酒店预订系统

HRS(Hotel Reservation System)

软件详细设计描述文档

V1.0 正式版

冰与火小组

2016-10-29

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **修改人员** | **日期** | **变更原因** | **版本号** |
| 王洪凯 | 2016-10-27 | 最初草稿 | V1.0 草稿 |
| 王洪凯 | 2016-10-29 | 评审后的正式版 | V1.0 正式版 |

**目录**

[1、 引言 4](#_Toc4898)

[1.1 编制目的 4](#_Toc26964)

[1.2 词汇表 4](#_Toc3134)

[1.3 参考资料 4](#_Toc22446)

[2、 产品概述 4](#_Toc10954)

[3、 体系结构设计概述 4](#_Toc23390)

[4、 结构视角 4](#_Toc11607)

[4.1 业务逻辑层的分解 4](#_Toc10240)

[4.1.1 userbl模块 5](#_Toc21298)

[4.1.2 Browsebl模块 8](#_Toc31047)

[4.1.3 Orderbl模块 14](#_Toc25721)

[4.1.4 Hotelbl模块 20](#_Toc19776)

[4.1.5 Memberbl模块 25](#_Toc3012)

[4.1.6 Commentbl模块 30](#_Toc15840)

[4.1.7 Strategy模块 33](#_Toc5638)

[4.1.8 Credit模块 39](#_Toc23507)

[5、 开发视角 43](#_Toc4764)

# 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，达到指导后续软件构造的目的，同时实现测试人员及用户的沟通。

本报告面向开发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 | 词汇含义 | 备注 |
| HRS | 酒店预订系统 |  |
|  |  |  |

## 1.3 参考资料

连锁商店管理系统（MSCS）软件详细设计描述文档

# 产品概述

参考酒店预订系统用例文档和酒店预订系统软件需求规格说明文档中对产品的概括描述。

# 体系结构设计概述

参考酒店预订系统概要设计文档中对体系结构设计的概述

# 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中的开发包图。

### 4.1.1 userbl模块

1. 模块概述

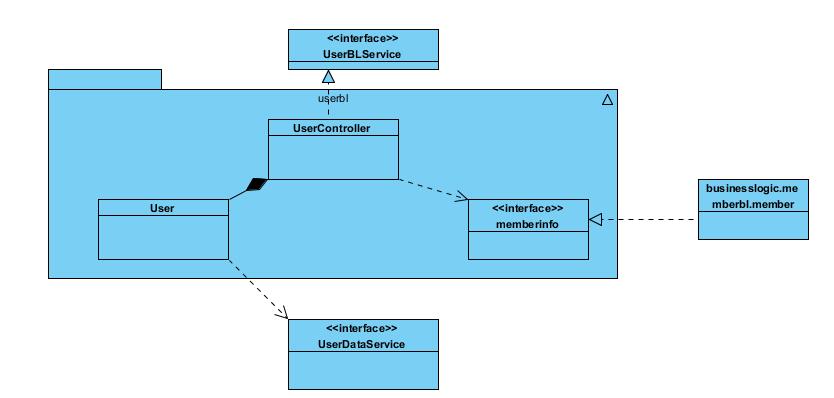
userbl模块承担的需求参见需求规格说明文档中登录/注册相关内容及相关非功能需求

userbl模块承担的职责和接口见体系设计文档中的userbl相关的职责和接口

1. 整体结构

依先前体系设计逻辑模型和相关文档的说明，为了增加灵活性降低耦合，增加可复用性，在界面层和业务逻辑层增加接口层，在数据层和业务逻辑层也增加接口层，通过接口进行调用，其中，使用usercontroller将模块里的业务逻辑和逻辑操作相分开，将登陆登出及注册委托给user，同时在登陆成功后调用相关member的接口进行初始化。

userbl 模块的设计如下图所示



userbl模块各个类的职责由下表所示

|  |  |
| --- | --- |
| 类 | 职责 |
| UserController | 负责Userbl模块的逻辑操作，实现对应于登陆登出和注册所需要的服务 |
| User | 用户登录相关的领域模型，拥有用户数据的用户名和密码，可以帮助完成登录登出相关的操作 |

1. 模块内部类的接口规范

UserController、User和Register的接口规范如下表所示

UserController:

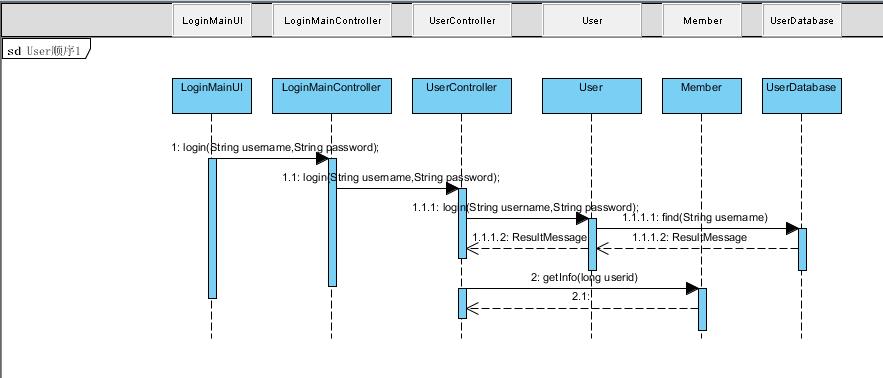
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| UserController.login | 语法 | public ResultMessage login(String username,String password); |
| 前置条件 | 用户准备登陆且输入符合规范 |
| 后置条件 | 调用User的login方法 |
| UserController.register | 语法 | public ResultMessage logout(String username,String password); |
| 前置条件 | 用户选择登陆且输入符合规范 |
| 后置条件 | 调用User的register方法 |
| UserController.logout | 语法 | public ResultMessage logout(long userid); |
| 前置条件 | 用户已登录 |
| 后置条件 | 调用User的logout方法 |
| UserController.getInfo | 语法 | public MemberVO getInfo(long userid); |
| 前置条件 | 一个用户已登录，且登录成功并拿到相应userid |
| 后置条件 | 调用Member的getInfo方法 |
| 需要的接口 | | |
| 服务名 | 服务 | |
| User.login(String username,String password); | 提供用户登录服务 | |
| User.register(String username,String password); | 加入一个用户对象 | |
| User.logout(long userid); | 登出一个用户 | |
| Member.getInfo(long userid); | 获取一个用户的信息 | |

User:

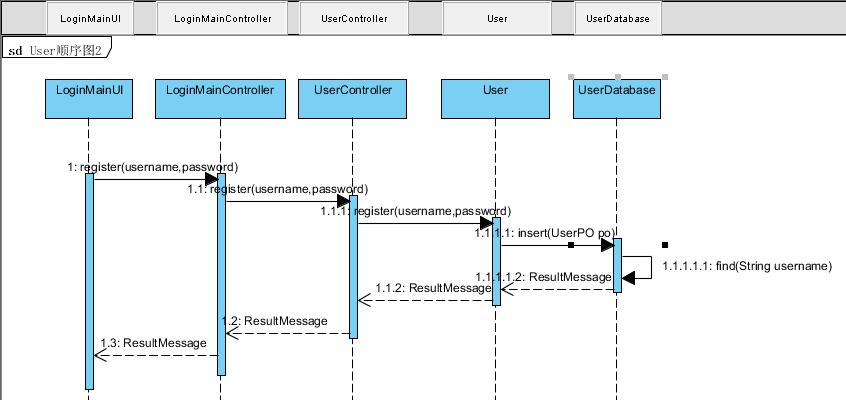
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| User.login | 语法 | public ResultMessage login(String username,String password); |
| 前置条件 | 已经开始一个登录行为 |
| 后置条件 | 根据传入的参数进行判断，并返回登录结果 |
| User.register | 语法 | public ResultMessage logout(String username,String password); |
| 前置条件 | 已经开始一个注册行为 |
| 后置条件 | 若该用户名没有被注册，新增一个User到数据库中 |
| User.logout | 语法 | public ResultMessage logout(long userid); |
| 前置条件 | 用户已登录 |
| 后置条件 | 数据库中对应的用户状态置为未登录 |
| 需要的接口 | | |
| 服务名 | 服务 | |
| UserDataService.insert(UserPO po); | 插入单一持久化对象 | |
| UserDataService.update(UserPO po); | 更新单一持久化对象 | |
| UserDataService.find(String username); | 根据username查找单一持久化对象 | |
| UserDataService.find(long userid); | 根据Id查找单一持久化对象 | |

1. 业务逻辑层的动态模型

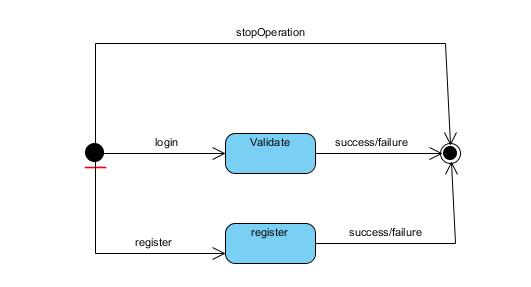
下图显示当用户输入登录的用户名和密码之后，User业务逻辑层处理的各个对象之间的协作：



下图为用户注册输入需要注册的用户名和密码之后，User业务逻辑层处理各个对象之间的协作：



下图描述了User对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作，当有用户执行登入或登出，sale会进行验证并返回结果，停止操作也可以直接进入终态：



