李凯 朴圣哲 狄逸枫 傅林华

物流快递系统

软件详细设计模型

软件详细设计模型

**1.业务逻辑层的分解**

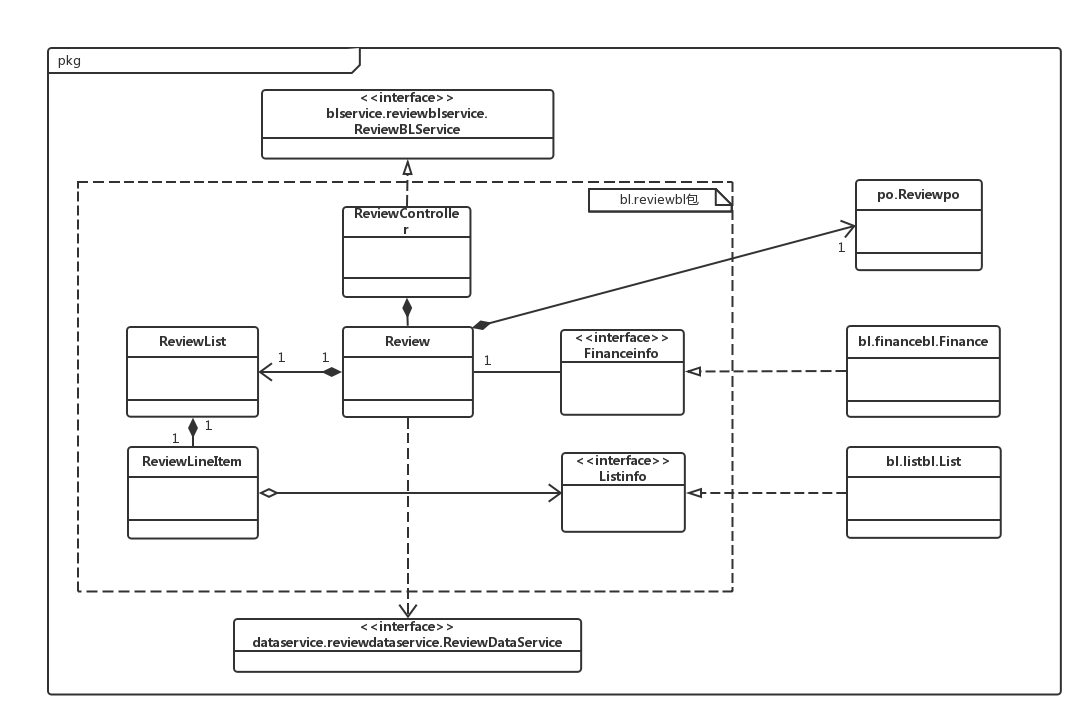
业务逻辑层的开发包图参见软件体系结构文档图3。

**1.1 reviewbl模块**

1.1.1整体结构

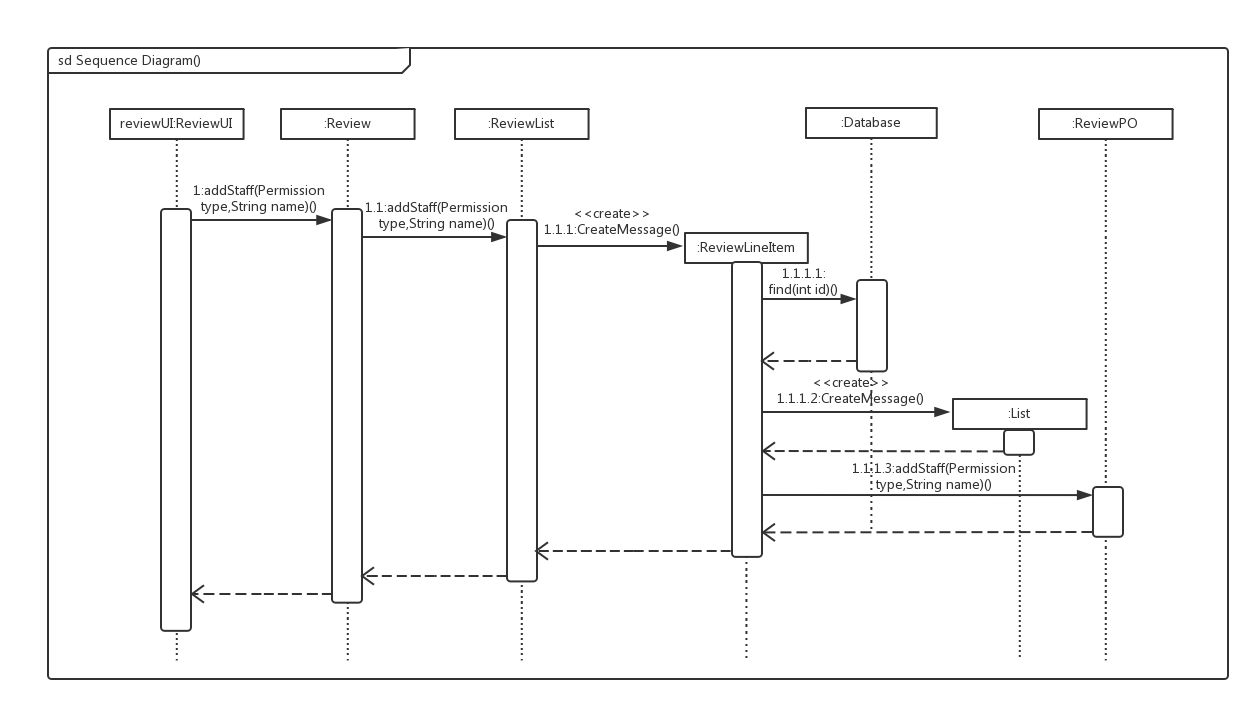
根据体系结构的设计,我们将系统分为展现层、业务逻辑层、数据层。每一层之间为了增加灵活性,我们会添加接口。比如展示层和业务逻辑层之间,我们添加blservice.reviewblservice. ReviewBLService接口。业务逻辑层和数据层之间添加 dataservice.reviewdataservice. ReviewDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 ReviewController,这样ReviewController会将对审查的业务逻辑处理委托给Review对象。ReviewPO是作为审查记录的持久化对象被添加到设计模型中去的。而ReviewList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。

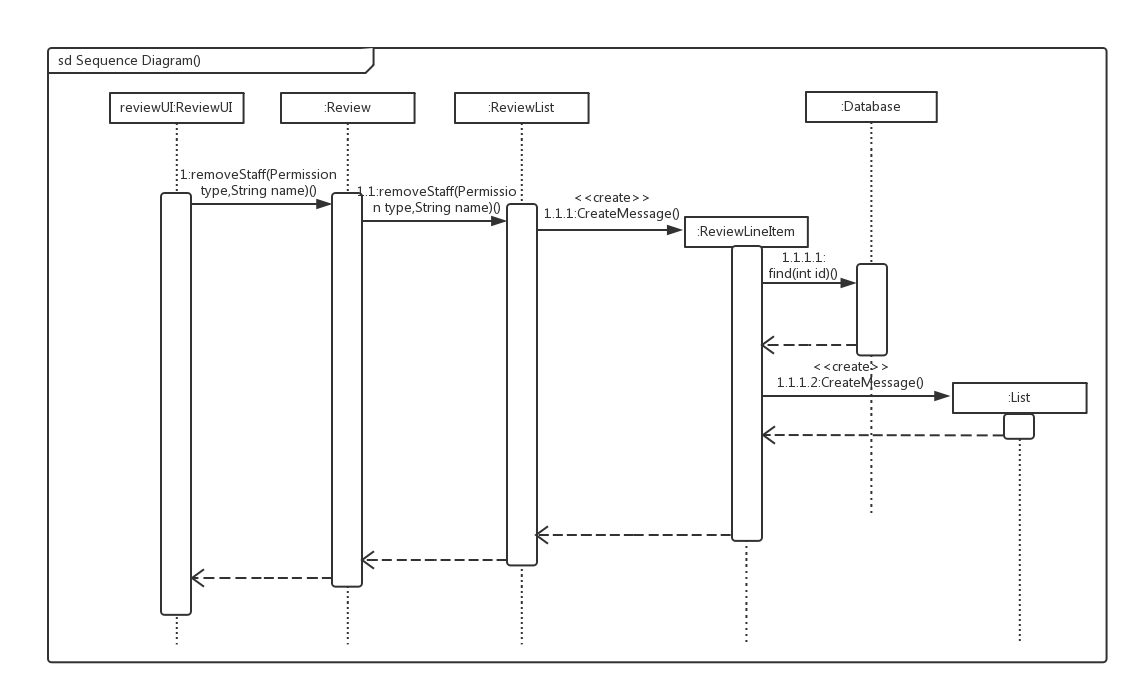
reviewbl模块的设计如图7所示。

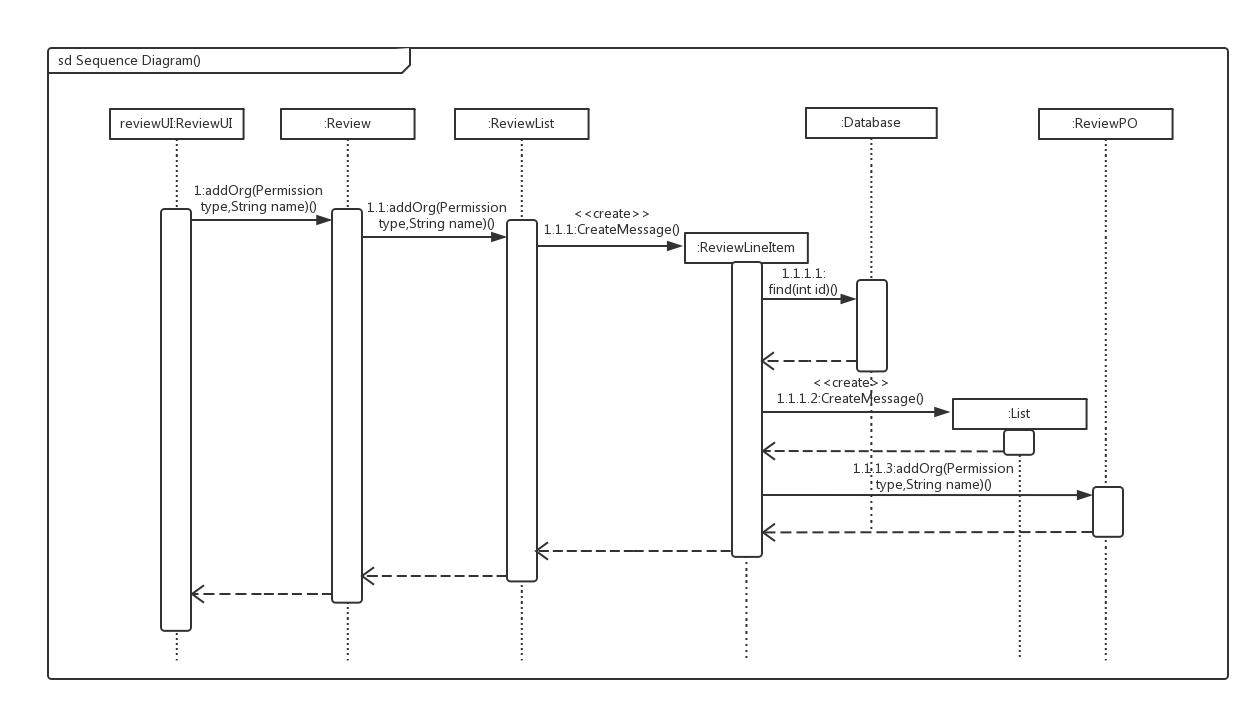
图7 reviewbl模块各个类的设计

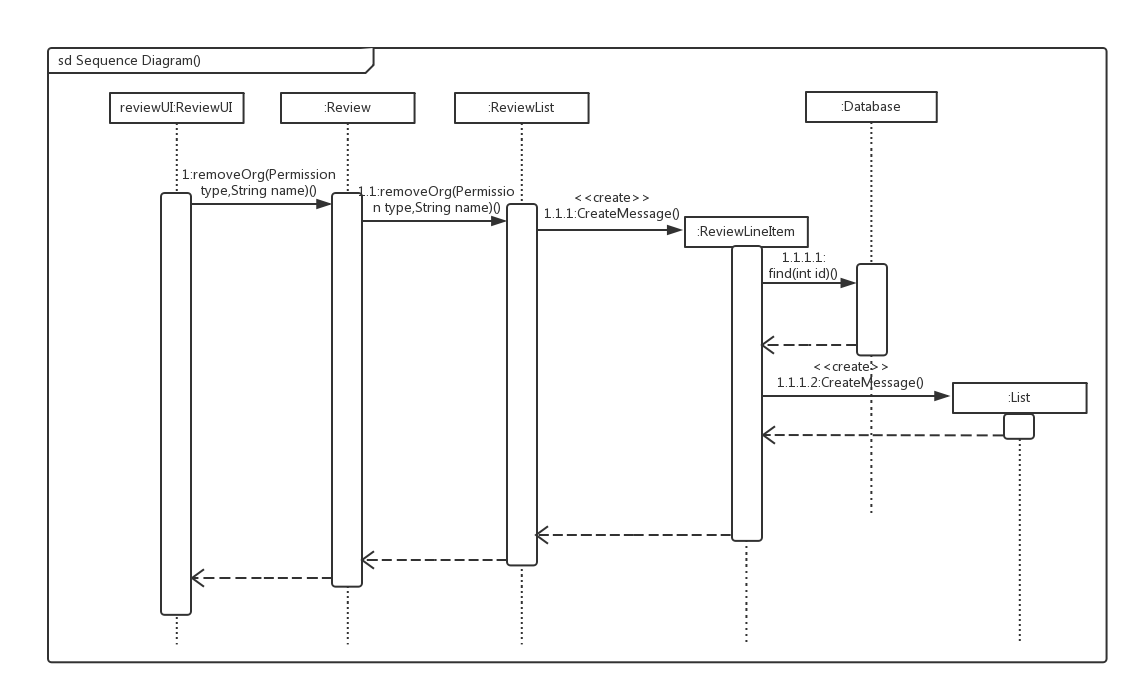
1.1.2业务逻辑层的动态模型

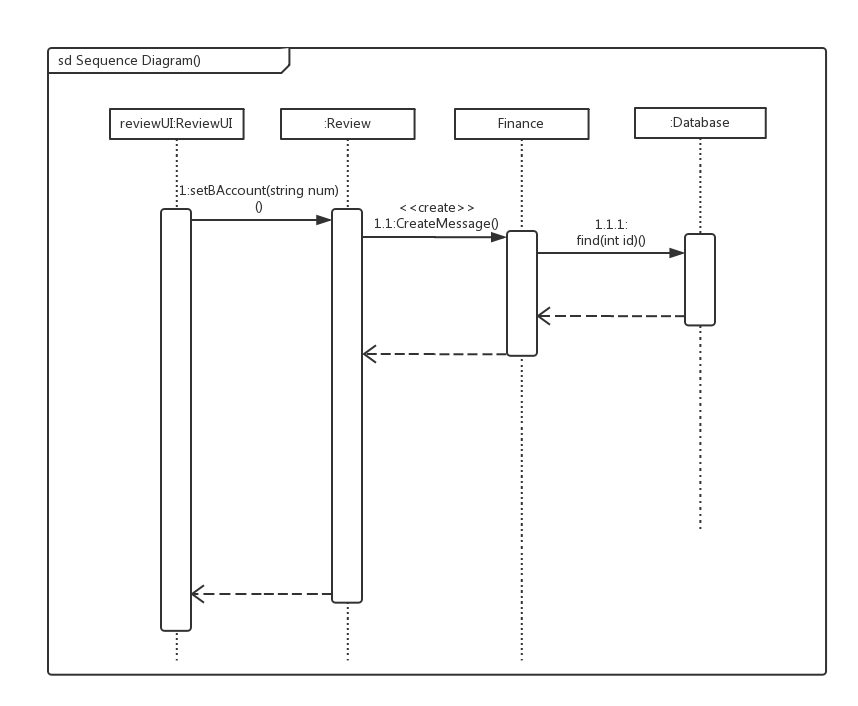
图8表明了物流快递系统中，当用户输入相关信息之后，审查业务逻辑处理的相关对象之间的协作。

