**DS酒店管家**

**软件详细设计描述文档**

南京大学dark sky团队

常德隆

曹洪荣

陈进

目录

[一． 引言 3](#_Toc465373654)

[1.1编制目的 3](#_Toc465373655)

[二． 产品概述 3](#_Toc465373656)

[三． 体系结构设计概述 3](#_Toc465373657)

[四．结构视角 4](#_Toc465373658)

[4.1 userbl模块 4](#_Toc465373659)

[4.2 customerbl模块 4](#_Toc465373660)

[4.3 hotelbl模块 4](#_Toc465373661)

[4.4 roombl模块 4](#_Toc465373662)

[4.5 orderbl模块 4](#_Toc465373663)

[4.6 promotionbl模块 4](#_Toc465373664)

[4.7 webworkerbl模块 4](#_Toc465373665)

1. [依赖视角 4](#_Toc465373666)

# 

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **修改人员** | **日期** | **变更原因** | **版本号** |
| 常德隆 | 2016-10-31 | 最初草稿 | V1.0 草稿 |
| 常德隆 | 2016-11-6 | 补充了顺序图和状态图 | v2.0 |

# 一．引言

## 1.1编制目的

本报告详细完成对DS酒店管家系统的详细设计，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编写，是了解系统的导航。

# 二．产品概述

参考DS酒店管家系统用例文档和软件需求规格说明文档中对产品的概括描述。

# 三．体系结构设计概述

# 四．结构视角

## 4.1 userbl模块

**(1)模块描述**

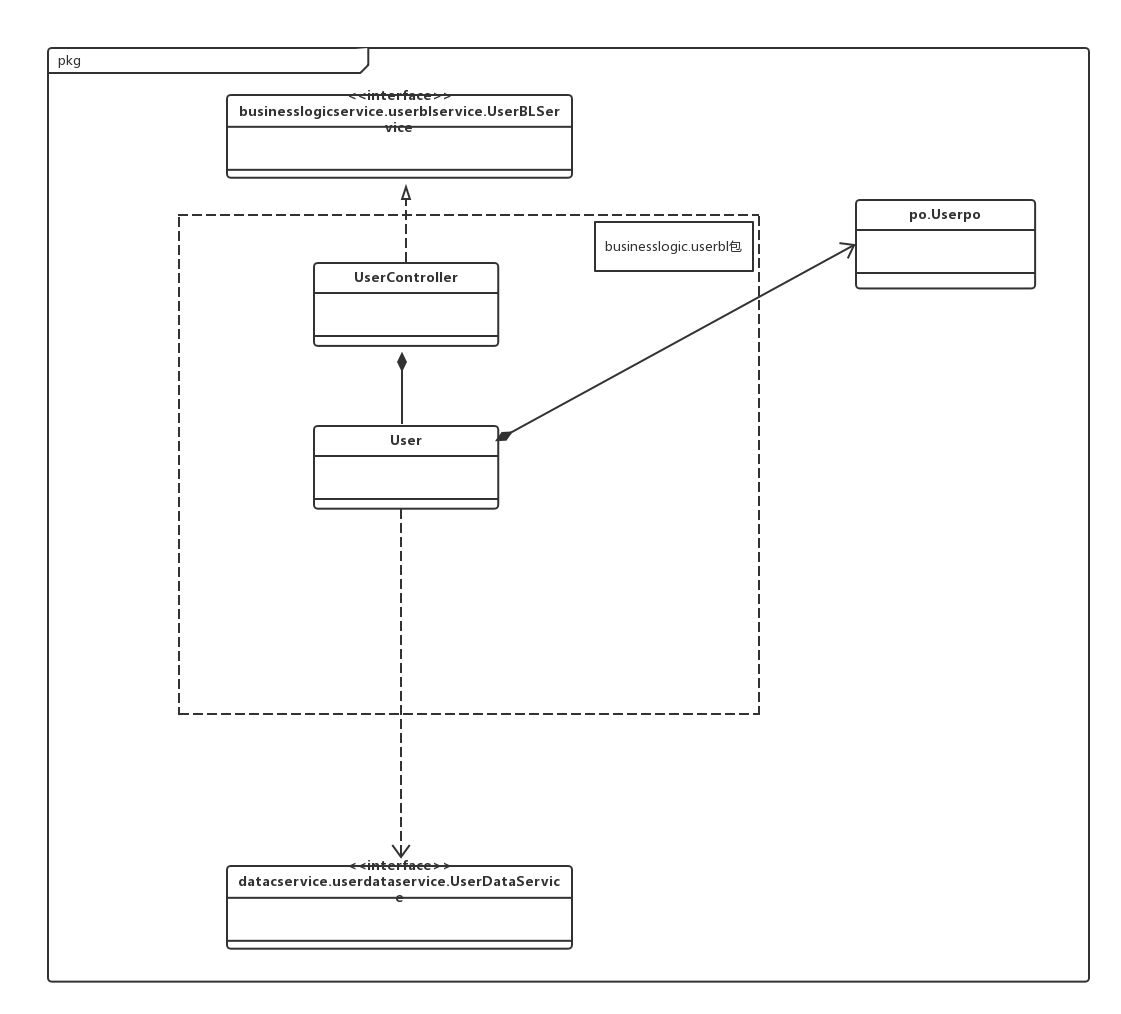
userbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Userbl模块的职责及接口参见软件系统结构描述文档

**(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。比如展示层和逻辑层之间，我们添加businessloginservice.userblservice UserBLService，业务逻辑层和数据层之间我们添加dataservice.userdataservice. UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将对销售的业务逻辑处理委托给User对象。UserPO是作为账户信息的持久化对象被添加到设计模型中去的。

userbl模块的设计如图所示。



user模块的各个类的职责如图所示

**（3）模块内部类的接口示范**

UserController和User的接口示范分别如表所示。

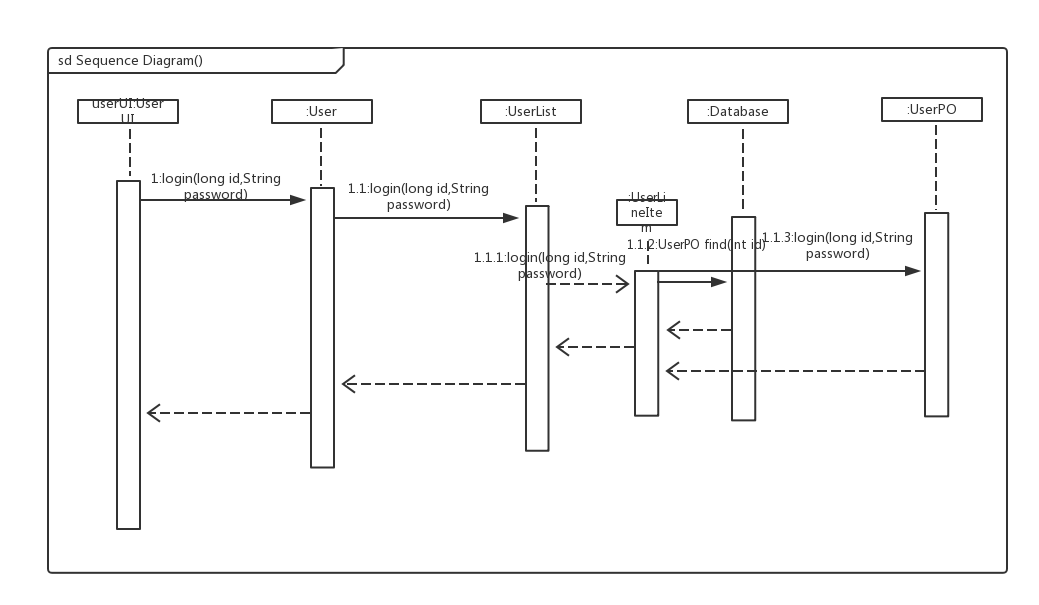
**UserController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public UserPO login(long id,String password) |
| 前置条件 | 已创建一个User领域对象，并且输入符合输入规则 |
| 后置条件 | 调用User领域对象的login方法 |
| 需要的服务（需接口） | | |
| 服务名 | | 服务 |
| User.login(long id,String password) | | 加入一个账户对象 |

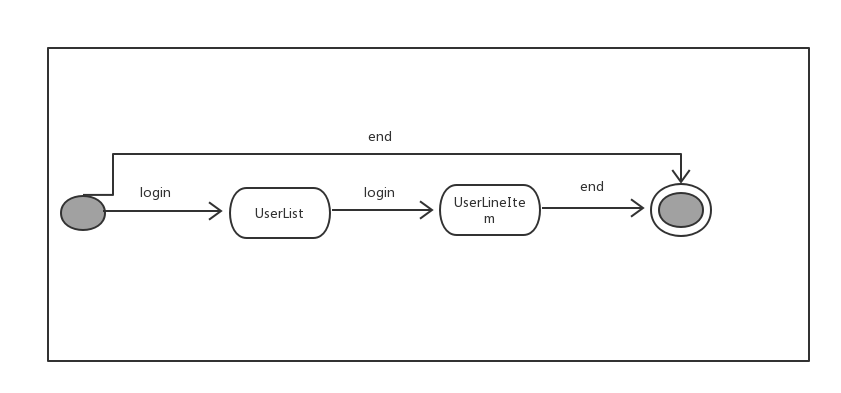
**User的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public UserPO login(long id,String password) |
| 前置条件 | 输入有效的账户，密码 |
| 后置条件 | 根据id查找用户是否存在，根据用户信息验证密码是否正确，如果通过验证，则记录当前用户 |
| 需要的服务（需接口） | | |
| 服务名 | | 服务 |
| UserDataService.find（int id） | | 根据ID查找用户信息 |
| UserDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| UserDataService.insert(UserPO po) | | 插入单一持久化对象 |
| UserDataService.delete(UserPO po) | | 删除单一持久化对象 |
| UserDataService.update(UserPO po) | | 更新单一持久化对象 |
| UserDataService.init | | 初始化持久化数据库 |
| UserDataService.finish | | 结束持久化数据库的使用 |

**（4）下图表明了酒店系统中，当用户输入账户密码之后，业务逻辑处理的相关对象之间的协作。**

****

User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。**

## 4.2 customerbl模块

**(1)模块概述**

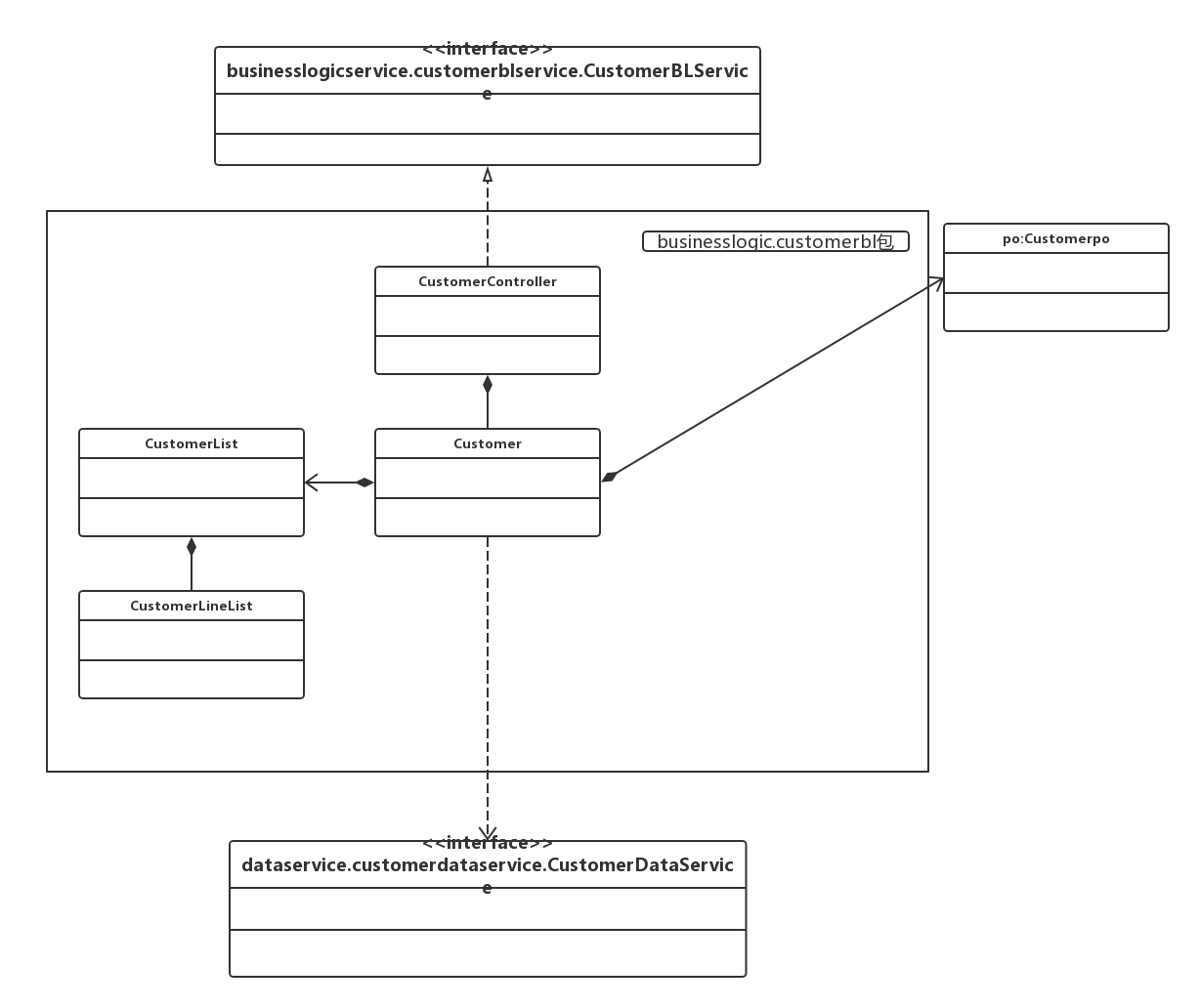
customerbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Customerbl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businesslogicservice.customerblservice.Customerblservice接口。业务逻辑层和数据层之间添加dataservice.customerdataservice.CustomerDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CustomerController,这样CustomerController会将对客户信息管理的业务逻辑处理委托给Customer对象。CustomerPO是作为客户信息的持久化对象被添加到设计模型中去的。CustomerLineItem包含有客户数据类型和数据信息，而CustomerList封装了关于CustomerLineItem的数据集合的数据结构。