1. 业务逻辑层设计原理

采用委托式控制器风格，由controller统一调度，将任务委托给对应的对象进行处理。

### 4.1.2 Browsebl模块

1. 模块概述

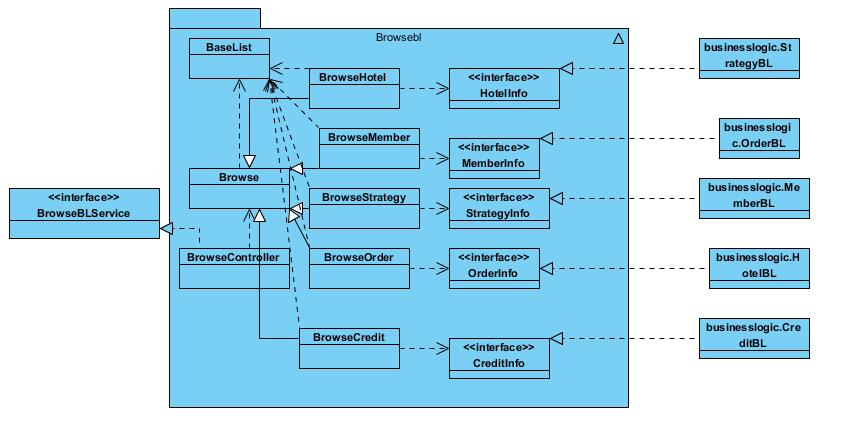
browseBl模块承担的需求参见需求规格说明文档中浏览订单信息（3个），浏览酒店信息，制定策略中隐含的浏览策略信息等相关隐含或非隐含的内容及相关非功能需求

browseBl模块承担的职责和接口见体系设计文档中的browseBl相关的职责和接口

1. 整体结构

具体总体概述见userbl模块的整体结构前半段描述，这里只介绍内部具体，使用BrowseController将模块里的业务逻辑和逻辑操作相分开，直接控制Browse（这里Browse是一个基类），以适配器的风格，将浏览订单信息委托给BrowseOrderList，将浏览策略信息委托给BrowseStrategyList,各具体类型的适配器分别调用其他模块接口完成自己的工作。具体的返回也有相应的基本列表包装！

browsebl 模块设计如下：



browsebl各个类的职责如下

|  |  |
| --- | --- |
| 类 | 职责 |
| BrowseController | 负责Browsebl模块的逻辑操作，实现对应于各类信息浏览所需要的服务 |
| Browse | 浏览信息相关的领域模型，是一个基类适配器，它能转化具体类型的接口 |
| BrowseHotel | 具体的适配Ataptee，适配对象，实现具体的浏览酒店信息服务 |
| BrowseMember | 具体的适配Ataptee，适配对象，实现具体的浏览会员信息服务 |
| BrowseStrategy | 具体的适配Ataptee，适配对象，实现具体的浏览策略信息服务 |
| BrowseOrder | 具体的适配Ataptee，适配对象，实现具体的浏览订单信息服务 |
| BrowseCredit | 具体的适配Ataptee，适配对象，实现具体的浏览信用信息服务 |

1. 模块内部类的接口规范

BrowseController、Browse和具体的Browse\*的接口规范如下表所示

BrowseController:

所有的对Browse的调用，类型会根据类型声明成具体的类型，而实际为相应的具体的adaptee的引用

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseController.getOrdersInfo | 语法 | Public List<OrderVO> getOrdersInfo(long userid,OrderType type); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 调用Browse的getData方法，参数为(Role role,long id,OrderType) |
| BrowseController.getHotelOrdersInfo | 语法 | Public List<OrderVO> getHotelOrdersInfo(long hotelid,OrderType type); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 调用Browse的getData方法，参数为(Role role,long id,OrderType) |
| BrowseController.getWEBOrdersInfo | 语法 | Public List<OrderVO> getWEBOrdersInfo(); |
| 前置条件 | 网站营销人员已登录 |
| 后置条件 | 调用Browse的getData方法，参数为(Role role,long id,OrderType) 其中id为-1，OrderType默认为异常 |
| BrowseController.getCreditInfo | 语法 | Public List<CreditVO> getCreditInfo(long userid); |
| 前置条件 | 客户已登录 |
| 后置条件 | 调用相应Browse的getData方法 参数为该id |
| BrowseController.getHotels | 语法 | Public List<HotelVO> getHotels(RuleVO vo); |
| 前置条件 | 客户已登录 |
| 后置条件 | 调用相应Browse的getData方法 参数为该RuleVO |
| BrowseController.getHotel | 语法 | Public HotelVO getHotel(long hotelid); |
| 前置条件 | 客户已登录 |
| 后置条件 | 调用相应Browse的getData方法 参数为该id |
| BrowseController.getStrategyInfo | 语法 | Public StrategyVO getStrategyInfo(int orderid); |
| 前置条件 | 用户选择一个订单 |
| 后置条件 | 调用相应Browse的getData方法 参数为该id |
| BrowseController.getMemberInfo | 语法 | Public MemberVO getMemberInfo(long memberid); |
| 前置条件 | 用户已登录 |
| 后置条件 | 调用相应Browse的getData方法 参数为该id |
| BrowseController.getBrowseInfo | 语法 | Public List<BrowseVO> getBrowseInfo(long userid); |
| 前置条件 | 客户已登录 |
| 后置条件 | 调用相应Browse的getData方法 参数为该id |
| BrowseController.clear | 语法 | Public void clear(long userid); |
| 前置条件 | 客户已登录 |
| 后置条件 | 调用Browse的clear方法 |
| 需要的服务(需接口) | | |
| Hotel.getHotels(RulePO po) | 得到基于rule的酒店信息列表 | |
| Hotel.getInfo(long Hotelid) | 得到hotel的详细信息 | |
| Credit.getInfo(long userID) | 得到user的信用信息 | |
| Order.getuserInfo(long userid,int method) | 得到user的对应method的对应类别的订单信息 | |
| Order.getHotelInfoloing hotelid,int method) | 得到hotel的对应method的对应类别的订单信息 | |
| Order.getWEBInfo() | 得到当天当前时间前所有的异常订单信息 | |
| Member.getInfo(long userid) | 得到对应的会员信息 | |
| Strategy.getInfo(long orderid) | 得到对应订单的策略信息 | |

BaseList 基列表类封装对应的基对象，包含大量对具体对象的集合，格式如下

BaseList<T> 其属性有 list<T>为引用（里面可以只为1个）

Browse类的接口规范如下：

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| Browse.getData | 语法 | Public BaseList<T> getData(Role role,long userid,OrderType type); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据其泛型T，jvm自己调用具体的adaptee的相应方法并返回 |
| Browse.getData | 语法 | Public BaseList<T> getData(long id); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 根据其泛型T，jvm自己调用具体的adaptee的相应方法并返回（返回一个VO即对应BaseList只有一个VO对象） |
| Browse.getData | 语法 | Public BaseList<T> getData(RuleVO vo); |
| 前置条件 | 网站营销人员已登录 |
| 后置条件 | 根据其泛型T，jvm自己调用具体的adaptee的相应方法并返回 |
| Browse.clear | 语法 | Public void clear(long userid); |
| 前置条件 | 客户已登录 |
| 后置条件 | 清空数据库中相应的用户的浏览记录 |
| 需要的服务(需接口) | | |
| DatabaseFactory.getBrowseDatabase | 得到Browse数据库服务的引用 | |
| BrowseDataService.insert(BrowsePO po) | 在Browse数据库中插入po（搜索记录） | |
| BrowseDateService.getInfo(userid) | 返回一个用户的浏览/搜索记录 | |
| BrowseDataService.clear(long userid) | 清空一个用户的浏览搜索记录 | |
| DatabaseFactory.getBrowseDatabase | 得到Browse数据库服务的引用 | |

BrowseOrder的类的接口规范如下（其继承基类Browse，类型为OrderVO）（只描述实现的接口，继承下未改动的查看父类）：

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseOrder.getData | 语法 | Public BaseList<OrderVO> getData(Role role,long userid,OrderType type); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | Role，调用相应的Order方法，获得相应列表后包装并返回 |
| 需要的服务(需接口) | | |
| Order.getuserInfo(long userid,int method) | 得到user的对应method的对应类别的订单信息 | |
| Order.getHotelInfoloing hotelid,int method) | 得到hotel的对应method的对应类别的订单信息 | |
| Order.getWEBInfo() | 得到当天当前时间前所有的异常订单信息 | |

BrowseCredit的类的接口规范如下（其继承基类Browse，类型为CreditVO）（只描述实现的接口，继承下未改动的查看父类）：

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseCredit.getData | 语法 | Public BaseList<CreditVO> getData(long userid); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据传入的userid调用credit的getInfo方法 |
| 需要的服务(需接口) | | |
| Credit.getInfo(long userID) | 得到user的信用信息 | |

BrowseHotel的类的接口规范如下（其继承基类Browse，类型为HotelVO）（只描述实现的接口，继承下未改动的查看父类）：

|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseHotel.getData | 语法 | Public BaseList<HotelVO> getData(long hotelid); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据传入的hotelid,调用hotel 的getinfo方法，包装信息后返回包装的信息 |
| 需要的服务(需接口) | | |
| Hotel.getHotels(RulePO po) | 得到基于rule的酒店信息列表 | |
| Hotel.getInfo(long Hotelid) | 得到hotel的详细信息 | |

BrowseMember的类的接口规范如下（其继承基类Browse，类型为MemberVO）（只描述实现的接口，继承下未改动的查看父类）：

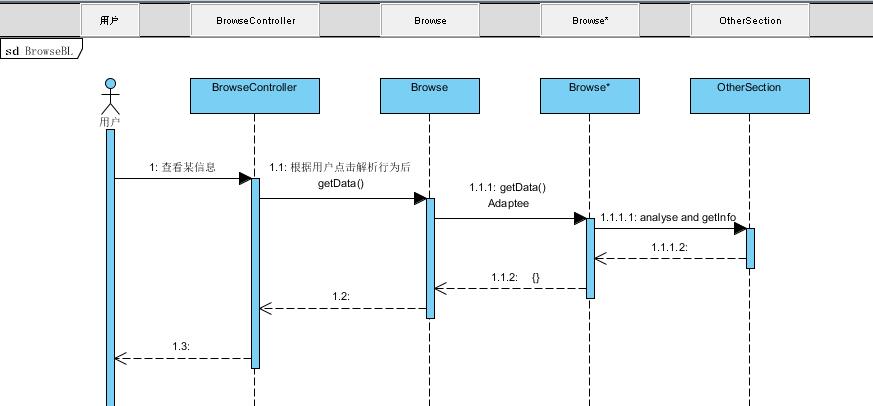
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseMember.getData | 语法 | Public BaseList<MemberVO> getData(long userid); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据传入的userid调用member的方法  长度为13，则为getInfo  长度为1，则为getList |
| 需要的服务(需接口) | | |
| Member.getInfo(long userid) | 得到对应的会员信息 | |
| Member.getList(int type); | 得到对应的用户列表 | |

BrowseStrategy的类的接口规范如下（其继承基类Browse，类型为StrategyVO）（只描述实现的接口，继承下未改动的查看父类）：

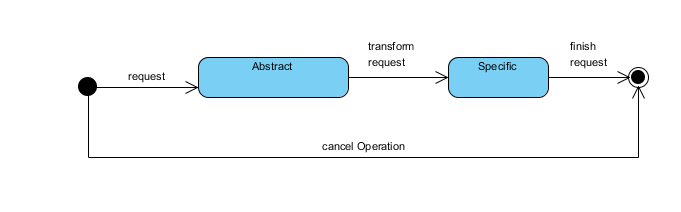
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| BrowseStrategy.getData | 语法 | Public BaseList<StrategyVO> getData(long orderid); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据传入的orderid，调用Strategy的getInfo方法 |
| 需要的服务(需接口) | | |
| Strategy.getInfo(long orderid) | 得到对应订单的策略信息 | |

