酒店预订系统

详细设计文档

V0.3

南京大学软件学院

**Leftovers**工作组

2016-10

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 郭浩滨 | 2016-10-26 | 建立整体框架，完成部分文档 | V0.1 |
| 吴游杰 | 2016-10-29 | 添加personnel（差图片）和hotel模块 | V0.2 |
| 吴志成 | 2016-10-29 | 1. 添加User、order模块 2. 添加数据层的分解 | V0.3 |

**目录**

[1 引言 2](#_Toc465539839)

[1.1 编制目的 2](#_Toc465539840)

[1.2 词汇表 2](#_Toc465539841)

[1.3 参考资料 3](#_Toc465539842)

[2 产品概述 3](#_Toc465539843)

[3 体系结构设计概述 4](#_Toc465539844)

[4 结构视角 4](#_Toc465539845)

[4.1 展示层的分解 4](#_Toc465539846)

[4.1.1 UI层设计概要 4](#_Toc465539847)

[4.1.2 UI层内部包的职责与划分 4](#_Toc465539848)

[4.2 业务逻辑层的分解 4](#_Toc465539849)

[4.2.1 User模块 4](#_Toc465539850)

[4.2.2 Personnel模块 16](#_Toc465539851)

[4.2.3 Hotel模块 22](#_Toc465539852)

[4.2.4 Order模块 28](#_Toc465539853)

[4.2.5 Promotion模块 37](#_Toc465539854)

[4.3 数据层的分解 44](#_Toc465539855)

[4.3.1 DataFactoryService模块 44](#_Toc465539856)

[4.3.2 UserDataService模块 45](#_Toc465539857)

[4.3.3 PersonnelDataService模块 46](#_Toc465539858)

[4.3.4 HotelDataService模块 47](#_Toc465539859)

[4.3.5 OrderDataService模块 48](#_Toc465539860)

[4.3.6 PromotionDataService模块 49](#_Toc465539861)

[5 依赖视角 50](#_Toc465539862)

# 引言

## 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。

## 词汇表

|  |  |
| --- | --- |
| **缩写或单词** | **解释** |
| HOS | 酒店预订系统 |
| member | 会员 |
| user | 客户 |
| personnel | 工作人员 |
| hotel | 酒店 |
| room | 客房 |
| credit | 信用 |
| evaluation | 评价 |
| comment | 评论 |
| rank | 评分 |
| hotelpromotion | 酒店促销策略 |
| webpromotion | 网站营销策略 |
| order | 订单 |
| unexecuted | 未执行的（订单） |
| executed | 已执行的（订单） |
| abnormal | 异常的（订单） |
| canceled | 已撤销的（订单） |
| appeal | 异常订单申诉 |

## 参考资料

《酒店预订系统用例文档V1.2.11》

《酒店预订系统需求规格说明文档V1.1.5》

《酒店预订系统体系结构设计文档V1.1.2》

# 产品概述

参考酒店预订系统用例文档及酒店预订系统规格说明文档对产品的概括描述。

随着工作和生活节奏的加快，人们的差旅出行需要迅速提升，但是传统上的线下酒店预订模式有着很多不方便之处：

* 因为目的地往往是外地，无法全面了解当地的酒店情况，选择范围大大受限，难以选择到最适合的酒店；
* 提前预订的双方都有风险，酒店担心恶意预订，顾客担心酒店不符合预期；
* 酒店要及时掌握预订顾客的行程变化也会比较麻烦，需要频繁的长途联系；
* 虽然过去的体验能够给未来带来参考，但是时间一长很多信息就会被遗忘掉，过去的参考作用就受到限制。（事实上，过去的出行体验会大大影响人们未来的选择）

酒店预订系统就是为满足xx创业公司创业发展需求开发的，它包括一个数据集中服务器和多个客户端。数据集中服务器将所有的数据储存起来进行维护。用户通过客户端完成日常任务，客户端与数据集中服务器采取实时通信的方式完成数据交换。

# 体系结构设计概述

参考酒店预订系统体系结构设计文档的描述。

主要采用分层的架构方式，采用分布式的部署方式，将数据存储在服务器端，客户端

通过RMI方式调用获取数据。

# 结构视角

## 展示层的分解

### UI层设计概要

### UI层内部包的职责与划分

## 业务逻辑层的分解

业务逻辑层的开发包图请查看体系结构设计文档。

### User模块

1. 模块概述：

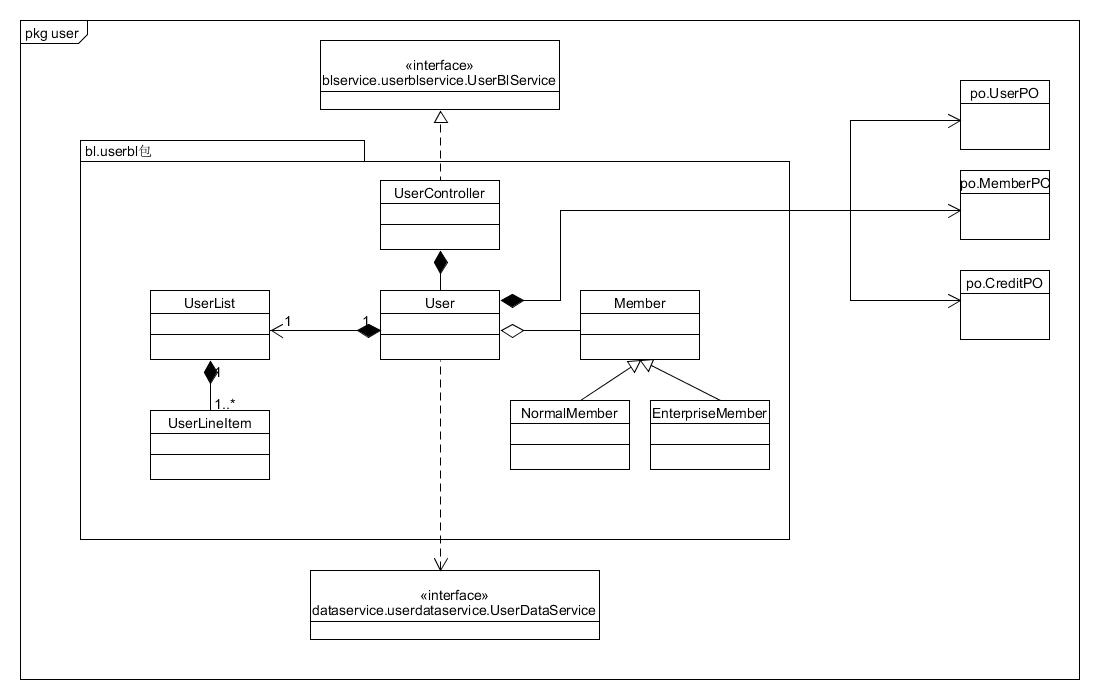
User模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

User模块的职责和接口参见软件系统结构描述文档内对该模块的描述。

1. 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。在展示层和业务逻辑层之间添加UserBlService接口。在业务逻辑层和数据层之间添加UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，添加UserController，这样UserController会将对客户管理的业务逻辑处理委托给User对象。UserPO是作为账户信息的持久性对象被添加到设计模型中去的。

**userbl模块的设计如图：**



**userbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责对应于客户管理所需要的服务 |
| User | 客户管理的领域模型对象，拥有客户的信息，可以帮助完成客户界面所需要的服务 |

1. 模块内部类的接口规范

**UserController的接口规范**

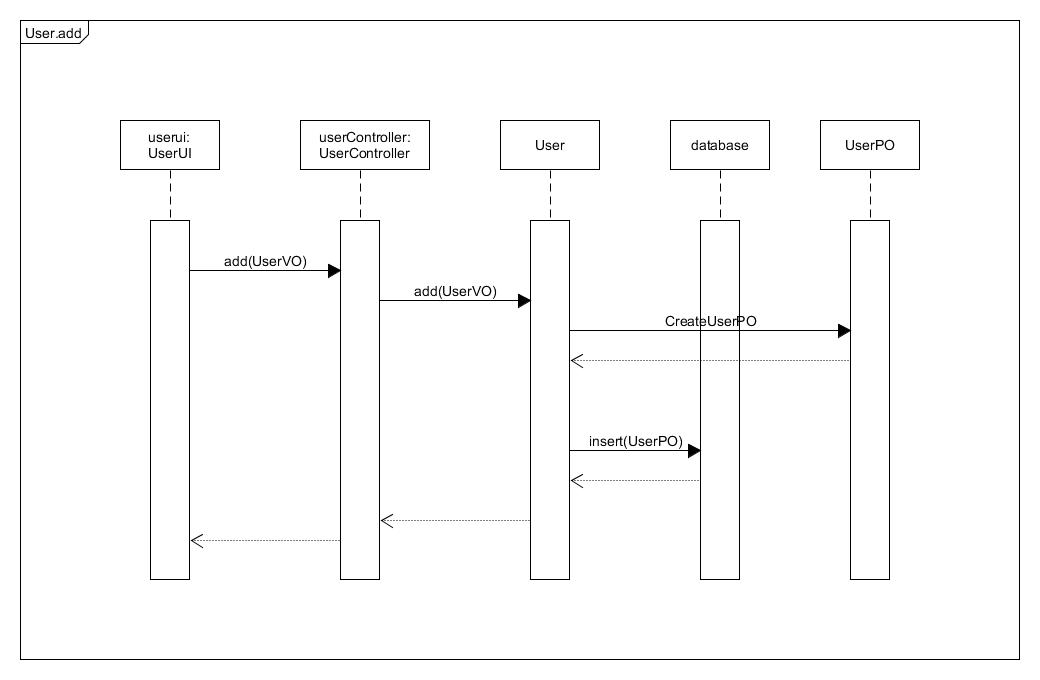
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public ResultMessage login(UserVO userVO) |
| 前置条件 | 已创建一个User领域对象，并且用户名/password输入符合输入规则 |
| 后置条件 | 调用User领域对象的login方法 |
| UserController.logout | 语法 | public ResultMessage logout() |
| 前置条件 | 已创建一个User领域对象，客户已登录 |
| 后置条件 | 调用User领域对象的logout方法 |
| UserController.add | 语法 | public ResultMessage add(UserVO userVO) |
| 前置条件 | 已创建一个User领域对象，账户信息符合规则且齐全，且不与已注册的用户名重复 |
| 后置条件 | 调用User领域对象的add方法 |
| UserController.del | 语法 | public ResultMessage del(UserVO userVO) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的del方法 |
| UserController.find | 语法 | public UserVO find(String username) |
| 前置条件 | 已创建一个User领域对象，管理员已登录 |
| 后置条件 | 调用User领域对象的find方法 |
| UserController.modify | 语法 | public ResultMessage modify(UserVO userVO) |
| 前置条件 | 已创建一个User领域对象，用户已登录，账户信息符合规则且齐全 |
| 后置条件 | 调用User领域对象的setInfo方法 |
| UserController.showList | 语法 | public ArrayList<UserVO> showList() |
| 前置条件 | 已创建一个User领域对象，管理员已登陆 |
| 后置条件 | 调用User领域对象的showList方法 |
| UserController.getMemberType | 语法 | public MemberType getMemberType(String username) |
| 前置条件 | 已创建一个User领域对象，酒店工作人员/网站营销人员已登录 |
| 后置条件 | 调用User领域对象的getMemberType方法 |
| UserController.addCredit | 语法 | public ResultMessage addCredit(String username, double amount) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的addCredit方法 |
| UserController.deductCredit | 语法 | public ResultMessage deductCredit(String username, double amount) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的deductCredit方法 |
| UserController.showCreditRecord | 语法 | public CreditVO showCreditRecord (String username) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的showCreditRecord方法 |
| UserController.regisiterMember | 语法 | public ResultMessage regisiterMember(MemberVO memberVO) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的regisiterMember方法 |
| UserController.deleteMember | 语法 | public ResultMessage deleteMember(String username) |
| 前置条件 | 已创建一个User领域对象 |
| 后置条件 | 调用User领域对象的deleteMember方法 |
| 需要的接口（需接口） | | |
| 服务名 | 服务内容 | |
| User.login | 客户登录 | |
| User.logout | 客户登出 | |
| User.add | 客户注册 | |
| User.del | 删除一个用户 | |
| User.find | 查询一个用户信息 | |
| User. modify | 修改用户的信息 | |
| User.showList | 显示（全部）用户列表 | |
| User.getMemberType | 获得客户的会员类型 | |
| User.addCredit | 增加客户的信用值 | |
| User. deductCredit | 减少客户的信用值 | |
| User. showCreditRecord | 查看信用记录 | |
| User. regisiterMember | 客户注册会员 | |
| User. deleteMember | 删除会员 | |

