详细设计描述文档

快递物流系统软件

学校： 南京大学

院系： 软件学院

编写小组： 33

|  |  |  |  |
| --- | --- | --- | --- |
| 更新人员 | 更新日期 | 更新内容 | 更新版本 |
| 黄迪璇 | 2015/11/2 | 构建了文档的基础框架 | v0.0 |
| 黄迪璇 | 2015/11/5 | 添加pamanagementbl的结构视角 | v0.1 |
| 黄迪璇 | 2015/11/6 | 添加pamanagementbl的状态图 | v0.2 |
| 邓逸鹏、何永俊、许玥琪 | 2015/11/15 | 添加了其余所有模块的结构视角 | v0.3 |
| 黄迪璇 | 2015/11/16 | 添加了目录和索引、依赖视角 | v1.0 |

目录

1 引言 3

1.1编制目的 3

1.2词汇表 3

1.3导读 3

1.4文档约定 3

1.5感谢 4

1.6参考资料 4

2 产品概述 4

3 体系结构设计概述 4

4 结构视角 5

4.1业务逻辑层的分解 5

4.1.1pamanagementsl模块 5

4.1.2strategysl模块 26

4.1.3auditsl模块 34

4.1.4 reportsl模块 38

4.1.5Receivesl模块 42

4.1.6transportsl模块 50

4.1.7checksl模块 56

4.1.8financesl模块 59

4.1.9usersl模块 70

4.1.10stocksl模块 78

4.1.11ioputsl模块 84

4.1.12sendsl模块 90

4.2用户界面层的行为 96

5.依赖视角 97

# 1 引言

## 1.1编制目的

本文档承接快递物流系统软件体系描述文档，继续完成快递物流系统的详细设计工作，达到指导后续的软件构造的目的，同时实现设计人员与构造人员、测试人员、用户之间的交流。

本文档的预期读者为开发人员、构造人员、测试人员、最终用户以及项目管理人员。

## 1.2词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
| ELS | 快递物流系统 | 为本次工程的开发产品 |
| vo | 值对象 | 用于业务层之间的数据传递 |
| po | 持久对象 | 与数据库中的表对应的Java对象 |

## 1.3导读

该报告分为5个部分，第一部分是引言，主要是介绍文档的基本信息；第二部分是产品概述，是快递物流系统的概括性描述；第三部分是对体系结构设计的概述；第四、第五和第六部分分别是结构视角、依赖视角，从多个视角对软件设计进行详细的描述。

## 1.4文档约定

对文档的的任何改动都应经过质量保证人员的统一，详细改动由小组成员共同商议。

## 1.5感谢

本用例文档由小组成员黄迪璇编写，何永俊负责审阅，详细描述由何永俊、许玥琪、邓逸鹏和黄迪璇共同完成，感谢他们的辛勤付出。

## 1.6参考资料

《软件工程与计算（卷二）——软件开发的技术基础》 机械工程出版社

《快递物流系统软件需求规格说明文档v0.5》

《快递物流系统软件体系描述文档v0.8》

# 2 产品概述

快递物流系统是为一本地民营物流企业开发的业务系统，开发的目的是帮助该企业处理日常的重点业务，包括收发快件，库存管理，财务操作和人员机构管理。

通过快递物流系统的应用，期望系统上线运行六个月后，保持合理库存，提高用户服务体验，增加业务额，提高财务人员工作效率，为经理的决策做支持。

# 3 体系结构设计概述

参考《快递物流系统软件体系描述文档v0.8》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请参考《快递物流系统软件体系描述文档v0.8》

### 4.1.1pamanagementsl模块

(1)模块描述

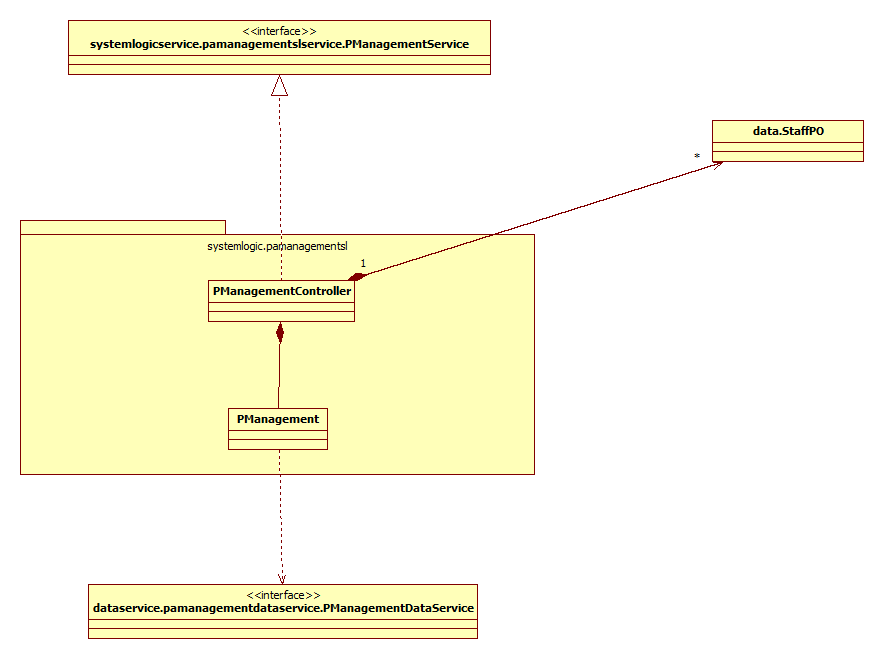
pamanagementsl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

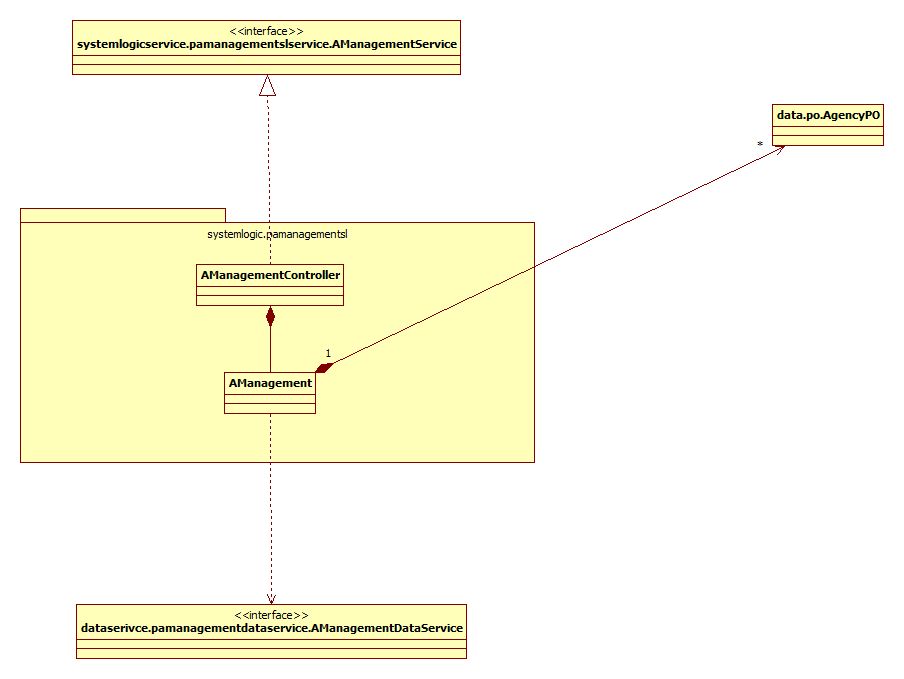
pamanagementsl模块的职责及接口参见快递物流系统软件体系描述文档。

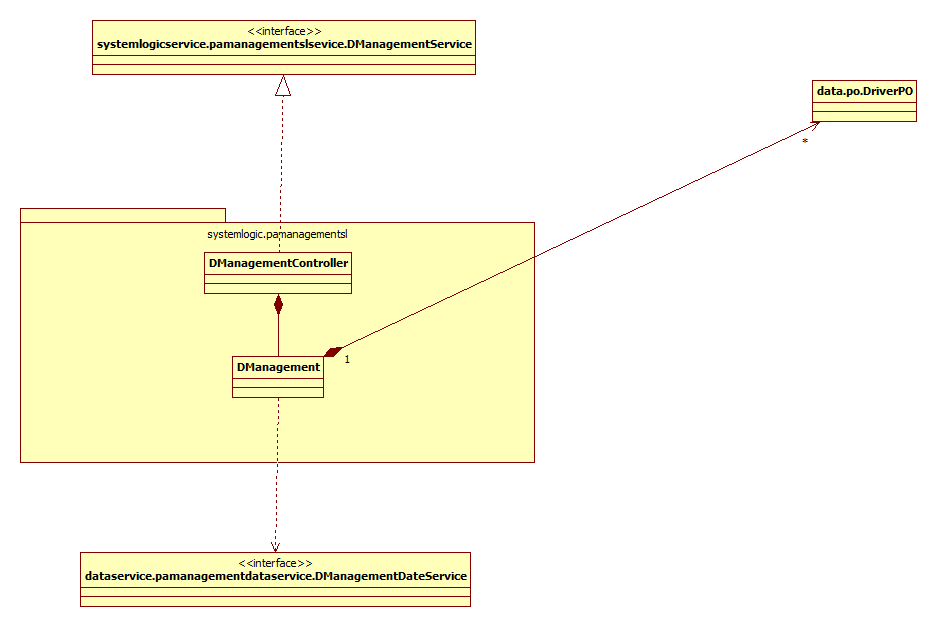
(2)整体结构

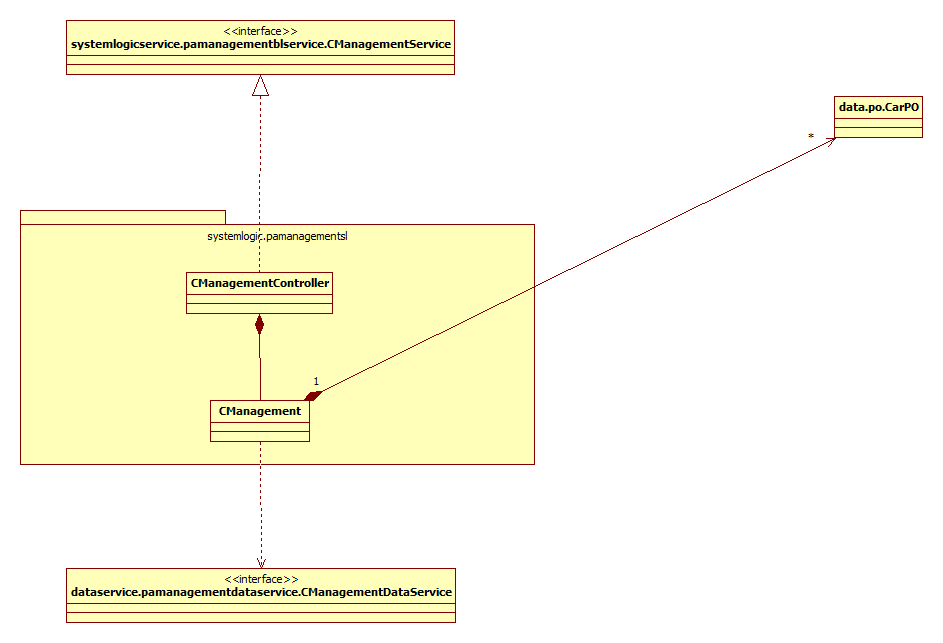
根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了pamanagementslservice和pamanagementdataservice两个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了PManagementController、AManagementController、DManagementController、CManagementController，这样它们会将对人员机构管理的系统逻辑处理委托给PManagement、AMangement、DManagement、CManagement对象。AgencyPO、StaffPO、DriverPO、CarPO是作为人员机构管理信息的持久化对象被添加到设计模型上的。

pamanagementsl模块的图如下图所示。









pamanagementsl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应登录界面所需要的服务 |
| PManagementController | 负责实现人员管理界面所需要的服务 |
| AManagementController | 负责实现机构管理界面所需要的服务 |
| DManagementController | 负责实现司机管理界面所需要的服务 |
| CManagementController | 负责实现车辆管理界面所需要的服务 |
| PManagement | 人员管理的领域模型对象，拥有一次人员管理所持有的人员信息单列表、操作记录等信息，可以帮助完成人员管理界面所需要的服务 |
| AManagement | 机构管理的领域模型对象，拥有一次机构管理所持有的机构信息单列表、操作记录等信息，可以帮助完成机构管理界面所需要的服务 |
| DManagement | 司机管理的领域模型对象，拥有一次司机管理所持有的司机信息单列表、操作记录等信息，可以帮助完成司机管理界面所需要的服务 |
| CManagement | 车辆管理的领域模型对象，拥有一次车辆管理所持有的车辆信息单列表、操作记录等信息，可以帮助完成车辆管理界面所需要的服务 |

(3)模块内部类的接口规范

PManagementController和PManagement的接口规范如下表

PManagementController的接口规范

|  |  |  |
| --- | --- | --- |
| PManagementController.select | 语法 | public StaffVO select(String id); |
| 前置条件 | 已开启新的一轮人员管理 |
| 后置条件 | 调用PManagement领域对象的select方法 |
| PManagementController.delect | 语法 | public void delect(String id); |
| 前置条件 | 在一轮的人员信息管理，已选择好需要删除的人员信息单 |
| 后置条件 | 调用PManagement领域对像的delect方法 |
| PManagementContorller.revise | 语法 | public StaffVO revise(String id); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被选择 |
| 后置条件 | 调用PManagement领域对象的revise方法 |
| PManagementController.saveChange | 语法 | public void saveChange(StaffVO vo); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被修改结束 |
| 后置条件 | 调用PManagement领域对象的saveChange方法 |
| PManagementController.save | 语法 | public void save(StaffVO vo); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被添加 |
| 后置条件 | 调用PManagement领域对象的save方法 |
| PManagementController.add | 语法 | public void add(); |
| 前置条件 | 在人员管理的过程中，添加按钮被点击 |
| 后置条件 | 调用PManagement领域对象的add方法 |
| PManagementController.getAllStaff | 语法 | public ArrayList<StaffVO> getAllStaff(); |
| 前置条件 | 开启新的一轮人员管理 |
| 后置条件 | 调用PManagement领域对象的getAllStaff方法 |
| PManagementController.endPManagement | 语法 | public void endPManagement(); |
| 前置条件 | 人员管理已开始 |
| 后置条件 | 调用PManagement领域对象的endPManagement方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| PManagement.select | 返回人员详细信息 |
| PManagement.delect(String id) | 根据id删除人员 |
| PManagement.delect(StaffVO vo) | 根据对象删除人员 |
| PManagement.revise(String id) | 根据ID返回可修改的人员信息单 |
| PManagement.saceChange | 保存修改后的人员信息单 |
| PManagement.save | 保存新增的人员信息单 |
| PManagement.add | 返回新的人员信息单 |
| PManagement.getAllStaff | 返回所有的人员信息单 |
| PManagement.endPManagement | 结束人员管理 |

Pmanagement的接口规范

|  |  |  |
| --- | --- | --- |
| PManagement.select | 语法 | public StaffVO select(String id); |
| 前置条件 | 开启新的一轮人员管理 |
| 后置条件 | 返回该人员的详细信息 |
| PManagement.delect | 语法 | public void delect(String id); |
| 前置条件 | 一开启新一轮的人员信息管理，已选择好需要删除的人员信息单，删除按钮被点击 |
| 后置条件 | 在一个人员管理回合中，删除该人员信息单，持久化更新本次删除的信息单，结束人员信息删除 |
| PManagement.revise | 语法 | public StaffVO revise(String id); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被选择，修改按钮被点击 |
| 后置条件 | 开启人员信息单对象的修改方法 |
| PManagement.saveChange | 语法 | public void saveChange(StaffVO vo); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被修改，保存按钮被点击 |
| 后置条件 | 在一次人员管理的回合中，持久化更新人员信息单对象的数据，结束本次人员信息修改 |
| PManagement.save | 语法 | public void save(StaffVO vo); |
| 前置条件 | 在人员管理的过程中，人员信息单已经被添加，保存按钮被点击 |
| 后置条件 | 在一次人员管理的回合中，持久化更新人员信息单对象的数据，结束本次人员信息添加 |
| PManagement.add | 语法 | public void add(); |
| 前置条件 | 在人员管理的过程中，添加按钮被点击 |
| 后置条件 | 返回新的人员信息单 |
| PManagement.getAllStaff | 语法 | public ArrayList<StaffVO> getAllStaff(); |
| 前置条件 | 开启新的一轮人员管理 |
| 后置条件 | 返回所有的人员信息单 |
| PManagement.endPManagement | 语法 | public void endPManagement(); |
| 前置条件 | 人员管理已开始 |
| 后置条件 | 结束本次人员管理，持久化更新涉及的领域对象的数据 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| PManagementDataService.find(String id) | 根据id返回的单一持久化对象 |
| PManagementDataService.insert(StaffPO po) | 插入单一持久化对象 |
| PManagementDataService.delect(StaffPO po) | 根据对象删除单一持久化对象 |
| PManagementDataService.delect(String id) | 根据id删除单一持久化对象 |
| PManagementDataService.update(StaffPO po) | 更新单一持久化对象 |
| PManagementDataService.findAll(); | 返回所有的持久化对象 |

AManagementController和AManagement的接口规范如下表

AManagementController的接口规范

|  |  |  |
| --- | --- | --- |
| AManagementController.select | 语法 | public AgencyVO select(String id); |
| 前置条件 | 已开启新的一轮机构管理 |
| 后置条件 | 调用AManagement领域对象的select方法 |
| AManagementController.delect | 语法 | public void delect(String id); |
| 前置条件 | 在一轮的机构信息管理，已选择好需要删除的机构信息单 |
| 后置条件 | 调用AManagement领域对像的delect方法 |
| AManagementContorller.revise | 语法 | public AgencyVO revise(String id); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被选择 |
| 后置条件 | 调用AManagement领域对象的revise方法 |
| AManagementController.saveChange | 语法 | public void saveChange(AgencyVO vo); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被修改结束 |
| 后置条件 | 调用AManagement领域对象的saveChange方法 |
| AManagementController.save | 语法 | public void save(AgencyVO vo); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被添加 |
| 后置条件 | 调用AManagement领域对象的save方法 |
| AManagementController.add | 语法 | public void add(); |
| 前置条件 | 在机构管理的过程中，添加按钮被点击 |
| 后置条件 | 调用AManagement领域对象的add方法 |
| AManagementController.getAllAgency | 语法 | public ArrayList<AgencyVO> getAllAgency(); |
| 前置条件 | 开启新的一轮机构管理 |
| 后置条件 | 调用AManagement领域对象的getAllAgency方法 |
| AManagementController.endAManagement | 语法 | public void endAManagement(); |
| 前置条件 | 机构管理已开始 |
| 后置条件 | 调用AManagement领域对象的endAManagement方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| AManagement.select | 返回机构详细信息 |
| AManagement.delect(String id) | 根据id删除机构 |
| AManagement.delect(AgencyVO vo) | 根据对象删除机构 |
| AManagement.revise(String id) | 根据ID返回可修改的机构信息单 |
| AManagement.saceChange | 保存修改后的机构信息单 |
| AManagement.save | 保存新增的机构信息单 |
| AManagement.add | 返回新的机构信息单 |
| AManagement.getAllAgency | 返回所有的机构信息单 |
| AManagement.endAManagement | 结束机构管理 |

AManagement的接口规范

|  |  |  |
| --- | --- | --- |
| AManagement.select | 语法 | public AgencyVO select(String id); |
| 前置条件 | 开启新的一轮机构管理 |
| 后置条件 | 返回该机构的详细信息 |
| AManagement.delect | 语法 | public void delect(String id); |
| 前置条件 | 已开启新一轮的机构信息管理，已选择好需要删除的机构信息单，删除按钮被点击 |
| 后置条件 | 在一个机构管理回合中，删除该机构信息单，持久化更新本次删除的信息单，结束机构信息删除 |
| AManagement.revise | 语法 | public AgencyVO revise(String id); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被选择，修改按钮被点击 |
| 后置条件 | 开启机构信息单对象的修改方法 |
| AManagement.saveChange | 语法 | public void saveChange(AgencyVO vo); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被修改，保存按钮被点击 |
| 后置条件 | 在一次机构管理的回合中，持久化更新机构信息单对象的数据，结束本次机构信息修改 |
| AManagement.save | 语法 | public void save(AgencyVO vo); |
| 前置条件 | 在机构管理的过程中，机构信息单已经被添加，保存按钮被点击 |
| 后置条件 | 在一次机构管理的回合中，持久化更新机构信息单对象的数据，结束本次机构信息添加 |
| AManagement.add | 语法 | public void add(); |
| 前置条件 | 在机构管理的过程中，添加按钮被点击 |
| 后置条件 | 返回新的机构信息单 |
| AManagement.getAllAgency | 语法 | public ArrayList<AgencyVO> getAllAgency(); |
| 前置条件 | 开启新的一轮机构管理 |
| 后置条件 | 返回所有的机构信息单 |
| AManagement.endAManagement | 语法 | public void endAManagement(); |
| 前置条件 | 机构管理已开始 |
| 后置条件 | 结束本次机构管理，持久化更新涉及的领域对象的数据 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| AManagementDataService.find(String id) | 根据id返回的单一持久化对象 |
| AManagementDataService.insert(AgencyPO po) | 插入单一持久化对象 |
| AManagementDataService.delect(AgencyPO po) | 根据对象删除单一持久化对象 |
| AManagementDataService.delect(String id) | 根据id删除单一持久化对象 |
| AManagementDataService.update(AgencyPO po) | 更新单一持久化对象 |
| AManagementDataService.findAll(); | 返回所有的持久化对象 |

