酒店管理系统

Management System of Hotels

详细设计描述文档

V1.1

南京大学软件学院 Octopus 小组

成员：周沁涵、桑田、钱柯宇、潘潇睿

2016-10-26

目录

更新历史

修改人员	日期	变更原因	版本号
全体成员	2016-10-26	最初草稿	V0.0.0 草稿
钱柯宇、潘潇睿 桑田	2016-10-28	初步完成除 Manager 模块和 Marketer 模块之外的所有模块	V1.0.0
全体成员	2016-11-2	重新撰写、分布了 controller 模块，在原有文档基础上进行了内容的重构	V1.1.0
全体成员	2016-11-6	对文档所有模块进行细节完善和缺失添加	V1.1.1

1. 引言

1.1 编制目的

本报告详细完成对酒店管理系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2 词汇表

词汇名称	词汇含义	备注
_ui	表示某展示层	
_bl	表示某逻辑层	
_data	表示某数据层	

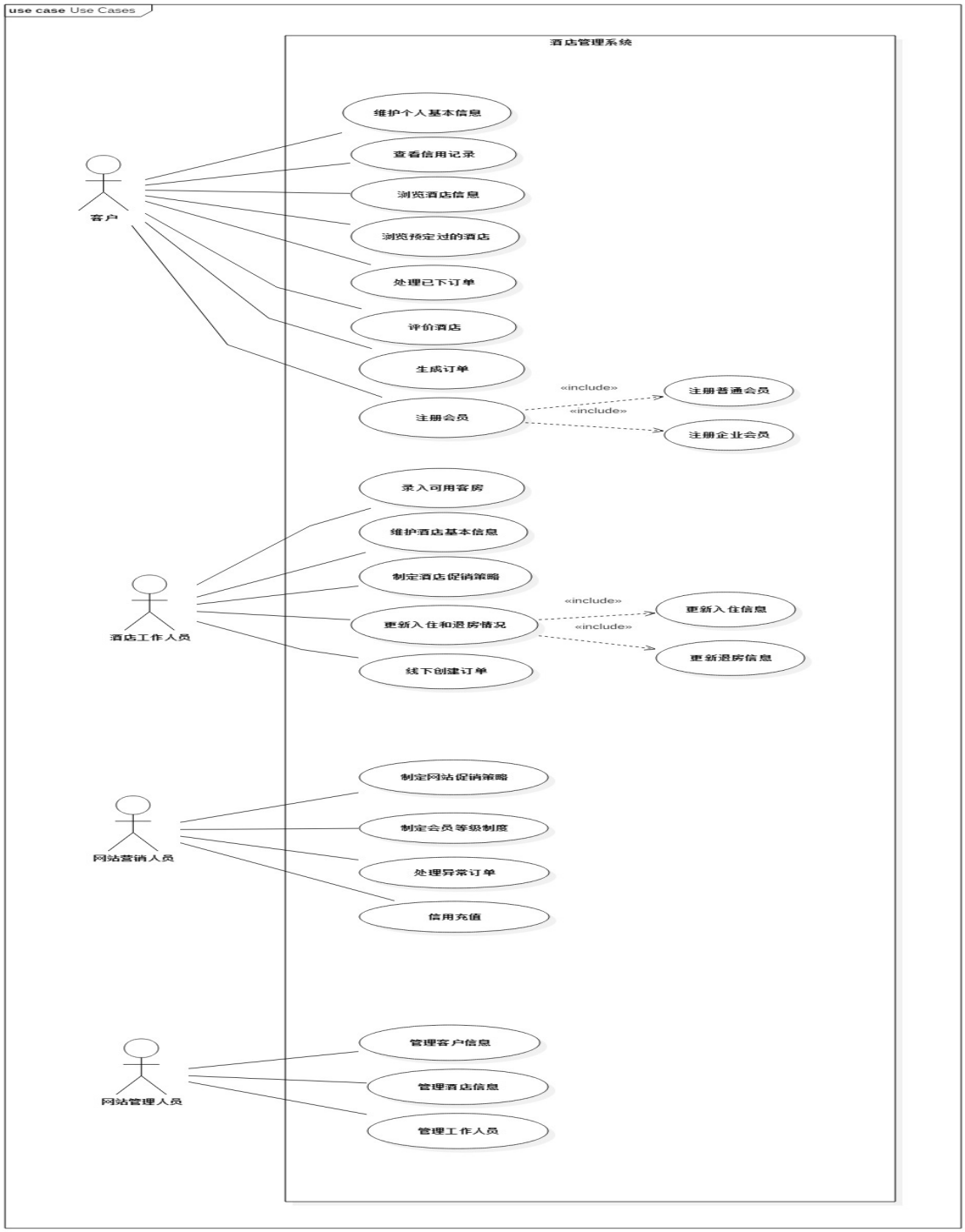
1.3 参考资料

1. IEEE std 1471-2000

2. 丁二玉，刘钦.计算与软件工程（卷二）[M]机械工业出版 2012：134—182

2. 产品概述

参考酒店管理系统用例文档和酒店管理系统软件需求规格说明文档中队产品的概括描述。酒店管理系统主要是应用于建立线上酒店预订系统，并添加营销策略与管理,主要功能见用例图如下。