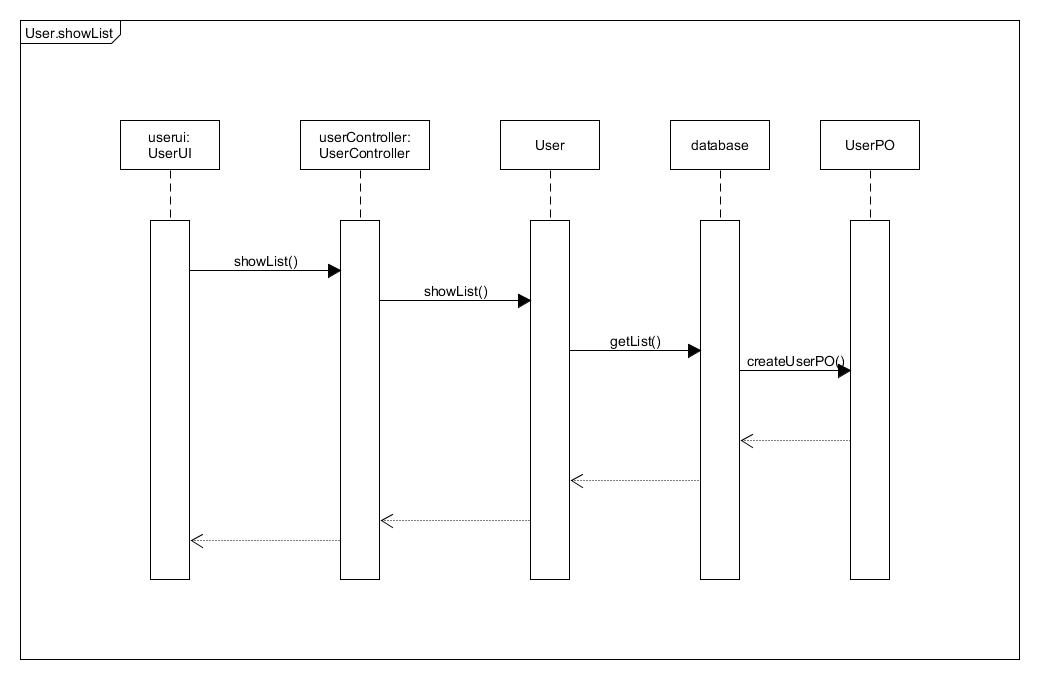
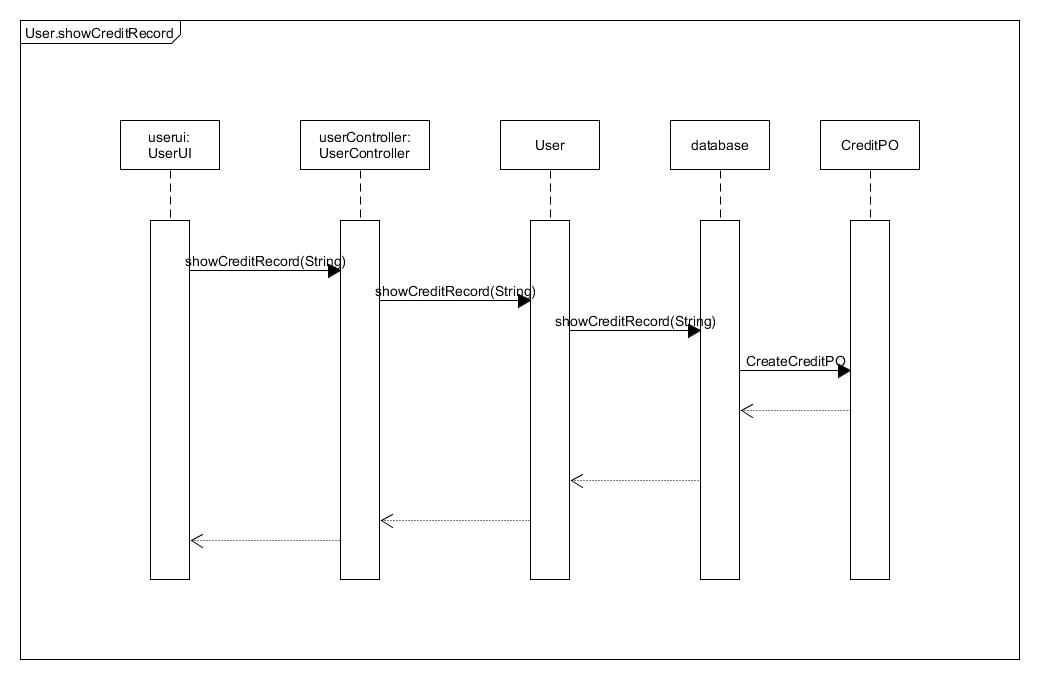
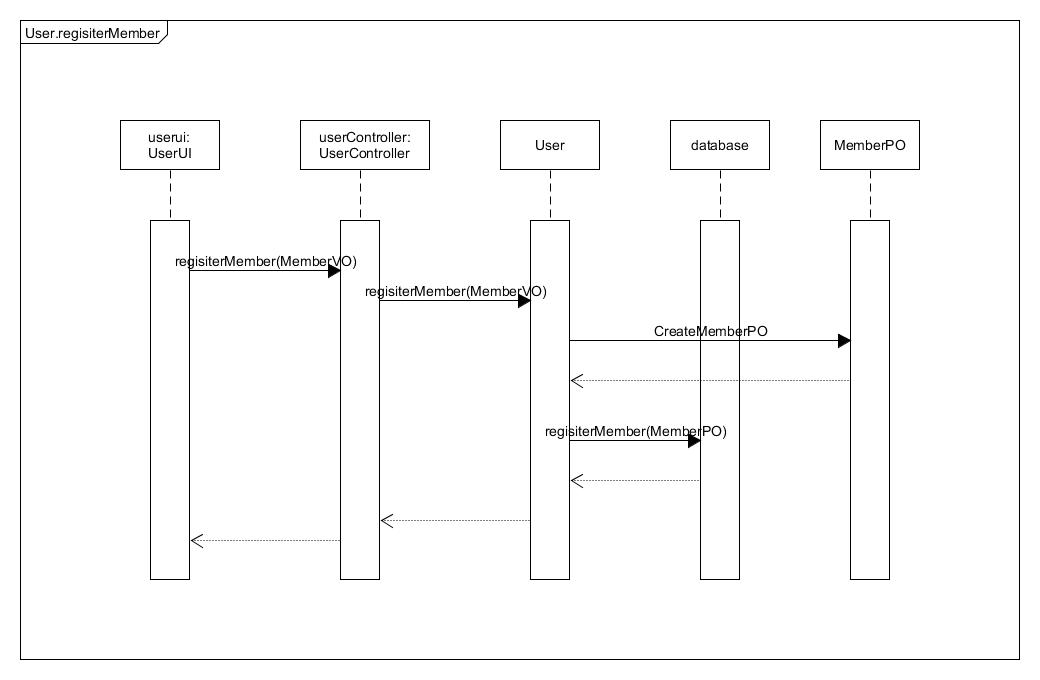
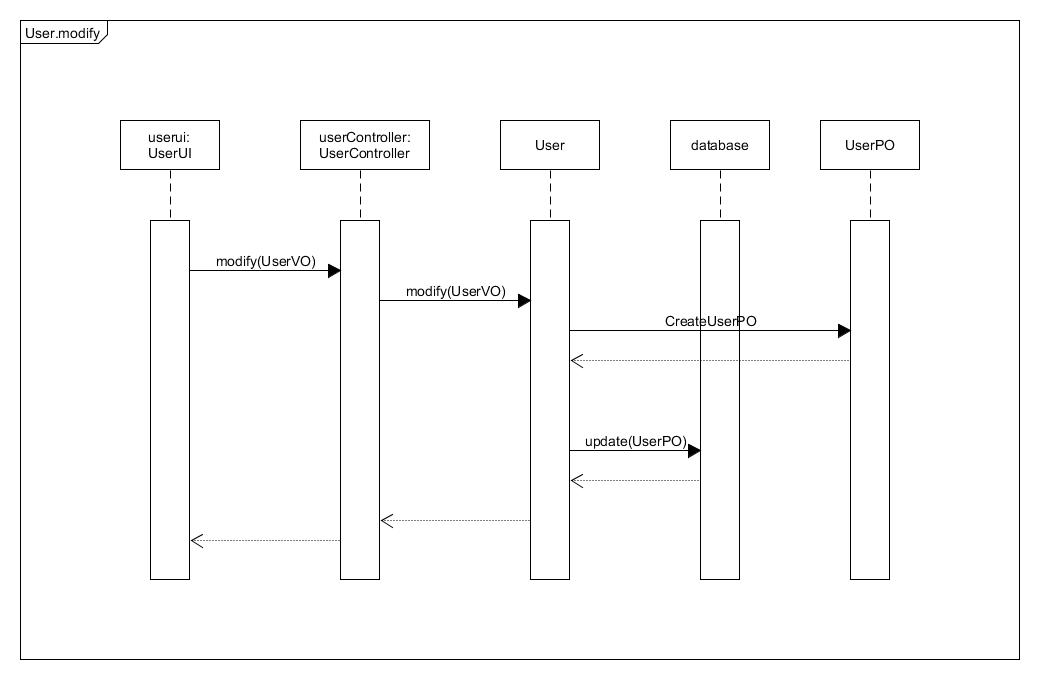
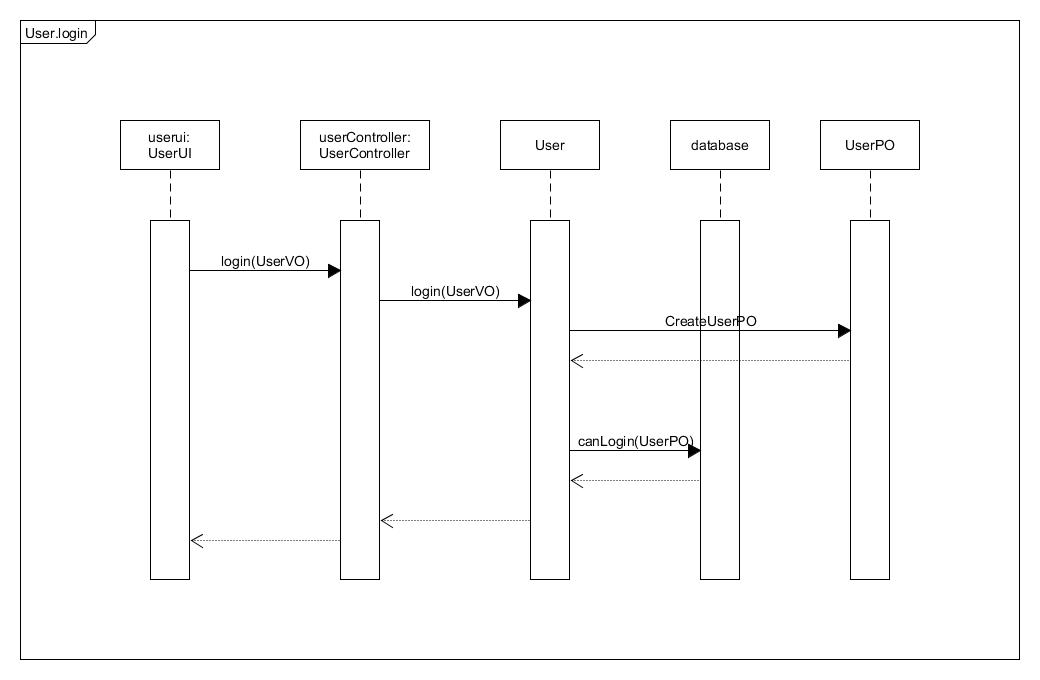
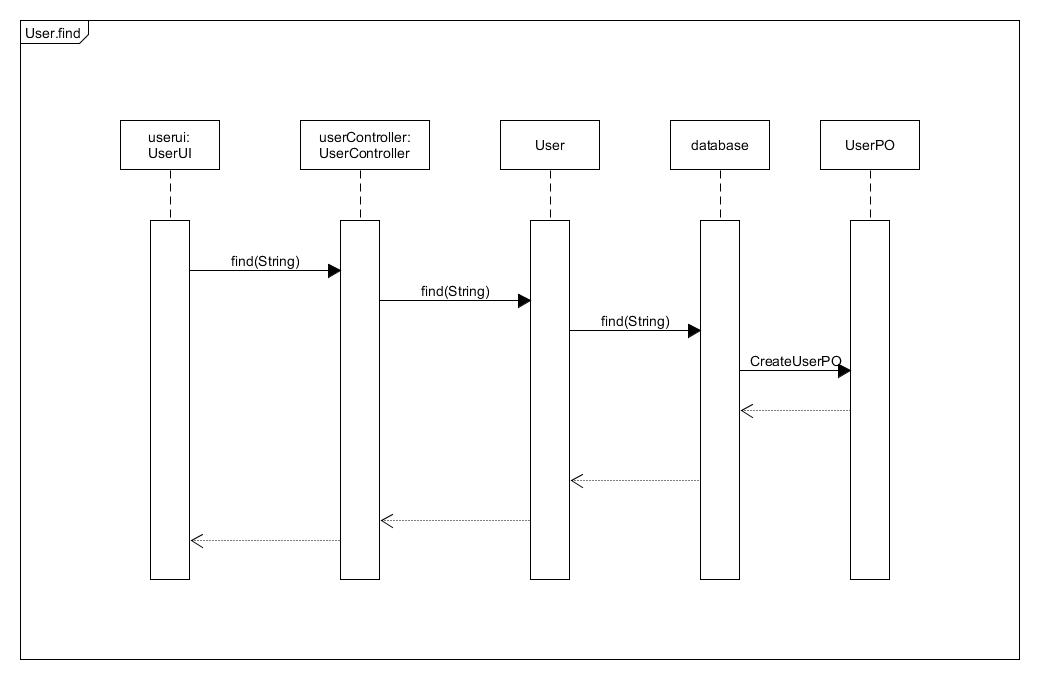
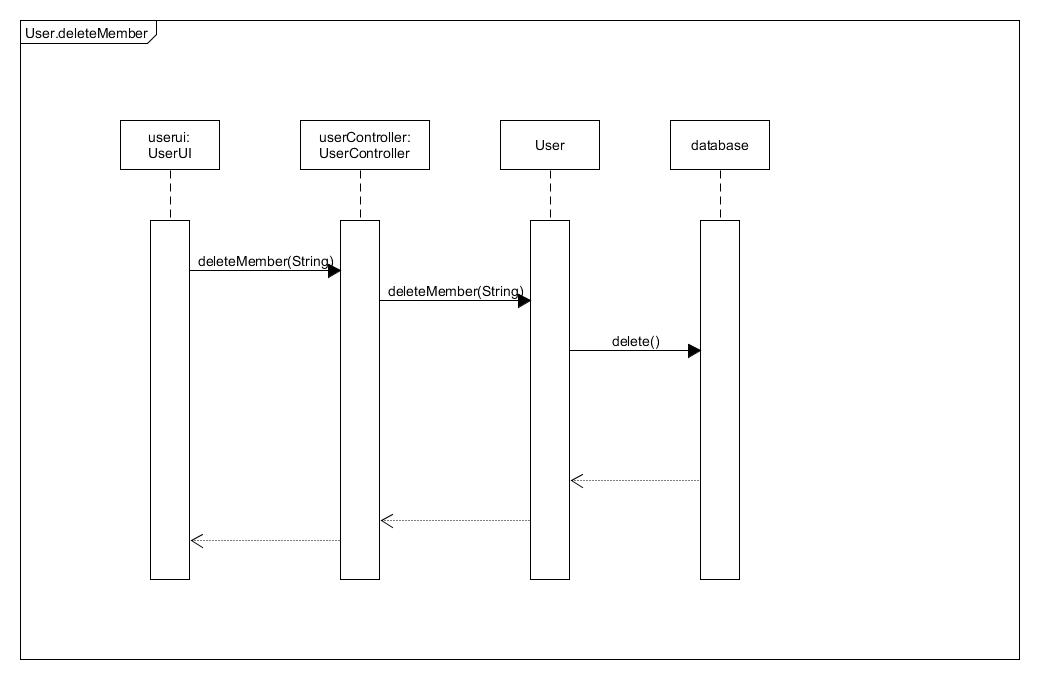
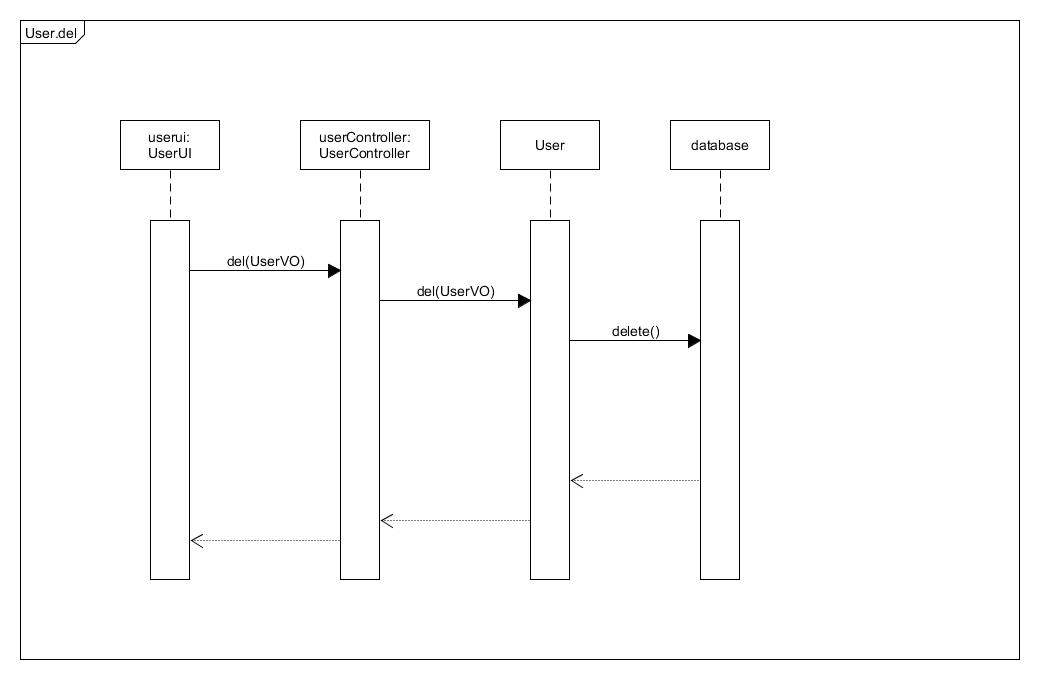
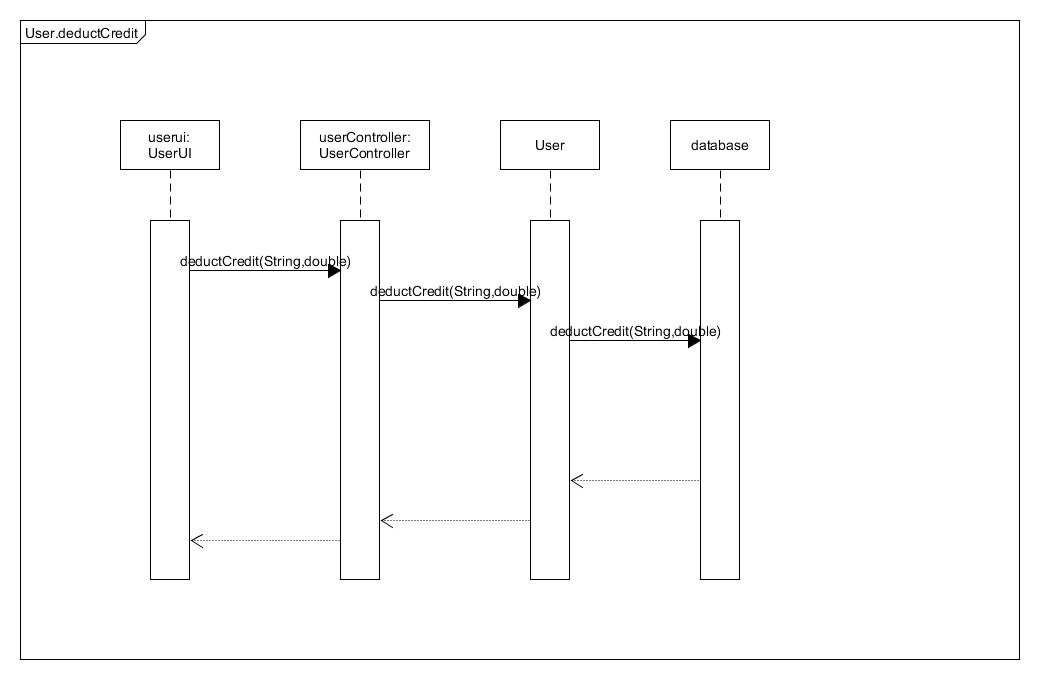
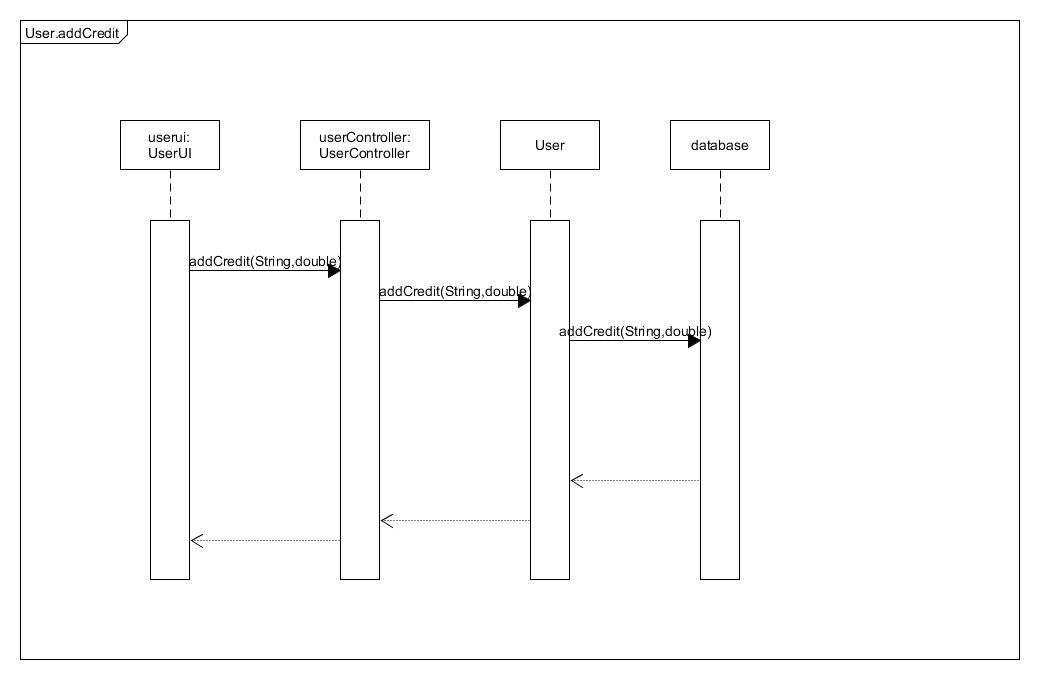
**User的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public ResultMessage login(UserVO userVO) |
| 前置条件 | password符合输入规则且对应的客户未登录 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证的结果 |
| User.logout | 语法 | public ResultMessage logout() |
| 前置条件 | 客户已登录 |
| 后置条件 | 系统更改用户登录状态 |
| User.add | 语法 | public ResultMessage add(UserVO userVO) |
| 前置条件 | 账户信息符合规则且齐全，且不与已注册的用户名重复 |
| 后置条件 | 系统创建该账户并持久化增加该账户数据 |
| User.del | 语法 | public ResultMessage del(UserVO userVO) |
| 前置条件 | 无（删除表明该数据必然时存在的） |
| 后置条件 | 系统永久性 |
| User.find | 语法 | public UserVO find(String username) |
| 前置条件 | 管理员已登录 |
| 后置条件 | 系统返回查找到的账户信息 |
| User.modify | 语法 | public ResultMessage modify(UserVO userVO) |
| 前置条件 | 客户已登录，账户信息符合规则且齐全 |
| 后置条件 | 系统修改该账户并持久化保存该账户数据 |
| User.showList | 语法 | public ArrayList<UserVO> showList() |
| 前置条件 | 管理员已登陆 |
| 后置条件 | 系统返回目前客户列表 |
| User.getMemberType | 语法 | public MemberType getMemberType(String username) |
| 前置条件 | 酒店工作人员/网站营销人员已登录 |
| 后置条件 | 系统返回客户会员类型 |
| User.addCredit | 语法 | public ResultMessage addCredit(String username, double amount) |
