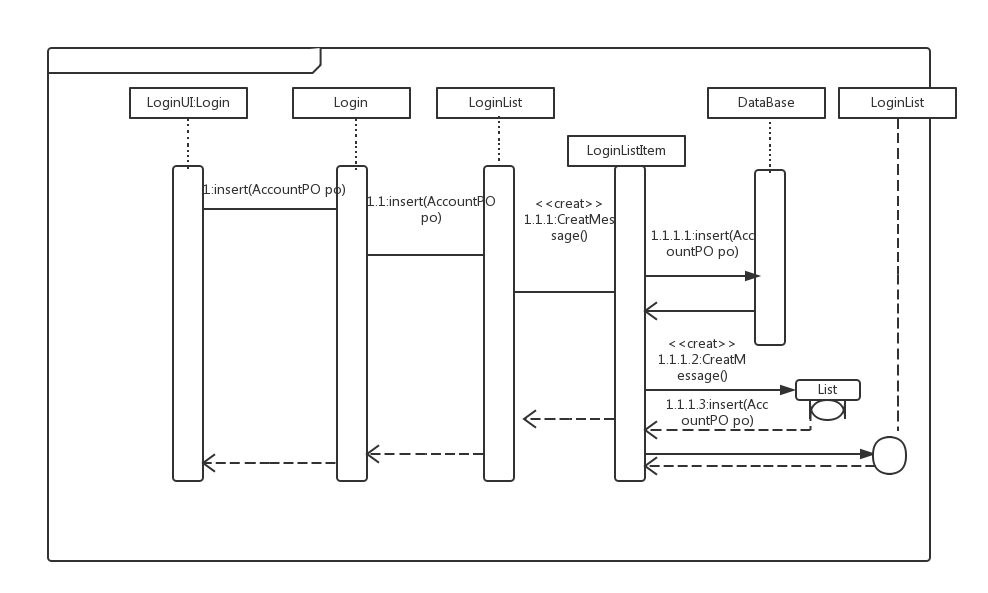
图2表明了物流快递系统中，当用户输入相关信息之后，审查业务逻辑处理的相关对象之间的协作。