(4)业务逻辑层的动态模型

下图显示当用户请求查看具体的信息时，Browsebl模块各个类之间的协作时序图



下图描述了Browse对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作。当客户发起了一个请求，Browse处于抽象状态，控制器解析后调用相应方法（并声明了具体类型），Browse变为具体状态并处理请求，处理完后结束



（5）业务逻辑层设计原理

采用适配器模式风格，由controller统一调度，将任务委托给Browse，Browse对类型进行转换后处理结束。

### 4.1.3 Orderbl模块

1. 模块描述

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模块承担的职责和接口见体系设计文档中的orderbl相关的职责和接口

1. 整体结构

依先前体系设计逻辑模型和相关文档的说明，为了增加灵活性降低耦合，增加可复用性，在界面层和业务逻辑层增加接口层，在数据层和业务逻辑层也增加接口层，通过接口进行调用。

将系统分为展示层、业务逻辑层和数据层。每层之间添加接口。

在展示层和业务逻辑层之间，我们添加businesslogicservice.OrderBLservice.OrderBLservice接口。

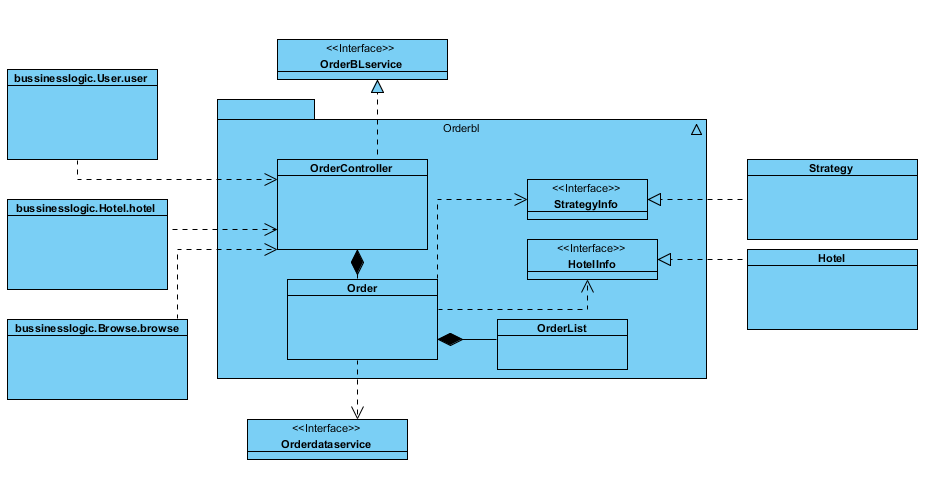
在业务逻辑层和数据层之间，我们添加dataservice.OrderDataservice接口。

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会将对销售的业务逻辑处理委托给Order对象。

OrderList是Order的容器类。

OrderVO是作为订单的持久化对象被添加到设计模型中去的。

Orderbl 模块设计如下：



Orderbl模块各个类的职责入下表所示：

|  |  |
| --- | --- |
| 模块 | 职责 |
| OrderController | 负责Orderbl模块的逻辑操作，实现和订单相关界面所需要的服务 |
| Order | 订单的领域模型对象，拥有订单数据的状态、开始时间、结束时间、入住人数、有无儿童等，可以帮助完成订单相关的操作 |

1. 模块内部类的接口规范

OrderController和Order的接口规范如下表所示

OrderController：

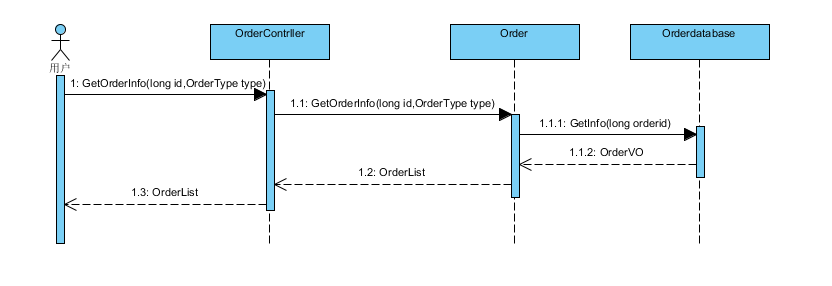
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| OrderController.getuserInfo | 语法 | Public OrderList getuserInfo(long userid,OrderType type); |
| 前置条件 | 已创建一个Order领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Order领域对象的getuserInfo方法 |
| OrderController.getHotelInfo | 语法 | Public OrderList getHotelInfo(long hotelid,OrderType type); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的getHotelInfo方法 |
| OrderController.getWEBInfo | 语法 | Public OrderList getWEBInfo(); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的getWEBInfo方法 |
| OrderController.create | 语法 | Public ResultMessage create(OrderVO vo); |
| 前置条件 | User对象发出create请求 |
| 后置条件 | 调用Order领域对象的create方法 |
| OrderController.Abnormal | 语法 | Public void abnormal(OrderVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的Abnormal方法 |
| OrderController.userCancel | 语法 | Public ResultMessage userCancel(OrderVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的userCancel方法 |
| OrderController.execute | 语法 | Public ResultMessage execute(OrderVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的execute方法 |
| OrderController.WEBCancel | 语法 | Public ResultMessage WEBCancel(OrderVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的WEBCancel方法 |
| OrderController. isUsed | 语法 | Public boolean isUsed (StrategyVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的isUsed方法 |
| OrderController.getTotal | 语法 | Public double getTotal(OrderVO vo); |
| 前置条件 | 已创建一个Order领域对象 |
| 后置条件 | 调用Order领域对象的getTotal方法 |
| 需要的服务(需接口) | | |
| Order.getuserInfo(long userid,OrderType type) | 用户获取一个对应类型的订单信息 | |
| Order.getHotelInfo(long hotelid,OrderType type) | 酒店工作人员获取一个对应类型的订单信息 | |
| Order.getWEBInfo() | 网站营销人员获取一个对应类型的订单信息 | |
| Order.create(OrderVO vo) | 创建一个订单对象 | |
| Order.Abnormal(OrderVO vo) | 改变订单状态为异常 | |
| Order.userCancel(OrderVO vo) | 用户删除一个订单 | |
| Order.execute(OrderVO vo) | 改变订单状态为执行 | |
| Order.WEBCancel(OrderVO vo) | 网站营销人员删除一个订单 | |
| Order.getTotal(OrderVO vo) | 获得订单总价 | |

Order：

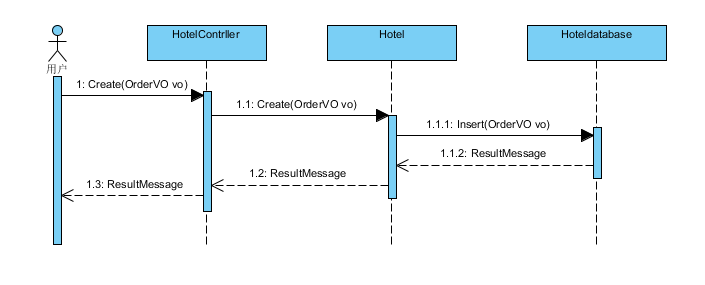
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| Order.getuserInfo | 语法 | Public OrderList getuserInfo(long userid,OrderType type); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 根据对应的type 查找对应类型的对应用户订单信息 |
| Order.getHotelInfo | 语法 | Public OrderList getHotelInfo(long hotelid,OrderType type); |
| 前置条件 | 酒店管理人员已登录 |
| 后置条件 | 根据对应的type 查找对应类型的对应用户订单信息 |
| Order.getWEBInfo | 语法 | Public OrderList > getWEBInfo(); |
| 前置条件 | 网站营销人员已登录 |
| 后置条件 | 根据对应的TYPE 查找对应类型的当天当时间异常订单信息 |
| Order.create | 语法 | Public ResultMessage create(OrderVO vo); |
| 前置条件 | 客户已登录 |
| 后置条件 | 创建订单并存储 |
| Order.Abnormal | 语法 | Public void abnormal(OrderVO vo); |
| 前置条件 | 订单由未执行变为异常 |
| 后置条件 | 改变订单状态并存储 |
| Order.userCancel | 语法 | Public ResultMessage userCancel(OrderVO vo); |
| 前置条件 | 用户已登录 |
| 后置条件 | 所orderid所对应订单为未执行订单，撤销该订单 |
| Order.execute | 语法 | Public ResultMessage execute(OrderVO vo); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 根据对应订单，判断订单状态，执行或补登记执行订单，并作出信用值的相应改变 |
| Order.WEBCancel | 语法 | Public ResultMessage WEBCancel(OrderVO vo); |
| 前置条件 | 网站营销人员已登录且线下已判断订单可撤销 |
| 后置条件 | 根据对应的订单号，撤销订单，并返回状态 |
| Order.isUsed | 语法 | Public boolean isUsed (StrategyVO vo); |
| 前置条件 | 用户要求计算订单价格 |
| 后置条件 | 判断是否符合促销策略条件，并返回 |
| Order.getTotal | 语法 | Public double getTotal(OrderVO vo); |
| 前置条件 | 无 |
| 后置条件 | 计算订单总价，并返回 |
| 需要的服务(需接口) | | |
| DatabaseFactory.getOrderDatabase | 得到Order数据库服务的引用 | |
| OrderDataService.insert(OrderPO po) | 在Order数据库中插入po | |
| OrderDataService.changeStatus(OrderPO po); | 改变订单的状态 | |
| OrderDataService.getInfo(long orderid); | 获取订单的信息 | |
| Hotel.getRooms(long hotelid) | 得到该hotel的房间信息 | |
| Hotel.setRoomInfo(RoomVO vo) | 锁定该酒店的相应的房间 | |
| Credit.add/decrese(long userid,int value); | 增加/减少该user的信用值信息 | |
| Member.getInfo(userid) | 得到对应的会员信息 | |
| Strategy.getInfo(orderid) | 得到对应订单的策略信息 | |

1. 业务逻辑层的动态模型

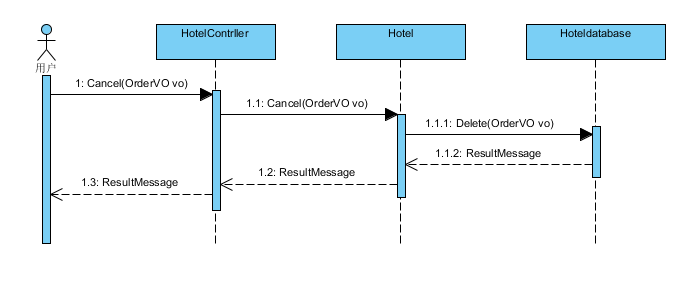
下图显示当客户或酒店工作人员或网站营销人员请求查看订单信息时，Orderbl模块各个类之间的协作时序图：