customerbl模块的设计如图所示：



customerbl模块各个类的职责如表所示：

|  |  |
| --- | --- |
| 模块 | 职责 |
| loginController | 负责实现对应于登录界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的帐号和密码，可以解决登录问题 |
| Customer | 客户的领域模型对象，拥有姓名，联系电话，信用值，会员信息等信息，可以帮助完成个人信息维护界面所需要的服务 |
| CustomerController | 负责实现个人信息维护界面所需要的服务 |

**(3)模块内部类的接口规范**

CustomerController和Customer的接口规范如表所示：

CustomerController的接口规范

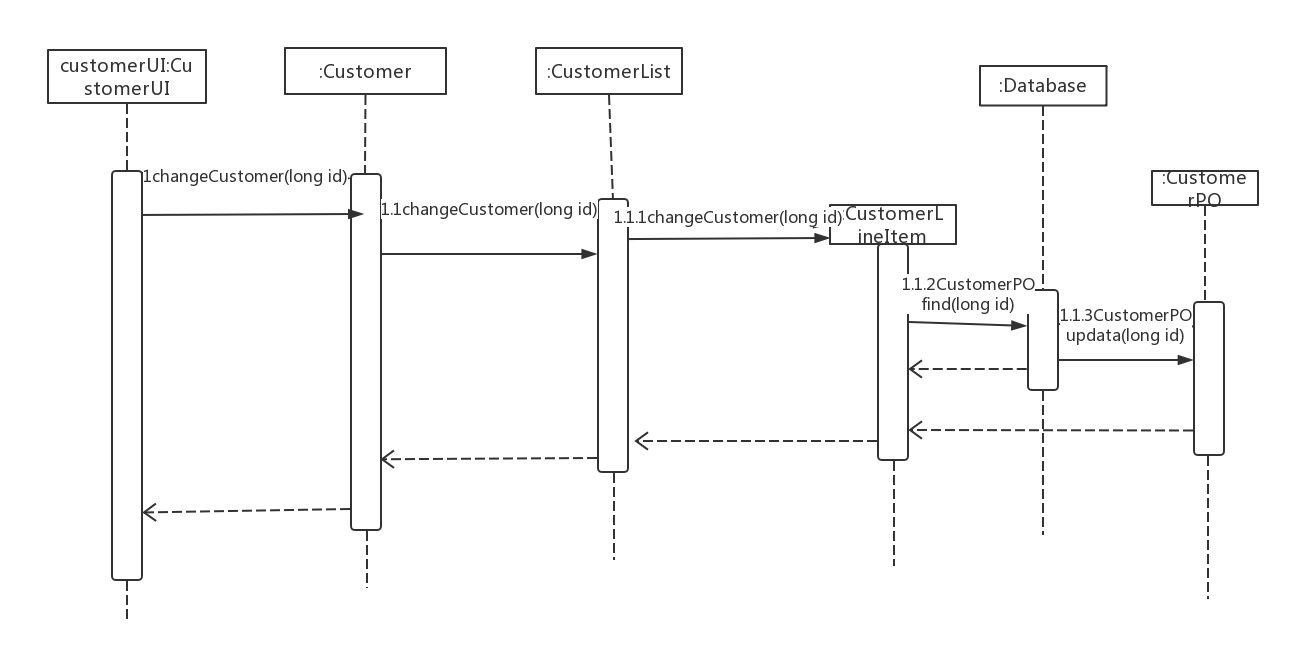
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CustomerController.getCustomer | 语法 | public CustomerPO getCustomer(long id) |
| 前置条件 | 已创建一个Customer领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Customer领域对象的getCustomer方法 |
| CustomerController.changeCustomer | 语法 | public void changeCustomer(long id) |
| 前置条件 | 已创建一个Customer领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Customer领域对象的changeCustomer方法 |
| 需要的服务（需接口） | | |
| 服务名 | | 服务 |
| Customer.getCustomer | | 获得客户的信息 |
| Customer.changeCustomer | | 改变客户信息 |

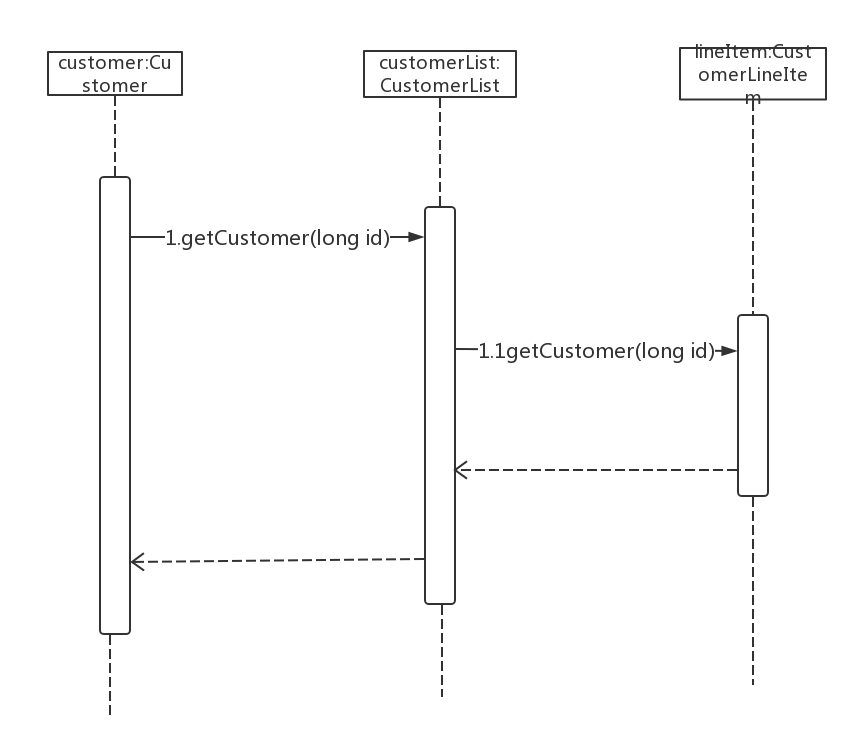
Customer的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Customer.getCustomer | 语法 | public CustomerPO getCustomer(long id) |
| 前置条件 | 当要获得用户信息时 |
| 后置条件 | 返回用户信息 |
| Customer.changeCustomer | 语法 | public void changeCustomer(long id) |
| 前置条件 | 客户请求改变信息 |
| 后置条件 | 更新客户信息 |
| 需要的服务（需接口） | | |
| CustomerDataService.find(long id) | | 根据id查找单一持久化对象 |
| CustomerDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| CustomerDataService.insert(CustomerPO po) | | 插入单一持久化对象 |
| CustomerDataService.delete(CustomerPo po) | | 删除单一持久化对象 |
| CustomerDataService.update(CustomerPO po) | | 更新单一持久化对象 |
| CustomerDataService.init | | 初始化持久化数据库 |
| CustomerDataService.finish | | 结束持久化数据库的使用 |

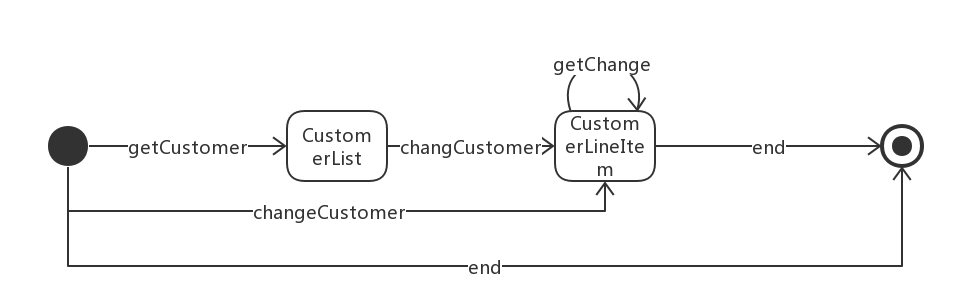
**(4)业务逻辑层的动态模型**

下图表示了修改用户信息的顺序图和得到用户信息的顺序图





Customer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。

## 4.3 hotelbl模块

**(1)模块描述**

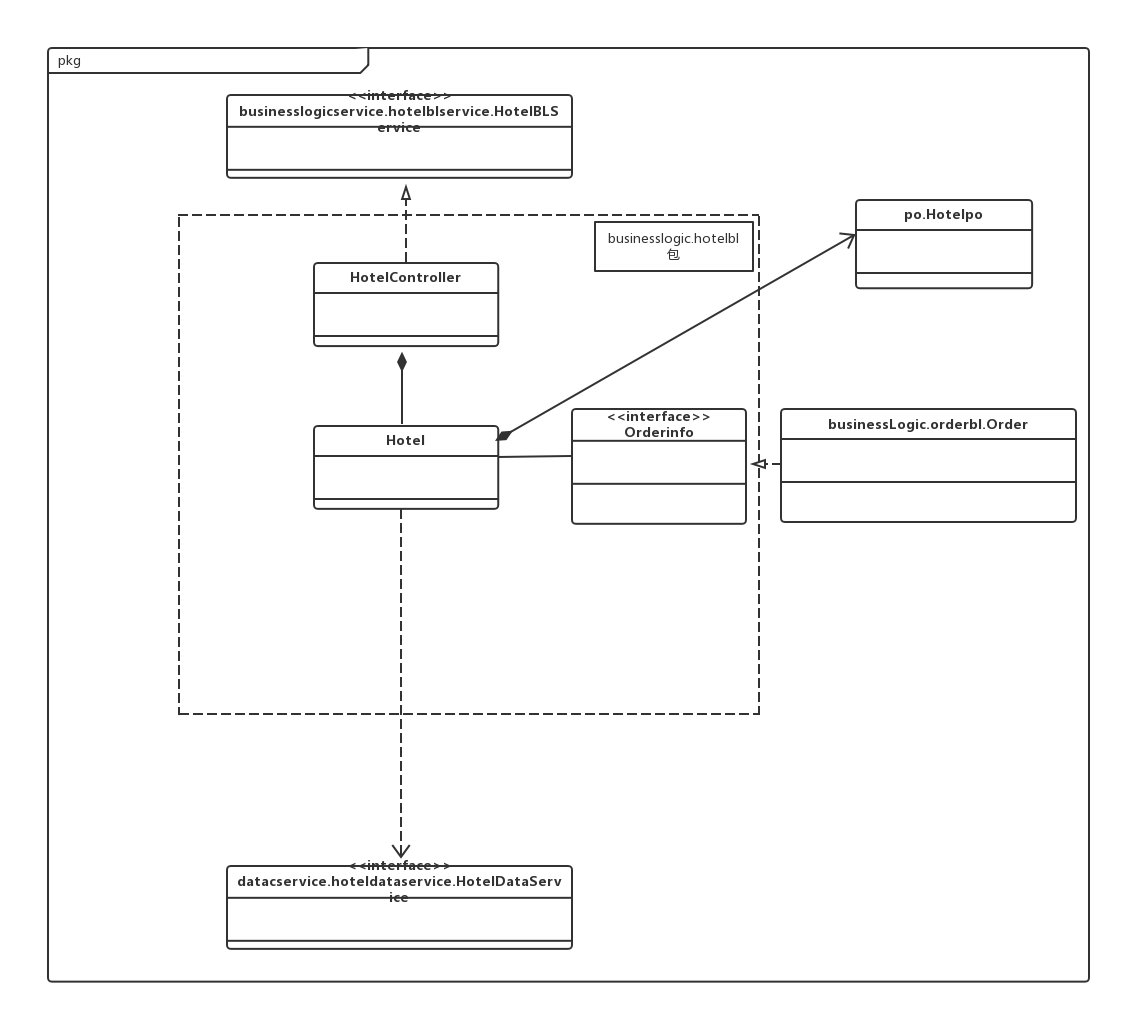
hotelbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Hotelbl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。比如展示层和逻辑层之间，我们添加businessloginservice. hotelblservice.HotelBLService，业务逻辑层和数据层之间我们添加dataservice. hoteldataservice. HotelDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了HotelController，这样HotelController会将对销售的业务逻辑处理委托给Hotel对象。HotelPO是作为账户信息的持久化对象被添加到设计模型中去的。OrderInfo根据依赖倒置原则