图2 添加职员的顺序图

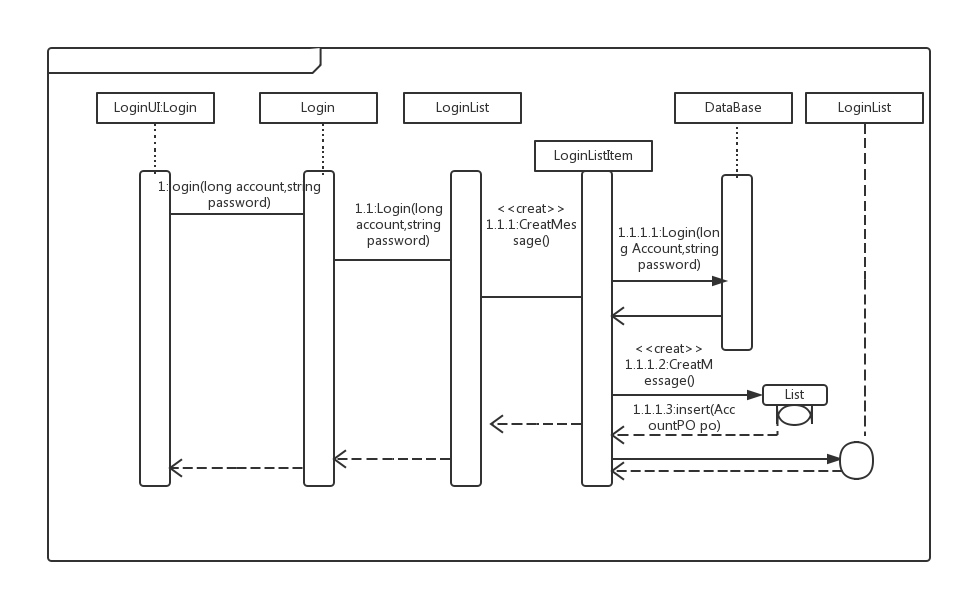


图3.登录的顺序图

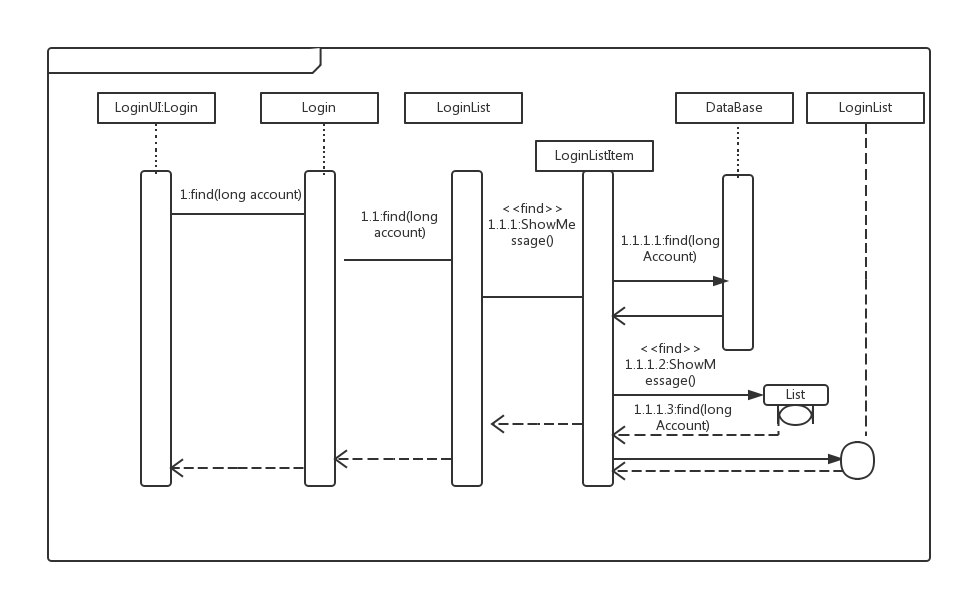