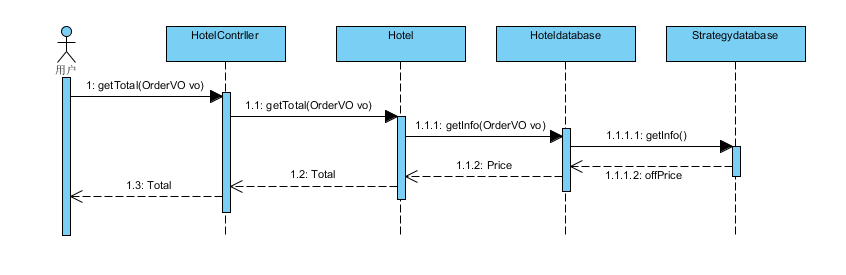
下图显示当客户要求创建订单时，Orderbl模块各个类之间的协作时序图：



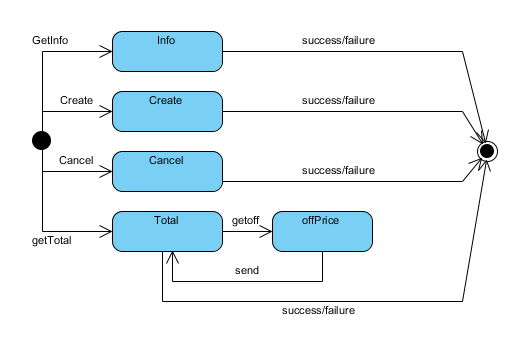
下图显示当客户或网站营销人员要求撤销订单时，Orderbl模块各个类之间的协作时序图：



下图显示当客户要求计算订单总价时，Orderbl模块各个类之间的协作时序图：



下图描述了Hotel对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作。随着Getinfo、SetInfo、Cancel或getTotal方法被UI调用，Order处理请求并结束。



1. 业务逻辑层的设计原理

采用控制器模式风格，由controller统一调度，将任务委托给Order，Order处理后结束。

### 4.1.4 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模块承担的职责和接口见体系设计文档中的hotelbl相关的职责和接口

（2）整体结构

依先前体系设计逻辑模型和相关文档的说明，为了增加灵活性降低耦合，增加可复用性，在界面层和业务逻辑层增加接口层，在数据层和业务逻辑层也增加接口层，通过接口进行调用。

将系统分为展示层、业务逻辑层和数据层。每层之间添加接口。

在展示层和业务逻辑层之间，我们添加businesslogicservice.HotelBLservice.HotelBLservice接口。

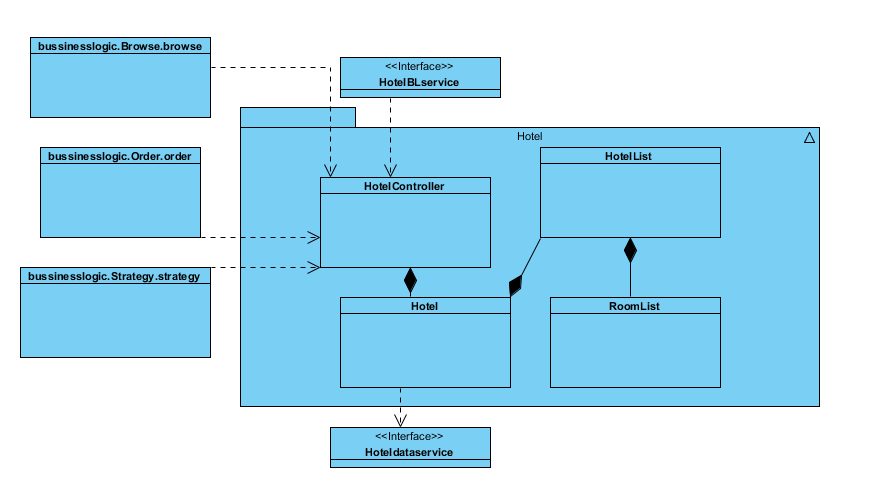
在业务逻辑层和数据层之间，我们添加dataservice.HotelDataservice接口。

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。

HotelList是Hotel的容器类。

HotelList可以委托给RoomList。

HotelVO是作为订单的持久化对象被添加到设计模型中去的。



Hotelbl模块各个类的职责入下表所示

|  |  |
| --- | --- |
| 模块 | 职责 |
| HotelController | 负责Hotelbl模块的逻辑操作，实现和酒店相关界面所需要的服务 |
| Hotel | 酒店的领域模型对象，拥有酒店数据的酒店名称、商圈、地址、简介、星级、设施服务等，可以帮助完成酒店相关的操作 |

（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：

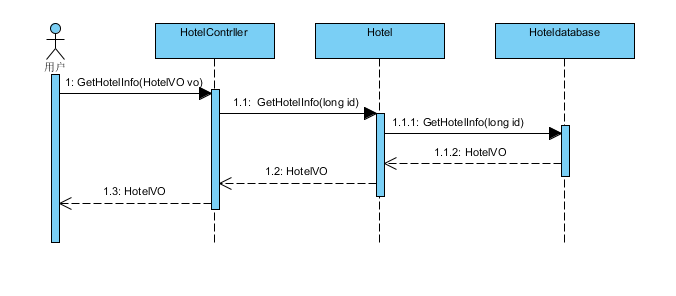
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| HotelController.GetHotelInfo | 语法 | Public HotelVO GetHotelInfo(long id); |
| 前置条件 | 已创建一个Hotel领域对象 |
| 后置条件 | 调用Hotel领域对象的GetHotelInfo方法 |
| Hotel.GetRoomInfo | 语法 | Public HotelVO GetRoomInfo(long id); |
| 前置条件 | 已创建一个Room领域对象 |
| 后置条件 | 调用Hotel领域对象的GetRoomInfo方法 |
| HotelController.SetHotelInfo | 语法 | Public ResultMessage SetHotelInfo(HotelVO vo); |
| 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SetHotelInfo方法 |
| HotelController.SetRoomInfo | 语法 | Public ResultMessage SetRoomInfo(long id, RoomVO vo); |
| 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Hotel领域对象的SetRoomInfo方法 |
| HotelController.AddHotel | 语法 | Public ResultMessage AddHotel(HotelVO vo); |
| 前置条件 | WebManager对象发出add请求 |
| 后置条件 | 调用Hotel领域对象的AddHotel方法 |
| HotelController.AddRoom | 语法 | Public ResultMessage AddRoom(RoomVO vo); |
| 前置条件 | Hotelworker对象发出add请求 |
| 后置条件 | 调用Hotel领域对象的AddRoom方法 |
| Hotel.DeleteHotel | 语法 | Public ResultMessage DeleteHotel(HotelVO vo); |
| 前置条件 | WebManager对象发出delete请求 |
| 后置条件 | 调用Hotel领域对象的DeleteHotel方法 |
| Hotel.DeleteRoom | 语法 | Public ResultMessage DeleteRoom(RoomVO vo); |
| 前置条件 | Hotelworker对象发出delete请求 |
| 后置条件 | 调用Hotel领域对象的DeleteRoom方法 |
| 需要的服务(需接口) | | |
| Hotel.GetHotelInfo(long id) | 获取一个酒店对象的信息 | |
| Hotel.GetRoomInfo(long id) | 获取一个房间对象的信息 | |
| Hotel.SetHotelInfo(HotelVO vo) | 修改一个酒店对象的信息 | |
| Hotel.SetRoomInfo(long id, RoomVO vo) | 修改一个房间对象的信息 | |
| Hotel.AddHotel(HotelVO vo) | 创建一个酒店对象 | |
| Hotel.AddRoom(RoomVO vo) | 创建一个房间对象 | |
| DeleteHotel(HotelVO vo) | 删除一个酒店对象 | |
| DeleteRoom(RoomVO vo) | 删除一个房间对象 | |

Hotel：

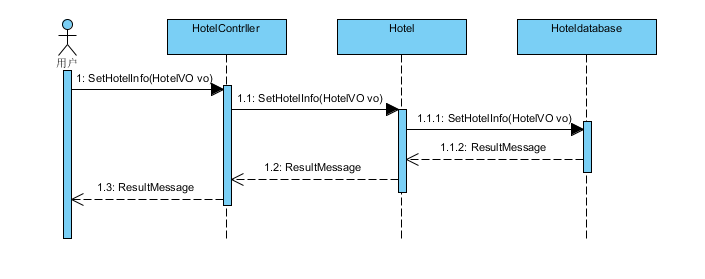
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| Hotel.GetHotelInfo | 语法 | Public HotelVO GetHotelInfo(long id); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 无 |
| Hotel.GetRoomInfo | 语法 | Public HotelVO GetRoomInfo(long id); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 无 |
| Hotel.SetHotelInfo | 语法 | Public ResultMessage SetHotelInfo(HotelVO vo); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 更新酒店信息 |
| Hotel.SetRoomInfo | 语法 | Public ResultMessage SetRoomInfo(long id, RoomVO vo); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 更新房间信息 |
| Hotel.AddHotel | 语法 | Public ResultMessage AddHotel(HotelVO vo); |
| 前置条件 | 网站管理人员已登录 |
| 后置条件 | 生成新的酒店信息 |
| Hotel.AddRoom | 语法 | Public ResultMessage AddRoom(RoomVO vo); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 生成新的房间信息 |
| Hotel.DeleteHotel | 语法 | Public ResultMessage DeleteHotel(HotelVO vo); |
| 前置条件 | 网站管理人员已登录 |
| 后置条件 | 删除酒店信息 |
| Hotel.DeleteRoom | 语法 | Public ResultMessage DeleteRoom(RoomVO vo); |
| 前置条件 | 酒店工作人员已登录 |
| 后置条件 | 删除房间信息 |
| 需要的服务(需接口) | | |
| DatabaseFactory.getHotelDatabase | 得到Hotel数据库服务的引用 | |
| HotelDataService.insertHotel(HotelPO po) | 在数据库中插入vo对象 | |
| HotelDataService.getHotelInfo(HotelID id) | 得到Hotel信息 | |
| HotelDataService.setHotelInfo(HotelPO po) | 更改Hotel信息 | |
| HotelDataService.insertRoom(RoomPO po) | 在数据库中插入vo对象 | |
| HotelDataService.getRoomInfo(HotelID id) | 得到Room信息 | |
| HotelDataService.setRoomInfo(RoomPO po) | 更改Room信息 | |