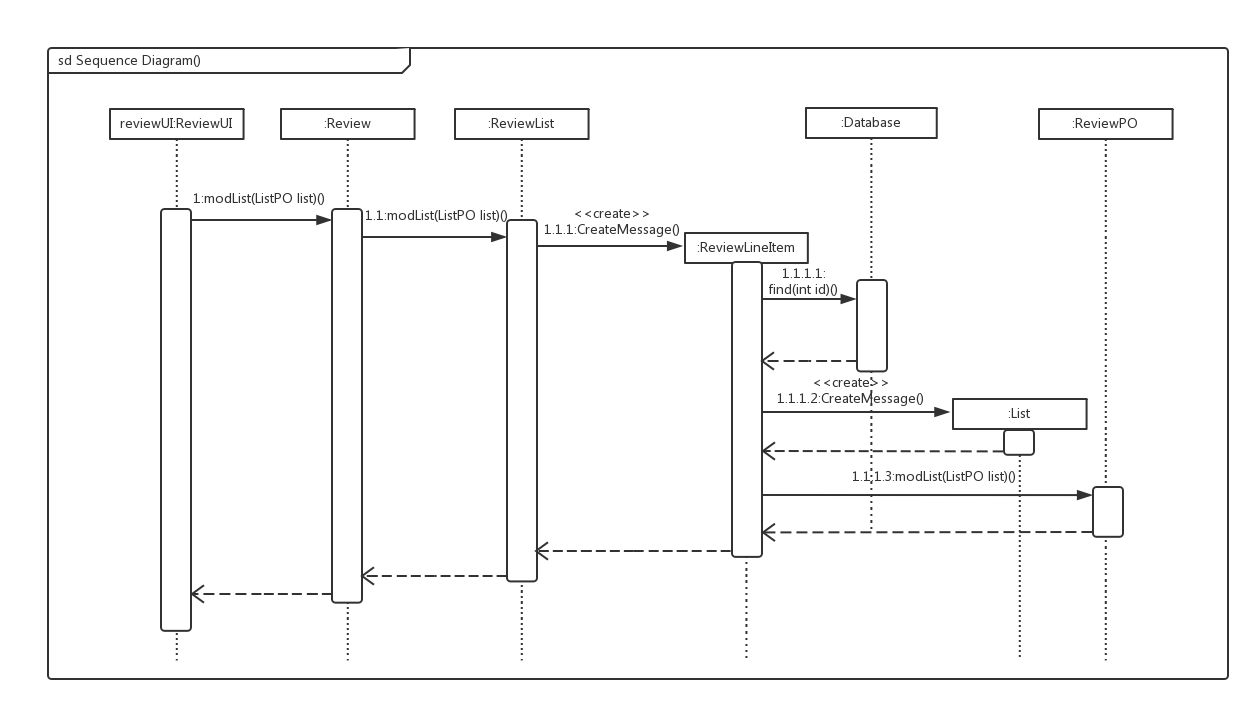
图8

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。

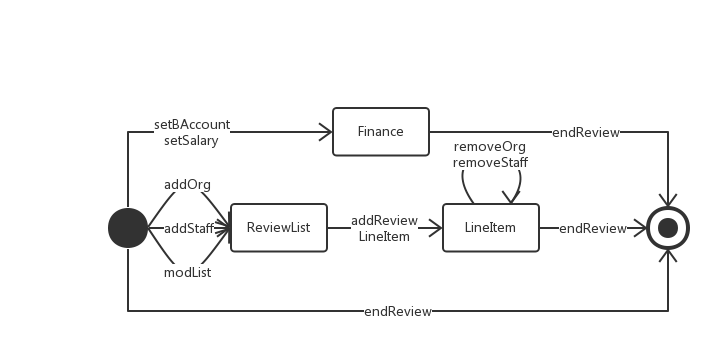


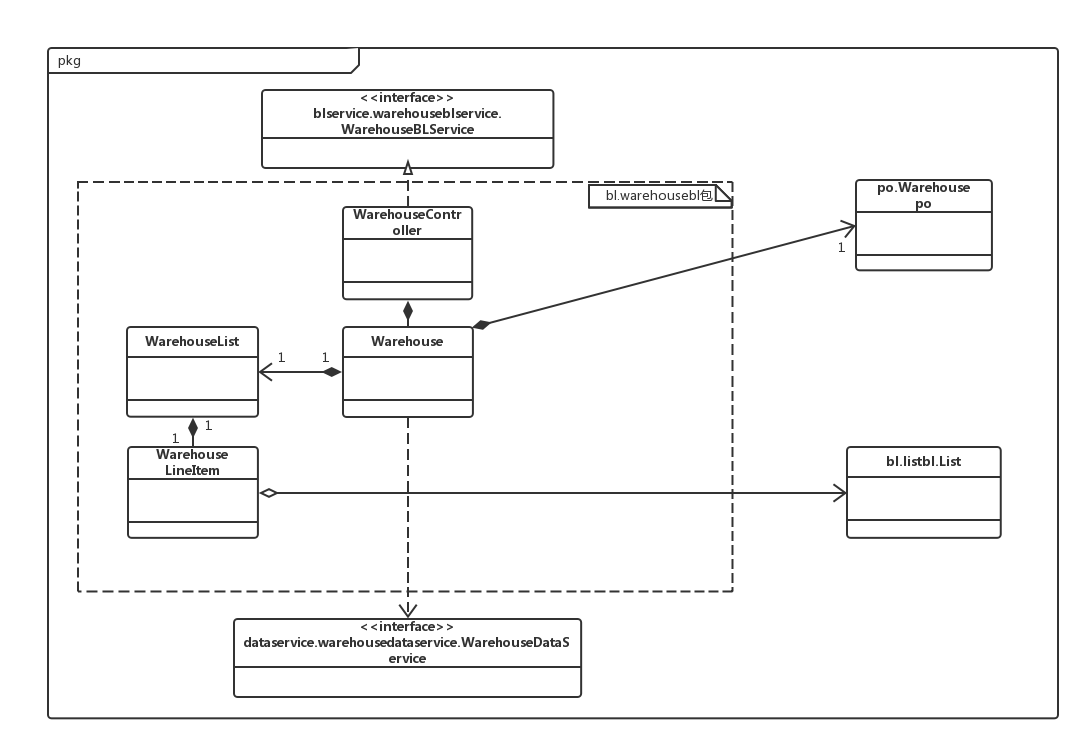
图9 Review对象状态图

**1.2 warehousebl模块**

1.2.1整体结构

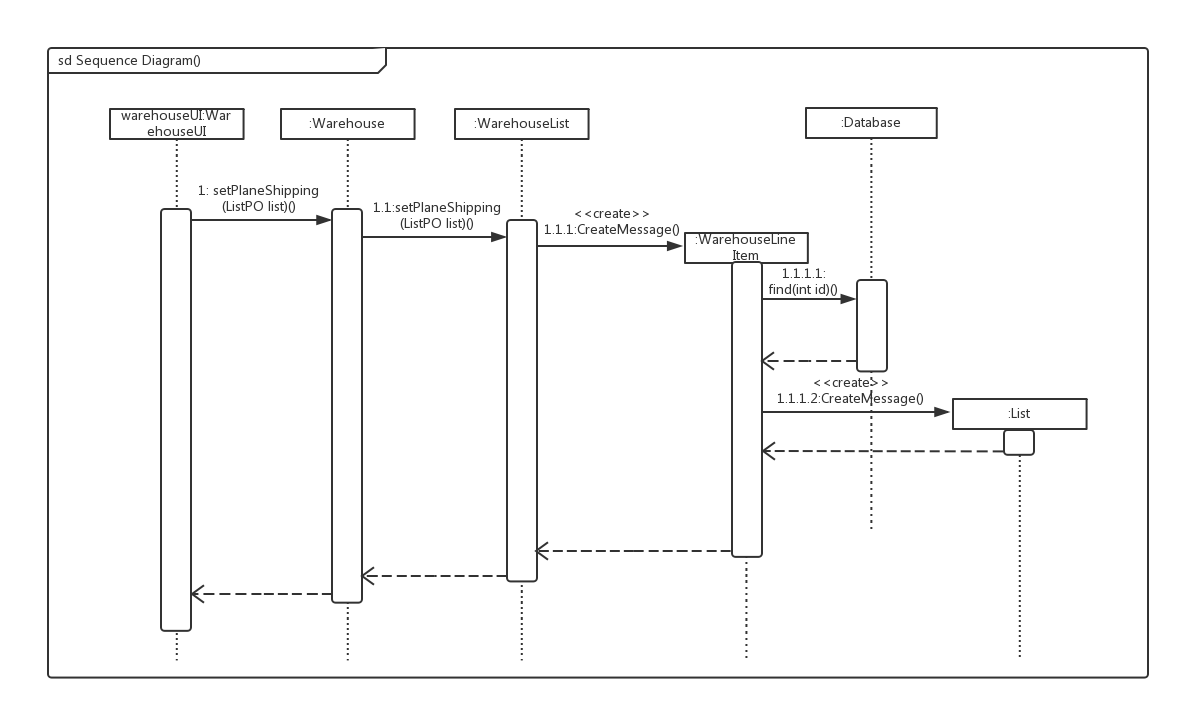
根据体系结构的设计,我们将系统分为展现层、业务逻辑层、数据层。每一层之间为了增加灵活性,我们会添加接口。比如展示层和业务逻辑层之间,我们添加blservice.warehouseblservice.WarehouseBLService接口。业务逻辑层和数据层之间添加 dataservice.warehousedataservice.WarehouseDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 WarehouseController,这样WarehouseController会将对中转中心的业务逻辑处理委托给Warehouse对象。WarehousePO是作为审查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。

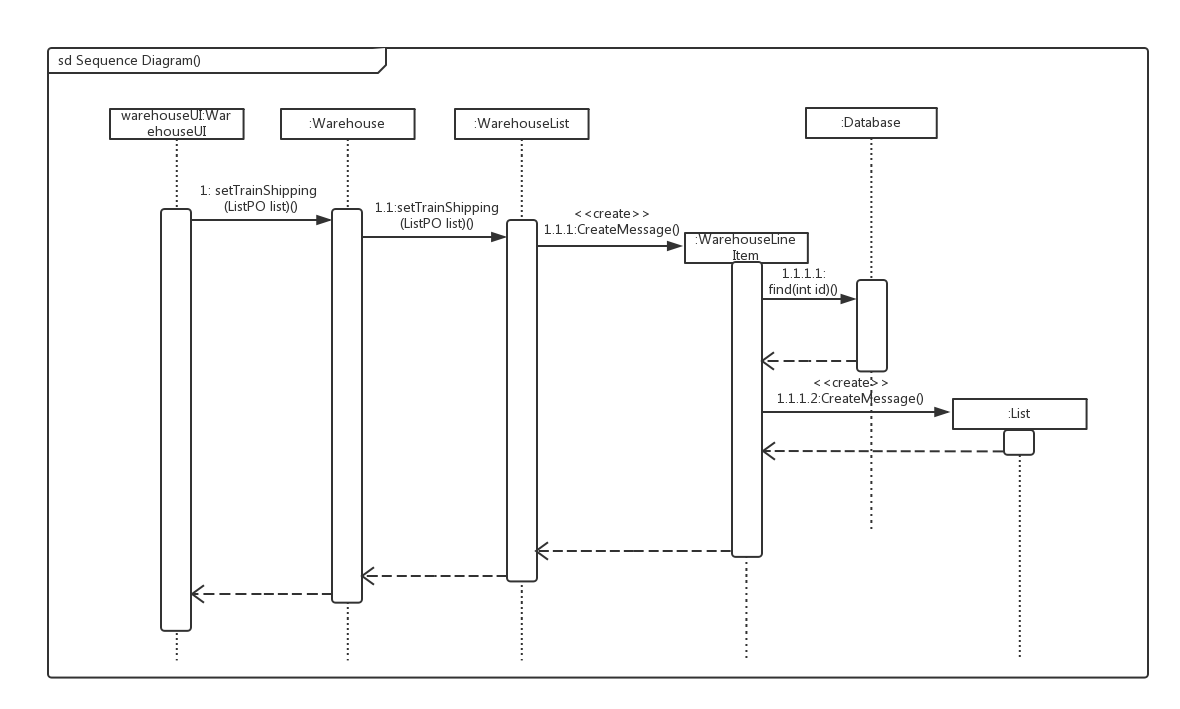
warehousebl模块的设计如图13所示。

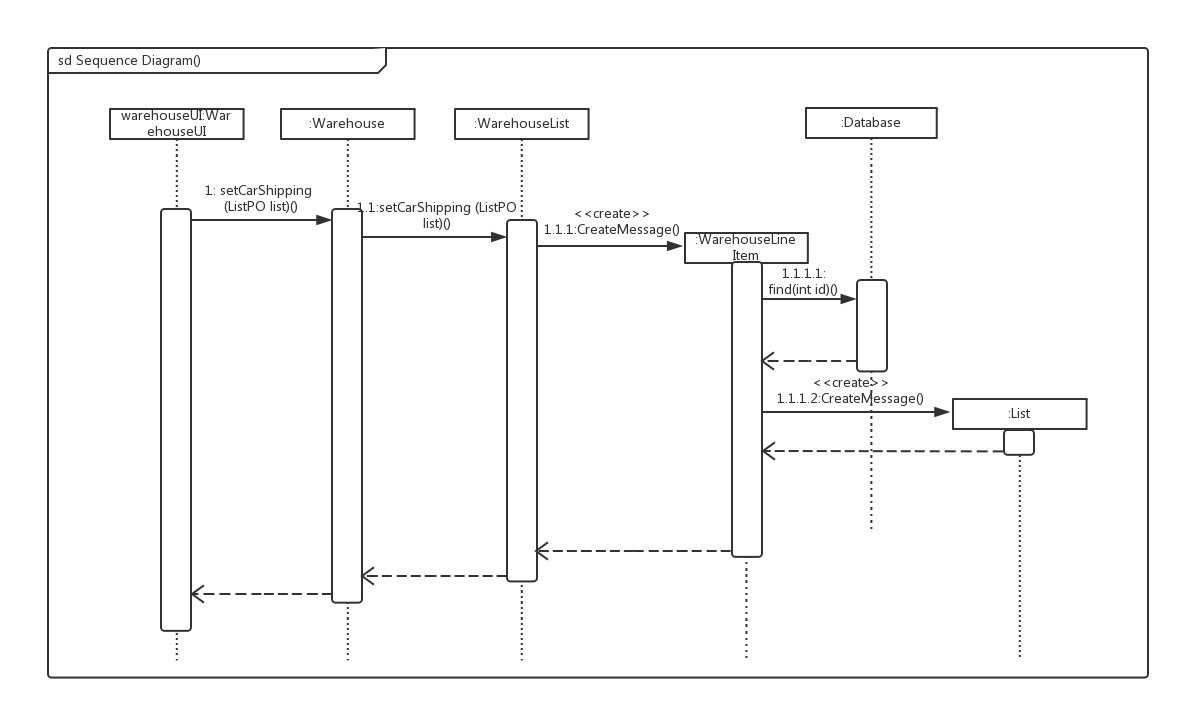
图13 warehousebl模块各个类的设计

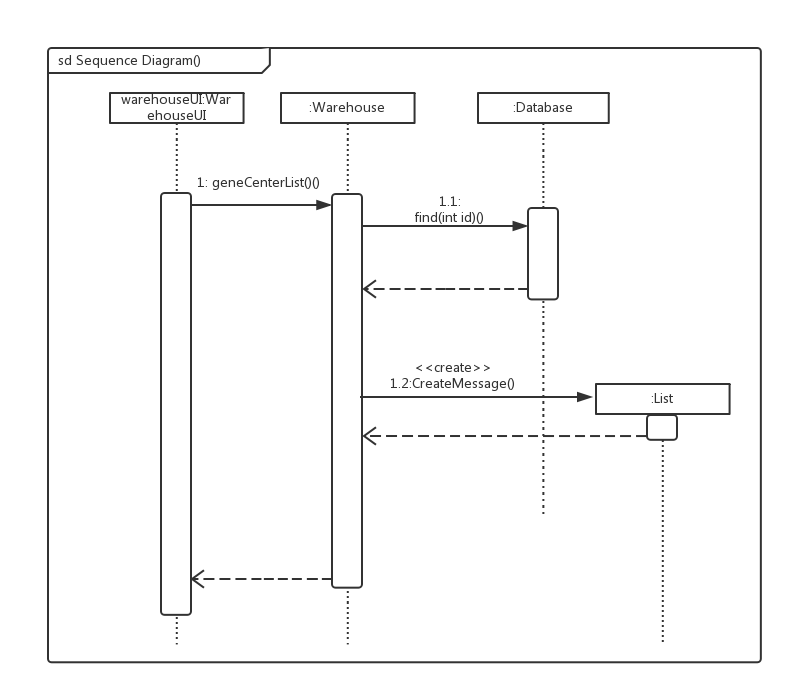
1.2.2业务逻辑层的动态模型

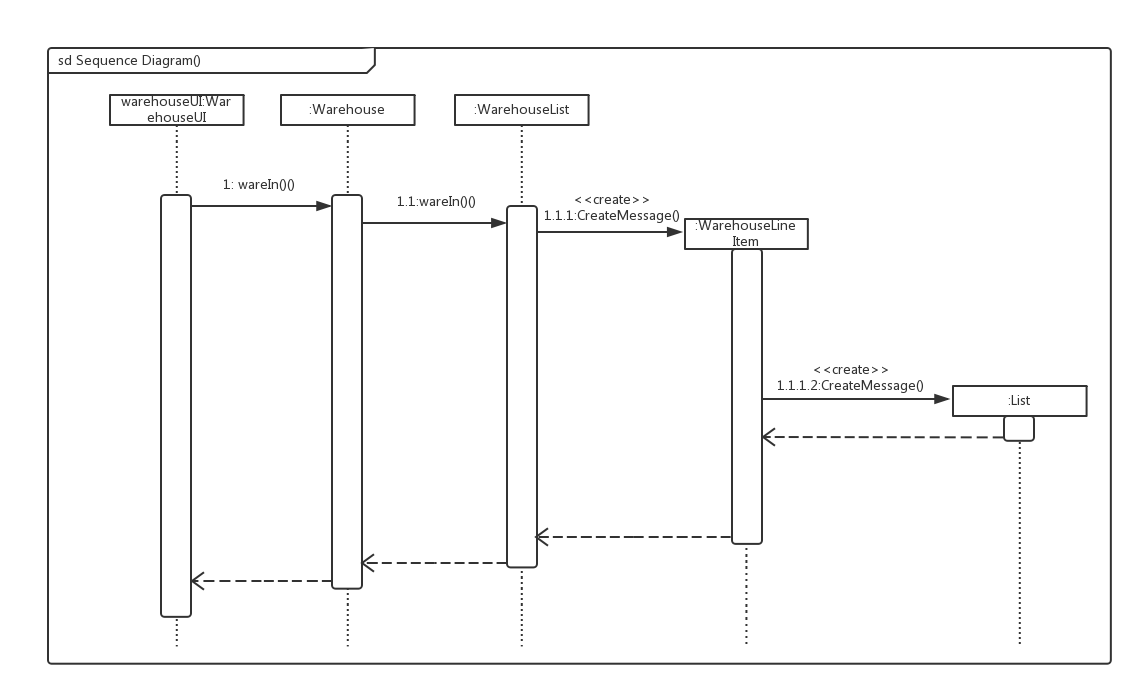
图14表明了物流快递系统中，当用户输入相关信息之后，中转中心业务逻辑处理的相关对象之间的协作。