DManagementController和DManagement的接口规范如下表

DManagementController的接口规范

|  |  |  |
| --- | --- | --- |
| DManagementController.select | 语法 | public DriverVO select(String id); |
| 前置条件 | 已开启新的一轮司机管理 |
| 后置条件 | 调用DManagement领域对象的select方法 |
| DManagementController.delect | 语法 | public void delect(String id); |
| 前置条件 | 在一轮的司机信息管理，已选择好需要删除的司机信息单 |
| 后置条件 | 调用DManagement领域对象的delect方法 |
| DManagementContorller.revise | 语法 | public DriverVO revise(String id); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被选择 |
| 后置条件 | 调用DManagement领域对象的revise方法 |
| DManagementController.saveChange | 语法 | public void saveChange(DriverVO vo); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被修改结束 |
| 后置条件 | 调用DManagement领域对象的saveChange方法 |
| DManagementController.save | 语法 | public void save(DiverVO vo); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被添加 |
| 后置条件 | 调用DManagement领域对象的save方法 |
| DManagementController.add | 语法 | public void add(); |
| 前置条件 | 在司机管理的过程中，添加按钮被点击 |
| 后置条件 | 调用DManagement领域对象的add方法 |
| DManagementController.getAllDriver | 语法 | public ArrayList<DriverVO> getAllDriver(); |
| 前置条件 | 开启新的一轮司机管理 |
| 后置条件 | 调用DManagement领域对象的getAllDriver方法 |
| DManagementController.endDManagement | 语法 | public void endDManagement(); |
| 前置条件 | 司机管理已开始 |
| 后置条件 | 调用DManagement领域对象的endDManagement方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| DManagement.select | 返回司机详细信息 |
| DManagement.delect(String id) | 根据id删除司机 |
| DManagement.delect(DriverVO vo) | 根据对象删除司机 |
| DManagement.revise(String id) | 根据ID返回可修改的司机信息单 |
| DManagement.saceChange | 保存修改后的司机信息单 |
| DManagement.save | 保存新增的司机信息单 |
| DManagement.add | 返回新的司机信息单 |
| DManagement.getAllDriver | 返回所有的司机信息单 |
| DManagement.endDManagement | 结束司机管理 |

DManagement的接口规范

|  |  |  |
| --- | --- | --- |
| DManagement.select | 语法 | public DriverVO select(String id); |
| 前置条件 | 开启新的一轮司机管理 |
| 后置条件 | 返回该司机的详细信息 |
| DManagement.delect | 语法 | public void delect(String id); |
| 前置条件 | 已开启新一轮的司机信息管理，已选择好需要删除的司机信息单，删除按钮被点击 |
| 后置条件 | 在一个司机管理回合中，删除该司机信息单，持久化更新本次删除的信息单，结束司机信息删除 |
| DManagement.revise | 语法 | public DriverVO revise(String id); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被选择，修改按钮被点击 |
| 后置条件 | 开启司机信息单对象的修改方法 |
| DManagement.saveChange | 语法 | public void saveChange(DriverVO vo); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被修改，保存按钮被点击 |
| 后置条件 | 在一次司机管理的回合中，持久化更新司机信息单对象的数据，结束本次机构信息修改 |
| DManagement.save | 语法 | public void save(DriverVO vo); |
| 前置条件 | 在司机管理的过程中，司机信息单已经被添加，保存按钮被点击 |
| 后置条件 | 在一次司机管理的回合中，持久化更新司机信息单对象的数据，结束本次机构信息添加 |
| DManagement.add | 语法 | public void add(); |
| 前置条件 | 在司机管理的过程中，添加按钮被点击 |
| 后置条件 | 返回新的机构信息单 |
| DManagement.getAllDriver | 语法 | public ArrayList<DriverVO> getAllDriver(); |
| 前置条件 | 开启新的一轮司机管理 |
| 后置条件 | 返回所有的司机信息单 |
| DManagement.endDManagement | 语法 | public void endDManagement(); |
| 前置条件 | 司机管理已开始 |
| 后置条件 | 结束本次司机管理，持久化更新涉及的领域对象的数据 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DManagementDataService.find(String id) | 根据id返回的单一持久化对象 |
| DManagementDataService.insert(DriverPO po) | 插入单一持久化对象 |
| DManagementDataService.delect(DriverPO po) | 根据对象删除单一持久化对象 |
| DManagementDataService.delect(String id) | 根据id删除单一持久化对象 |
| DManagementDataService.update(DriverPO po) | 更新单一持久化对象 |
| DManagementDataService.findAll(); | 返回所有的持久化对象 |

CManagementController和CManagement的接口规范如下表

CManagementController的接口规范

|  |  |  |
| --- | --- | --- |
| CManagementController.select | 语法 | public CarVO select(String id); |
| 前置条件 | 已开启新的一轮车辆管理 |
| 后置条件 | 调用CManagement领域对象的select方法 |
| CManagementController.delect | 语法 | public void delect(String id); |
| 前置条件 | 在一轮的车辆信息管理，已选择好需要删除的车辆信息单 |
| 后置条件 | 调用CManagement领域对像的delect方法 |
| CManagementContorller.revise | 语法 | public CarVO revise(String id); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被选择 |
| 后置条件 | 调用CManagement领域对象的revise方法 |
| CManagementController.saveChange | 语法 | public void saveChange(CarVO vo); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被修改结束 |
| 后置条件 | 调用CManagement领域对象的saveChange方法 |
| CManagementController.save | 语法 | public void save(CarVO vo); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被添加 |
| 后置条件 | 调用CManagement领域对象的save方法 |
| CManagementController.add | 语法 | public void add(); |
| 前置条件 | 在车辆管理的过程中，添加按钮被点击 |
| 后置条件 | 调用CManagement领域对象的add方法 |
| CManagementController.getAllCar | 语法 | public ArrayList<CarVO> getAllCar(); |
| 前置条件 | 开启新的一轮车辆管理 |
| 后置条件 | 调用CManagement领域对象的getAllCar方法 |
| CManagementController.endCManagement | 语法 | public void endCManagement(); |
| 前置条件 | 车辆管理已开始 |
| 后置条件 | 调用CManagement领域对象的endCManagement方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| CManagement.select | 返回车辆详细信息 |
| CManagement.delect(String id) | 根据id删除车辆 |
| CManagement.delect(CarVO vo) | 根据对象删除车辆 |
| CManagement.revise(String id) | 根据ID返回可修改的车辆信息单 |
| CManagement.saceChange | 保存修改后的车辆信息单 |
| CManagement.save | 保存新增的车辆信息单 |
| CManagement.add | 返回新的车辆信息单 |
| CManagement.getAllCar | 返回所有的车辆信息单 |
| CManagement.endCManagement | 结束车辆管理 |

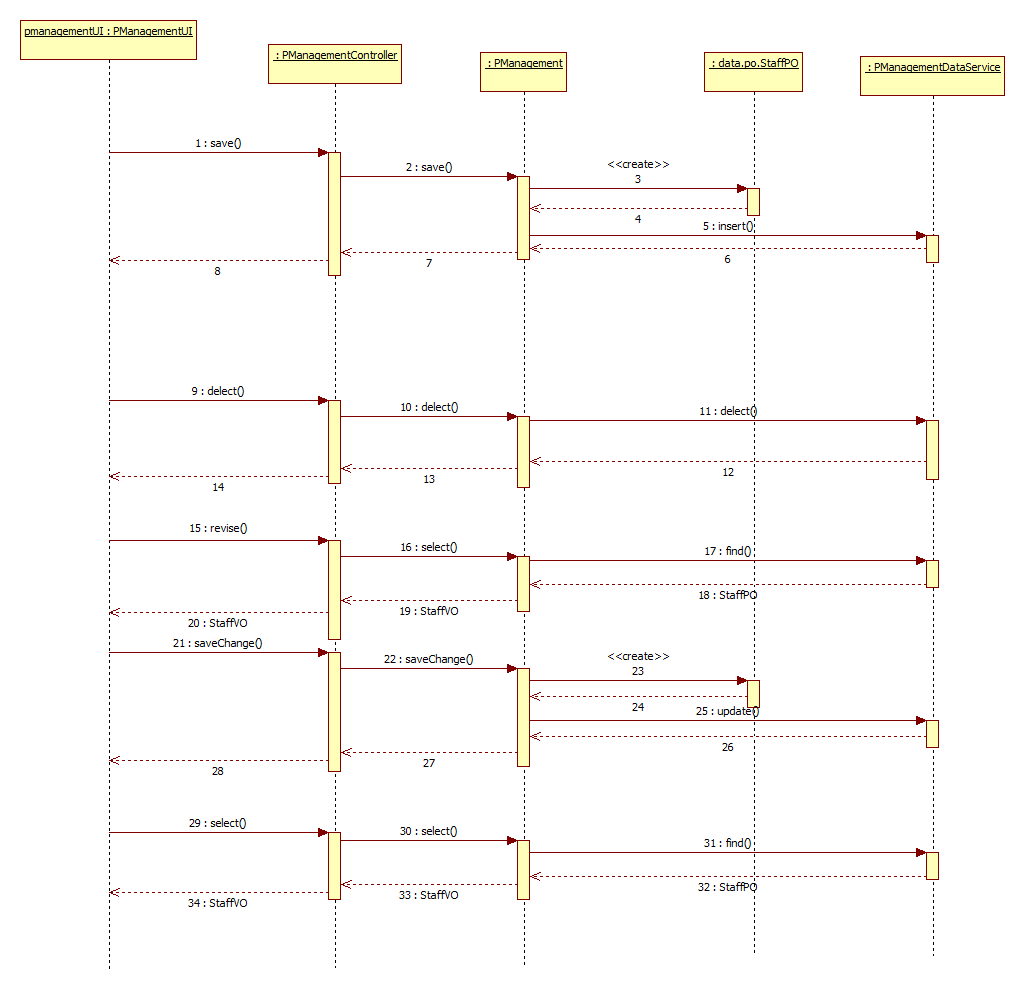
CManagement的接口规范

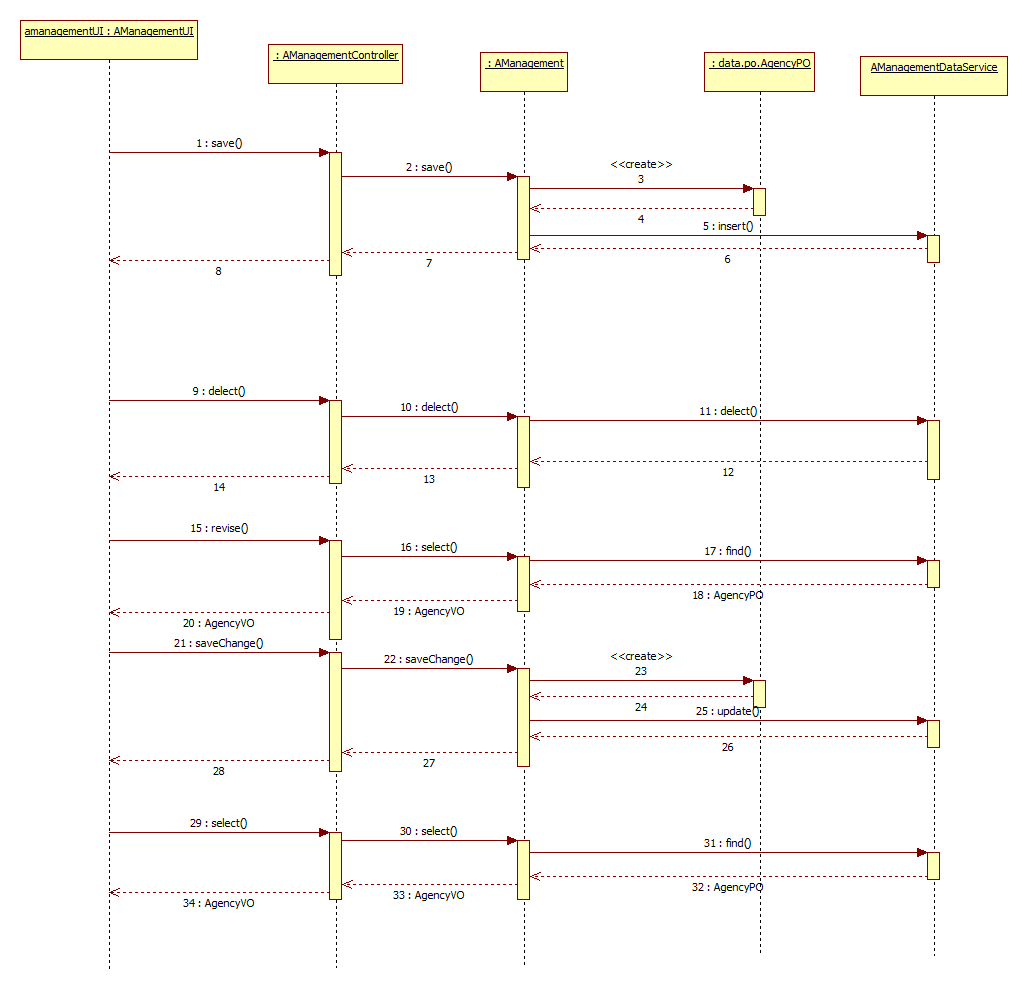
|  |  |  |
| --- | --- | --- |
| CManagement.select | 语法 | public CarVO select(String id); |
| 前置条件 | 开启新的一轮车辆管理 |
| 后置条件 | 返回该车辆的详细信息 |
| CManagement.delete | 语法 | public void delect(String id); |
| 前置条件 | 已开启新一轮的车辆信息管理，已选择好需要删除的车辆信息单，删除按钮被点击 |
| 后置条件 | 在一个车辆管理回合中，删除该车辆信息单，持久化更新本次删除的信息单，结束车辆信息删除 |
| CManagement.revise | 语法 | public CarVO revise(String id); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被选择，修改按钮被点击 |
| 后置条件 | 开启车辆信息单对象的修改方法 |
| CManagement.saveChange | 语法 | public void saveChange(CarVO vo); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被修改，保存按钮被点击 |
| 后置条件 | 在一次车辆管理的回合中，持久化更新车辆信息单对象的数据，结束本次车辆信息修改 |
| CManagement.save | 语法 | public void save(CarVO vo); |
| 前置条件 | 在车辆管理的过程中，车辆信息单已经被添加，保存按钮被点击 |
| 后置条件 | 在一次车辆管理的回合中，持久化更新车辆信息单对象的数据，结束本次车辆信息添加 |
| CManagement.add | 语法 | public void add(); |
| 前置条件 | 在车辆管理的过程中，添加按钮被点击 |
| 后置条件 | 返回新的车辆信息单 |
| CManagement.getAllCar | 语法 | public ArrayList<CarVO> getAllCar(); |
| 前置条件 | 开启新的一轮车辆管理 |
| 后置条件 | 返回所有的车辆信息单 |
| CManagement.endCManagement | 语法 | public void endCManagement(); |
| 前置条件 | 车辆管理已开始 |
| 后置条件 | 结束本次车辆管理，持久化更新涉及的领域对象的数据 |

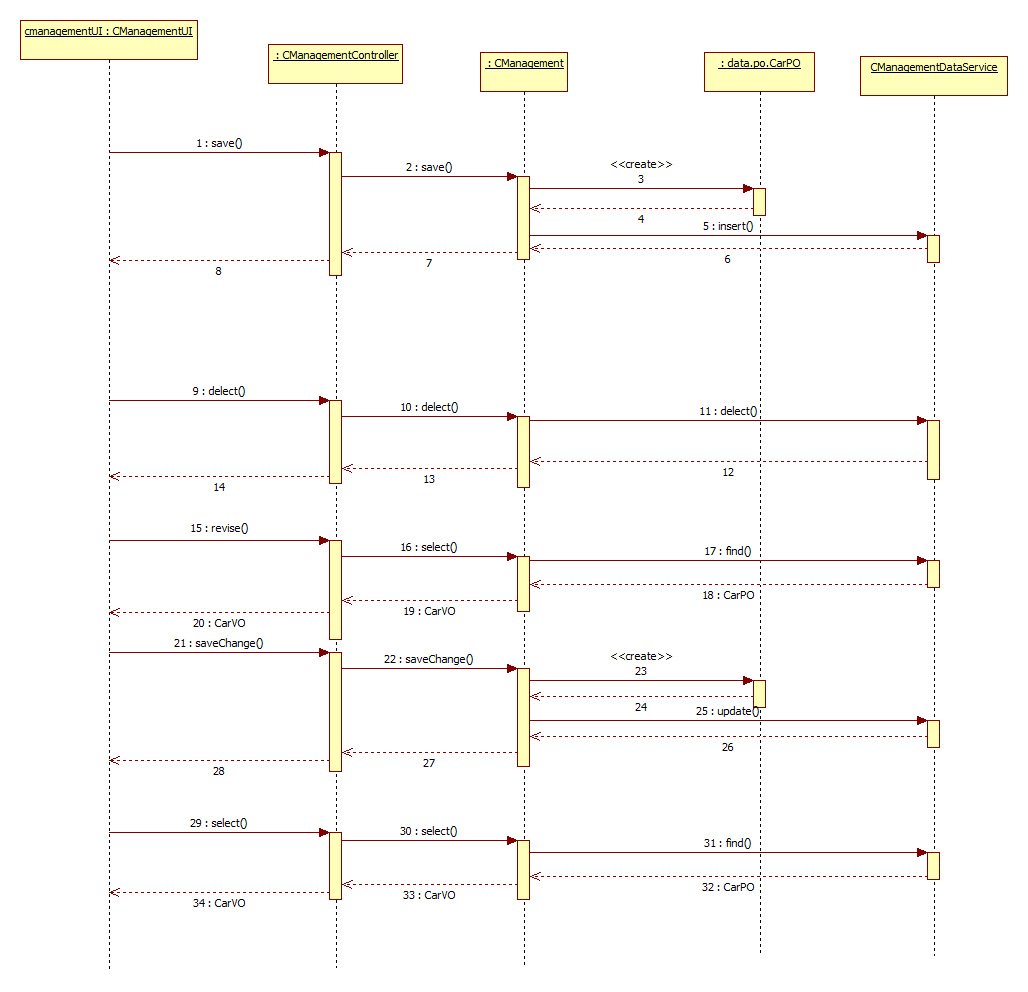
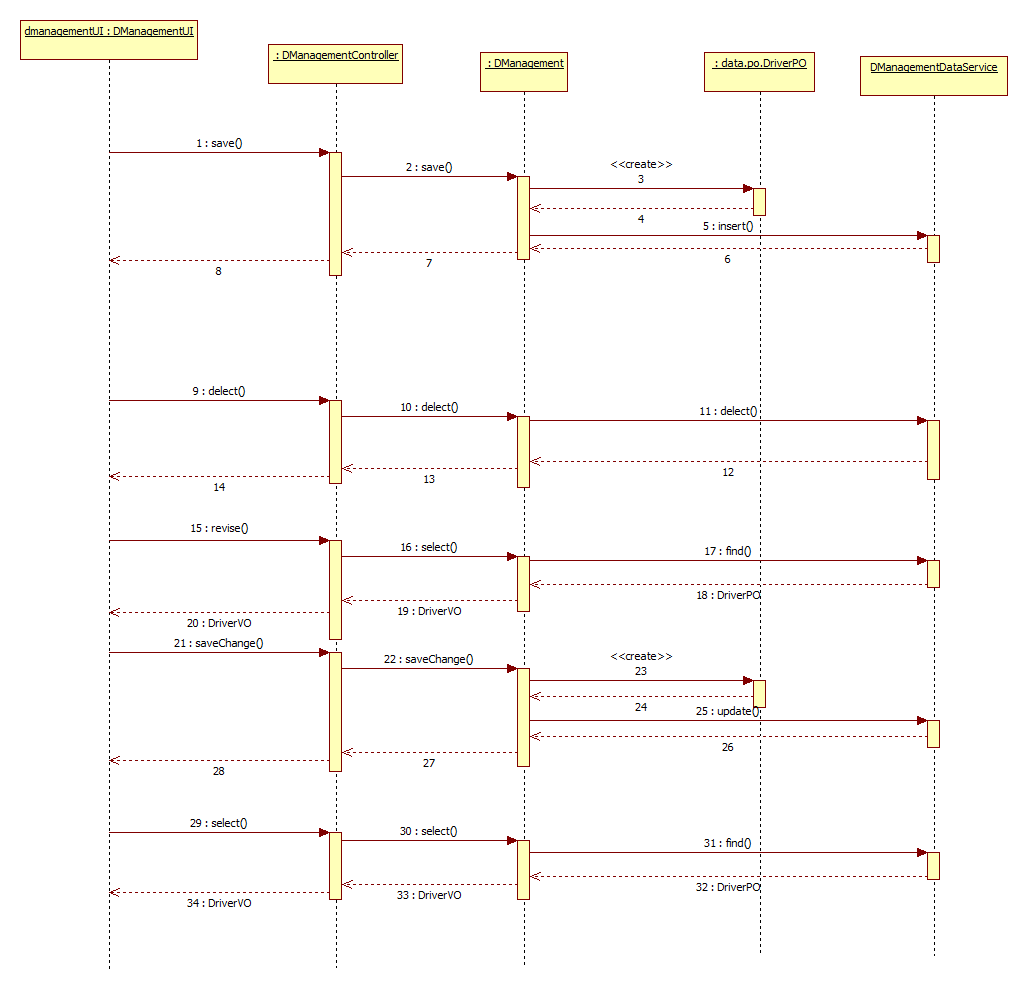
需要的服务（需接口）

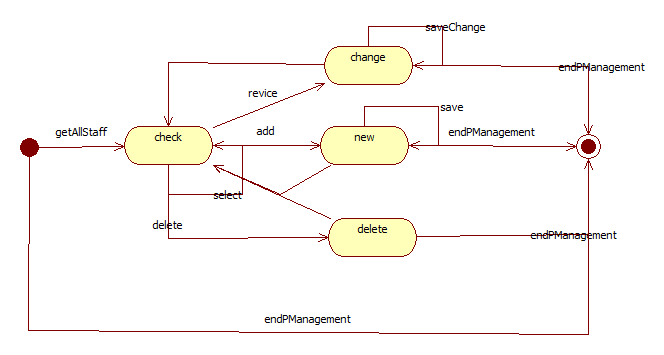
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| CManagementDataService.find(String id) | 根据id返回的单一持久化对象 |
| CManagementDataService.insert(CarPO po) | 插入单一持久化对象 |
| CManagementDataService.delect(CarPO po) | 根据对象删除单一持久化对象 |
| CManagementDataService.delect(String id) | 根据id删除单一持久化对象 |
| CManagementDataService.update(CarPO po) | 更新单一持久化对象 |
| CManagementDataService.findAll(); | 返回所有的持久化对象 |