（4）业务逻辑层的动态模型

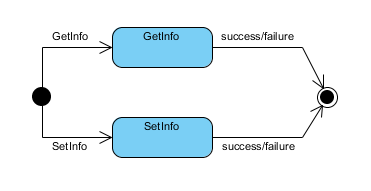
下图显示当酒店工作人员请求查看酒店信息时，Hotelbl模块各个类之间的协作时序图：



下图显示当酒店工作人员请求修改酒店信息时，Hotelbl模块各个类之间的协作时序图：



下图描述了Hotel对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作。随着Getinfo或SetInfo方法被UI调用，Hotel处理请求并结束。



（5）业务逻辑层的设计原理

采用控制器模式风格，由controller统一调度，将任务委托给Hotel，Hotel处理后结束。

### 4.1.5 Memberbl模块

(1)模块概述

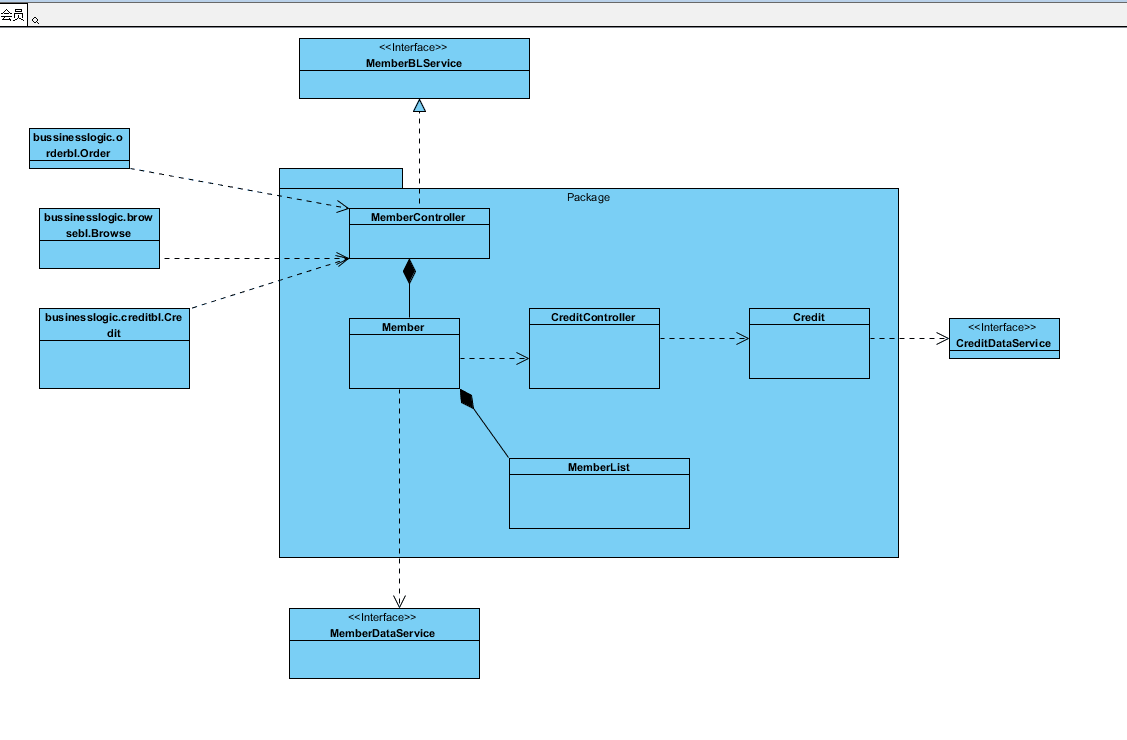
memberbl模块承担的需求参见需求规格说明文档中注册会员相关内容及相关非功能需求

memberbl模块承担的职责和接口见体系设计文档中的memberbl相关的职责和接口

(2)整体结构

依先前体系设计逻辑模型和相关文档的说明，为了增加灵活性降低耦合，增加可复用性，在界面层和业务逻辑层增加接口层，在数据层和业务逻辑层也增加接口层，通过接口进行调用，其中，使用membercontroller将模块里的业务逻辑和逻辑操作相分开，将注册删除及获取会员信息委托给member.

memberbl 模块的设计如下图所示



memberbl模块各个类的职责由下表所示

|  |  |
| --- | --- |
| 类 | 职责 |
| MemberController | 负责实现有关会员权利的服务 |
| Member | 会员的领域模型对象，拥有一个会员的各种属性（包括信用值、ID、联系方式、企业名称、生日、会员标记）以及注册VIP、获得会员信息、更改会员信息的方法，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

MemberController、Member的接口规范如下表所示

MemberController:

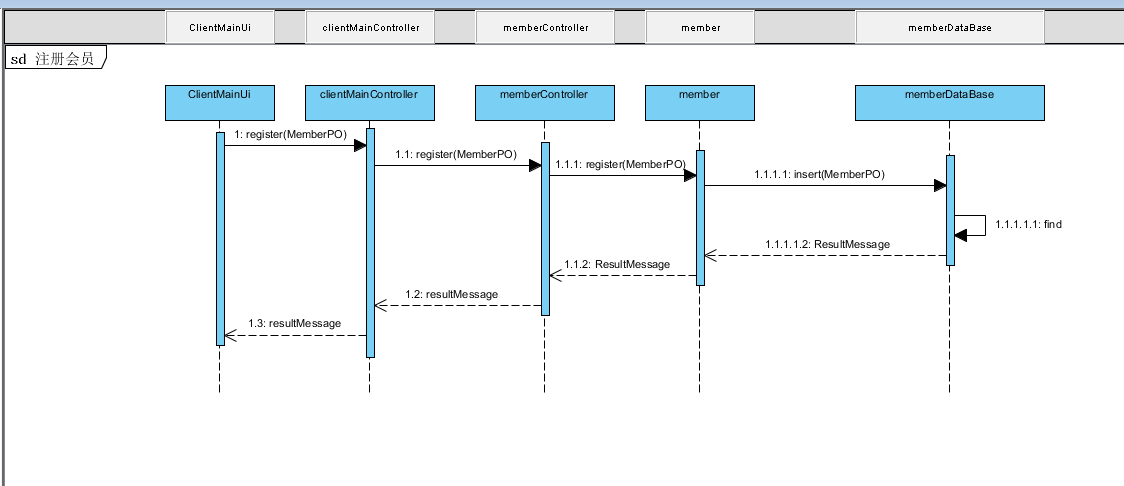
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| MemberController.register | 语法 | Public ResultMessage register(VIPVO vo); |
| 前置条件 | 已创建一个member领域对象，并且输入符合输入规则 |
| 后置条件 | 调用member领域对象的register方法 |
| MemberController.getinfo | 语法 | Public MemberPO getInfo(long Userid); |
| 前置条件 | 已创建一个member领域对象 |
| 后置条件 | 调用member领域对象的getinfo方法 |
| MemberController.cancel | 语法 | Public ResultMessage cancel(long Userid); |
| 前置条件 | 已创建一个member领域对象 |
| 后置条件 | 调用member领域对象的cancel方法 |
| MemberController.changeinfo | 语法 | Public ResultMessage changeInfo(MemberVO) |
| 前置条件 | 已创建一个member领域对象，并且输入符合规则 |
| 后置对象 | 调用member领域对象的changeInfo方法 |
| MemberController.manageInfo | 语法 | Public memberlist manageInfo() |
| 前置条件 | 已创建一个member领域对象 |
| 后置条件 | 调用member领域对象的manageInfo方法 |
| MemberController.delete | 语法 | Public ResultMessage delete(long userid) |
| 前置条件 | 已创建一个member领域对象 |
| 后置条件 | 调用member领域对象的delete方法 |
| 需要的接口 | | |
| 服务名 | 服务 | |
| Member.register(String phonenumber,String companyname,Time time) | 创建一个会员VIP对象 | |
| Member.getinfo(long Userid) | 返回一个会员的详细信息（包括会员ID、联系方式、用户名、企业名称、生日、信用值） | |
| Member.cancel(long Userid) | 删除一个会员对象 | |
| Member.changeInfo(MmeberVO) | 更改会员的基本信息 | |
| Member.manageInfo() | 返回所有用户信息 | |
| Member.delete(long userid) | 在用户管理时删除一个用户信息 | |

Member:

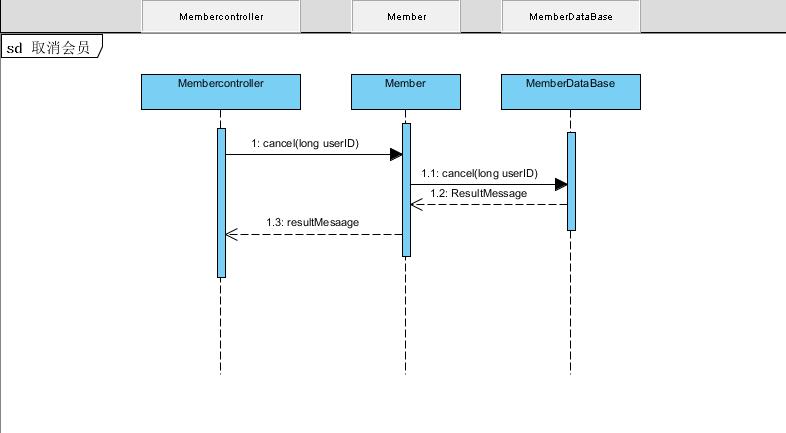
|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| Member.register | 语法 | Public ResultMessage register(VIPVO vo); |
| 前置条件 | 启动一个会员注册回合 |
| 后置条件 | 在一个会员注册回合中，增加一个会员信息 |
| Member.getinfo | 语法 | Public Memberinfo getInfo(long Userid); |
| 前置条件 | 已经完成会员注册 |
| 后置条件 | 返回根据Id相匹配的会员信息 |
| Member.cancel | 语法 | Public ResultMessage cancel(long Userid); |
| 前置条件 | 已经完成会员注册且当前信用值不足标准信用值 |
| 后置条件 | 删除与id相匹配的会员信息 |
| Member.changeInfo | 语法 | Public ResultMessage changeInfo(MemberVO) |
| 前置条件 | 启动一个会员修改信息回合 |
| 后置条件 | 修改会员的信息 |
| Member.manageInfo | 语法 | Public Memberlist manageInfo() |
| 前置条件 | 启动一个用户管理回合 |
| 后置条件 | 查看数据库中的会员信息 |
| Member.delete | 语法 | Public ResultMessage (long userid) |
| 前置条件 | 启动一个用户管理回合 |
| 后置条件 | 删除数据库中一个用户的信息 |
| 需要的接口 | | |
| 服务名 | 服务 | |
| MemberDataService.insert(MemberPO po) | 插入单一持久化对象 | |
| MemberDataService.getinfo(long Userid) | 返回一个单一持久化对象的信息 | |
| MemberDataService.cancel(long Userid) | 删除一个会员对象/用户对象 | |
| MemverDataService.Changeinfo(MemberPO po) | 更新单一持久化对象 | |
| DatabaseFactory.getMemberDatabase | 得到Member数据库服务的引用 | |

