

酒店预订系统软件详细设计描述文档

Winter Studio



2016-10-28

编著：朱润之 徐天泽 杨凯 张磊

南京大学软件学院

目录

[1.引言 1](#_Toc470288066)

[1.1编制目的 1](#_Toc470288067)

[1.2词汇表 1](#_Toc470288068)

[1.3参考资料 2](#_Toc470288069)

[1.4 变更记录 2](#_Toc470288070)

[2.产品概述 2](#_Toc470288071)

[3.体系结构概述 2](#_Toc470288072)

[4.结构视角 2](#_Toc470288073)

[4.1 业务逻辑层的分解 2](#_Toc470288074)

[4.1.1 ordersbl模块 3](#_Toc470288075)

[4.1.2 Loginbl模块 9](#_Toc470288076)

[4.1.3 userbl模块 14](#_Toc470288077)

[4.1.4 hotel模块 21](#_Toc470288078)

[4.1.5 roombl模块 26](#_Toc470288079)

[4.1.6promotionbl模块 30](#_Toc470288080)

[4.1.7 memberbl模块 33](#_Toc470288081)

[4.1.8 Evaluatebl模块 37](#_Toc470288082)

[5.依赖视角 40](#_Toc470288083)

# 1.引言

## 1.1编制目的

本报告详细完成对酒店预订系统的详细设计，达到知道后续软件构造的谜底，同时实现和测试人员的沟通

本报告面向开发人员，测试人员，和最终用户而编写，是了解系统的导航。

## 1.2词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
|  |  |  |

## 1.3参考资料

## 1.4 变更记录

|  |  |  |  |
| --- | --- | --- | --- |
| 修改时间 | 修改人员 | 修改内容 | 版本号 |
| 10.28 | 小组四人 | 整合 | V1.0 |
| 12.20 | 张磊 | 修改了Member，Rromotions模块 | V2.0 |
| 12.23 | 杨凯 | 修改了user，evaluate模块 | V3.0 |
|  |  |  |  |
|  |  |  |  |

# 2.产品概述

本系统是为方便酒店预订而提供的网络酒店预订系统，由服务器端，客户端，酒店工作人员端，网站营销人员端，网站管理人员端组成，完成对订单，酒店等相关内容的操作，具体参考酒店预订系统用例文档以及酒店预订系统软件需求规格说明文档中对产品的概括性描述

# 3.体系结构概述

本系统采用分层风格，将系统分为3层：展示层（presentation），业务逻辑层（businesslogic）,数据层（data），展示层负责用户交互的实现，业务逻辑层负责业务逻辑的实现，数据层负责数据的持久化和访问。具体参见酒店预订系统体系结构描述文档对体系结构的概述。

# 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描述文档。

### 4.1.1 ordersbl模块

（1）模块概述

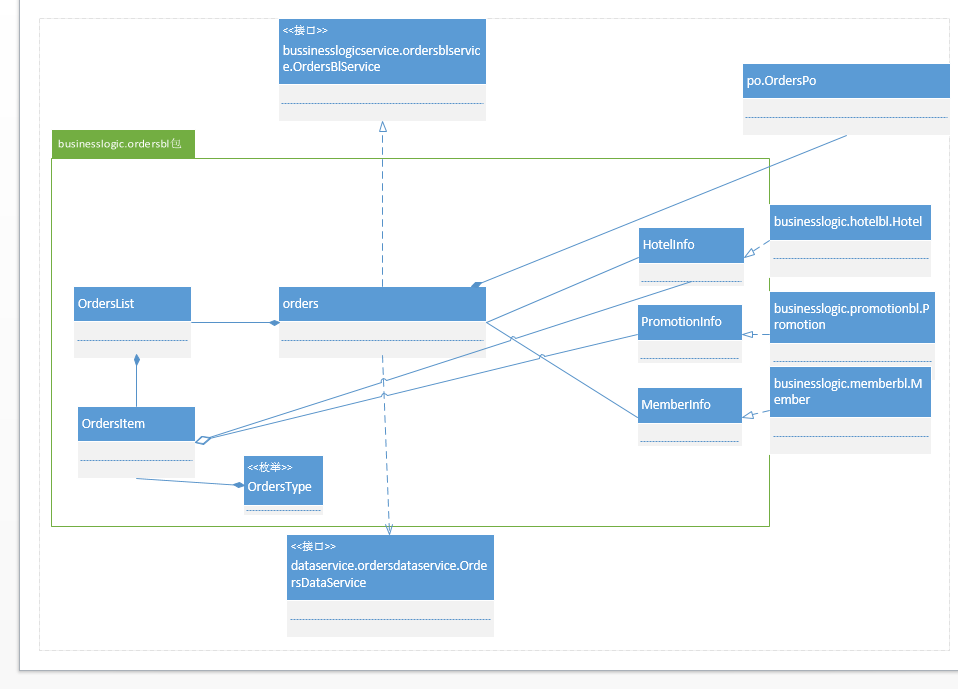
ordersbl模块承担的需求参见需求规格说明文档功能需求以及相关非功能需求

ordersbl模块的职责以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（presentation），业务逻辑层（businesslogic），数据层（data）。每一层之间为了增加灵活性，我们会添加接口。例如展示层与业务逻辑层之间，我们添加bussinesslogicservice.ordersblservice.OrdersBlService接口。业务逻辑层和数据层之间，我们添加了dataservice.ordersdataservice.OrdersDataService接口。Orders类主要负责逻辑的控制与职责的分配。OrdersPo是作为订单历史记录以及订单数据的持久化对象被添加到设计模型中去的。OrdersItem以及OrdersList作为记录类添加，OrdersItem保有订单的详细信息以及相应的根据promotioninfo计算金额的职责。OrdersList封装了关于OrdersItem的数据集合的数据结构的秘密。OrdersType作为枚举类，枚举出可能出现的订单状态，组合于OrdersItem。HotelInfo,PromotionInfo,MemberInfo都是根据依赖导致原则，为了消除循环依赖儿产生的接口

ordersbl模块的设计如下图所示



Ordersbl模块各个类的职责如下表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| Orders | 订单的领域模型对象，拥有一次订单所持有的会员，酒店，房间类型，房间数量，总价格，入住人数，订单时间，退房时间，最晚订单执行时间，订单类型用户信用值等信息，可以帮助完成与订单有关界面所需要的服务 |
| OrdersList | 记录订单的对象，拥有该次服务的会员信息，会员所持有的订单列表。 |
| OrdersItem | 负责记录保存一次订单的所有信息，并根据促销策略，计算出最优总价 |
| OrderType | 枚举类，负责枚举出可能出现的所有订单状态 |
|  |  |

（3）模块内部类的接口规范

OrdersItem,Orders,OrdersList 的接口规范如下表所示

**Orders的借口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Orders.view | **语法** | **public OrderVo view()** |
| **前置条件** | **订单已经创建** |
| **后置条件** | **将需要展示的订单的vo传给展示层** |
| Orders.add | **语法** | **public ResultMessage add( int peopleNum,int roomNum, Date beginDate, Date endDate, Date deadline)** |
| **前置条件** | **date符合规则，已经setUp已执行** |
| **后置条件** | **根据输入的信息创建一个orderItem，失败返回失败信息** |
| Orders.revoke | **语法** | **public ResultMessage revoke(int orderId)** |
| **前置条件** | **orderId存在** |
| **后置条件** | **根据orderId查找对应订单，根据撤销时间，订单类型，进行撤销操作， 成功返回success以及扣除的信用值，失败返回结果信息** |
| Orders.confirm | **语法** | **public void confirm()** |
| **前置条件** | **已完成订单信息填写** |
| **后置条件** | **生成订单，结束此次订单生成任务，持久化更新涉及的领域对象的数据** |
| Orders.cancel | **语法** | **public void cancel()** |
| **前置条件** | **已完成订单信息填写** |
| **后置条件** | **删除order对象的信息，跳转到订单信息页面** |
| Orders.recover | **语法** | **public void recover(int orderId)** |
| **前置条件** | **工作人员请求撤销** |
| **后置条件** | **根据orderId查找到指定订单，将状态变更为已执行** |
| Orders.getprice | **语法** | **public double getprice(int roomid)** |
| **前置条件** | **完成订单填写** |
| **后置条件** | **根据roomid查询到对应房间，返回单价** |
| Orders.changeError | **语法** | **public void changeError(int orderId)** |
| **前置条件** | **网站营销人员在进行撤销异常订单** |
| **后置条件** | **根据orderId查找到指定订单，将状态变更为已撤销** |
| Orders.setUp | **语法** | **public void changeError(int orderId)** |
| **前置条件** | **无** |
| **后置条件** | **根据传入的信息设定订单的初始状态** |
| Orders.getList | **语法** | **public OrdersListMock getList()** |
| **前置条件** | **已经有一个订单列表被创建** |
| **后置条件** | **得到该用户的全部订单列表** |
| Orders.getHistoryHotel | **语法** | **publicArrayList<String> getHistoyHotel(String memberId)** |
| **前置条件** | **已经执行setUp** |
| **后置条件** | **查找历史订单中所有的历史酒店，返回全部的酒店列表** |
| Orders.getOrderHistory | **语法** | **public ArrayList<OrderVO> getOrderHistory(String memberId,String HotelName)** |
| **前置条件** | **传入的酒店名和用户ID存在** |
| **后置条件** | **根据用户ID和酒店名查找该用户在该酒店的全部订单** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| OrderDataService.find(int id) | **根据ID进行查找单一持久化对象** |
| OrderDataService.find(String field,int orderType) | **根据字段名和类型查找多个持久化对象** |
| OrderDataService.insert(OrdersPO po) | **插入单一持久化对象** |
| OrderDataService.delete(OrdersPO po) | **删除单一持久化对象** |
| OrderDataService.update(OrdersPO po) | **更新单一持久化对象** |
| DatabaseFactory.getOrdersDatabase | **得到Orders数据库的服务引用** |
| DatabaseFactory.getPromotionDatabase | **得到Promotion数据库的服务引用** |
| UserinfoService.findUser(int Userid) | **根据ID查找User，得到UserPO对象** |
| PromotioninfoService.getWebPromtion | **得到网站促销策略** |
| PromotioninfoService.getHotelPromtion(String hotelName) | **根据hotelName得到酒店促销策略** |
| Hotelinfo.findHotel(String hotelName) | **根据hotelName得到HotelPO对象** |
| Roominfo.findRoom(String hotelName) | **根据hotelName得到RoomPO对象** |
| Memberinfo.findMember(String userID) | **根据userID得到MemberPO对象** |
| DatabaseFactory.getHotelDatabase | **得到Hotels数据库的服务引用** |
| DatabaseFactory.getRoomDatabase | **得到Rooms数据库的服务引用** |
| DatabaseFactory.getMembersDatabase | **得到Members数据库的服务引用** |
| OrderDataService.insert(OrdersPO po) | **插入OrderPO对象** |
| OrderDataService.delete(OrdersPO po) | **删除OrderPO对象** |
| OrderDataService.update(OrdersPO po) | **更新OrderPO对象** |
| MembersInfo.getCredit() | **得到该用户的信用信息** |
| OrdersList.findOrder(int orderId) | **根据orderId在list中查找得到OrderItem** |
| OrdersList.add(OrderItem item) | **在list中添加OrderItem** |
| OrdersList.getList(OrderType) | **根据OrderType查找列表返回相应的OrdersList** |