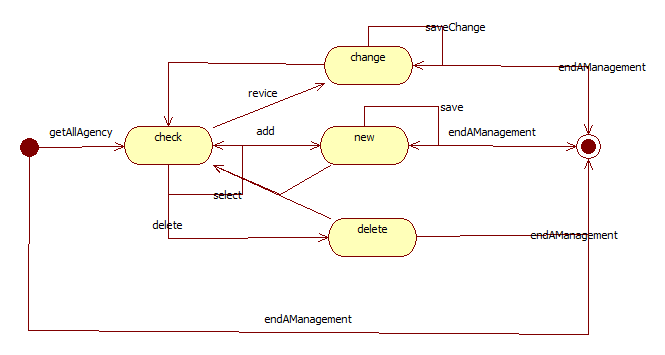
(4)pamanagementsl模块的动态模型

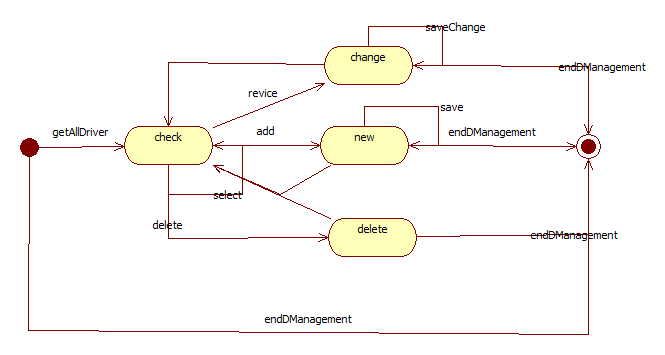


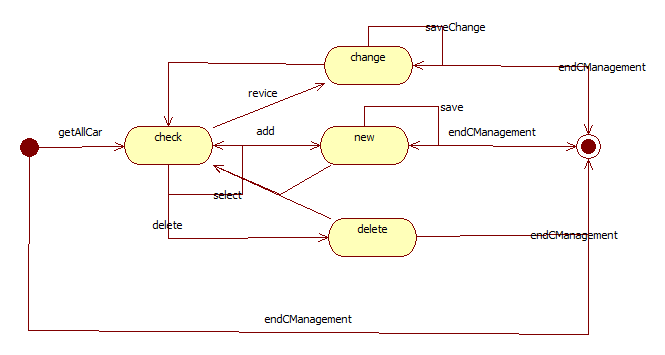












### 4.1.2strategysl模块

(1)模块描述

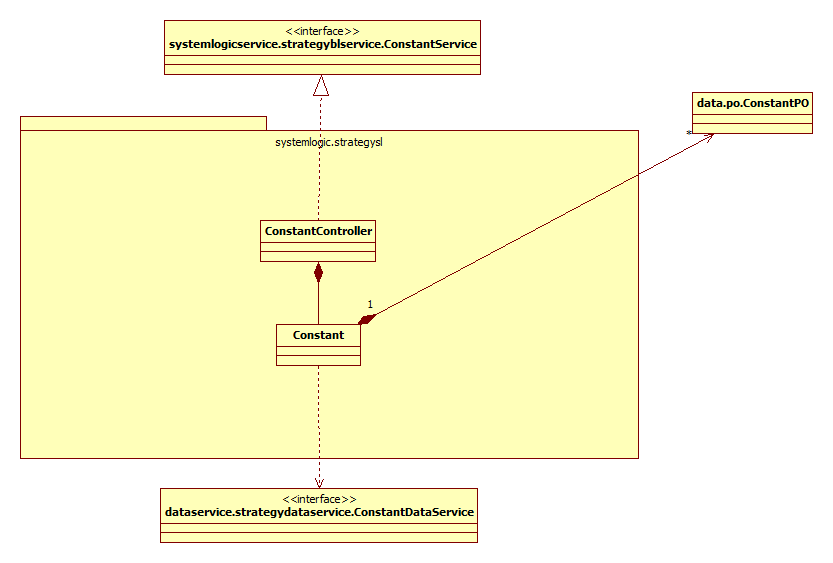
strategysl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

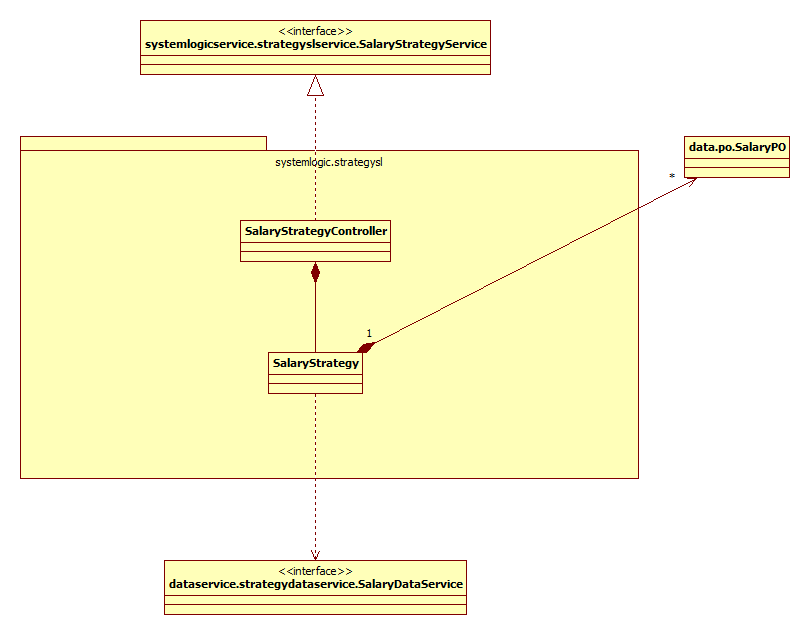
strategysl模块的职责及接口参见快递物流系统软件体系描述文档。

(2)整体结构

根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了strategyslservice和strategydataservice两个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了ConstantController和SalaryStrategyController，这样它们会将对制定策略的系统逻辑处理委托给对象Constant和SalaryStrategy。ConstantPO、SalaryPO是作为制定常量和制定薪水策略的持久化对象被添加到设计模型上的。

strategybl模块的设计如下图所示





strategysl模块各个类的职责如下表所示。

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| ConstantController | 负责实现制定常量界面所需要的服务 |
| SalaryStrategyController | 负责实现制定薪水策略界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的id和密码，可以解决登录问题 |
| Constant | 制定常量的领域模型对象，拥有常量表、操作记录等信息，可以帮助完成制定常量界面所需要的服务 |
| SalaryStrategy | 制定薪水策略的领域模型对象，拥有薪水策略列表、操作记录等信息，可以帮助完成制定薪水策略界面所需要的服务 |

(3)模块内部的接口规范

ConstantController和Constant的接口规范如下表

ConstantController的接口规范

|  |  |  |
| --- | --- | --- |
| ConstantController.getConstant | 语法 | public ConstantVO getConstant(); |
| 前置条件 | 开启新的一轮制定常量 |
| 后置条件 | 调用Constant领域对象的getConstant方法 |
| ConstantController.newConstant | 语法 | public void newConstant(); |
| 前置条件 | 已经创建了一个Constant对象，制定新的常量表的请求已发出 |
| 后置条件 | 调用Constant领域对象的newConstant方法 |
| ConstantController.save | 语法 | public void save(ConstantVO vo); |
| 前置条件 | 已创建一个Constant对象，常量表的信息填写完整并符合填写规则 |
| 后置条件 | 调用Constant领域对象的save方法 |
| ConstantController.endConstant | 语法 | public void endConstant(); |
| 前置条件 | 已创建一个Constant对象 |
| 后置条件 | 调用Constant领域对象的endConstant方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| Constant.getConstant | 返回常量表 |
| Constant.newConstant | 返回新的常量表 |
| Constant.save | 保存新的常量表 |
| Constant.endConstant | 结束制定常量任务 |

Constant的接口规范

|  |  |  |
| --- | --- | --- |
| Constant.getConstant | 语法 | public ConstantVO getConstant(); |
| 前置条件 | 开启新的一轮制定常量 |
| 后置条件 | 返回先前的所制定的常量表 |
| Constant.newConstant | 语法 | public void newConstant(); |
| 前置条件 | 开启新的一轮制订常量，制定新的常量表的按钮被点击 |
| 后置条件 | 返回常量制定表 |
| Constant.save | 语法 | public void save(ConstantVO vo); |
| 前置条件 | 在制定常量的过程中，保存按钮被点击 |
| 后置条件 | 在一个制定常量的回合中，持久化更新本次制定的常量。 |
| Constant.endConstant | 语法 | public void endConstant(); |
| 前置条件 | 在制定常量的过程中，结束按钮被点击 |
| 后置条件 | 在一个制定常量的回合中，持久化更新本次制定的常量，结束本次制定常量回合 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| ConstantDataService.find() | 返回最顶端单一持久化对象 |
| ConstantDataService.insert(ConstantPO po) | 插入单一持久化对象 |
| ConstantDataService.delect(ConstantPO po) | 删除单一持久化对象 |
| ConstantDataService.delect(String id) | 根据id删除单一持久化对象 |
| ConstantDataService.update(ConstantPO po) | 更新单一持久化对象 |

SalaryStrategyController和SalaryStrategy的接口规范如下表

SalaryStrategyController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SalaryStrategyController.getAllSalaryStrategy | 语法 | public ArrayList<SalaryVO> getAllSalaryStrategy(); |
| 前置条件 | 已创建新的SalaryStrategy领域对象 |
| 后置条件 | 调用SalaryStrategy的getAllSalaryStrategy方法 |
| SalaryStrategyController.newSalary | 语法 | public void newSalaryVO(); |
| 前置条件 | 已创建新的SalaryStrategy领域对象，新建薪水策略的请求出现 |
| 后置条件 | 调用SalaryStrategy对象的newSalary方法 |
| SalaryStrategyController.save | 语法 | public void save(SalaryVO vo); |
| 前置条件 | 已创建新的SalaryStrategy领域对象，新的薪水策略填写完整并填写符合规则 |
| 后置条件 | 调用SalaryStrategy对象的save方法 |
| SalaryStrateyController.endSalaryStrategy | 语法 | public void endSalaryStrategy(); |
| 前置条件 | 已创建新的SalaryStrategy领域对象 |
| 后置条件 | 调用SalaryStrategy对象的endSalaryStrategy方法 |
| SalaryStrategyController.saveChange | 语法 | public void saveChange(SalaryVO vo); |
| 前置条件 | 已创建新的SalaryStrategy领域对象，薪水策略修改完成并修改的信息符合规则 |
| 后置条件 | 调用SalaryStrategy对象的saveChange方法 |
| SalaryStrategyController.select | 语法 | public SalaryVO select(String id); |
| 前置条件 | 已创建新的SalaryStrategy领域对象，薪水策略被选择 |
| 后置条件 | 调用Salary对象的select方法 |
| SalaryStrategyController.revise | 语法 | public void revise(); |
| 前置条件 | 已创建新的SalaryStrategy领域对象，薪水策略被选择并请求修改 |
| 后置条件 | 调用SalaryStrategy的revise方法 |

**需要的服务（需接口）**

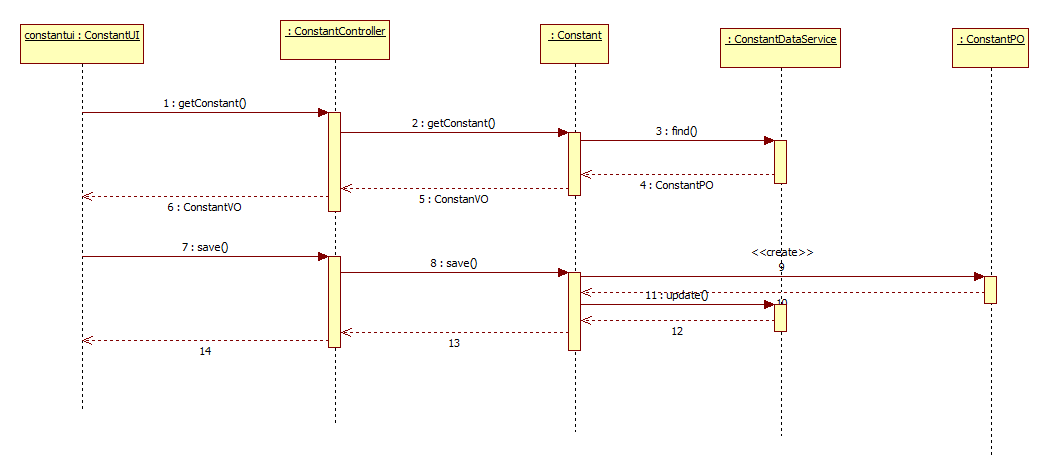
|  |  |
| --- | --- |
| **服务名** | **服务** |
| SalaryStrategy.getAllSalaryStrategy | **返回所有的薪水策略** |
| SalaryStrategy.newSalary | **返回新的薪水策略** |
| SalaryStrategy.save(SalaryVO vo) | **保存新的薪水策略** |
| SalaryStrategy.endSalaryStrategy | **结束制定薪水策略任务** |
| SalaryStrategy.saveChange(SalaryVO vo) | **保存修改的薪水策略** |
| SalaryStrategy.select | **返回所选择的薪水策略详细信息** |
| SalaryStrategy.revise | **返回可修改的薪水策略** |

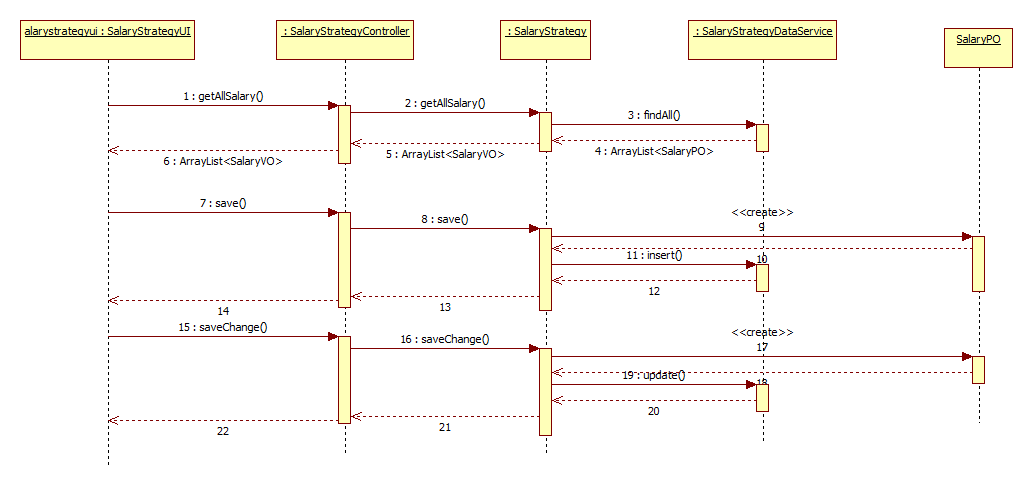
SalaryStrategy的接口规范

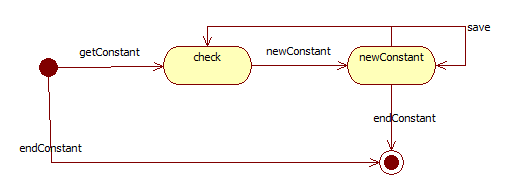
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SalaryStrategy.getAllSalaryStrategy | 语法 | public ArrayList<SalaryVO> getAllSalaryStrategy(); |
| 前置条件 | 开启新的一轮制定薪水策略 |
| 后置条件 | 返回先前的所制定的薪水策略 |
| SalaryStrategy.newSalary | 语法 | public void newSalaryVO(); |
| 前置条件 | 开启新的一轮制定薪水策略，制定新的薪水策略的按钮被点击 |
| 后置条件 | 返回薪水策略制定表 |
| SalaryStrategy.save | 语法 | public void save(SalaryVO vo); |
| 前置条件 | 在制定薪水策略的过程中，保存按钮被点击 |
| 后置条件 | 在一个制定薪水策略的回合中，持久化更新本次制定的薪水策略。 |
| SalaryStratey.endSalaryStrategy | 语法 | public void endSalaryStrategy(); |
| 前置条件 | 在制定薪水策略的过程中，结束按钮被点击 |
| 后置条件 | 在一个制定薪水策略的回合中，持久化更新本次制定的薪水策略，结束本次制定心塞策略回合 |
| SalaryStrategy.saveChange | 语法 | public void saveChange(SalaryVO vo); |
| 前置条件 | 在制定薪水策略的过程中，保存按钮被点击 |
| 后置条件 | 在一个制定薪水策略的回合中，持久化更新本次制定的薪水策略。 |
| SalaryStrategy.select | 语法 | public SalaryVO select(String id); |
| 前置条件 | 在制定薪水策略的过程中，薪水策略被选择 |
| 后置条件 | 在一个制定薪水策略的回合中，返回被选择的薪水策略详细详细信息 |
| SalaryStrategy.revise | 语法 | public void revise(); |
| 前置条件 | 在制定薪水策略的过程中，薪水策略已被选择 |
| 后置条件 | 在一个制定薪水策略的回合中，返回可修改的被选择的薪水策略详细详细信息 |

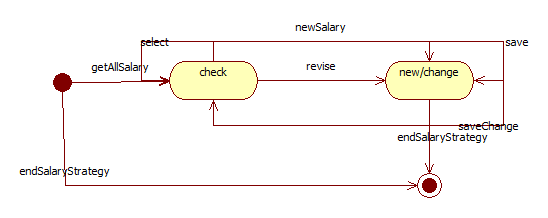
需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| SalaryStrategyDataService.findAll() | 返回所有持久化对象 |
| SalaryStrategyDataService.insert(SalaryPO po) | 插入单一持久化对象 |
| SalaryStrategyDataService.delect(SalaryPO po) | 删除单一持久化对象 |
| SalaryStrategyDataService.delect(String id) | 根据id删除单一持久化对象 |
| SalaryStrategyDataService.delect(ArrayList<SalaryPO >salarys) | 删除单一持久化对象 |
| SalaryStrategyDataService.update(SalaryPO po) | 更新单一持久化对象 |
| SalaryStrategyDataService.update(ArrayList<SalaryPO> salarys) | 删除多个持久化对象 |









(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有各自的控制器委托给不同的领域对象。

### 4.1.3auditsl模块

(1)模块描述

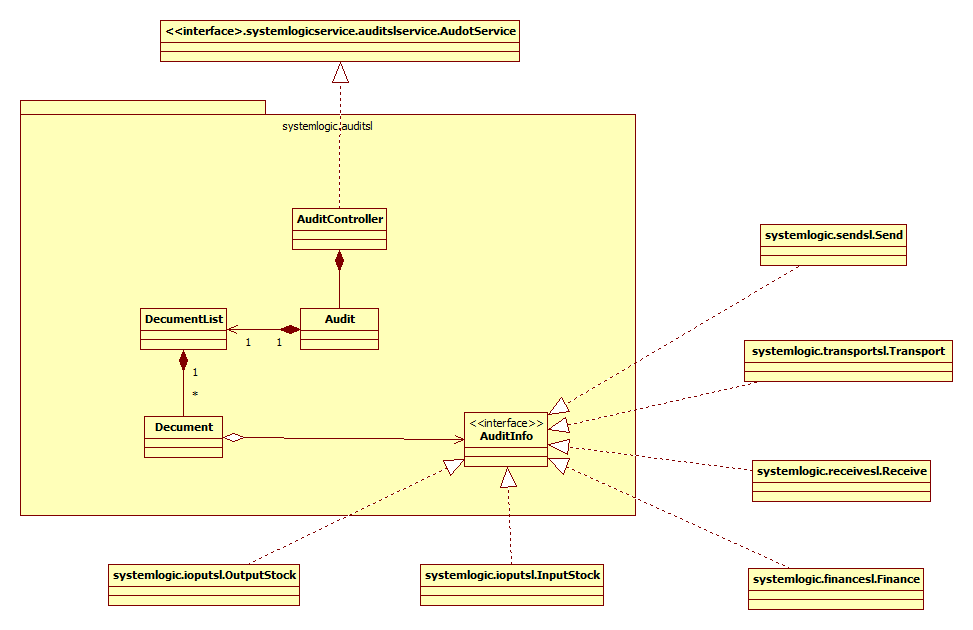
auditbl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

auditbl模块的职责及接口参见快递物流系统软件体系描述文档。

(2)整体结构

根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了auditservice一个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了AuditController，这样它们会将对审判单据的系统逻辑处理委托给对象Audit。Decument和DecumentList是需要审判的单据的容器类。DecumentPO是作为需要审判的单据的持久化对象的超类或者泛形添加到设计模型上去的。Auditsl模块的数据层是依赖于其他的需要审判的单据的dataservice。AuditInfo是为了消除循环依赖而产生的接口。

audtisl模块的设计如下图所示



(3)模块内部类的接口规范

auditsl模块的接口规范

AuditController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| AuditController.select | 语法 | Public DecumentVO select(String id); |
| 前置条件 | 已创建审判单据领域对象，待审判单据被选择 |
| 后置条件 | 调用Audit领域对象的select方法 |
| AuditController.adopt | 语法 | public void adopt(String id); |
| 前置条件 | 已创建审判单据领域对象，待审判单据被请求通过审判 |
| 后置条件 | 调用Audit领域对象的adopt方法 |
| AuditController.revise | 语法 | public void revise(String id); |
| 前置条件 | 已创建审判单据领域对象，修改请求 |
| 后置条件 | 调用Audit领域对象的revice方法 |
| AuditControllre.saveChange | 语法 | public void saveChange(DecumentVO vo); |
| 前置条件 | 已创建审判单据领域对象，单据内容已被修改，修改的内容符合规则，请求保存 |
| 后置条件 | 调用单据Audit领域对象的saveChange方法 |
| AuditController.selectNum | 语法 | public ArrayList<DecumentVO> selecNumt(String []ids); |
| 前置条件 | 已创建审判单据领域对象 |
| 后置条件 | 调用Audit领域对象的selectNum方法 |
| AuditController.getAllDecument | 语法 | Public ArrayList<DecumentVO> getDecumnet(); |
| 前置条件 | 已创建新的审判单据对象 |
| 后置条件 | 调用Audit领域对象的getAllDecument |
| AuditController.endAudit | 语法 | Public void endAudit(); |
| 前置条件 | 已创建审判单据领域对象 |
| 后置条件 | 调用Audit领域对象的endAudit方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| Audit.select(String id) | 返回待审判的单据 |
| Audit.adopt(String id) | 将单据的状态由排队中更改为审核后 |
| Audit.revise() | 返回单据可修改的信号 |
| Audit.saveChange(DecumentVO vo) | 保存修改后的单据内容 |
| Audit.selectNum(String [] ids) | 进行多单据审核 |
| Audit.getAllDecument() | 返回所有待审核单据 |
| Audit.endAudit() | 结束审判单据 |

