

灯具进销存系统（JXCS）

灯具进销存系统（JXCS）软件详细设计描述文档



2014-11-3

南京大学仙林校区软件学院ROOM404

小队成员：金鑫

李斐

梁思宇

周宇宸

目录

[0.更新历史 2](#_Toc402825446)

[1.引言 2](#_Toc402825447)

[2.产品概述 2](#_Toc402825448)

[3.体系结构设计概述 2](#_Toc402825449)

[4.结构视角 2](#_Toc402825450)

[4.1业务逻辑层的分解 3](#_Toc402825451)

[4.1.1 commoditybl模块 3](#_Toc402825452)

[4.1.2 accountbl模块 9](#_Toc402825453)

[4.1.3 financialbl模块 14](#_Toc402825454)

[4.1.4 initializationbl 模块 18](#_Toc402825455)

[4.1.5 invoice 模块 22](#_Toc402825456)

[4.1.6 promotion 模块 26](#_Toc402825457)

[4.1.7 salesbl模块 31](#_Toc402825458)

[4.1.8 systemlogbl模块 37](#_Toc402825459)

[4.1.9 userbl模块 41](#_Toc402825460)

[5.依赖视角 44](#_Toc402825461)

# 0.更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| **更新日期** | **更新说明** | **更新人员** | **版本号** |
| 2014-11-03 | 编写第一版软件详细设计描述文档 | 金鑫 | V1.0 |
| 2014-11-15 | 更新模块借口信息 | 周宇宸 | V2.0 |

# 1.引言

1.1编制目的

本报告详细完成对灯具公司进销存系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
| JXCS | 进销存系统 |  |

1.3参考资料

# 2.产品概述

参考灯具公司进销存系统用例文档和灯具公司进销存系统软件需求规格说明文档中对产品的概括描述。

# 3.体系结构设计概述

参考灯具公司进销存系统概要设计文档中对体系结构设计的概述。

# 4.结构视角

## 4.1业务逻辑层的分解

业务逻辑层的开发包图参见软件体系结构文档图3。

### 4.1.1 commoditybl模块

(1)模块概述

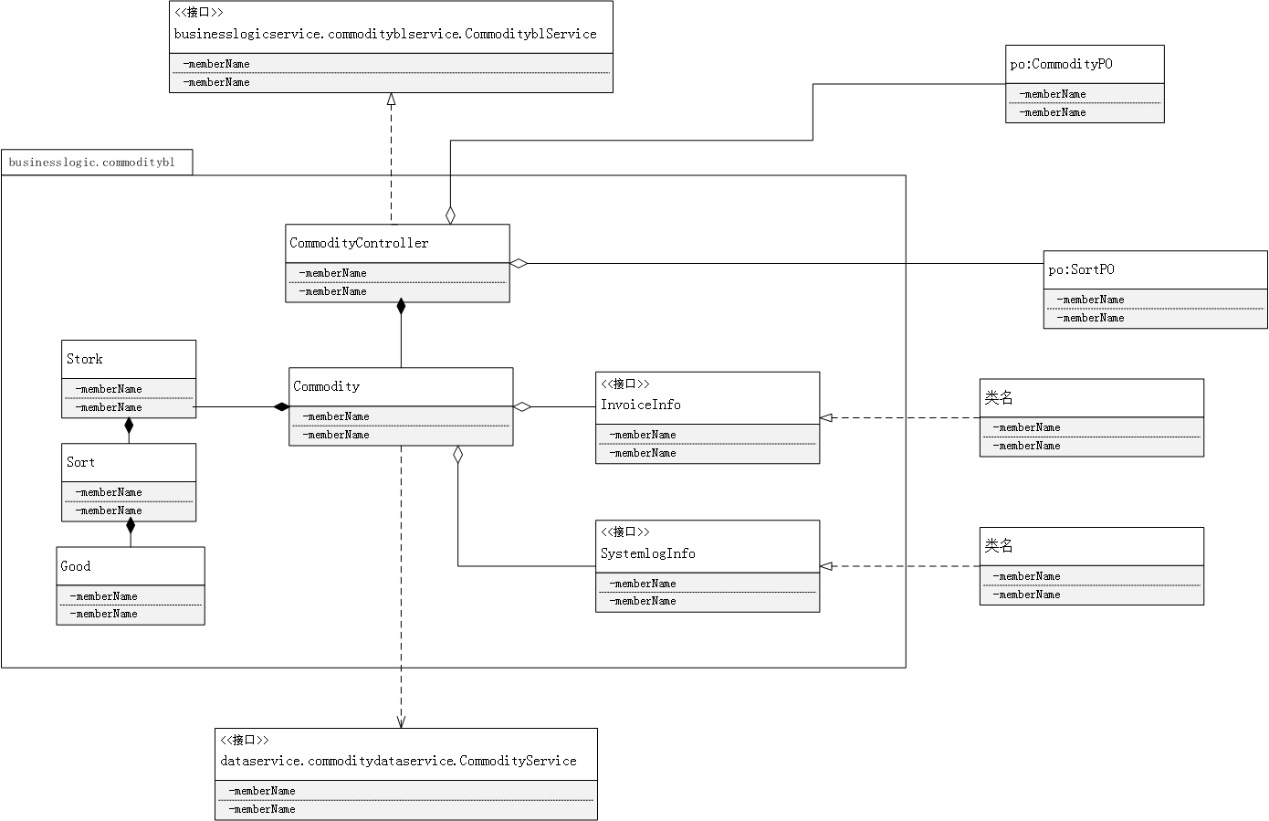
Commoditybl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Commoditybl模块的职责及借款参见软件体系结构描述文档。

(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。Commodity模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.commodityblservice.CommodityblService接口与业务逻辑层与数据层之间的dataservice.commoditydataservice.CommodityblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommodityController，这样CommodityController会将对商品与库存的业务逻辑处理委托给Commodity对象。CommodityPO是作为商品的持久化对象被添加到设计模型中去的。SortPO是作为商品种类的持久化对象被添加到设计模型中去的。而Stork保存有简单的库存信息，完成搜索与判断重复的逻辑操作，管理Sort对象。Sort保存有简单的商品信息，负责商品的查找。Good是简单的商品对象，拥有基本的商品信息。SystemlogInfo、InvoiceInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Commoditybl模块的设计如图



Commoditybl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| CommodityController | 负责实现商品界面所需要的服务 |
| Commodity | 库存领域模型对象，拥有库存与商品的信息，可以完成库存管理与商品管理操作 |
| Stork | 临时管理库存中商品与分类，保存最基本的商品与分类信息 |
| Sort | 商品分类，临时保存最基本的商品信息 |
| Good | 商品，保存有商品的名称与型号 |

(3)模块内部类的接口规范

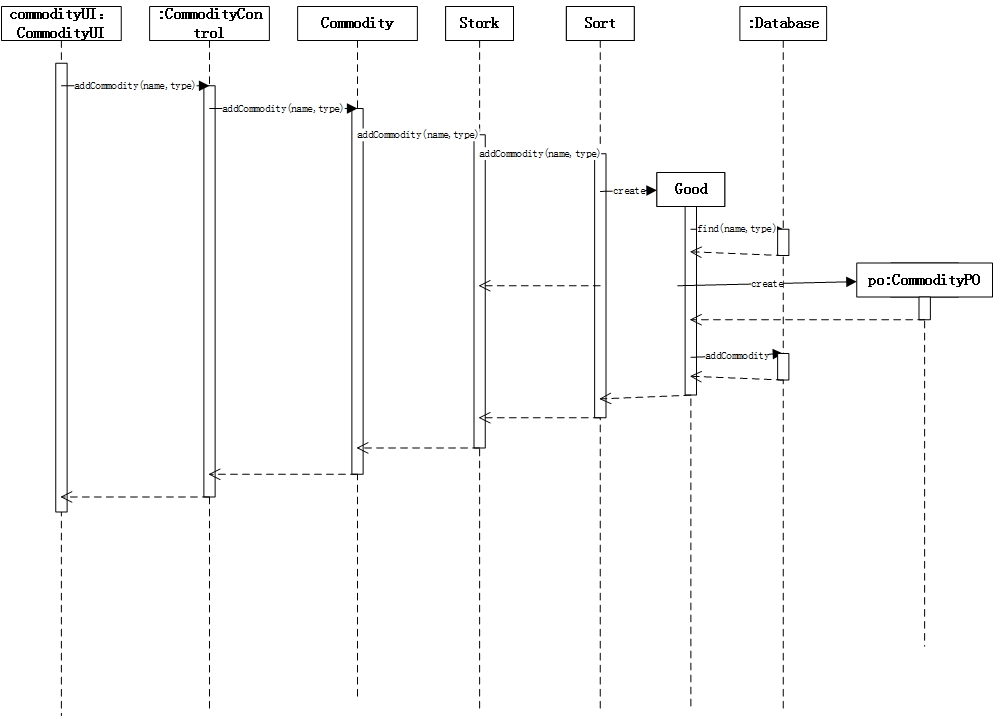
CommodityController接口规范:

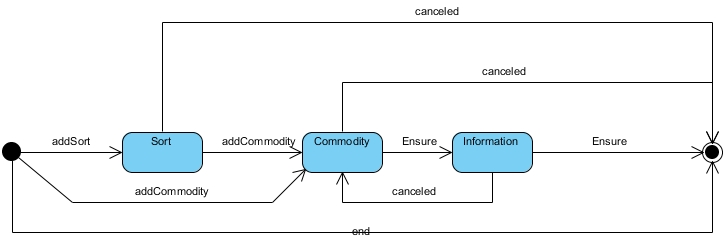
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| CommodityController.addCommodity | 语法 | Public ResultMessage addCommodity(String name, String type)  Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的addCommodity方法 |
| CommodityController.delCommodity | 语法 | Public ResultMessage delCommodity(String name, String type) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的delCommodity方法 |
| CommodityController.updateCommodity | 语法 | Public ResultMessage updateCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的updateCommodity方法 |
| CommodityController.searchCommodity | 语法 | Public ResultMessage searchCommodity(String word) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的searchCommodity方法 |
| CommodityController.addSort | 语法 | Public ResultMessage addSort(String name) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的addSort方法 |
| CommodityController.delSort | 语法 | Public ResultMessage delSort(String name) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的delSort方法 |
| CommodityController.updateSort | 语法 | Public ResultMessage updateSort(String name1, String name2) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的updateSort方法 |
| CommodityController.Examine | 语法 | Public ResultMessage examine(String time1, String time2) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的Examine方法 |
| CommodityController.Inventory | 语法 | Public ResultMessage inventory() |
| 前置条件 | 已创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity领域对象的Inventory方法 |
| CommodityController.addGift | 语法 | Public ResultMessage addGift(String name, String type) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的addGift方法 |
| CommodityController.delGift | 语法 | Public ResultMessage delGift(String name, String type) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的delGift方法 |
| CommodityController.patch | 语法 | Public ResultMessage patch(String name, String type, int number) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的patch方法 |
| CommodityController.warn | 语法 | Public ResultMessage warn(String name, String type, int number) |
| 前置条件 | 已创建一个Commodity领域对象，输入符合规则 |
| 后置条件 | 调用Commodity领域对象的warn方法 |
| 需要的服务 | | |
| 服务名 | 服务 | |
| Commodity.addCommodity(String name, String type, int in\_price, int out\_price) | 在分类中添加商品 | |
| Commodity.delCommodity(String name, String type) | 删除商品 | |
| Commodity.updateCommodity(String name, String type, int in\_price, int out\_price) | 更新商品信息 | |
| Commodity.searchCommodity(String word) | 根据关键词搜索商品 | |
| Commodity.addSort(String name) | 添加分类 | |
| Commodity.delSort(String name) | 删除分类 | |
| Commodity.updateSort(String name1, String name2) | 更新分类 | |
| Commodity.examine(String time1, String time2) | 查看库存 | |
| Commodity. inventory() | 盘点库存 | |
| Commodity.addGift(String name, String type) | 添加赠品对象 | |
| Commodity.delGift(String name, String type) | 删除赠品对象 | |
| Commodity.patch(String name, String type, int number) | 库存中商品报溢或报损 | |
| Commodity.warn(String name, String type, int number) | 设置商品的警戒线 | |