图3.查找的顺序图

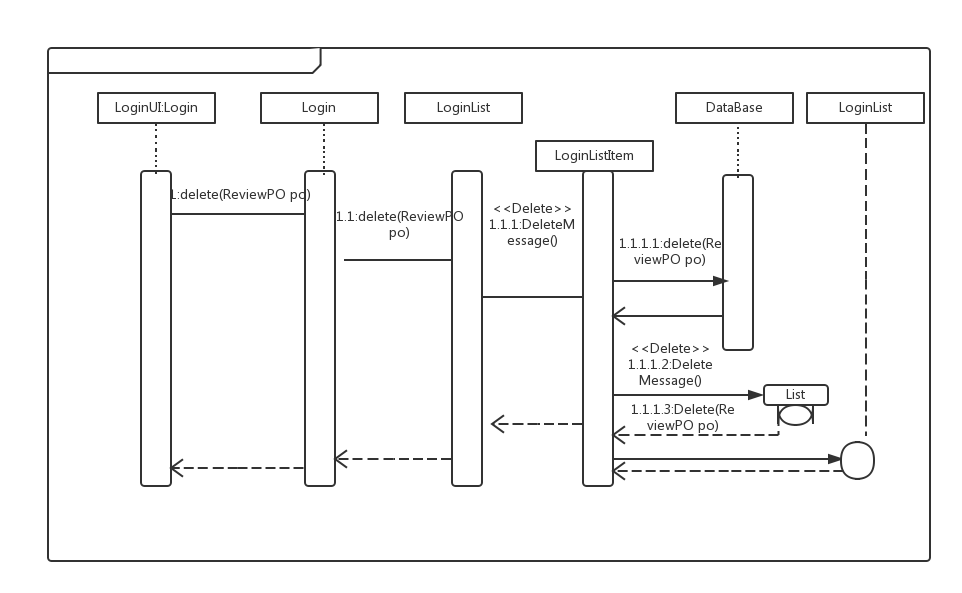
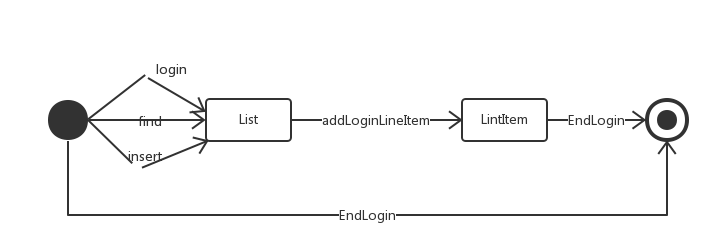