(4)业务逻辑层的动态模型

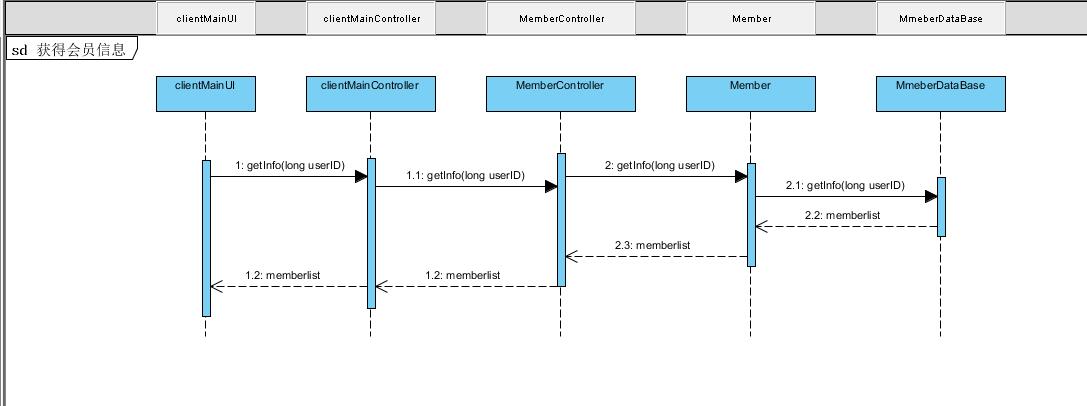
下图显示当用户请求注册会员后，Member业务逻辑层处理的各个对象之间的协作：



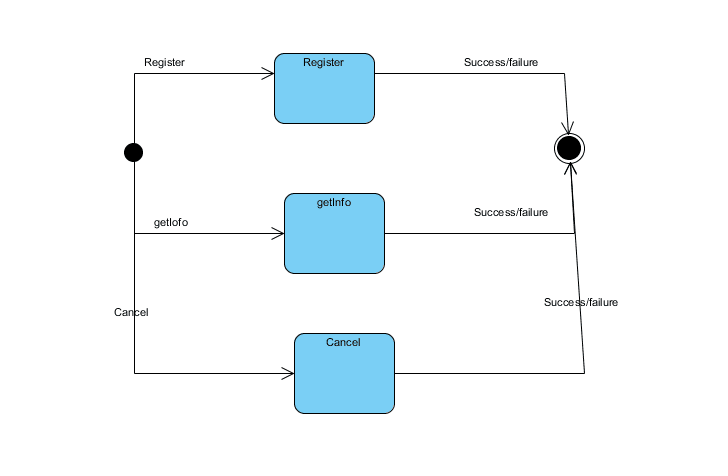
下图为用户信用值不足之后，Member业务逻辑层处理各个对象之间的协作：



下图为用户请求获取会员详细信息之后，Member业务逻辑层处理各个对象之间的协作：



下图描述了Mmeber对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作，当有用户执行注册或查看个人详细信息，状态会进行验证并返回结果：



(5)业务逻辑层设计原理

采用委托式控制器风格，由controller统一调度，将任务委托给对应的对象进行处理。

### 4.1.6 Commentbl模块

1. 模块概述

commentBl模块承担的需求参见需求规格说明文档中评价等相关隐含或非隐含的内容及相关非功能需求

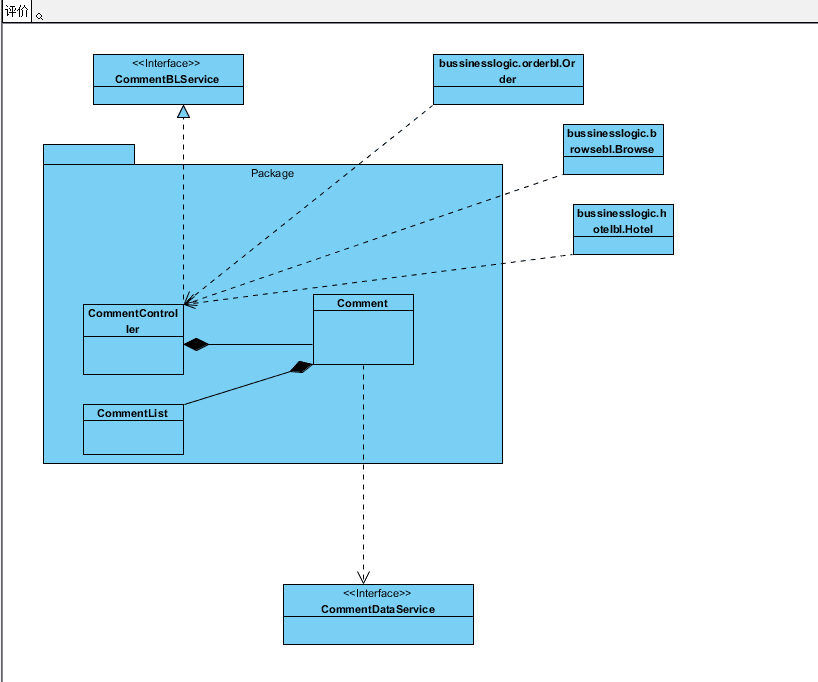
commentBl模块承担的职责和接口见体系设计文档中的commentBl相关的职责和接口

1. 整体结构

依先前体系设计逻辑模型和相关文档的说明，为了增加灵活性降低耦合，增加可复用性，在界面层和业务逻辑层增加接口层，在数据层和业务逻辑层也增加接口层，通过接口进行调用，其中，使用commentcontroller将模块里的业务逻辑和逻辑操作相分开，将评价及获取评价委托给comment，同时在登陆成功后调用相关member的接口进行初始化。

具体总体概述见userbl模块的整体结构前半段描述，这里只介绍内部具体，使用BrowseController将模块里的业务逻辑和逻辑操作相分开，直接控制Browse（这里Browse是一个基类），以适配器的风格，将浏览订单信息委托给BrowseOrderList，将浏览策略信息委托给BrowseStrategyList,各具体类型的适配器分别调用其他模块接口完成自己的工作。具体的返回也有相应的基本列表包装！

Commentbl 模块设计如下：



commentbl各个类的职责如下

|  |  |
| --- | --- |
| 类 | 职责 |
| commentController | 负责commentbl模块的逻辑操作，实现客户对订单评价及酒店管理人员对订单回评所需要的服务 |
| comment | 评价的领域模型对象，拥有一个订单评价对应的评分评价和文字评价以及提供进行评价和获取评价的方法，可以帮助完成评价界面所需要的服务 |

1. 模块内部类的接口规范

commentController、comment的接口规范如下表所示

commentController:

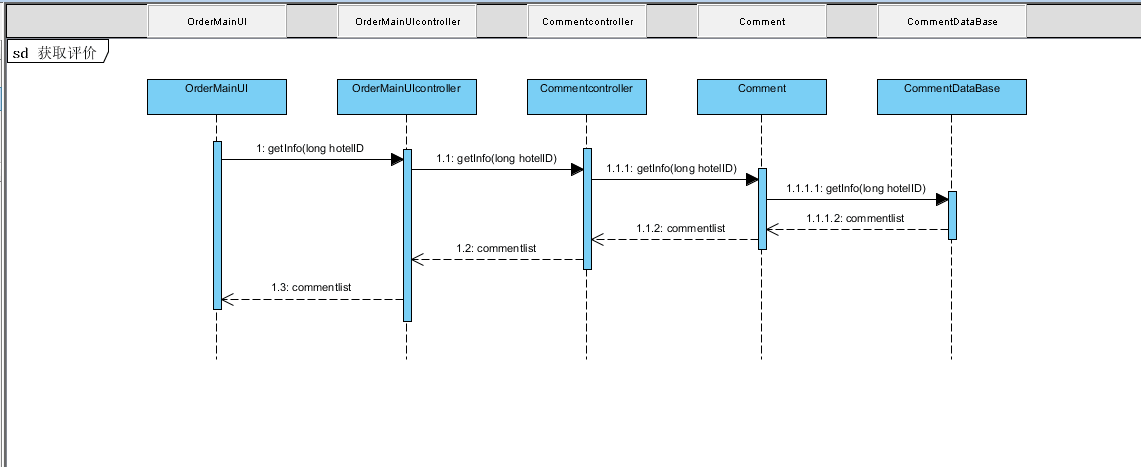
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| commentController.getinfo | 语法 | Public Commentlist gethotleInfo(long hotelid); |
| 前置条件 | 已创建一个comment领域对象 |
| 后置条件 | 调用comment领域对象的getinfo方法 |
| commentController.review | 语法 | Public ResultMessage review(CommentVO vo); |
| 前置条件 | 已创建一个comment领域对象，并且输入符合输入规则 |
| 后置条件 | 调用comment领域对象的review方法 |
| 需要的服务(需接口) | | |
| Comment.getinfo(long hotelid) | 得到基于rule的酒店信息列表 | |
| Comment.review(commentVO vo) | 得到hotel的详细信息 | |

comment类的接口规范如下：

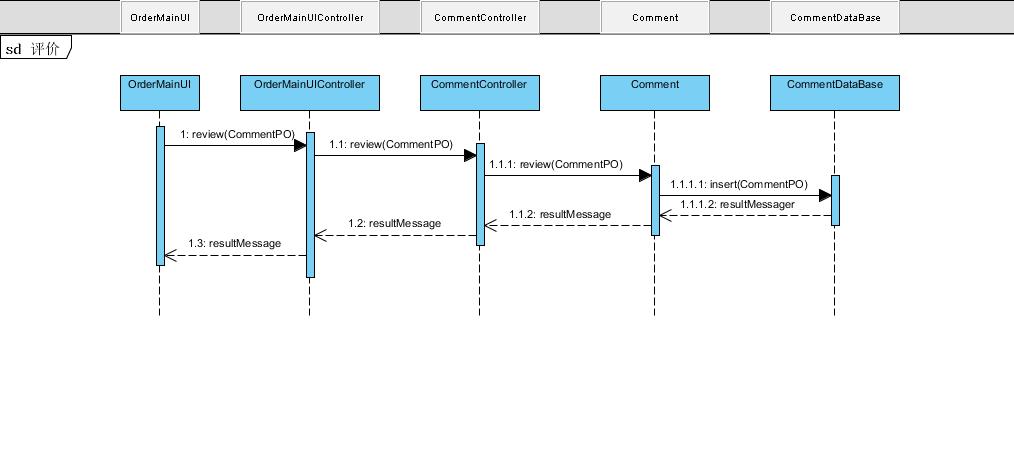
|  |  |  |
| --- | --- | --- |
| 提供的服务(供接口) | | |
| comment.getinfo | 语法 | Public Commentlist getinfo(long hotelid); |
| 前置条件 | 用户已登录，且这里用户为客户 |
| 后置条件 | 得到对应Hotelid的用户评价 |
| Comment.review | 语法 | Public ResultMessage review(CommentVO vo); |
| 前置条件 | 用户已经登录并启动一个评论回合 |
| 后置条件 | 添加一个commentVO到comment数据库 |
| 需要的服务(需接口) | | |
| DatabaseFactory.getCommentDatabase | 得到Comment数据库服务的引用 | |
| CommentDataService.insert(CommentPO po) | 插入单一持久化对象 | |
| CommentDataService.getInfo(long hotelID) | 得到一个单一持久化对象信息 | |