Commodity接口规范:

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Commodity.addCommodity | 语法 | Public ResultMessage addCommodity(String name, String type)  Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 输入的in\_price和out\_price为正整数 |
| 后置条件 | 添加商品，返回是否成功 |
| Commodity.delCommodity | 语法 | Public ResultMessage delCommodity(String name, String type) |
| 前置条件 | 无 |
| 后置条件 | 删除商品，返回是否成功 |
| Commodity.updateCommodity | 语法 | Public ResultMessage updateCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 输入的in\_price和out\_price为正整数 |
| 后置条件 | 更新商品，返回是否更新成功 |
| Commodity.searchCommodity | 语法 | Public ResultMessage searchCommodity(String word) |
| 前置条件 | 无 |
| 后置条件 | 返回搜索结果 |
| Commodity.addSort | 语法 | Public ResultMessage addSort(String name) |
| 前置条件 | 无 |
| 后置条件 | 添加分类，返回是否添加成功 |
| Commodity.delSort | 语法 | Public ResultMessage delSort(String name) |
| 前置条件 | 无 |
| 后置条件 | 删除分类，返回是否删除成功 |
| Commodity.updateSort | 语法 | Public ResultMessage updateSort(String name1, String name2) |
| 前置条件 | 无 |
| 后置条件 | 修改分类，返回是否修改成功 |
| Commodity.Examine | 语法 | Public ResultMessage examine(String time1, String time2) |
| 前置条件 | 输入的时间格式正确 |
| 后置条件 | 返回查看结果 |
| Commodity.Inventory | 语法 | Public ResultMessage inventory() |
| 前置条件 | 无 |
| 后置条件 | 返回盘点结果 |
| Commodity.addGift | 语法 | Public ResultMessage addGift(String name, String type) |
| 前置条件 | 无 |
| 后置条件 | 增加赠品，返回是否成功 |
| Commodity.delGift | 语法 | Public ResultMessage delGift(String name, String type) |
| 前置条件 | 无 |
| 后置条件 | 删除赠品，返回是否成功 |
| Commodity.patch | 语法 | Public ResultMessage patch(String name, String type, int number) |
| 前置条件 | 输入的number为数字 |
| 后置条件 | 生成报溢报损单，返回是否成功 |
| Commodity.warn | 语法 | Public ResultMessage warn(String name, String type, int number) |
| 前置条件 | 输入的number为数字 |
| 后置条件 | 返回是否设置成功 |
| 需要的服务 | | |
| 服务名 | 服务 | |
| CommodityDataService.findGood(String name, String type) | 根据name和type查找商品对象 | |
| CommodityDataService.findName(String name) | 根据name查找商品对象 | |
| CommodityDataService.findType(String type) | 根据type查找商品对象 | |
| CommodityDataService.addGood(CommodityPO po1, SortPO po2) | 在分类中添加商品对象 | |
| CommodityDataService.delGood(CommodityPO po) | 删除商品对象 | |
| CommodityDataService.updateGood(CommodityPO po1, CommodityPO po2) | 更新商品对象信息 | |
| CommodityDataService.addSort(SortPO po) | 添加分类对象 | |
| CommodityDataService.delSort(SortPO po) | 删除分类对象 | |
| CommodityDataService.updateSort(SortPO po1, SortPO po2) | 更新分类对象 | |
| CommodityDataService.addGift(CommodityPO po) | 添加赠品对象 | |
| CommodityDataService.findGift(String name, String type) | 查找赠品对象 | |
| CommodityDataService.delGift(CommodityPO po) | 删除赠品对象 | |
| CommodityDataService.getAll() | 获得所有商品对象 | |
| CommodityDataService.clear() | 清空所有商品与分类对象 | |

(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由各自的控制器委托给不同的领域对象。

### 4.1.2 accountbl模块

(1)模块概述

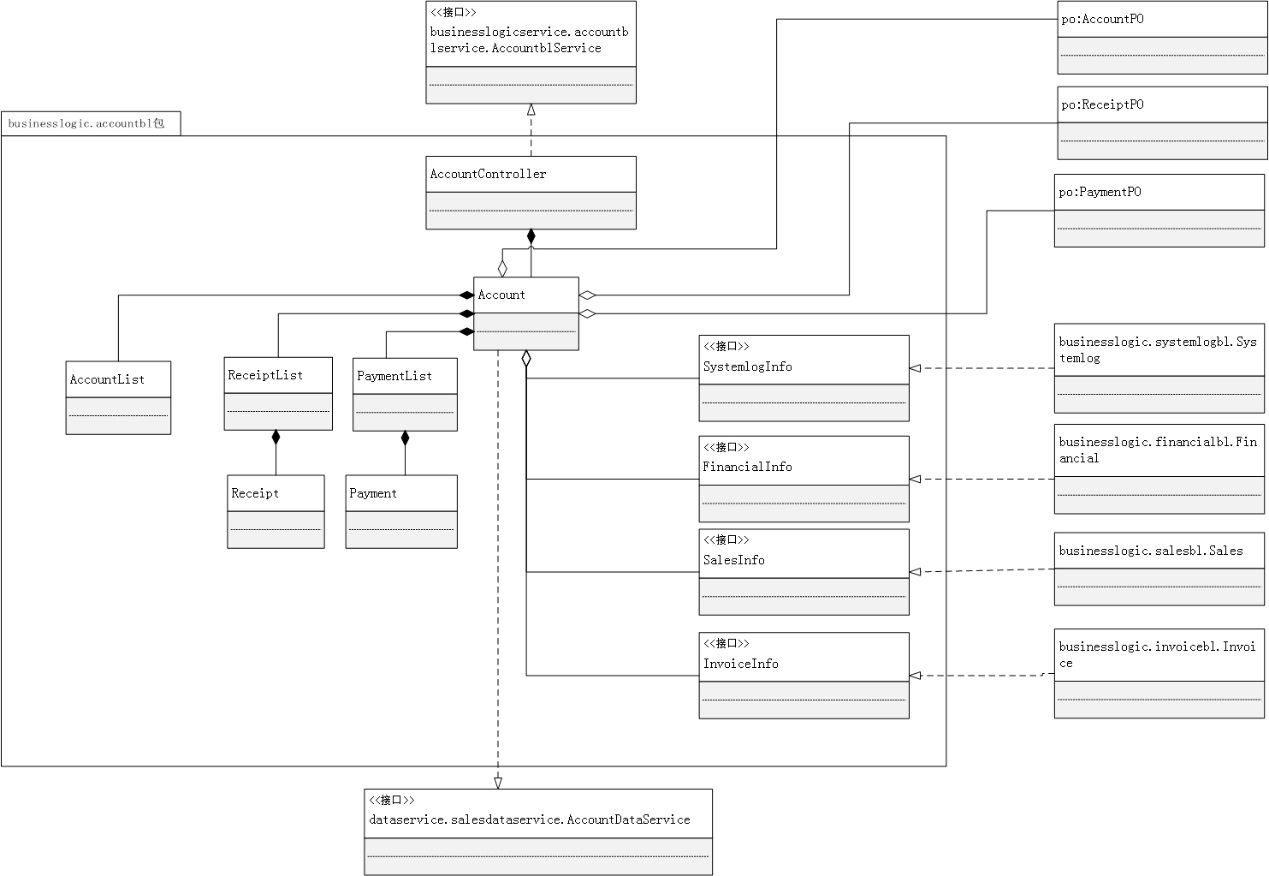
Accountbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Accountbl模块的职责及借款参见软件体系结构描述文档。

(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。Account模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.accountblservice.AccountblService接口与业务逻辑层与数据层之间的dataservice.accountdataservice.AccountblService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会将对账户的业务逻辑处理委托给Account对象。AccountPO是作为银行账户的持久化对象被添加到设计模型中去的。ReceiptPO是作为收款单的持久化对象被添加到设计模型中去的。PaymentPO是作为付款单的持久化对象被添加到设计模型中去的。而Account保存简单的账户信息，Receipt为收款的转账列表中的一项，ReceiptList负责管理Receipt，并计算收款总额。Payment为付款的条目清单中的一项。PaymentList负责管理Payment，并计算付款总额。SystemlogInfo、FinancialInfo、SalesInfo、InvoiceInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Accountbl模块的设计如图



Accountbl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountController | 负责实现账户界面所需要的服务 |
| Account | 收付款以账户管理领域模型对象，拥有一次收款、付款以及账户等信息，可以帮助完成账户界面所需要的服务 |
| ReceiptList | 管理此次操作中的收款的转账列表，完成计算收款总额操作 |
| Receipt | 收款操作中的转账列表的一项 |
| PaymentList | 管理此次操作中的付款的条目清单，完成计算付款总额的操作 |
| Payment | 付款操作中的条目清单的一项 |

(3)模块内部类的接口规范

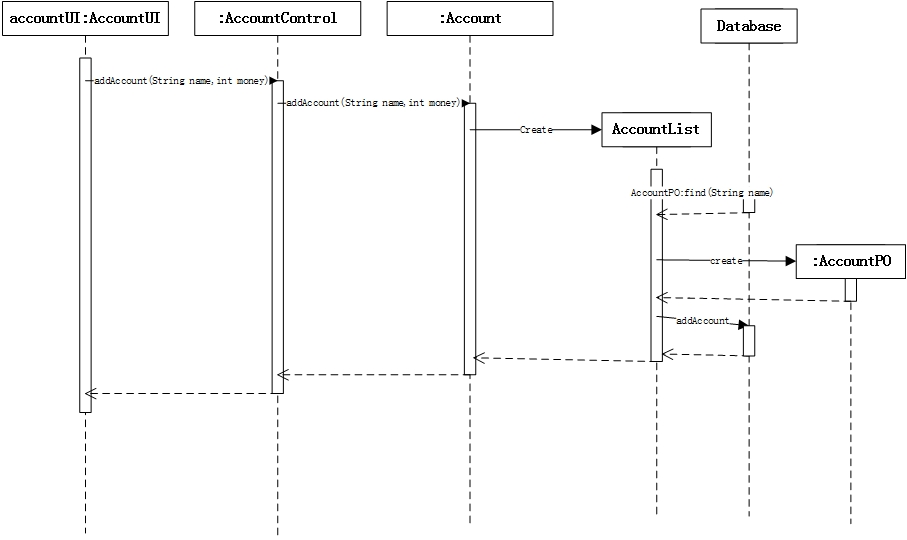
AccountController接口规范：

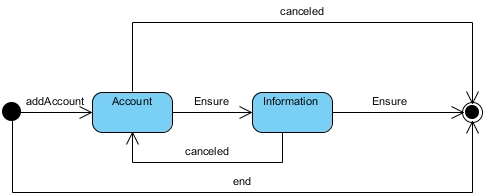
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| AccountController.checkAccount() | 语法 | Public ResultMessage checkAccount() |
| 前置条件 | 已创建一个Account领域对象 |
| 后置条件 | 调用Account领域对象的checkAccount方法 |
| AccountController.addAccount | 语法 | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | 已创建一个Account领域对象，输入符合规则 |
| 后置条件 | 调用Account领域对象addAccount的方法 |
| AccountController.delAccount | 语法 | Public ResultMessage delAccount(String name) |
| 前置条件 | 已创建一个Account领域对象，输入符合规则 |
| 后置条件 | 调用Account领域对象delAccount的方法 |
| AccountController.updateAccount | 语法 | Public ResultMessage updateAccount(String name1, String name2) |
| 前置条件 | 已创建一个Account领域对象，输入符合规则 |
| 后置条件 | 调用Account领域对象updateAccount的方法 |
| AccountController.searchAccount | 语法 | Public ResultMessage searchAccount(String word) |
| 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searchAccount的方法 |
| AccountController.addReceipt | 语法 | Public ResultMessage addReceipt(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) |
| 前置条件 | 已创建一个Account领域对象，输入符合规则 |
| 后置条件 | 调用Account领域对象addReceipt的方法 |
| AccountController.addPayment | 语法 | Public ResultMessage addPayment(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) |
| 前置条件 | 已创建一个Account领域对象，输入符合规则 |
| 后置条件 | 调用Account领域对象addPayment的方法 |
| 需要的服务 | | |
| Account.checkAccount() | | 获得当前账户信息 |
| Account.addAccount (String name, int money) | | 添加账户 |
| Account.delAccount(String name) | | 删除账户 |
| Account.updateAccount(String name1, String name2) | | 更新账户 |
| Account.searchAccount(String word) | | 根据word查找账户 |
| Account.addReceipt(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) | | 添加收款单 |
| Account.addPayment(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) | | 添加付款单 |