图4.删除的顺序图

如图3所示的状态图描述了Login对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着login、insert、find方法被UI调用，Login进入List状态；之后通过添加对象进入LineItem状态。

图3 Login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。

其他略。

4.1.8 accountbl模块

（1）模块描述

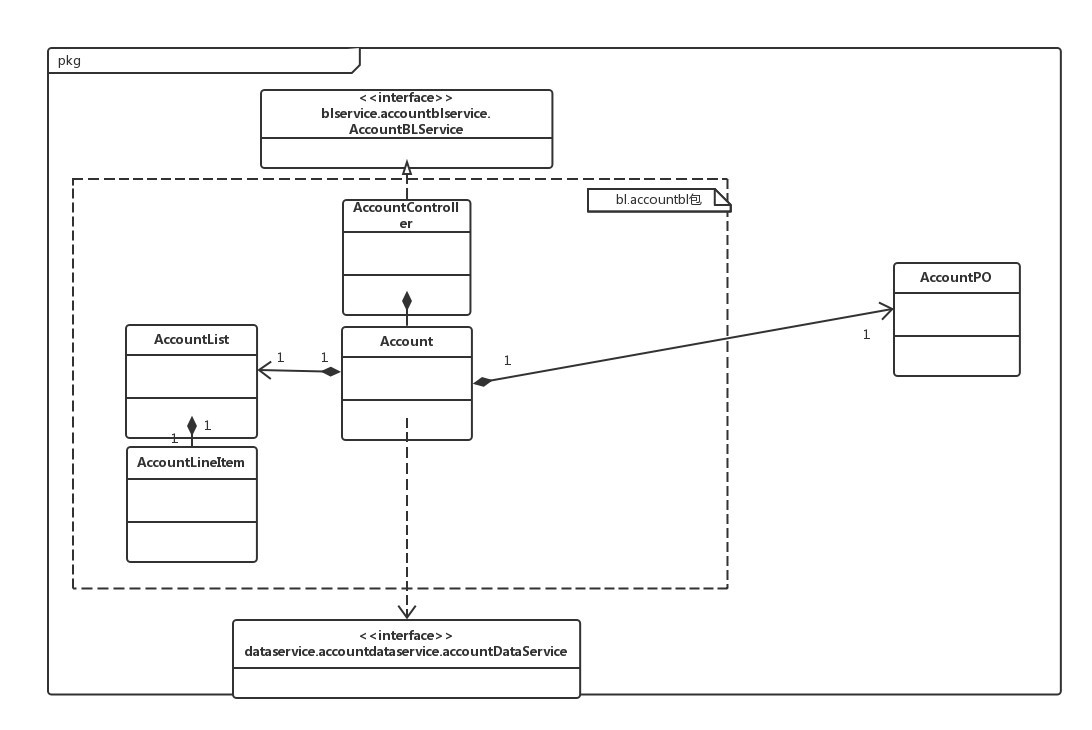
accountbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

accountbl模块的职责及接口参见软件系统结构描述文档表16。

（2）整体结构

根据体系结构的设计,我们将系统分为展现层、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blservice.accountblservice. AccountBLService接口。业务逻辑层和数据层之间添加 dataservice.accountdataservice.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会将对销售的业务逻辑处理委托给Account对象。AccountPO是作为审查记录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1所示。



accountbl模块各个类的职责如表1所示。

**表1Accountbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| LogController | 负责实现登录所需要的服务 |
| AccountController | 负责实现审查界面所需要的服务 |
| Log | 登录的领域模型对象,可以解决登录问题。 |
| Account | 登录的领域模型对象,拥有一次登录需的身份、账号、密码等信息,可以帮助完成审查界面所需要的服务。 |