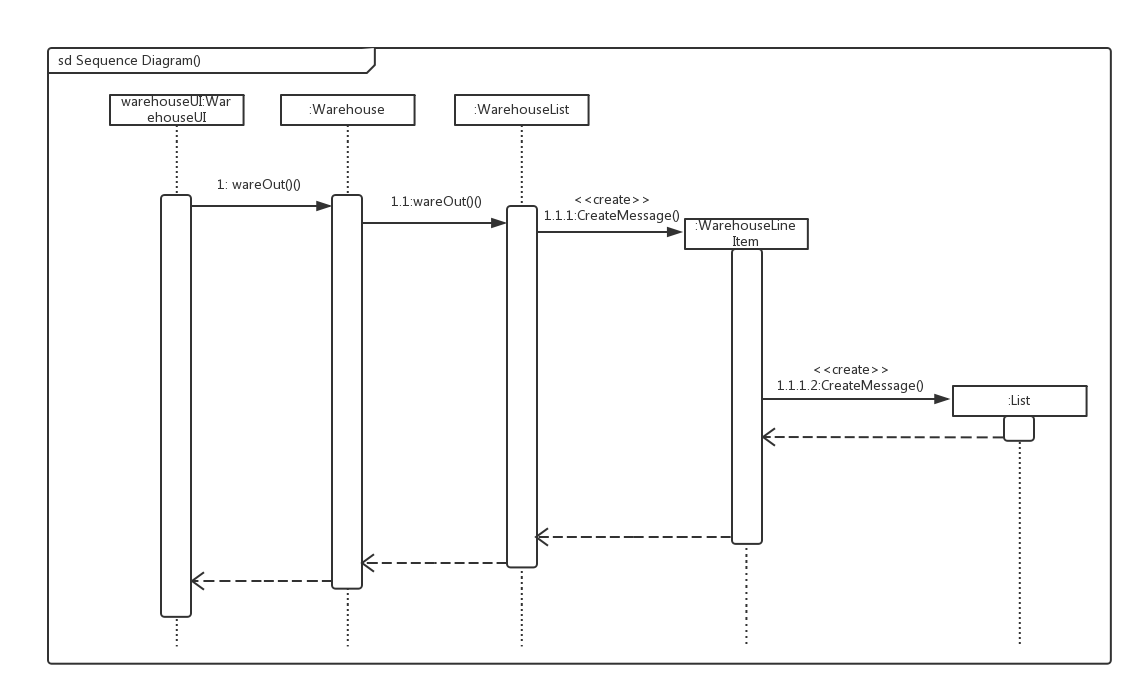
图14 

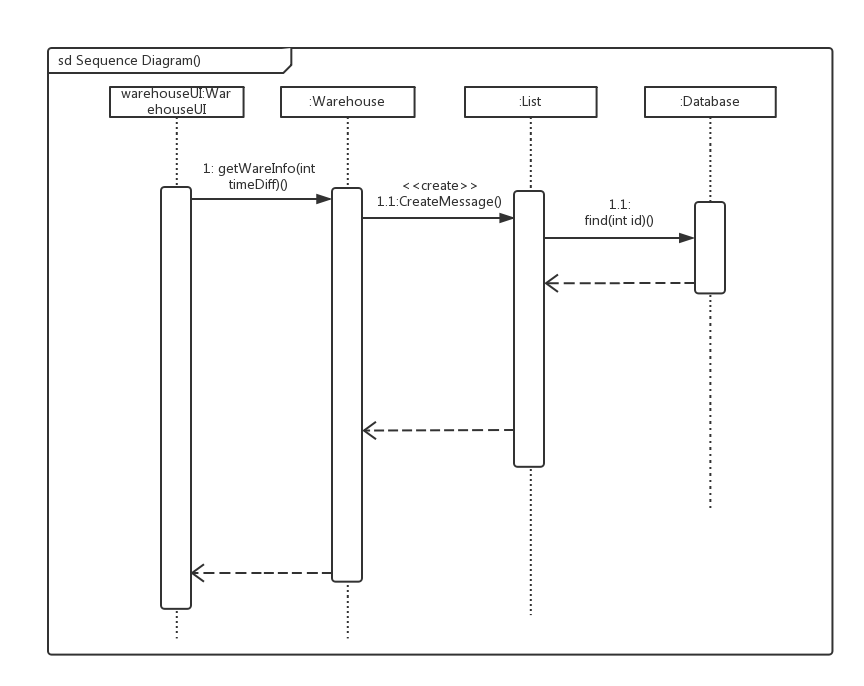
飞机装运管理的顺序图

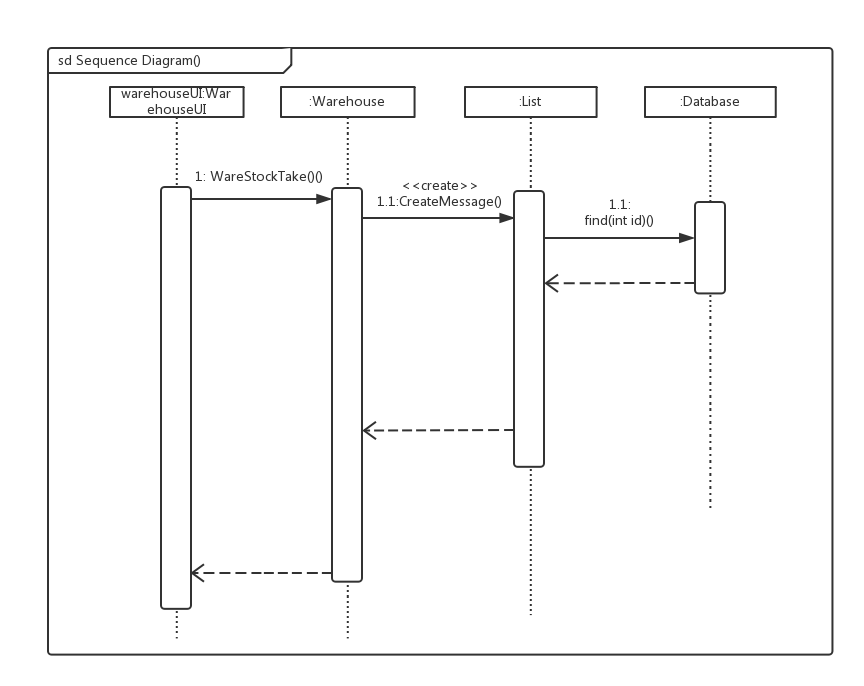
火车装运管理的顺序图

汽车装运管理的顺序图

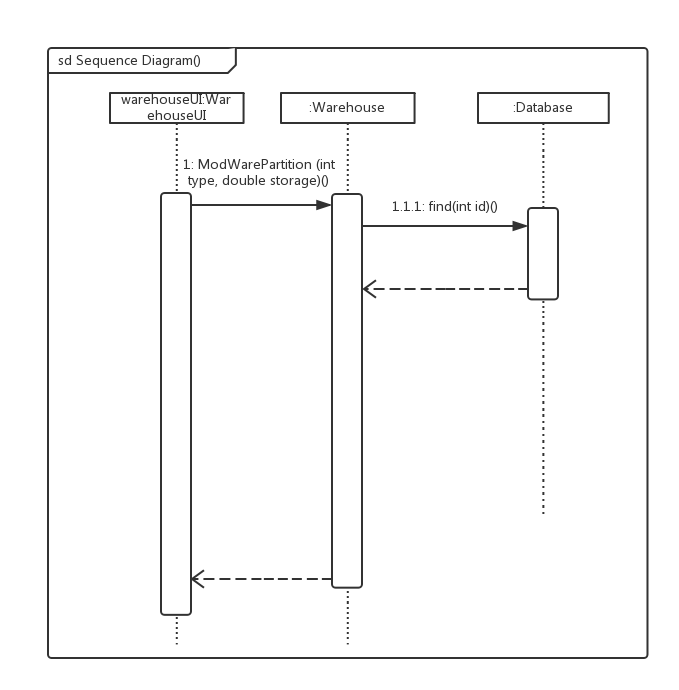
中转接收的顺序图

入库的顺序图

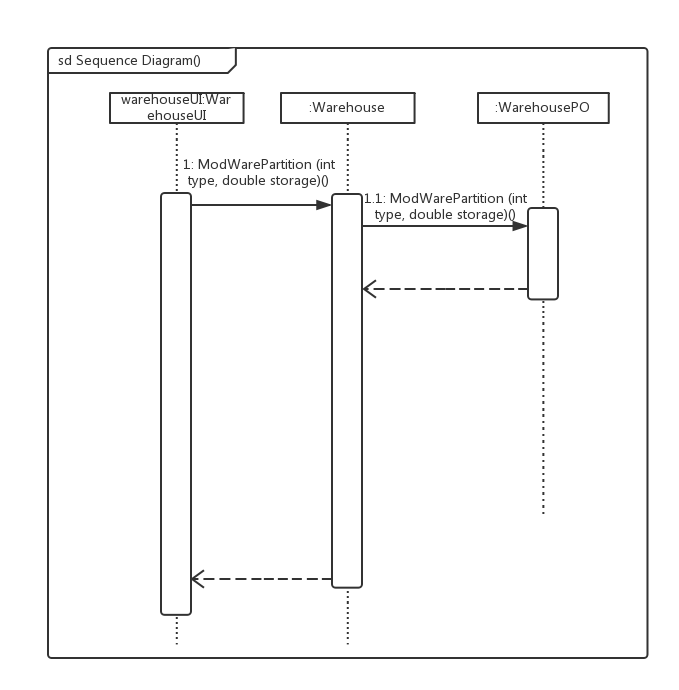
出库的顺序图

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。

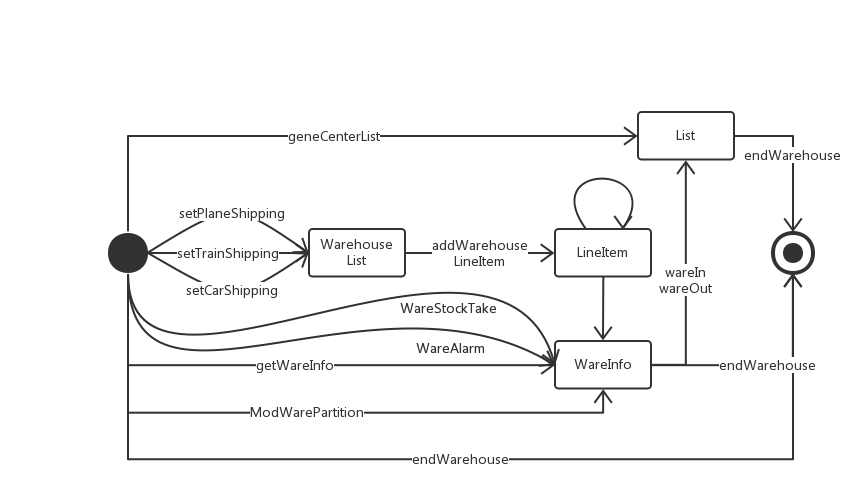


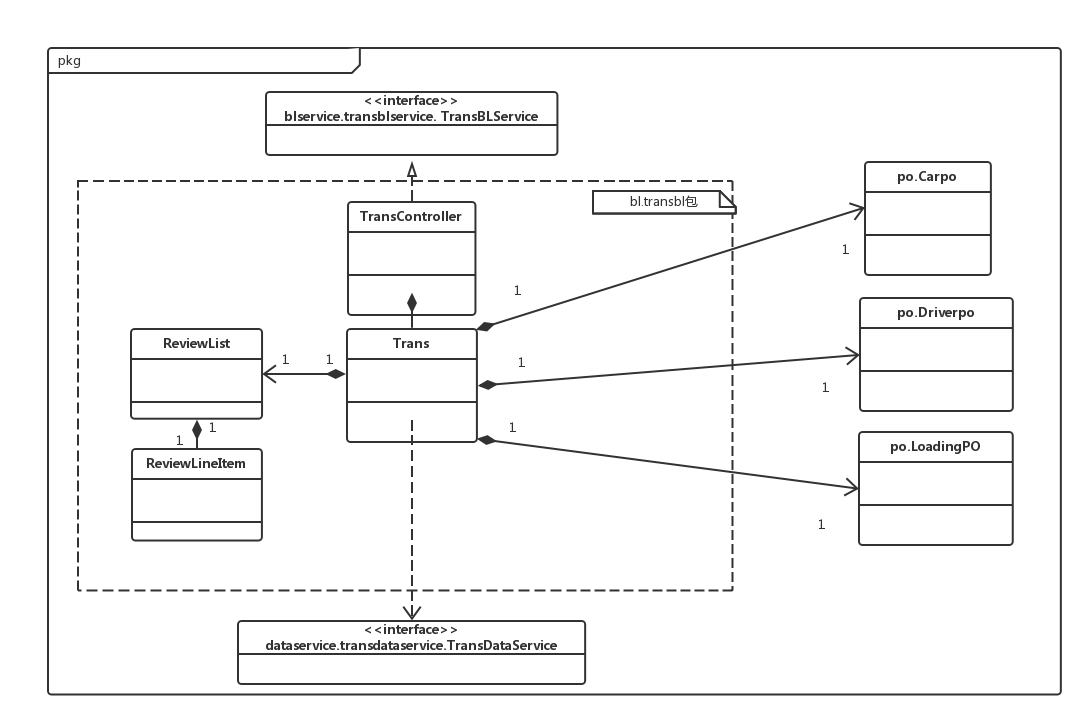
图15 Warehouse对象状态图

**1.3 transbl模板**

1.3.1整体结构

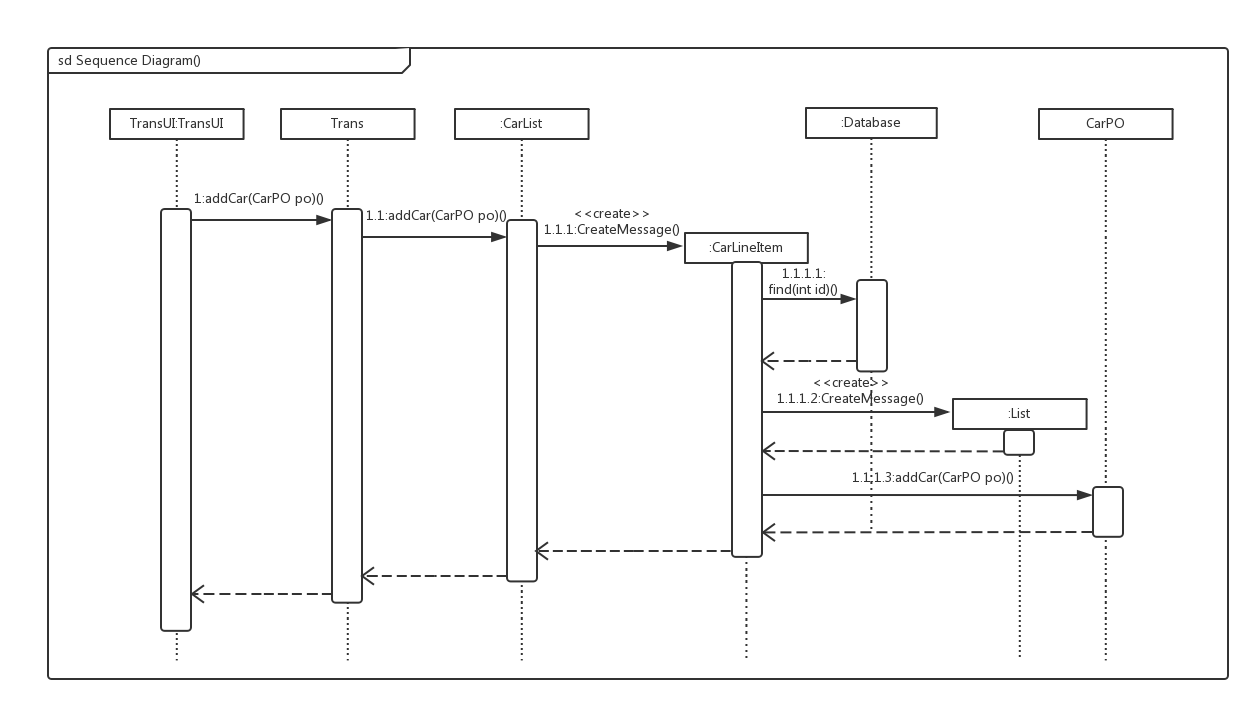
根据体系结构的设计,我们将系统分为展现层、业务逻辑层、数据层。每一层之间为了增加灵活性,我们会添加接口。比如展示层和业务逻辑层之间,我们添加blservice.transblservice. ReviewBLService接口。业务逻辑层和数据层之间添加 dataservice.reviewdataservice. transDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 TransController,这样TransController会将对运算的业务逻辑处理委托给Trans对象。CarPO,DriverPO,LoadingPO是作为运输任务的持久化对象被添加到设计模型中去的。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。