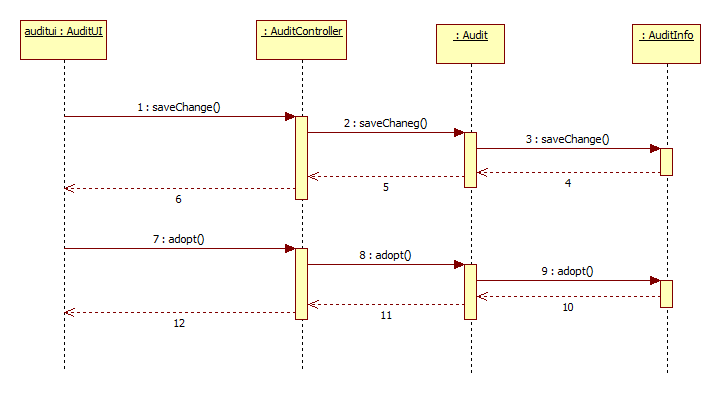
Audit的接口规范

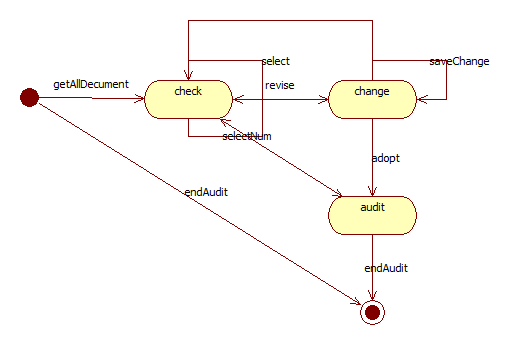
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Audit.select | 语法 | Public DecumentVO select(String id); |
| 前置条件 | 在审判单据的过程中，该单据被鼠标点击 |
| 后置条件 | 返回该单据的详细内容 |
| Audit.adopt | 语法 | public void adopt(String id); |
| 前置条件 | 在审判单据过程中，通过审核按钮被点击 |
| 后置条件 | 将单据的状态由由提交状态更改为审批后状态，持久化更新单据对象的数据，结束本次审判单据回合 |
| Audit.revise | 语法 | public void revise(String id); |
| 前置条件 | 在审判单据过程中，单据已经被选择，修改按钮被点击 |
| 后置条件 | 调用单据对象的修改方法 |
| Audit.saveChange | 语法 | public void saveChange(DecumentVO vo); |
| 前置条件 | 在修改单据的过程中，保存按钮被点击 |
| 后置条件 | 调用单据对象的保存方法。持久化更新单据对象的数据 |
| Audit.selectNum | 语法 | public ArrayList<DecumentVO> selecNumt(String []ids); |
| 前置条件 | 在审判单据的过程中，单据的多选按钮被点击 |
| 后置条件 | 记录所选择的单据 |
| Audit.getAllDecument | 语法 | Public ArrayList<DecumentVO> getDecumnet(); |
| 前置条件 | 开启新一轮的审判单据 |
| 后置条件 | 返回所有待审核单据的信息 |
| Audit.endAudit | 语法 | Public void endAudit(); |
| 前置条件 | 在审判单据的过程中，结束按钮被点击 |
| 后置条件 | 结束审判单据，持久化更新涉及的领域对象数据 |

需要的服务（需接口）

参见快递物流系统软件体系描述文档对于dataservice的描述

(4)业务逻辑层的动态模型





(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有各自的控制器委托给不同的领域对象。

### 4.1.4 reportsl模块

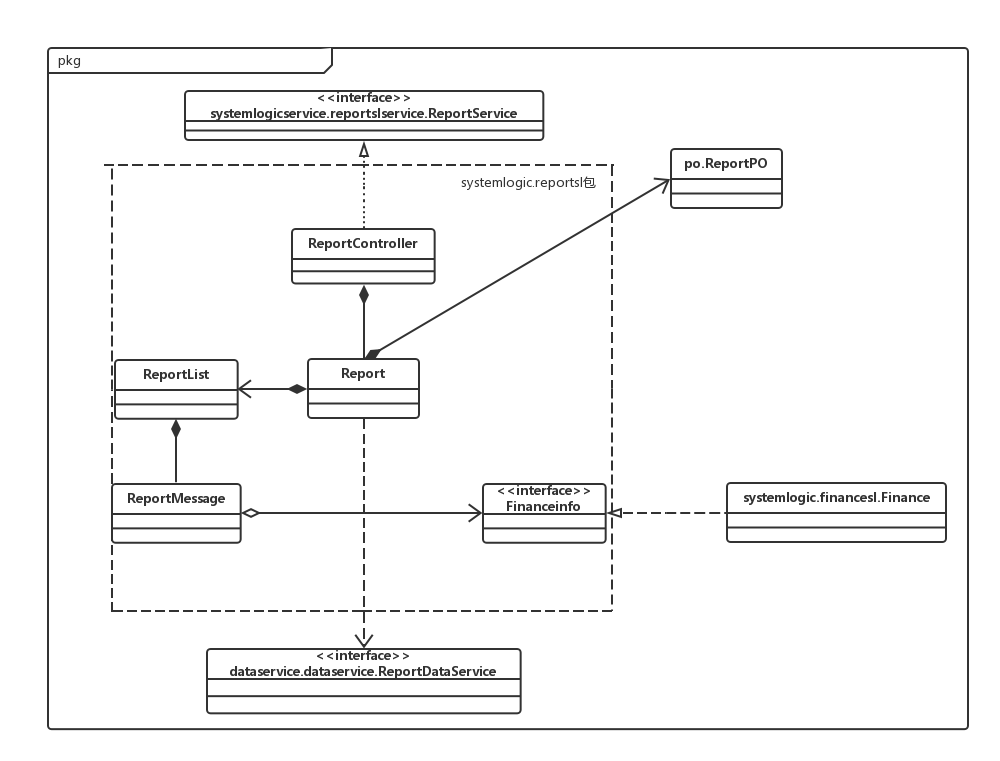
(1)模块概述

reportsl模块承担需求参见需求规格说明文档功能需求及相关非功能需求。

reportsl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.reportslservice.ReportService接口。业务逻辑层和数据层之间添加dataservice.dataservice.ReportDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了ReportController，这样ReportController会将对统计报表的业务逻辑处理委托给Report对象。ReportPO是作为统计报表记录的持久化对象被添加到设计模型中去的。而ReportList和ReportMessage的添加是Finance类的容器类。ReportMessage保有营业厅财务信息的数据，及相应的计算小计职责。而ReportList封装了关于ReportMessage的数据集合的数据结构的秘密和计算总收入，总支出，总利润等职责。FinanceInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。



Reportsl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 |
| ReportController | 负责实现统计报表界面所需要的服务 |
| Report | 统计报表的领域模型对象，拥有一次统计报表所持有的营业厅的总收入，总支出，总利润，时间，成本收益表，经营情况表等信息，可以帮助完成统计报表界面所需要的服务 |

(3)模块内部类的接口规范

ReportController的接口规范

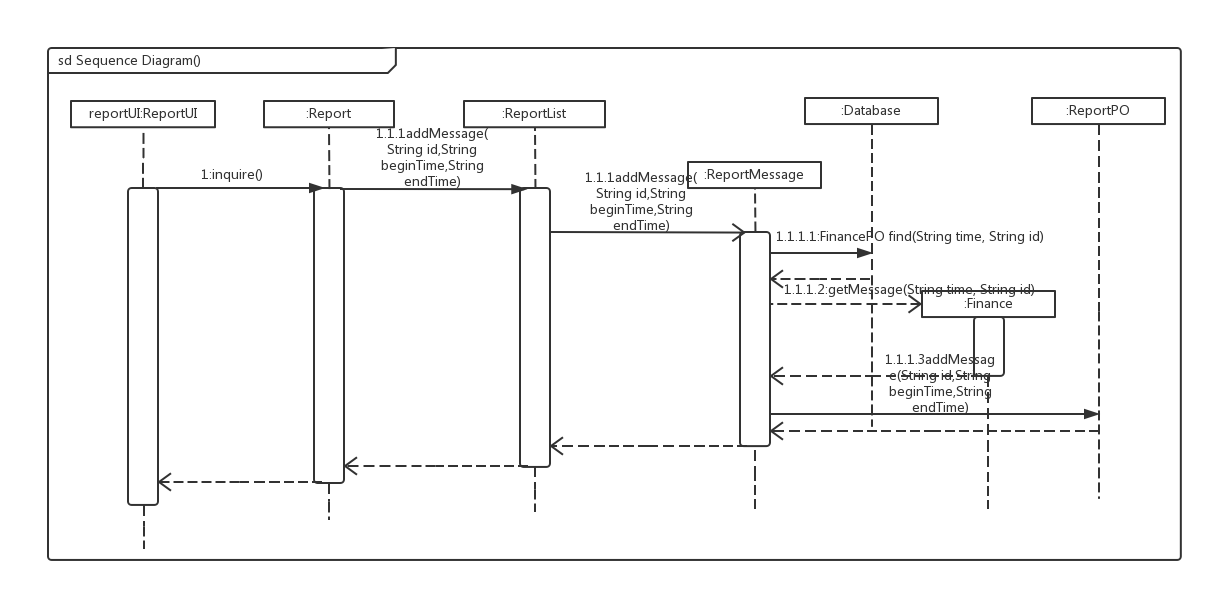
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| ReportController.inquire | 语法 | | public void inquire（） |
| 前置条件 | | 已创建一个Report领域对象 |
| 后置条件 | | 调用Report领域的inquire方法 |
| ReportController.addMessage | 语法 | | Public ReportVO addMessage(String id,String beginTime,String endTime) |
| 前置条件 | | 已创建一个Report领域对象，并且输入符合输入规则 |
| 后置条件 | | 调用Report领域的addMessage方法 |
| ReportController.exportReport | 语法 | | public void exportReport(ReportVO reportvo) |
| 前置条件 | | 已创建一个Report领域对象,已添加时间和营业厅，并且输入符合输入规则 |
| 后置条件 | | 调用Report领域的exportReport方法 |
| ReportController.printReport | 语法 | | public void printReport(ReportVO reportvo) |
| 前置条件 | | 已创建一个Report领域对象,已添加时间和营业厅，并且输入符合输入规则 |
| 后置条件 | | 调用Report领域的printReport方法 |
| ReportController.endReport | 语法 | | public void endReport（） |
| 前置条件 | | 已创建一个Report领域对象 |
| 后置条件 | | 调用Report领域的endReport方法 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| Report.inquire() | | 加入一个Report对象 | |
| Report.addMessage(String id,String beginTime,String endTime) | | 输入时间和营业厅，计算并生成相应的成本收益表或者经营情况表 | |
| Report.exportReport(ReportVO reportvo) | | 导出统计报表 | |
| Report.printReport(ReportVO reportvo) | | 打印统计报表 | |
| Report.void endReport() | | 结束统计报表 | |

Report模块的接口规范

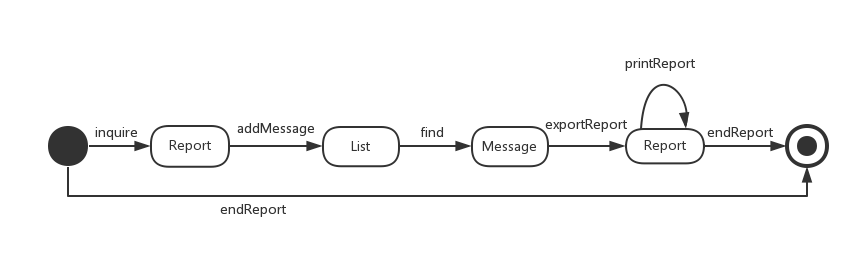
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Report.inquire | 语法 | public void inquire（） | |
| 前置条件 | 启动一个统计报表回合 | |
| 后置条件 | 系统请求输入起始时间、终止时间和被查询营业厅 | |
| Report.addMessage | 语法 | Public ReportVO addMessage(String id,String beginTime,String endTime) | |
| 前置条件 | 已请求查询统计报表 | |
| 后置条件 | 在一个统计报表回合中，增加起始时间、终止时间和被查询营业厅 | |
| Report.exportReport | 语法 | public void exportReport(ReportVO reportvo) | |
| 前置条件 | 已生成统计报表 | |
| 后置条件 | 导出统计报表 | |
| Report.printReport | 语法 | public void printReport(ReportVO reportvo) | |
| 前置条件 | 已生成统计报表 | |
| 后置条件 | 打印统计报表 | |
| Report.endReport | 语法 | public void endReport() | |
| 前置条件 | 已生产统计报表 | |
| 后置条件 | 结束此次统计报表回合 | |
| 需要的服务（需接口） | | | |
| 服务名 | | | 服务 |
| ReportDataService.find(String id) | | | 根据id进行查找单一持久化对象 |
| ReportDataService.finds(String field, String id) | | | 根据字段名和值进行查找多个持久化对象 |
| ReportDataService.insert(ReportPO po) | | | 插入单一持久化对象 |
| ReportDataService.delete(ReportPO po) | | | 删除单一持久化对象 |
| ReportDataService.update(ReportPO po) | | | 更新单一持久化对象 |
| DatabaseFactory.getReportDatabase | | | 得到Report数据库的服务的引用 |

(4)业务逻辑层的动态模型

下图表明了快递物流系统中，当用户输入营业厅和时间之后，统计报表业务逻辑处理的相关对象之间的合作。



下图描述了Report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)业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象

### 4.1.5Receivesl模块

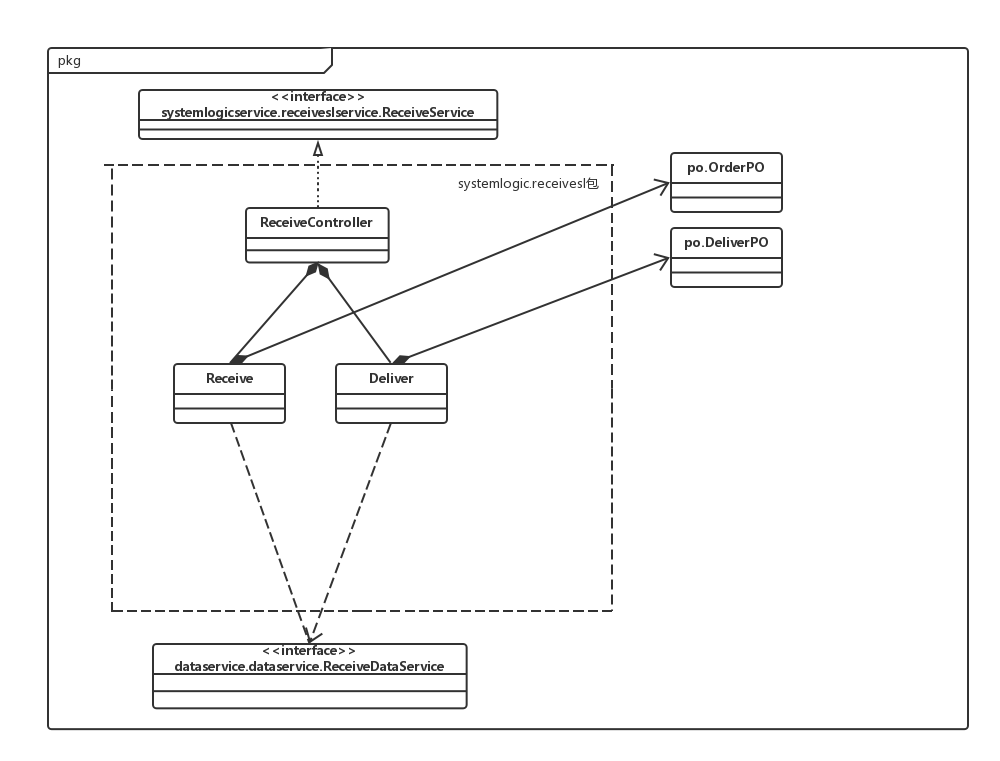
(1)模块概述

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

Receivesl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.receiveslservice.ReceiveService接口。业务逻辑层和数据层之间添加dataservice.dataservice.ReceiveDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了ReceiveController，这样ReceiveController会将对派件的业务逻辑处理委托给Receive和Deliver对象。OrderPO是作为派件记录的持久化对象被添加到设计模型中去的。



Receivesl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 |
| ReceiveController | 负责实现派件管理界面所需要的服务 |
| Receive | 收件的领域模型对象，拥有一次收件所持有的收件人和收件日期，订单等信息，可以帮助完成派件管理界面所需要的服务 |
| Deliver | 派件的领域模型对象，拥有一次派件所持有的派件人和派件日期，派件单等信息，可以帮助完成派件管理界面所需要的服务 |

(3)模块内部类的接口规范

ReceiveController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ReceiveController.addExpress | 语法 | public OrderVO addExpress(String name,String time, String orderNumber) |
| 前置条件 | 已创建一个Receive领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Receive领域对象的addExpress方法 |
| ReceiveController.endReceive | 语法 | public void endReceive() |
| 前置条件 | 已创建一个Receive领域对象 |
| 后置条件 | 调用Receive领域对象的endReceive方法 |
| ReceiveController.saveExpress | 语法 | public void saveExpress(OrderVO ordervo) |
| 前置条件 | 已创建一个Receive领域对象，已添加订单信息 |
| 后置条件 | 调用Receive领域对象的saveExpress方法 |
| ReceiveController.newDeliver | 语法 | public DeliverVO newDeliver(String time) |
| 前置条件 | 已创建一个Deliver领域对象 |
| 后置条件 | 调用Deliver领域对象的newDeliver方法 |
| ReceiveController.printDeliver | 语法 | public void printDeliver(String deliverNumber) |
| 前置条件 | 已创建一个Deliver领域对象 |
| 后置条件 | 调用Deliver领域对象的printDeliver方法 |
| ReceiveController.endReceive | 语法 | public void endReceive() |
| 前置条件 | 已创建一个Deliver领域对象 |
| 后置条件 | 调用Deliver领域对象的endReceive方法 |
| ReceiveController.getDeliver | 语法 | public DeliverVO getDeliver(String deliverNumber) |
| 前置条件 | 已创建一个Deliver领域对象 |
| 后置条件 | 调用Deliver领域对象的getDeliver方法 |
| ReceiveController.saveDeliver | 语法 | public void saveDeliver(DeliverVO delivervo) |
| 前置条件 | 已创建一个Deliver领域对象，已生成派件单 |
| 后置条件 | 调用Deliver领域对象的saveDeliver方法 |

|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| Receive.addExpress(String name,String time, String orderNumber) | 添加一个订单对象 |
| Receive.endReceive() | 结束派件管理 |
| Receive.saveExpress(OrderVO ordervo) | 保存收件信息 |
| Deliver.newDeliver(String time) | 生成派件单 |
| Deliver.printDeliver(String deliverNumber) | 打印派件单 |
| Deliver.getDeliver(String deliverNumber) | 查找并返回派件单 |
| Deliver.saveDeliver(DeliverVO delivervo) | 保存派件单 |
| Deliver.endReceive() | 结束派件管理 |

Receive模块的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Receive.addExpress | 语法 | public OrderVO addExpress(String name,String time, String orderNumber) |
| 前置条件 | 启动一个派件回合 |
| 后置条件 | 在一个派件回合内，增加快件号，收件人，收件时间（系统自动记录） |
| Receive.endReceive | 语法 | public void endReceive() |
| 前置条件 | 派件信息已提交 |
| 后置条件 | 结束此次派件回合，持久化更新涉及的领域对象的数据 |
| Receive.saveExpress | 语法 | public void saveExpress(OrderVO ordervo) |
| 前置条件 | 已添加收件信息 |
| 后置条件 | 系统保存收件信息 |

|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| ReceiveDataService.findO(String orderNumber) | 根据orderNumber进行查找单一持久化对象 |
| ReceiveDataService.findsO(String field, String orderNumber) | 根据字段名和值进行查找多个持久化对象 |
| ReceiveDataService.insertO(OrderPO po) | 插入单一持久化对象 |
| ReceiveDataService.deleteO(OrderPO po) | 删除单一持久化对象 |
| ReceiveDataService.updateO(OrderPO po) | 更新单一持久化对象 |
| DatabaseFactory.getReceiveDatabase | 得到Receive数据库的服务的引用 |