| 前置条件 | 无 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| User.deductCredit | 语法 | public ResultMessage deductCredit(String username, double amount) |
| 前置条件 | 无 |
| 后置条件 | 持久化更新涉及的领域对象的数据 |
| User. showCreditRecord | 语法 | public CreditVO showCreditRecord (String username) |
| 前置条件 | 客户已登录 |
| 后置条件 | 系统返回客户信用记录 |
| User.regisiterMember | 语法 | public ResultMessage regisiterMember (MemberVO memberVO) |
| 前置条件 | 客户已登录且信用值大于1000 |
| 后置条件 | 创建一个新的会员对象并持久化该对象 |
| User.deleteMember | 语法 | public ResultMessage deleteMember (String username) |
| 前置条件 | 客户已是会员 |
| 后置条件 | 根据username搜索并删除对应的持久化对象 |
| 需要的接口（需接口） | | |
| 服务名 | 服务内容 | |
| UserDataService.canLogin(UserPO userPO) | 根据提供的VO判断是否允许登录 | |
| UserDataService.findByUsername(String username) | 根据用户名查找持久化对象 | |
| UserDataService.findByType(UserType userType) | 根据userType搜索并返回相应的用户列表 | |
| UserDataService.insert(UserPO userPO) | 根据提供的VO判断是否允许注册 | |
| UserDataService.delete(UserPO userPO) | 删除单一持久化对象 | |
| UserDataService.update(UserPO userPO) | 修改单一持久化对象 | |
| UserDataService.getList() | 显示用户列表 | |
| UserDataService.addCredit(String username, double amount) | 持久化增加客户信用值 | |
| UserDataService.deductCredit(String username, double amount) | 持久化扣除客户信用值 | |
| UserDataService. showCreditRecord (String username) | 根据用户名查找持久化对象 | |
| UserDataService.insertMember(MemberPO memberPO) | 插入单一持久化对象 | |
| UserDataService.deleteMember(String username) | 删除单一持久化对象 | |