Account接口规范：

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Account.checkAccount() | 语法 | Public ResultMessage checkAccount() |
| 前置条件 | 管理人员为总经理或财务经理 |
| 后置条件 | 返回账户信息 |
| Account.addAccount | 语法 | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | 需要添加账户时，管理人员为总经理或财务经理 |
| 后置条件 | 返回是否添加成功 |
| Account.delAccount | 语法 | Public ResultMessage delAccount(String name) |
| 前置条件 | 需要删除账户时，管理人员为总经理或财务经理 |
| 后置条件 | 返回是否删除账户成功 |
| Account.updateAccount | 语法 | Public ResultMessage updateAccount(String name1, String name2) |
| 前置条件 | 需要更新账户信息时，管理人员为总经理或财务经理 |
| 后置条件 | 完成账户信息修改，返回是否成功修改 |
| Account.searchAccount | 语法 | Public ResultMessage searchAccount(String word) |
| 前置条件 | 操作员是总经理或财务经理 |
| 后置条件 | 返回查找结果 |
| Account.addReceipt | 语法 | Public ResultMessage addReceipt(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) |
| 前置条件 | 收到货款时，输入的money合法 |
| 后置条件 | 完成收款单，交由总经理审批 |
| Account.addPayment | 语法 | Public ResultMessage addPayment(String customer\_name, String clerk, String account\_name, int money, int all\_money, String ps) |
| 前置条件 | 完成付款时，输入的money合法 |
| 后置条件 | 完成付款单，交由总经理审批 |
| 需要的服务 | | |
| AccountDataService.getAccount() | | 获得当前账户信息 |
| AccountDataService.addAccount(AccountPO po) | | 添加账户AccountPO对象 |
| AccountDataService.delAccount(AccountPO po) | | 删除账户AccountPO对象 |
| AccountDataService.updateAccount(AccountPO po1, AccountPO po2) | | 更新账户AccountPO对象 |
| AccountDataService.findAccount(String name) | | 根据name查找账户对象 |
| AccountDataService.getAllAccount() | | 获得所有的账户对象 |
| AccountDataService.addReceipt(ReceiptPO po) | | 添加收款单ReceiptPO对象 |
| AccountDataService.addPayment(PaymentPO po) | | 添加付款单PaymentPO对象 |
| AccountDataService.getAllReceipt() | | 获得所有的收款单对象 |
| AccountDataService.getAllPayment() | | 获得所有的付款单对象 |
| AccountDataService.clear | | 清除所有对象 |

(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由各自的控制器委托给不同的领域对象。

### 4.1.3 financialbl模块

(1)模块概述

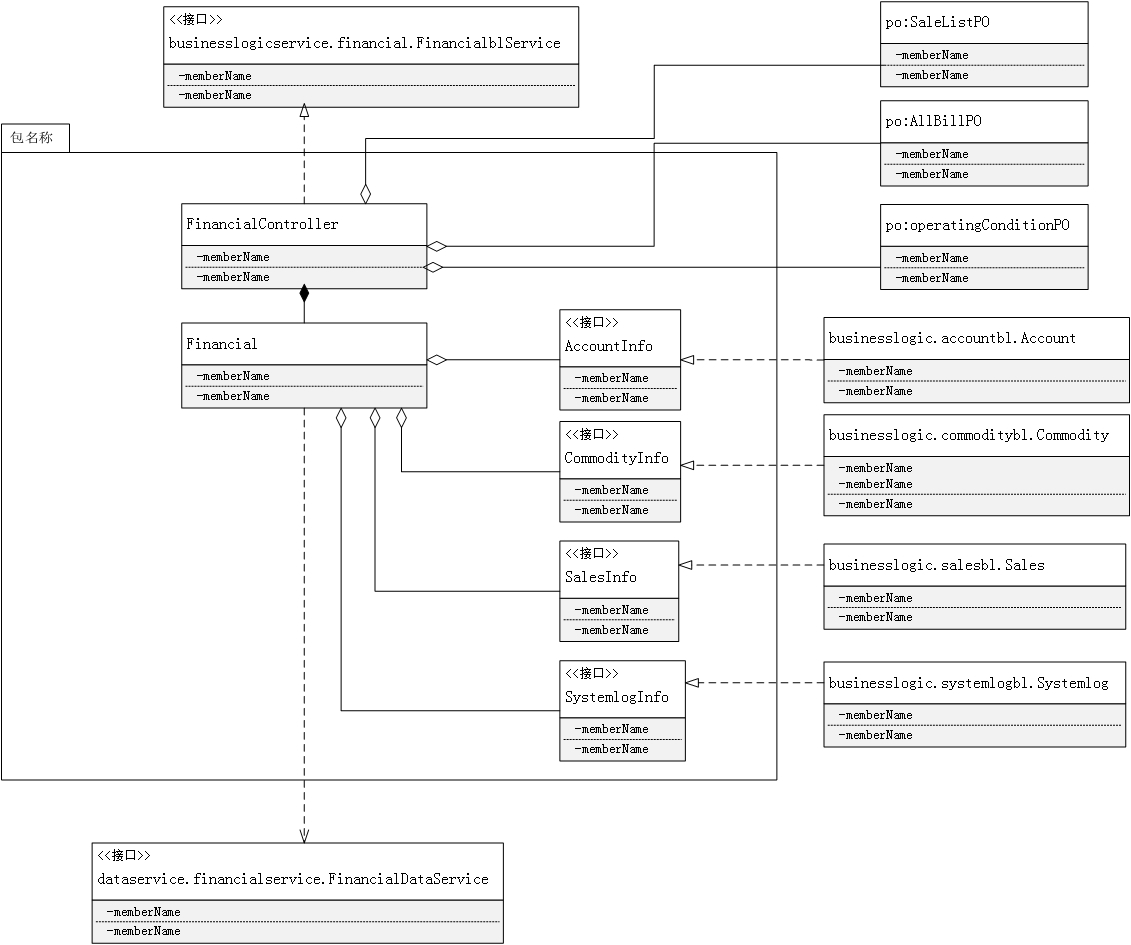
Financialbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Financialbl模块的职责及借款参见软件体系结构描述文档。

(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。Financial模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.financialblservice.FinancialblService接口与业务逻辑层与数据层之间的dataservice.financialdataservice.FinancialblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了FinancialController，这样FinancialController会将对账户的业务逻辑处理委托给Financial对象。SalesListPO是作为销售明细的持久化对象被添加到设计模型中去的。AllBillPO是作为经营历程的持久化对象被添加到设计模型中去的。OperatingConditionPO是作为经营情况的持久化对象被添加到设计模型中去的。SystemlogInfo、AccountInfo、SalesInfo、CommodityInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Financialbl模块的设计如图



Financialbl模块各个类的职责如表

|  |  |
| --- | --- |
| 模块 | 职责 |
| FinancialController | 负责实现财务表单界面所需要的服务 |
| Financial | 财务账单领域模型对象，拥有经过加工后的账单信息，可以帮助完成财务表单界面所需要的服务 |

(3)模块内部类的接口规范

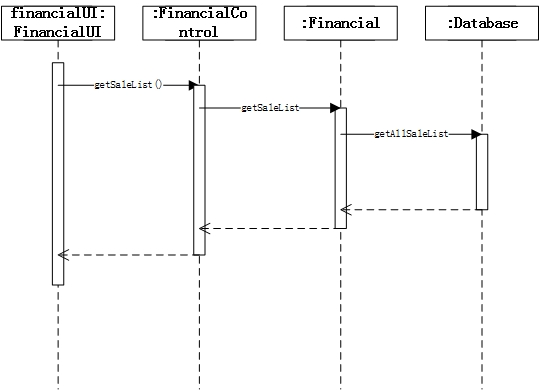
FinancialController接口规范：

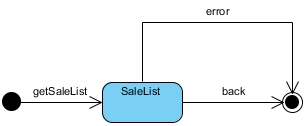
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| FinancialController.saleList | 语法 | Public ResultMessage saleList(String time1, String time2, String good\_name, String good\_type, String customer\_name, String clerk, int warehouse) |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的saleList方法 |
| FinancialController.allBill | 语法 | Public ResultMessage allBill(String time1, String time2, String note\_type, String customer\_name, String clerk, int warehouse) |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的allBill方法 |
| FinancialController.operatingCondition | 语法 | Public ResultMessage operatingCondition(String time1, String time2) |
| 前置条件 | 已创建一个Financial领域对象，输入符合规则 |
| 后置条件 | 调用Financial领域对象的operatingCondition方法 |
| 需要的服务 | | |
| Financial.saleList(String time1, String time2, String good\_name, String good\_type, String customer\_name, String clerk, int warehouse) | 根据条件查询销售明细 | |
| Financial.allBill(String time1, String time2, String note\_type, String customer\_name, String clerk, int warehouse) | 根据条件查询经营历程 | |
| Financial.operatingCondition(String time1, String time2) | 根据条件查询经营情况 | |

Financial接口规范：

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Financial.saleList | 语法 | Public ResultMessage saleList(String time1, String time2, String good\_name, String good\_type, String customer\_name, String clerk, int warehouse) |
| 前置条件 | 操作员是经过验证的财务人员 |
| 后置条件 | 查询销售明细，返回操作是否成功 |
| Financial.allBill | 语法 | Public ResultMessage allBill(String time1, String time2, String note\_type, String customer\_name, String clerk, int warehouse) |
| 前置条件 | 操作员是经过验证的财务人员 |
| 后置条件 | 查询单据，返回操作是否成功 |
| Financial.operatingCondition | 语法 | Public ResultMessage operatingCondition(String time1, String time2) |
| 前置条件 | 操作员是经过验证的财务人员 |
| 后置条件 | 查看经营情况，返回操作是否成功 |
| 需要的服务 | | |
| FinancialDataService.getSaleList() | 获得所有销售明细对象 | |
| FinancialDataService.getAllBill() | 获得所有经营历程对象 | |
| FinancialDataService.getOperatingCondition() | 获得所有经营情况对象 | |
| FinancialDataService.addSaleList(SaleListPO po) | 添加销售明细的单据对象 | |
| FinancialDataService.addAllBill(AllBillPO po) | 添加经营历程的单据对象 | |
| FinancialDataService.addOperatingCondition(OperatingConditionPO po) | 添加经营情况的单据对象 | |
| FinancialDataService.clear() | 清空所有对象 | |