OrdersList接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrdersList.add | **语法** | **public void add(OrderItem item)** |
| **前置条件** | **无** |
| **后置条件** | **在列表中添加一个OrderItem** |
| OrdersList.findOrder | **语法** | **public OrderItem findOrder(int orderId)** |
| **前置条件** | **无** |
| **后置条件** | **根据传入的id查找对应的OrderItem** |
| OrdersList.getList | **语法** | **public ArrayList<OrderItem> getList(OrderType)** |
| **前置条件** | **list已经创建** |
| **后置条件** | **根据传入的orderType查找列表，返回对应type的arraylist** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| OrderDataService.find(String field,int orderType) | **根据字段名和类型查找多个持久化对象** |
| OrderDataService.insert(OrdersPO po) | **插入单一持久化对象** |
| OrderDataService.delete(OrdersPO po) | **删除单一持久化对象** |
| OrderDataService.update(OrdersPO po) | **更新单一持久化对象** |
| OrderDataService.addList(OrdersList list,string memberId) | **插入一个订单列表** |

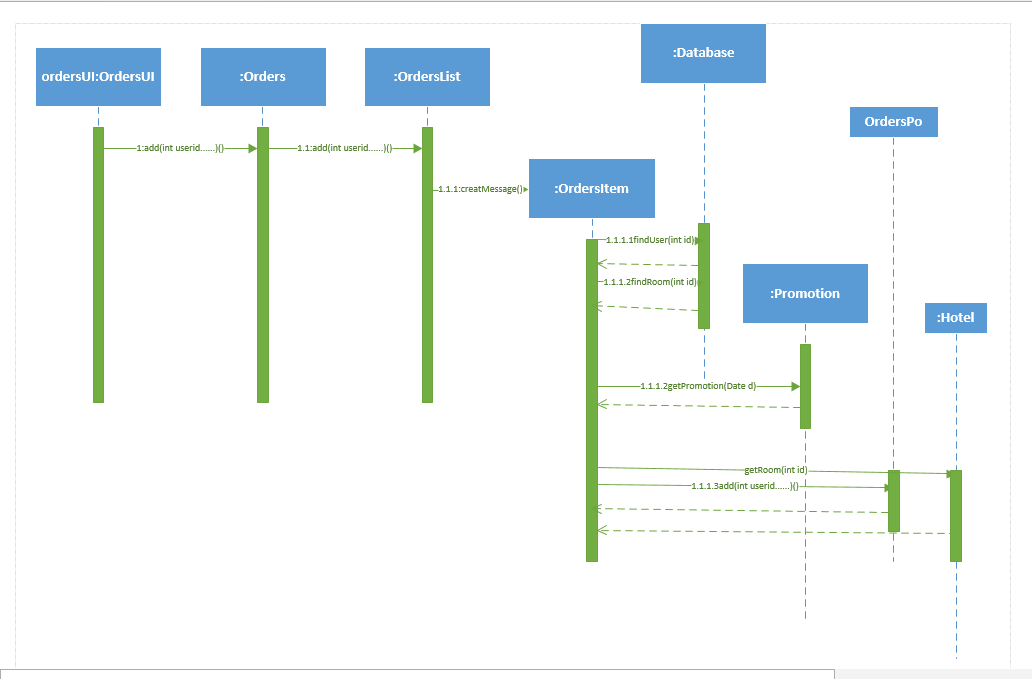
OrderItem的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderItem.geName | **语法** | **public String getName()** |
| **前置条件** | **无** |
| **后置条件** | **得到订单的用户名** |
| OrderItem.getMemberId | **语法** | **public String getMemberId()** |
| **前置条件** | **无** |
| **后置条件** | **得到订单的用户id** |
| OrderItem.getHotel() | **语法** | **public String getHotel()** |
| **前置条件** | **无** |
| **后置条件** | **得到订单的酒店名** |
| OrderItem.getType | **语法** | **public RoomType getType()** |
| **前置条件** | **无** |
| **后置条件** | **得到房间的类型** |
| OrderItem.calPrice | **语法** | **public double calPrice()** |
| **前置条件** | **无** |
| **后置条件** | **计算订单总价** |

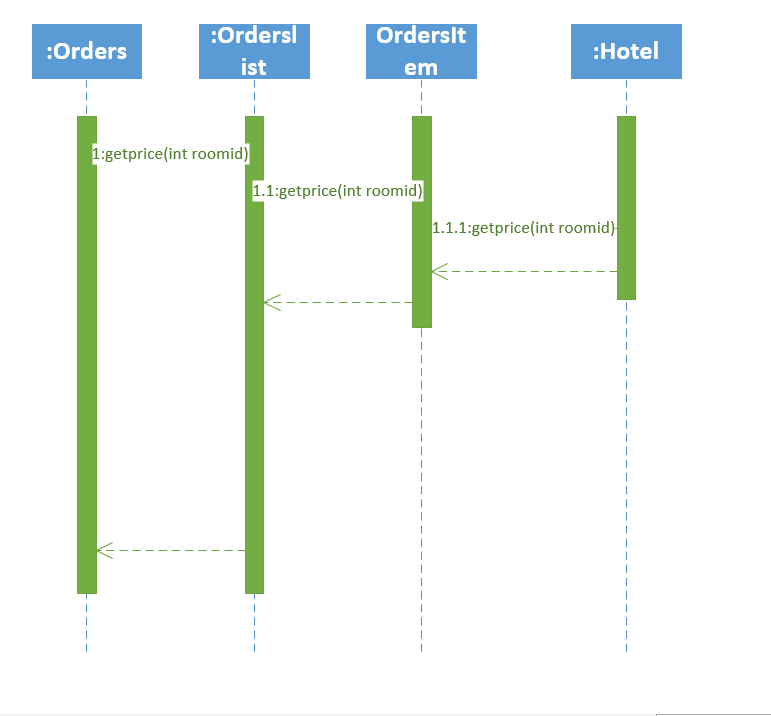
|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| MembersInfo.getName | **得到用户的用户名** |
| MembersInfo.getId | **得到用户的ID** |
| HotelsInfo.getName | **得到酒店的酒店名** |
| RoomsInfo.getRoomType | **得到房间的类型** |
| Promotions.getPrice | **根据促销策略计算价格** |

（4）业务逻辑层的动态模型

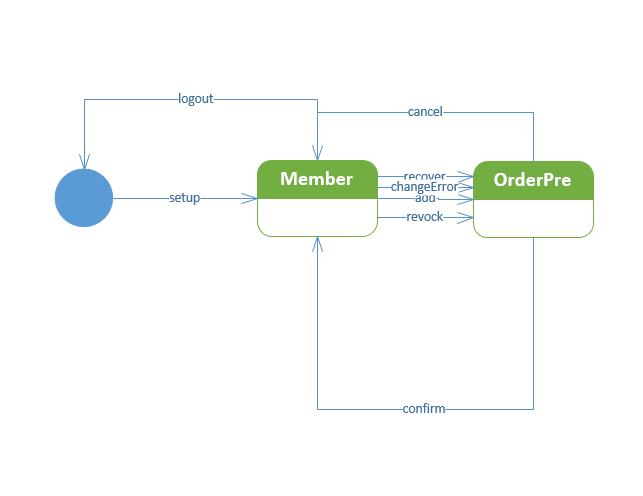
下图表示了酒店预订预订系统中，当用户输入订单信息，请求生成订单后，订单业务逻辑处理的相关对象之间的协作。



下图表示了Orders对象得到房间单价的顺序图



下图表示的状态图描述了Orders对象的生存周期的状态序列，引起转移的时间，以及因状态转移而伴随的动作。随着setup被ui调用，orders进入member状态，通过logout返回初始状态。之后调用add，revoke,recover,changeError等方法，Orders进入OrderPre状态，显示订单的预览草稿。可以通过cancel返回Member状态。之后调用confirm，完成订单添加流程，order回到member状态。



（5）业务逻辑层的设计原理

利用集中式控制风格，每个界面需要的业务逻辑集中在order中，由该对象来决定如何分配职责。

### 4.1.2 Loginbl模块

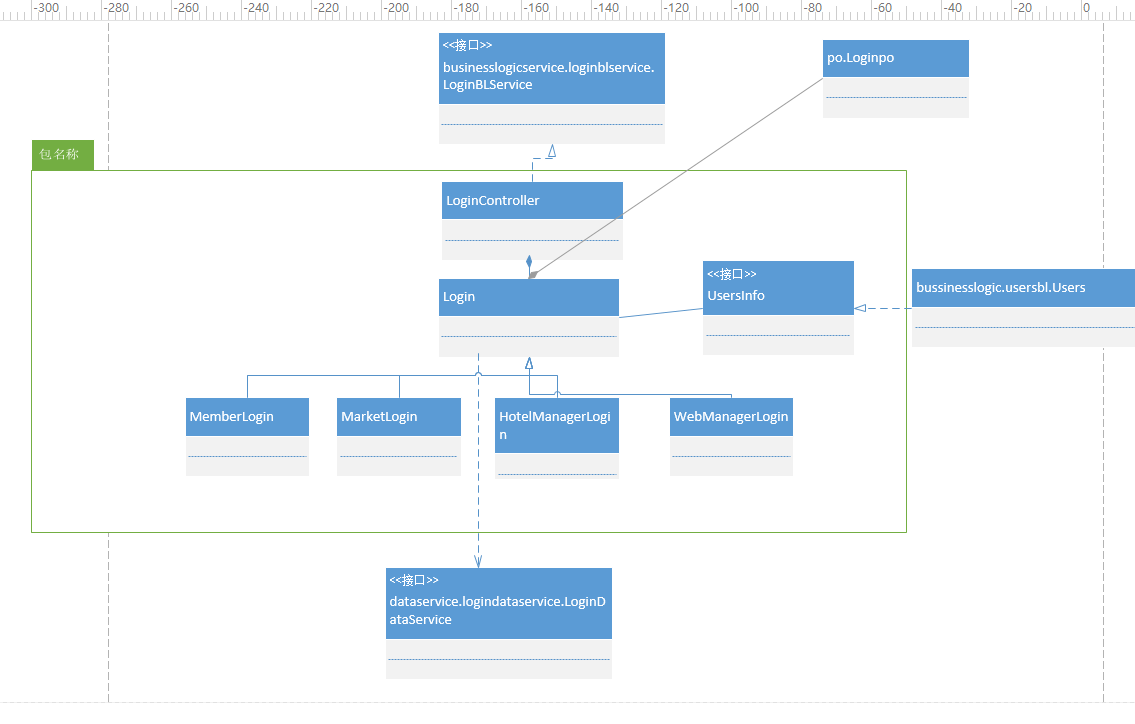
（1）模块概述

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

loginbl模块的职责与接口参见软件系统结构描述文档

（2）整体结构

为了隔离业务逻辑职责和逻辑控制职责，我们增加了LoginController，这样LoginController将不同的登陆请求分配给不同的Login对象。Loginpo是作为登陆信息的持久化对象被添加到设计模型里面去的。Login作为抽象的父类,而 MememberLogin,WebManangerLogin,HotelManagerLogin,MarketLogin作为子类继承Login。



Loginbl模块各个类的职责如表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登陆界面所需要的服务 |
| Login | 登陆的抽象父类，拥有登陆所需要的一般方法以及信息 |
| MemberLogin | 会员登录的对象，拥有会员的用户名密码以及信息，可以解决会员的登陆问题 |
| WebManagerLogin | 网站管理人员登录的对象，拥有网站管理人员的用户名密码以及信息，可以解决网站管理人员的登陆问题 |
| HotelManagerLogin | 酒店工作人员登录的对象，拥有酒店工作人员的用户名密码以及信息，可以解决酒店工作人员的登陆问题 |
| MarketLogin | 网站营销人员登录的对象，拥有网站营销人员的用户名密码以及信息，可以解决网站营销人员的登陆问题 |