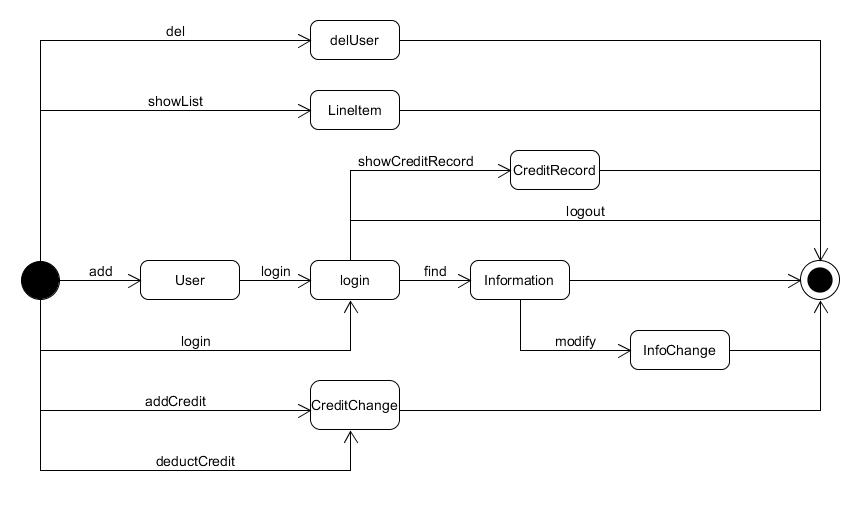
1. 业务逻辑层的动态模型

在客户管理中，业务分为登录、注册、删除、修改客户信息、查询客户信息、注册会员、删除会员、客户列表、增加客户信用值、扣除客户信用值、查看信用记录，其中顺序图如下图所示：





下图描述了User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。



1. 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。

### Personnel模块

1、模块概述：

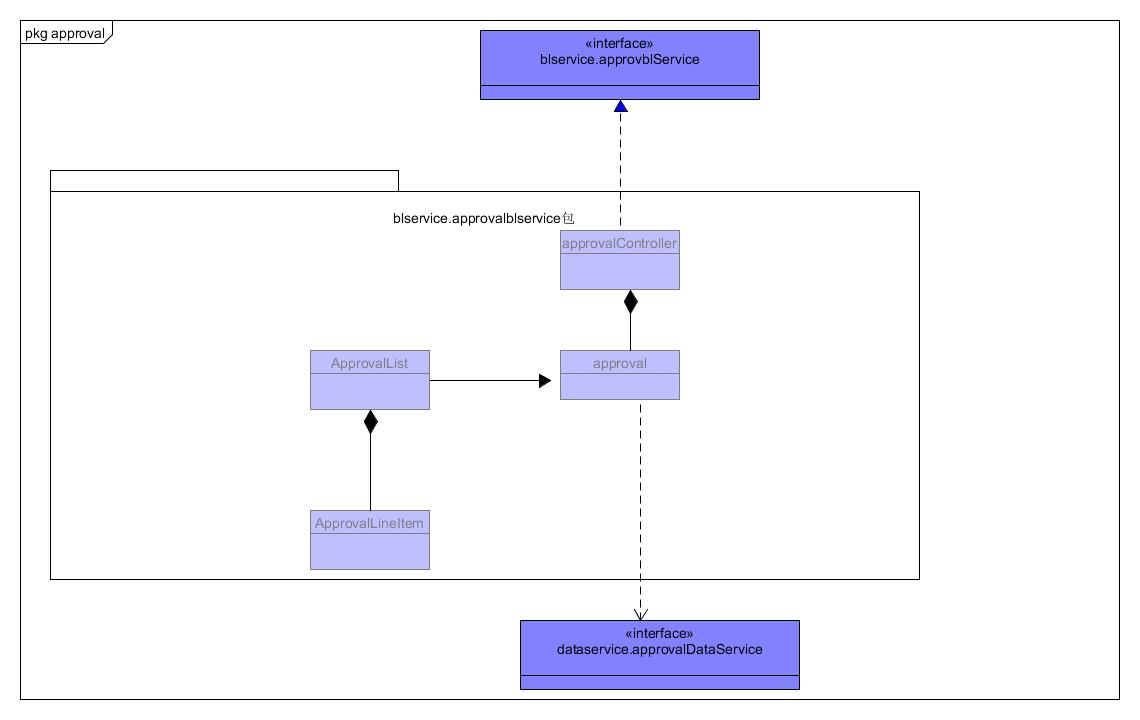
**Personnel**模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

**Personnel**模块的职责和接口参见软件系统结构描述文档内对该模块的描述。

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。在展示层和业务逻辑层之间添加PersonnelblService接口。在业务逻辑层和数据层之间添加PersonneldataService接口。为了隔离业务逻辑职责和逻辑控制职责，添加PersonnelController，这样PersonnelController会将对工作人员管理的业务逻辑处理委托给Personnel对象。PersonnelPO是作为账户信息的持久性对象被添加到设计模型中去的。

**Personnel模块中各个类的设计：**



**Personnelbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| PersonnelController | **实现酒店查找及管理部分需要的服务** |
| Personnnel | **实现酒店查找及管理，包括增删改查4项功能** |

1. 模块内部类的接口规范

**PersonnelController模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PersonnelController.login | PersonnelController.login | PersonnelController.login |
| 前置条件 | 已经创建一个Approval领域对象，并且输入符合语法规则信息符合规则且齐全 |
| 后置条件 | 调用Approval类的getDocs方法 |
| PersonnelController.logout | PersonnelController.logout | PersonnelController.logout |
| 前置条件 | 已经创建一个Approval领域对象，并且输入符合语法规则信息符合规则且齐全 |
| 后置条件 | 调用Approval类的approveone方法 |
| PersonnelController.showList | PersonnelController.showList | PersonnelController.showList |
| 前置条件 | 已经创建一个Approval领域对象，并且输入符合语法规则信息符合规则且齐全 |
| 后置条件 | 调用Approval类的approvemany方法 |
| PersonnelController.add | | |
| 服务名 | 服务内容 | |
| Transport.getDoc(DocType type) | 从货物流转包中获取各种单据 | |
| PersonnelController.del | PersonnelController.del | |
| Store.getDoc(DocType type) | 获取库存相关的单据 | |
| Approval.getbills | 获得所有单据 | |
| PersonnelController.modify | PersonnelController.modify | |
| Approval.approvemany | 审批多个单据 | |

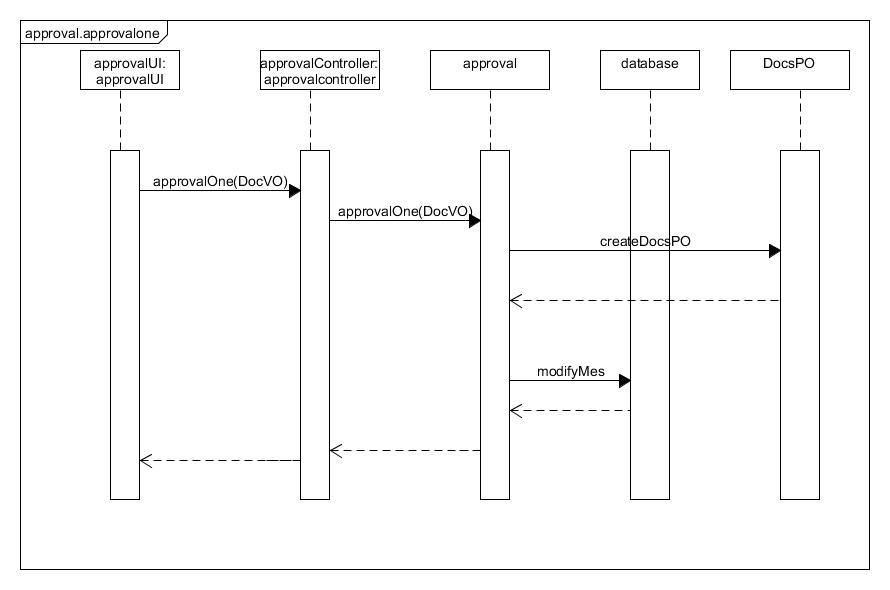
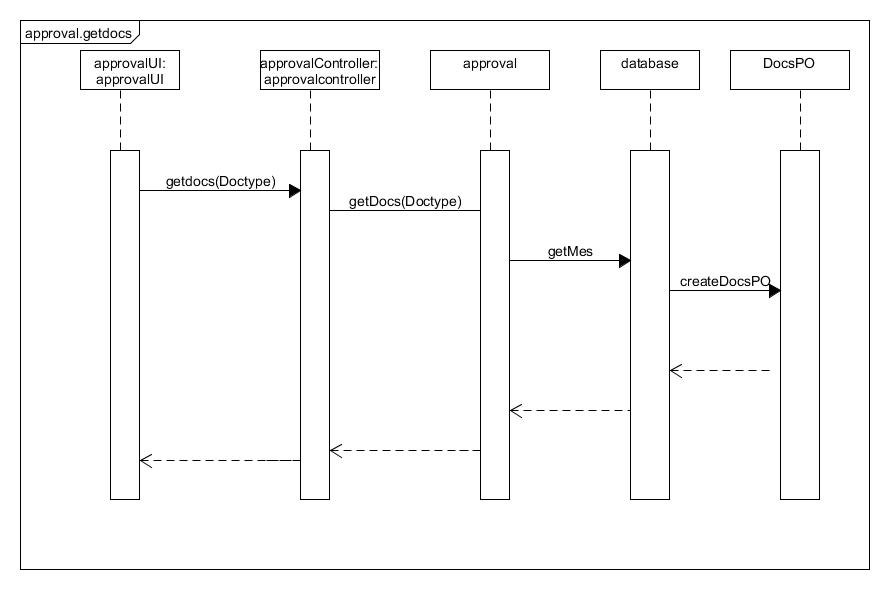
**Personnelbl模块的接口规范**