(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由各自的控制器委托给不同的领域对象。

### 4.1.4 initializationbl 模块

(1)模块描述

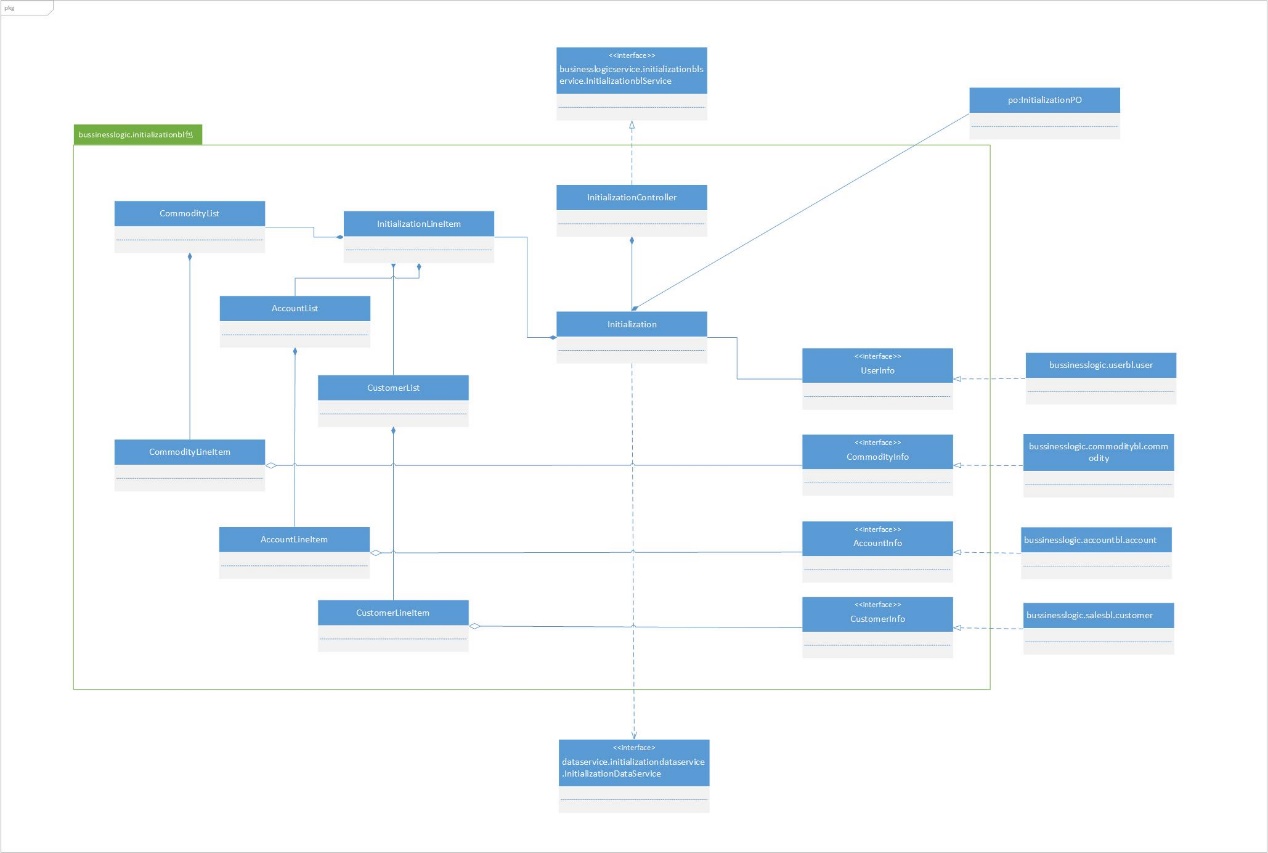
Initializationbl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Initializationbl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.initializationbl.InitializationblService接口。业务逻辑层和数据层之间添加dataservice.intializationdata.InitializtionDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InitializationController，这样InitializationController会将对期初建账的业务逻辑处理委托给Initialization对象。InitializationPO时作为期初建账记录的持久化对象被添加到设计模型中去的。而CommodityLineItem,AccountLineItem,CustomerLineItem和CommodityList，AccountList，CustomerList的添加分别是CommodityInfo，AccountInfo，CustomerInfo的容器类。CommodityLineItem，CustomerLineItem和AccountLineItem分别包有商品信息（商品分类，某一商品的名称，类别，型号，进价/售价（默认为上年平均），最近进价和最近售价（留空）），客户信息（客户分类，某一客户的名称，联系方式，应收应付），银行账户信息（账户名称，余额）。而CommodityList，CustomerList和AccountList分别封装了关于添加的CommodityLineItem，CustomerLineItem和AccountLineItem的数据集合的数据结构的秘密。InitializationLineItem封装了CommodityList，CustomerList和AccountList的数据集合的数据结构的秘密。UserInfo,CommodityInfo,AccountInfo和CustomerInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Initializationbl模块的设计如下图所示



Initializationbl模块各个类的设计

Initializationbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| InitializationController | 负责实现期初建账界面所需要的服务。 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名，ID，职位和密码，可以解决登录问题。 |
| Initialization | 期初建账的领域模型对象，拥有一次期初建账所持有的新增商品信息，客户信息和账户信息，可以帮助完成期初建账界面所需要的服务。 |

(3)模块内部类的接口规范

InitializationController 和 Initialization的接口规范如下表

InitializationController的接口规范

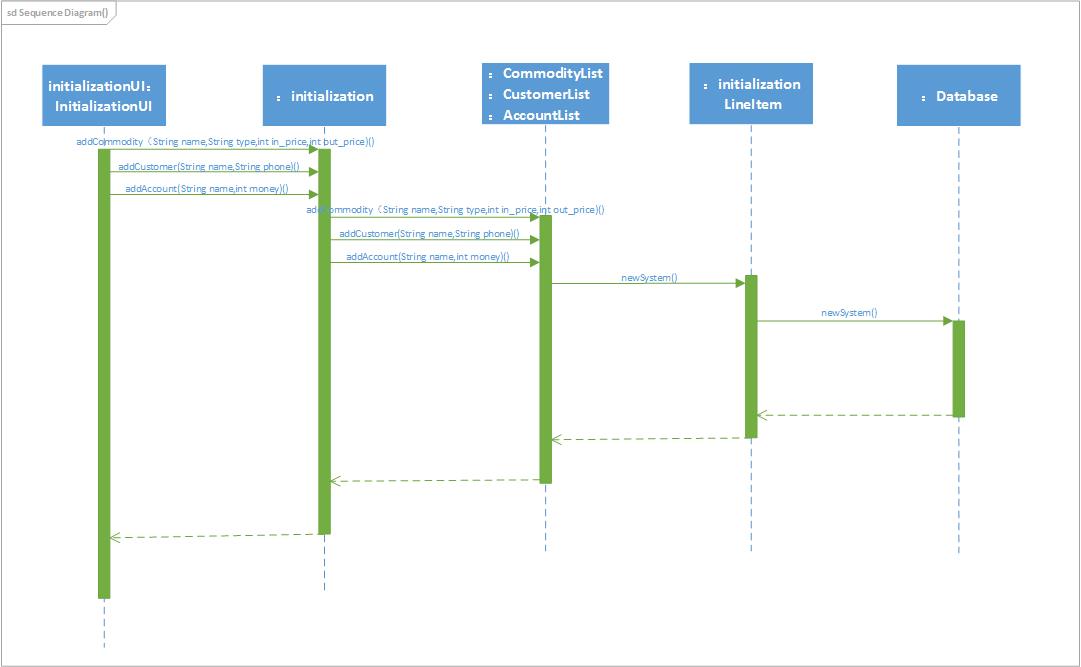
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| InitializationController.newSystem | 语法 | | Public ResultMessage newSystem() |
| 前置条件 | | 已创建一个Initialization领域对象，并且操作人员输入完成期初建账的信息，包括商品，客户，账户。 |
| 后置条件 | | 调用Initialization领域对象的newSystem方法 |
| InitializationController.addCommodity | 语法 | | Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addCommodity方法 |
| InitializationController.addCustomer | 语法 | | Public ResultMessage addCustomer(String name, String phone) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addCustomer方法 |
| InitializationController.addAccount | 语法 | | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | | 已创建一个Initialization领域对象，并且输入符合规则。 |
| 后置条件 | | 调用Initialization领域对象的addAccount方法 |
| InitializationController.showInformation | 语法 | | Public ResultMessage showInformation() |
| 前置条件 | | 已创建一个Initialization领域对象。 |
| 后置条件 | | 调用Initialization领域对象的showInformation方法 |
| 需要的服务（需接口） | | | |
| 服务名 | | 服务 | |
| Initialization.newSystem | 创建一个新的系统，并替换旧系统。 | | |
| Initialization.addCommodity | 记录此商品信息，等待替换库存内商品 | | |
| Initialization.addCustomer | 记录此客户信息，等待替换客户信息 | | |
| Initialization.addAccount | 记录此账户信息，等待替换账户信息 | | |
| Initialization.showInformation | 返回显示期初信息 | | |

Initialization的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Initialization.newSystem | 语法 | Public ResultMessage newSystem() |
| 前置条件 | 操作人员输入完成期初建账的信息，包括商品，客户，账户 |
| 后置条件 | 旧系统内容将被删除，新系统替换旧系统 |
| Initialization.addCommodity | 语法 | Public ResultMessage addCommodity(String name, String type, int in\_price, int out\_price) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此商品信息，等待替换库存内商品 |
| Initialization.addCustomer | 语法 | Public ResultMessage addCustomer(String name, String phone) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此客户信息，等待替换客户信息 |
| Initialization.addAccount | 语法 | Public ResultMessage addAccount(String name, int money) |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 记录此账户信息，等待替换账户信息 |
| Initialization.showInformation | 语法 | Public ResultMessage showInformation() |
| 前置条件 | 操作人员为财务人员 |
| 后置条件 | 返回期初信息 |
| 需要的服务 | | |
| InitializationDataService.getInformation | 查看InitializationPO对象的信息 | |

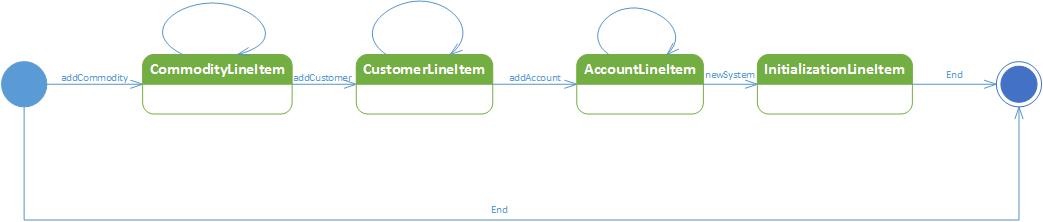
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户输入要添加的商品信息、客户信息和账户信息之后，期初建账逻辑处理的相关对象之间协作。



期初建账的顺序图

下图所示的状态图描述了Initialization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addCommodity方法被UI调用，Initialization进入CommodityLineItem状态，接着addCustomer方法被 UI调用，Initialization进入CustomerLineItem状态，接着addAccount方法被UI调用，Initialization进入AccountLineItem状态。之后通过添加新帐进入InitializationLineItem状态。