hotelbl模块的设计如图所示。



**（3）模块内部类的接口示范**

RoomController和Room的接口示范分别如表所示。

**RoomController的接口规范**

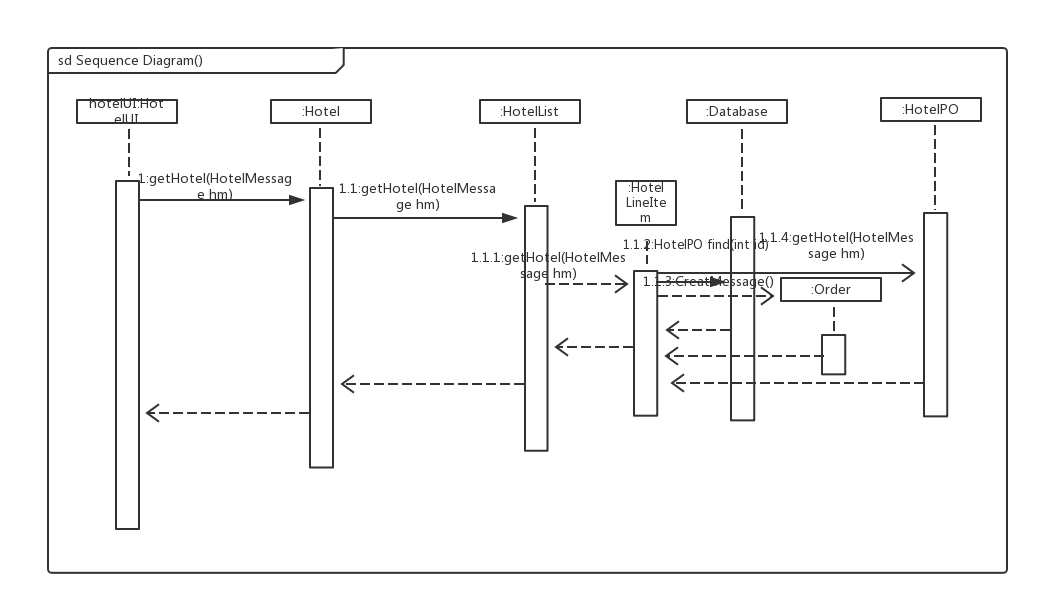
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| HotelController.getHotel | 语法 | public HotelPO getHotel(HotelMessage hm) |
| 前置条件 | 已创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel领域对象的getHotel方法 |
| HotelController.addEvaluation | 语法 | public void addEvaluation (long id，String str) |
| 前置条件 | 已创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel领域对象的addEvaluation方法 |
| HotelController.changeHotel | 语法 | Public void changeHotel(HotelPO rm) |
| 前置条件 | 已创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel领域对象的changeHotel方法 |
| 需要的服务（需接口） | | |
| Hotel.getHotel(HotelMessage hm) | | 加入一个酒店信息 |
| Hotel.addEvaluation(long id,String str) | | 加入一个评价信息 |
| Hotel.changeHotel(HotelPO rm) | | 加入一个酒店信息 |

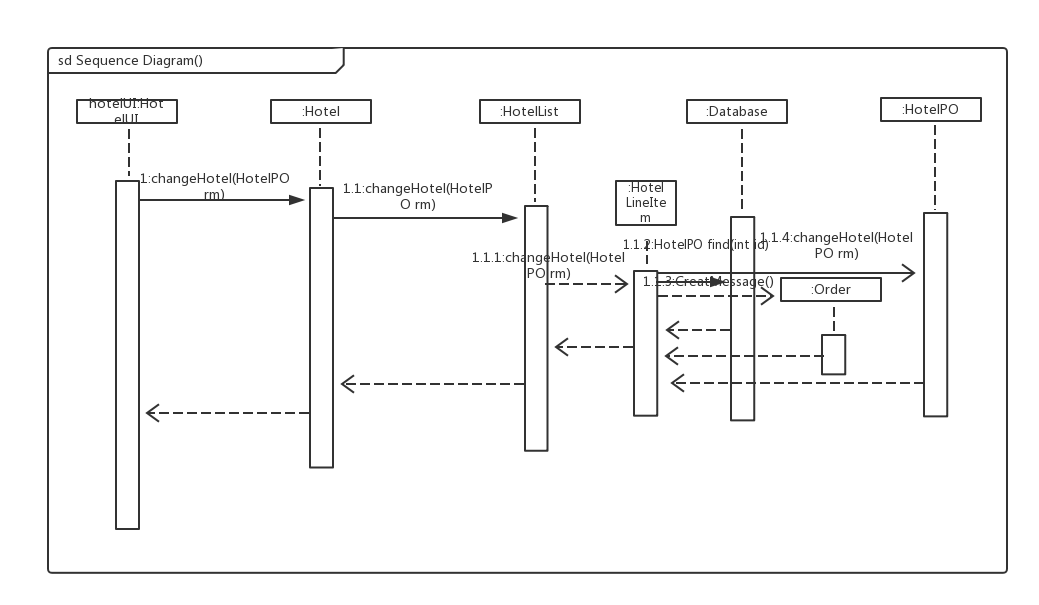
**Hotel的接口规范**

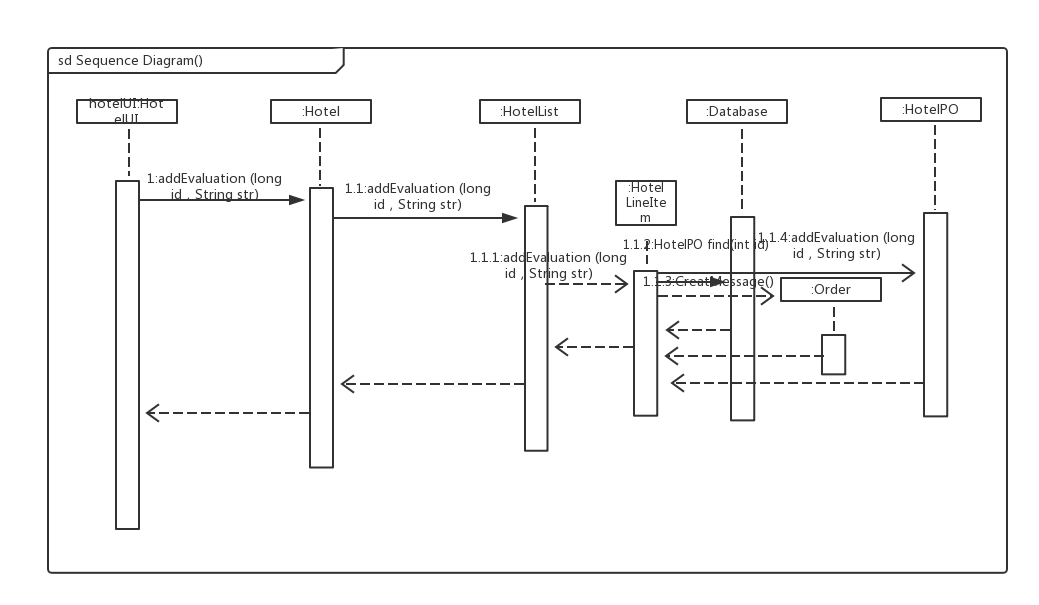
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Hotel.getHotel | 语法 | public HotelPO getHotel(HotelMessage hm) |
| 前置条件 | 要获得酒店信息 |
| 后置条件 | 返回酒店信息 |
| Hotel.addEvaluation | 语法 | public void addEvaluation (long id，String str) |
| 前置条件 | 有客户给该酒店评价 |
| 后置条件 | 把得到的评分和评价插入到数据库中 |
| Hotel.changeHotel | 语法 | Public void changeHotel(HotelPO rm) |
| 前置条件 | 酒店信息发生更改 |
| 后置条件 | 更新持久化数据 |
| 需要的服务（需接口） | | |
| HotelDataService.find(long id) | | 根据id查找单一持久化对象 |
| HotelDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| HotelDataService.insert(HotelPO po) | | 插入单一持久化对象 |
| HotelDataService.delete(HotelPo po) | | 删除单一持久化对象 |
| HotelDataService.update(HotelPO po) | | 更新单一持久化对象 |
| HotelDataService.init | | 初始化持久化数据库 |
| HotelDataService.finish | | 结束持久化数据库的使用 |

（4）业务逻辑层的动态模型

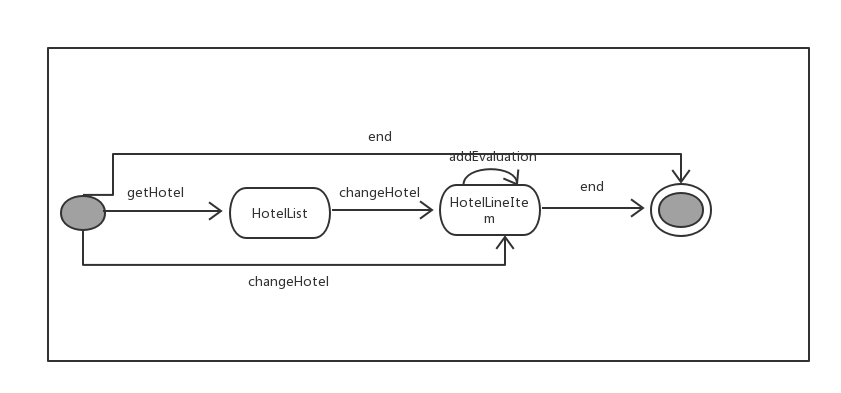
以下三图分别表示了当得到酒店信息，修改酒店信息，给予酒店评价的顺序图







Hotel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。

## 4.4 roombl模块

**(1)模块描述**

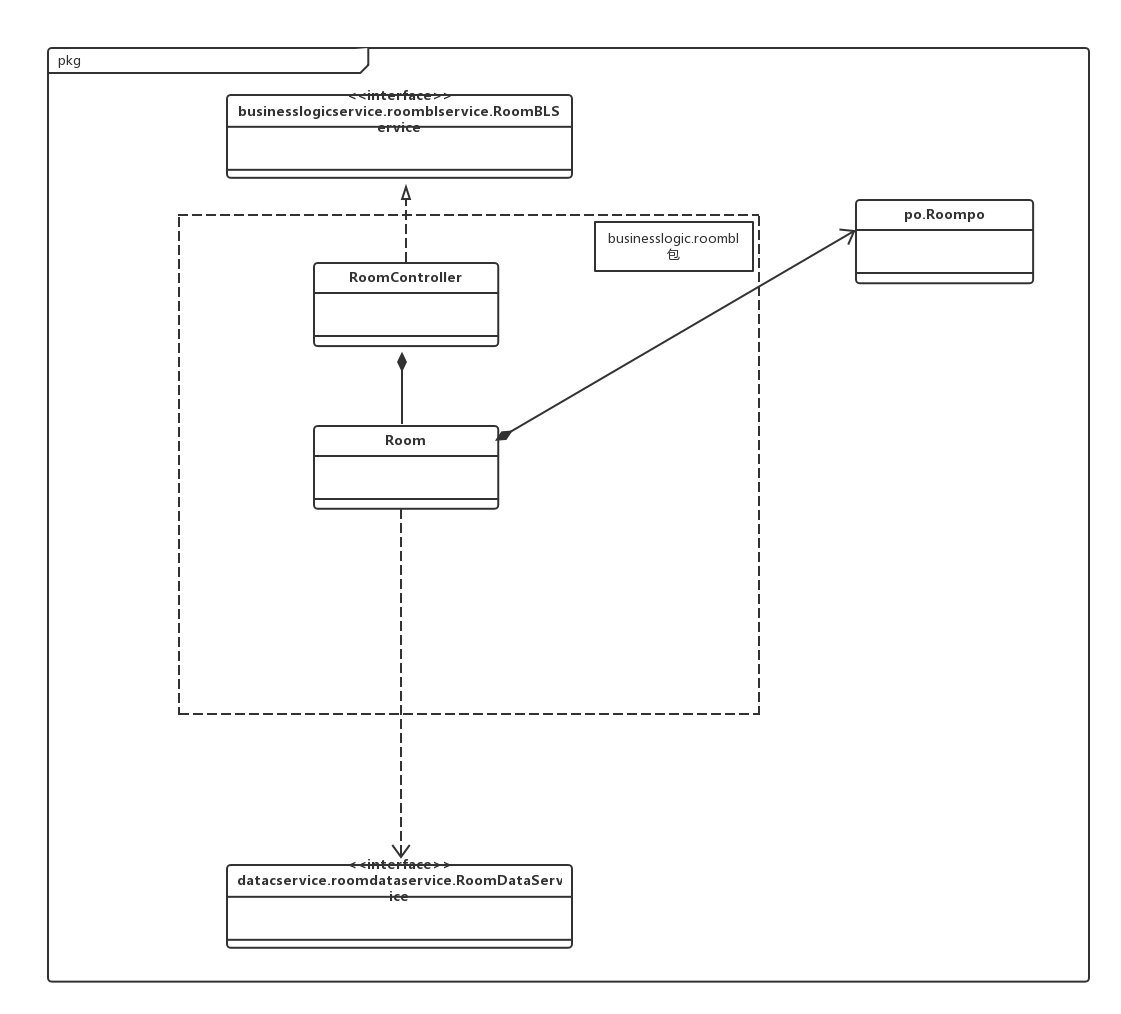
roombl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Hotelbl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。比如展示层和逻辑层之间，我们添加businessloginservice. roomblservice.RoomBLService，业务逻辑层和数据层之间我们添加dataservice. roomdataservice. RoomDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了RoomController，这样RoomController会将对销售的业务逻辑处理委托给Room对象。RoomPO是作为账户信息的持久化对象被添加到设计模型中去的。