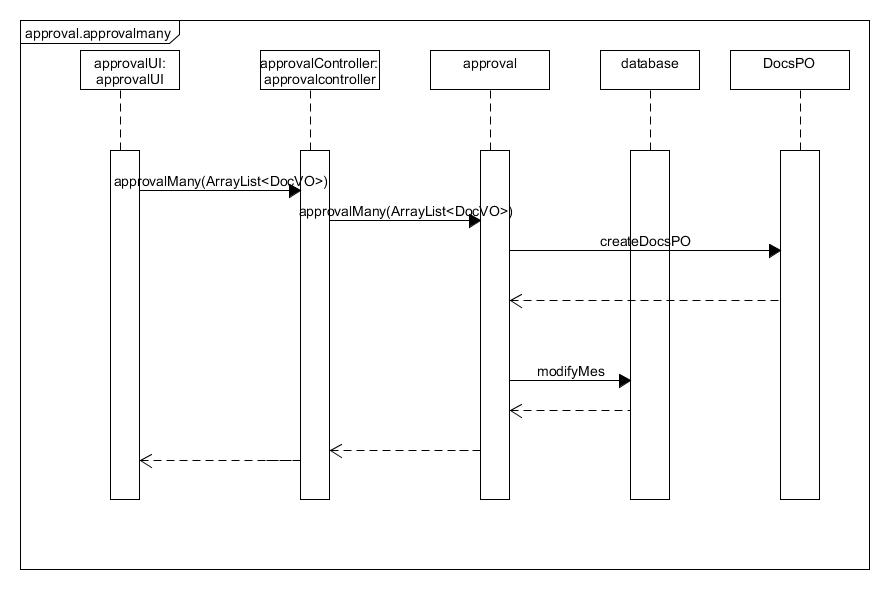
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Personnel.login | 语法 | public ResultMessage login(PersonnelVO personnelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统设置账户登录状态为已登录 |
| Personnel.logout | 语法 | public ResultMessage logout() |
| 前置条件 | 工作人员已登录 |
| 后置条件 | 系统设置账户登录状态为未登录 |
| Personnel.showList | 语法 | Public ArrayList<PersonnelVO> showList() |
| 前置条件 | 管理员已经登录 |
| 后置条件 | 系统返回personnel的列表 |
| Personnel.add | 语法 | public ResultMessage add(PersonnelVO personnelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统添加新的账户信息，并持久化更新相关数据 |
| Personnel.del | 语法 | public ResultMessage del(PersonnelVO personnelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统持久化删除该账户数据 |
| Personnel.modify | 语法 | public ResultMessage modify(PersonnelVO personnelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统修改账户数据，并持久化更新相关数据 |
| Personnel.find | 语法 | public PersonnelVO find(long personnelID) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统返回符合personnelID的账户的详细信息 |

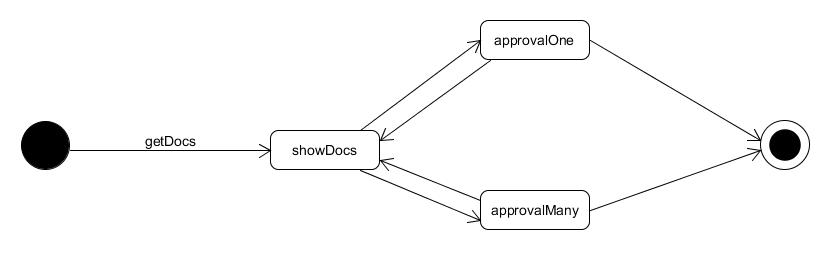
|  |  |
| --- | --- |
| 需要的接口（需接口） | |
| 服务名 | 服务内容 |
| PersonnelDataService.canLogin(PersonnelPO personnelPO) | 根据提供的PO判断是否允许登录 |
| PersonnelDataService.getList(long personnelID) | 显示工作人员列表 |
| PersonnelDataService.insert(PersonnelPO personnelPO) | 根据提供的PO判断是否允许添加 |
| PersonnelDataService.delete(PersonnelPO personnelPO) | 删除单一持久化对象 |
| PersonnelDataService.update(PersonnelPO personnelPO) | 修改单一持久化对象 |
| PersonnelDataService.findByPersonnelID() | 根据ID查找持久化单一对象 |

1. 业务逻辑层的动态模型

在审批单据中，业务分为获得单据信息、单个审批单据、批量审批单据，它们的顺序图

如下图所示：



下图描述了Approval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。

1. 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

### Hotel模块

1、模块概述：

**Hotel**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。

**Hotel**模块的职责和接口参见软件系统结构描述文档内对该模块的描述。

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。在展示层和业务逻辑层之间添加HotelblService接口。在业务逻辑层和数据层之间添加HoteldataService接口。为了隔离业务逻辑职责和逻辑控制职责，添加HotelController，这样HotelController会将对酒店查找及管理的业务逻辑处理委托给Hotel对象。HotelPO是作为账户信息的持久性对象被添加到设计模型中去的。

**图4-1-3-1 Hotelbl模块各个类的设计**



hotelbl模块各个类的职责如表4-1-3-2所示。

**表4-1-3-2 Hotelbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| HotelController | **实现酒店查找及管理部分需要的服务** |
| Hotel | **实现酒店查找及管理，包括增删改查4项功能** |

3、模块内部类的接口规范

HotelController和Hotel的接口规范如表4-1-3-3和表4-1-3-4所示

**表4-1-3-3 HotelController的接口规范**

|  |  |  |  |
| --- | --- | --- | --- |
| 提供的服务（供接口） | | | |
| HotelController.find | | 语法 | public HotelVO find(long hotelID) |
| 前置条件 | 已经创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel类的find方法，返回符合hotelID的酒店的详细信息 |
| HotelController.showList | | 语法 | Public ArrayList<HotelVO> showList(HotelVO hotelVO) |
| 前置条件 | 已经创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel类的showList方法，系统返回符合提供的HotelVO包含的信息的酒店列表 |
| HotelController.add | | 语法 | public ResultMessage add(HotelVO hotelVO) |
| 前置条件 | 已经创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel类的add方法，系统将新的酒店添加到数据库中，持久化更新相关数据 |
| HotelController.del | | 语法 | public ResultMessage del(long hotelID) |
| 前置条件 | 已经创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel类的del方法，系统从数据库中永久性删除酒店信息，并持久化更新相关数据 |
| HotelController.modify | | 语法 | public ResultMessage modify(HotelVO) |
| 前置条件 | 已经创建一个Hotel领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Hotel类，系统永久性修改该酒店的数据，并持久化更新相关数据 |
| 需要的服务（需接口） | | | | | |
| 服务名 | | 服务 | | | |
| Hotel.find | | 查看酒店详细信息 | | | |
| Hotel.showList | | 查看酒店列表 | | | |
| Hotel.add | | 新增酒店信息 | | | |
| Hotel.del | | 删除酒店信息 | | | |
| Hotel.modify | | 修改酒店信息 | | | |

**Hotelbl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Hotel.find | 语法 | public HotelVO find(long hotelID) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统返回符合hotelID的酒店的详细信息 |
| Hotel.showList | 语法 | Public ArrayList<HotelVO> showList(HotelVO hotelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统返回符合提供的HotelVO包含的信息的酒店列表 |
| Hotel.add | 语法 | public ResultMessage add(HotelVO hotelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统添加新的酒店信息，并持久化更新相关数据 |
| Hotel.del | 语法 | public ResultMessage del(long hotelID) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统删除酒店信息，并持久化更新相关数据 |
| Hotel.modify | 语法 | public ResultMessage modify(HotelVO hotelVO) |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 系统修改该酒店的数据，并持久化更新相关数据 |

|  |  |
| --- | --- |
| 需要的接口（需接口） | |
| 服务名 | 服务内容 |
| HotelDataService.findByHotelID(long hotelID) | 根据ID进行查找单一持久化对象 |
| HotelDataService.findByHotelPO(HotelPO hotelPO) | 根据提供的VO进行查找多个持久化对象 |
| HotelDataService.insert(HotelPO hotelPO) | 插入单一持久化对象 |
| HotelDataService.delete(HotelPO hotelPO) | 删除单一持久化对象 |
| HotelDataService.update(HotelPO hotelPO) | 更新单一持久化对象 |

4、业务逻辑层的动态模型

在酒店查找及管理业务中，分为查找酒店、查看酒店详细信息、新增酒店、删除酒店和修改酒店信息功能。

图4-1-3-2表明了对酒店查找及管理的相关对象之间的协作。











**图4-1-3-2酒店查找及管理**

图4-1-3-3 所示的状态图描述了Hotel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。



**图4-1-3-3 Hotel对象状态图**

### Order模块

1、模块概述：

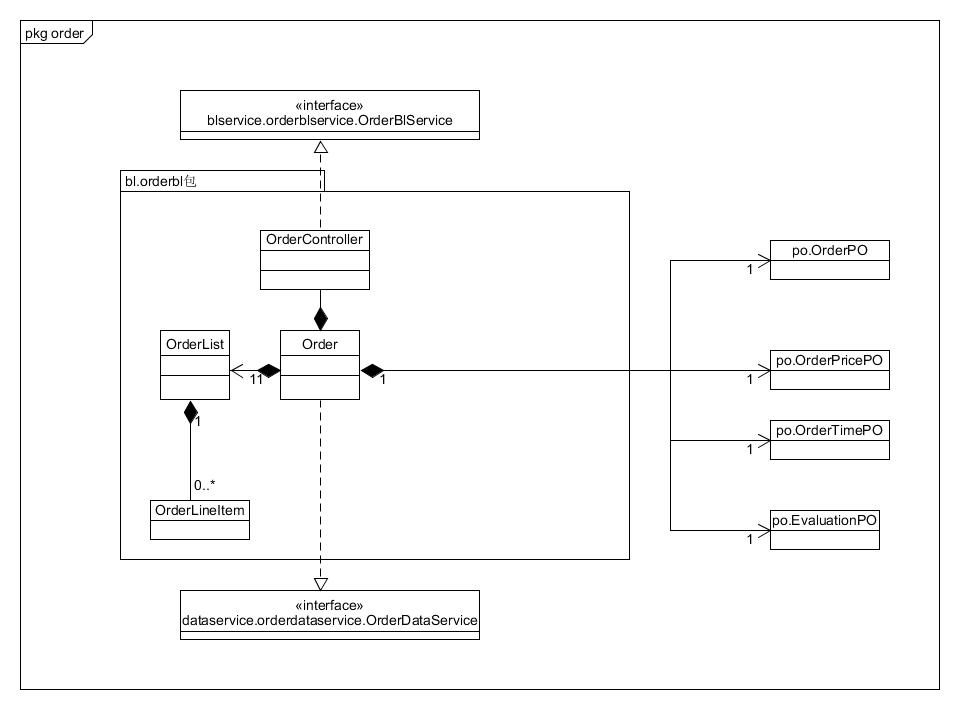
**Order**模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