Initialization对象状态图

(5)业务逻辑层的设计原理

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

### 4.1.5 invoice 模块

(1)模块描述

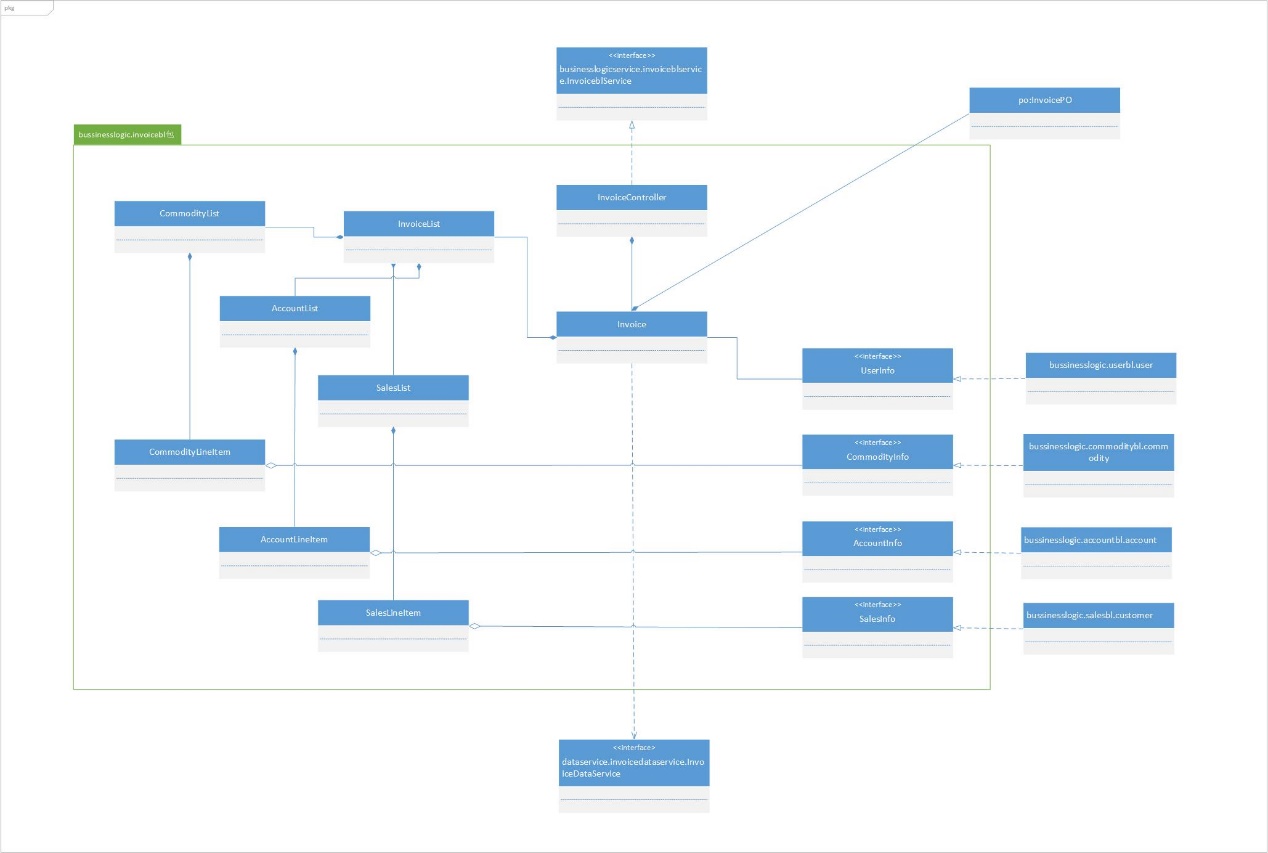
Invoicebl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Invoicebl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.invoicebl.InvoiceblService接口。业务逻辑层和数据层之间添加dataservice.invoicedata.InvoiceDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InvoiceController，这样InvoiceController会将审批单据的业务逻辑处理委托给Invoice对象。InvoicePO是作为审批单据记录的持久化对象被添加到设计模型中去的。而CommodvityLineItem,AccountLineItem,SalesLineItem和CommodityList，AccountList，SalesList的添加分别是CommodityInfo，AccountInfo，SalesInfo的容器类。CommodityLineItem，SalesLineItem和AccountLineItem分别包有库存进货单信息，进货单、销售单、进货退货单、销售退货单信息，收款单、付款单信息。而CommodityList，SalesList和AccountList分别封装了关于添加的CommodityLineItem，SalesLineItem和AccountLineItem的数据集合的数据结构的秘密。InvoiceList封装了CommodityList，SalesList和AccountList的数据集合的数据结构的秘密。UserInfo,CommodityInfo,AccountInfo和SalesInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Invoicebl模块的设计如下图所示：



Invoicebl模块各个类的设计

Invoicebl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| InvoiceController | 负责实现期初建账界面所需要的服务。 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名，ID，职位和密码，可以解决登录问题。 |
| Invoice | 审批单据的领域模型对象，拥有一次审批单据所持有的新增的单据信息，可以帮助完成审批单据界面所需要的服务。 |

(3)模块内部类的接口规范

InvoiceController的接口规范

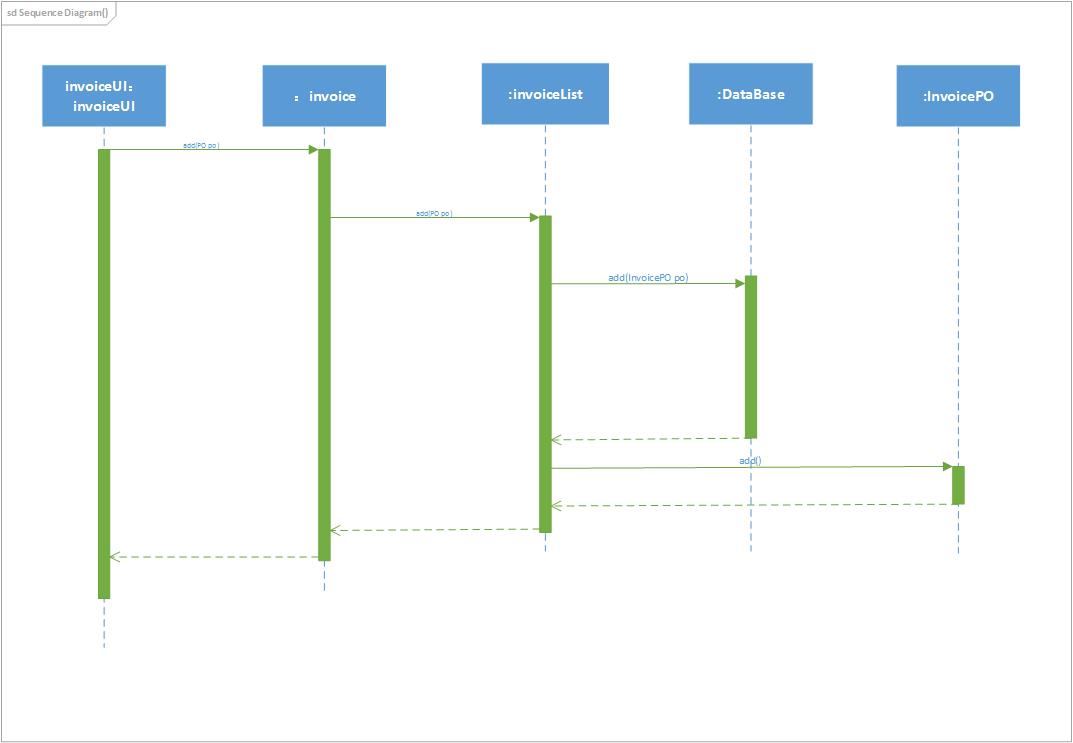
|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| InvoiceController.show | | 语法 | Public ResultMessage show() |
| 前置条件 | 已创建一个Invoice领域对象 |
| 后置条件 | 调用Invoice领域对象的show方法 |
| InvoiceController.pass | | 语法 | Public ResultMessage pass(int note) |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的pass方法 |
| InvoiceController.refuse | | 语法 | Public ResultMessage refuse(int note) |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的refuse方法 |
| InvoiceController.add | | 语法 | Public ResultMessage add(GiftPO po)  Public ResultMessage add(ImportPO po)  Public ResultMessage add(ExportPO po)  Public ResultMessage add(Import\_ReturnPO po)  Public ResultMessage add(Export\_ReturnPO po)  Public ResultMessage add(ReceiptPO po)  Public ResultMessage add(PaymentPO po) |
| 前置条件 | 已创建一个Invoice领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Invoice领域对象的add方法 |
| 需要的服务（需接口） | | | |
| 服务名 | 服务 | | |
| Invoice.show | 显示待审核单据 | | |
| Invoice.pass | 通过单据审批 | | |
| Invoice.refuse | 拒绝单据审批 | | |
| Invoice.add | 将新生成的单据加入待审批单据，等待总经理审批 | | |

Invoice的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Invoice.show | 语法 | | Public ResultMessage show() |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 开始审核单据，返回单据信息 |
| Invoice.pass | 语法 | | Public ResultMessage pass(int note) |
| 前置条件 | | 操作人员为总经理，存在正在审批的单据 |
| 后置条件 | | 按照单据内容修改库存、客户、账户信息 |
| Invoice.refuse | 语法 | | Public ResultMessage refuse(int note) |
| 前置条件 | | 操作人员为总经理 |
| 后置条件 | | 拒绝单据 |
| Invoice.add | 语法 | | Public ResultMessage add(GiftPO po)  Public ResultMessage add(ImportPO po)  Public ResultMessage add(ExportPO po)  Public ResultMessage add(Import\_ReturnPO po)  Public ResultMessage add(Export\_ReturnPO po)  Public ResultMessage add(ReceiptPO po)  Public ResultMessage add(PaymentPO po) |
| 前置条件 | | 操作人员为总经理，库存管理人员，销售人员、财务人员填写好单据 |
| 后置条件 | | 等待总经理审批 |
| 需要的服务 | | | |
| InvoiceDataServoice.getInvoice() | | 获得一个单据InvoicePO对象 | |
| InvoiceDataServoice.addInvoice(InvoicePO po) | | 添加单据InvoicePO对象 | |
| InvoiceDataServoice.delInvoice(InvoicePO po) | | 移除单据InvoicePO对象 | |
| InvoiceDataService.getAllInvoice() | | 获得所有单据InvoicePO对象 | |
| InvoiceDataService.clear() | | 清空所有单据 | |

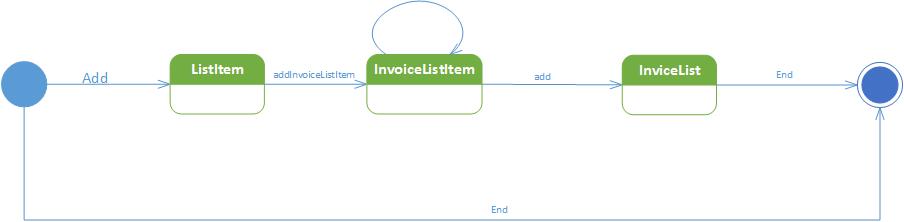
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户创建新的进货单、销售单、进货退货单、销售退货单、收款单、付款单或库存赠送单之后，总经理审批单据逻辑处理的相关对象之间协作。



审批单据的顺序图

下图所示的状态图描述了Invoice对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着add方法被UI调用，Invoice进入ListItem状态，接着addInvoiceListItem方法调用，Invoice进入InvoiceLineItem状态。之后通过向InvoiceList中添加新的待审批的单据要进入InvoiceList状态。