roombl模块的设计如图所示。



**（3）模块内部类的接口示范**

RoomController和Room的接口示范分别如表所示。

**RoomController的接口规范**

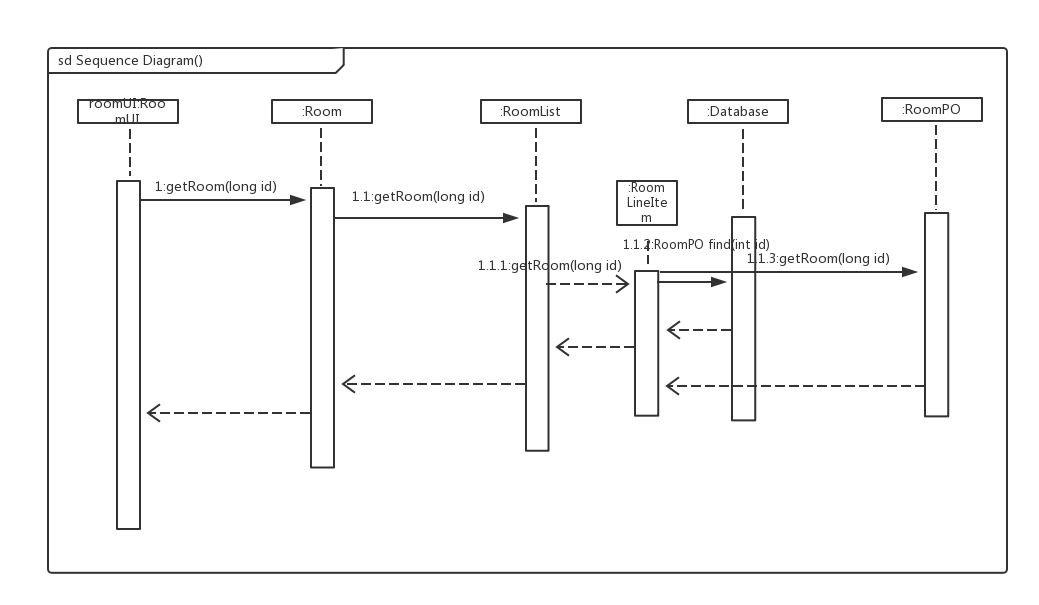
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| RoomController.getRoom | 语法 | public HotelPO getRoom(long id) |
| 前置条件 | 已创建一个Room领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Room领域对象的getRoom方法 |
| RoomController.changeRoom | 语法 | public void change(RoomPO rm) |
| 前置条件 | 已创建一个Room领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Room领域对象的changeRoom方法 |
| RoomController.addRoom | 语法 | Public void addRoom(RoomPO rm) |
| 前置条件 | 已创建一个Room领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Room领域对象的addRoom方法 |
| 需要的服务（需接口） | | |
| Room.getRoom(long id) | | 加入一个房间信息 |
| Room.changeRoom(RoomPO rm) | | 加入一个房间信息 |
| Room.addRoom(RoomPO rm) | | 加入一个房间信息 |

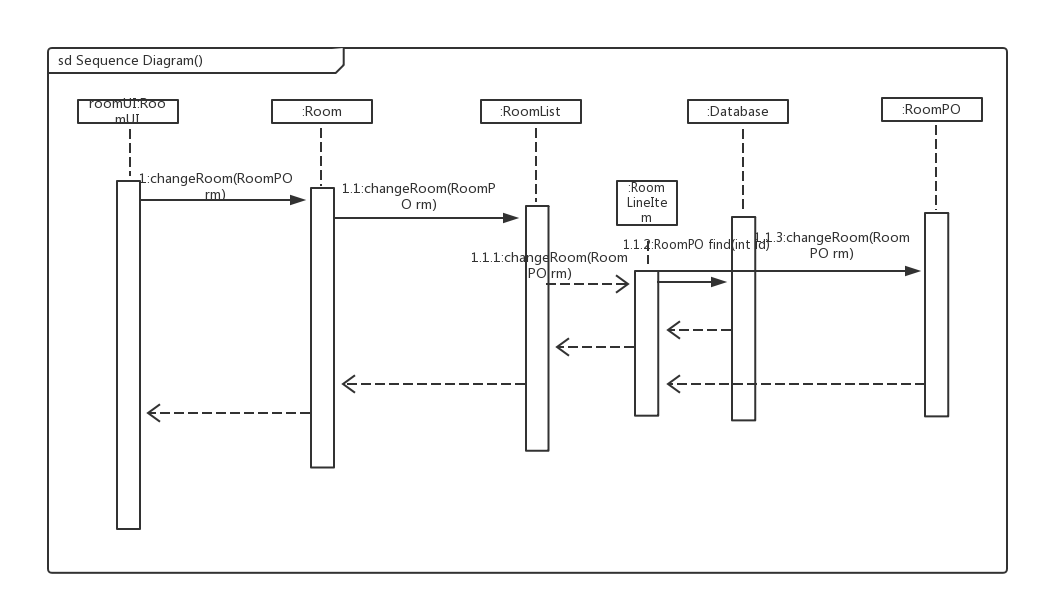
**Room的接口规范**

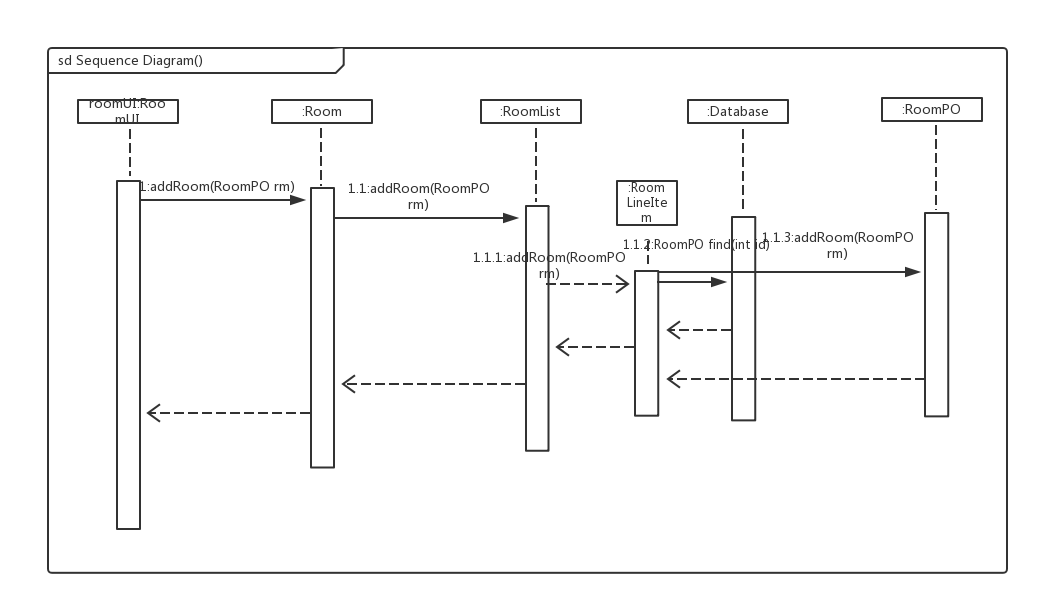
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Room.getRoom | 语法 | public HotelPO getRoom(long id) |
| 前置条件 | 要显示酒店的房间信息 |
| 后置条件 | 返回房间信息 |
| Room.changeRoom | 语法 | public void changeRoom(RoomPO rm) |
| 前置条件 | 房间状态改变 |
| 后置条件 | 改变持久化数据中的房间状态 |
| Room.addRoom | 语法 | Public void addRoom(RoomPO rm) |
| 前置条件 | 酒店申请添加房间 |
| 后置条件 | 在持久化数据库中添加一个房间的信息 |
| 需要的服务（需接口） | | |
| RoomDataService.find(long id) | | 根据id查找单一持久化对象 |
| RoomDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| RoomDataService.insert(RoomPO po) | | 插入单一持久化对象 |
| RoomDataService.delete(RoomPo po) | | 删除单一持久化对象 |
| RoomDataService.update(RoomPO po) | | 更新单一持久化对象 |
| RoomDataService.init | | 初始化持久化数据库 |
| RoomDataService.finish | | 结束持久化数据库的使用 |