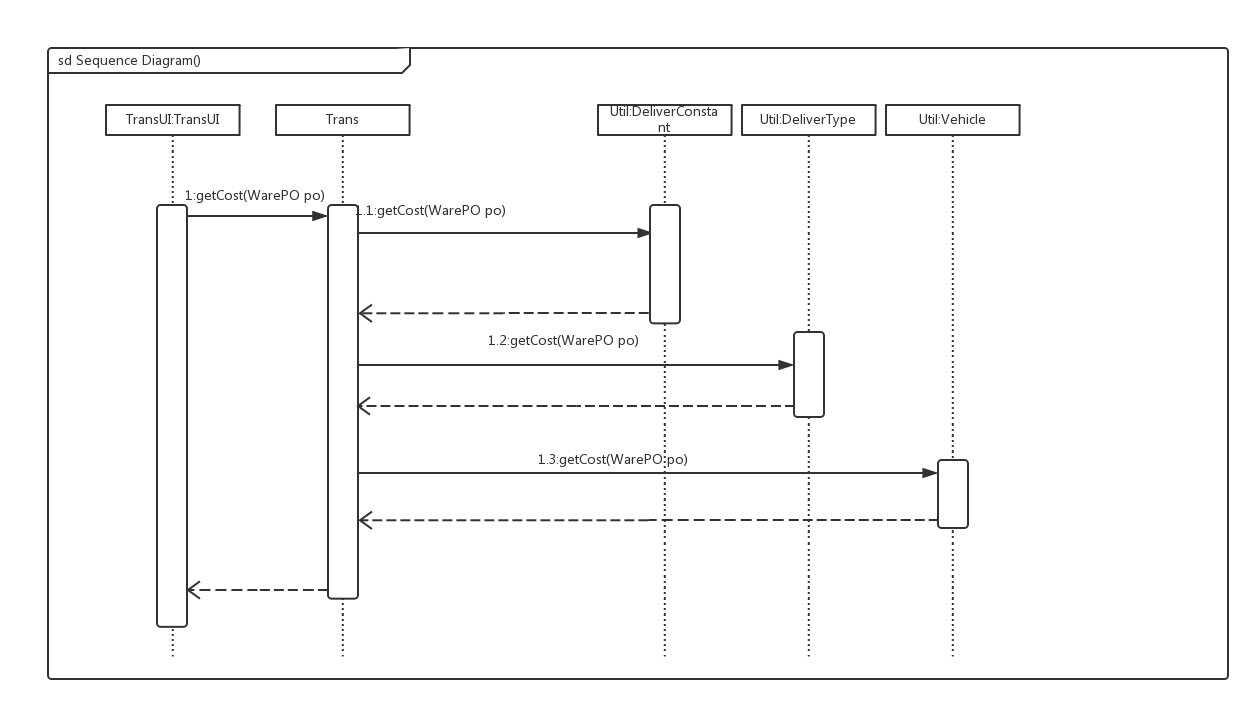


1.3.2业务逻辑层的动态模型

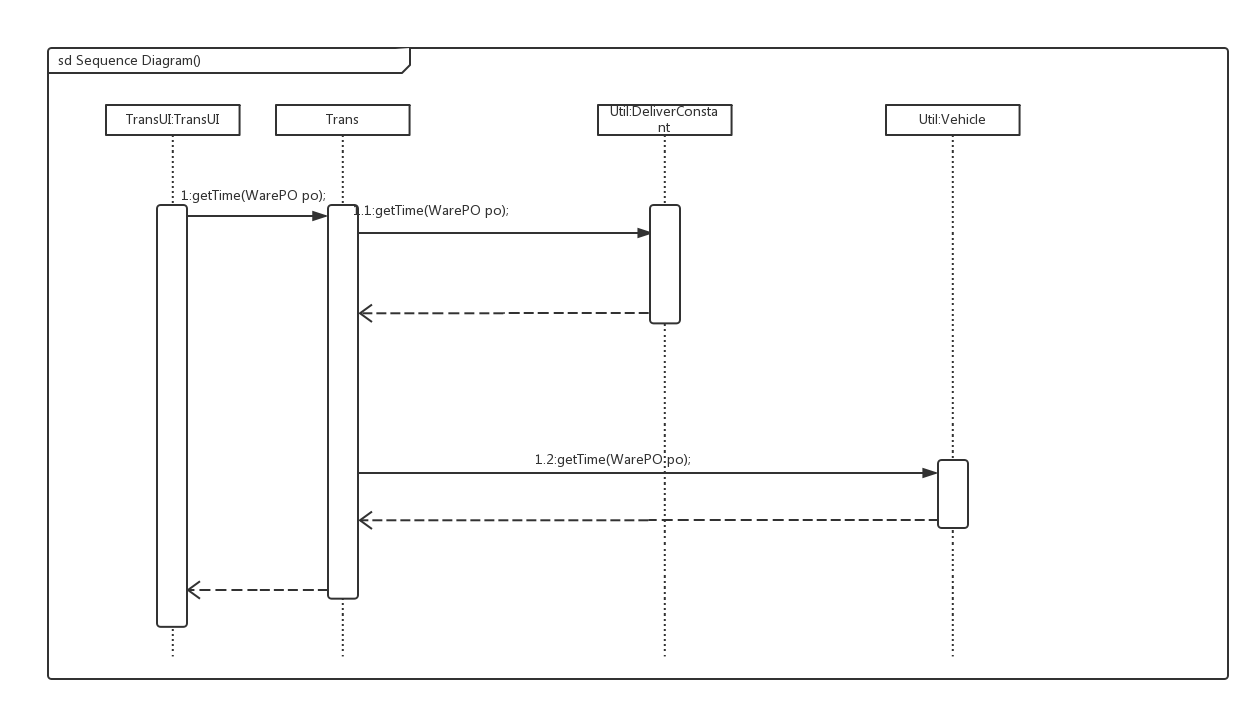
如图表明了物流快递系统中，当用户输入相关信息之后，运输逻辑处理的相关对象之间的协作。