（3）模块内部类的接口规范

LoginController和Login的接口规范如下表所示

LoginController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| LoginController.login | **语法** | **public ResultMessage login(String id,String password, UserType type)** |
| **前置条件** | **已创建对应的login领域对象，并且输入符合规则** |
| **后置条件** | **调用相应的login对象的login方法** |
| LoginController.logout | **语法** | **public void logout(String id)** |
| **前置条件** | **已创建对应的login领域对象，并且已执行login** |
| **后置条件** | **调用相应的login对象的logout方法** |
| LoginController.getMessage | **语法** | **public Loginpo getMessage(Sting id)** |
| **前置条件** | **已创建对应的login领域对象，并且已执行login，并且未执行logout方法** |
| **后置条件** | **调用相应的login对象的getMessage方法** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| Login.login | **进行登录操作，返回对应信息** |
| Login.logout | **进行登出操作** |
| Login.getMessage | **得到对应的登录信息** |

Login的接口规范

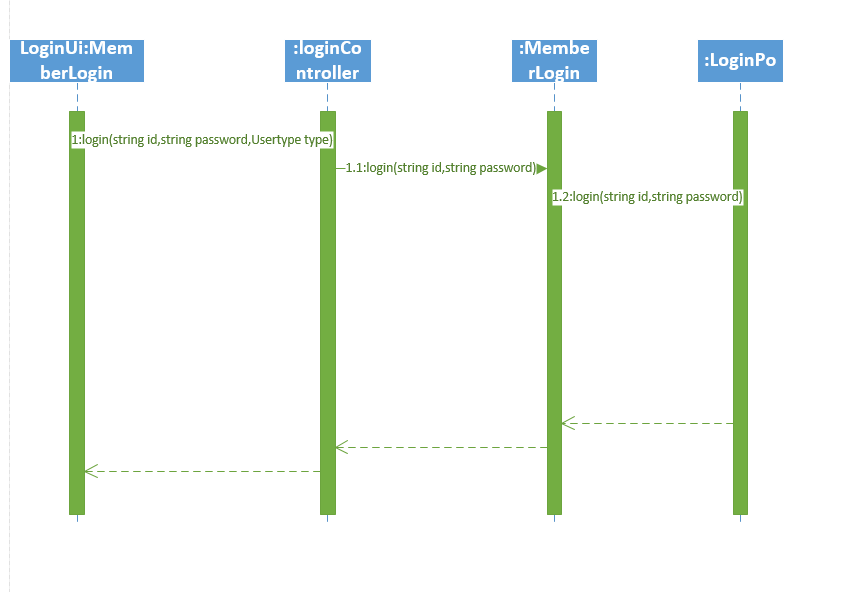
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Login.login | **语法** | **public ResultMessage login(String id,String password, UserType type)** |
| **前置条件** | **id存在，输入符合规则** |
| **后置条件** | **进行登录操作，成功返回true，失败返回失败信息** |
| Login.logout | **语法** | **public void logout(String id)** |
| **前置条件** | **已执行login** |
| **后置条件** | **进行登出操作，状态变为登出** |
| Login.getMessage | **语法** | **public Loginpo getMessage(Sting id)** |
| **前置条件** | **已执行login，并且未执行logout方法** |
| **后置条件** | **查找并返回对应id的loginpo对象** |

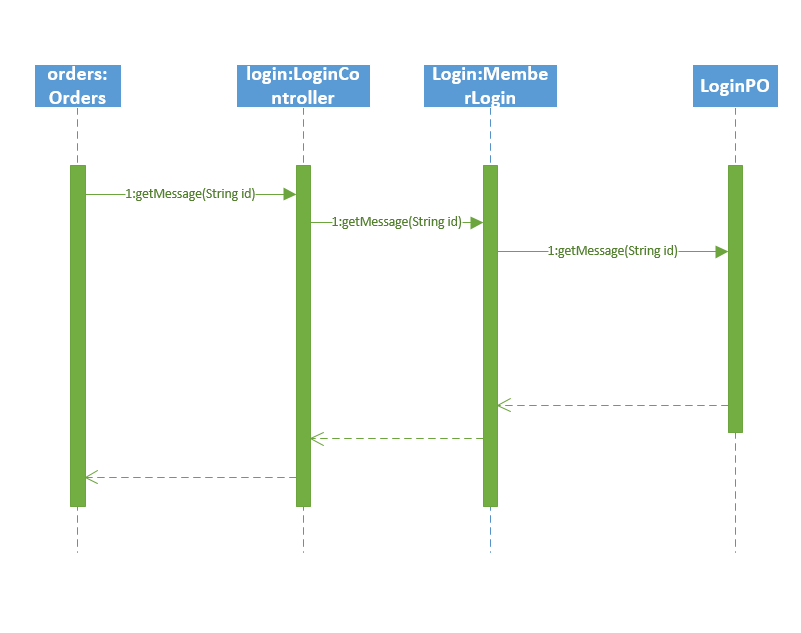
|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| User.getUser（String id） | **得到对应的userpo对象信息** |

MememberLogin,WebManangerLogin,HotelManagerLogin,MarketLogin作为子类继承Login，所需要的接口一致，只是各自的信息以及实现不同。

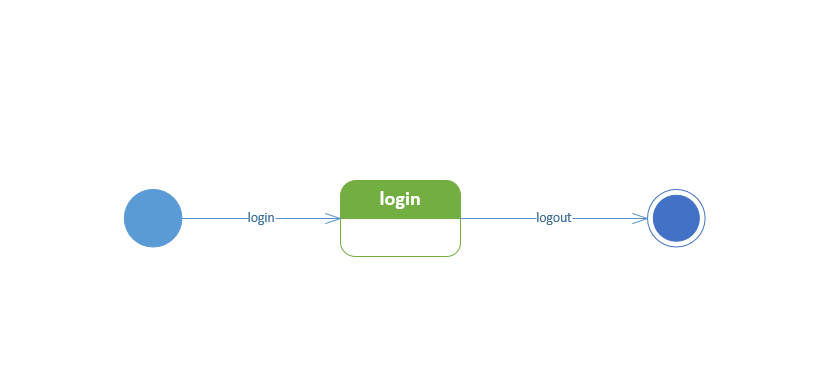
（4）业务逻辑层的动态模型

下两图分别为会员登录以及得到会员登录信息的顺序图





下图所示的状态图描述了login对象的生存期间的状态序列，随着login方法被调用，Login进入登录状态，之后通过logout结束。



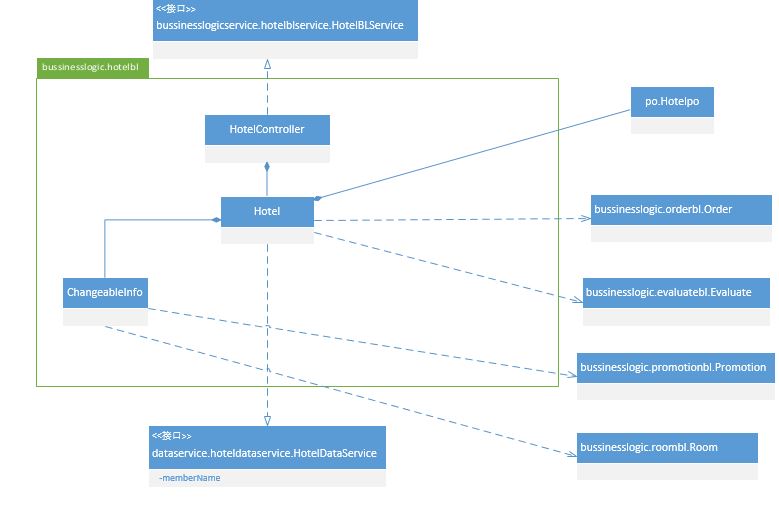
（1）模块概述

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模块承担的职责与接口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（presentation），业务逻辑层（businesslogic），数据层（data）。每一层之间为了增加灵活性，我们会添加接口。例如展示层与业务逻辑层之间，我们添加hotellogicservice. hotelblservice.HotelBlService接口。业务逻辑层和数据层之间，我们添加了hotelservice. hoteldataservice. HotelDataService接口。为了隔离业务逻辑职责与逻辑控制职责，我们增加了HotelController，这样HotelController会将不同的Hotel类相关事务委托给Hotel对象。HotelPo是作为酒店信息的持久化对象被添加到设计模型中去的。ChangeableInfo的添加是Promotion类、Room类的容器类，保有选中酒店的促销策略信息、房间信息，及相应的逻辑处理酒店的可用房间、促销策略的职责。根据依赖倒置原则，与Hotel发生双向依赖的Order类的接口被写入了orderlbl，故Hotel单向依赖这个类。除此之外，Hotel类还需要依赖Evaluate类。



### 4.1.3 userbl模块

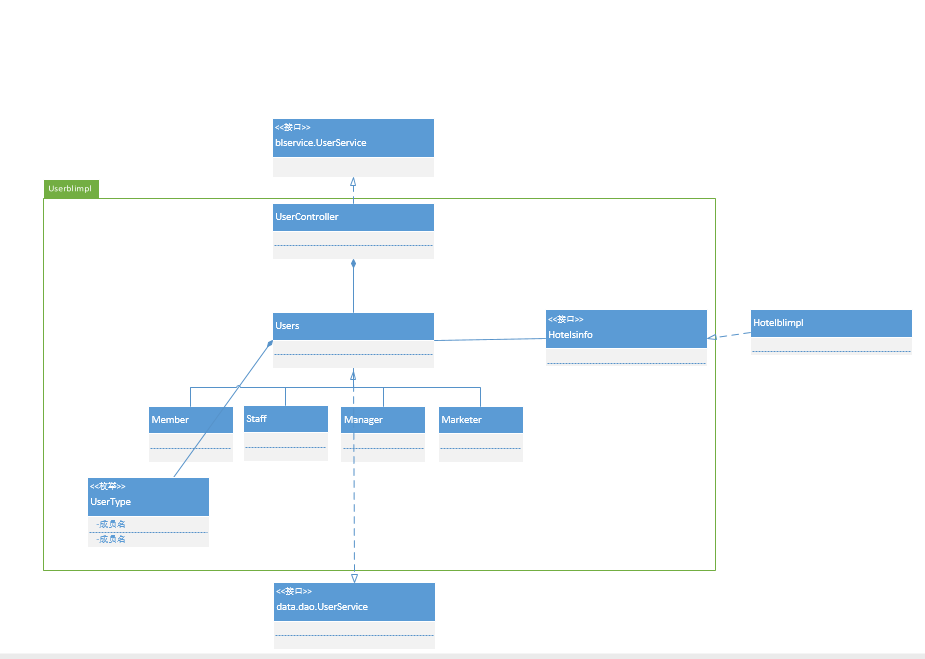
(1).模块概述

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。

(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。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将User的业务逻辑处理委托给User对象。UserPO是作为使用者信息的持久化对象被添加到设计模型中去的。UserType作为枚举类，枚举出可能出现的用户类型。HotelInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。UserPoVoTran是用来进行VO和PO之间的转换的。Usersblimpl作为抽象的父类，Memberblimpl、Staffblimpl、Managerblimpl、Marketerblimpl继承了这个Usersblimpl。设计类图如图。



Userbl模块各个类的职责如下图所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserControllerimpl | 负责实现与用户有关的界面所需要的服务 |
| Users | 使用者的领域模型对象，拥有用户的信息，帮助完成关于用户信息的修改，用户的查询，用户的添加和删除，以及登录，作为抽象父类 |
| Memberblimpl | 会员的领域模型对象，拥有会员的信息，帮助完成会员信息的修改，查询，添加和删除，以及登录 |
| Staffimpl | 酒店工作人员的领域模型对象，拥有工作人员的信息，帮助完成信息的修改，查询，添加，以及登录 |
| Manageimplr | 网站管理人员的领域模型对象，拥有管理人员的信息，帮助完成登录 |
| Marketerimpl | 网站促销人员的领域模型对象，拥有促销人员的信息，帮助完成信息的修改，查询，添加，以及登录 |
| UserType | 枚举类，枚举出所有的用户类型 |
| UserVoPoTran | 进行VO和PO之间的转换 |