（4）业务逻辑层的动态模型

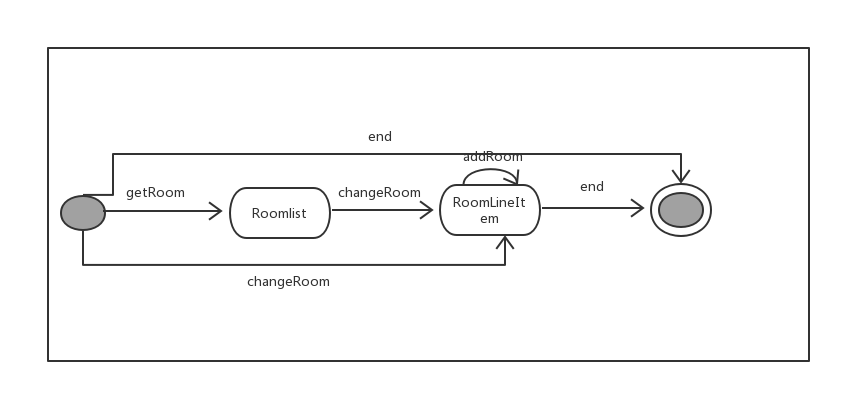
以下三图分别表示了当得到房间信息，修改房间信息，房间信息发生永久改变的顺序图







Room对象的状态图



（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。

## 4.5 orderbl模块

**(1)模块概述**

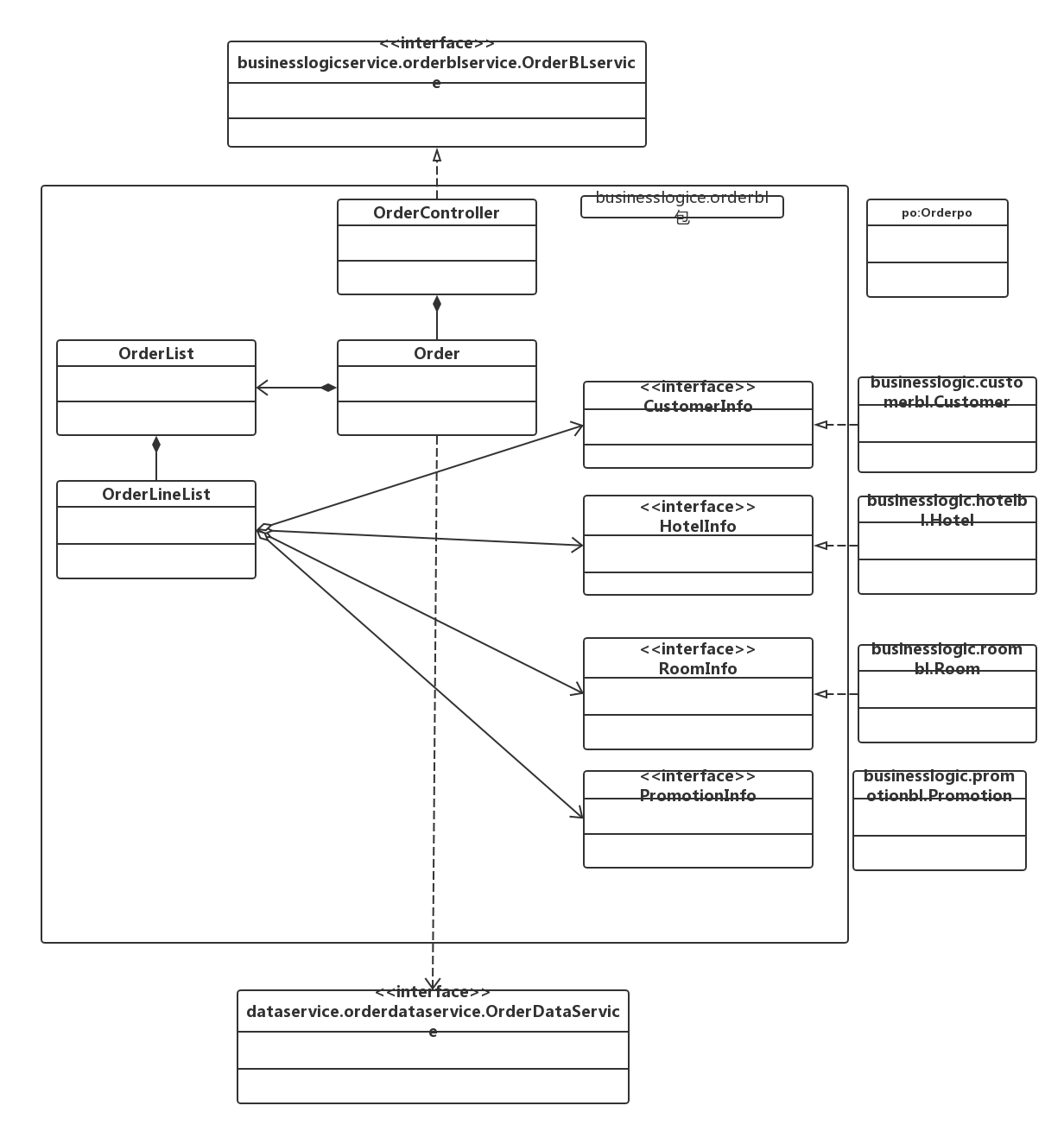
orderbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

Orderbl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businesslogicservice.orderblservice.Orderblservice接口。业务逻辑层和数据层之间添加dataservice.orderdataservice.OrderDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OrderController,这样OrderController会将对订单操作的业务逻辑处理委托给Order对象。OrderPO是作为订单信息的持久化对象被添加到设计模型中去的。OrderList和OrderLineItem的添加是CustomerInfo、HotelInfo、RoomInfo的容器类。OrderLineItem包含有订单的时间数据和客户、酒店数据。而OrderList封装了关于OrderLineItem的数据集合的数据结构。CustomerInfo、HotelInfo、RoomInfo都是依据依赖倒置原则，为了消除循环而产生的接口。

orderbl模块的设计如图所示：



orderbl模块各个类的职责如表所示：

|  |  |
| --- | --- |
| 模块 | 职责 |
| loginController | 负责实现对应于登录界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户数据的帐号和密码，可以解决登录问题 |
| Order | 客户的领域模型对象，拥有姓名，联系电话，信用值，会员信息等信息，可以帮助完成个人信息维护界面所需要的服务 |
| OrderController | 负责实现个人信息维护界面所需要的服务 |

**(3)模块内部类的接口规范**

OrderController和Order的接口规范如表所示：

OrderController的接口规范

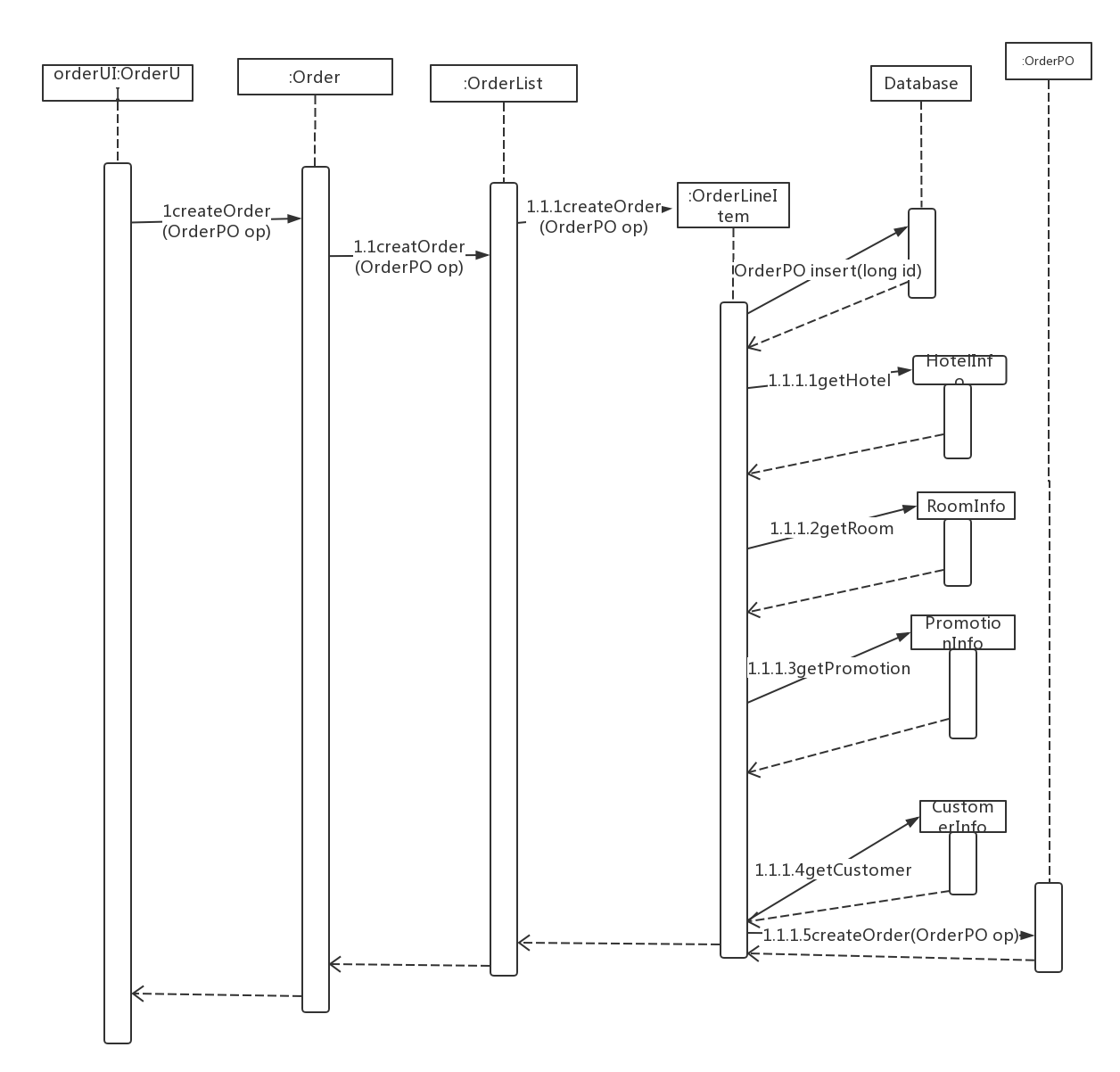
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderController.createOrder | 语法 | public void createOrder(OrderPO om) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的createOrder方法 |
| OrderController.getOrder | 语法 | public OrderPO getOrder(long id) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的getOrder方法 |
| OrderController.cancelOrder | 语法 | public void cancelOrder(long id) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的cancelOrder方法 |
| OrderController.completeOrder | 语法 | public void completeOrder(long id) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的completeOrder方法 |
| OrderController.recoverOrder | 语法 | Public void recoverOrder(long id) |
| 前置条件 | 已经创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的recoverOrder方法 |
| 需要的服务（需接口） | | |
| Order.createOrder | | 创建一个订单 |
| Order.getOrder | | 获得一个订单的信息 |
| Order.cancelOrder | | 将一个订单设为已撤销状态 |
| Order.completeOrder | | 将一个订单设为正常状态 |
| Order.recoverOrder | | 将一个已撤销订单设为正常状态 |

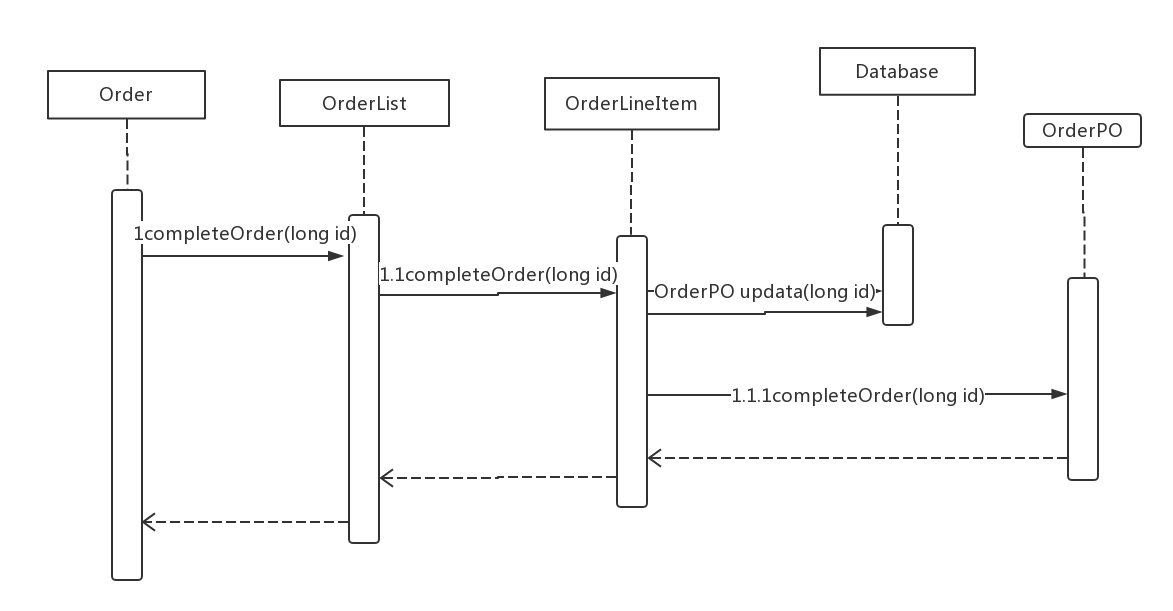
Order的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Orders.createOrder | 语法 | public void createOrder(OrderPO om) |
| 前置条件 | 启动一个预订服务 |
| 后置条件 | 在持久化数据库中添加一个订单 |
| Orders.getOrder | 语法 | public OrderPO getOrder(long id) |
| 前置条件 | 要获得一个订单的信息 |
| 后置条件 | 返回一个订单的信息 |
| Orders.cancelOrder | 语法 | public void cancelOrder(long id) |
| 前置条件 | 客户申请撤销订单 |
| 后置条件 | 系统将订单设置为已撤销状态 |
| Order.completeOrders | 语法 | public void completeOrder(long id) |
| 前置条件 | 客户按时办理入住 |
| 后置条件 | 系统将订单设置为已执行状态 |
| Order.recoverOrder | 语法 | Public void recoverOrder(long id) |
| 前置条件 | 客户延期入住，经过酒店或网站的审核 |
| 后置条件 | 系统将订单设为已执行状态 |
| 需要的服务（需接口） | | |
| OrdersDataService.find(long id) | | 根据id查找单一持久化对象 |
| OrdersDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| OrdersDataService.insert(SalesPO po) | | 插入单一持久化对象 |
| OrdersDataService.delete(SalesPo po) | | 删除单一持久化对象 |
| OrdersDataService.update(SalesPO po) | | 更新单一持久化对象 |
| OrdersDataService.init | | 初始化持久化数据库 |
| OrdersDataService.finish | | 结束持久化数据库的使用 |