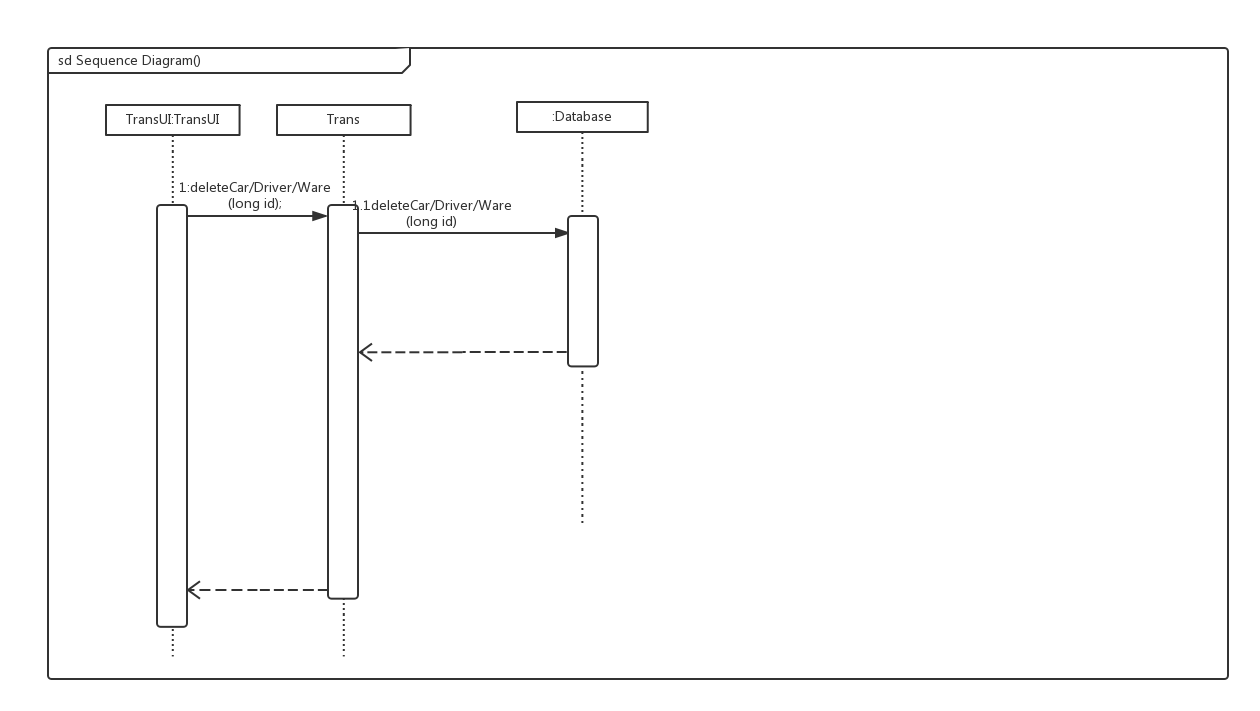
**图 添加车辆的顺序图**



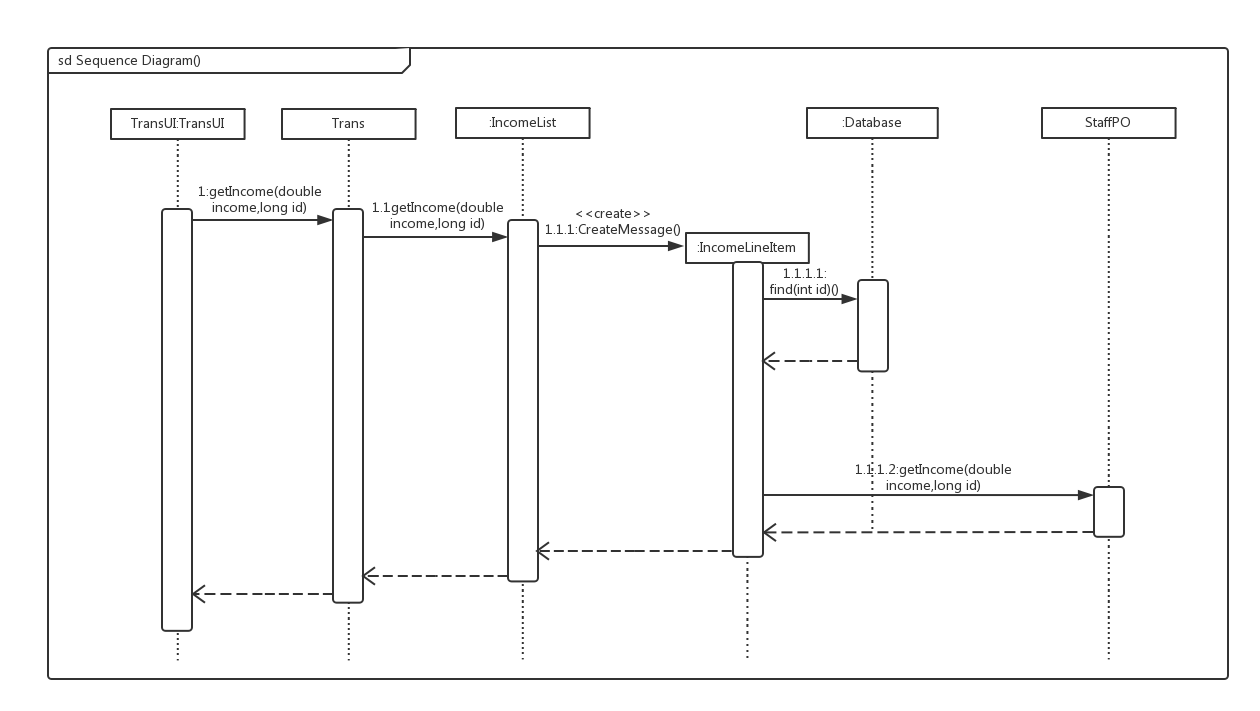
**图 得到报价的顺序图**



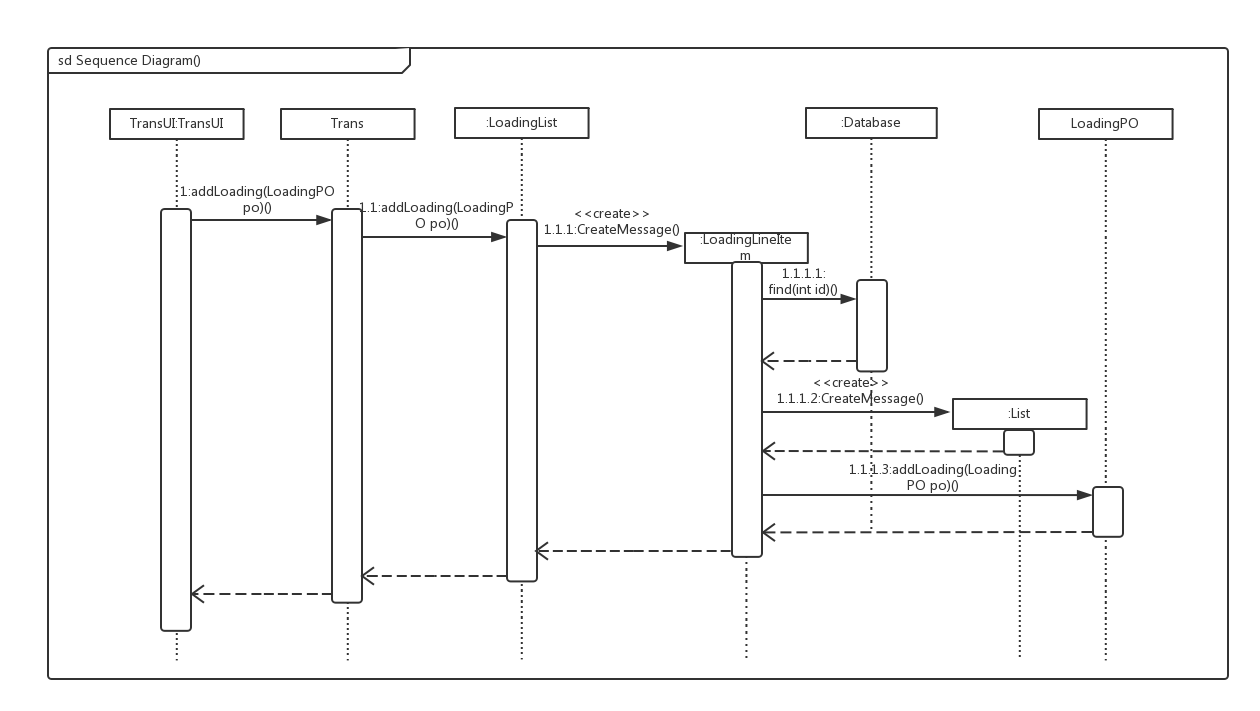
**图 得到预计时间的顺序图**



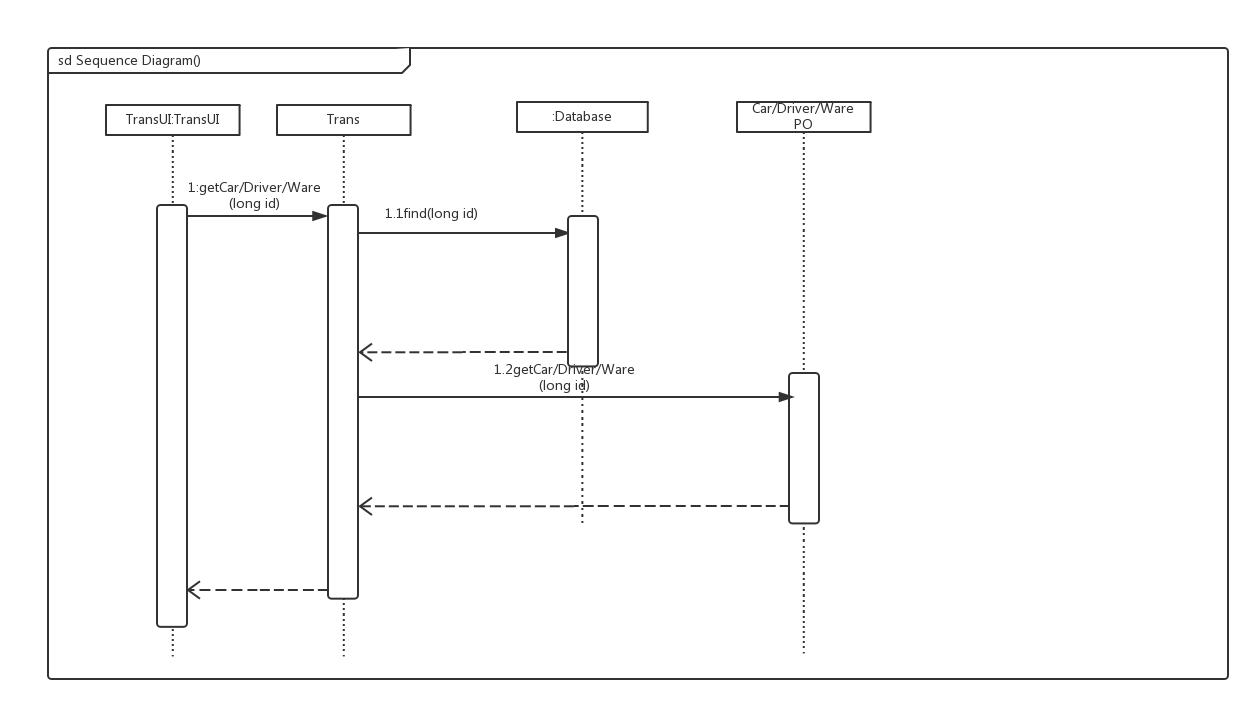
**图 删除车辆（司机，货物）的顺序图**



**图 收款单的顺序图**

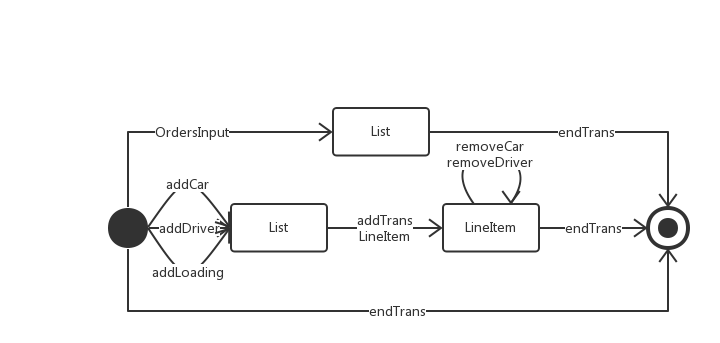


**图 添加装运信息的顺序图**



**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。

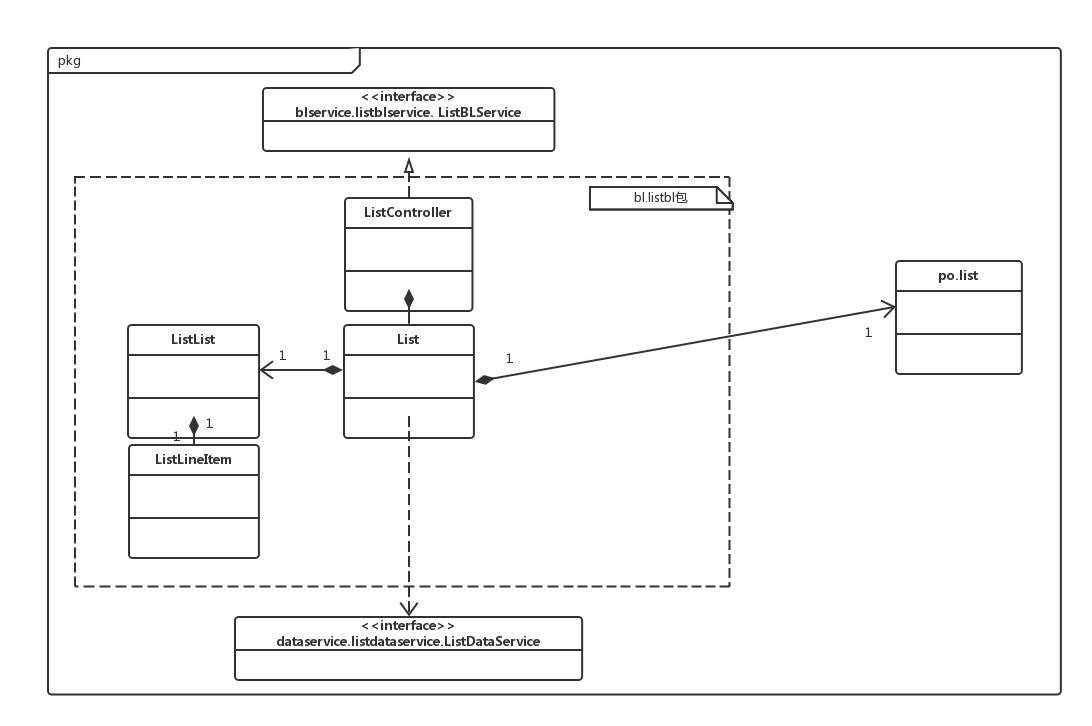


**1.4 listbl 模块**

1.4.1整体结构

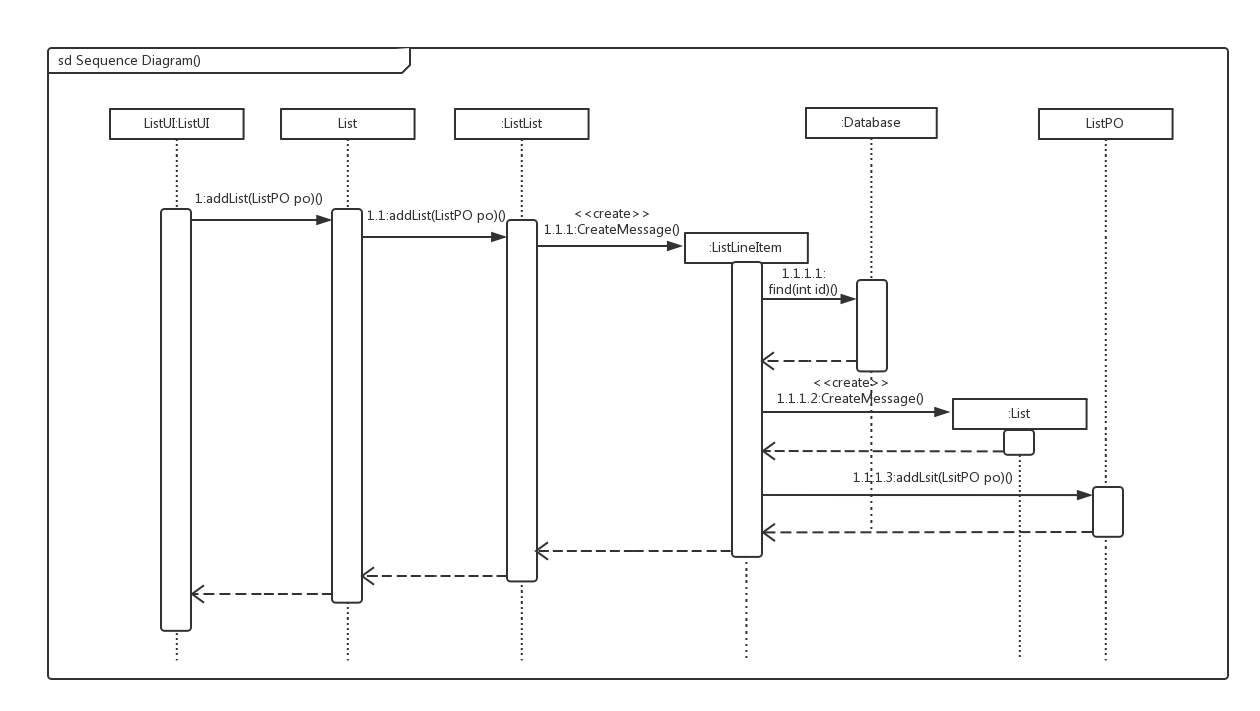
根据体系结构的设计,我们将系统分为展现层、业务逻辑层、数据层。每一层之间为了增加灵活性,我们会添加接口。比如展示层和业务逻辑层之间,我们添加blservice. listblservice. ListBLService接口。业务逻辑层和数据层之间添加 dataservice.listdataservice. ListDataService接口。为了隔离业务逻辑职责和逻辑控制职责,我们增加了 ListController,这样ListController会将对单据的业务逻辑处理委托给List对象。listPO是作为单据记录的持久化对象被添加到设计模型中去的。而ListList和ListLineItem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。

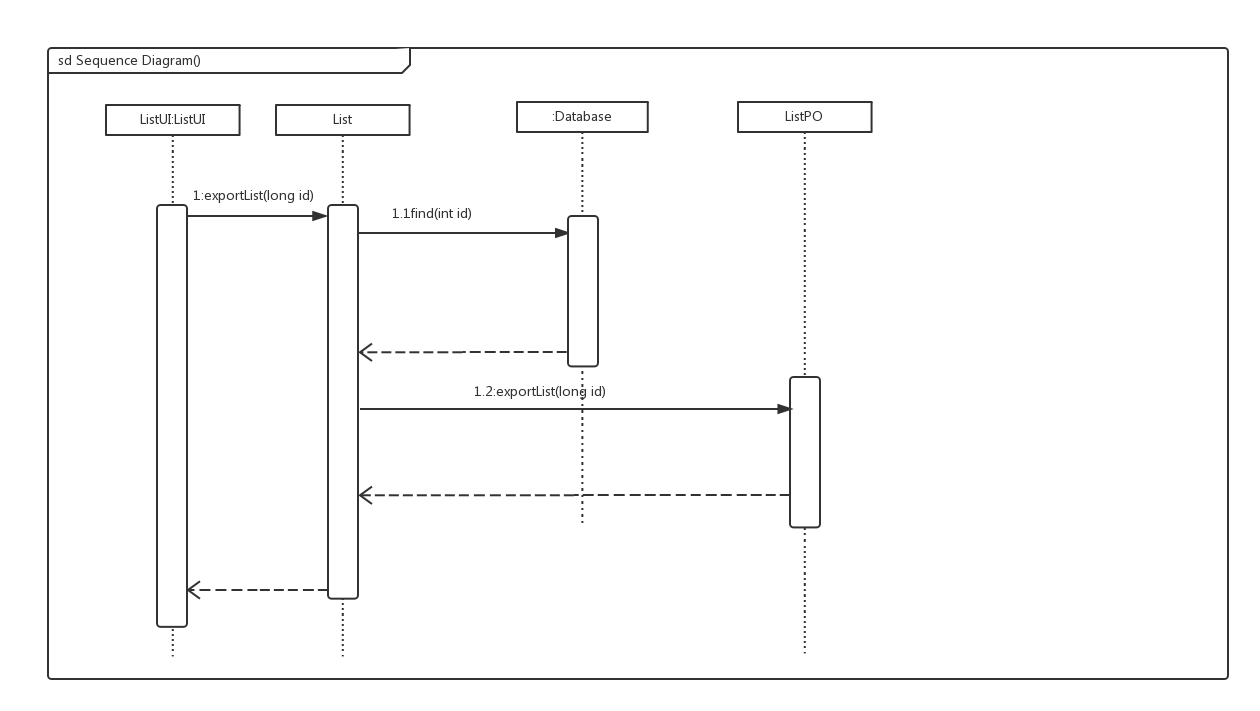


1.4.2业务逻辑层的动态模型

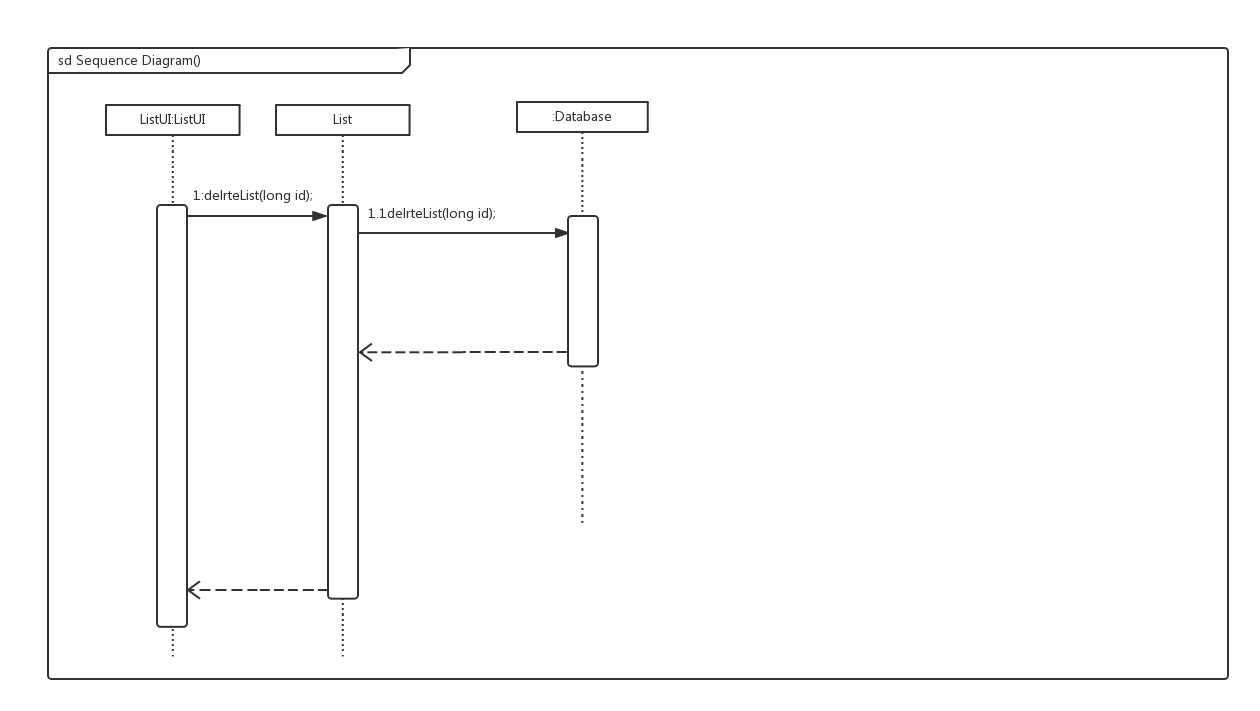
如图表明了物流快递系统中，当用户输入相关信息之后，单据业务逻辑处理的相关对象之间的协作。