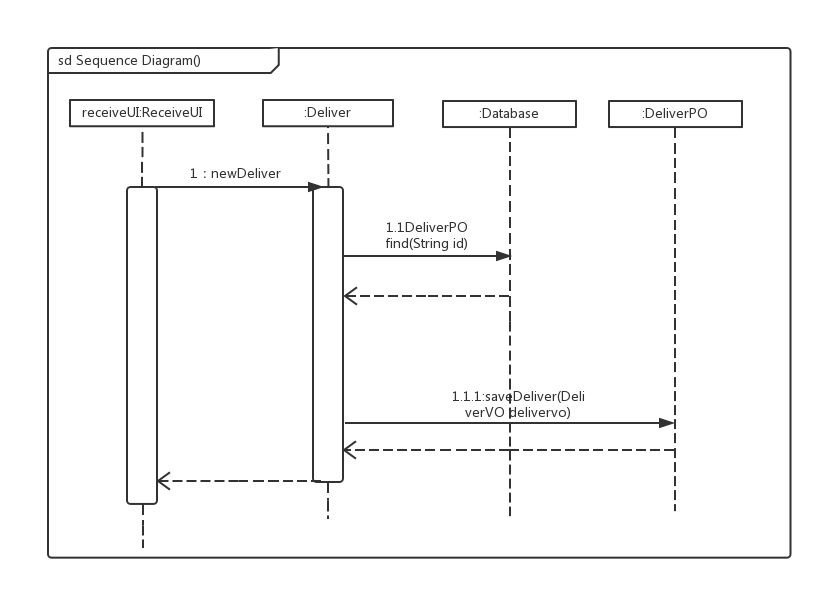
Deliver模块的接口规范

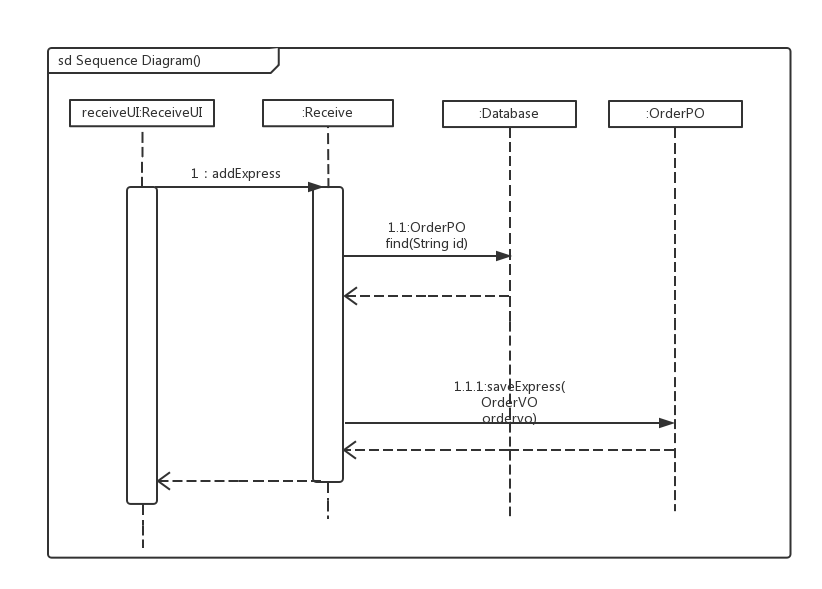
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Deliver.newDeliver | 语法 | public DeliverVO newDeliver(String time) |
| 前置条件 | 无 |
| 后置条件 | 返回新的派件单信息 |
| Deliver.printDeliver | 语法 | public void printDeliver(String deliverNumber) |
| 前置条件 | 已生成新的派件单 |
| 后置条件 | 打印派件单 |
| Deliver.endReceive | 语法 | public void endReceive() |
| 前置条件 | 派件信息已提交 |
| 后置条件 | 结束此次派件回合，持久化更新涉及的领域对象的数据 |
| Deliver.getDeliver | 语法 | public DeliverVO getDeliver(String deliverNumber) |
| 前置条件 | 启动一个派件回合，deliverNumber符合输入规则 |
| 后置条件 | 返回派件单信息 |
| Deliver.saveDeliver | 语法 | public void saveDeliver(DeliverVO delivervo) |
| 前置条件 | 已生成派件单 |
| 后置条件 | 系统保存派件单 |

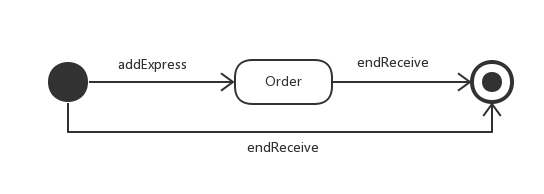
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| ReceiveDataService.findD(String deliverNumber) | 根据deliverNumber进行查找单一持久化对象 |
| ReceiveDataService.findsD(String field, String deliverNumber) | 根据字段名和值进行查找多个持久化对象 |
| ReceiveDataService.insertD(DeliverPO po) | 插入单一持久化对象 |
| ReceiveDataService.deleteD(DeliverPO po) | 删除单一持久化对象 |
| ReceiveDataService.updateD(DeliverPO po) | 更新单一持久化对象 |
| DatabaseFactory.getReceiveDatabase | 得到Receive数据库的服务的引用 |

(4)业务逻辑层的动态模型

下图表明了快递物流系统中，当用户查询派件单之后，派件管理业务逻辑处理的相关对象之间的协作。

 下图表明了快递物流系统中，当用户收件之后，派件管理业务逻辑处理的相关对象之间的协作。

 下图描述了Receive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。



下图描述了Deliver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)业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象

### 4.1.6transportsl模块

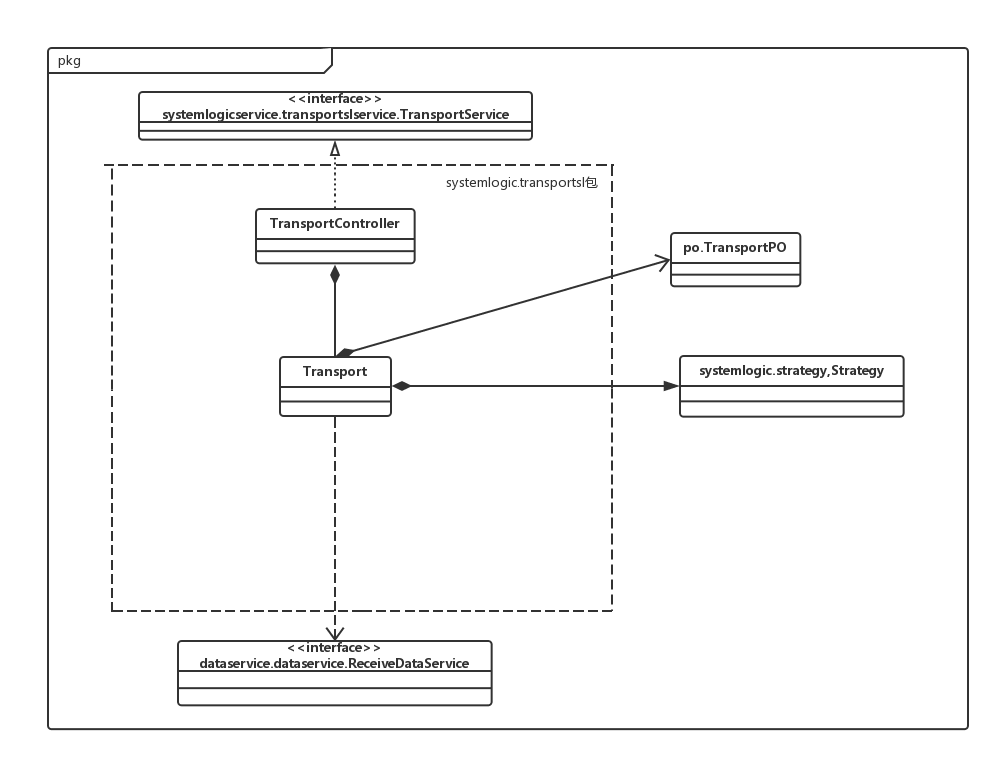
(1)模块概述

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

transportsl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.transportslservice.TransportService接口。业务逻辑层和数据层之间添加dataservice.dataservice.TransportDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了TransportController，这样TransportController会将对派件的业务逻辑处理委托给Transport对象。TransportPO是作为派件记录的持久化对象被添加到设计模型中去的。



(3)模块内部类的接口规范

TransportController模块的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| TransportController.addMember | 语法 | public TransportVO addMember(String id) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的addMember方法 |
| TransportController.addExpress | 语法 | public TransportVO addExpress(String orderNumber) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的addExpress方法 |
| TransportController.addMessage | 语法 | public TransportVO addMessage(String id, String beginTime, String endTime) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的addMessage方法 |
| TransportController.addTraffic | 语法 | public TransportVO addTraffic(String id) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的addTraffic方法 |
| TransportController.addCondition | 语法 | public TransportVO addMember(String orderNumber, String condition) |
| 前置条件 | 已创建一个Transport领域对象，已添加订单信息，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的addCondition方法 |
| TransportController.getTransport | 语法 | public TransportVO getTransport(String transportid) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的getTransport方法 |
| TransportController.endTransport | 语法 | public void endTransport() |
| 前置条件 | 已创建一个Transport领域对象 |
| 后置条件 | 调用Transport领域的endTransport方法 |
| TransportController.printTransport | 语法 | public void getTransport(String id) |
| 前置条件 | 已创建一个Transport领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Transport领域的printTransport方法 |
| TransportController.saveTransport | 语法 | public void saveTransport(TransportVO transportvo) |
| 前置条件 | 已创建一个Transport领域对象 |
| 后置条件 | 调用Transport领域的saveTransport方法 |
| TransportController.choose | 语法 | public TransportVO choose(String sign) |
| 前置条件 | 已创建一个Transport领域对象 |
| 后置条件 | 调用Transport领域的choose方法 |

|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| Transport.addMember(String id) | 添加相关人员编号 |
| Transport.addExpress(String orderNumber) | 添加一个订单对象 |
| Transport.addMessage(String id, String beginTime, String endTime) | 添加运输单信息（营业厅编号，时间） |
| Transport.addTraffic(String id) | 添加汽运编号 |
| Transport.addMember(String orderNumber, String condition)po) | 添加订单的货运到达状态 |
| Transport.getTransport(String transportid) | 查找并返回运输单 |
| Transport.endTransport() | 结束装运管理 |
| Transport.printTransport(String id) | 打印运输单 |
| Transport.saveTransport(TransportVO transportvo) | 保存运输单 |
| Transport.choose(String sign) | 选择运输单 |

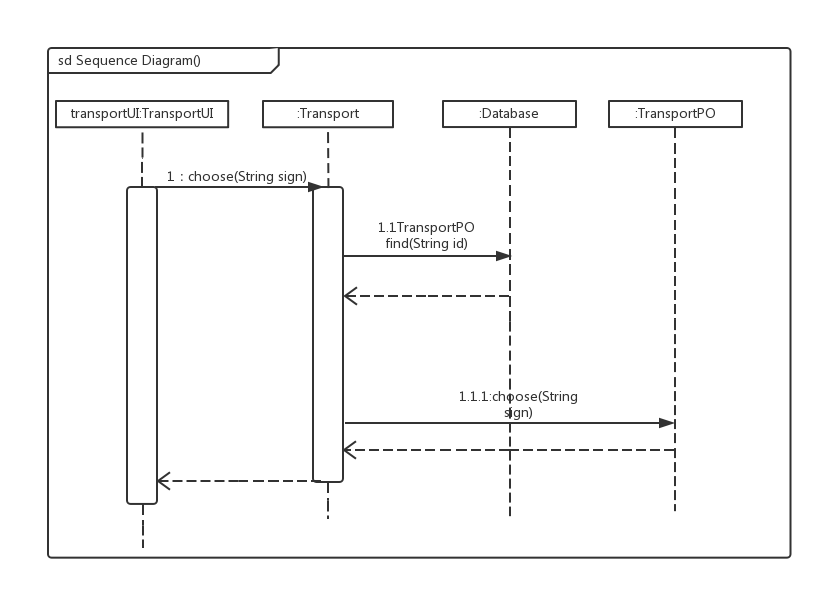
Transport模块的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Transportsl.addMember | 语法 | public TransportVO addMember(String id) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，增加相关人员信息（押运员，接收员，监装员，司机编号） |
| Transportsl.addExpress | 语法 | public TransportVO addExpress(String orderNumber) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，增加相关订单信息 |
| Transportsl.addMessage | 语法 | public TransportVO addMessage(String id, String beginTime, String endTime) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，增加营业厅或中转中心编号，时间 |
| Transportsl.addTraffic | 语法 | public TransportVO addTraffic(String id) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，增加运输工具信息（汽运，车辆编号） |
| Transportsl.addCondition | 语法 | public TransportVO addMember(String orderNumber, String condition) |
| 前置条件 | 已输入商品 |
| 后置条件 | 增加指定货物的到达状态 |
| Transportsl.getTransport | 语法 | public TransportVO getTransport(String transportid) |
| 前置条件 | 启动一个运输回合，运输单编号符合输入规则 |
| 后置条件 | 在一个运输回合中，返回运输单信息 |
| Transportsl.endTransport | 语法 | public void endTransport() |
| 前置条件 | 无 |
| 后置条件 | 结束此次运输回合，持久化更新涉及的领域对象的数据 |
| Transportsl.printTransport | 语法 | public void printTransport(String id) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，打印运输单 |
| Transportsl.saveTransport | 语法 | public void saveTransport(TransportVO transportvo) |
| 前置条件 | 启动一个运输回合，已生成运输单 |
| 后置条件 | 在一个运输回合中，保存运输单信息 |
| Transportsl.choose | 语法 | public TransportVO choose(String sign) |
| 前置条件 | 启动一个运输回合 |
| 后置条件 | 在一个运输回合中，返回指定的运输单 |

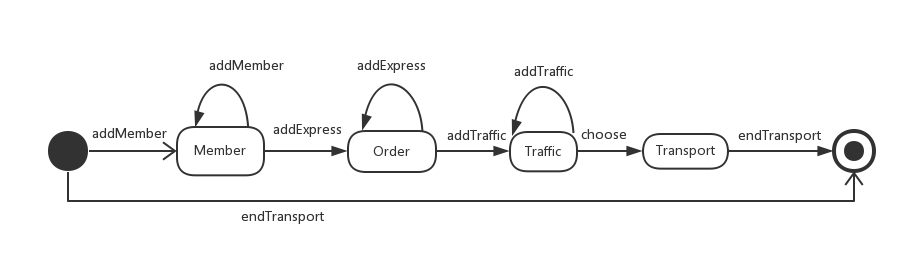
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| TransportDataService.find(String id) | 根据id进行查找单一持久化对象 |
| TransportDataService.finds(String field, String id) | 根据字段名和值进行查找多个持久化对象 |
| TransportDataService.insert(TransportPO po) | 插入单一持久化对象 |
| TransportDataService.delete(TransportPO po) | 删除单一持久化对象 |
| TransportDataService.update(TransportPO po) | 更新单一持久化对象 |
| DatabaseFactory.getTransportDatabase | 得到Transport数据库的服务的引用 |

(4)业务逻辑层的动态模型

下图表明了快递物流系统中，当用户选择运输单之后，运输管理业务逻辑处理的相关对象之间的协作。



下图描述了Transport对象的生存期间的状态序列、引起转移的视角，以及因状态转移而伴随的动作。



(5)业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

### 4.1.7checksl模块

(1)模块概述

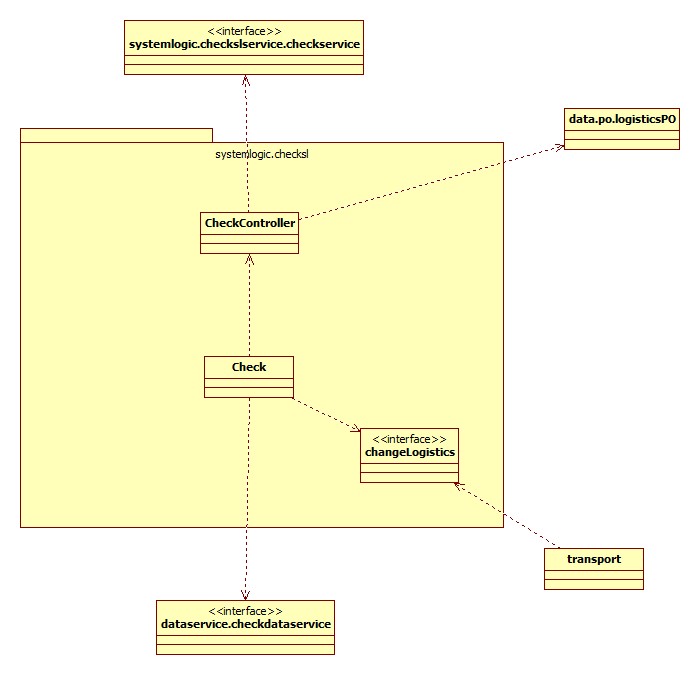
checksl模块承担需求参见需求规格说明文档功能需求及相关非功能需求。

checksl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.checkslservice.CheckService接口。业务逻辑层和数据层之间添加dataservice.dataservice.CheckDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了CheckController，这样CheckController会将对统计报表的业务逻辑处理委托给Check对象。LogisticsPO是作为统计报表记录的持久化对象被添加到设计模型中去的。

checksl模块的图如下图所示：



checksl模块的各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| CheckController | 负责实现物流信息查询界面所需要的服务 |
| Check | 物流信息查询界面的领域模型对象，拥有一次物流信息查询所持有的物流信息、操作记录等信息，可以帮助完成物流信息查询界面所需要的服务 |

(3)模块内部类的接口规范

checkcontroller的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| CheckController.orderMessageCheck | 语法 | public LogisticsVO  orderNumberCheck(String oredernumber); |
| 前置条件 | 已经创建一个Check领域对象，ordernumber符合输入规则 |
| 后置条件 | 调用Check领域的ordeNumberCheck方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| Check.orderNumberCheck(String ordernumber) | 进行一次物流信息查询 |

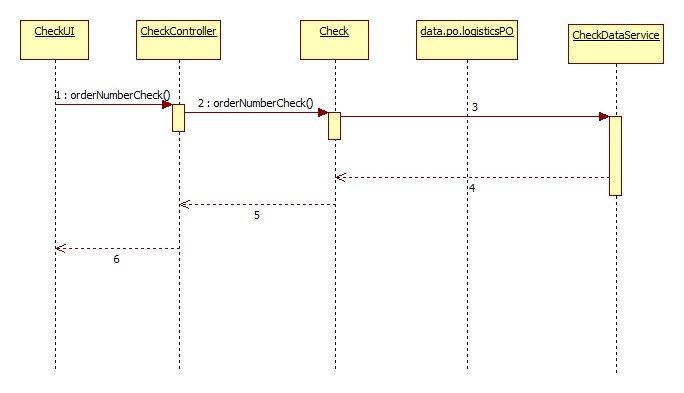
checksl模块的接口规范

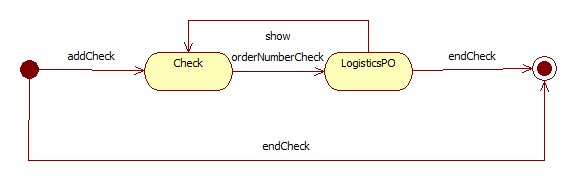
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Check.orderMessageCheck | 语法 | public LogisticsVO  orderNumberCheck(String oredernumber); |
| 前置条件 | ordernumber符合输入规则 |
| 后置条件 | 查找是否存在响应的OrderMessage,根据输入的ordernumber返回所对应的物流信息 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getCheckDatabase | 得到Check数据库服务的引用 |
| CheckDataService.find(String ordernumber) | 根据ordernumber进行查找单一持久化对象 |
|  |  |

(4)业务逻辑层的动态模型





(5)业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象

### 4.1.8financesl模块

(1)模块概述

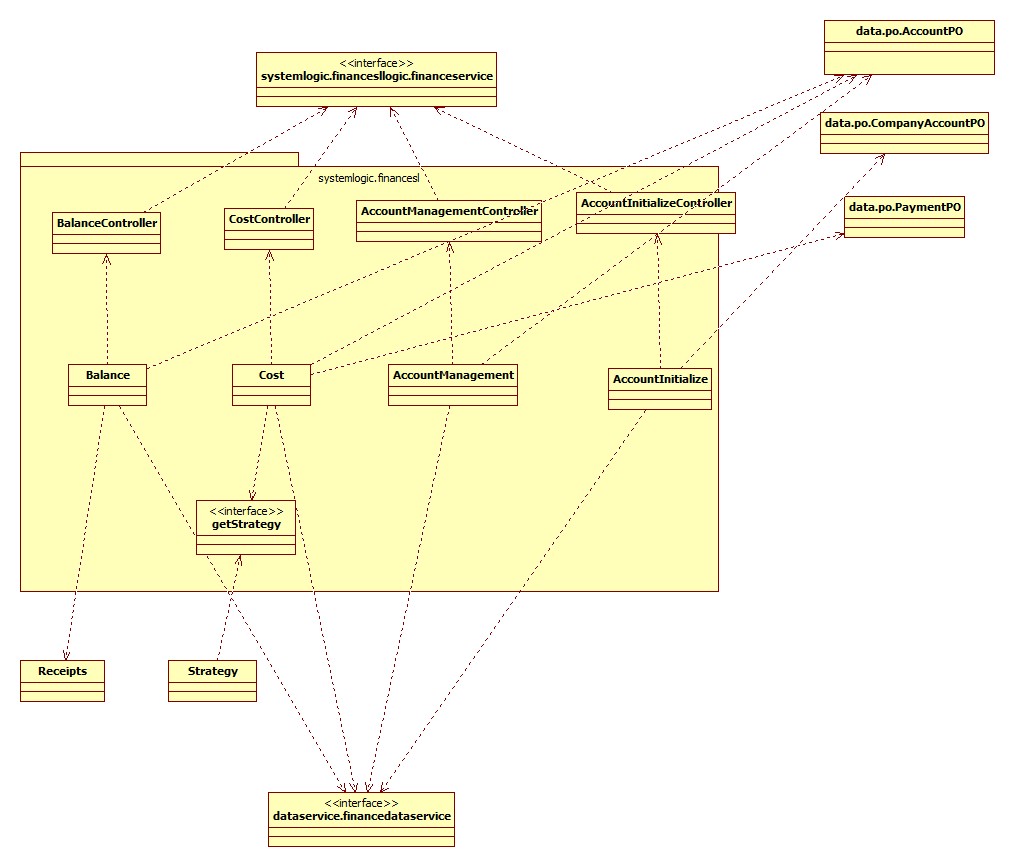
financesl模块承担需求参见需求规格说明文档功能需求及相关非功能需求。

financesl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.receiveslservice.FinanceService接口。业务逻辑层和数据层之间添加dataservice.dataservice.FinanceDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了BalanceController,CostController,AccountManagementController,AccountInitializeController，这样FinanceController会将对结算管理、成本管理、账户管理和账目初始化的业务逻辑处理委托给Balance、Cost、AccountManagement、AccountInitialize对象。AccountPO、CompanyAccountPO、PaymentPO是作为财务模块记录的持久化对象被添加到设计模型中去的。