Invoice对象状态图

(5)业务逻辑层的设计原理

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

### 4.1.6 promotion 模块

(1)模块描述

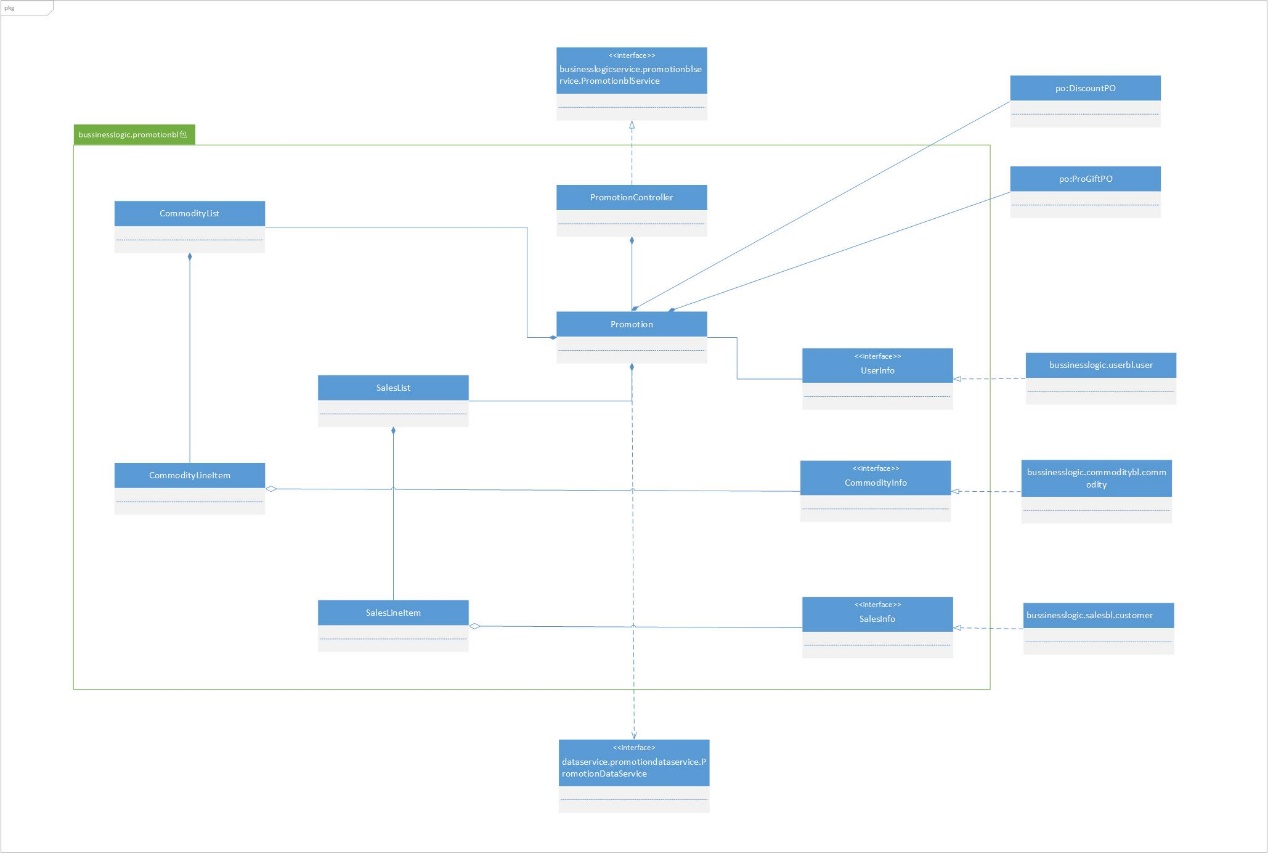
Promotionbl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Promotionbl 模块的职责及接口参见软件体系结构描述文档表10

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.promotionbl.PromotionblService接口。业务逻辑层和数据层之间添加dataservice.promotiondata.InvoiceDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PromotionController，这样PromotionController会将制定促销策略的业务逻辑处理委托给Promotion对象。DiscountPO，ProGiftPO是作为促销策略记录的持久化对象被添加到设计模型中去的。而CommodvityLineItem,SalesLineItem和CommodityList，SalesList的添加分别是CommodityInfo，SalesInfo的容器类。CommodityLineItem，SalesLineItem分别包有要赠送的商品信息和销售中的折扣信息，并根据该信息对销售做相应改变。而CommodityList，SalesList分别封装了关于添加的CommodityLineItem和SalesLineItem的数据集合的数据结构的秘密。PromotionList封装了CommodityList和SalesList的数据集合的数据结构的秘密。UserInfo,CommodityInfo和SalesInfo都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。

Promotionbl模块的设计如下图所示：



Promotionbl模块各个类的设计

Invoicebl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| PromotionController | 负责实现制定销售策略界面所需要的服务。 |
| User | 系统用户的领域模型对象，拥有用户数据的姓名，ID，职位和密码，可以解决登录问题。 |
| Promotion | 制定销售策略的领域模型对象，拥有一次制定销售策略所持有的新增的销售策略信息，可以帮助完成制定销售策略界面所需要的服务。 |
| 模 块 | 职 责 |
| LoginController | 负责实现对应于登录界面所需要的服务。 |
| InitializationController | 负责实现期初建账界面所需要的服务。 |

(3)模块内部类的接口规范

PromotionController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionController.makeDiscount | 语法 | Public ResultMessage makeDiscount(int start\_money, int discount, int end\_money, String time1, String time2) |
| 前置条件 | 已创建一个Promotion领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Promotion领域对象的makeDiscount方法 |
| PromotionController.delPromotion | 语法 | Public ResultMessage delPromotion(String note) |
| 前置条件 | 已创建一个Promotion的领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Promotion领域对象的delPromotion方法 |
| PromotionController.makeGift | 语法 | Public ResultMessage makeGift(start\_money, String time1, String time2) |
| 前置条件 | 已创建一个Promotion的领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Promotion领域对象的makeGift方法 |
| PromotionController.search() | 语法 | Public ResultMessage search() |
| 前置条件 | 已创建一个Promotion的领域对象 |
| 后置条件 | 调用Promotion领域对象的search方法 |
| PromotionController.makeOther | 语法 | Public ResultMessage makeOther(String word) |
| 前置条件 | 已创建一个Promotion的领域对象 |
| 后置条件 | 调用Promotion领域对象的makeOther方法 |

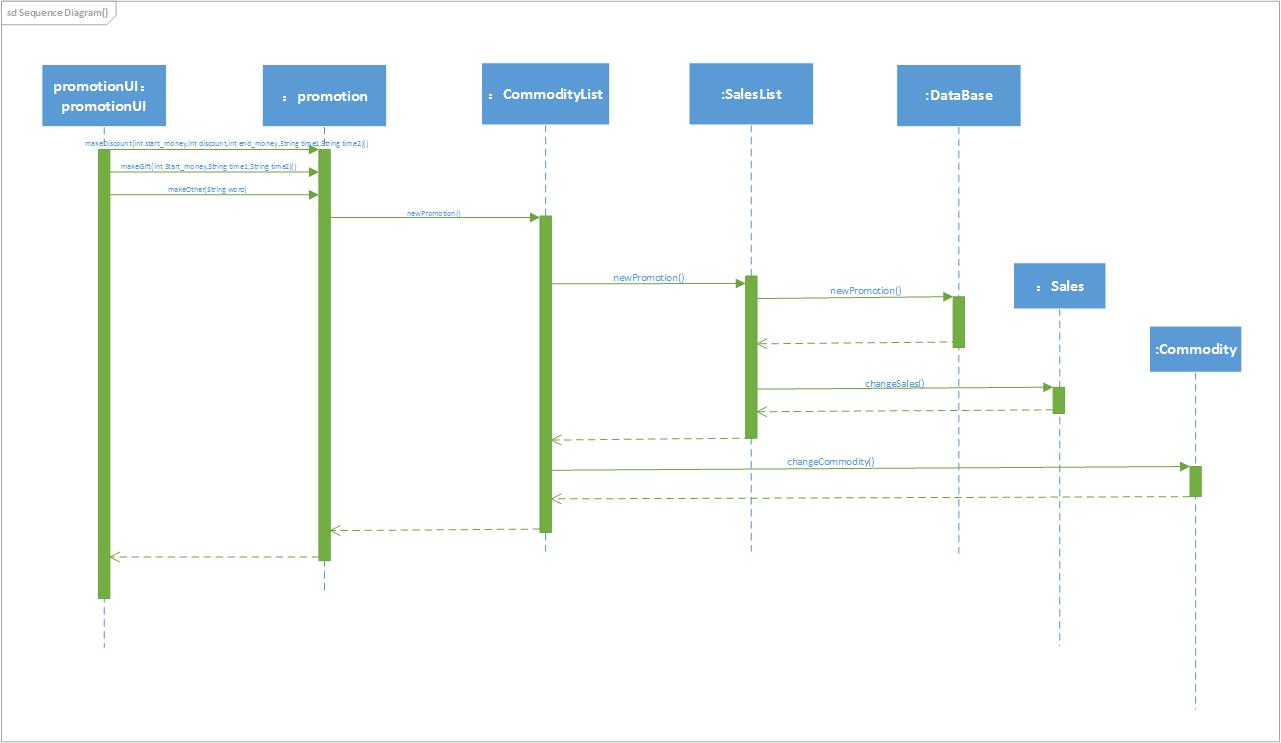
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| 服务名 | 服务 |
| Promotion.makeDiscount | 制作折价优惠 |
| Promotion.delPromotion | 取消优惠 |
| Promotion.makeGift | 制作赠品优惠 |
| Promotion.search() | 返回显示的促销策略信息 |
| Promotion.makeOther | 制作其他优惠 |

Promotion的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| Promotion.makeDiscount | 语法 | | Public ResultMessage makeDiscount(int start\_money, int discount, int end\_money, String time1, String time2) |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 制作好折价优惠之后，立即更新至销售 |
| Promotion.delPromotion | 语法 | | Public ResultMessage delPromotion(String note) |
| 前置条件 | | 存在已经制定好的销售策略，操作用户为总经理 |
| 后置条件 | | 取消优惠后，立即更新至销售 |
| Promotion.makeGift | 语法 | | Public ResultMessage makeGift(start\_money, String time1, String time2) |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 制作好赠品优惠之后，立即更新至销售 |
| Promotion.search() | 语法 | | Public ResultMessage search() |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 返回显示的促销策略信息 |
| Promotion.makeOther | 语法 | | Public ResultMessage makeOther(String word) |
| 前置条件 | | 操作用户为总经理 |
| 后置条件 | | 制作好其他优惠之后，立即更新至销售 |
| 需要的服务（需接口） | | | |
| PromotionDataService.addDiscount(DiscountPO po) | | 添加折价促销DiscountPO方案对象 | |
| PromotionDataService.addGift(ProGiftPO po) | | 添加赠品促销ProGiftPO方案对象 | |
| PromotionDataService.addRecord() | | 添加促销记录 | |
| PromotionDataService.delDiscount(DiscountPO po) | | 删除折价促销DiscountPO方案对象 | |
| PromotionDataService.delGift(ProGiftPO po) | | 删除赠品促销ProGiftPO方案对象 | |
| PromotionDataservice.getDiscount() | | 获得当前折价促销方案对象 | |
| PromotionDataService.getGift() | | 获得当前赠品促销方案对象 | |
| PromotionDataService.show() | | 获得历史所有促销方案对象 | |

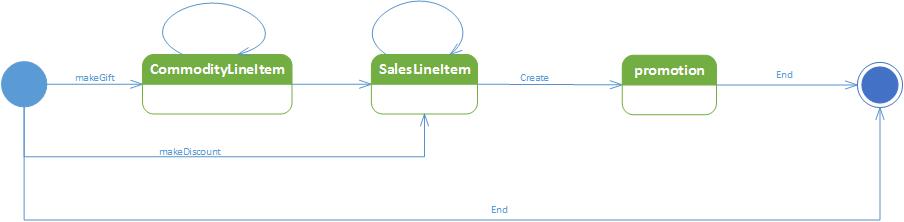
(4)业务逻辑层的动态模型

下图表明了在进销存系统中，当用户创建新的进货单、销售单、进货退货单、销售退货单、收款单、付款单或库存赠送单之后，总经理审批单据逻辑处理的相关对象之间的协作。



创建新的销售策略的顺序图

下图所示的状态图描述了Promotion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makeGift方法被UI调用，Promotion进入CommodityLineItem状态，接着进入SalesLineItem状态。UI可以通过直接调用makeDiscount方法直接进入SalesLineItem状态。之后通过创建新的销售策略进入promotion状态。