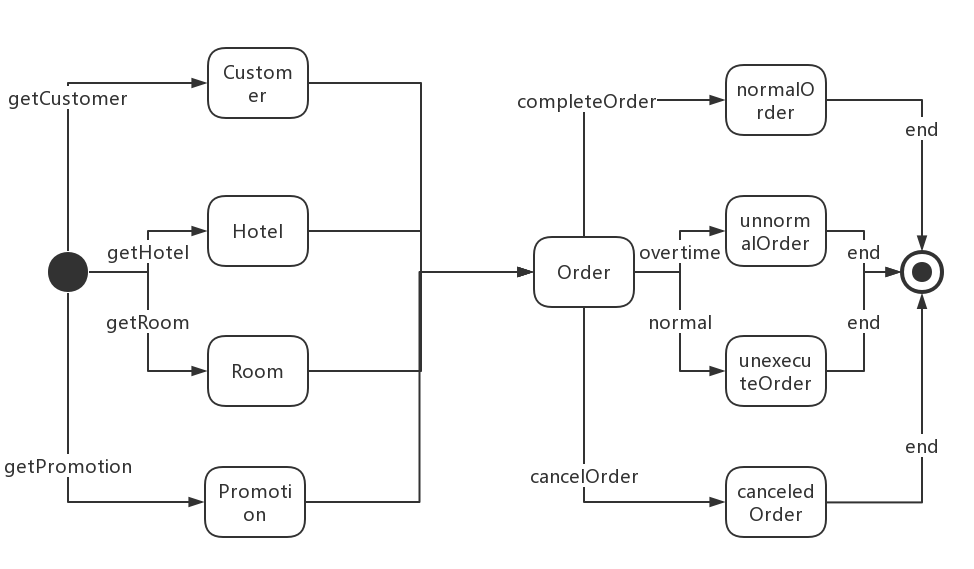
**(4)业务逻辑层的动态模型**

下图分别表示了生成订单、完成订单的顺序图





Order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。

## 4.6 promotionbl模块

**⑴模块概述**

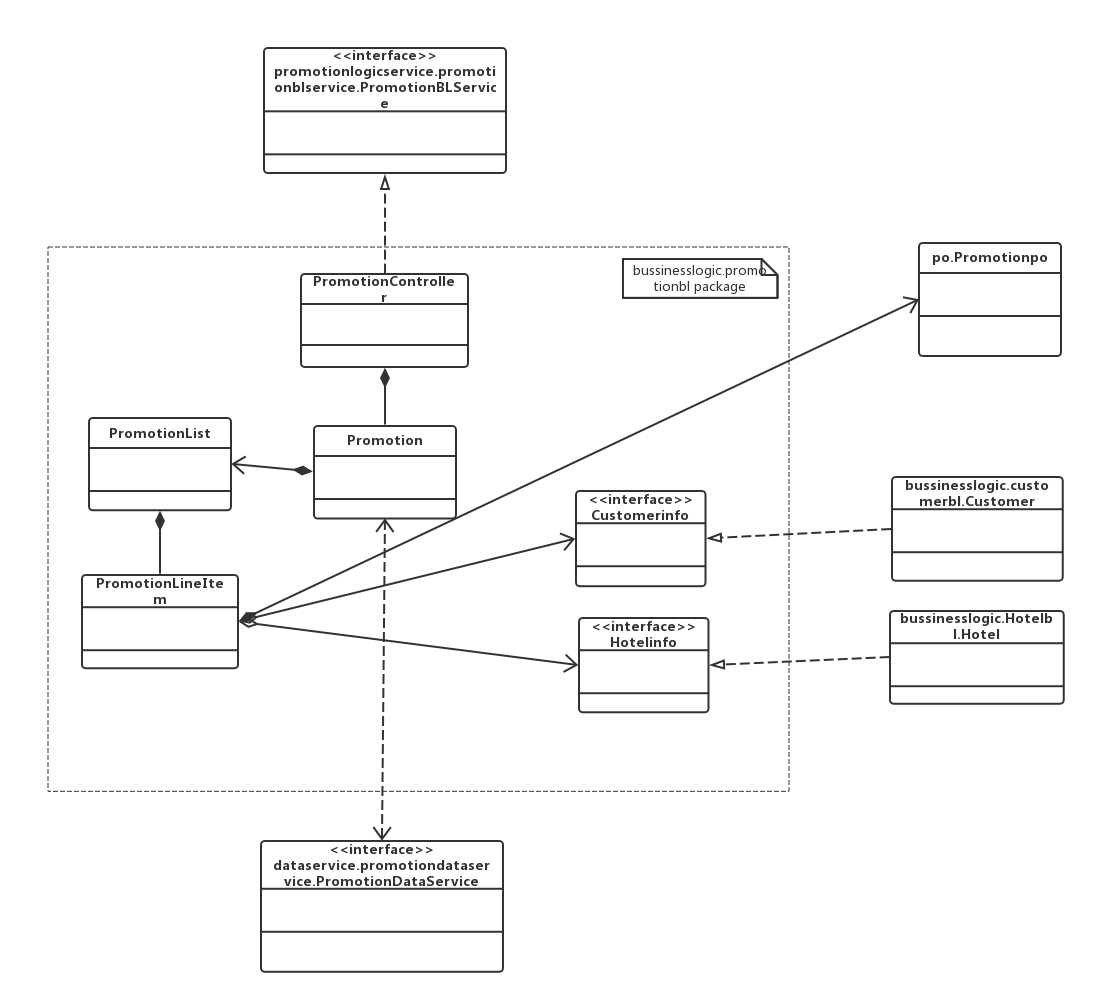
Promotionbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Promotionbl模块的职责及接口参见软件系统结构文档表10。

**⑵整体结构**

根据体系结构的设计，我们将系统分为展示层、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。比如展示层和逻辑层之间，我们添加businesslogicservice.saleblservice.SalesBLService接口。业务逻辑层和数据层之间添加dataservice.promotiondataservice.PromotionDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PromotionControlller，这样它会将对促销策略制定的业务逻辑委托给Promotion对象。PromotionPO是作为策略信息的持久化对象被添加到设计模型中的。而PromotionList和PromotionLineItem则是作为策略信息的容器类。PromotionLineItem存储了策略信息的数据，而PromotionList则封装了PromotionLineItem数据的集合。CustomerInfo和HotelInfo都是根据依赖倒置原则，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 |
| PromotionControlller | 负责实现策略制定界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户的ID和密码，可以解决登录问题 |
| Promotion | 策略的领域模型对象，拥有策略所需的客户，酒店等信息，可以帮助完成策略设定界面所需要的服务 |

**⑶模块内部类的接口规范**

Promotion和PromotionController.的接口规范分别如下标所示。

表 Promotion的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Promotion.usePromotion | 语法 | public promotionPO usePromotion(long id) |
| 前置条件 | 下一个订单 |
| 后置条件 | 下一个订单后，使用促销 |
| Promotion.addPromotion | 语法 | public promotionPO changePromotion(long id) |
| 前置条件 | 有新的促销策略 |
| 后置条件 | 有新的促销策略后，更新促销的服务信息 |
| Promotion.getPromotion | 语法 | Public String getPromotion() |
| 前置条件 | 需要展示策略的文本信息时 |
| 后置条件 | 返回促销策略的文本信息 |
| 需要的服务（需接口） | | |
| PromotionDataService.find(long id) | | 根据id查找单一持久化对象 |
| PromotionDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| PromotionDataService.insert(WebworkerPO po) | | 插入单一持久化对象 |
| PromotionDataService.delete(WebworkerPo po) | | 删除单一持久化对象 |
| PromotionDataService.update(WebworkerPO po) | | 更新单一持久化对象 |
| PromotionDataService.init | | 初始化持久化数据库 |
| PromotionDataService.finish | | 结束持久化数据库的使用 |

表 PromotionController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionController.usePromotion | 语法 | public promotionPO usePromotion(long id) |
| 前置条件 | 下一个订单 |
| 后置条件 | 下一个订单后，使用促销 |
| PromotionController..changePromotion | 语法 | public promotionPO changePromotion(long id) |
| 前置条件 | 有新的促销策略 |
| 后置条件 | 有新的促销策略后，更新促销的服务信息 |
| PromotionController..getPromotion | 语法 | Public String getPromotion() |
| 前置条件 | 需要展示策略的文本信息时 |
| 后置条件 | 返回促销策略的文本信息 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Promotion.usePromotio(long id) | 使用促销策略 | |
| Promotion.addPromotion(PromotionPO po) | 添加促销策略 | |
| Promotion.getPromotion(long id) | 获取促销策略信息 | |