Financesl模块的图如下图所示：



financesl模块的各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| BalanceController | 负责实现结算管理界面所需要的服务 |
| CostController | 负责实现成本管理界面所需要的服务 |
| AccountManagementController | 负责实现账户管理界面所需要的服务 |
| AccountInitializeController | 负责实现账目初始化界面所需要的服务 |
| Balance | 结算管理界面的领域模型对象，拥有一次结算管理所持有的收款单、操作记录等信息，可以帮助完成结算管理界面所需要的服务 |
| Cost | 成本管理界面的领域模型对象，拥有一次成本管理所持有的付款单、操作记录等信息，可以帮助完成成本管理界面所需要的服务 |
| AccountManagement | 账户管理界面的领域模型对象，拥有一次账户管理所持有的账户列表、操作记录等信息，可以帮助完成账户管理所需要的服务 |
| AccountInitialize | 账目初始化界面的领域模型对象，拥有一次账目初始化所持有的账目、操作记录等信息，可以帮助完成账目初始化界面所需要的服务 |
| LoginController | 负责实现对应于登录界面所需要的服务 |

1. 模块内部类的接口规范

financesl模块的controller接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| BalanceController.getBalanceMessage | 语法 | public ArrayList<ReceiptsVO> getBalanceMessage(String office,String starttime,String endtime); |
| 前置条件 | 已创建一个Balance领域对象，输入的时间段符合输入规则 |
| 后置条件 | 调用Balance领域里的getBalanceMessage方法 |
| CostControllert.setPayment | 语法 | public PaymentVO setPayment(AccountVO account,String paymentType,StaffVO receiver); |
| 前置条件 | 已创建一个Cost领域对象，输入的账户、款项类型、收款人符合输入规则; |
| 后置条件 | 调用Cost领域里的setPayment方法 |
| CostController.computePayment | 语法 | public PaymentVO computePayment(PaymentVO payment); |
| 前置条件 | 已创建一个Cost领域对象，输入款项类型为工资 |
| 后置条件 | 调用Cost领域里的computePayment方法 |
| CostController.payPayment | 语法 | public void payPayment(Payment payment); |
| 前置条件 | 已创建一个Cost领域对象 |
| 后置条件 | 调用Cost领域里的payPayment方法 |
| CostController.computePayment | 语法 | public PaymentVO computePayment(Payment payment,double money); |
| 前置条件 | 已创建一个Cost领域对象，输入款项类型为奖金或租金 |
| 后置条件 | 调用Cost领域里的computePayment方法 |
| AccountManagementController.addAccount | 语法 | public Account addAccount(String accountname,double accountmoney); |
| 前置条件 | 已创建一个AccountManagement领域对象，输入的账户名和账户余额符合输入规则 |
| 后置条件 | 调用AccountManagement领域里的addAccount方法 |
| AccountManagementController.deleteAccount | 语法 | public ResultMessage deleteAccount(Account account); |
| 前置条件 | 已创建一个AccountManagement领域对象 |
| 后置条件 | 调用AccountManagement领域里的deleteAccount方法 |
| AccountManagementController.fixAccount | 语法 | public Account fixAccount(Account account,String accountname); |
| 前置条件 | 已创建一个AccountManagement领域对象,，输入符合规范 |
| 后置条件 | 调用AccountManagement领域里的fixAccount方法 |
| AccountManagementController.seekAccount | 语法 | public ArrayList<AccountPO> seekAccount(String accountname); |
| 前置条件 | 已创建一个AccountManagement领域对象 |
| 后置条件 | 调用AccountManagement领域里的seekAccount方法 |
| AccountInitializeController.  initialize | 语法 | public CompanyAccount initialize(CompanyAccount oldaccount,String customer,String agency,String people,String stock,String bankaccount); |
| 前置条件 | 已创建一个AccountInitialize领域对象 |
| 后置条件 | 调用AccountInitialize领域里的initializet方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| Balance.getBalanceMessage(String office,String starttime,String endtime) | 获得收款单信息 |
| Cost.setPayment(AccountVO account,String paymentType,StaffVO receiver) | 根据输入建立付款单 |
| Cost.computePayment(PaymentVO payment) | 根据策略，计算付款单 |
| Cost.payPayment(Payment payment) | 支付付款单，计算账户余额等 |
| Cost.computePayment(Payment payment,double money) | 在支付款项类型为奖金和租金时，人工输入支付款项数目 |
| AccountManagement.addAccount(String accountname,double accountmoney) | 新增账户 |
| AccountManagement.deleteAccount(Account account) | 删除账户 |
| AccountManagement.fixAccount(Account account,String accountname) | 修改账户 |
| AccountManagement.seekAccount(String accountname) | 查询账户 |
| AccountInitialize.  initialize(CompanyAccount oldaccount,String customer,String agency,String people,String stock,String bankaccount) | 账目初始化，根据输入和以往账目信息，新建账目 |

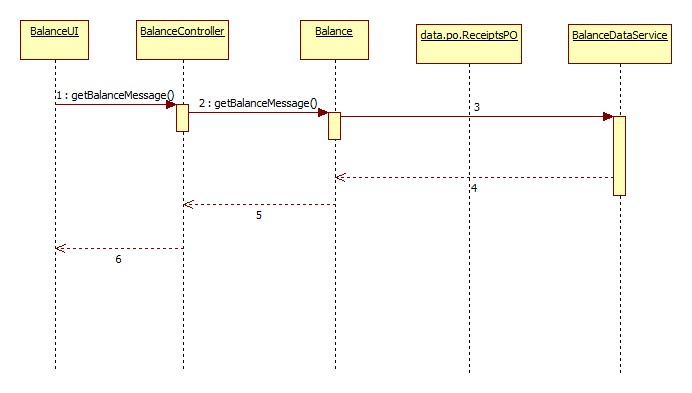
financesl模块的接口规范

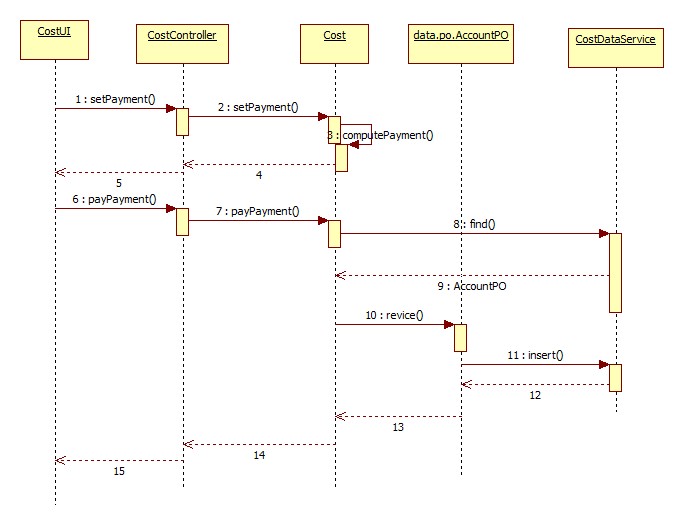
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Balance.getBalanceMessage | 语法 | public ArrayList<ReceiptsVO> getBalanceMessage(String office,String starttime,String endtime); |
| 前置条件 | 选择的营业厅，输入的时间段符合输入规则 |
| 后置条件 | 返回该条件下的结算详细内容 |
| Cost.setPayment | 语法 | public PaymentVO setPayment(AccountVO account,String paymentType,StaffVO receiver); |
| 前置条件 | 输入的账户、款项类型、收款人符合输入规则; |
| 后置条件 | 返回一个填写了初步信息的Payment |
| Cost.computePayment | 语法 | public PaymentVO computePayment(PaymentVO payment); |
| 前置条件 | 已经通过setPayment设置了payment的初步信息；输入的款项类型为工资 |
| 后置条件 | 返回一个拥有应支付款项信息的Payment |
| Cost.payPayment | 语法 | public void payPayment(Payment payment); |
| 前置条件 | 已经通过computePayment计算了所需支付款项; |
| 后置条件 | 根据所选账户支付所需支付款项 |
| Cost.computePayment | 语法 | public PaymentVO computePayment(Payment payment,double money); |
| 前置条件 | 已经通过setPayment设置了payment的初步信息；输入的款项类型为奖金或租金 |
| 后置条件 | 返回一个拥有应支付款项信息的Payment |
| AccountManagement.addAccount | 语法 | public Account addAccount(String accountname,double accountmoney); |
| 前置条件 | 输入的账户名和账户余额符合输入规则 |
| 后置条件 | 返回一个新的Account对象 |
| AccountManagement.selectAccount | 语法 | public Account selectAccount(String accountname); |
| 前置条件 | 在账户管理的过程中，某个账户被选中 |
| 后置条件 | 返回一个所选中的账户的信息 |
| AccountManagement.deleteAccount | 语法 | public ResultMessage deleteAccount(Account account); |
| 前置条件 | 在选中账户后，进行了账户删除操作 |
| 后置条件 | 在数据库中删除相对应的账户信息 |
| AccountManagement.fixAccount | 语法 | public Account fixAccount(Account account,String accountname); |
| 前置条件 | 在选中账户后，进行了账户修改操作 |
| 后置条件 | 在数据库中更新相对应的账户信息 |
| AccountManagement.seekAccount | 语法 | public ArrayList<AccountPO> seekAccount(String accountname); |
| 前置条件 | 在账户管理页面，进行了账户查找信息 |
| 后置条件 | 返回一个符合查询信息的Account列表 |
| AccountInitialize.  initialize | 语法 | public CompanyAccount initialize(CompanyAccount oldaccount,String customer,String agency,String people,String stock,String bankaccount); |
| 前置条件 | 在账目初始化页面，进行了新建账目操作 |
| 后置条件 | 返回一个新建的账目 |

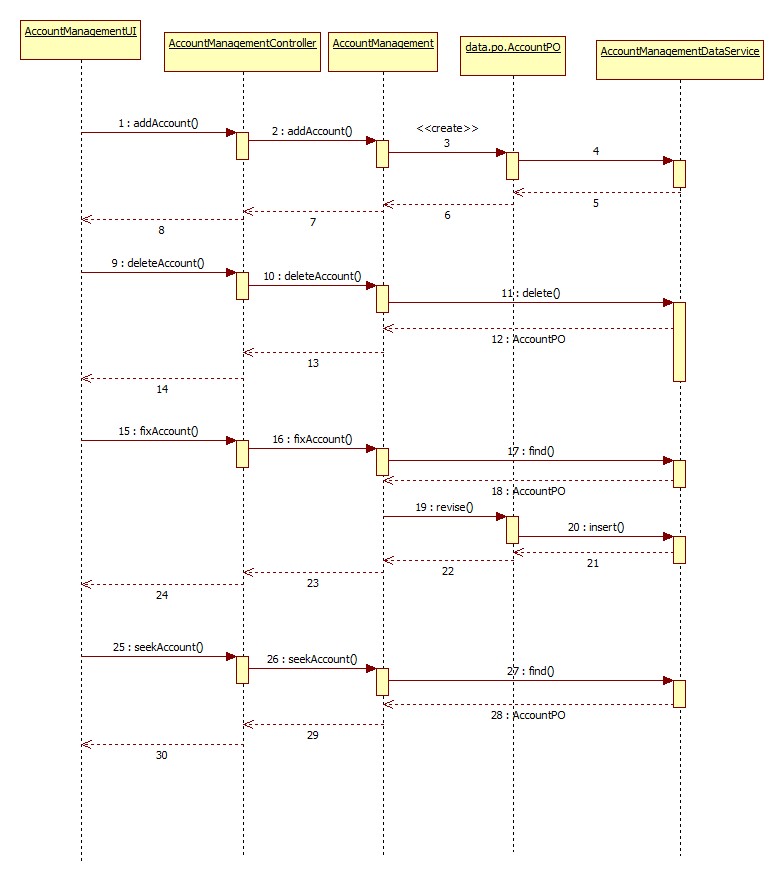
需要的服务（需接口）

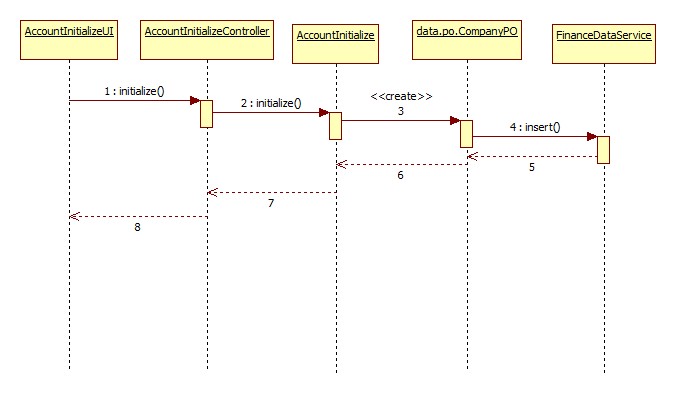
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getFinanceDatabase | 得到finance数据库服务的引用 |
| FinanceDataService.find(String id) | 根据id进行查找单一持久化对象 |
| FinanceDataService.finds(String[]ids) | 根据ids查找多个持久化对象 |
| FinanceDataService.insert(FinancePO PO) | 插入单一持久化对象 |
| FinanceDataService.delect(FinancePO PO) | 删除单一持久化对象 |

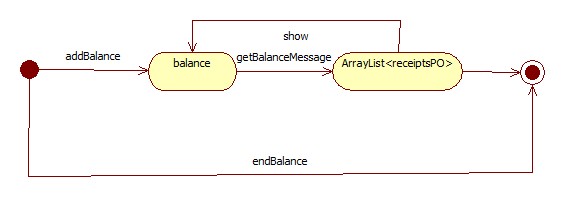
(4)业务逻辑层的动态模型

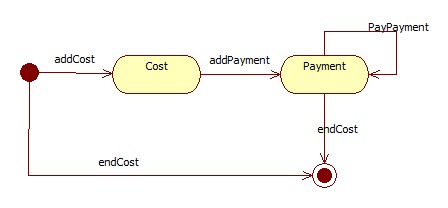


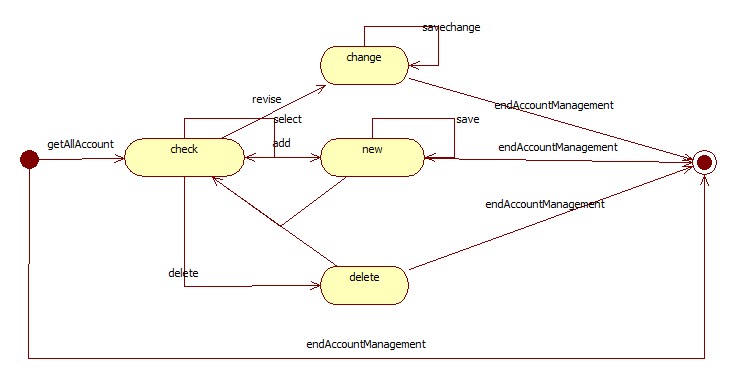


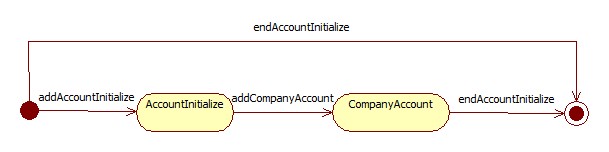












（5）业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

### 4.1.9usersl模块

(1)模块概述

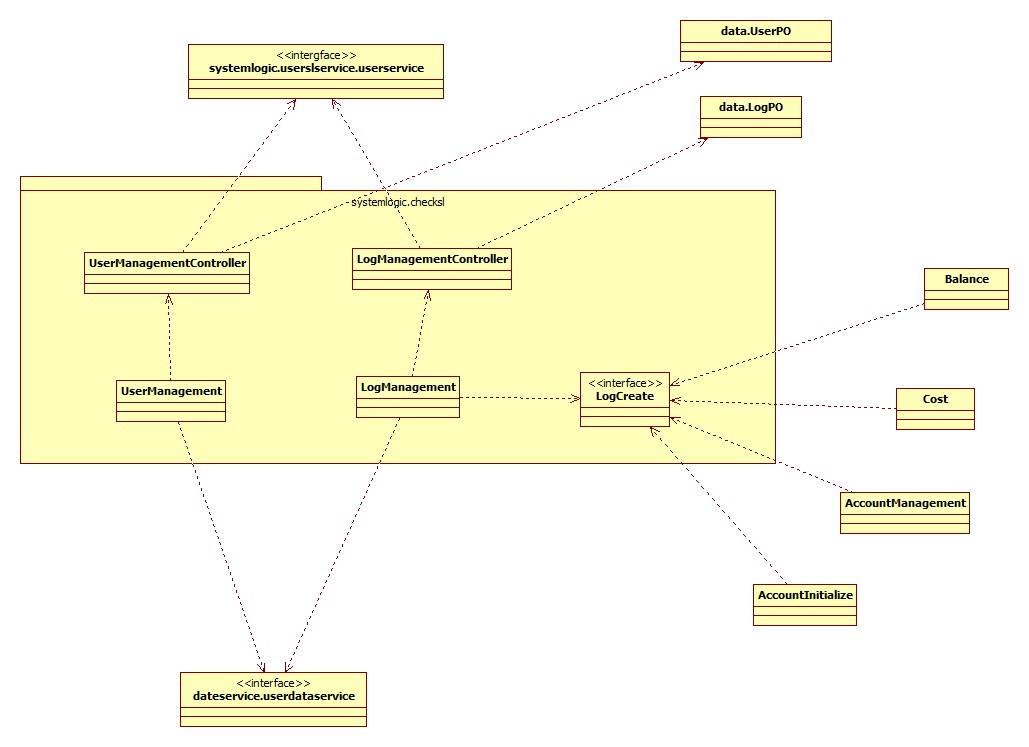
usersl模块承担需求参见需求规格说明文档功能需求及相关非功能需求。

usersl模块的职责及接口参见软件系统结构描述文档。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加systemlogicservice.usersklservice.Userservice接口。业务逻辑层和数据层之间添加dataservice.dataservice.UserDataService接口。为了隔离业务逻辑层和逻辑控制职责，我们增加了UserMangementController,LogManagementController，这样UserMangementController和LogManagementController会将对用户管理和日志管理的业务逻辑处理委托给UserMangement、LogManagement对象。UserPO、LogPO是作为用户模块记录的持久化对象被添加到设计模型中去的。

usersl模块的图如下图所示：其中实现logCreate接口以finance模块为例，实际上所有系统操作都要实现这个接口，为了简便只画出部分。



Usersl模块的各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserManagementController | 负责实现用户管理管理界面所需要的服务 |
| LogManagementController | 负责实现日志管理界面所需要的服务 |
| UserManagement | 用户管理界面的领域模型对象，拥有一次用户管理所持有的用户列表、操作记录等信息，可以帮助完成用户管理所需要的服务 |
| LogManagement | 日志管理界面的领域模型对象，拥有一次账目初始化所持有的日志列表、操作记录等信息，可以帮助完成日志管理界面所需要的服务 |
| LoginController | 负责实现对应于登录界面所需要的服务 |

(3)模块内部类的接口规范

usersl模块的接口规范

UserManagementController的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| UserManagementController.login | 语法 | Public ResultMessage login(Srting id,String password) | |
| 前置条件 | password符合输入规则 | |
| 后置条件 | 调用UserManagement领域的login方法 | |
| LogManagementController.getLog | 语法 | public LogVO[] getLog(String office,String staff,String time); | |
| 前置条件 | 输入的营业厅、人员、时间符合输入规则; | |
| 后置条件 | 调用LogManagement领域的getLog方法 | |
| UserManagementController.select | 语法 | public UserVO select(String id); | |
| 前置条件 | 开启新的一轮用户管理 | |
| 后置条件 | 调用UserManagement领域的select方法 | |
| UserManagementController.delect | 语法 | public ResultMessage delect(String id); | |
| 前置条件 | 一开启新一轮的用户信息管理，已选择好需要删除的人员信息单，删除按钮被点击 | |
| 后置条件 | 调用UserManagement领域的delect方法 | |
| UserManagementController.revise | 语法 | public void revise(String id); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被选择，修改按钮被点击 | |
| 后置条件 | 调用UserManagement领域的revise方法 | |
| UserManagementControllersaveChange | 语法 | public ResultMessage saveChange(UserVO vo); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被修改，保存按钮被点击 | |
| 后置条件 | 调用UserManagement领域的saveChange方法 | |
| UserManagementController.save | 语法 | public ResultMessage save(UserVO vo); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被添加，保存按钮被点击 | |
| 后置条件 | 调用UserManagement领域的save方法 | |
| UserManagementController.add | 语法 | public UserVO add(String name,String accountnumber,String code,String privileges); | |
| 前置条件 | 在用户管理的过程中，添加按钮被点击 | |
| 后置条件 | 调用UserManagement领域的add方法 | |
| UserManagementController.getAllUser | 语法 | public ArrayList<UserVO> getAllUser(); | |
| 前置条件 | 开启新的一轮用户管理 | |
| 后置条件 | 调用UserManagement领域的getAllUser方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| UserManagement.login | 返回登录信息 |
| LogManagement.getLog | 获取对应条件日志信息 |
| UserManagement.select | 获取用户信息 |
| UserManagement.delect | 删除用户对象 |
| UserManagement.revise | 修改用户信息 |
| UserManagement.saveChange | 保存修改信息 |
| UserManagement.save | 结束用户信息修改 |
| UserManagement.add | 新增用户信息 |
| UserManagement.getAllUser | 获取用户列表 |