（3）模块内部类的结构规范

AccountController和Account的接口规范如表2和表3所示。

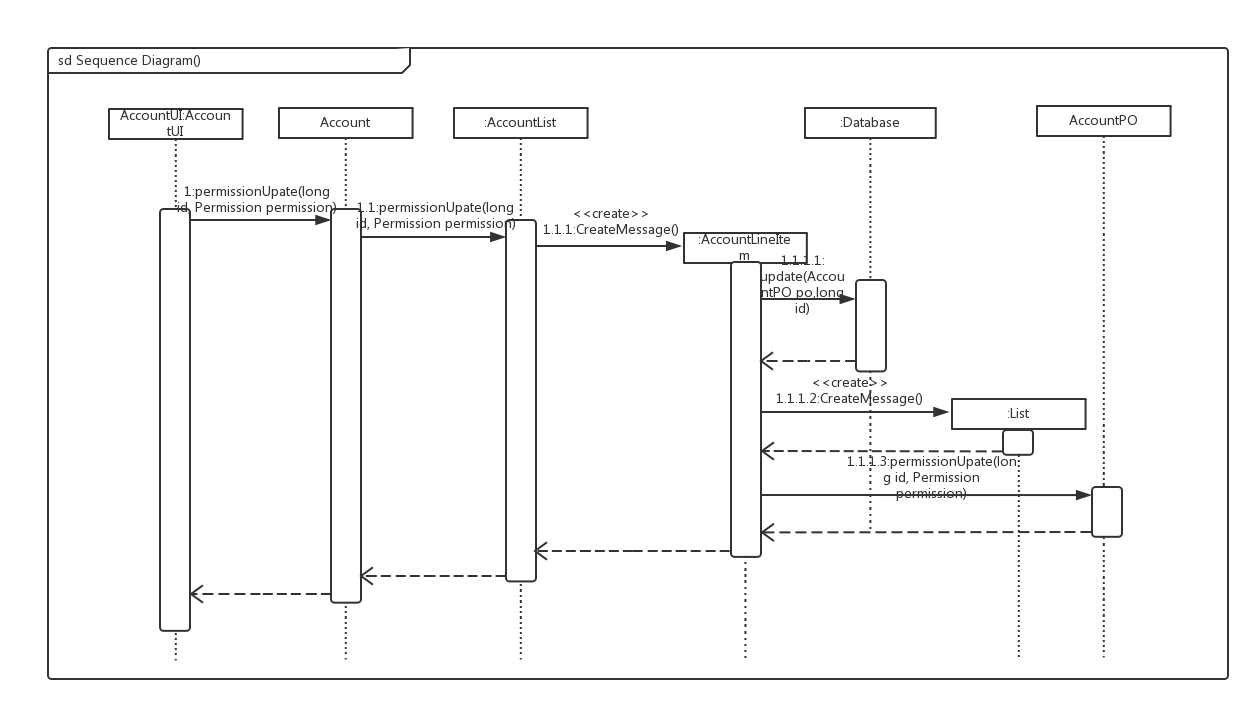
**表2 AccountController的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| AccountController. accountRev (long id) ; | 语法 | public ResultMessage accountRev (long id) ; |
| 前置条件 | 已创建一个Account领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Account领域对象的accountRev方法 |
| AccountController. permissionUpate (long id, Permission permission) ; | 语法 | public ResultMessage permissionUpate (long id, Permission permission) ; |
| 前置条件 | 已创建一个Account领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Account领域对象的permissionUpate方法 |
| AccountController.endAccount() | 语法 | P public void endAccount(); |
| 前置条件 | 已创建一个Account领域对象， |
| 后置条件 | 调用Account领域对象的endAccount方法 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| AccountController. getStaffDatabase | 得到管理人员数据库的引用 | |
| AccountController.find(AccountPO po) | 查找一个Account对象 | |
| AccountController.upadata(AccountPO po) | 数据库中修改一个Account对象 | |