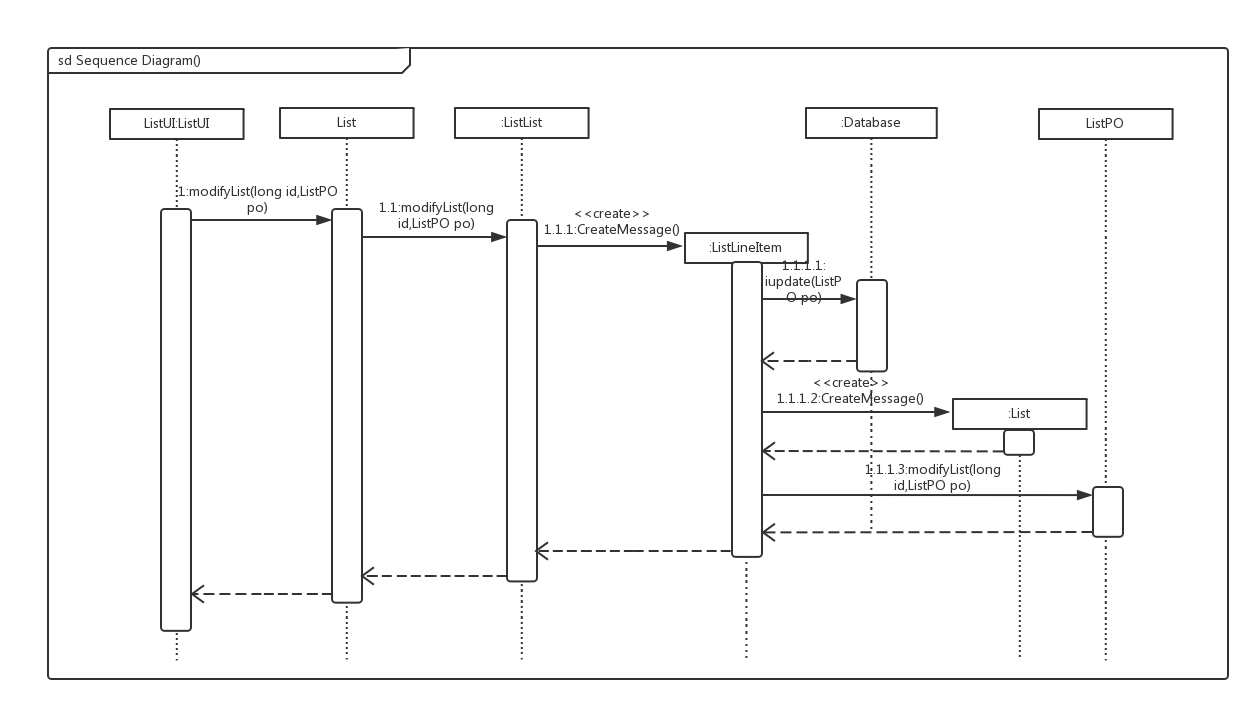
**图 添加单据顺序图**



**图 导出单据顺序图**

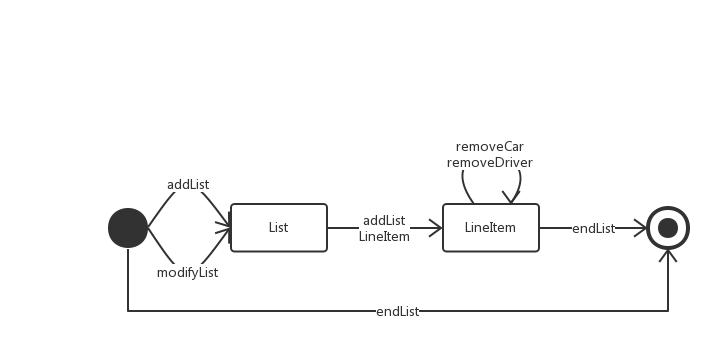


**图 删除单据顺序图**



**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。

****

**1.5 inquirebl模块**

1.5.1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。

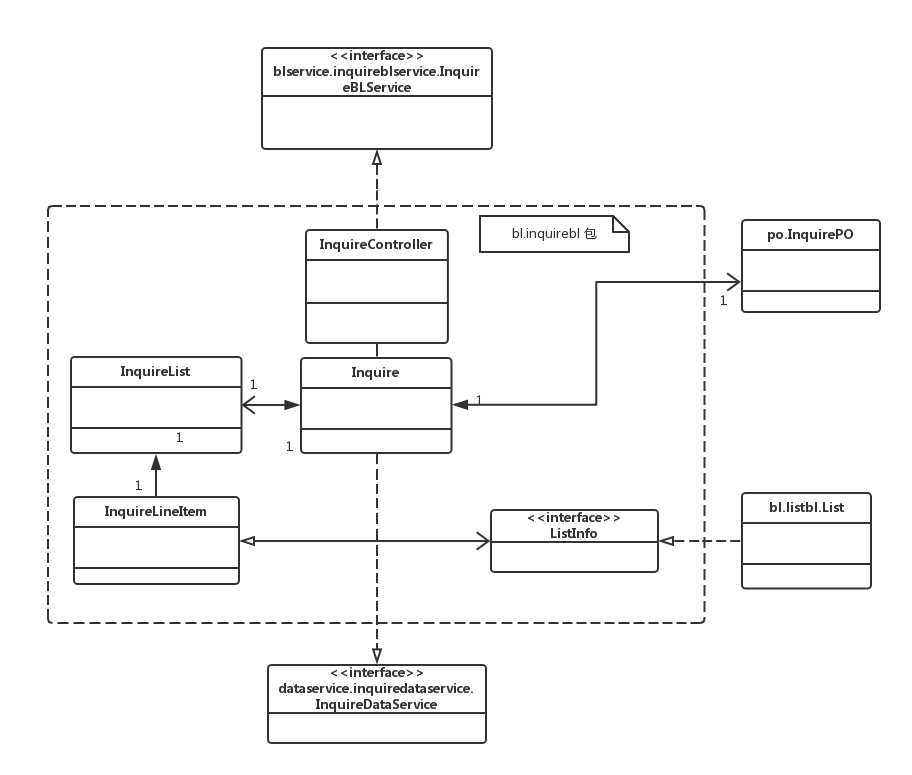


图1 inquirebl模块各个类的设计

1.5.2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。

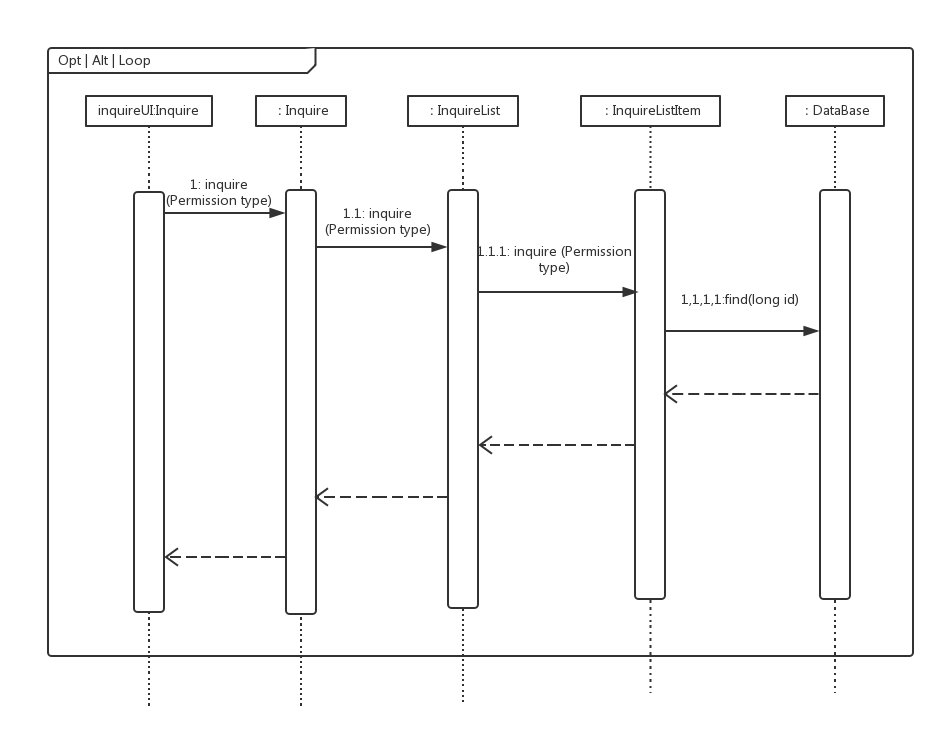


图2 查询快件的流程图

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。

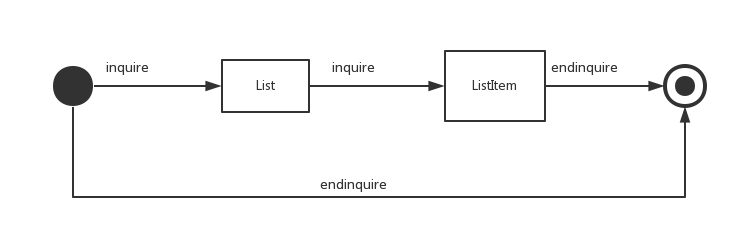


图3 Inquire对象状态图

**1.6 financebl模块**

1.6.1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。

financebl模块的设计如图1所示。

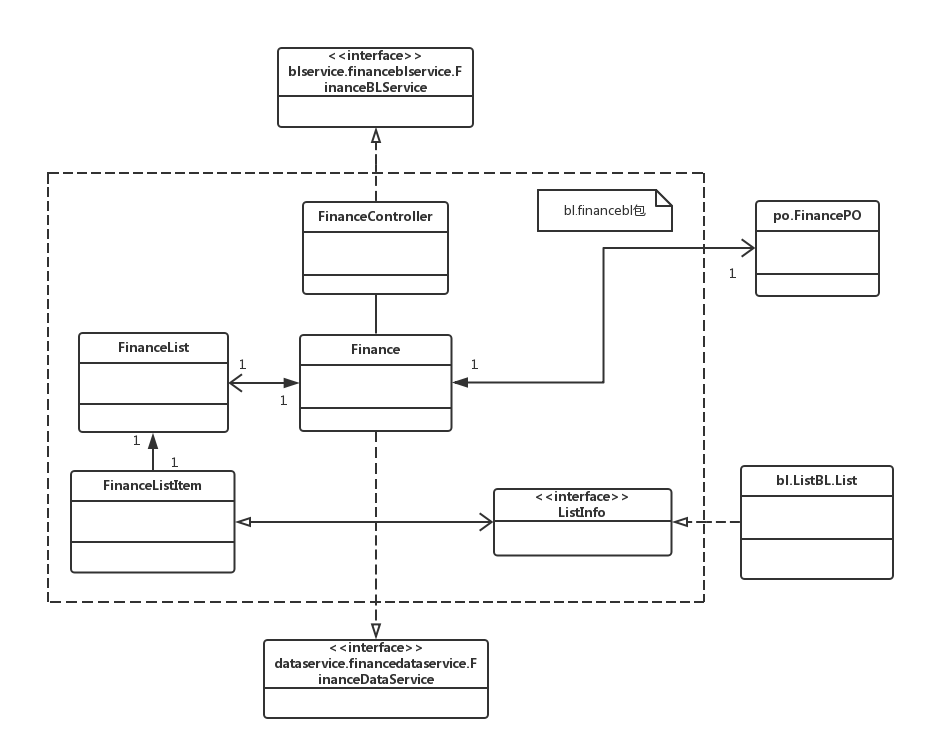
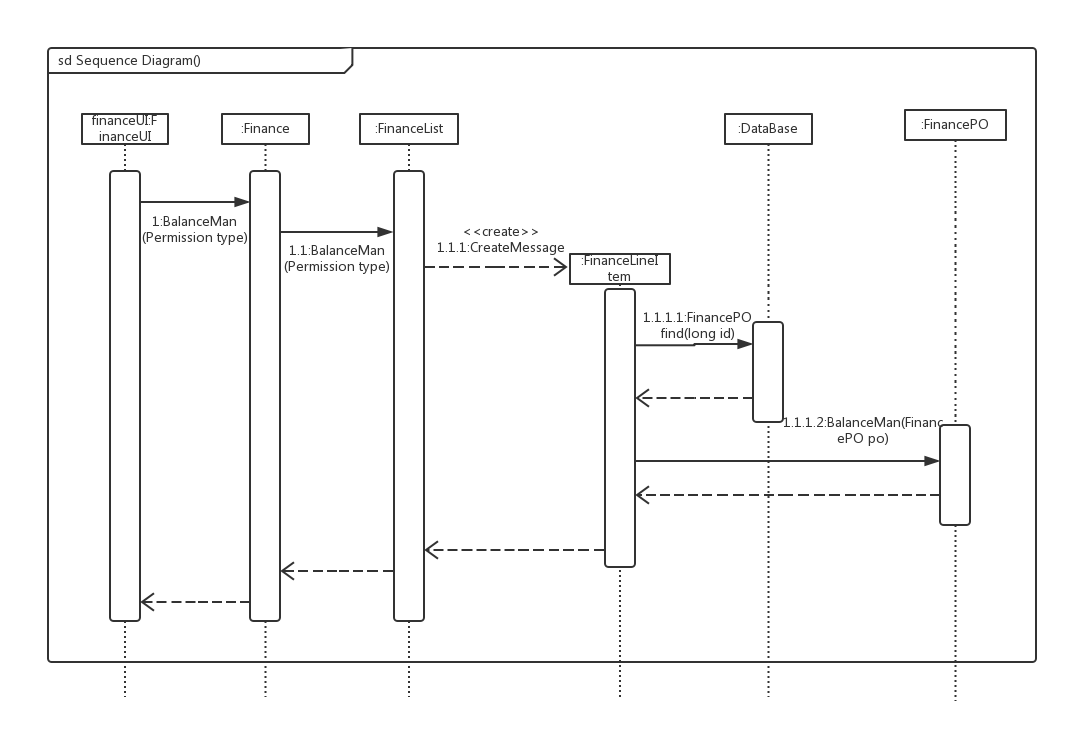