**Order**模块的职责和接口参见软件系统结构描述文档内对该模块的描述。

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。在展示层和业务逻辑层之间添加OrderBlService接口。在业务逻辑层和数据层之间添加OrderDataService接口。为了隔离业务逻辑职责和逻辑控制职责，添加OrderController，这样OrderController会将对订单管理的业务逻辑处理委托给Order对象。Order PO是作为订单信息的持久性对象被添加到设计模型中去的。

**orderbl模块各个类的设计**



orderbl模块各个类的职责如表所示。

|  |  |
| --- | --- |
| 模块 | 职责 |
| OrderController | 负责对应于订单所需要的服务 |
| Order | 订单的领域模型对象，拥有订单的信息，可以帮助完成订单所需要的服务 |

3、模块内部类的接口规范

**OrderController模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| OrderController.find | 语法 | public OrderVO find(String orderID) |
| 前置条件 | 已创建一个Order领域对象，用户已登录 |
| 后置条件 | 调用Order领域对象的find方法 |
| OrderController.add | 语法 | public ResultMessage add(OrderVO orderVO) |
| 前置条件 | 已创建一个Order领域对象，用户已登录，订单信息符合规则且齐全 |
| 后置条件 | 调用Order领域对象的add方法 |
| OrderController.modify | 语法 | public ResultMessage modify(OrderVO orderVO) |
| 前置条件 | 已创建一个Order领域对象，用户已登录，订单信息符合规则且齐全 |
| 后置条件 | 调用Order领域对象的modify方法 |
| OrderController.evaluate | 语法 | public ResultMessage evaluate(OrderVO orderVO) |
| 前置条件 | 已创建一个Order领域对象，用户已登录，订单状态为已执行 |
| 后置条件 | 调用Order领域对象的modify方法 |
| OrderController.cancel | 语法 | public ResultMessage cancel() |
| 前置条件 | 已创建一个Order领域对象，用户已登录 |
| 后置条件 | 调用Order领域对象的cancel方法 |
| OrderController.showList | 语法 | public ArrayList<OrderVO> showList(OrderVO orderVO) |
| 前置条件 | 已创建一个Order领域对象，用户已登录 |
| 后置条件 | 调用Order领域对象的showList方法 |
| OrderController. showListByUsername | 语法 | public ArrayList<OrderVO> showListByUsername(String username) |
| 前置条件 | 已创建一个Order领域对象，用户已登录 |
| 后置条件 | 调用Order领域对象的showListByUsername方法 |
| OrderController.showListByUsername\_orderType | 语法 | public ArrayList<OrderVO> showListByUsername\_orderType(String username,OrderType orderType) |
| 前置条件 | 已创建一个Order领域对象，用户已登录 |
| 后置条件 | 调用Order领域对象的showListByUsername\_orderType方法 |
| OrderController. showEvaluationListByHotelID | 语法 | public ArrayList<CreditVO> showEvaluationListByHotelID(long hotelID) |
| 前置条件 | 已创建一个Order领域对象，输入符合规则 |
| 后置条件 | 调用Order领域对象的showEvaluationListByHotelID方法 |
| OrderController. changeOrderType | 语法 | public ResultMessage changeOrderType(String orderID) |
| 前置条件 | 已创建一个Order领域对象，订单已存在 |
| 后置条件 | 调用Order领域对象的changeOrderType方法 |

|  |  |
| --- | --- |
| 需要的接口（需接口） | |
| 服务名 | 服务内容 |
| Order.find | 查询一个订单 |
| Order.add | 生成一个订单 |
| Order.modify | 修改一个订单 |
| Order.evaluate | 评价一个订单 |
| Order.cancel | 撤销一个订单 |
| Order.showList | 显示订单列表 |
| Order. showListByUsername | 根据用户名查找订单列表 |
| Order.showListByUsername\_orderType | 根据用户名、订单类型查找订单列表 |
| Order. showEvaluationListByHotelID | 根据酒店id获得订单评价列表 |
| Order. changeOrderType | 改变订单类型 |