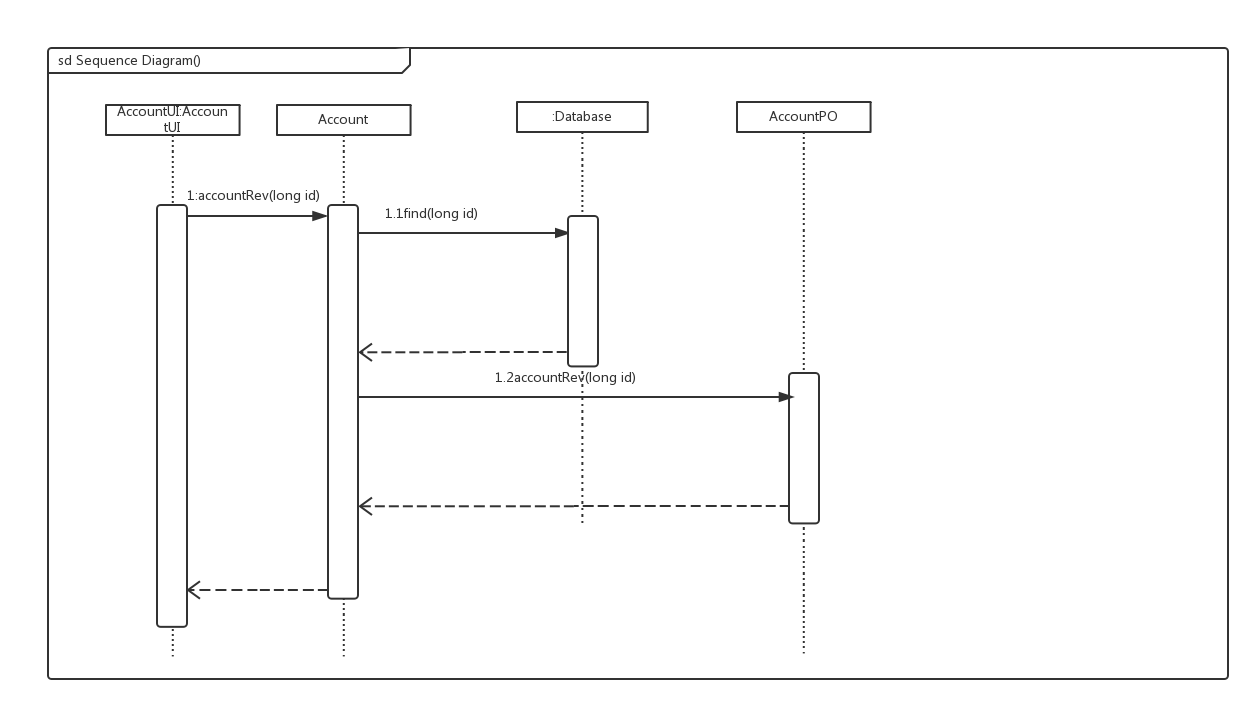
**表3 Account的接口规范**

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| Account. accountRev (long id) ; | 语法 | public ResultMessage accountRev (long id) ; |
| 前置条件 | 已创建一个Account领域对象，并且输入符合输入规则 |
| 后置条件 | 返回账号信息 |
| Account. permissionUpate (long id, Permission permission) ; | 语法 | public ResultMessage permissionUpate (long id, Permission permission) ; |
| 前置条件 | 已创建一个Account领域对象，并且输入符合输入规则 |
| 后置条件 | 修改账号信息 |
| Account.endAccount() | 语法 | P public void endAccount(); |
| 前置条件 | 已创建一个Account领域对象， |
| 后置条件 | 结束任务 |
| **需要的服务（需接口）** | | |
| 服务名 | 服务 | |
| AccountDataService.insert(AccountPO po) | 插入单一持久化对象 | |
| AccountDataService.updata(AccountPO po,long id) | 修改一个持久化对象 | |
| AccountDataService.delete(long id) | 删除单一持久化队形 | |
| AccountDataService.find(long id) | 根据用户名进行查找单一持久化对象 | |

（4）业务逻辑层的动态模型

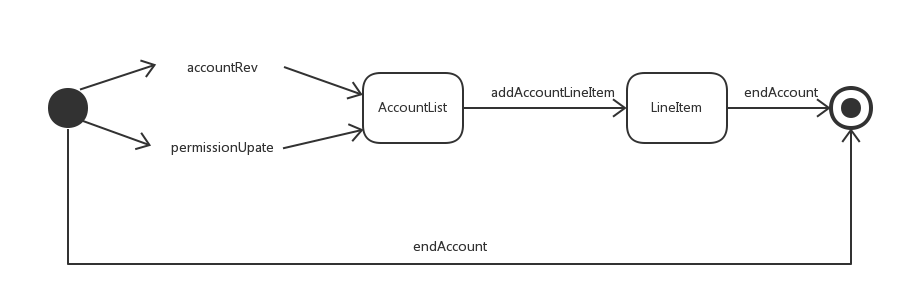


**图 修改账号信息顺序图**



**图 得到账号信息顺序图**

下图所示的状态图描述了account对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着permissionUpate、accountRev方法被UI调用，account进入List状态；之后通过添加对象进入LineItem状态。