(3)模块内部类的接口规范

UserController , Userblimpl的接口规范如下

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.find | 语法 | public UserVO find(String userId, UserType usertype) |
| 前置条件 | 已创建一个UserController领域对象,输入符合规范 |
| 后置条件 | 调用相应的User领域对象的find方法 |
| UserController.revoke | 语法 | public ResultMessage revoke(String userId, UserType type,UserVO user) |
| 前置条件 | 已创建User领域对象 |
| 后置条件 | 调用相应的User领域对象的revoke方法 |
| UserController.addStaff | 语法 | public ResultMessage add(StaffVO) |
| 前置条件 | 已创建Staff领域对象 |
| 后置条件 | 调用Staff领域对象的add方法 |
| UserController.addMember | 语法 | public ResultMessage addMember(MemberInformationVO member) |
| 前置条件 | 已创建Member领域对象 |
| 后置条件 | 调用Member领域对象的register方法 |
| UserController.addMarketer | 语法 | public ResultMessage add(MarketerVO) |
| 前置条件 | 已创建Marketer领域对象 |
| 后置条件 | 调用Marketer领域对象的add方法 |
| UserController.getAllUsers | 语法 | Public ArrayList<String> getAllusers(UserType type) |
|  | 前置条件 | 需要所有的成员 |
|  | 后置条件 | 调用data.dao的方法 |
| UserController.isStaffExist | 语法 | Public boolean isHotelHasStaff(String hotelName) |
|  | 前置条件 | 需要判断酒店是否存在工作人员 |
|  | 后置条件 | 调用相应staffblimpl的方法 |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| User.find(String userId, UserType usertype) | 通过UserType和UserId来查找某个用户 |
| User.revoke(String userId, UserType type,UserVO user) | 根据userid和type查找原来的user对象，按照新的user修改信息 |
| User.getAllUsers(UserType type) | 返回所有的用户列表 |
| Staff.add(StaffVO staff) | 添加一个工作人员 |
| Member.register(MmeberInformationVO member) | 用户注册 |
| Marketerblimpl.addMarketer(MarketerVO  marketerVO) | 营销人员注册 |
| StaffBlimpl.isStaffExist(String HotelName) | 判断酒店是否存在工作人员 |
| UserDao.getAllUsers(UserType type) | 得到所有的用户列表 |

Userblimpl表示Staffblimpl 、memberblimpl、marketerblimpl、managerblimpl

Memberblimpl ,StaffBlimpl,Marketerblimpl,Managerblimpl的部分接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| \*\*.revoke | **语法** | **public ResultMessage revoke(String userId, UserType type,UserVO user)** |
| **前置条件** | **userId和userType存在且匹配，并且新的user和旧的有修改的地方** |
| **后置条件** | **修改原有的UserId的信息，将其改为User** |
| \*\*.find | **语法** | **Public UserVO find(String userId, UserType type)** |
| **前置条件** | **UserId和type匹配** |
| **后置条件** | **返回新的UserPO信息** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| UserDao.find(String id,UserType) | **根据ID进行查找单一持久化对象** |
| UserDao.update(String id,UserVO user, UserType type) | **根据字段名和类型查找多个持久化对象** |

\*\*表示memberblimpl 、staffblimpl 、marketerblimpl、managerblimpl

Memberblimpl其他接口规范

|  |  |  |
| --- | --- | --- |
| Memberblimpl.register | 语法 | Public ResultMessage register(MemberInformationVO member) |
|  | **前置条件** | **Id不重复** |
|  | **后置条件** | **添加新的member到数据库** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| UserDao.add(UserPO po) | **插入**UserPO**对象** |

Staffblimpl的其他借口规范

|  |  |  |
| --- | --- | --- |
| staffblimpl.addStaff | 语法 | Public ResultMessage addStaff(StaffVO member) |
|  | **前置条件** | **Id不重复** |
|  | **后置条件** | **添加新的staff到数据库** |
| Staffblimpl.isStaffExist | **语法** | **Public Boolean isStaffExist(String HotelName)** |
|  | **前置条件** | **酒店存在** |
|  | **后置条件** |  |

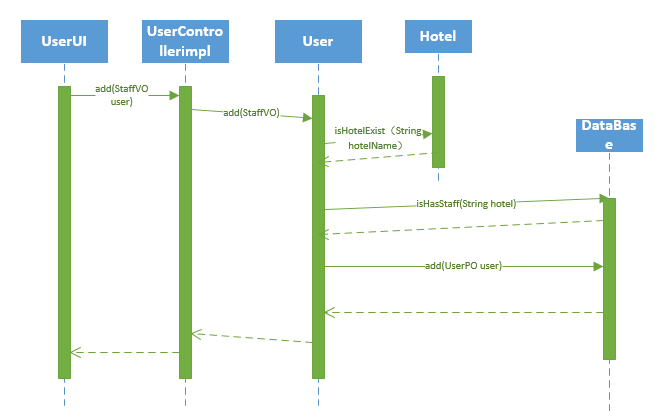
|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| UserDao.add(UserPO po) | **插入**UserPO**对象** |
| UserDao.isHotelExist(String HotelName) | **判断酒店是否存在工作人员** |

Marketerblimpl其他接口规范

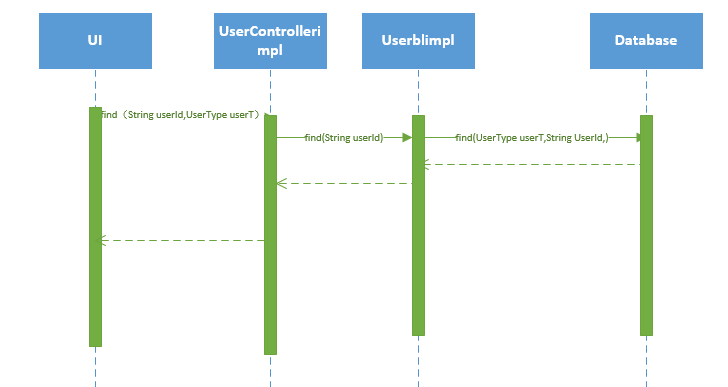
|  |  |  |
| --- | --- | --- |
| Marketerblimpl.addMarketer | 语法 | Public ResultMessage addMarketer(MarketerVO marketer) |
|  | **前置条件** | **Id不重复** |
|  | **后置条件** | **添加新的Marketer到数据库** |

|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| UserDao.add(UserPO po) | **插入**UserPO**对象** |

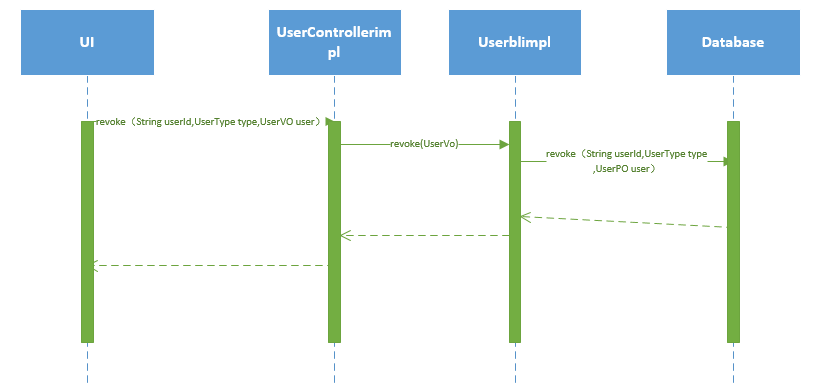
下图表示添加酒店工作人员时，各个类的协作关系



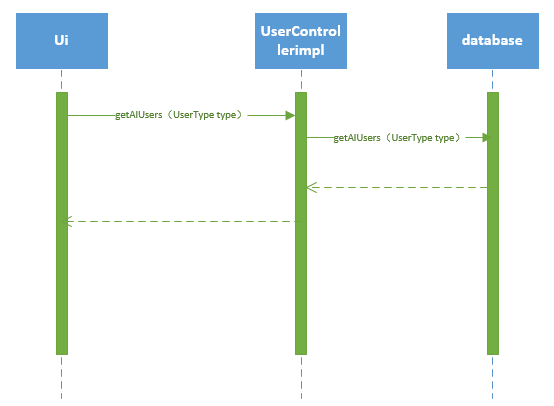
下图是查找用户时，各个类的协作关系



下图表示修改用户信息时，各个类的协作关系

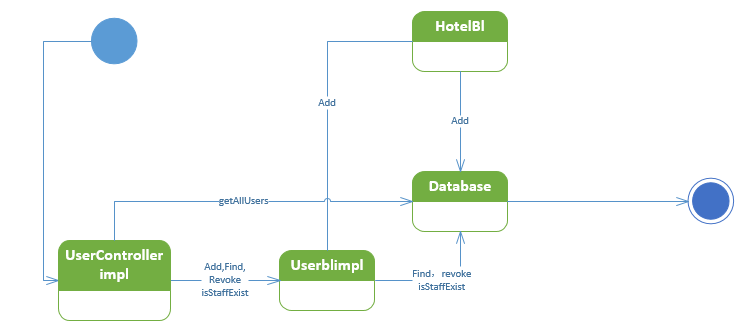


下图表示得到所有用户列表时各个类的协作关系



（4）业务逻辑层的动态模型

下图表示的状态图描述了User对象的生存周期的状态序列，引起专一的时间，以及因状态转移而伴随的动作。随着userController被调用，从add，find，revoke，isStaffExist方法进入相应的userblimpl状态，从getAllUsers进入database，然后进入database状态，随后结束。



### 4.1.4 hotel模块

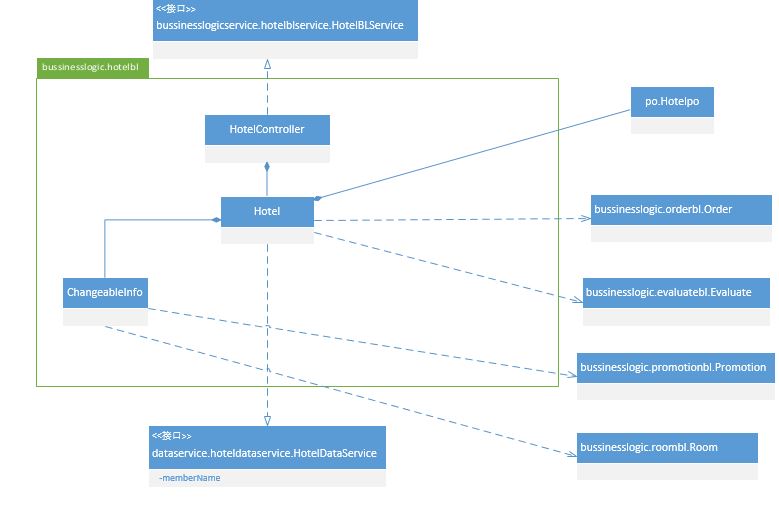
（1）模块概述

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模块承担的职责与接口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（presentation），业务逻辑层（businesslogic），数据层（data）。每一层之间为了增加灵活性，我们会添加接口。例如展示层与业务逻辑层之间，我们添加hotellogicservice. hotelblservice.HotelBlService接口。业务逻辑层和数据层之间，我们添加了hotelservice. hoteldataservice. HotelDataService接口。为了隔离业务逻辑职责与逻辑控制职责，我们增加了HotelController，这样HotelController会将不同的Hotel类相关事务委托给Hotel对象。HotelPo是作为酒店信息的持久化对象被添加到设计模型中去的。ChangeableInfo的添加是Promotion类、Room类的容器类，保有选中酒店的促销策略信息、房间信息，及相应的逻辑处理酒店的可用房间、促销策略的职责。根据依赖倒置原则，与Hotel发生双向依赖的Order类的接口被写入了orderlbl，故Hotel单向依赖这个类。除此之外，Hotel类还需要依赖Evaluate类。



hotelbl模块各个类的职责如下图所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| HotelController | 负责实现对应于酒店相关界面所需要的服务，包括添加酒店界面、维护酒店信息界面、搜索、浏览、评价酒店界面 |
| Hotel | 系统用户的领域模型对象，拥有酒店所持有的地址、商圈、名称、星级、评分、历史评价信息等信息，可以帮助完成添加酒店、维护已有酒店信息，搜索、浏览、评价酒店界面所需要的服务 |
| ChangeableInfo | 酒店类的容器类辅助类，负责记录酒店不断变化的房间信息、促销策略信息，并完成相应的逻辑处理工作 |