图1 financebl模块各个类的设计

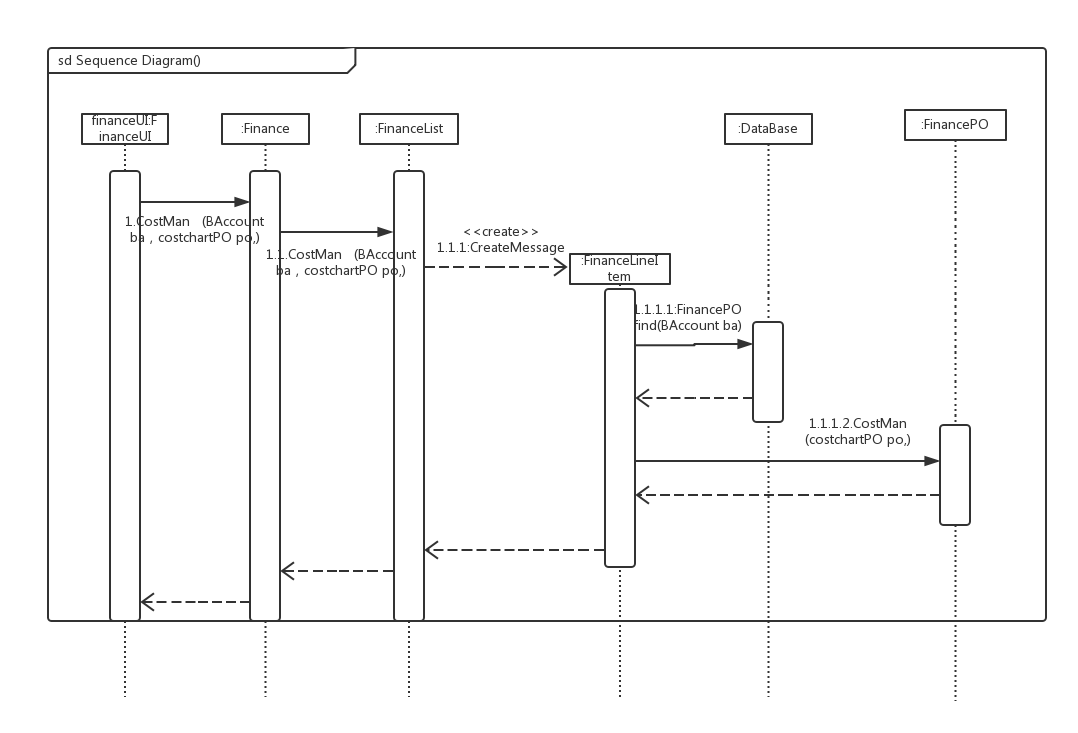
1.6.2业务逻辑层的动态模型

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。

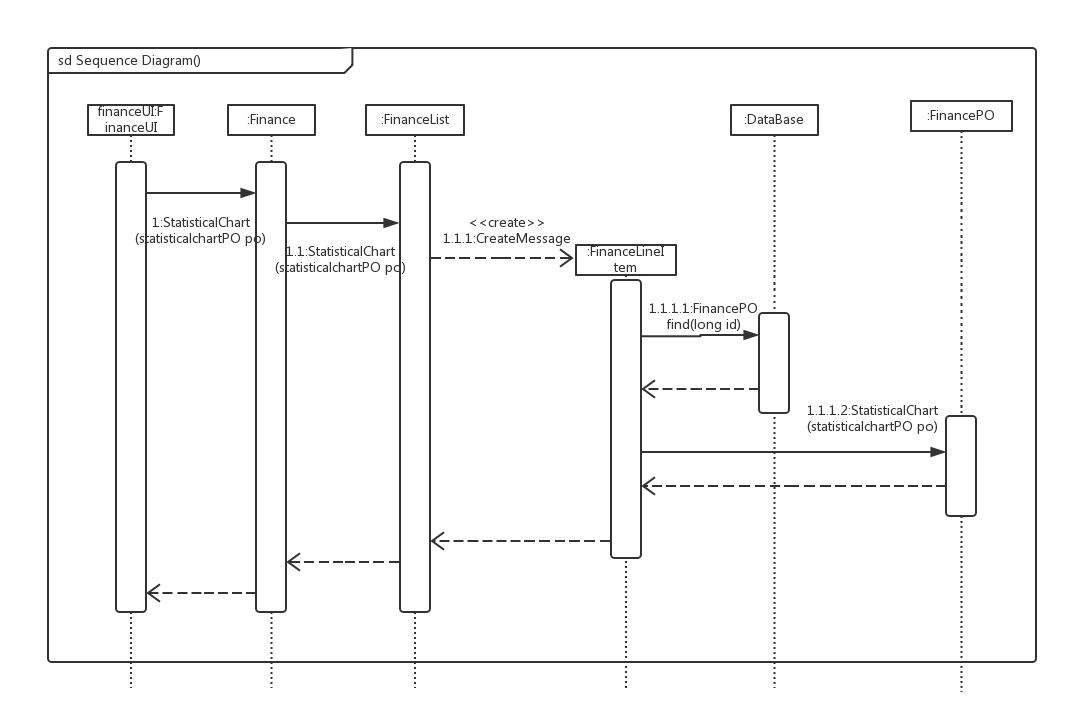
图2



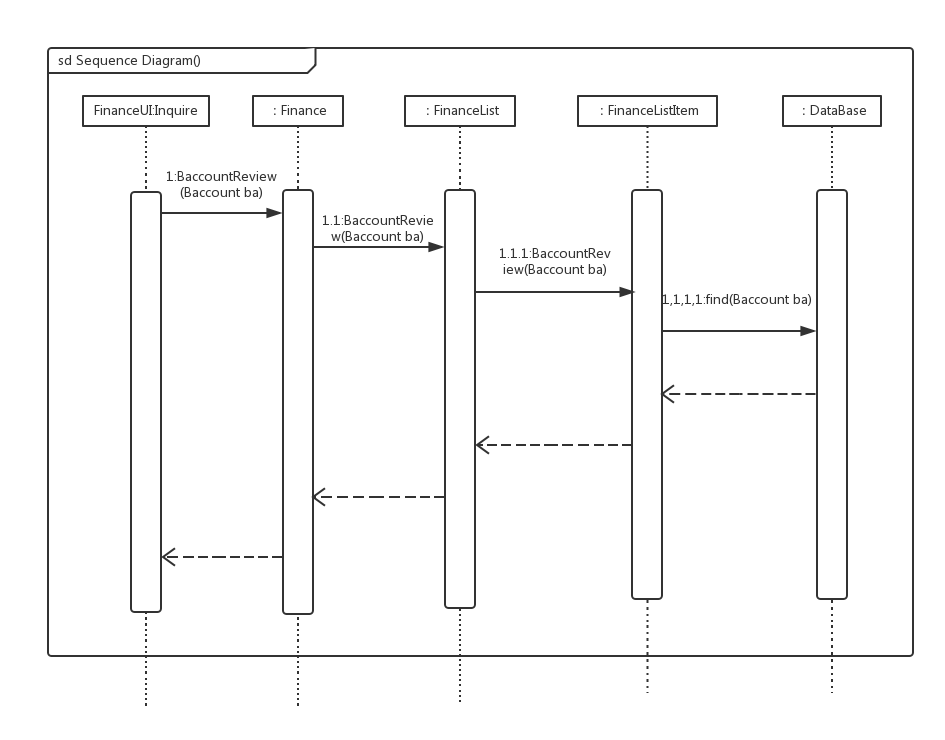
结算管理的流程图



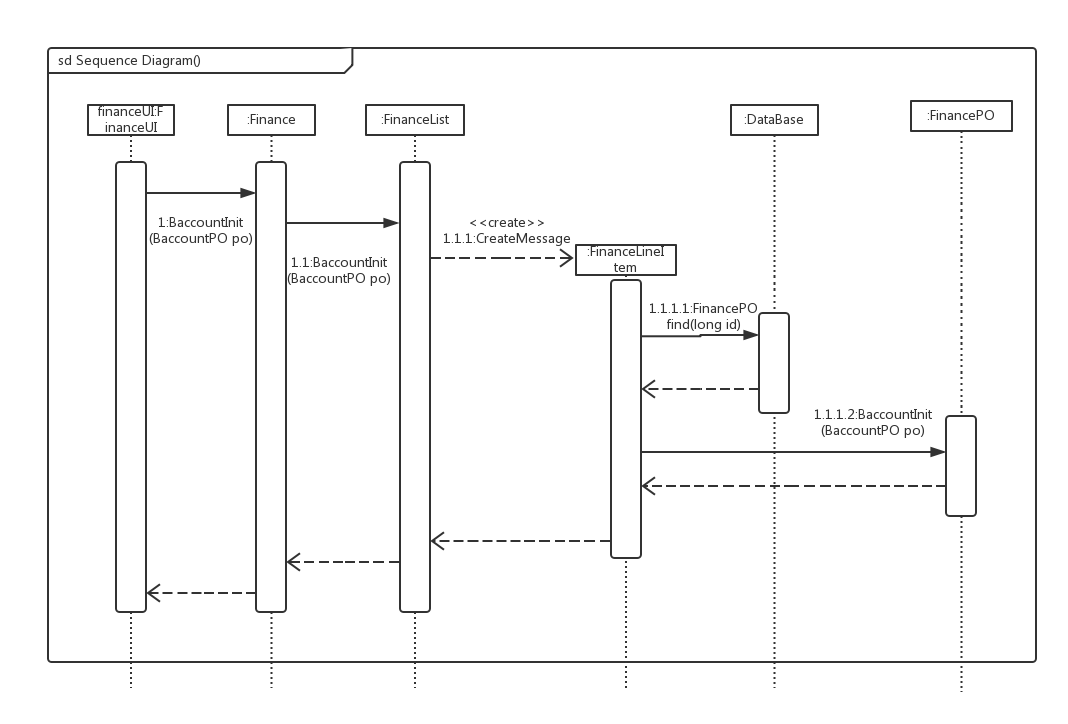
成本管理的流程图



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Finance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，Finance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

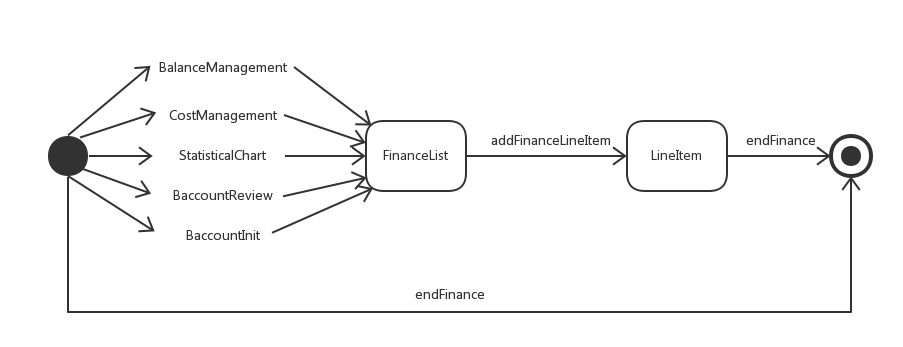


图3 Finance对象状态图

**1.7 loginbl模块**

1.7.1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。

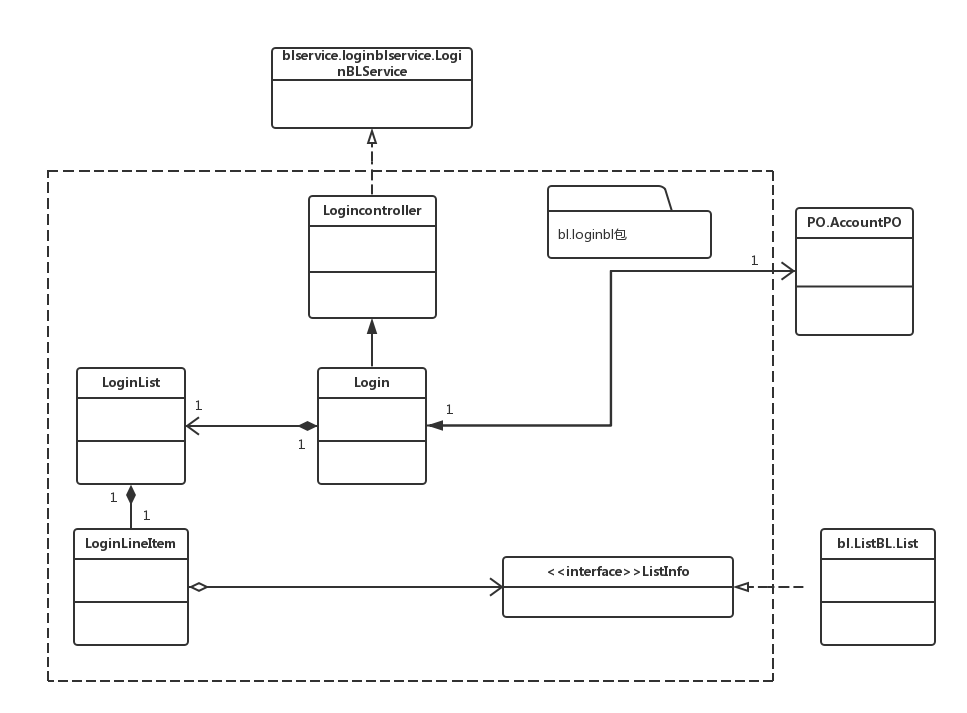


图1 loginbl模块各个类的设计

1.7.2业务逻辑层的动态模型

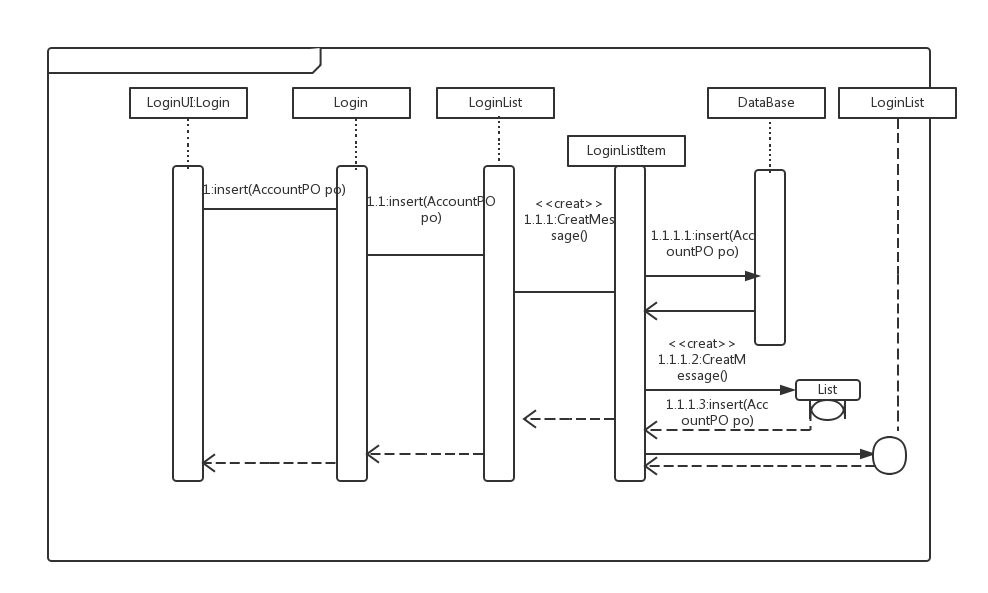
图2表明了物流快递系统中，当用户输入相关信息之后，审查业务逻辑处理的相关对象之间的协作。

图2 添加职员的顺序图

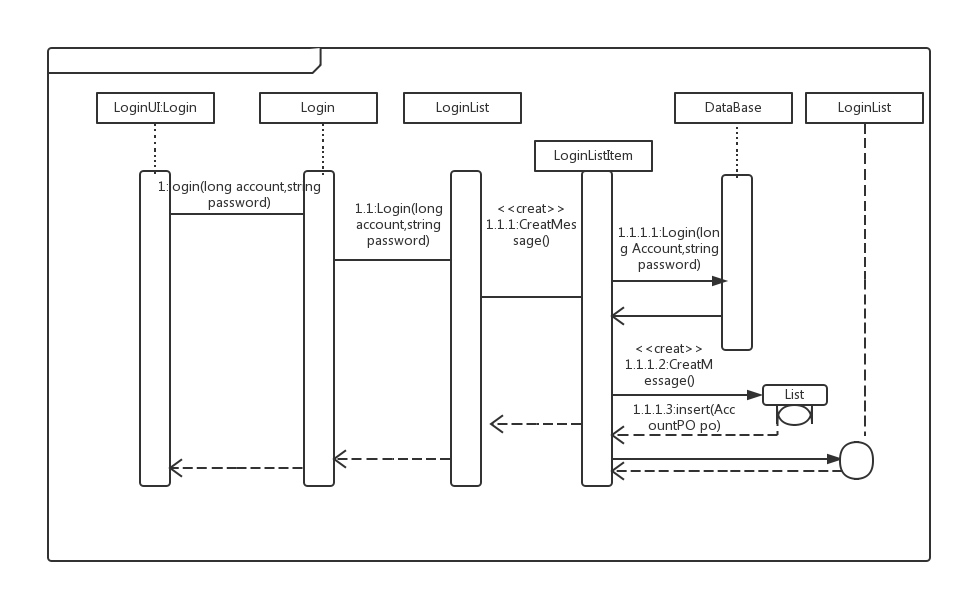


图3.登录的顺序图

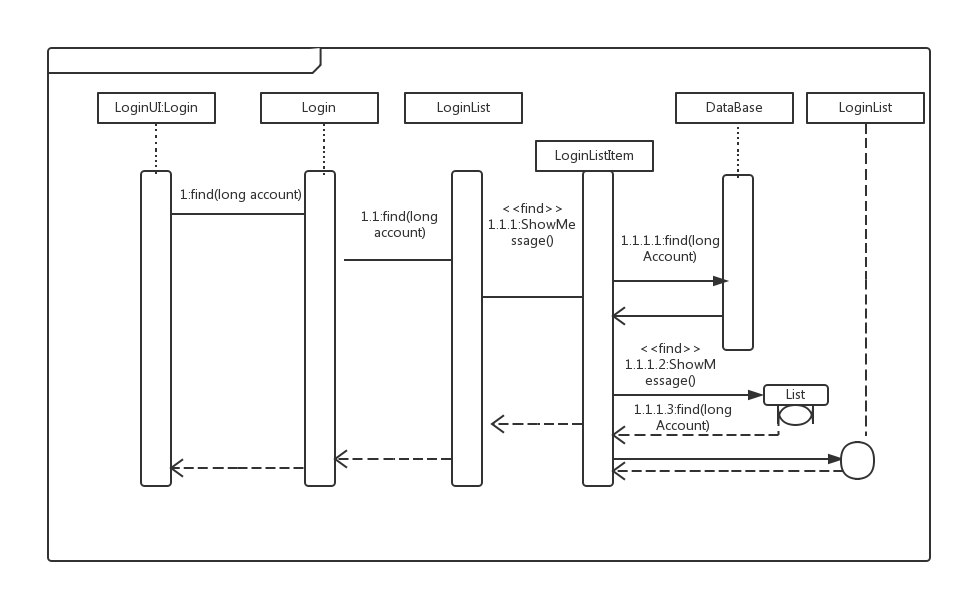


图3.查找的顺序图

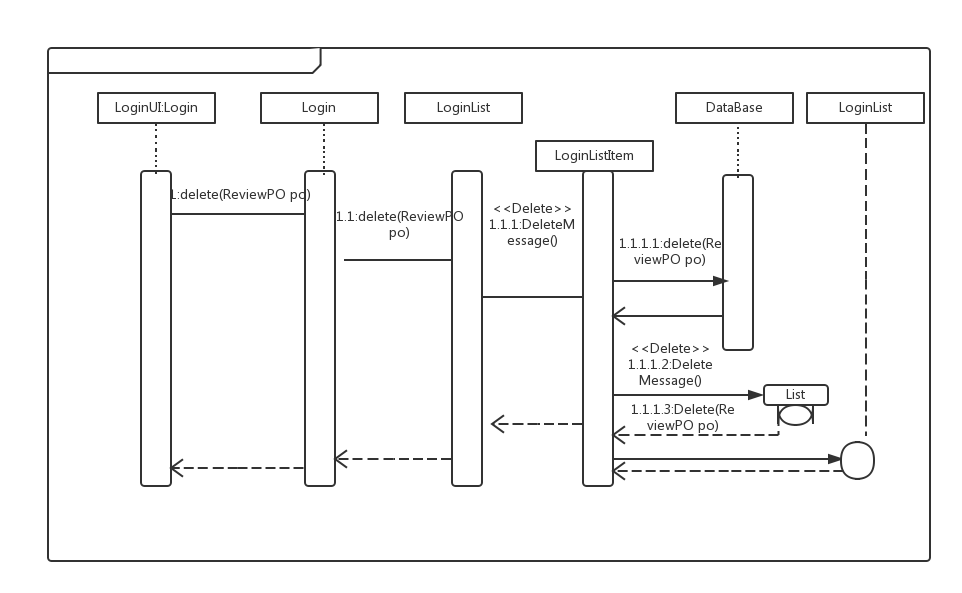
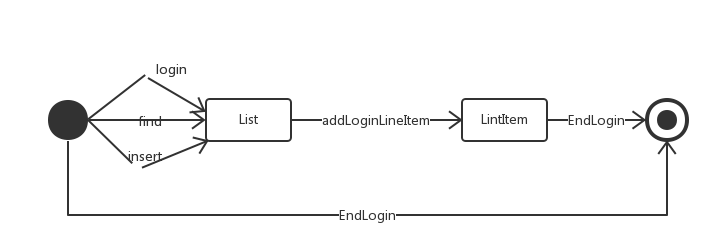