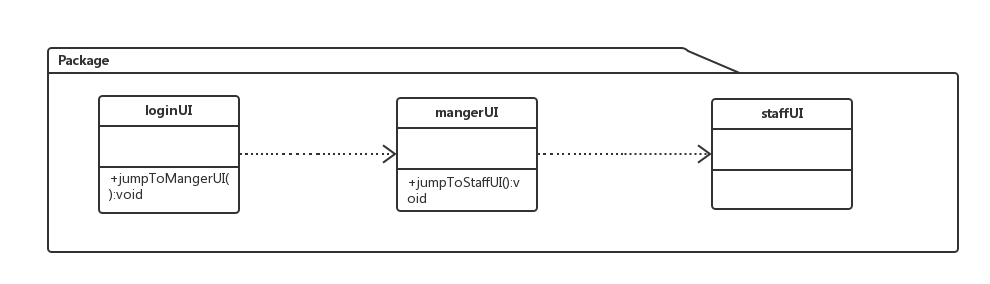
（5）业务逻辑层的设计原理

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。

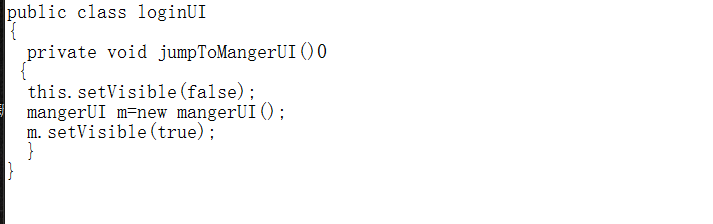
其他略。

4.2界面跳转

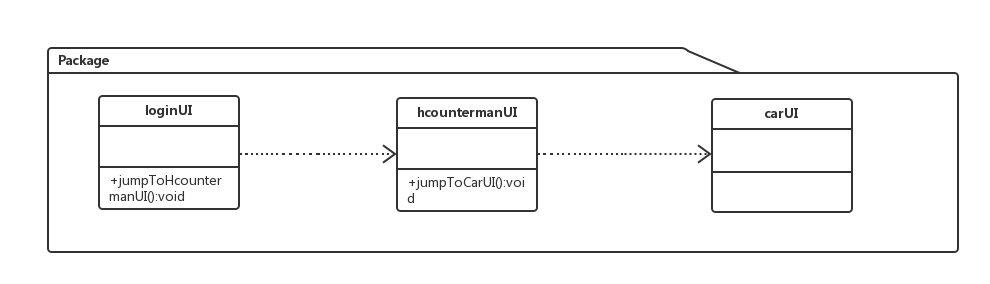
采用分散式控制风格如图