Promotion对象状态图

(5)业务逻辑层的设计原理

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

### 4.1.7 salesbl模块

(1)模块概述

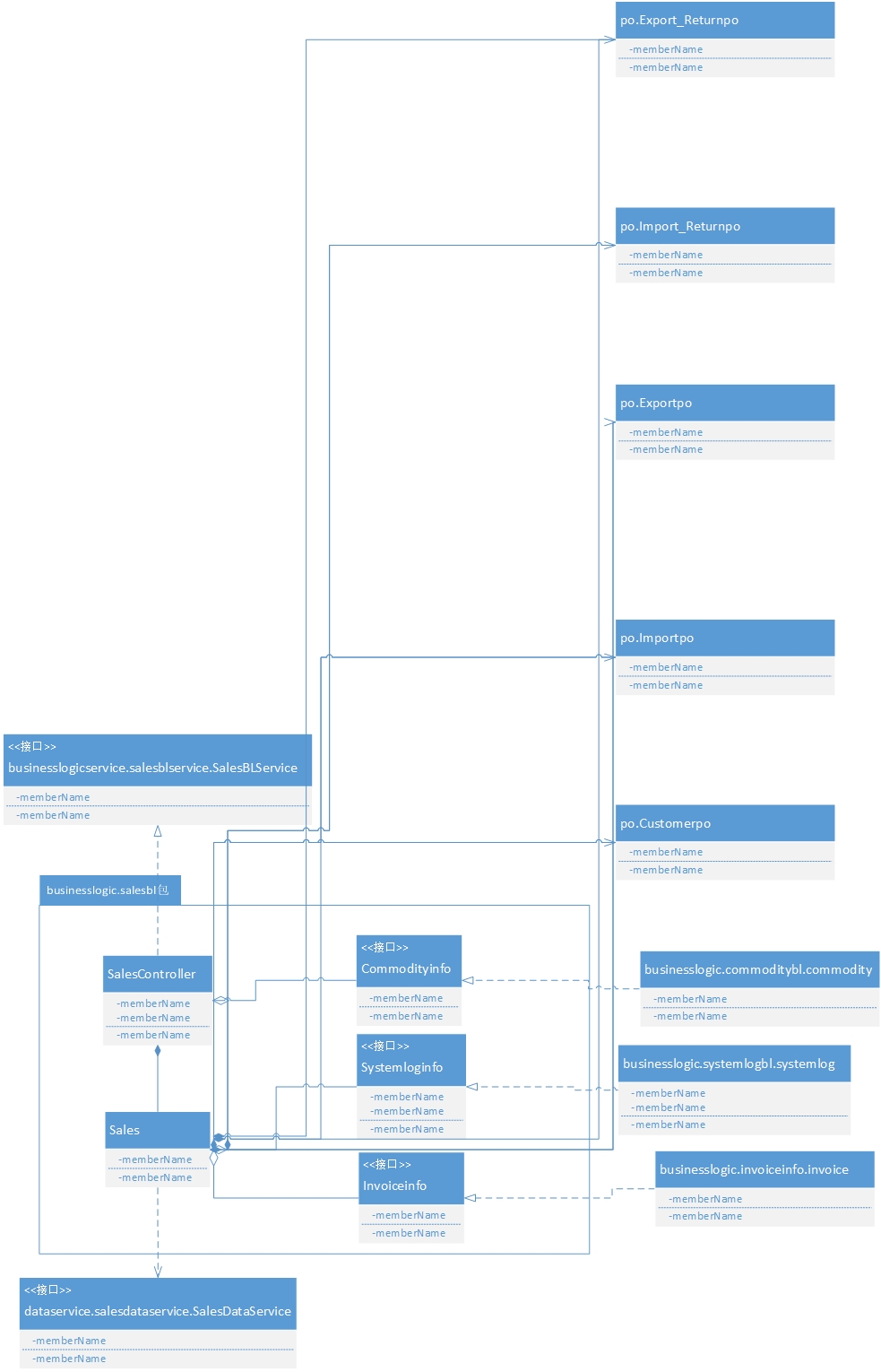
salesbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

salesbl模块的职责及接口参见软件系统结构描述文档表9中的salesbl的接口规范。

(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。sales模块对应的接口为展示层与业务逻辑层之间的businesslogicservice.salesblservice.SalesblService接口与业务逻辑层与数据层之间的dataservice.salesdataservice.SalesblService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SalesController，这样SalesController会将对销售的业务逻辑处理委托给Sales对象。ImportPO是作为进货的持久化对象被添加到设计模型中去的。Import\_ReturnPO是作为进货退货的持久化对象被添加到设计模型中去的。CustomerPO是作为客户的持久化对象被添加到设计模型中去的。ExportPO是作为销售的持久化对象被添加到设计模型中去的。Export\_ReturnPO是作为销售退货的持久化对象被添加到设计模型中去的。SystemlogInfo、InvoiceInfo、CustomerInfo都是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Salesbl模块的设计如图



(3)模块内部类的接口规范

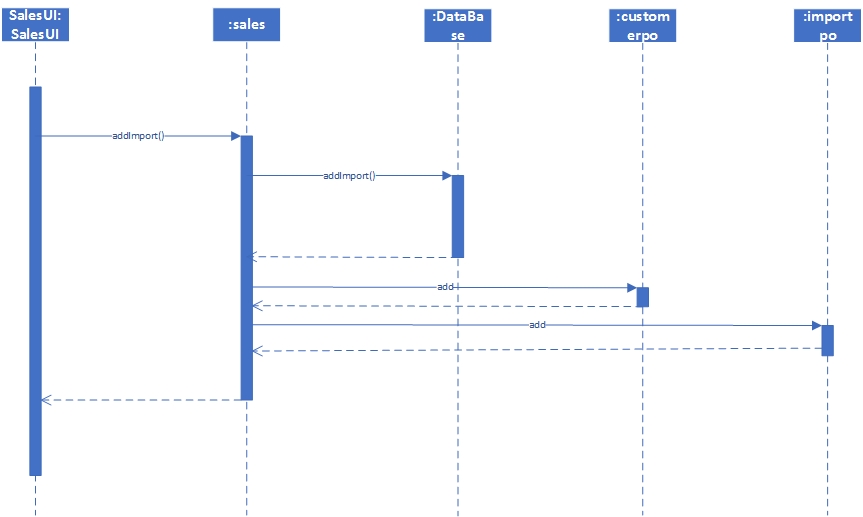
SalesController的接口规范

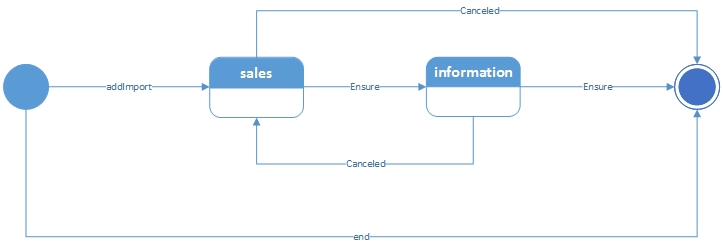
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SalesController.addCustomer | 语法 | Public ResultMessage addCustomer(String name, int level, String phone, String zip, String mail, int money, Stirng clerk) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的addCustomer方法 |
| SalesController.delCustomer | 语法 | Public ResultMessage delCustomer(String name) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的delCustomer方法 |
| SalesController.updateCustomer | 语法 | Public ResultMessage updateCustomer(String name, int level, String phone, String zip, String mail, int money, Stirng clerk) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的updateCustomer方法 |
| SalesController.searchCustomer | 语法 | Public ResultMessage searchCustomer(String word) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的searchCustomer方法 |
| SalesController.addImport | 语法 | Public ResultMessage addImport(String name, int warehouse, String good\_name, String good\_type, String ps, int number) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的addImport方法 |
| SalesController.addImport\_Return | 语法 | Public ResultMessage addImport\_Return(String name, int warehouse, String good\_name, String good\_type, String ps, int number)  Public ResultMessage addImport\_Return(String note, int number) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的addImport\_Return方法 |
| SalesController.addExport | 语法 | Public ResultMessage addExport(String name, int warehouse, String good\_name, String good\_type, String ps, int number, int price, int discount) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的addExport方法 |
| SalesController.addExport\_Return | 语法 | Public ResultMessage addExport\_Return(String name, int warehouse, String good\_name, String good\_type, String ps, int number, int price, int discount)  Public ResultMessage addExport\_Return(String note, int number) |
| 前置条件 | 已创建一个Sales领域对象，输入符合规则 |
| 后置条件 | 调用Sales领域对象的addExport\_Return方法 |
| 需要的服务 | | |
| Sales.addCustomer(CustomerPO po) | 添加客户 | |
| Sales.delCustomer(CustomerPO po) | 删除客户 | |
| Sales.updateCustomer(CustomerPO po1, CustomerPO po2) | 更新客户 | |
| Sales.findCustomer(String name) | 根据name查找客户 | |
| Sales.addImport(ImportPO po) | 添加进货单 | |
| Sales.addImport\_Return(Import\_ReturnPO po) | 添加进货退货单 | |
| Sales.addExport(ExportPO po) | 添加销售单 | |
| Sales.addExport\_Return(Export\_ReturnPO po) | 添加销售退货单 | |

Salesbl的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Sales.addCustomer | 语法 | Public ResultMessage addCustomer(String name, int level, String phone, String zip, String mail, int money, Stirng clerk) |
| 前置条件 | 输入的phone,zip和money合法 |
| 后置条件 | 返回提示是否成功添加客户 |
| Sales.delCustomer | 语法 | Public ResultMessage delCustomer(String name) |
| 前置条件 | 无 |
| 后置条件 | 返回提示是否成功删除客户 |
| Sales.updateCustomer | 语法 | Public ResultMessage updateCustomer(String name, int level, String phone, String zip, String mail, int money, Stirng clerk) |
| 前置条件 | 有客户的信息发生改变，需要更新 |
| 后置条件 | 返回提示是否成功更新客户信息 |
| Sales.searchCustomer | 语法 | Public ResultMessage searchCustomer(String word) |
| 前置条件 | 无 |
| 后置条件 | 模糊搜索，返回搜索结果 |
| Sales.addImport | 语法 | Public ResultMessage addImport(String name, int warehouse, String good\_name, String good\_type, String ps, int number) |
| 前置条件 | 发生进货时，输入的warehouse和number合法 |
| 后置条件 | 完成进货单后交由总经理审批 |
| Sales.addImport\_Return | 语法 | Public ResultMessage addImport\_Return(String name, int warehouse, String good\_name, String good\_type, String ps, int number)  Public ResultMessage addImport\_Return(String note, int number) |
| 前置条件 | 发生进货退货时，输入的warehouse和number合法 |
| 后置条件 | 完成进货退货后，交由总经理审批 |
| Sales.addExport | 语法 | Public ResultMessage addExport(String name, int warehouse, String good\_name, String good\_type, String ps, int number, int price, int discount) |
| 前置条件 | 发生销售时，输入的warehouse、number、price和discount合法 |
| 后置条件 | 完成销售单，交由总经理审批 |
| Sales.addExport\_Return | 语法 | Public ResultMessage addExport\_Return(String name, int warehouse, String good\_name, String good\_type, String ps, int number, int price, int discount)  Public ResultMessage addExport\_Return(String note, int number) |
| 前置条件 | 发生销售时，输入的warehouse、number、price和discount合法 |
| 后置条件 | 完成销售退货，交由总经理审批 |
| 需要的服务 | | |
| SalesDataService.addCustomer(CustomerPO po) | 添加客户CustomerPO对象 | |
| SalesDataService.delCustomer(CustomerPO po) | 删除客户CustomerPO对象 | |
| SalesDataService.updateCustomer(CustomerPO po1, CustomerPO po2) | 更新客户CustomerPO对象信息 | |
| SalesDataService.findCustomer(String name) | 根据name查找客户Customer对象 | |
| SalesDataService.addImport(ImportPO po) | 添加进货单ImportPO对象 | |
| SalesDataService.addImport\_Return(Import\_ReturnPO po) | 添加进货退货单Import\_ReturnPO对象 | |
| SalesDataService.addExport(ExportPO po) | 添加销售单ExportPO对象 | |
| SalesDataService.addExport\_Return(Export\_ReturnPO po) | 添加销售退货单Export\_ReturnPO对象 | |
| SalesDataService.getAllCustomer() | 获得所有的客户对象 | |
| SalesDataService.getAllImport() | 获得所有的进货单对象 | |
| SalesDataService.getAllImport\_Return() | 获得所有的进货退货单对象 | |
| SalesDataService.getAllExport() | 获得所有的销售单对象 | |
| SalesDataService.getAllExport\_Return() | 获得所有的销售退货单对象 | |
| SalesDataService.clear() | 清除所有对象 | |