（3）模块内部类的接口规范

HotelController和Hotel的接口规范如下表所示

**HotelController**的接口规范

|  |  |  |
| --- | --- | --- |
| **提供的服务（供接口）** | | |
| HotelController.searchHotel | 语法 | public ArrayList<HotelVO> searchHotel (HotelVO h,SequencingStrategy s); |
| 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searchHotel方法 |
| HotelController.browseHotelDetails | 语法 | public HotelVO browseHotelDetails (String HotelName,SequencingStrategy s); |
| 前置条件 | 已创建一个Hotel领域对象， 已经浏览或者搜索酒店 |
| 后置条件 | 调用Hotel领域对象的browseHotelDetails方法 |
| HotelController.evaluateHotel | 语法 | public ResultMessage evaluateHotel(string evaluation,string HotelName); |
| 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evaluateHotel方法 |
| HotelController.updateInfo | 语法 | public ResultMessage updateInfo(HotelVO h); |
| 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updateInfo方法 |
| HotelController.addHotel | 语法 | public ResultMessage addHotel(HotelVO h); |
| 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addHotel方法 |
| HotelController. findHotel | 语法 | public HotelPO findHotel(String hotelName) |
| 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findHotel方法 |
| HotelController. findHotel | 语法 | public ResultMessage addHotel(HotelPO hotelPO) |
| 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findHotel方法 |
| HotelController. findRoom | 语法 | public ArrayList<RoomPO> findRoom(String hotelName) |
| 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findHotelRoom方法 |
| HotelController. addHotelEvaluation | 语法 | publicResultMessage addHotelEvaluation(EvaluatePO evaluatePO) |
| 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addHotelEvaluation方法 |
| **需要的服务（需接口）** | | |
| Hotel.searchHotel | 搜索或浏览符合条件的酒店 | |
| Hotel.browseHotelDetails | 浏览选择的酒店全部信息 | |
| Hotel.updateInfo | 维护某一酒店的基本信息 | |
| Hotel.addHotel | 添加酒店的信息 | |
| Hotel.evaluateHotel | 对某一酒店进行评分和评价 | |
| Hotel. findHotel | 查找某一酒店信息 | |
| Hotel.addHotel | 添加酒店 | |
| Hotel. findRoom | 查找某一酒店的房间信息 | |
| Hotel. addHotelEvaluation | 添加对某一酒店的评价与评分 | |

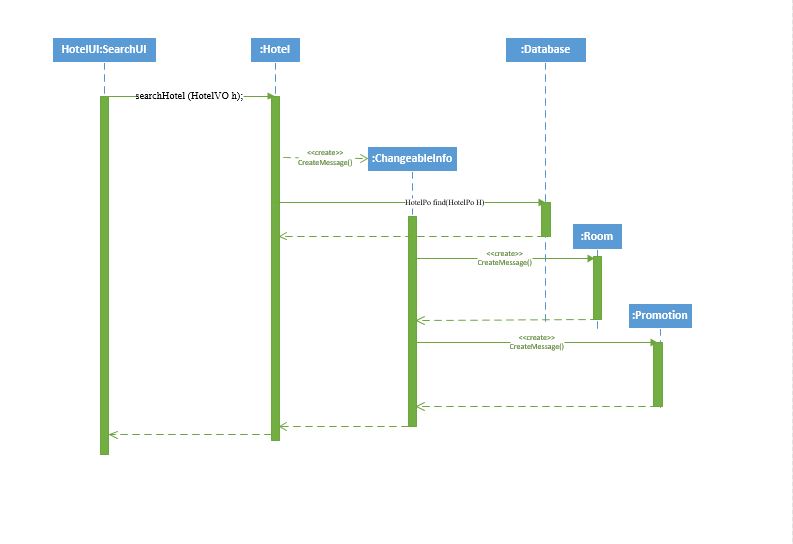
**Hotel**的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Hotel.searchHotel | 语法 | public ArrayList<HotelVO> searchHotel (HotelVO h, SequencingStrategy s); |
| 前置条件 | 已添加酒店所属商圈、地址以及酒店名称、房间、星级或评分区间 |
| 后置条件 | 返回符合条件的酒店列表，并根据SequencingStrategy给出相应的列表排列顺序 |
| Hotel.browseHotel | 语法 | public ArrayList <HotelVO> browseHotel(HotelVO h, SequencingStrategy s); |
| 前置条件 | 已添加酒店所属商圈、地址 |
| 后置条件 | 返回相应地址、商圈的酒店列表，并根据SequencingStrategy给出相应的列表排列顺序 |
| Hotel.browseHotelDetails | 语法 | public HotelVO browseHotelDetails (String HotelName,String userName); |
| 前置条件 | 已添加酒店名称 |
| 后置条件 | 返回酒店的**地址，简介，设施服务，客房类型,历史订单情况** |
| Hotel.evaluateHotel | 语法 | public ResultMessage evaluateHotel(string evaluation,string HotelName); |
| 前置条件 | 已添加对酒店的评价信息 |
| 后置条件 | 在一个评论回合中，增加会员的评价信息 |
| Hotel.updateInfo | 语法 | public ResultMessage updateInfo(HotelVO h); |
| 前置条件 | 已添加更新后的酒店信息 |
| 后置条件 | 在一个更新回合中，添加酒店的更新信息 |
| Hotel. findHotel | 语法 | public HotelPO findHotel(String hotelName) |
| 前置条件 | 已添加酒店名称 |
| 后置条件 | 在一个添加回合中，返回酒店相关信息 |
| Hotel. addHotel | 语法 | public ResultMessage addHotel(HotelPO hotelPO) |
| 前置条件 | 已添加酒店信息 |
| 后置条件 | 在一个添加回合中，返回添加是否成功的信息 |
| Hotel. findRoom | 语法 | public ArrayList<RoomPO> findRoom(String hotelName) |
| 前置条件 | 已添加酒店名称 |
| 后置条件 | 在一个添加回合中，返回酒店酒店相关信息 |
| Hotel. addHotelEvaluation | 语法 | publicResultMessage addHotelEvaluation(EvaluatePO evaluatePO) |
| 前置条件 | 已添加评价信息 |
| 后置条件 | 在一个添加回合中，返回添加是否成功的信息 |

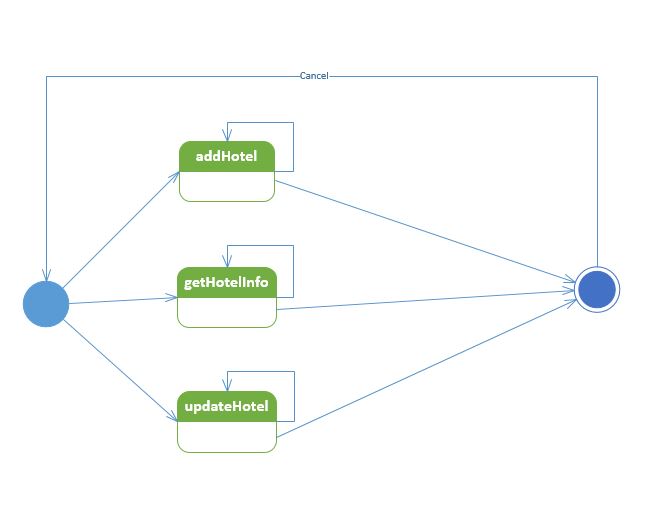
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| HotelDataService.find  (HotelPo h) | 根据输入的酒店信息查找多个持久化对象 |
| HotelDataService.insert  (HotelPo h) | 插入单一持久化对象 |
| HotelDataService.delete  (HotelPo h) | 删除单一持久化对象 |
| HotelDataService.update (HotelPo h) | 更新单一持久化对象 |
| DataFactoryService.getHot  elDataService | 得到酒店数据库的服务的引用 |
| Room.getRoomInfo(string HotelName) | 得到该酒店的房间信息 |
| Promotion.addHotel  (string HotelName) | 得到该酒店的折扣信息 |
| Order.getHotelHistory  (string UserID) | 得到某一会员的酒店预订历史记录 |
| Order.getOrderHistory  (string UserID,string HotelName) | 得到某一会员在某一酒店的订单历史记录 |

（4）业务逻辑层的动态模型

下图表示了酒店预订系统中，当用户选定了搜索酒店的地址和商圈、输入了其他信息后，Hotel业务逻辑处理的相关对象之间的协作



下图描述了Hotel对象在搜索酒店信息时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。



### 4.1.5 roombl模块

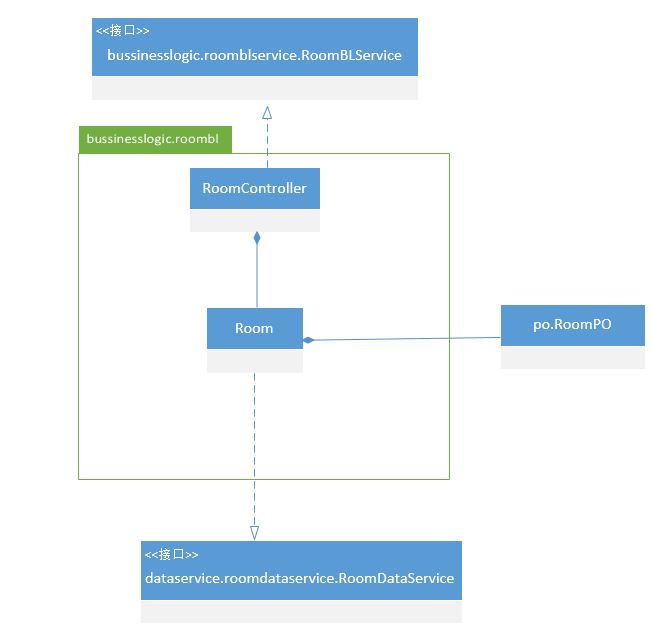
（1）模块概述

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。

Roombl模块承担的职责与接口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（presentation），业务逻辑层（businesslogic），数据层（data）。每一层之间为了增加灵活性，我们会添加接口。例如展示层与业务逻辑层之间，我们添加roomlogicservice. roomblservice. Room BlService接口。业务逻辑层和数据层之间，我们添加了roomservice. roomdataservice. RoomDataService接口。为了隔离业务逻辑职责与逻辑控制职责，我们增加了RoomController，这样RoomController会将不同的Hotel类相关事务委托给Room对象。RoomPo是作为酒店信息的持久化对象被添加到设计模型中去的。



roombl模块各个类的职责如下图所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| RoomController | 负责实现对应于酒店相关界面所需要的服务，包括添加酒店界面、维护酒店信息界面、搜索、浏览、评价酒店界面 |
| Room | 系统用户的领域模型对象，拥有酒店所持有的地址、商圈、名称、星级、评分、历史评价信息等信息，可以帮助完成添加酒店、维护已有酒店信息，搜索、浏览、评价酒店界面所需要的服务 |