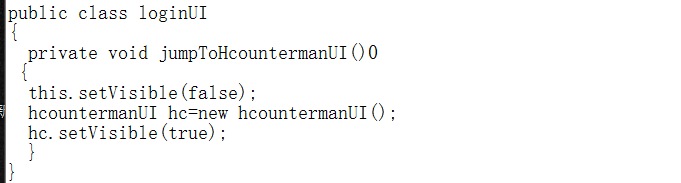
分散式控制风格类图



分散式控制风格代码



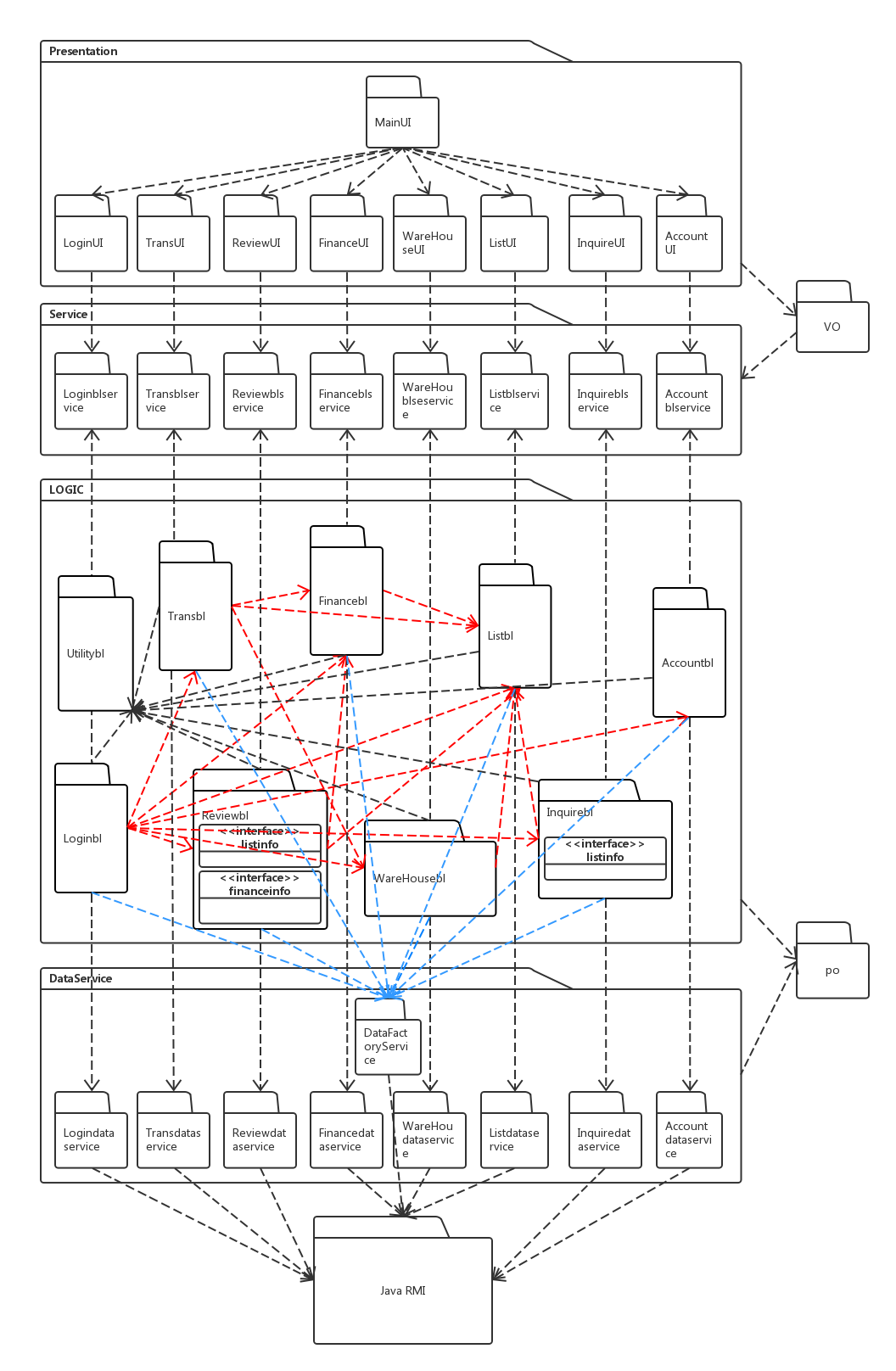
分散式控制风格类图



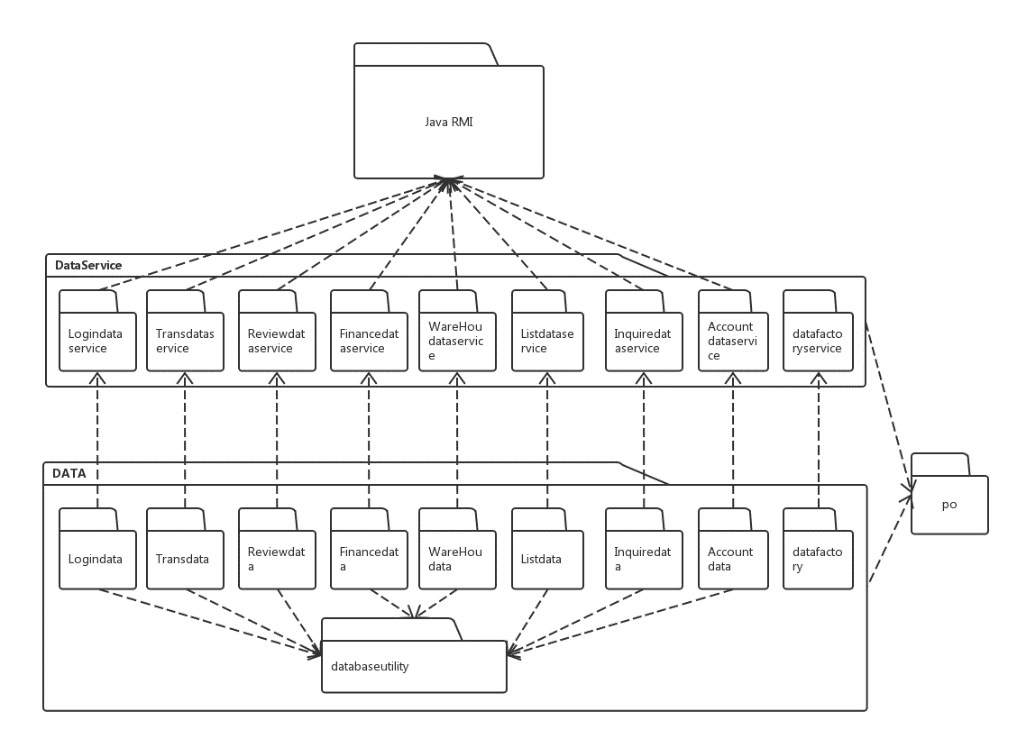
分散式控制风格代码

**5.依赖视角**

下图是客户端和服务器端各自的包的依赖关系



物流快递系统客户端开发包图



物流快递系统服务器端开发包图