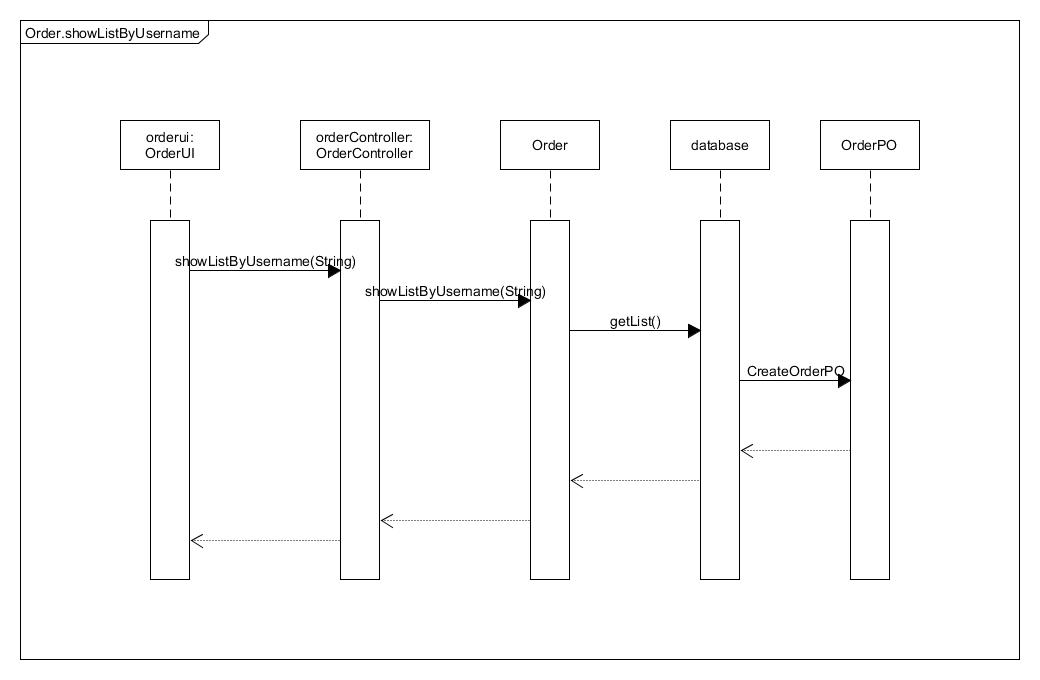
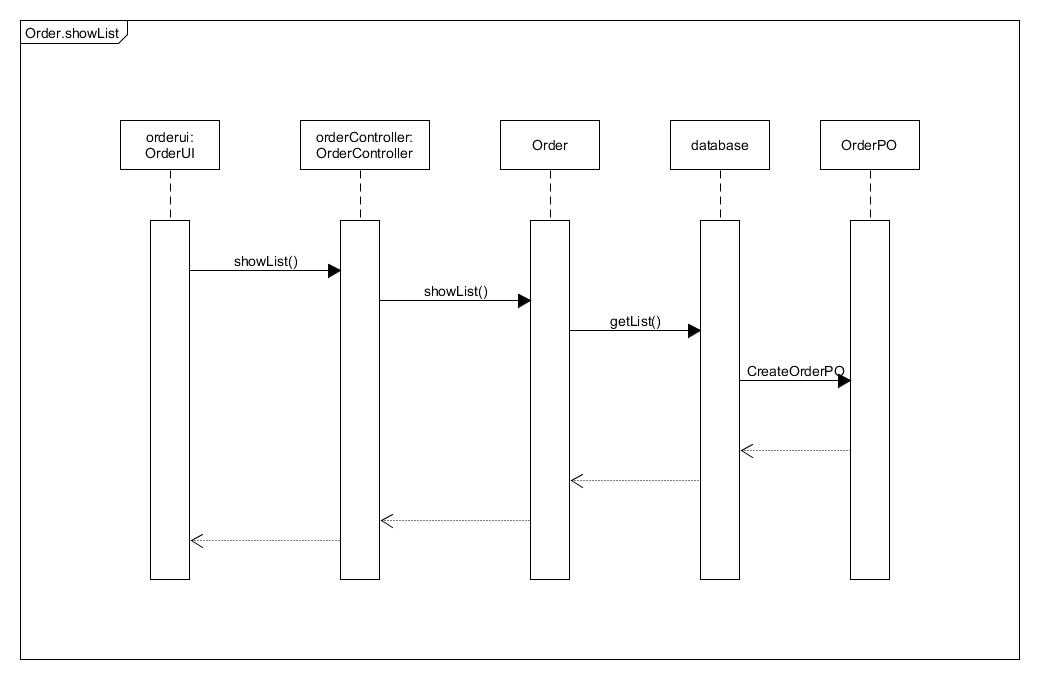
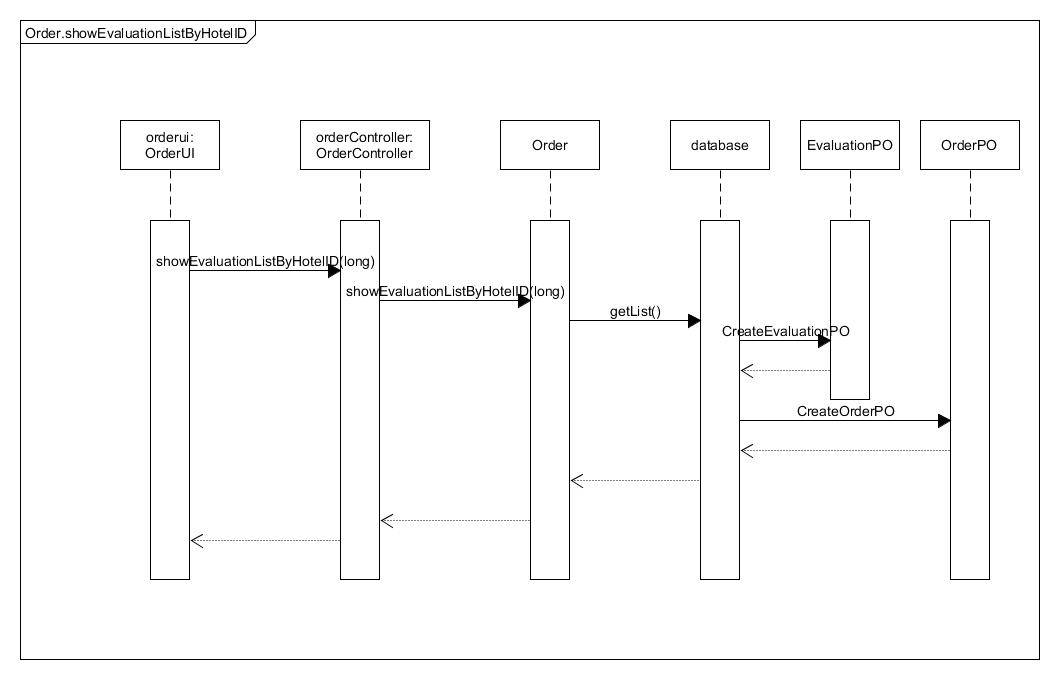
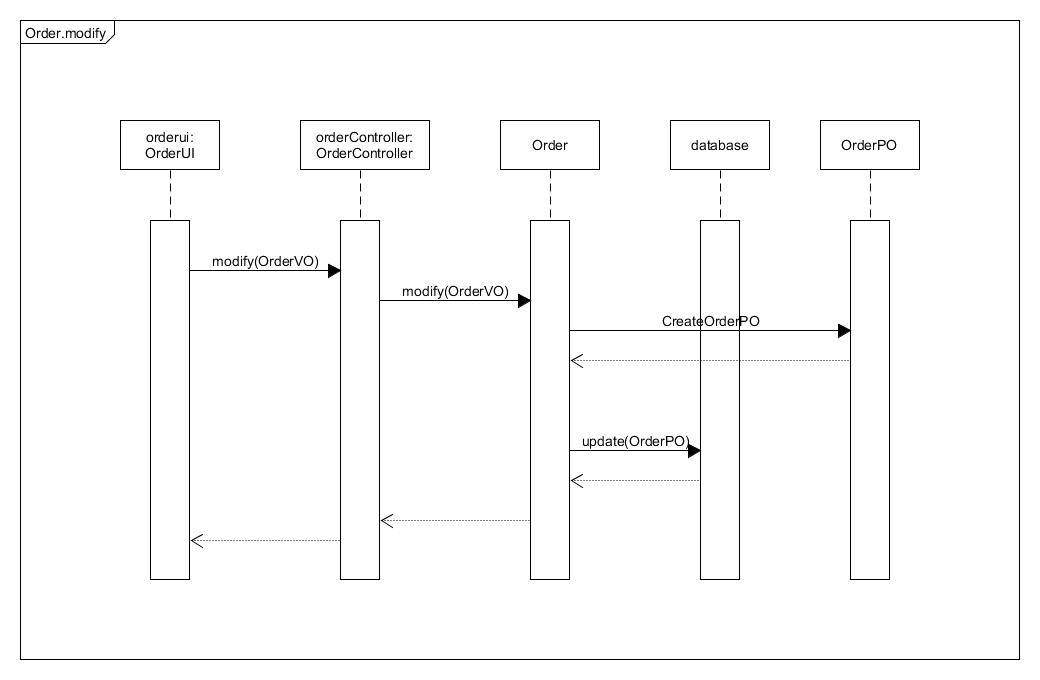
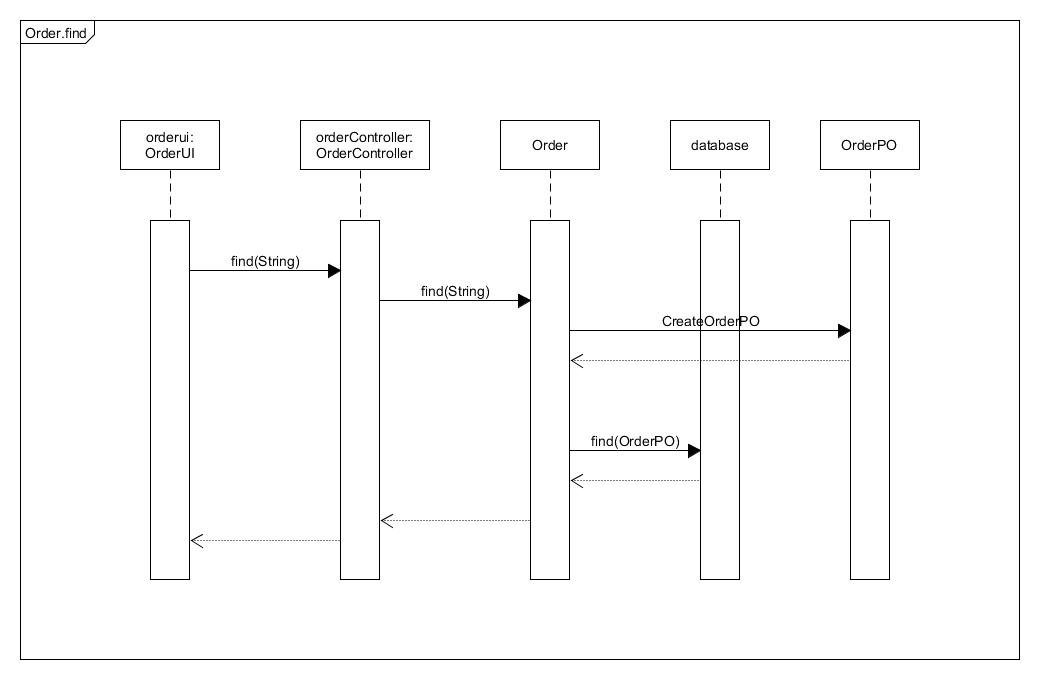
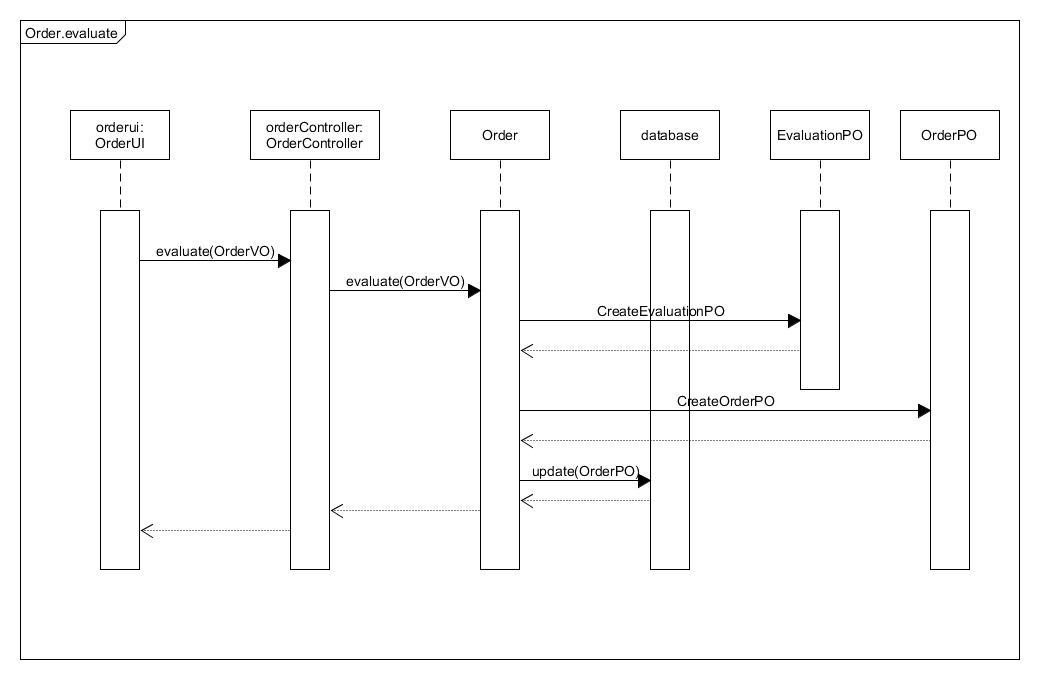
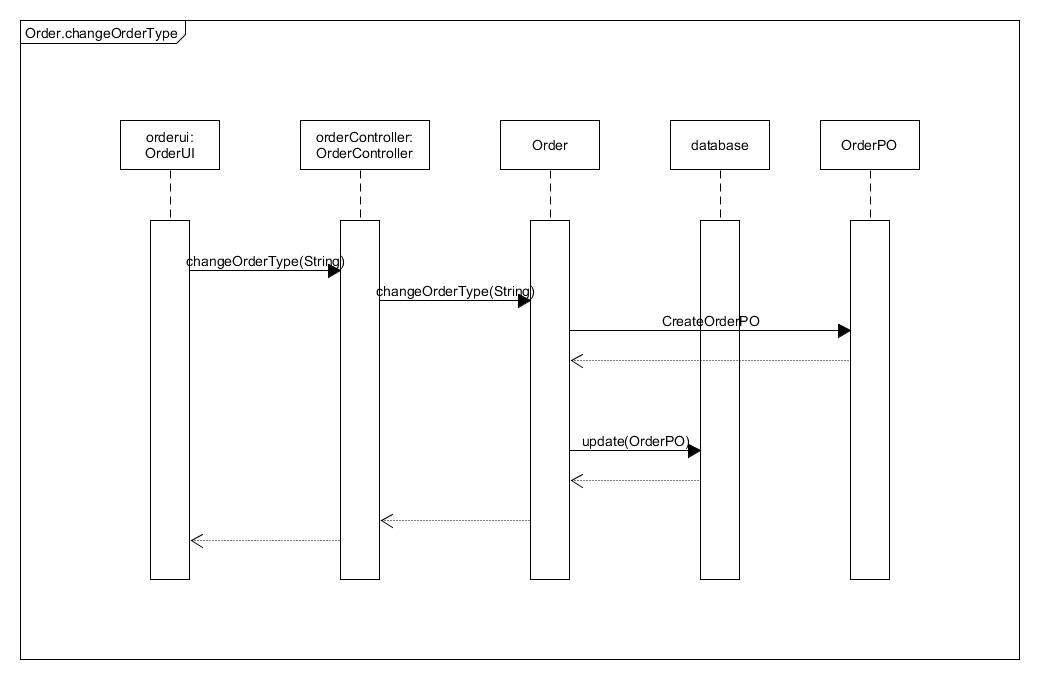
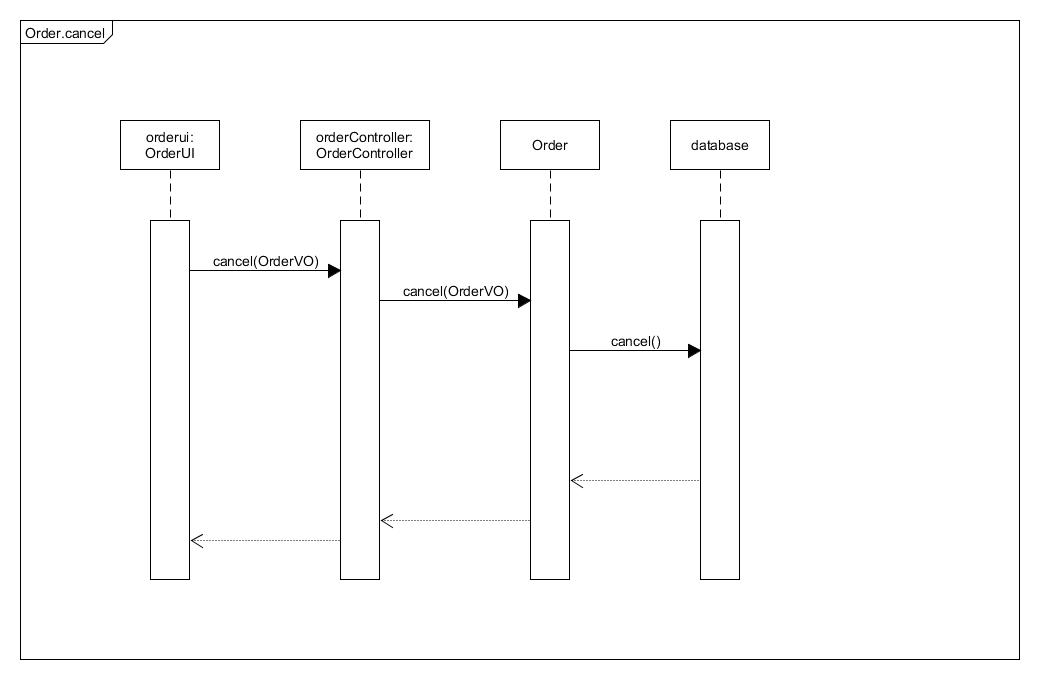
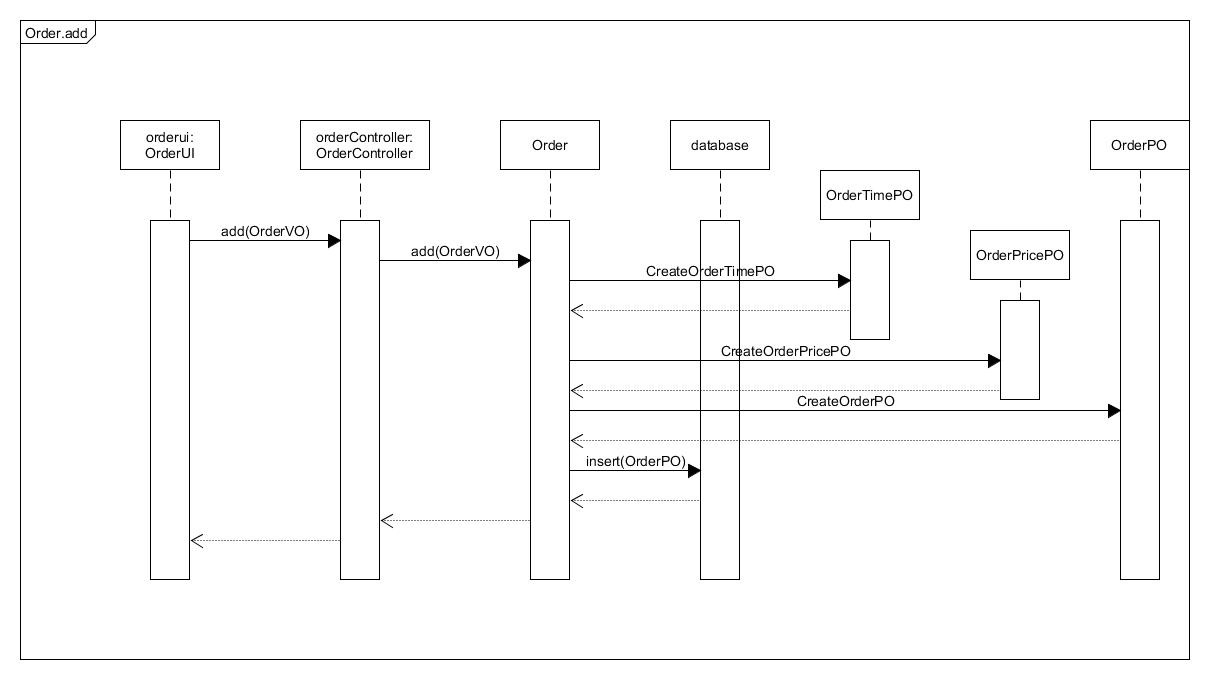
**orderbl模块的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Order.find | 语法 | public OrderVO find(String orderID) |
| 前置条件 | 用户已登录 |
| 后置条件 | 系统返回订单基本信息 |
| Order.add | 语法 | public ResultMessage add(OrderVO orderVO) |
| 前置条件 | 用户已登录，订单信息符合规则且齐全 |
| 后置条件 | 系统增加该订单并持久化保存该订单数据 |
| Order.modify | 语法 | public ResultMessage modify(OrderVO orderVO) |
| 前置条件 | 用户已登录，订单信息符合规则且齐全 |
| 后置条件 | 系统修改该订单并持久化保存该订单数据 |
| Order.evaluate | 语法 | public ResultMessage evaluate(OrderVO orderVO) |
| 前置条件 | 用户已登录，订单状态为已执行 |
| 后置条件 | 系统修改该订单评价信息并持久化保存该订单数据 |
| Order.cancel | 语法 | public ResultMessage cancel() |
| 前置条件 | 用户已登录 |
| 后置条件 | 系统修改该订单状态信息并持久化保存该订单数据 |
| Order.showList | 语法 | public ArrayList<OrderVO> showList(OrderVO orderVO) |
| 前置条件 | 用户已登录 |
| 后置条件 | 系统返回相应条件下的订单列表 |
| Order. showListByUsername | 语法 | public ArrayList<OrderVO> showListByUsername((String username) |
| 前置条件 | 用户已登录 |
| 后置条件 | 系统返回相应条件下的订单列表 |
| Order.showListByUsername\_orderType | 语法 | public ArrayList<OrderVO> showListByUsername\_orderType((String username,OrderType orderType) |
| 前置条件 | 用户已登录 |
| 后置条件 | 系统返回相应条件下的订单列表 |
| Order. showEvaluationListByHotelID | 语法 | public ArrayList<CreditVO> showEvaluationListByHotelID(long hotelID) |
| 前置条件 | 输入符合规则 |
| 后置条件 | 系统返回相应条件下的评价列表 |
| Order. changeOrderType | 语法 | public ResultMessage changeOrderType(String orderID) |
| 前置条件 | 订单已存在 |
| 后置条件 | 系统改变订单状态类型 |