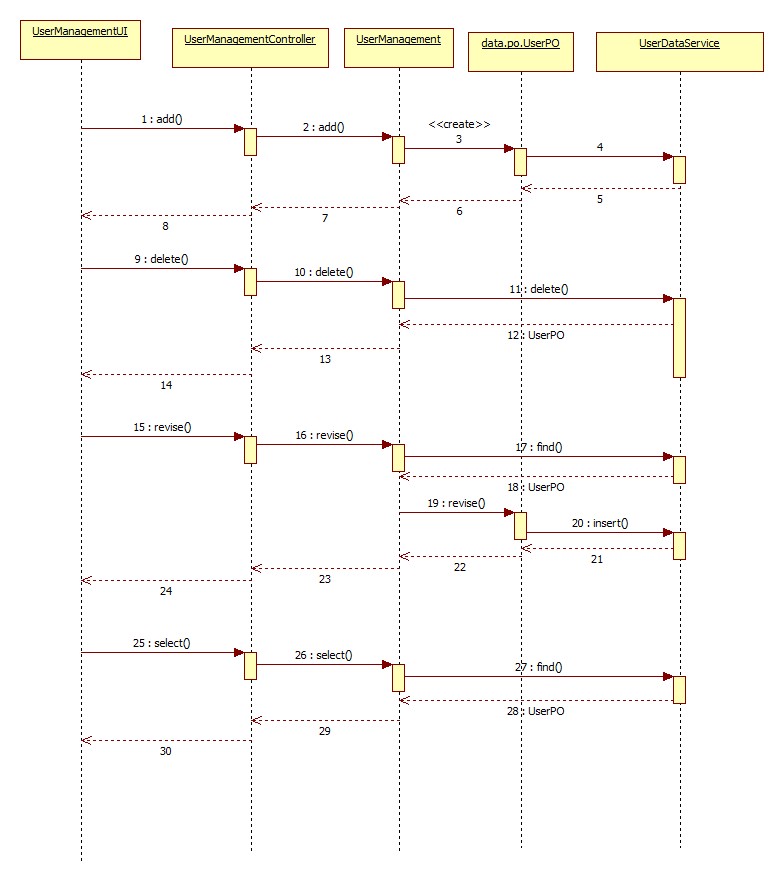
UserManagement的接口规范

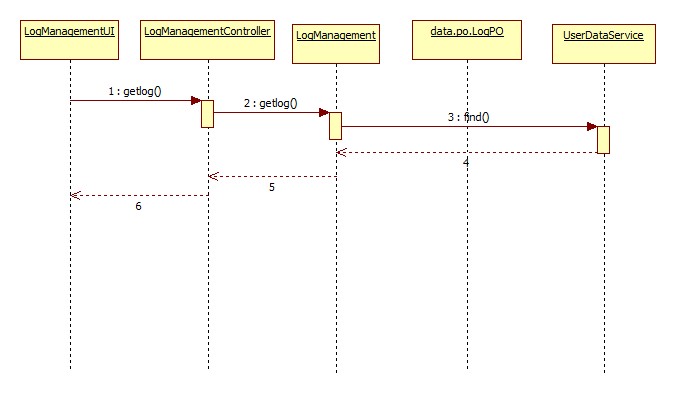
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务 | | | |
| UserManagement.login | 语法 | Public ResultMessage login(Srting id,String password) | |
| 前置条件 | password符合输入规则 | |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证结果 | |
| LogManagement.getLog | 语法 | public LogVO[] getLog(String office,String staff,String time); | |
| 前置条件 | 输入的营业厅、人员、时间符合输入规则; | |
| 后置条件 | 返回符合选择条件的log列表 | |
| UserManagement.select | 语法 | public UserVO select(String id); | |
| 前置条件 | 开启新的一轮用户管理 | |
| 后置条件 | 返回该人员的详细信息 | |
| UserManagement.delect | 语法 | public ResultMessage delect(String id); | |
| 前置条件 | 一开启新一轮的用户信息管理，已选择好需要删除的人员信息单，删除按钮被点击 | |
| 后置条件 | 在一个用户管理回合中，删除该用户信息单，持久化更新本次删除的信息单，结束用户信息删除 | |
| UserManagement.revise | 语法 | public void revise(String id); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被选择，修改按钮被点击 | |
| 后置条件 | 开启用户信息单对象的修改方法 | |
| UserManagement.saveChange | 语法 | public ResultMessage saveChange(UserVO vo); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被修改，保存按钮被点击 | |
| 后置条件 | 在一次用户管理的回合中，持久化更新用户信息单对象的数据，结束本次用户信息修改 | |
| UserManagement.save | 语法 | public ResultMessage save(UserVO vo); | |
| 前置条件 | 在用户管理的过程中，用户信息单已经被添加，保存按钮被点击 | |
| 后置条件 | 在一次用户管理的回合中，持久化更新用户信息单对象的数据，结束本次用户信息添加 | |
| UserManagement.add | 语法 | public UserVO add(String name,String accountnumber,String code,String privileges); | |
| 前置条件 | 在用户管理的过程中，添加按钮被点击 | |
| 后置条件 | 返回新的用户信息单 | |
| UserManagement.getAllUser | 语法 | public ArrayList<UserVO> getAllUser(); | |
| 前置条件 | 开启新的一轮用户管理 | |
| 后置条件 | 返回所有的用户信息单 |

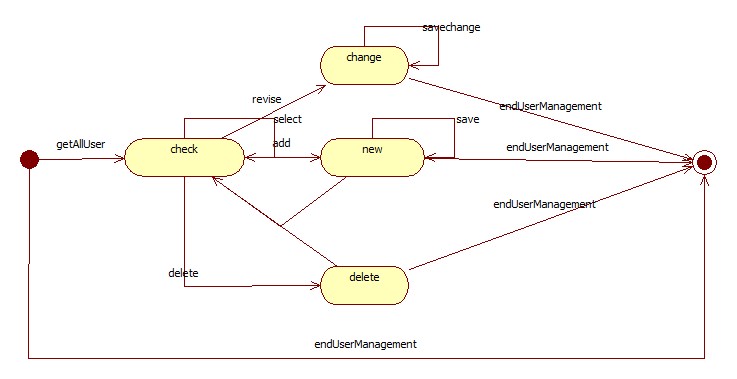
需要的服务（需接口）

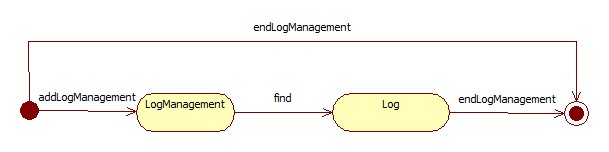
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getUserDatabase | 得到user数据库服务的引用 |
| UserDataService.findLog(String id) | 根据id进行查找单一持久化对象 |
| UserDataService.findsLog(String[]ids) | 根据ids查找多个持久化对象 |
| UserDataService.insertUserPO(UserPO PO) | 插入单一持久化对象 |
| UserDataService.delectUserPO(UserPO PO) | 删除单一持久化对象 |
| UserDataService.updateUserPO(UserPO PO) | 更新单一持久化对象 |
| UserDataService.findUser(String id) | 根据id进行查找单一持久化对象 |
| UserDataService.findsUser(String[]ids) | 根据ids查找多个持久化对象 |

(4)业务逻辑层的动态模型









(5)业务逻辑层的设计风格

利用委托式风格，每个节目需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

### 4.1.10stocksl模块

(1)模块描述

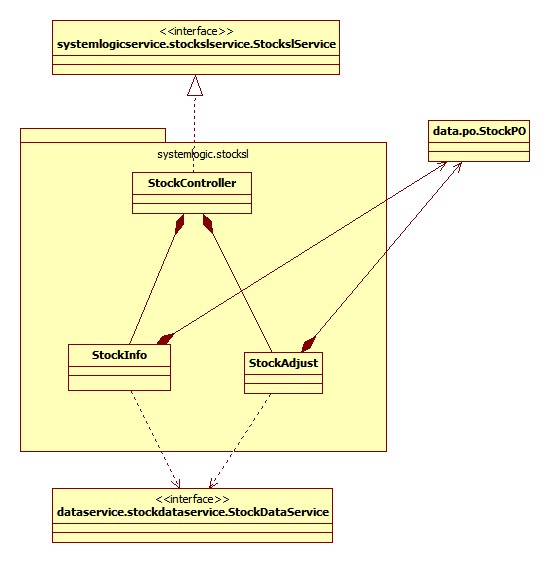
stocksl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

stocksl模块的职责及接口参见快递物流系统软件体系描述文档。

(2)整体结构

根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了stockslervice和stockdataservice两个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了StockController，这样它们会将对库存信息管理的系统逻辑处理委托给StockInfo、StockAdjust对象。StockPO是作为库存信息管理的持久化对象被添加到设计模型上的。

stocksl模块的图如下图所示。



stocksl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| StockController | 负责实现库存信息管理和库区调整界面所需要的服务 |
| StockInfo | 库存信息管理的领域模型对象，拥有一次库存信息管理所持有的库存信息列表、操作记录等信息，可以帮助完成库存信息管理界面所需要的服务 |
| StockAdjust | 库区调整的领域模型对象，拥有一次机构管理所持有的库区调整信息列表、操作记录等信息，可以帮助完成库区调整所需要的服务 |

(3)模块内部类的接口规范

StockController和StockInfo、StockAdjust的接口规范如下表

StockController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| StockController. show | 语法 | public StockInfoVO show (String[] time); |
| 前置条件 | 已创建一个StockInfo对象，并且输入符合输入规则 |
| 后置条件 | 调用StockInfo领域对象的show方法 |
| StockController.initialize | 语法 | public ResultMessage initialize(String initialinfo); |
| 前置条件 | 已创建一个StockInfo领域对象，并且符合输入规则。 |
| 后置条件 | 调用StockInfo领域对象的initialize方法 |
| StockController.check | 语法 | public ResultMessage check(); |
| 前置条件 | 已创建一个StockInfo领域对象 |
| 后置条件 | 调用StockInfo领域对象的check方法 |
| StockController.getArea | 语法 | public AreaVO[] getAreas(); |
| 前置条件 | 已创建一个StockAdjust领域对象 |
| 后置条件 | 调用StockAdjust领域对象的getAreas方法 |
| StockController.selectArea | 语法 | public AreaVO selectArea(String name); |
| 前置条件 | 已创建一个StockAdjust领域对象 |
| 后置条件 | 调用StockAdjust领域对象的selectArea方法 |
| StockController.range | 语法 | public void range(String adjustrange); |
| 前置条件 | 已创建一个StockAdjust领域对象，输入符合输入规则 |
| 后置条件 | 调用StockAdjust领域对象的range方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| StockInfo.show(String[] time) | 根据时间段返回库存信息 |
| StockInfo.initialize(String initialinfo) | 保存库存初始化信息 |
| StockInfo.check | 进行库存盘点 |
| StockAdjust.getArea | 返回库区信息 |
| StockAdjust.selectArea | 根据选择的库区返回库区范围 |
| StockAdjust.range | 记录并调整库区范围 |

StockInfo的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| StockInfo.show | 语法 | public StockInfoVO show (String[] time); |
| 前置条件 | 启动库存查看 |
| 后置条件 | 返回指定时间段内的库存信息 |
| StockInfo.initialize | 语法 | public ResultMessage initialize(String initialinfo); |
| 前置条件 | 启动库存信息初始化 |
| 后置条件 | 结束库存信息初始化，持久化更新涉及的领域对象的数据。 |
| StockInfo.check | 语法 | public ResultMessage check(); |
| 前置条件 | 启动库存盘点 |
| 后置条件 | 盘点库存，导出Excel文件 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getStockDatabase | 得到Stock数据库服务的引用 |
| StockDataService.findposition(int row,int shelf,int seat) | 根据位置进行查找单一持久化对象 |
| StockDataService.findpositions(ArrayList<int[][]> positions ) | 根据位置查找多个持久化对象 |
| StockDataService.insert(StockPO PO) | 插入单一持久化对象 |
| StockDataService.delete(StockPO PO) | 删除单一持久化对象 |
| StockDataService.update(StockPO PO) | 更新单一持久化对象 |

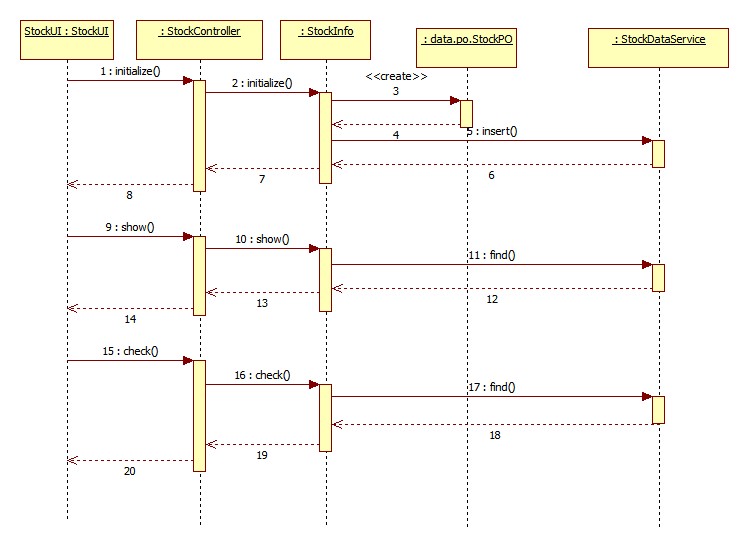
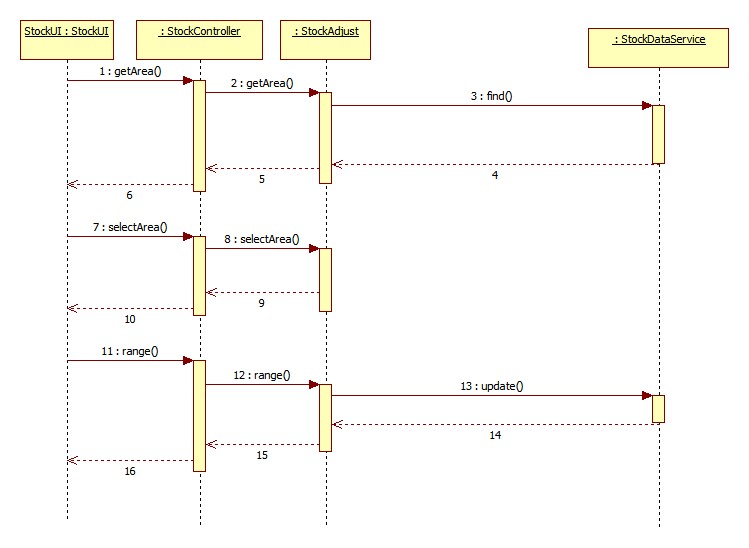
StockAdjust的接口规范

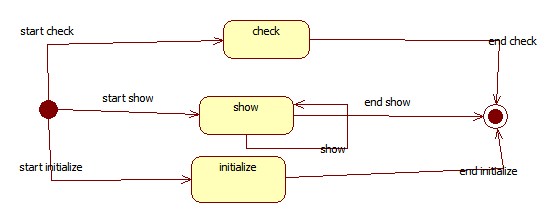
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| StockAdjust.getArea | 语法 | public AreaVO[] getAreas(); |
| 前置条件 | 启动库区调整 |
| 后置条件 | 返回库区信息 |
| StockAdjust.selectArea | 语法 | public AreaVO selectArea(String name); |
| 前置条件 | 启动库区调整 |
| 后置条件 | 根据库区信息返回库区范围 |
| StockAdjust.range | 语法 | public void range(String adjustrange); |
| 前置条件 | 启动库区调整 |
| 后置条件 | 记录并调整库区范围，结束库区调整，持久化更新涉及领域对象的数据 |

需要的服务（需接口）

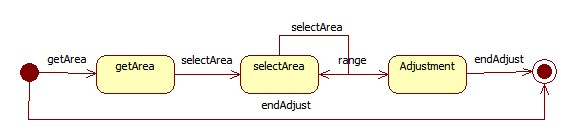
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getStockDatabase | 得到Stock数据库服务的引用 |
| StockDataService.findposition(int row,int shelf,int seat) | 根据位置进行查找单一持久化对象 |
| StockDataService.findpositions(ArrayList<int[][]> positions ) | 根据位置查找多个持久化对象 |
| StockDataService.insert(StockPO PO) | 插入单一持久化对象 |
| StockDataService.delete(StockPO PO) | 删除单一持久化对象 |
| StockDataService.update(StockPO PO) | 更新单一持久化对象 |

(4)业务逻辑层的动态模型





StockInfo对象状态图



StockAdjust 对象状态图

(5)业务逻辑层的设计风格

利用委托式风格，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领域的对象。

### 4.1.11ioputsl模块

(1)模块描述

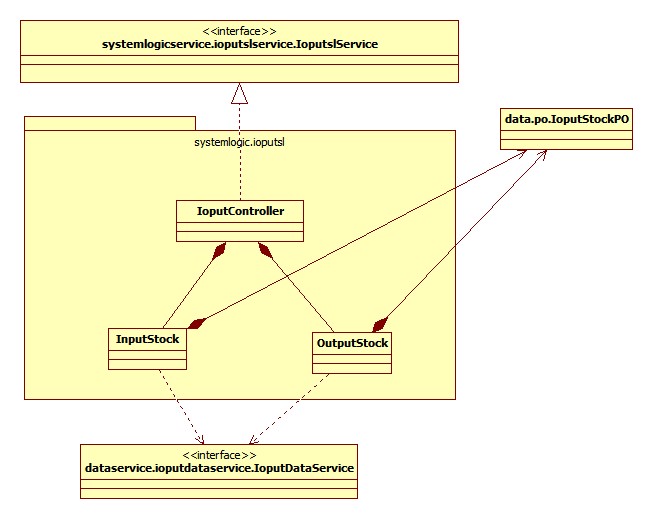
ioputsl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

ioputsl模块的职责及接口参见快递物流系统软件体系描述文档。

(2)整体结构

根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了ioputservice和ioputdataservice两个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了IoputStockController，这样它们会将对库存信息管理的系统逻辑处理委托给InputStock、OutputStock对象。IoputPO是作为库存信息管理的持久化对象被添加到设计模型上的。

ioputsl模块的图如下图所示。



ioputsl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| IoputController | 负责实现出入库管理界面所需要的服务 |
| InputStock | 入库管理的领域模型对象，拥有一次入库管理所持有的快递信息、操作记录等信息，可以帮助完成入库管理界面所需要的服务 |
| OutputStock | 出库管理的领域模型对象，拥有一次出库管理所持有的快递信息、操作记录等信息，可以帮助完成出库管理界面所需要的服务 |

IoputController和InputStock、OutputStock的接口规范如下表

IoputController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| IoputController.showInputInfo | 语法 | public InMessageVO showInputInfo (String id); |
| 前置条件 | 已创建一个InputStock领域对象，输入符合输入规则 |
| 后置条件 | 调用InputStock领域对象的showInfo方法 |
| IoputController.position | 语法 | public ResultMessage position (String position); |
| 前置条件 | 已创建一个InputStock领域对象，输入符合输入规则 |
| 后置条件 | 调用InputStock领域对象的position方法 |
| IoputController.showOutputInfo | 语法 | public OutMessageVO showOutputInfo(String id); |
| 前置条件 | 已创建一个OutputStock领域对象，输入符合输入规则 |
| 后置条件 | 调用OutputStock领域对象的showInfo方法 |
| IoputController.saveOutput | 语法 | public ResultMessage saveOutput() |
| 前置条件 | 已创建一个OutputStock领域对象 |
| 后置条件 | 调用OutputStock领域对象的saveOutput方法 |
| IoputController.report | 语法 | public ResultMessage report(String id); |
| 前置条件 | 已创建一个OutputStock领域对象，输入符合输入规则 |
| 后置条件 | 调用OutputStock领域对象的report方法 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 服务 |
| InputStock.showInfo | 返回快递信息 |
| InputStock.position | 记录快递位置信息，更新库存 |
| OutputStock.showInfo | 返回快递信息 |
| OutputStock.saveOutput | 记录快递出库信息 |
| OutputStock.report | 快递出库报错 |

InputStock的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InputStock.showInfo | 语法 | public InMessageVO showInfo (String id); |
| 前置条件 | 启动入库管理 |
| 后置条件 | 返回快递信息 |
| InputStock.position | 语法 | public ResultMessage position (String position); |
| 前置条件 | 已返回快递信息 |
| 后置条件 | 记录库存信息，结束入库管理，持久化更新涉及的领域对象的数据。 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getIoputDatabase | 得到Ioput数据库服务的引用 |
| IoputDataService.find(String id) | 根据id进行查找单一持久化对象 |
| IoputDataService.finds(String[]ids) | 根据ids查找多个持久化对象 |
| IoputDataService.findDate(String date) | 根据日期查找多个持久化对象 |
| IoputDataService.findDates(String[] date) | 根据日期段查找多个持久化对象 |
| IoputDataService.findTimes(String[] time) | 根据时间段寻找多个持久化对象 |
| IoputDataService.insert(IoputPO PO) | 插入单一持久化对象 |
| IoputDataService.delete(IoputPO PO) | 删除单一持久化对象 |
| IoputDataService.update(IoputPO PO) | 更新单一持久化对象 |

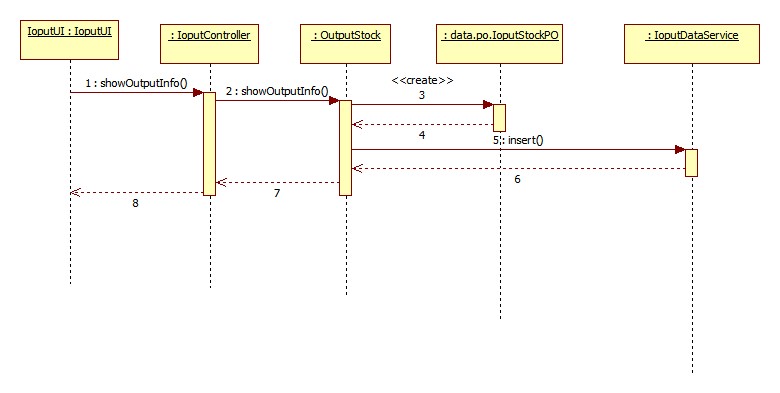
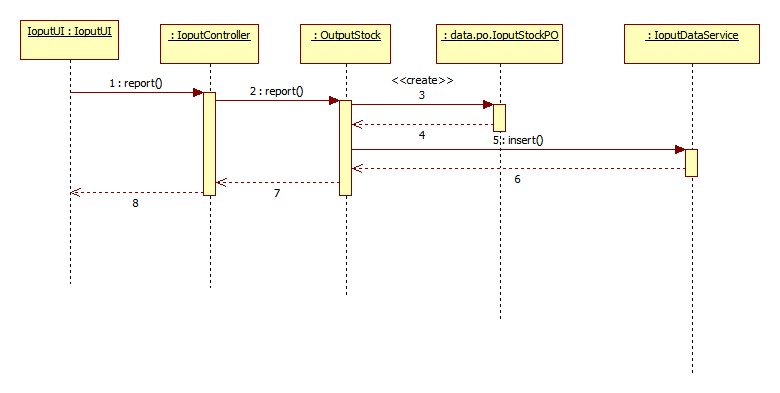
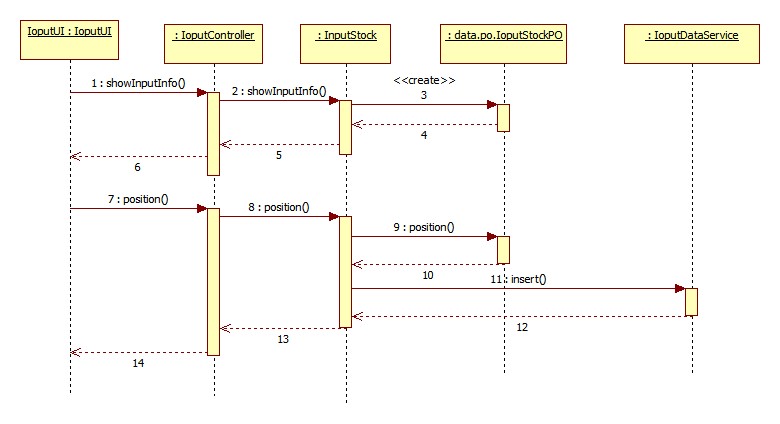
OuputStock的接口规范