图4.删除的顺序图

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。

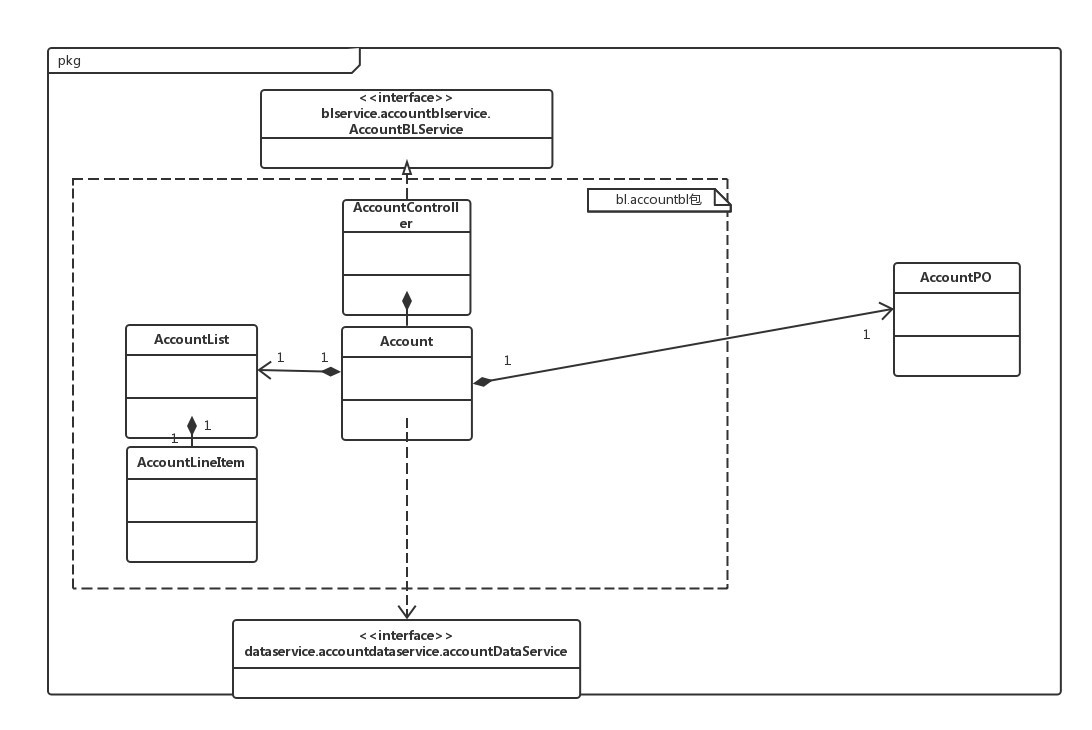
图3 Login对象状态图

**1.8 accountbl模块**

1.8.1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。

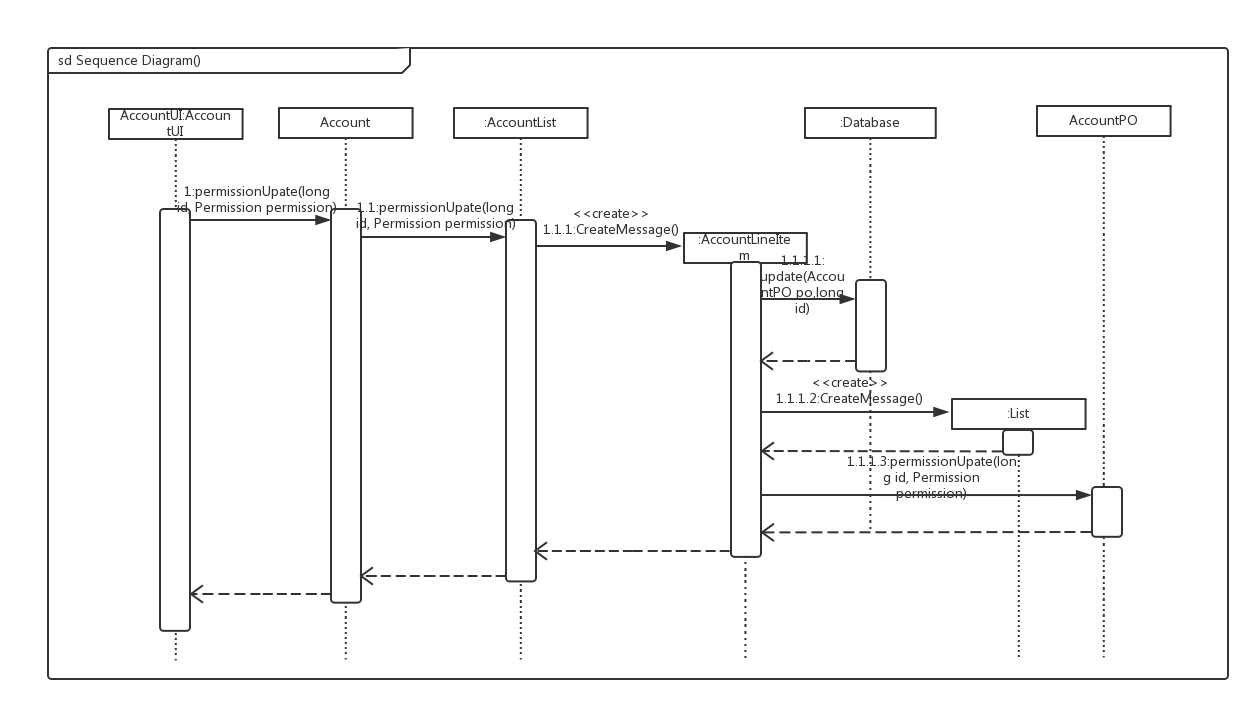
**表2 AccountController的接口规范**

|  |  |  |
| --- | --- | --- |
| **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 | |

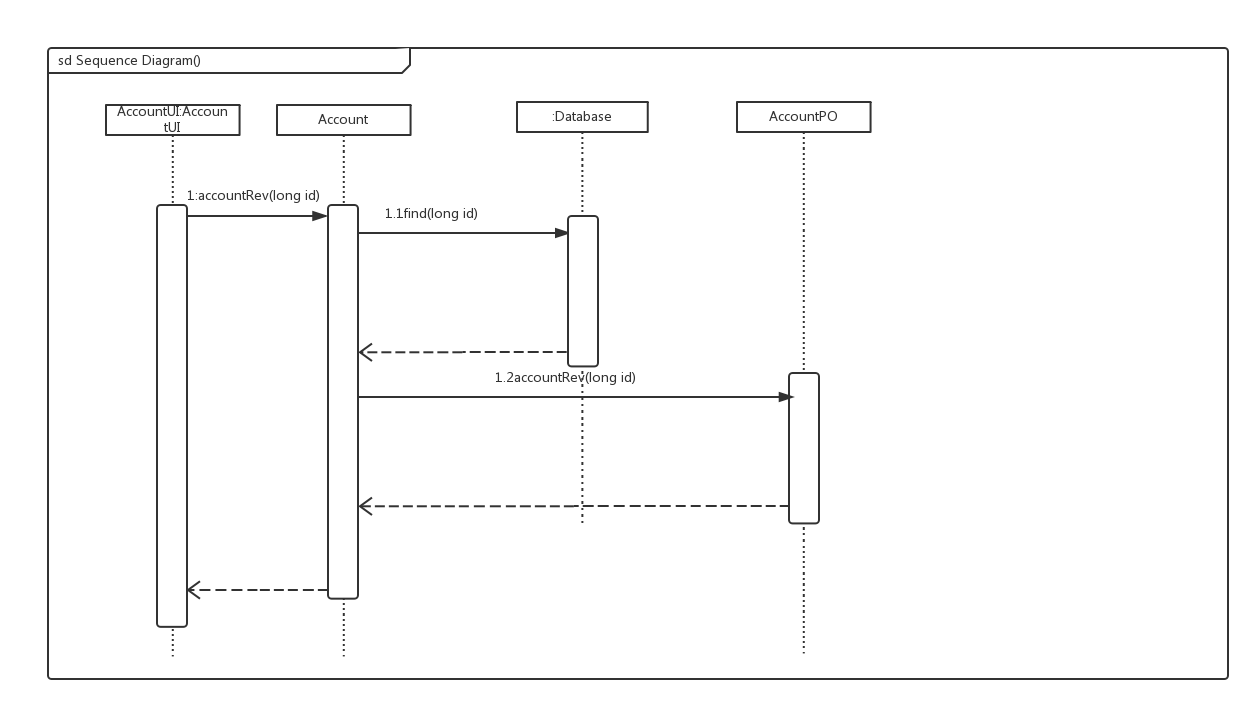
**表3 Account的接口规范**

|  |  |  |
| --- | --- | --- |
| **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

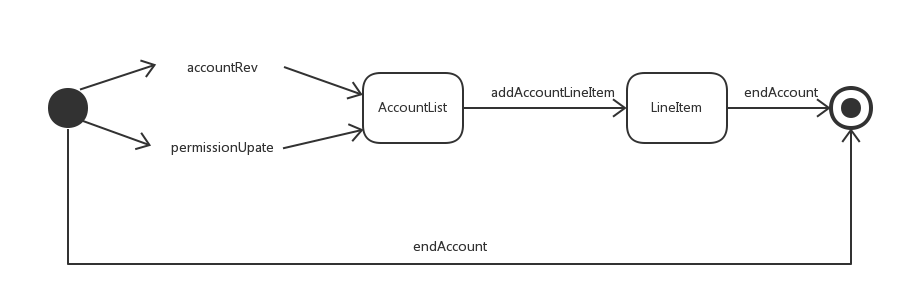


**图 修改账号信息顺序图**



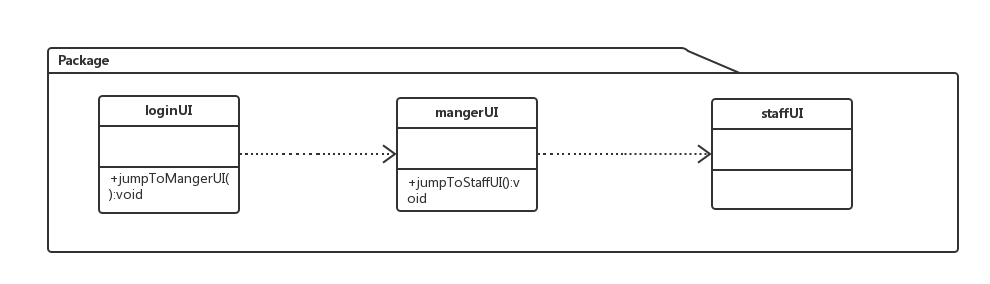
**图 得到账号信息顺序图**

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。

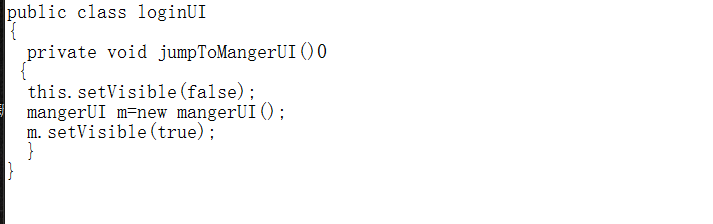


**2.界面跳转**

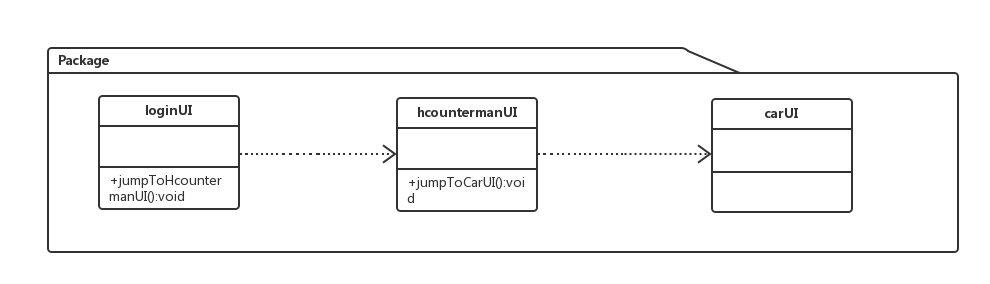
采用分散式控制风格如图



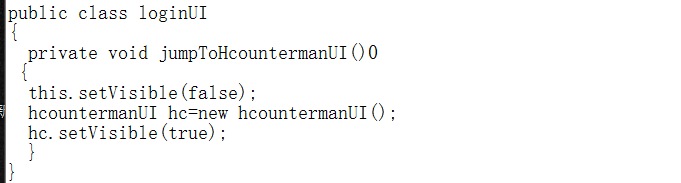
分散式控制风格类图



分散式控制风格代码



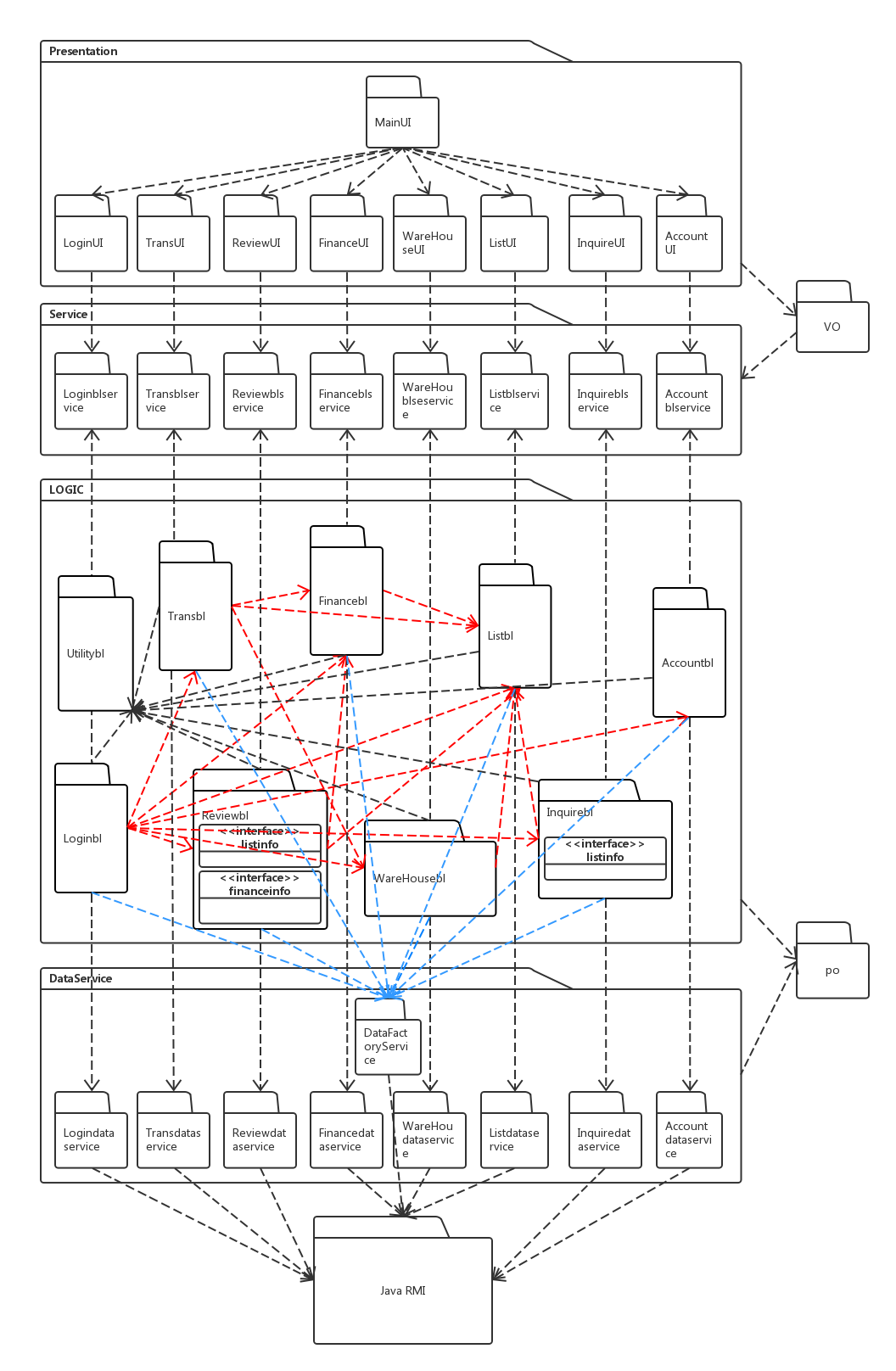
分散式控制风格类图



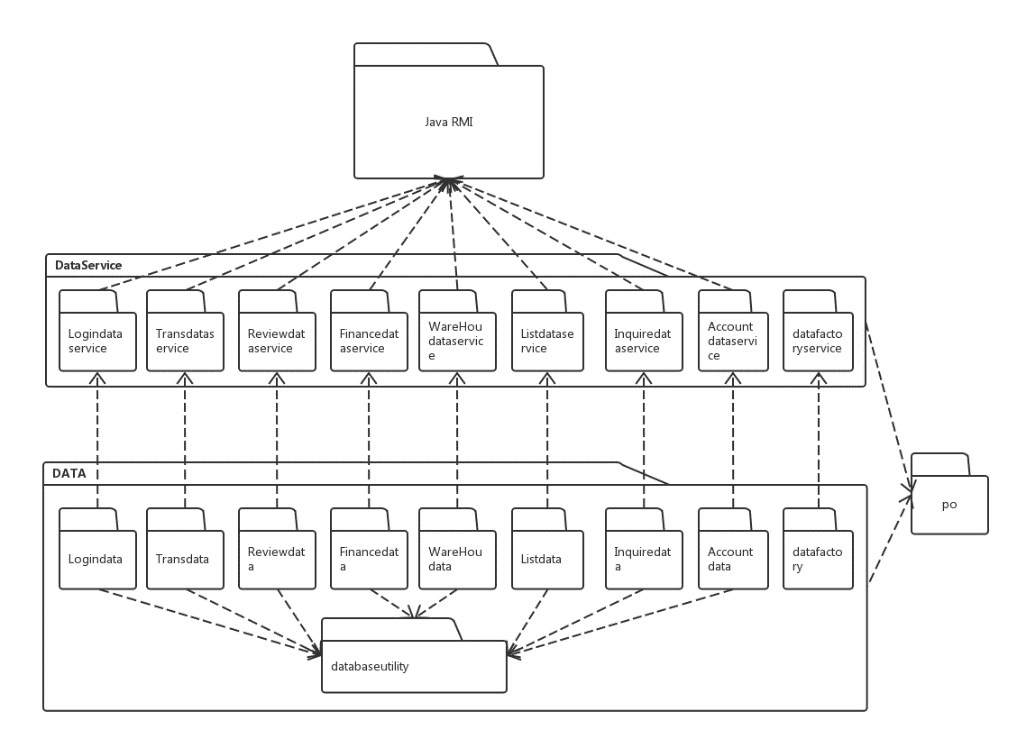
分散式控制风格代码

**3.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