（3）模块内部类的接口规范

RoomController和Room的接口规范如下表所示

**RoomController**的接口规范

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| RoomController.addRoom | 语法 | | public ResultMessage addRoom(RoomVo r) |
| 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 |
| RoomController.recordOccupancy | 语法 | | public ResultMessage recordOccupancy (OccupancyVo o) |
| 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recordOccupancy方法 |
| RoomController.recordCheckout | 语法 | | public ResultMessage recordCheckout (CheckoutVo c) |
| 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recordCheckout方法 |
| RoomController. getRoomInfo | 语法 | | public ArrayList<RoomPO> getRoomInfo(String HotelName) |
| 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recordCheckout方法 |
| **需要的服务（需接口）** | | | |
| Room.recordOccupancy | | 记录入住信息 | |
| Room.recordCheckout | | 记录退房信息 | |
| Room.addRoom | | 添加空房间 | |
| Room. getRoomInfo | | 查找某一酒店的房间信息 | |

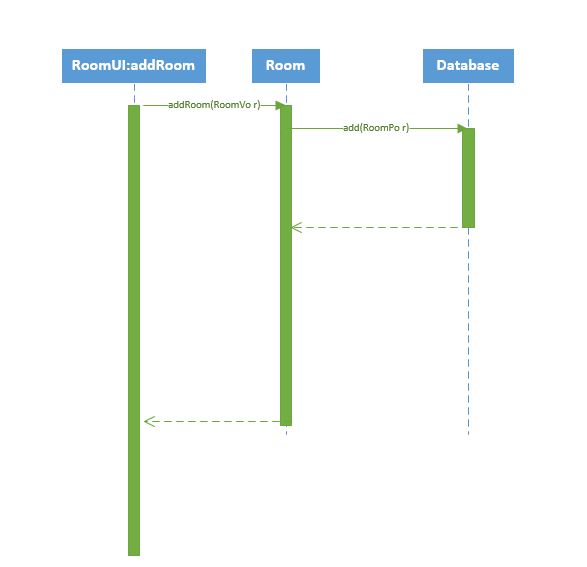
**Room** 的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Room.addRoom | 语法 | public ResultMessage addRoom(RoomVo r) |
| 前置条件 | 已添加房间的类型、数量、原始价格 |
| 后置条件 | 在一个添加回合中，增加房间信息 |
| Room.recordOccupancy | 语法 | public ResultMessage recordOccupancy (OccupancyVo o) |
| 前置条件 | 已添加入住类型、房间号、入住时间、预计离开时间 |
| 后置条件 | 在一个记录回合中，增加入住信息 |
| Room.recordCheckout | 语法 | public ResultMessage recordCheckout (CheckoutVo c) |
| 前置条件 | 已添加实际退房时间、房间号 |
| 后置条件 | 在一个记录回合中，增加退房信息 |
| Room. getRoomInfo | 语法 | public ArrayList<RoomPO> getRoomInfo(String HotelName) |
| 前置条件 | 已添加酒店名称 |
| 后置条件 | 在一个搜索回合中，返回房间相关信息 |

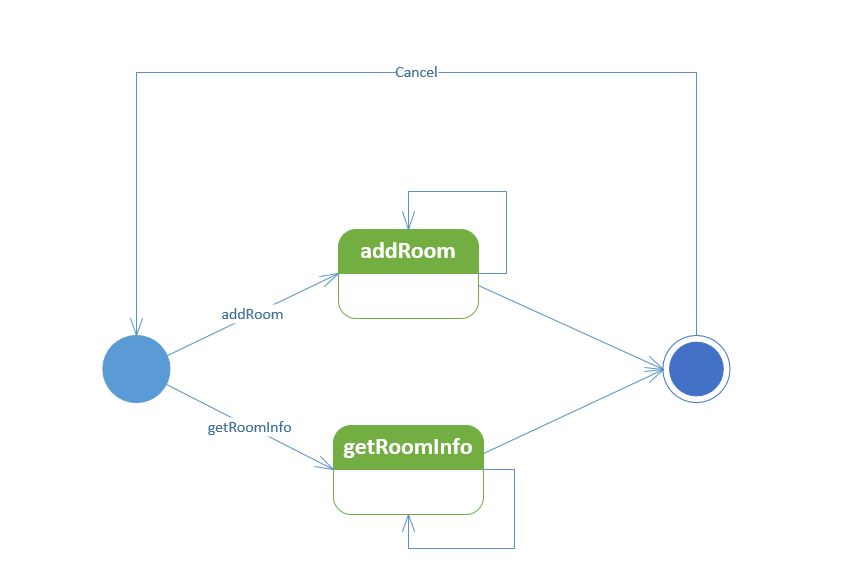
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| RoomDataService.find  (RoomPo h) | 根据输入的房间信息查找多个持久化对象 |
| RoomDataService.insert  (RoomPo h) | 插入单一持久化对象 |
| RoomDataService.delete  (RoomPo h) | 删除单一持久化对象 |
| RoomDataService.update (RoomPo h) | 更新单一持久化对象 |
| DataFactoryService.get  RoomDataService | 得到房间数据库的服务的引用 |

（4）业务逻辑层的动态模型

下图表示了酒店预订系统中，当酒店工作人员输入了房间的数量、类型、原始价格等信息后，Room业务逻辑处理的相关对象间的协作



下图描绘了Room对象在添加空房间的生存期间的状态序列、引起的转移的时间，以及因状态转移而伴随的动作。



### 4.1.6promotionbl模块

（1）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，业务逻辑层，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promotionblservice接口.Promotions对象是所有策略对象的父类，其他的策略对象PromotionI,II,III,IV,V,VI都要继承该对象。PromotionPO,HotelPO,MemberPO是作为管理信息的持久化对象被添加到设计模型中的。

Promotionbl模块的设计如图4.1.5-1所示

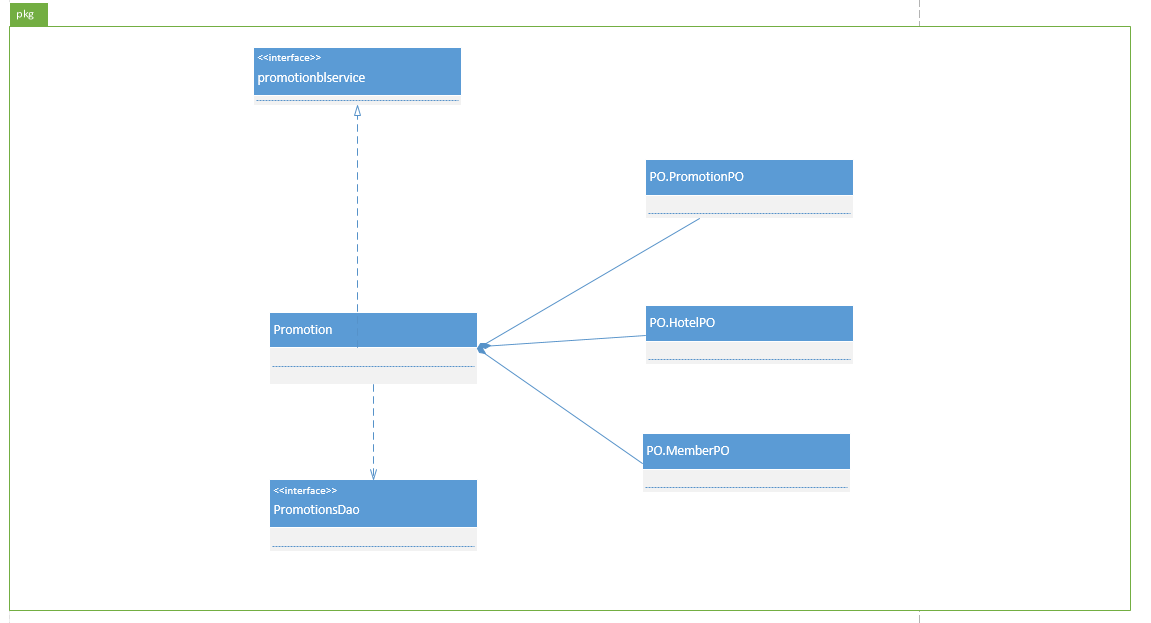


图4.1.5-1

Promotionbl模块各个类的职责如4.1.5-2所示

图4.1.5-2promotionbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
| Promotionservice | 促销策略的领域模型对象，拥有制定策略的种类，时间，折扣等信息，可以帮助添加策略，删除策略所需要的服务 |

（3）模块内部类的接口规范

Promotionservice的接口规范如下

Promotionservice模块的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Promotionservice.addPromotionsI | 语法 | public boolean addPromotionsI(PromotionsIVO promotionsIVO); |
| 前置条件 | 工作人员想要制定策略 |
| 后置条件 | 完成促销策略内容的制定，返回促销策略方案 |
| Promotionservice.deletePromotions | 语法 | Public boolean deletePromotions(PromotionsVO promotionsVO); |
| 前置条件 | 存在促销策略 |
| 后置条件 | 删除已有的促销策略，返回促销策略初始界面 |
| Promotionservice.getPrice | 语法 | Public PriceInfo getPrice(String hotel,double price,int roomNum,String userId,int days); |
| 前置条件 | 得到订单信息 |
| 后置条件 | 返回订单价格 |
| Promotionservice.getHotelPromotion | 语法 | public ArrayList<String> getHotelPromotion(String hotel); |
| 前置条件 | 存在该酒店 |
| 后置条件 | 返回酒店策略列表 |

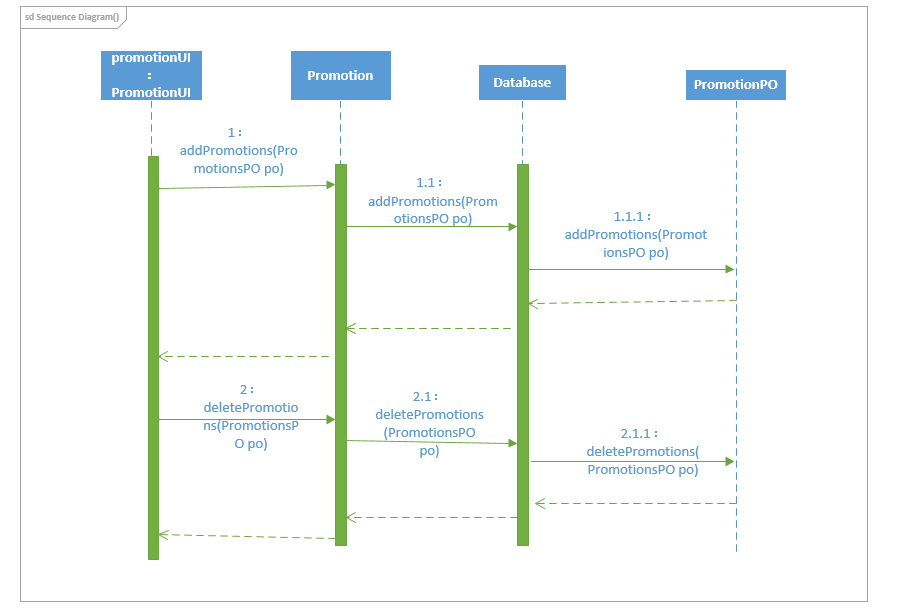
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| Promotiondao.addPromotions(PromotionPO po) | 插入单一持久化对象 |
| Promotiondao.deletePromotions(PromotionPO po) | 删除单一持久化对象 |