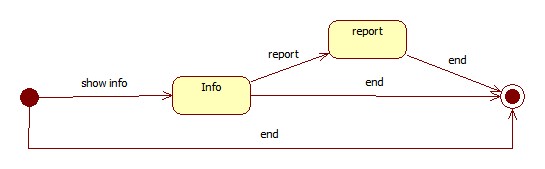
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OutputStock.showInfo | 语法 | public OutMessageVO showInfo(String id); |
| 前置条件 | 启动一轮出库管理 |
| 后置条件 | 根据快递条形码编号返回快递信息 |
| OutputStock.saveOutput | 语法 | public ResultMessage saveOutput() |
| 前置条件 | 启动一轮出库管理 |
| 后置条件 | 记录出库信息，结束出库管理，持久化更新涉及的领域对象的数据 |
| OutputStock.report | 语法 | public ResultMessage report(String id); |
| 前置条件 | 启动一轮出库管理 |
| 后置条件 | 记录报错，结束出库管理，持久化更新涉及的领域对象的数据 |

需要的服务（需接口）

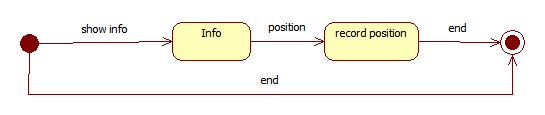
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getIoputDatabase | 得到Ioput数据库服务的引用 |
| IoputDataService.find(String id) | 根据id进行查找单一持久化对象 |
| IoputDataService.finds(String[]ids) | 根据ids查找多个持久化对象 |
| IoputDataService.findDate(String date) | 根据日期查找多个持久化对象 |
| IoputDataService.findDates(String[] date) | 根据日期段查找多个持久化对象 |
| IoputDataService.findTimes(String[] time) | 根据时间段寻找多个持久化对象 |
| IoputDataService.insert(IoputPO PO) | 插入单一持久化对象 |
| IoputDataService.delete(IoputPO PO) | 删除单一持久化对象 |
| IoputDataService.update(IoputPO PO) | 更新单一持久化对象 |

(4)业务逻辑层的动态模型





OutputStock对象状态图



InputStock对象状态图

(5)业务逻辑层的设计风格

利用委托式风格，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领域的对象

### 4.1.12sendsl模块

(1)模块描述

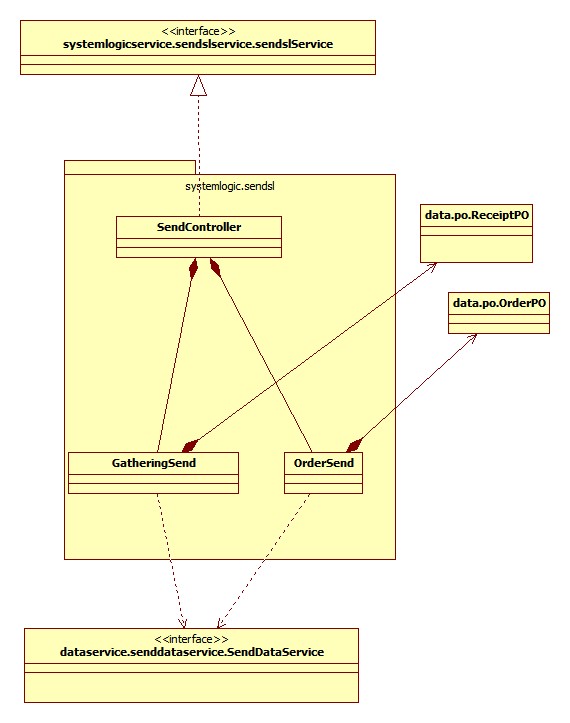
sendsl模块承担的需求参见快递物流系统需求规格说明文档功能需求及相关非功能需求。

sendsl模块的职责及接口参见快递物流系统软件体系描述文档。

(2)整体结构

根据体系结构的设计，采用分层风格，将系统分为展示层、系统逻辑层、数据层。每一层之间为了灵活性，添加了接口，以实现针对接口编程，隔离数据传输的职责，降低层与层之间耦合，添加了sendslervice和senddataservice两个接口。为了隔离系统逻辑职责和逻辑控制职责，我们添加了OrderSendController、GatheringSendController，这样它们会将对库存信息管理的系统逻辑处理委托给OrderSend、GatheringSend对象。OrderPO、ReceiptPO是作为库存信息管理的持久化对象被添加到设计模型上的。

sendsl模块的图如下图所示。



sendsl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| SendController | 负责实现订单输入和收款管理界面所需要的服务 |
| OrderSend | 订单输入的领域模型对象，拥有一次订单输入所持有的快递信息、操作记录等信息，可以帮助完成订单输入界面所需要的服务 |
| GatheringSend | 收款管理的领域模型对象，拥有一次收款管理所持有的收款单信息、操作记录等信息，可以帮助完成收款管理界面所需要的服务 |

(3)模块内部类的接口规范

SendController、OrderSend和GatheringSend的接口规范如下表

SendController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| SendController.calculate | 语法 | public OrderVO calculate(Sting[] info); |
| 前置条件 | 已创建一个OrderSend领域对象,并且输入符合输入规则 |
| 后置条件 | 调用OrderSend领域对象的calculate方法 |
| SendController.getChange | 语法 | public BillVO getChange(double cash); |
| 前置条件 | 已创建一个OrderSend领域对象 |
| 后置条件 | 调用OrderSend领域对象的getChange方法 |
| SendController.end | 语法 | public void orderend(BillVO bill,OrderVO order); |
| 前置条件 | 已创建一个OrderSend领域对象 |
| 后置条件 | 调用OrderSend领域对象的end方法 |
| SendController. receiptsNew | 语法 | public receiptsVO[] receiptsNew (String office); |
| 前置条件 | 已创建一个GatheringSend领域对象 |
| 后置条件 | 调用GatheringSend领域对象的receiptsNew方法 |
| SendController. receiptsCheck | 语法 | public receiptsVO[] receiptsCheck(String office,String date); |
| 前置条件 | 已创建一个GatheringSend领域对象 |
| 后置条件 | 调用GatheringSend领域对象的receiptsCheck方法 |
| SendController.computeOrdernumber | 语法 | public String orderend(); |
| 前置条件 | 已创建一个OrderSend领域对象 |
| 后置条件 | 调用OrderSend领域对象的computeOrdernumber方法 |
| SendController.computedue | 语法 | public String orderend(OrderVO order); |
| 前置条件 | 已创建一个OrderSend领域对象 |
| 后置条件 | 调用OrderSend领域对象的computedue方法 |

|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| OrderSend.calculate | 返回快递信息，计算的价格和预期到达的日期 |
| OrderSend.getChange | 根据支付的金额，计算找零的金额 |
| OrderSend.end | 结束订单输入，持久化更新订单的数据 |
| GatheringSend. receiptsNew | 返回生成的收款单信息，记录收款单数据 |
| GatheringSend. receiptsCheck | 返回收款单信息 |
| OrderSend.computeOrdernumber | 返回订单号 |
| OrderSend.computedue | 结束订单输入，更新订单的数据 |

OrderSend的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderSend.calculate | 语法 | public OrderVO calculate(Sting[] info); |
| 前置条件 | 启动订单输入回合 |
| 后置条件 | 返回快递信息，计算的价格和预期到达的日期 |
| OrderSend.getChange | 语法 | public BillVO getChange(double cash); |
| 前置条件 | 已计算价格 |
| 后置条件 | 根据支付的金额，返回找零的金额 |
| OrderSend.end | 语法 | public void orderend(BillVO bill,OrderVO order); |
| 前置条件 | 已完成支付 |
| 后置条件 | 结束订单输入，持久化更新订单的数据 |
| OrderSend.computeOrdernumber | 语法 | public String orderend(); |
| 前置条件 | 已完成支付 |
| 后置条件 | 返回10位订单号 |
| OrderSend.computedue | 语法 | public String orderend(OrderVO order); |
| 前置条件 | 已完成支付 |
| 后置条件 | 结束订单输入，持久化更新订单的数据 |

需要的服务（需接口）

|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getSendDatabase | 得到Send数据库服务的引用 |
| SendDataService.insert(OrderPO PO) | 插入单一持久化对象 |

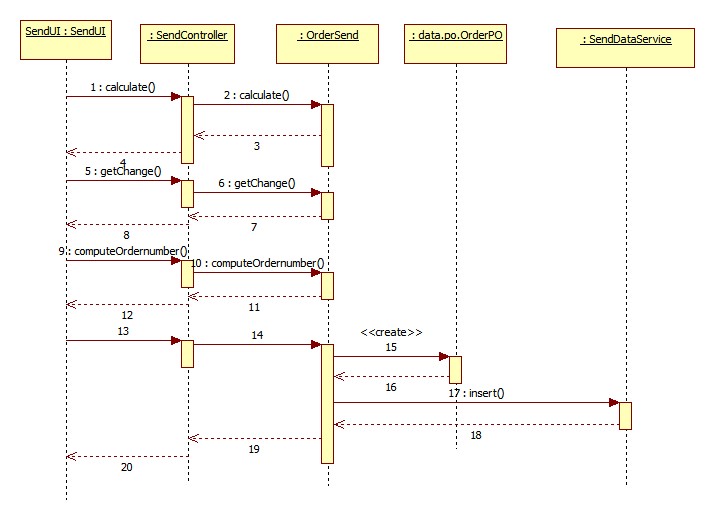
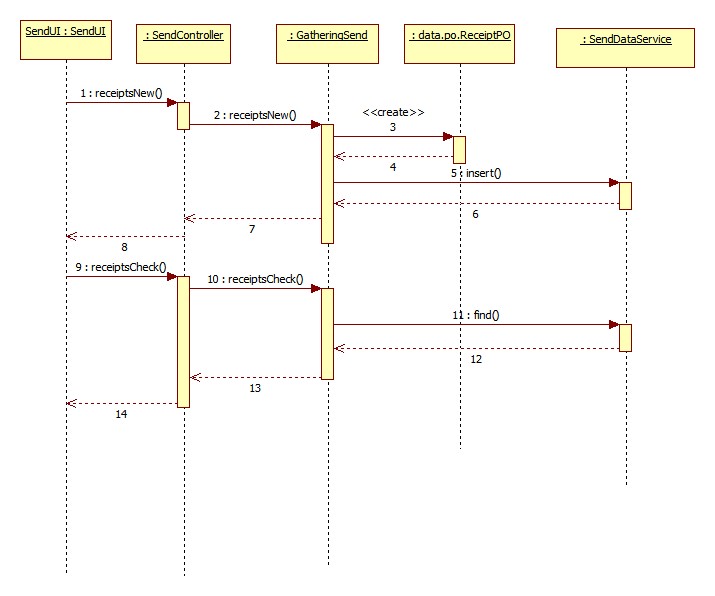
GatheringSend的接口规范

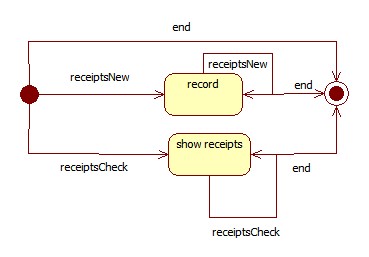
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| GatheringSend. receiptsNew | 语法 | public receiptsVO[] receiptsNew (String office); |
| 前置条件 | 启动收款管理 |
| 后置条件 | 系统计算并显示收款单信息，记录收款单数据 |
| GatheringSend. receiptsCheck | 语法 | public receiptsVO[] receiptsCheck(String office,String date); |
| 前置条件 | 启动收款管理 |
| 后置条件 | 返回收款单信息 |

需要的服务（需接口）

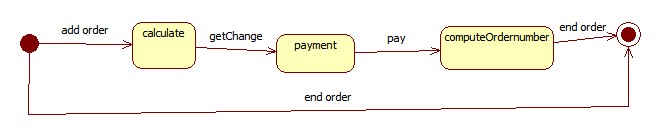
|  |  |
| --- | --- |
| 服务名 | 需要的服务 |
| DatabaseFactory.getSendDatabase | 得到Send数据库服务的引用 |
| SendDataService.insert(ReceiptPO PO) | 插入单一持久化对象 |

(4)业务逻辑层的动态模型





GatheringSend 对象状态图



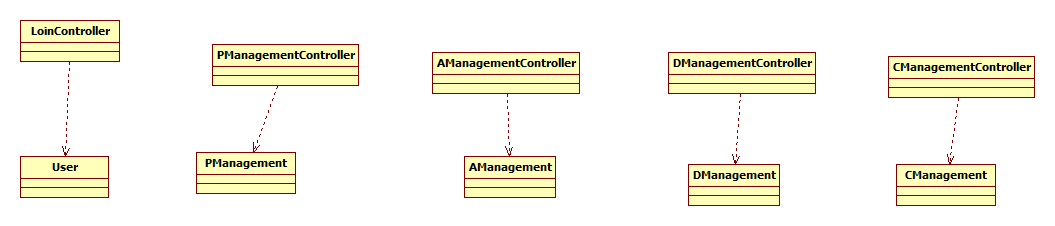
OrderSend 对象状态图

(5)业务逻辑层的设计风格

利用委托式风格，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领域的对象

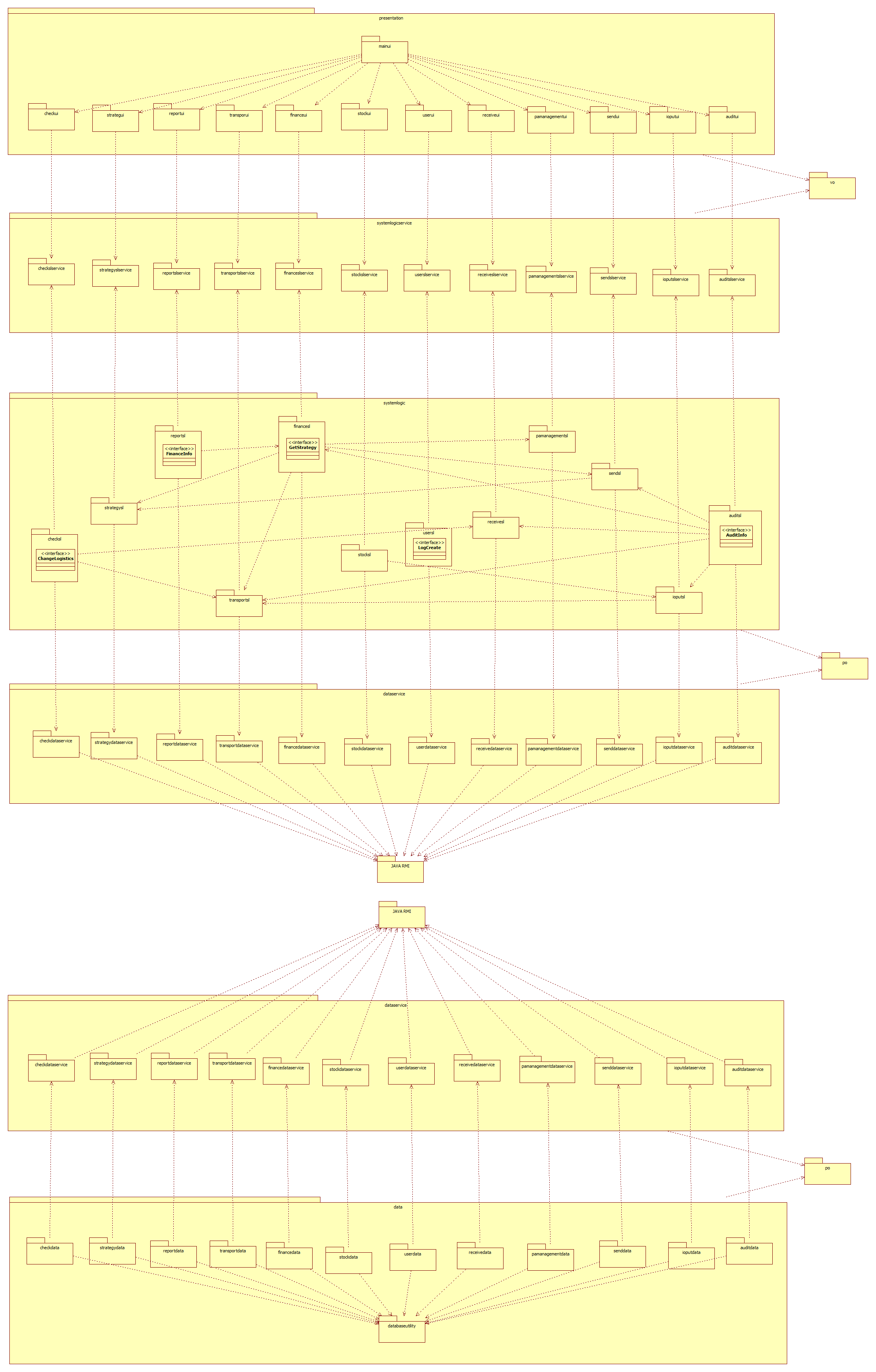
## 4.2用户界面层的行为

在界面层中界面的跳转的实现，我们决定采取委托式的控制风格，在快递物流洗头膏中业务逻辑层的设计可以分为两个子层：一个是Controller；一个是domain Object层。Controller层主要负责向界面层(presentation)提供服务，并且利用具体的damian Object来实现各个服务。Domain Object是业务逻辑层中各个领域模型。下图为pamanagementbl逻辑层基于委托式控制风格的业务逻辑设计

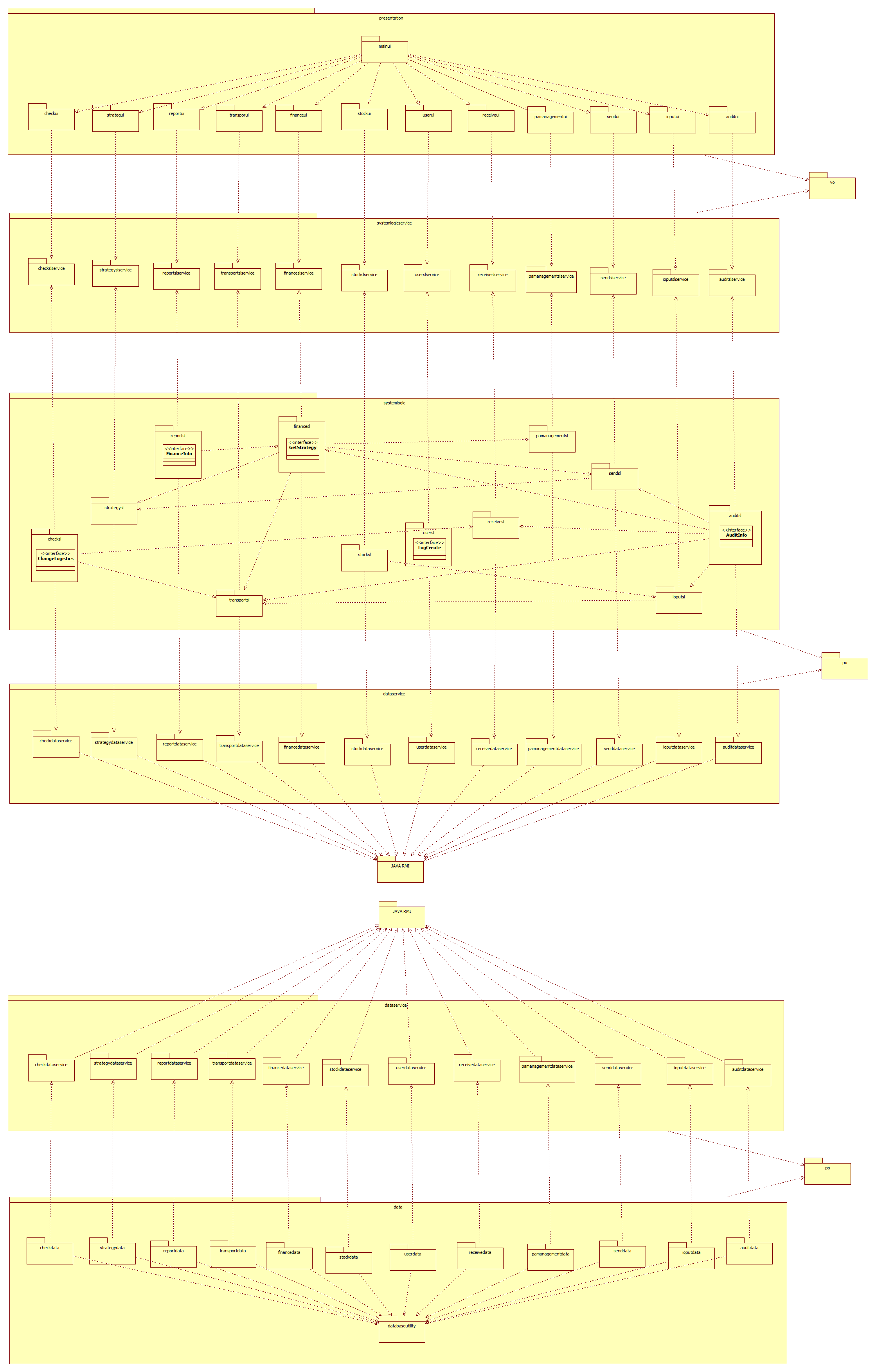


# 5.依赖视角

下面是客户端和服务端各自包之间的依赖关系



客户端包图



服务端包图