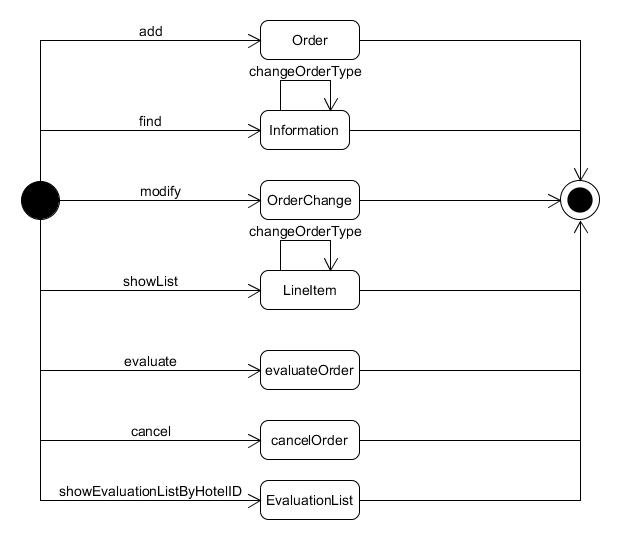
|  |  |
| --- | --- |
| 需要的接口（需接口） | |
| 服务名 | 服务内容 |
| OrderDataService.findByOrderID(String orderID) | 显示订单信息 |
| OrderDataService.find(OrderPO orderPO) | 显示相应条件下的订单列表 |
| OrderDataService.insert(OrderPO orderPO) | 增加单一持久化对象 |
| OrderDataService.update(OrderPO orderPO) | 修改单一持久化对象 |
| OrderDataService.getList() | 显示订单列表 |
| OrderDataService. findEvaluationListByHotelID | 显示酒店订单评价列表 |

4、业务逻辑层的动态模型

在订单管理中，业务分为生成订单、撤销订单、修改订单、获得订单列表、查询订单、评价订单、获得酒店评价列表、根据类型查看订单列表，它们的顺序图如下图所示：



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



5、业务逻辑层的设计原理

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。

### Promotion模块

1、

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。

## 数据层的分解

数据层模块的分解详见需求规格说明文档中的相关描述

### DataFactoryService模块

1. 模块概述

DataFactoryService模块是一个工厂类，用来创建其他的DataService实例，并将其统一管理起来。

1. 模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| DataFactory.getUserDataService | 语法 | public UserDataService getUserDataImpl() |
| 前置条件 | 无 |
| 后置条件 | 返回UserDataService对象 |
| DataFactory.getPersonnelDataService | 语法 | public PersonnelDataService getPersonnelDataImpl() |
| 前置条件 | 无 |
| 后置条件 | 返回PersonnelDataService对象 |
| DataFactory.getHotelDataService | 语法 | public HotelDataService getHotelDataImpl() |
| 前置条件 | 无 |
| 后置条件 | 返回HotelDataService对象 |
| DataFactory.getOrderDataService | 语法 | public OrderDataService getOrderDataImpl() |
| 前置条件 | 无 |
| 后置条件 | 返回OrderDataService对象 |
| DataFactory.getPromotionDataService | 语法 | public PromotionDataService getPromotionDataImpl() |
| 前置条件 | 无 |
| 后置条件 | 返回PromotionDataService对象 |

### UserDataService模块

（1）模块概述

UserDataService模块负责保存用户账户的信息，进行增删改查等操作。

（2）模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| UserDataService.init | 语法 | public void init() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 初始化持久化数据库 |
| UserDataService.canLogin | 语法 | public boolean canLogin(UserPO userPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 返回对应username的用户的是否可登录属性值 |
| UserDataService.findByUsername | 语法 | public UserPO findByUsername(String username) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按username进行查找返回相应的UserPO结果 |
| UserDataService.findByType | 语法 | public ArrayList<UserPO> findByType(UserType userType) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 返回对应类型的UserPO列表 |
| UserDataService.insert | 语法 | public void insert(UserPO userPO) throws RemoteException; |
| 前置条件 | 同样的username的po在Mapper中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| UserDataService.delete | 语法 | public void delete(UserPO userPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 删除一个po |
| UserDataService.update | 语法 | public void update(UserPO userPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 更新一个po |
| UserDataService.getList | 语法 | public ArrayList<UserPO> getList() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 返回所有UserPO的列表 |
| UserDataService.addCredit | 语法 | public void addCredit(String username, double amount) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 持久化增加对应username的客户的信用值 |
| UserDataService.deductCredit | 语法 | public void deductCredit(String username, double amount) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 持久化扣除对应username的客户的信用值 |
| UserDataService.finish | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |
| UserDataService.insertMember | 语法 | public void insertMember(MemberPO memberPO) throws RemoteException; |
| 前置条件 | 同样的username的po在Mapper中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| UserDataService.deleteMember | 语法 | public void deleteMember(MemberPO memberPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样username的po |
| 后置条件 | 删除一个po |

### PersonnelDataService模块

（1）模块概述

PersonnelDataService模块负责工作人员的增删改查等操作。

（2）模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PersonnelDataService.canLogin | 语法 | public boolean canLogin(PersonnelPO personnelPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样personnelname的po |
| 后置条件 | 返回对应personnelname的用户的是否可登录属性值 |

### HotelDataService模块

（1）模块概述

HotelDataService模块将负责酒店的增删改查，客房更新等操作。

（2）模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| HotelDataService.init | 语法 | public void init() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 初始化持久化数据库 |
| HotelDataService.findByHotelID | 语法 | public HotelPO findByHotelID(long hotelID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的HotelPO结果 |
| HotelDataService.findByHotelPO | 语法 | public ArrayList<HotelPO> findByHotelPO(HotelPO hotelPO) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按提供的HotelPO进行查找返回相应的HotelPO结果列表 |
| HotelDataService.insert | 语法 | public void insert(HotelPO hotelPO) throws RemoteException; |
| 前置条件 | 同样的ID的po在Mapper中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| HotelDataService.delete | 语法 | public void delete(HotelPO hotelPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 删除一个po |
| HotelDataService.update | 语法 | public void update(HotelPO hotelPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| HotelDataService.findRooms | 语法 | public ArrayList<RoomPO> findRooms(long HotelID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按提供的HotelID进行查找返回相应的RoomPO结果列表 |
| HotelDataService.finish | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |

### OrderDataService模块

（1）模块概述

OrderDataService模块负责保存订单的相关信息，供读取、保存、修改。

（2）模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务 | | |
| OrderDataService.init | 语法 | public void init() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 初始化持久化数据库 |
| OrderDataService.findByOrderID | 语法 | public OrderPO findByOrderID(long orderID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的OrderPO结果 |
| OrderDataService.find | 语法 | public ArrayList<OrderPO> findByType(OrderPO orderPO) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 返回相应条件下的ArrayList<OrderPO>结果 |
| OrderDataService.insert | 语法 | public void insert(OrderPO orderPO) throws RemoteException; |
| 前置条件 | 同样ID的po在Mapper中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| OrderDataService.update | 语法 | public void update(OrderPO orderPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| OrderDataService.getList | 语法 | public ArrayList<OrderPO> getList() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 返回ArrayList<OrderPO>结果 |
| OrderDataService. findEvaluationListByHotelID | 语法 | public ArrayList<EvaluationPO> findEvaluationListByHotelID(long hotelID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按hotelID进行查找返回相应的EvaluationPO结果 |
| OrderDataService.finish | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |

### PromotionDataService模块

（1）模块概述

PromotionDataService模块用来保存促销策略的的相关信息，供读取、保存、修改。

（2）模块内部接口规范

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionDataService.init | 语法 | public void init() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 初始化持久化数据库 |
| PromotionDataService.findByHotelID | 语法 | public ArrayList<PromotionPO> findByHotelID(long hotelID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按hotelID进行查找返回相应的ArrayList<PromotionPO>结果 |
| PromotionDataService.findByPromotionType | 语法 | public ArrayList<PromotionPO> findByPromotionType(PromotionType type) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按PromotionType进行查找返回相应的ArrayList<PromotionPO>结果 |
| PromotionDataService.findByPromotionID | 语法 | public PromotionPO findByPromotionID(long promotionID) throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 按ID进行查找返回相应的PromotionPO结果 |
| PromotionDataService.insert | 语法 | public void insert(PromotionPO promotionPO) throws RemoteException; |
| 前置条件 | 同样ID的po在Mapper中不存在 |
| 后置条件 | 在数据库中增加一个po记录 |
| PromotionDataService.delete | 语法 | public void delete(PromotionPO promotionPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 删除一个po |
| PromotionDataService.update | 语法 | public void update(PromotionPO promotionPO) throws RemoteException; |
| 前置条件 | 在数据库中存在同样ID的po |
| 后置条件 | 更新一个po |
| PromotionDataService.finish | 语法 | public void finish() throws RemoteException; |
| 前置条件 | 无 |
| 后置条件 | 结束持久化数据库的使用 |

# 依赖视角

下图分别为客户端和服务器端各自的包之间的联系：