注：Promotion表示各种策略的制定。

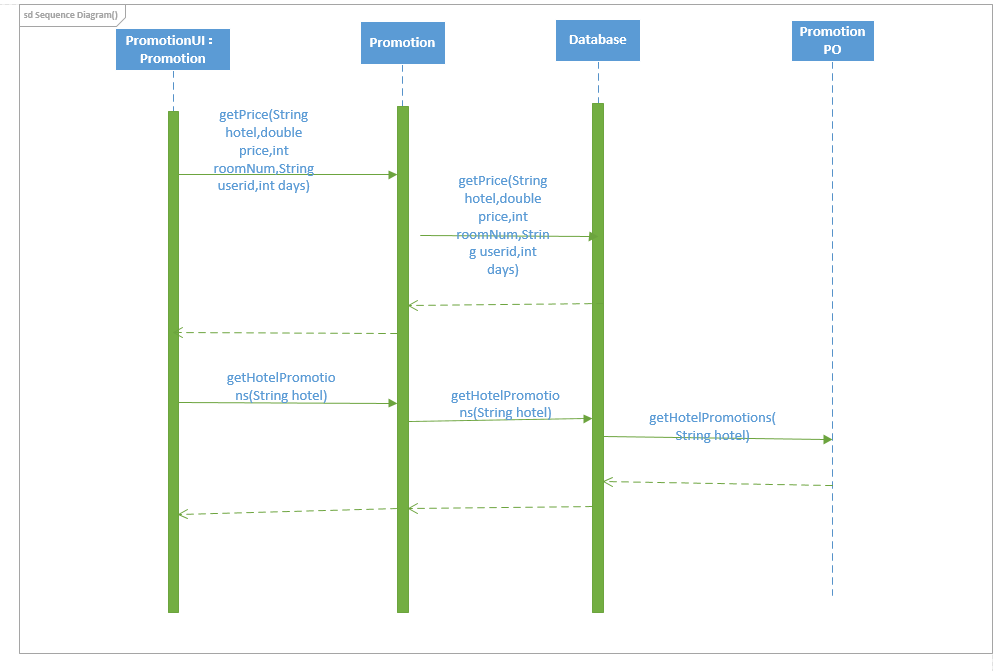
addPromotions有六个类似的方法，这里不再过多描述，只是方法的参数，实现不同

（4）业务逻辑层的动态模型

下图表明当工作人员确定好策略的时间折扣之后，策略业务逻辑处理的相关对象之间的协作

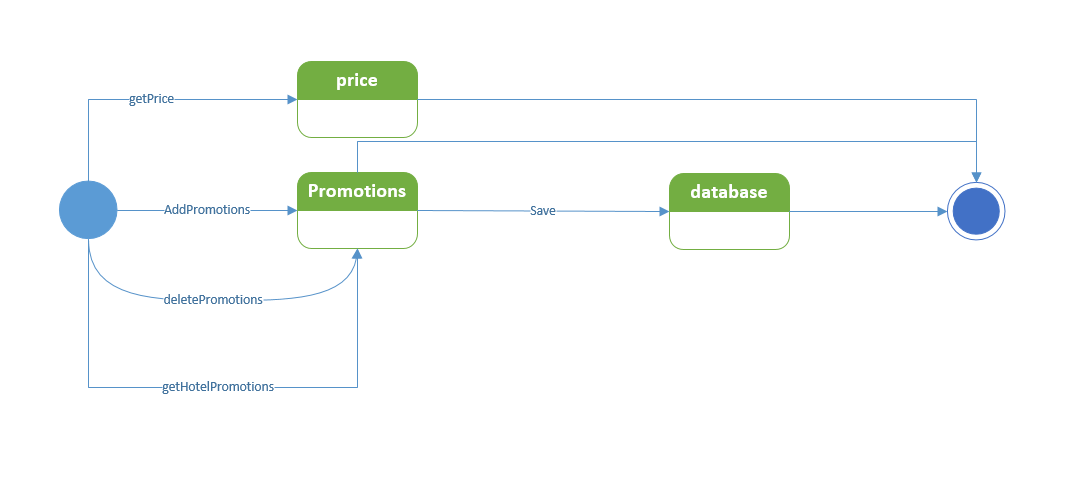


制定删除策略的顺序图



返回酒店策略和订单价格的顺序图

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addPromotions方法被UI调用，Promotion进入Promotions状态；之后通过save进入数据库。UI也可以直接调用deletePromotions方法进入Promotions状态。UI可以调用getprice方法进入price状态后返回price。UI也可以通过getHotelPromotions得到酒店策略



### 4.1.7 memberbl模块

（（1）模块概述

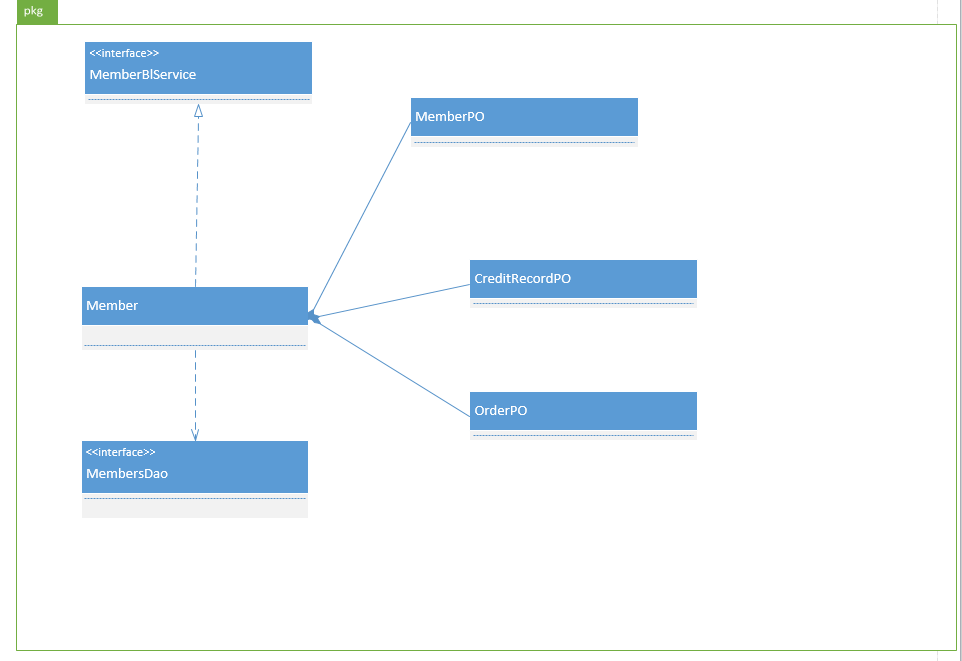
memberbl模块承担的需求参见需求规格说明文档功能需求及相关非功能需求

memberbl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，采用分层风格，将系统分为展示层，业务逻辑层，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memberservice接口。MemberPO,OrderVO是作为管理信息的持久化对象被添加到设计模型中的。

Memberbl模块的设计如下图



Memberbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
|  |  |
| Memberservice | 会员的领域模型对象，拥有会员的信息，包括其地址姓名，手机号等，可以帮助完成会员界面所需要的服务 |

（3）模块内部类的接口规范

Memberservice的接口规范如下4.1.6-1和4.1.6-2所示

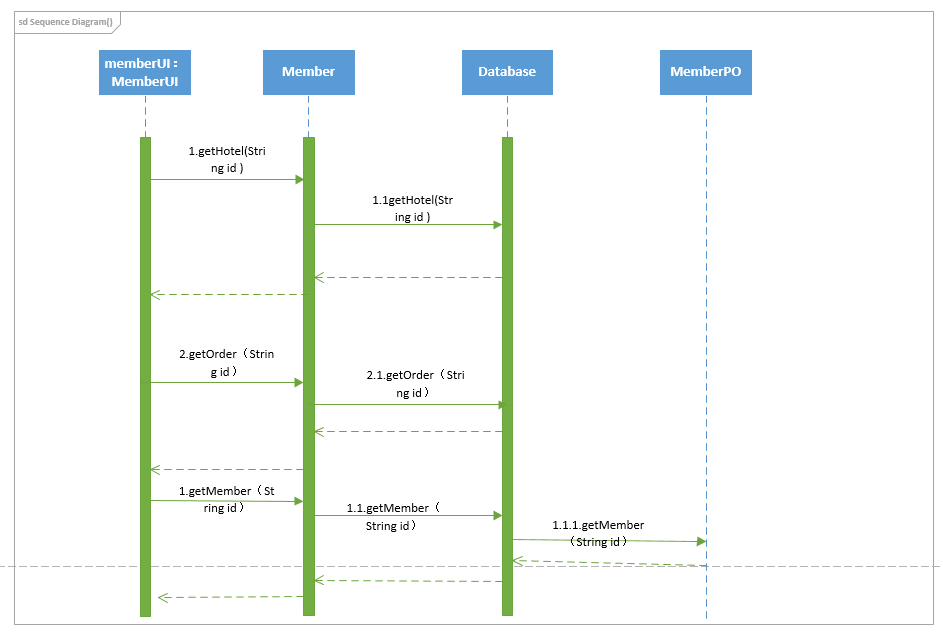
图4.1.6-2 Memberservice的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| Memberservice.getOrder | 语法 | public ArrayList<OrderVO> getOrder(String id); |
| 前置条件 | Id，符合输入规则, |
| 后置条件 | 查找该id下的用户，并返回该用户的历史订单列表 |
| Memberservice.getHotel | 语法 | public Map<String,ArrayList<OrderType>> getHotel(String id); |
| 前置条件 | Id符合规则存在该用户 |
| 后置条件 | 查找该id下的用户的历史酒店，返回历史酒店信息 |
| Memberservice.getMember | 语法 | Public MemberPO getMember(String id); |
| 前置条件 | 用户已登录,且id符合规则 |
| 后置条件 | 查找此id的用户，返回用户信息 |
| Memberservice.updateMemberCredit | 语法 | Public boolean updateMemberCredit(String id,double changecredit,int ordered,String action); |
| 前置条件 | 存在该用户 |
| 后置条件 | 更新该用户的信用记录 |
| Memberservice.getMemberCredit | 语法 | public ArrayList<CreditrecordPO> getMemberCreditRecord(String memberid); |
| 前置条件 | 存在该用户 |
| 后置条件 | 返回该用户的信用记录列表 |
| Memberservice.insertCreditRecord | 语法 | public void insertCreditRecord(CreditrecordPO po); |
| 前置条件 | 存在该用户 |
| 后置条件 | 插入信用记录到数据库 |

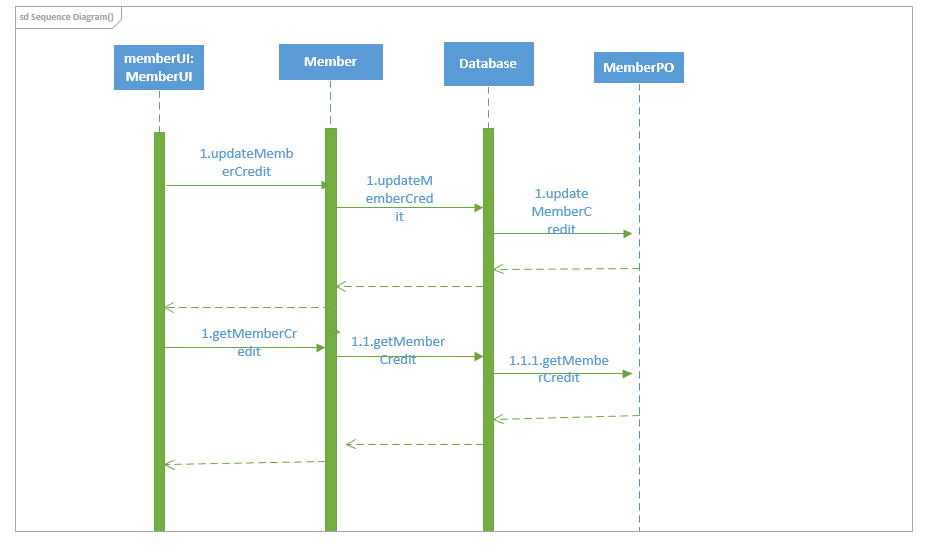
|  |  |
| --- | --- |
| 需要的服务（需接口） | |
| Data.dao.getOrder | 得到Order数据库的服务的引用 |
| Data.dao.getMember | 得到member数据库的服务的引用 |
| Data.dao.getCreditRecord | 得到CreditRecord数据库的服务的引用 |
| Data.dao.updateCreditRecord | 更新数据库中信用记录 |