(4)业务逻辑层的动态模型

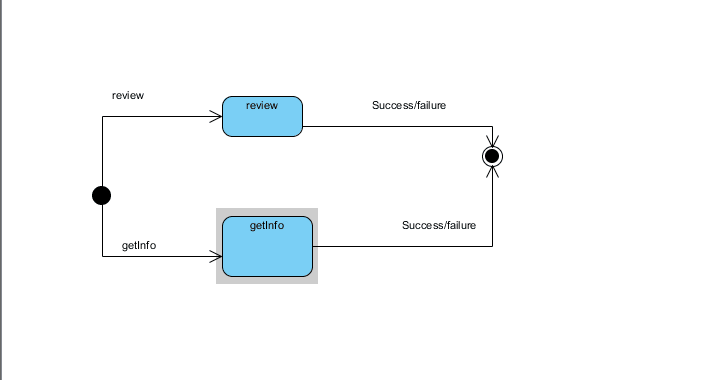
下图显示当用户请求查看评价时，Commentbl模块各个类之间的协作时序图



下图显示当用户请求对订单进行评价时，Commentbl模块各个类之间的协作时序图



下图描述了comment对象的状态图，描述了对象的生存期间的状态序列，引起转移的时间，以及相伴随的操作。当客户发起了一个请求，comment处于抽象状态，控制器解析后调用相应方法（并声明了具体类型），comment变为具体状态并处理请求，处理完后结束



（5）业务逻辑层设计原理

采用适配器模式风格，由controller统一调度，将任务委托给comment，comment对类型进行转换后处理结束。

### 4.1.7 Strategy模块

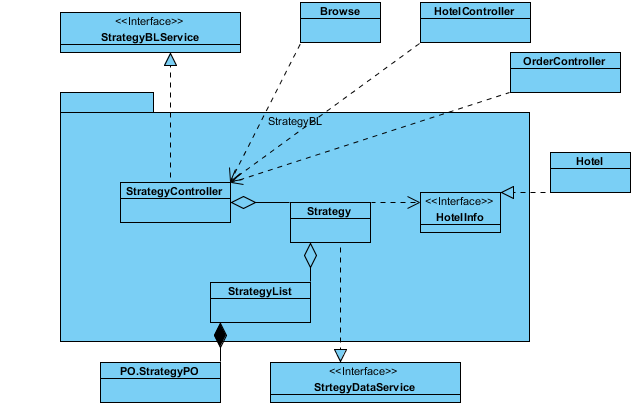
（1）模块概述

strategy模块承担的需求见需求规格说明书功能需求及相关非功能需求。

strategy模块的职责及接口参见软件体系结构描述文档表。

（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。每一层之间未来灵活性，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strategyblservice，strategydataservice两个接口。为了隔离业务逻辑职责和逻辑控制职责，我们添加了StrategyController，这样StrategyController将会将优惠策略相关的业务逻辑职责和逻辑控制委托给strategy对象。StrategyPO是作为优惠策略的持久化对象被添加到设计模型中的。StrategyList的添加是StrategyInfo的容器类。HotelInfo是根据依赖倒置原则，为了消除循环依赖而产生的接口。



|  |  |
| --- | --- |
| 类 | 职责 |
| StrategyController | 负责优惠策略制定界面所需要的服务 |
| Strategy | 优惠策略的领域模型对象 |
| StrategyList | 优惠策略信息集合的领域模型对象 |

（3）模块内部类的接口规范

StrategyController 、Strategy的接口规范如以下所示。

表1.1.2.1-1 StrategyController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| StrategyController.getStrategyInfo | 语法 | Public List<StrategyVO> getStrategyInfo(long hotelID) |
| 前置条件 | 请求获取相关酒店的所有策略信息 |
| 后置条件 | 调用Strategy领域对象的getStrategyInfo方法 |
| StrategyController.deleteStrategyInfo | 语法 | Public ResultMessage deleteStrategyInfo(long hotelID,long StrategyID) |
| 前置条件 | 请求删除策略对象 |
| 后置条件 | 调用Strategy领域对象的deleteStrategyInfo方法 |
| StrategyController.addStrategyInfo | 语法 | Public ResultMessage addRoomInfo(StrategyVO,vo) |
| 前置条件 | 请求增加一条策略信息 |
| 后置条件 | 调用Strategy领域对象的addStrategyInfo方法 |
| StrategyController.changeStrategyInfo | 语法 | Public ResultMessage changeStrategyInfo(StrategyVO,vo) |
| 前置条件 | 已创建一个Strategy领域对象的changeStrategyInfo方法 |
| 后置条件 | 调用Strategy领域对象的changeStrategyInfo方法 |
| StrategyController.getRoomInfo | 语法 | Public List<RoomVO> getRoomInfo(long hotelID) |
| 前置条件 | 请求获取酒店的房间信息 |
| 后置条件 | 调用StrategyController 领域对象的getRoomInfo对象 |
| 需要的服务（需接口） | | |
| 服务名 | | 服务 |
| Strategy.getStrategyInfo | | 获取酒店所有的策略信息列表 |
| Strategy.deleteStrategyInfo | | 删除酒店的某一策略信息 |
| Strategy.addStrategyInfo | | 增加酒店的某一策略信息 |
| Strategy.changeStrategyInfo | | 改变酒店的某一策略信息 |
| Strategy.getRoomInfo | | 获取酒店的客房信息 |

表1.1.2.1-2 Strategy的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Strategy.getStrategyInfo | 语法 | Public List<StrategyVO,vo> getStrategyInfo(long hotelID) |
| 前置条件 | StrategyController请求调用 |
| 后置条件 | 获取酒店所有策略信息的列表 |
| Strategy.deleteStrategyInfo | 语法 | Public ResultMessage deleteStrategyInfo(long hotelID,long strategyID) |
| 前置条件 | StrategyController请求调用 |
| 后置条件 | 删除此酒店的此策略 |
| Strategy.addStrategyInfo | 语法 | Public ResultMessage addStrategyInfo(StrategyVO,vo) |
| 前置条件 | StrategyController请求调用 |
| 后置条件 | 增加一条策略信息 |
| Strategy.changeStrategyInfo | 语法 | Public ResultMessage changeStrategyInfo(StrategyVO,vo) |
| 前置条件 | 此类型的客房存在优惠策略且StrategyController请求调用 |
| 后置条件 | 更改相应的策略信息 |
| Stategy.getRoomInfo | 语法 | Public List<roomVO> getRoomInfo(long hotelID) |
| 前置条件 | StrategyController请求调用 |
| 后置条件 | 返回此酒店的所有客房信息 |
| 需要的服务（需接口） | | |
| 服务名 | | 服务 |
| HotelController.getRoomInfo() | | 获得酒店所有的客房信息的持久化对象 |
| StrategyDataService.add(StrategyPO,po) | | 插入一条策略信息的持久化对象 |
| StrategyDataService.delete(StrategyPO,po) | | 删除一条策略信息的持久化对象 |
| StrategyDataService.change(StrategyPO,po) | | 修改一条策略信息的持久化对象 |

（4）动态模型：

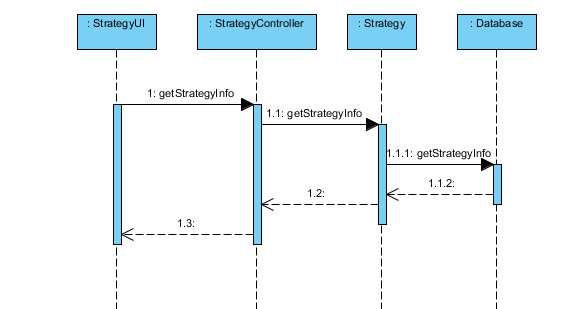


图1.1.2.1-1获取策略信息顺序图

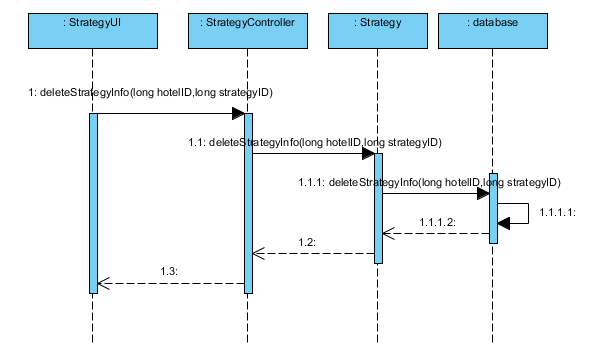


图1.1.2.1-2Strategy的删除策略信息顺序图

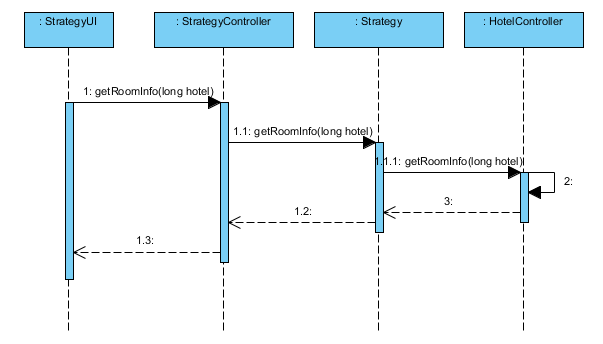


图1.1.2.1-3获取酒店客房信息顺序图

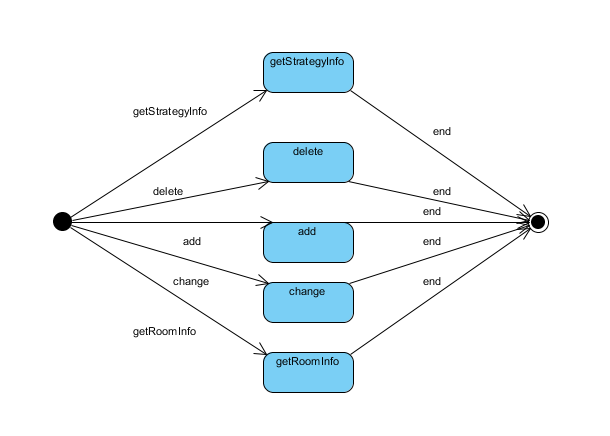


图1.1.2.1-5策略模块状态图

### 4.1.8 Credit模块

（1）模块概述

credit模块承担的需求见需求规格说明书功能需求及相关非功能需求。

credit模块的职责及接口参见软件体系结构描述文档表。

（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creditblservice，creditdataservice两个接口。为了隔离业务逻辑职责和逻辑控制职责，我们添加了creditController，这样creditController将会将信用值相关的业务逻辑职责和逻辑控制委托给credit对象。creditPO是作为信用值的持久化对象被添加到设计模型中的。