(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由各自的控制器委托给不同的领域对象。

### 4.1.8 systemlogbl模块

(1)模块概述

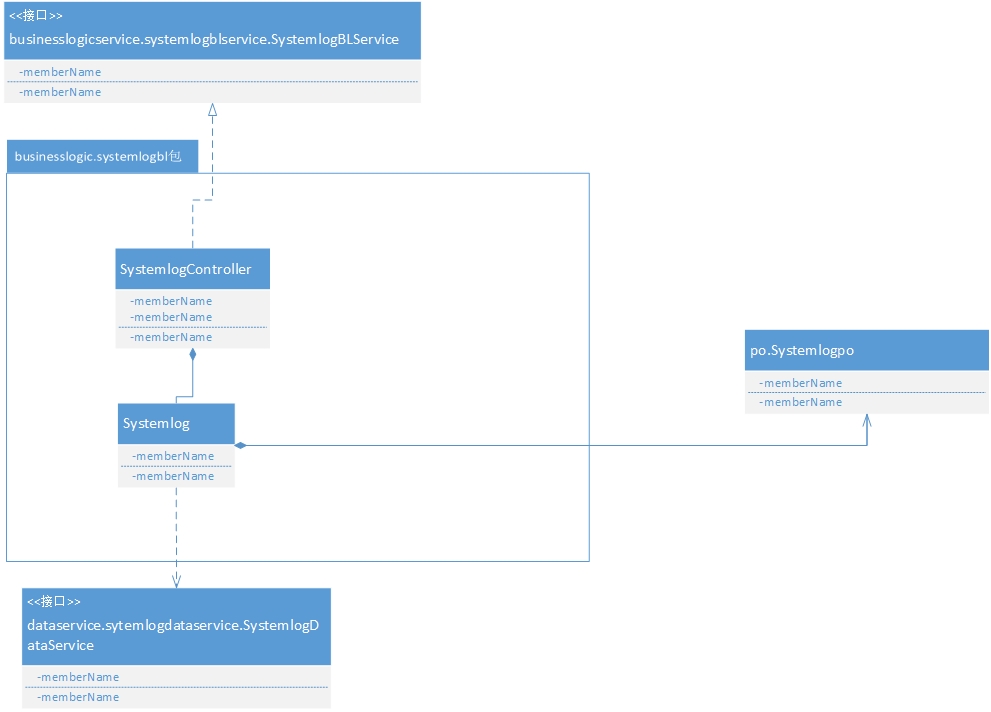
systemlogbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

systemlogbl模块的职责及接口参见软件系统结构描述文档表9中的systemlogbl的接口规范。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.systemlogblservice.SystemlogBLService接口。业务逻辑层和数据层之间添加dataservice.systemlogdataservice.SystemlogDataService接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了SystemlogController，这样SystemlogController会将对使用者的业务逻辑处理委托给Systemlog对象。SystemlogPO是作为使用者的持久化对象被添加到设计模型中去的。Systemloginfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Systemlogbl模块的设计如下图所示。



Systemlogbl模块各个类的职责如下表所示。

Systemlogbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| SystemlogController | 负责实现系统日志管理界面所需要的服务 |
| Systemlog | 系统日志管理的领域模型对象，拥有时间和操作等信息，可以帮助完成系统日志管理界面所需要的服务 |

(3)模块内部类的接口规范

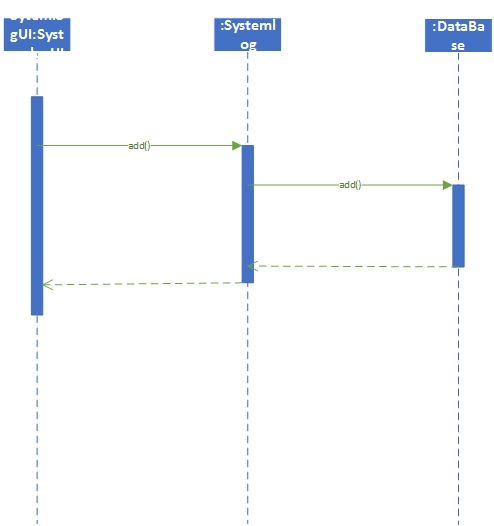
SystemlogController的接口规格

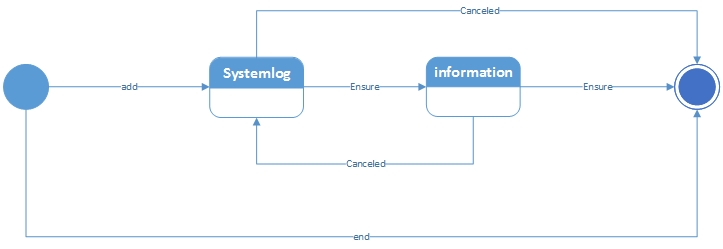
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| SystemlogController.show | 语法 | Public ResultMessage show() |
| 前置条件 | 已创建一个Systemlog领域对象，输入符合规则 |
| 后置条件 | 调用Systemlog领域对象的show方法 |
| SystemlogController.add | 语法 | Public ResultMessage add(String word) |
| 前置条件 | 已创建一个Systemlog领域对象，输入符合规则 |
| 后置条件 | 调用Systemlog领域对象的add方法 |
| 需要的服务 | | |
| Systemlog.show() | 显示日志内容 | |
| Systemlog.add(SystemlogPO po) | 添加日志内容 | |

Systemlog的接口规格

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Systemlog.show | 语法 | Public ResultMessage show() |
| 前置条件 | 操作人员为财务人员或总经理 |
| 后置条件 | 显示日志内容，返回是否成功 |
| Systemlog.add | 语法 | Public ResultMessage add(String word) |
| 前置条件 | 其他人员完成操作后，发送操作记录并存储 |
| 后置条件 | 添加日志内容，返回是否成功 |
| 需要的服务 | | |
| SystemlogDataService.get() | 获得日志SystemlogPO对象 | |
| SystemlogDataService.add(SystemlogPO po) | 添加日志SystemlogPO对象 | |
| SystemlogDataService.clear() | 清空日志对象 | |

(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由各自的控制器委托给不同的领域对象。

### 4.1.9 userbl模块

(1)模块概述

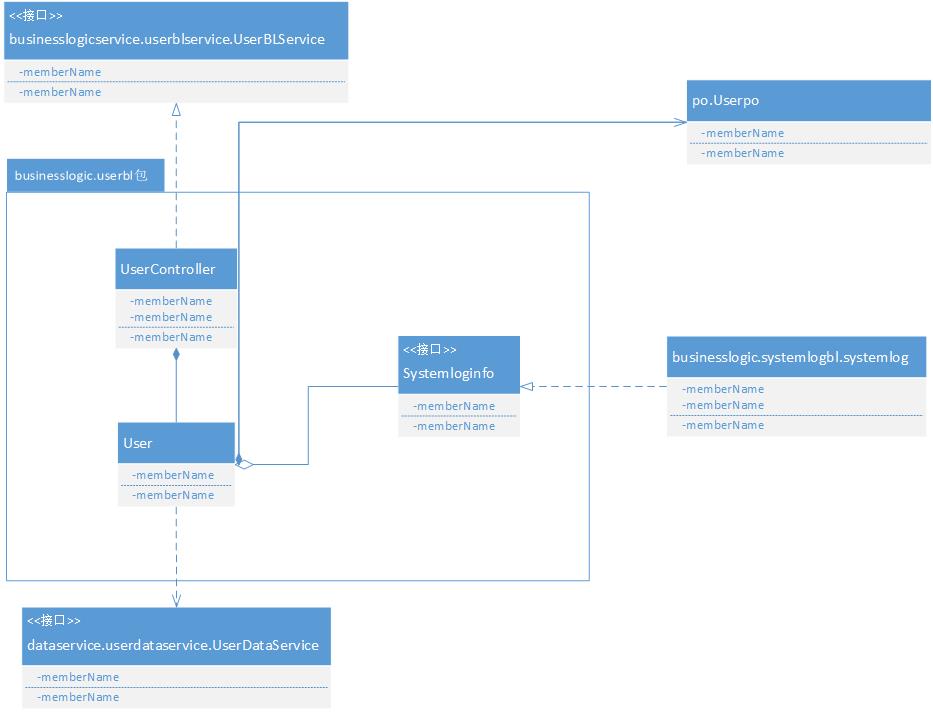
userbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

userbl模块的职责及接口参见软件系统结构描述文档表9中的userbl的接口规范。

(2)整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加businesslogicservice.userblservice.UserBLService接口。业务逻辑层和数据层之间添加dataservice.userdataservice.UserDataService接口。为了隔离业务逻辑指责和逻辑控制职责，我们增加了UserController，这样UserController会将对使用者的业务逻辑处理委托给User对象。UserPO是作为使用者的持久化对象被添加到设计模型中去的。Userinfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。

Userbl模块的设计如下图所示。



Userbl模块各个类的职责如下表所示。

Userbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务 |
| User | 用户管理的领域模型对象，拥有用户账号、密码、职责等信息，可以帮助完成用户管理界面所需要的服务 |

(3)模块内部类的接口规范

UserController和User的借口规范如下表所示

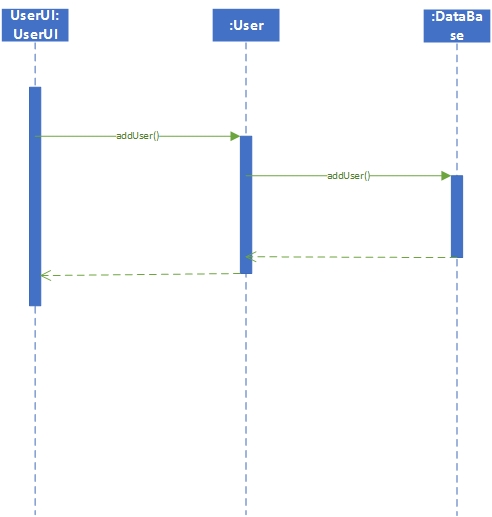
UserController的接口规范

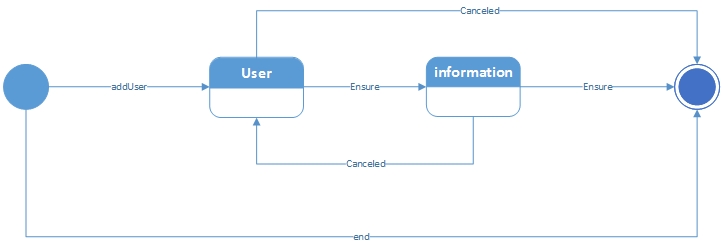
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| UserController.addUser | 语法 | Public ResultMessage(String name, String password, int duty) |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的addUser方法 |
| UserController.delUser | 语法 | Public ResultMessage(String name) |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的delUser方法 |
| UserController.show | 语法 | Public ResultMessage show() |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的show方法 |
| UserController.login | 语法 | Public ResultMessage login(String name, Stirng password) |
| 前置条件 | 已创建一个User领域对象，输入符合规则 |
| 后置条件 | 调用User领域对象的login方法 |
| 需要的服务 | | |
| User.addUser(UserPO po) | 添加用户 | |
| User.delUser(UserPO po) | 删除用户 | |
| User.show() | 显示用户信息 | |
| User.login() | 登陆 | |

User的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| User.addUser | 语法 | Public ResultMessage(String name, String password, int duty) |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 完成新账号的添加，可以开始使用 |
| User.delUser | 语法 | Public ResultMessage(String name) |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 删除账号信息 |
| User.show | 语法 | Public ResultMessage show() |
| 前置条件 | 管理员身份登陆系统 |
| 后置条件 | 返回显示的身份信息 |
| User.login | 语法 | Public ResultMessage login(String name, Stirng password) |
| 前置条件 | 有人尝试登陆时 |
| 后置条件 | 验证账号与密码，返回是否成功 |
| 需要的服务 | | |
| UserDataService.addUser(UserPO po) | 添加UserPO对象 | |
| UserDataService.delUser(UserPO po) | 删除UserPO对象 | |
| UserDataService.getUser(String name) | 根据名字查找UserPO对象 | |
| UserDataService.show() | 获得所有UserPO对象 | |
| UserDataService.clear() | 清空所有对象 | |

(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由各自的控制器委托给不同的领域对象。

# 5.依赖视角

下图是客户端和服务器端各自的包之间的依赖关系。