（4）业务逻辑层的动态模型

下图表明了当用户输入用户的信息（包括姓名，地址，手机号等）会员业务逻辑处理的相关对象之间的协作

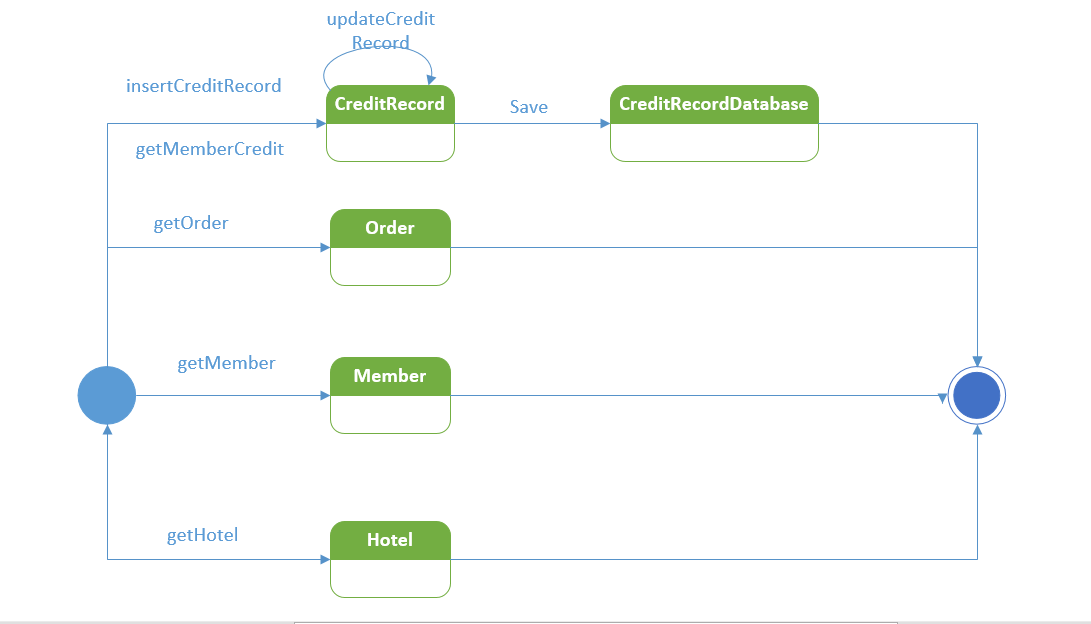


得到历史酒店，历史订单，会员信息的顺序图



获得历史订单，酒店的顺序图

如下图所示的状态图描述了Member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getMember方法被调用，Member进入Member状态，。UI也可以调用getOrder,getHotel分别进入Order，Hotel状态。UI可以通过insertCreditRecord进入CreditRecord。并可以在这个状态updateCreditRecord。最后通过save保存到Database中



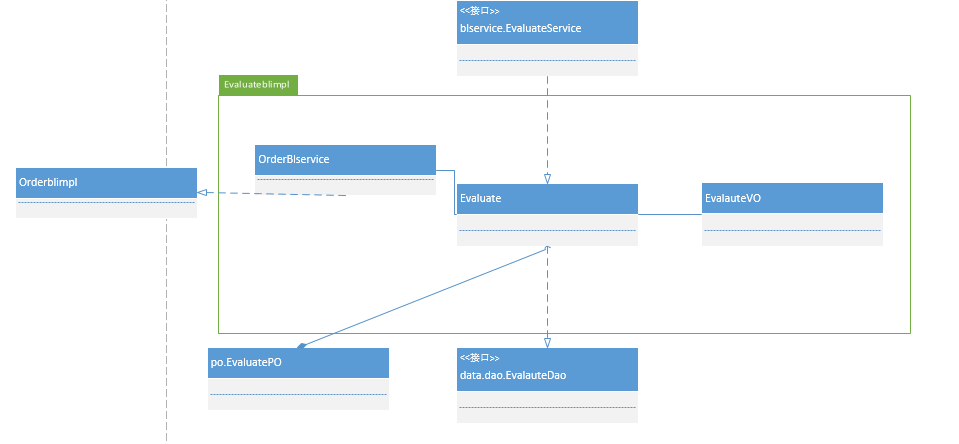
### 4.1.8 Evaluatebl模块

（1）模块概述

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

Evaluatebl模块的职责以及接口参见软件系统结构描述文档

根据体系结构的设计，我们将系统分为展示层（presentation），业务逻辑层（businesslogic），数据层（data）。每一层之间为了增加灵活性，我们会添加接口。例如展示层与业务逻辑层之间，我们添加bussinesslogicservice.evaluateblservice. EvaluateBlService接口。业务逻辑层和数据层之间，我们添加了dataservice. evaluatedataservice. EvaluateDataService接口。EvaluatePO 作为评价的历史记录以及持久化对象加入。



Evaluatebl模块各个类的职责如表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| Evaluateblimpl | 拥有评价的方法和信息 |

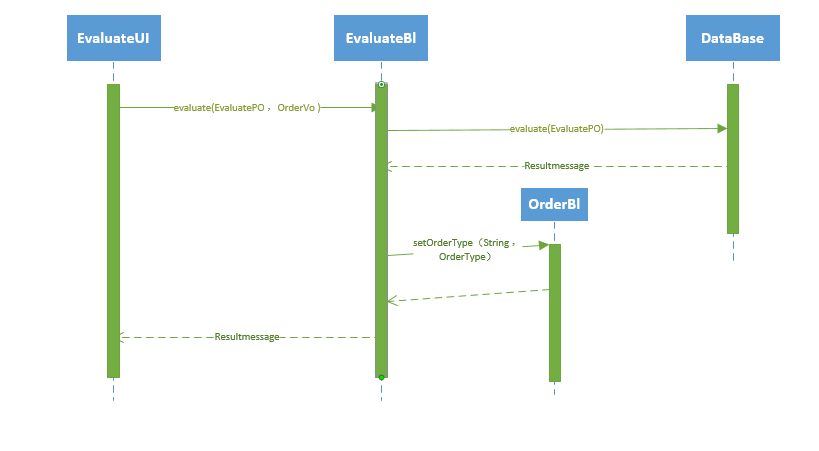
Evaluate的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Evaluateblimpl.evaluate | **语法** | **public ResultMessage evaluate(EvaluteVO vo,OrderVo orderVo)** |
| **前置条件** | **Order已经被执行且没有被评价过，score小于5** |
| **后置条件** | **进行评价，成功返回true，失败返回原因。** |
| Evaluateblimpl.getScore | **语法** | **public double getSore(String hotelName )** |
|  | **前置条件** | **酒店存在** |
|  | **后置条件** | **返回酒店的评分** |
| Evaluateblimpl.getComments | **语法** | **public ArrayList<String> evaluate(String hotelName)** |
|  | **前置条件** | **酒店存在** |
|  | **后置条件** | **返回所有的酒店评价信息** |

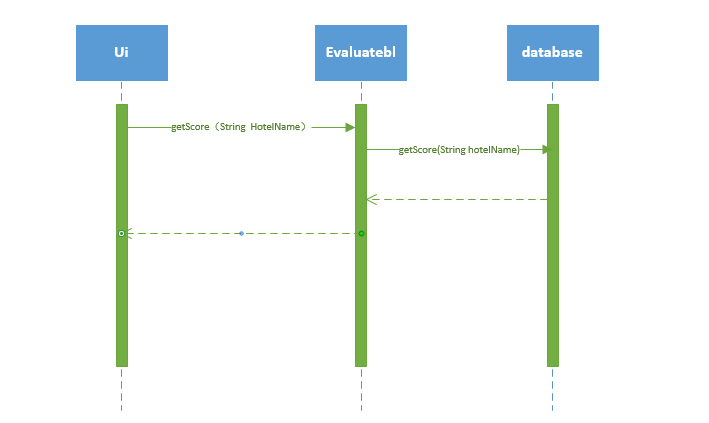
|  |  |
| --- | --- |
| 需要的接口 | |
| 服务名 | **服务** |
| EvaluateDao.addEvaluate(EvalautePO po) | **向数据库添加评价信息** |
| EvaluateDao.getEvalaution(String hotelName) | **根据酒店名得到酒店所有的评价** |
| EvalauteDao.getScore(String hotelName) | **根据酒店名得到酒店的评分** |
| OrderService.setType(int orderId,OrderType.type) | **将订单设为已评价的状态** |

（4）业务逻辑层的动态模型

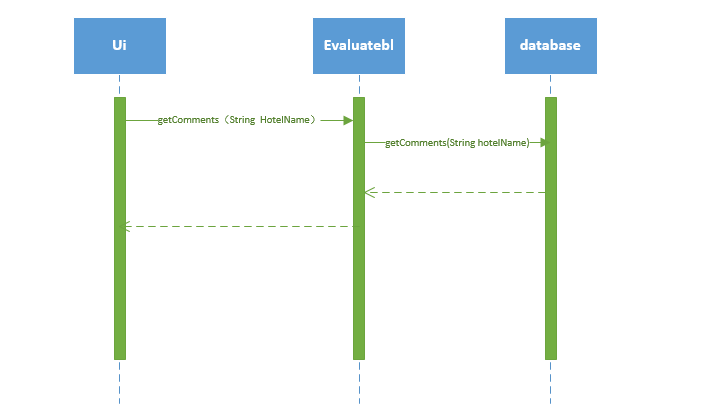
下图表示评价过程中，保存评价以及设置订单类型时相关对象的协作



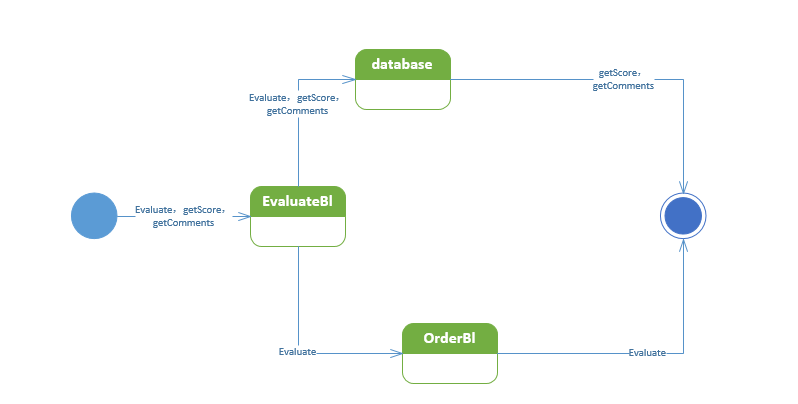
下图表示得到评分时，相关对象的协作



下图表示得到评价时，相关类的协作



下图所示的状态图描述了Evaluate对象的生存期间的状态序列。随着evaluate，getScore，getComments方法被调用，进入Evaluatebl状态，然后进入数据库查询，随后evaluate会进入orderbl进行修改订单类型，其他的直接结束。



# 5.依赖视角

下图是服务器端与客户端各自包之间的依赖关系