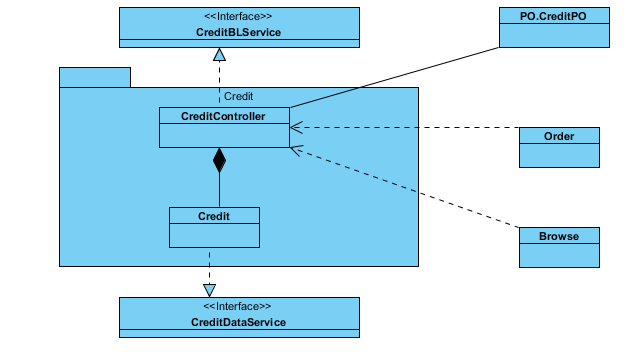


图1.2.2.1-1 credit模块各个类的设计

表1.2.2.1-1 credit模块各个类的职责

|  |  |
| --- | --- |
| 类 | 职责 |
| CreditController | 负责实现对应于信用值查看和充值界面所需的方法 |
| Credit | 负责管理信用值 |

（3）credit模块内部接口规范

表1.2.2.1-2 CreditController的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CreditController.getInfo | 语法 | Public List<CreditPO>getInfo(long userID) |
| 前置条件 | 请求查看用户信用信息明细 |
| 后置条件 | 调用Credit领域对象的getInfo方法 |
| CreditController.update | 语法 | Public ResultMessage update(long userID,int value,String style) |
| 前置条件 | 请求对信用值进行操作 |
| 后置条件 | 调用Credit领域对象的update方法 |
| CreditController.insert | 语法 | Public ResultMessage(CreditPO,po) |
| 前置条件 | 请求插入一条信用信息 |
| 后置条件 | 调用Credit领域对象的insert方法 |
| 需要的服务（需接口） | | |
| Credit.getInfo | | 得到用户的信用信息 |
| Credit.update | | 信用值更改 |
| Credit.insert | | 插入信用信息 |

表1.2.2.1-3 Credit的接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Credit.getInfo | 语法 | Public List<CreditPO>getInfo(long userID) |
| 前置条件 | 请求查看用户信用明细 |
| 后置条件 | 返回用户信用明细信息 |
| Credit.update | 语法 | Public ResultMessage update(long userID,int value,long style) |
| 前置条件 | 请求对客户信用值发生更改 |
| 后置条件 | 更改客户信用值并存储 |
| Credit.insert | 语法 | Public ResultMessage insert(CreditPO,po) |
| 前置条件 | 请求插入一条信用记录 |
| 后置条件 | 插入一条信用记录并存储 |
| 需要的服务（需接口） | | |
| DatabaseFactory.getCreditDatabase | | 得到Credit数据库服务的引用 |
| CreditDataService.update(long userID,int value,string style) | | 更新数据库中的相应信用值 |
| CreditDataService.insert(CreditPO,po) | | 在数据库中插入CreditPO对象 |

（4）动态模型

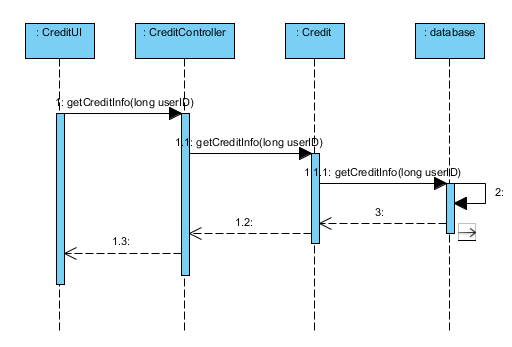


图1.1.2.1-2获取信用明细顺序图

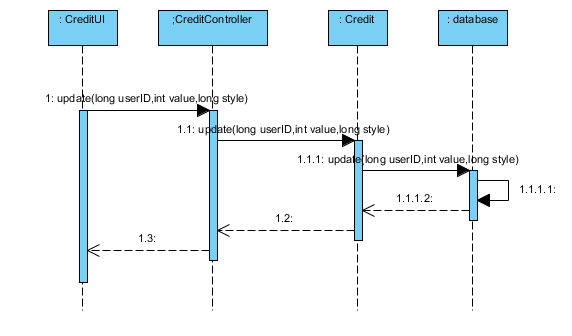


图1.1.2.1-3更新信用值顺序图

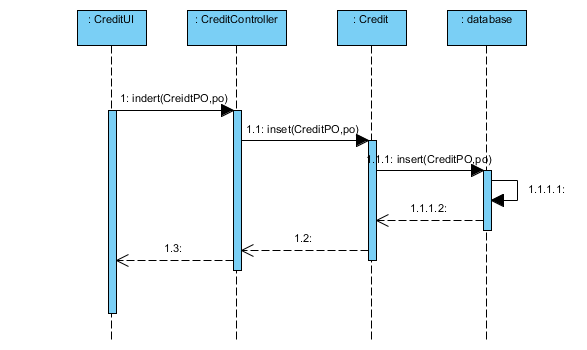


图1.1.2.1-4插入信用信息顺序图

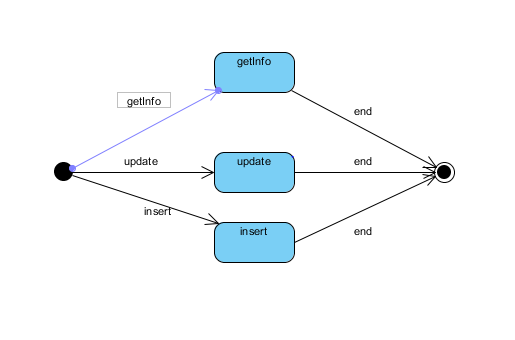
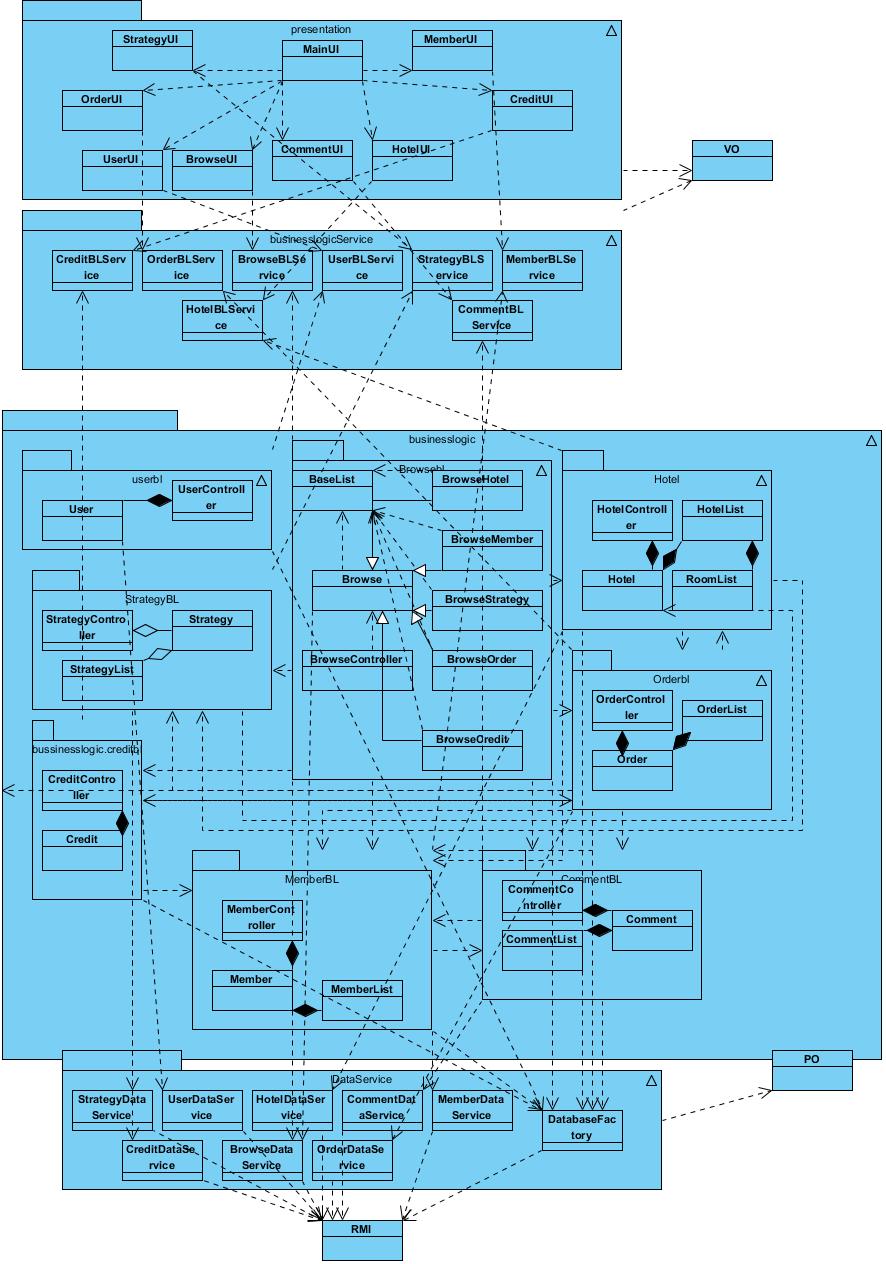


图1.1.2.1-4 Credit模块状态图

# 开发视角

客户端开发包图如下：



服务端开发包图如下：