**(4)业务逻辑层的动态模型**

如下图所示表明了酒店预订系统中，当用户输入新的策略信息时，promotion业务逻辑处理的相关对象之间的协作。

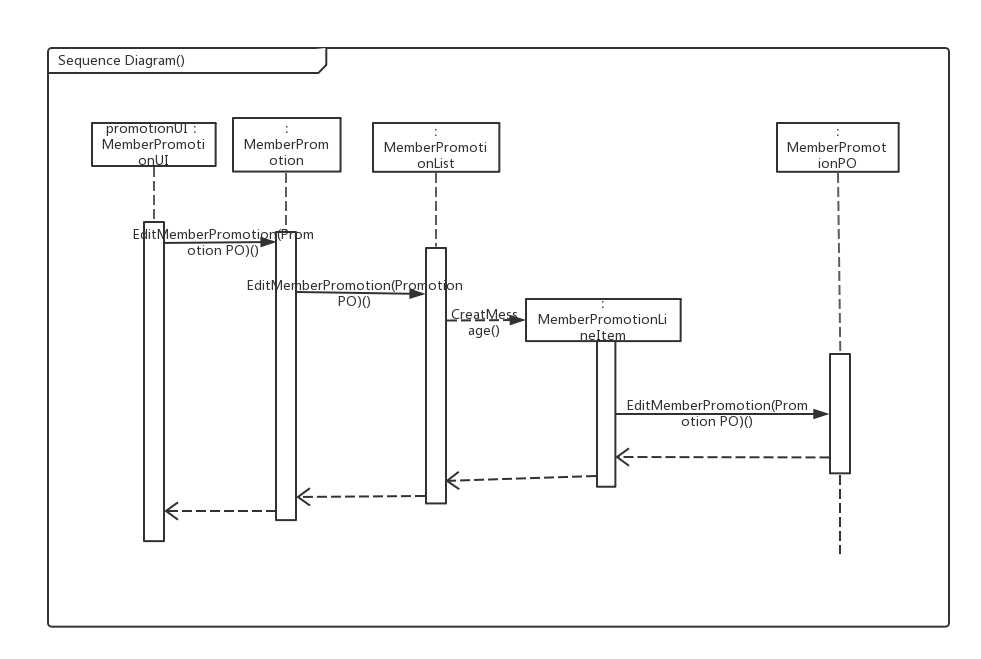


图1编辑会员制度的顺序图

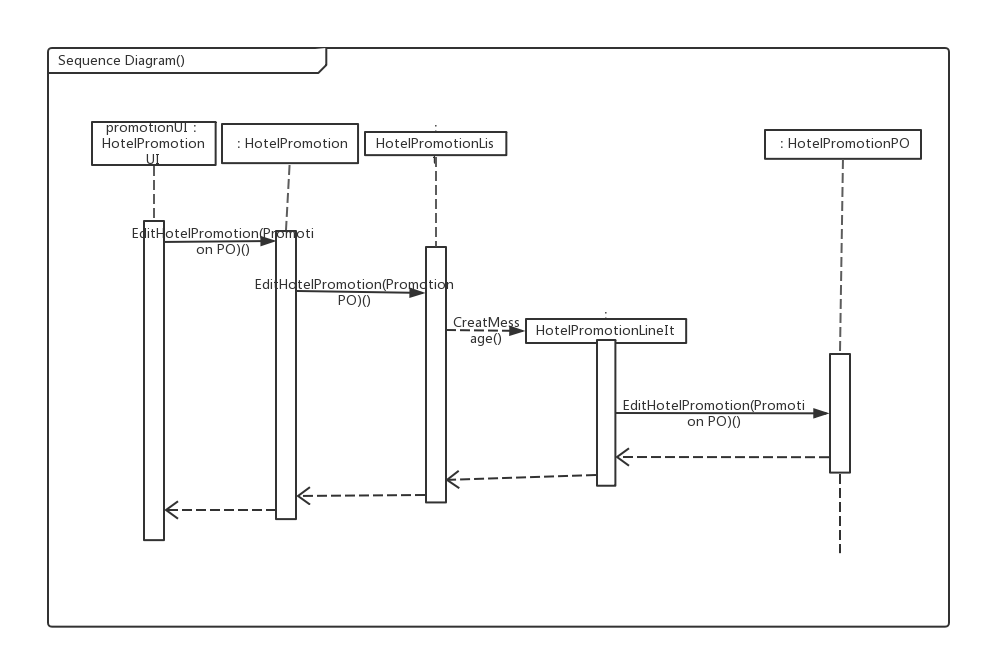


图2 编辑酒店促销策略的顺序图

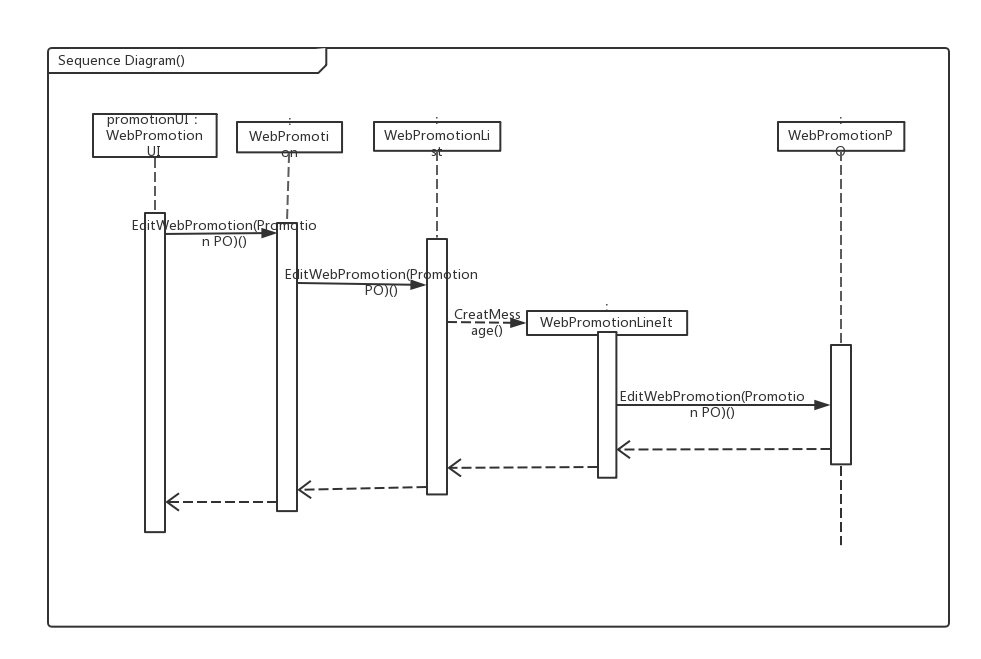
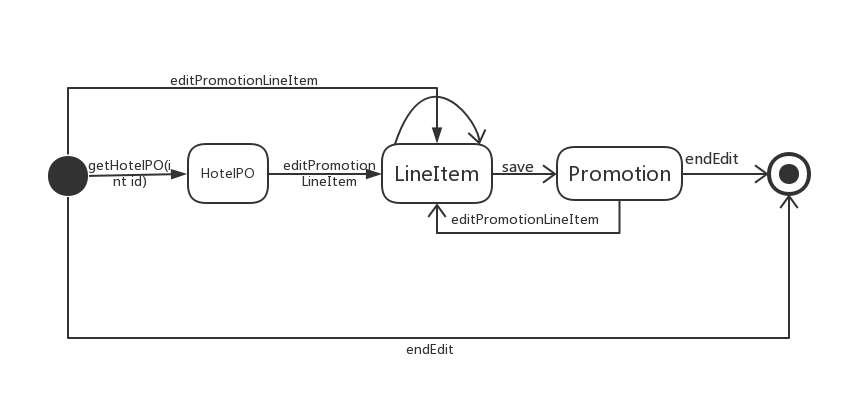


图3 编辑网站促销策略的顺序图

图3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，以及因转移而伴随的动作。随着getHotel()方法被调用，Promotion进入HotelPO状态，之后通过编辑项目进入LineItem状态。



**(5)业务逻辑层的设计原理**

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

## 4.7 webworkerbl模块

**⑴模块概述**

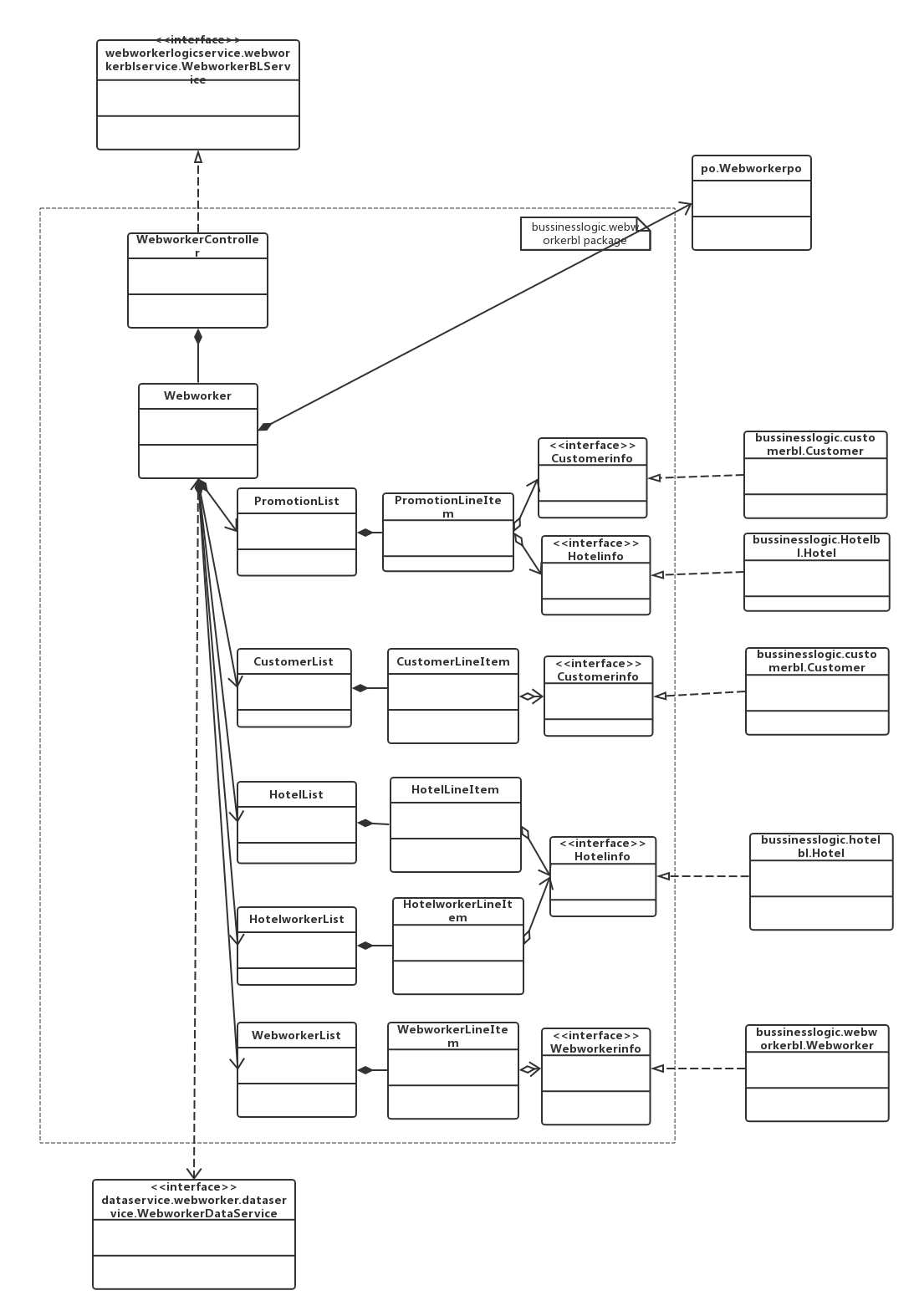
webworkerbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

webworkerbl模块的职责及接口参见软件系统结构文档 表10。

**⑵整体结构**

根据体系结构的设计，我们将系统分为展示层、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。比如展示层和逻辑层之间，我们添加businesslogicservice.webworkerblservice.WebworkerBLService接口。业务逻辑层和数据层之间添加dataservice.webworkerdataservice.WebworkerblDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加WebworkerblControlller，这样它会将对webworker的业务逻辑委托给webworkerbl对象。WebworkerPO是作为策略信息的持久化对象被添加到设计模型中的。

Webworkerblbl模块的设计如下图所示。



webworkerbl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 |
| WebworkerControlller | 负责实现策略制定界面所需要的服务 |
| User | 系统用户的领域模型对象，拥有用户的ID和密码，可以解决登录问题 |
| Webworker | 网站端管理及营销人员的领域模型对象，拥有执行对客户（包括注册会员），酒店酒店管理人员及网站营销人员管理的权限。 |

**⑶模块内部类的接口规范**

WebworkerController和Webworker的接口规范如下表所示。

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| WebworkerController.getAccount | 语法 | public String getAccount(hotelPO hm) |
| 前置条件 | 有人注册会员 |
| 后置条件 | 有人注册会员后，记录其信息 |
| 需要的服务（需接口） | | |
| Webworker.getAccount | | 根据id查找单一持久化对象 |

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Webworker.getAccount | 语法 | public String getAccount(hotelPO hm) |
| 前置条件 | 有人注册会员 |
| 后置条件 | 有人注册会员后，记录其信息 |
| 需要的服务（需接口） | | |
| WebworkerDataService.find(long id) | | 根据id查找单一持久化对象 |
| WebworkerDataService.finds(String field,int value) | | 根据两值查找多个持久化对象 |
| WebworkerDataService.insert(WebworkerPO po) | | 插入单一持久化对象 |
| WebworkerDataService.delete(WebworkerPo po) | | 删除单一持久化对象 |
| WebworkerDataService.update(WebworkerPO po) | | 更新单一持久化对象 |
| WebworkerDataService.init | | 初始化持久化数据库 |
| WebworkerDataService.finish | | 结束持久化数据库的使用 |

**(4)业务逻辑层的动态模型**

如下图所示表明了酒店预订系统中，当网站工作人员编辑用户信息时，webworker业务逻辑处理的相关对象之间的协作。

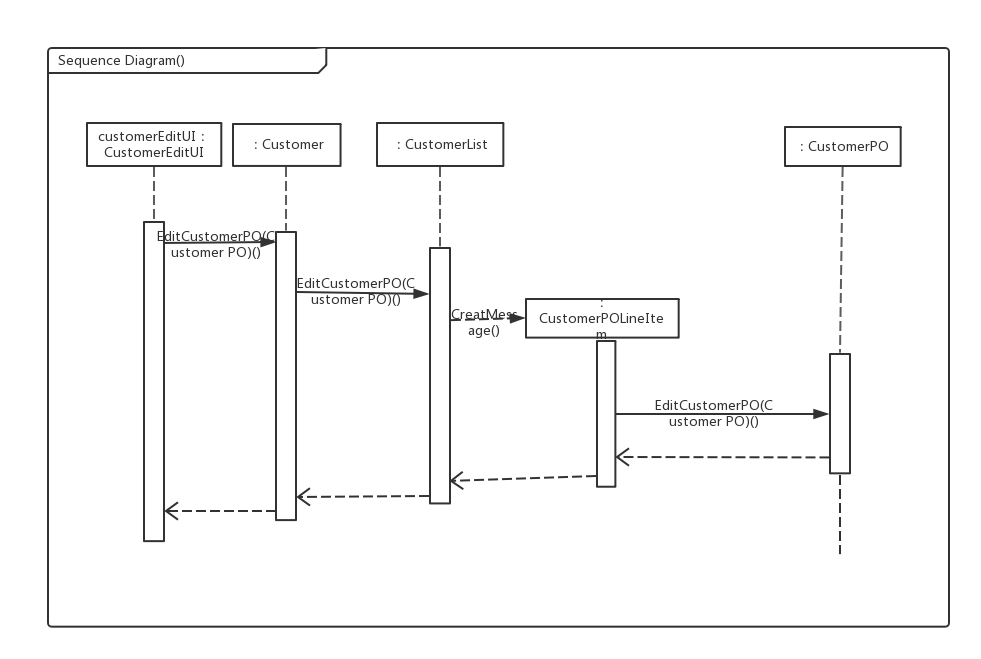


图1 编辑客户信息的顺序图

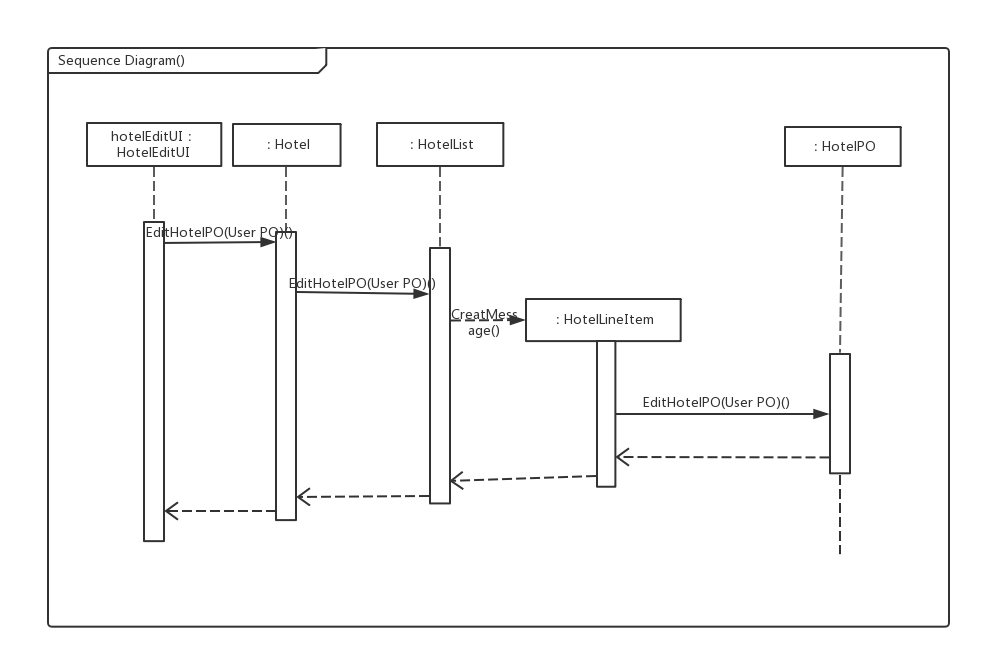


图2 编辑酒店信息的顺序图

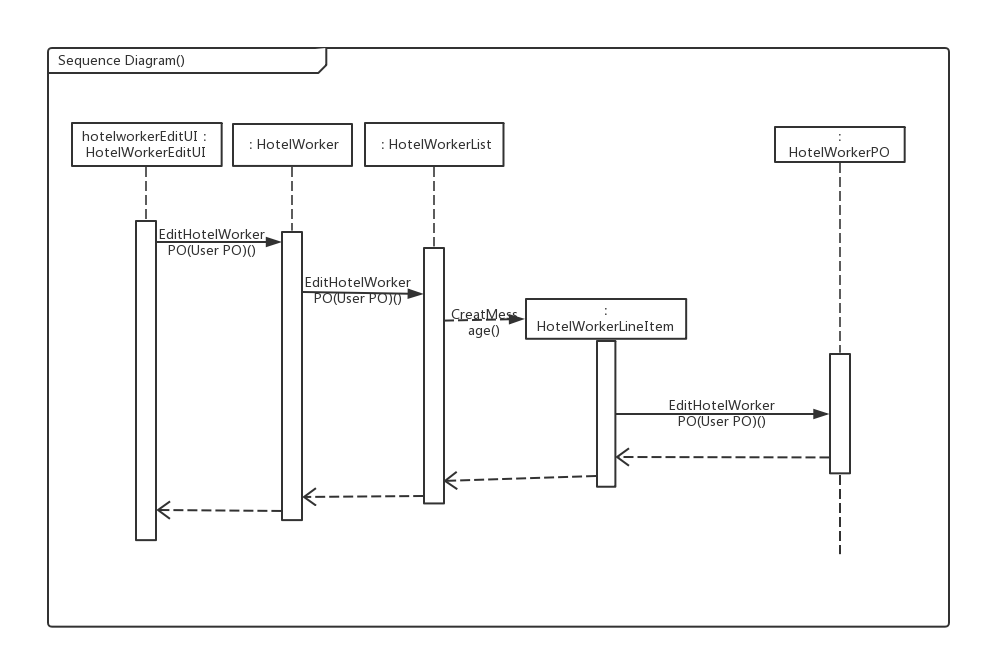


图3 编辑酒店管理人员信息的顺序图

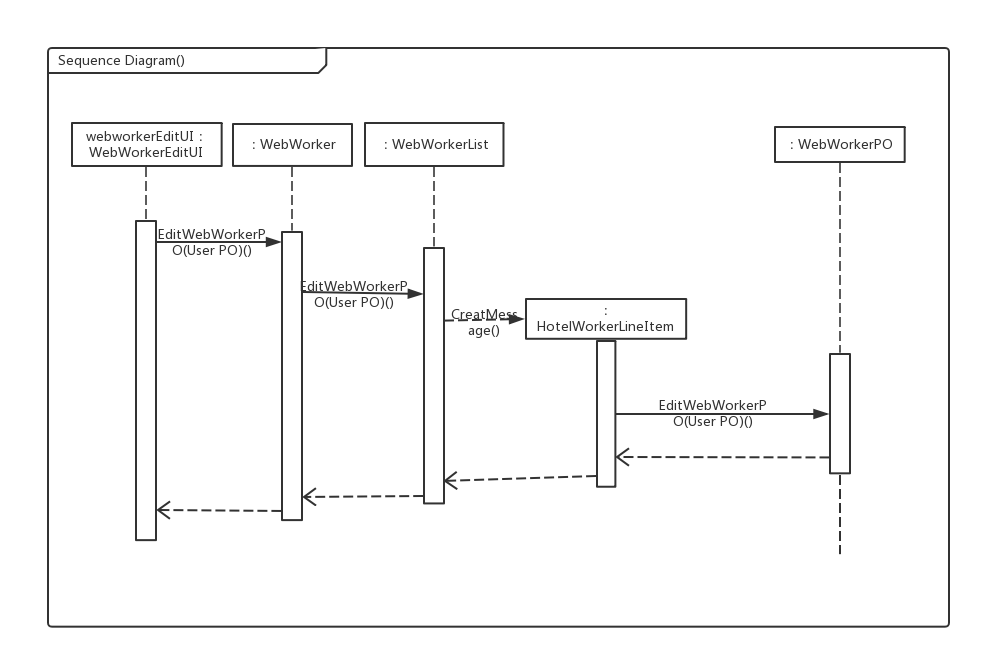
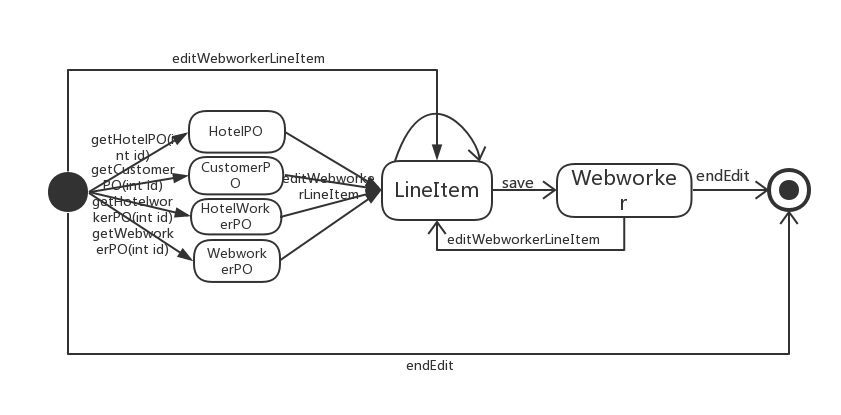


图4 编辑网站营销人员信息的顺序图

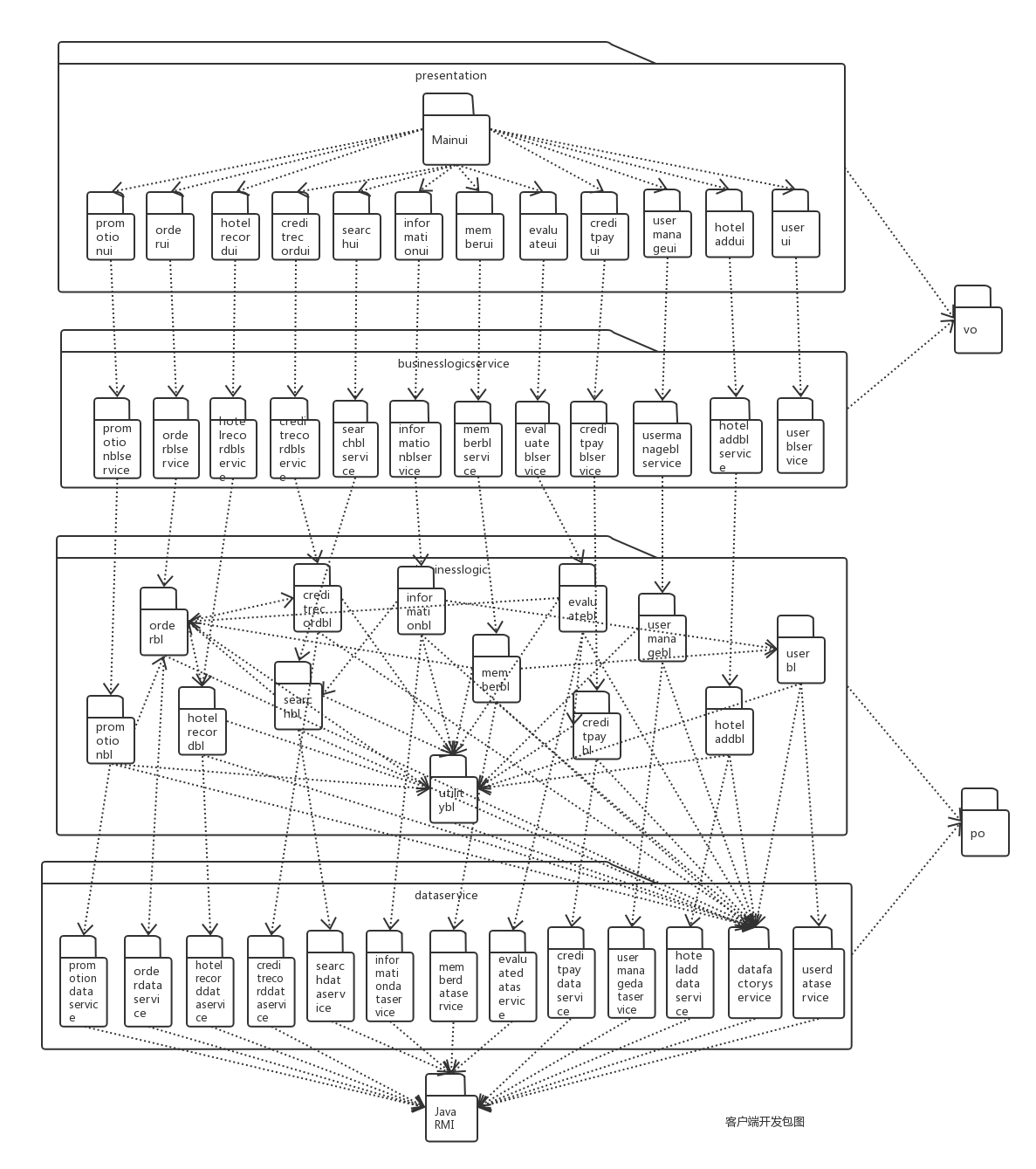
图4所示的状态图描述了Webworker对象的生存期间的状态序列、引起转移的事件，以及因转移而伴随的动作。随着getHotel()等方法被调用，Promotion进入HotelPO等状态，之后通过编辑项目进入LineItem状态。



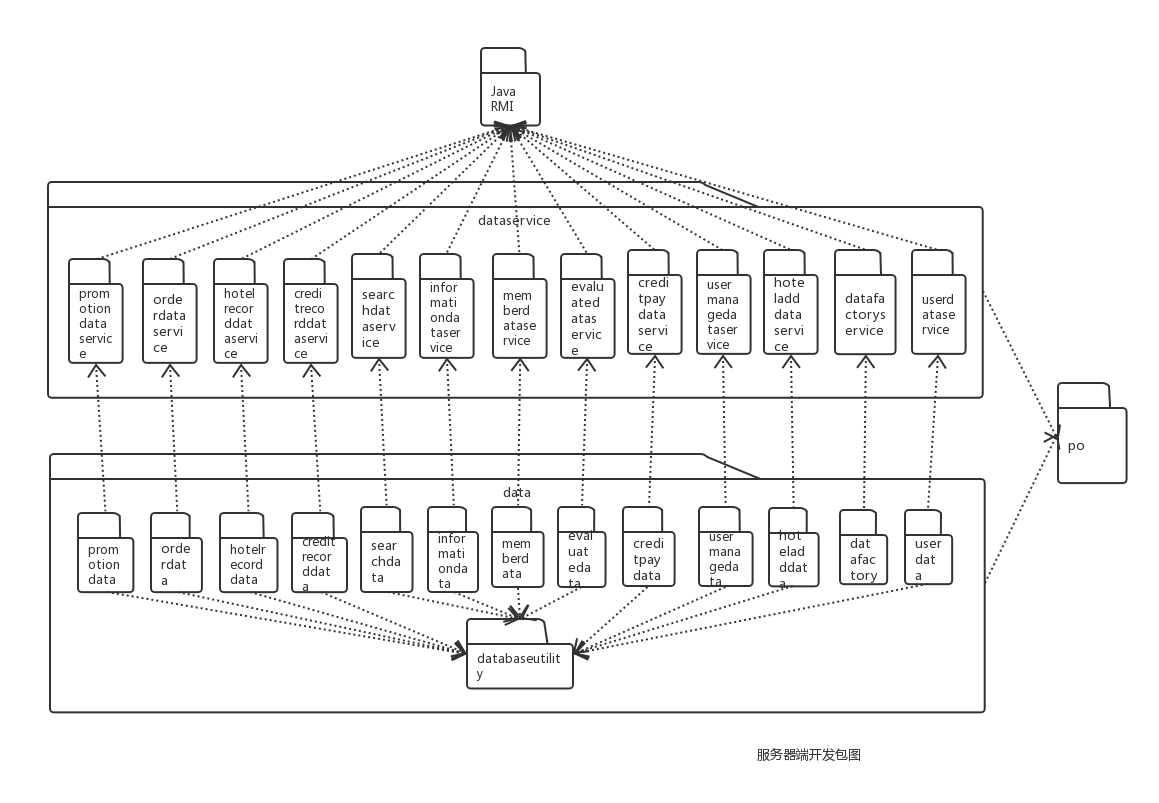
**(5)业务逻辑层的设计原理**

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

# 五．依赖视角



客户端包图



服务器端包图