3. 体系结构设计概述

酒店管理系统中，选择了分层体系结构风格，将系统分为 3 层(展示层、业务逻辑层、数据层)能够很好地示意整个高层抽象。展示层包含 GUI 页面的实现 ,业务逻辑层包含业务逻辑处理的实现，

数据层负责数据的持久化和访问。

4. 结构视角

4.1 Login_bl 模块

（1）模块概述

Login_bl 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。

Login_bl 模块的职责及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，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 Login_blservice 接口。为了隔离业务逻辑职责和逻辑控制职责，我们添加了 LoginController，这样 LoginController 将会将用户管理相关的业务逻辑职责和逻辑控制委托给 Login_bl 对象。

Login_bl 模块的设计如图 4.1.1 所示。

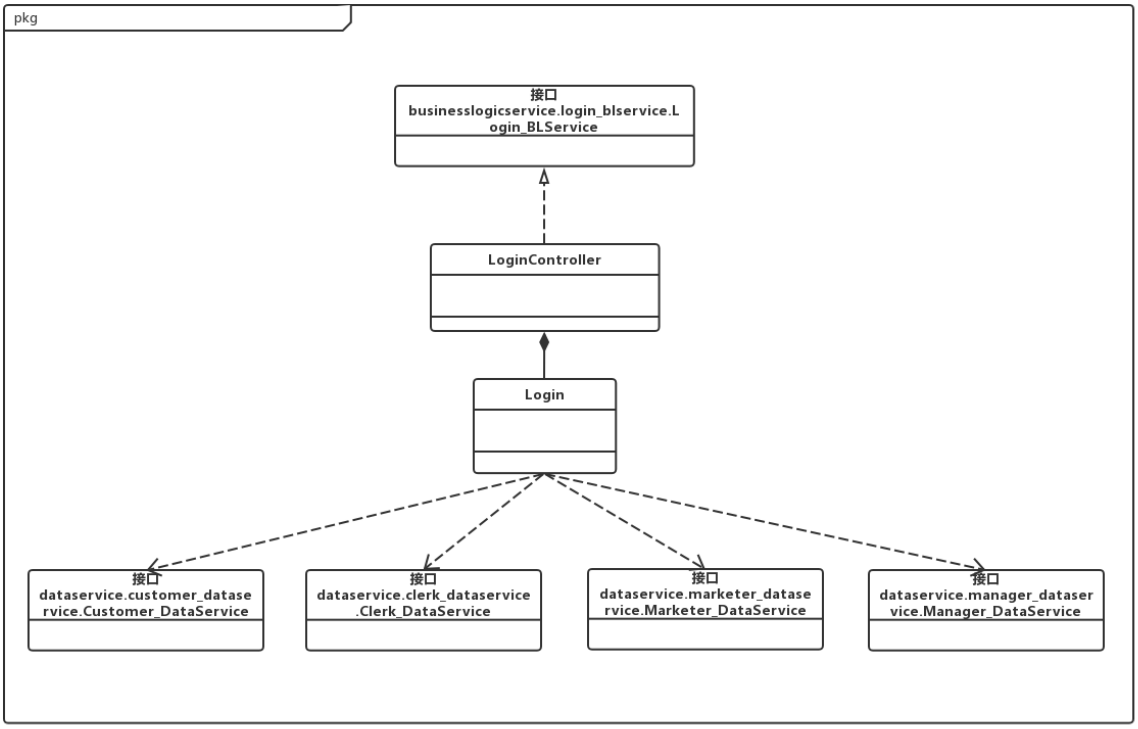


图 4.1.1 Login_bl 模块各个类的设计

Login_bl 模块各个类的职责如表 4.1.2 所示。

表 4.1.2 Login_bl 模块各个类的职责

模块	职责
LoginController	负责实现对应于登录界面所需的方法
Login	负责处理登陆相关的方法

（3）模块内部类的接口规范

LoginController 的接口规范如表 4.1.3 所示。

表 4.1.3 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)
	前置条件	已创建一个 Login 领域对象，并且已知账号和密码
	后置条件	调用 Login 领域对象的 login 方法

需要的服务（需接口）	
服务名	服务
Login.login	客户登录

Login 的接口规范如表 4.1.4 所示。

表 4.1.4 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)
	前置条件	启动登陆服务
	后置条件	查找账号密码是否对应，返回登录信息（登录是否成功）
需要的服务（需接口）		
服务名	服务	
Customer_data_service.find(String ID)	根据 ID 查找用户	
Clerk_data_service.find(String ID)	根据 ID 查找酒店工作人员	
Marketer_data_service.find(String ID)	根据 ID 查找网站营销人员	
Manager_data_service.find(String ID)	根据 ID 查找网站管理人员	

（4）Login_bl 模块的动态模型

图 4.1.5 表明了酒店管理系统中，当用户输入用户名和密码登录时，登录逻辑处理的相关对象之间的协作。

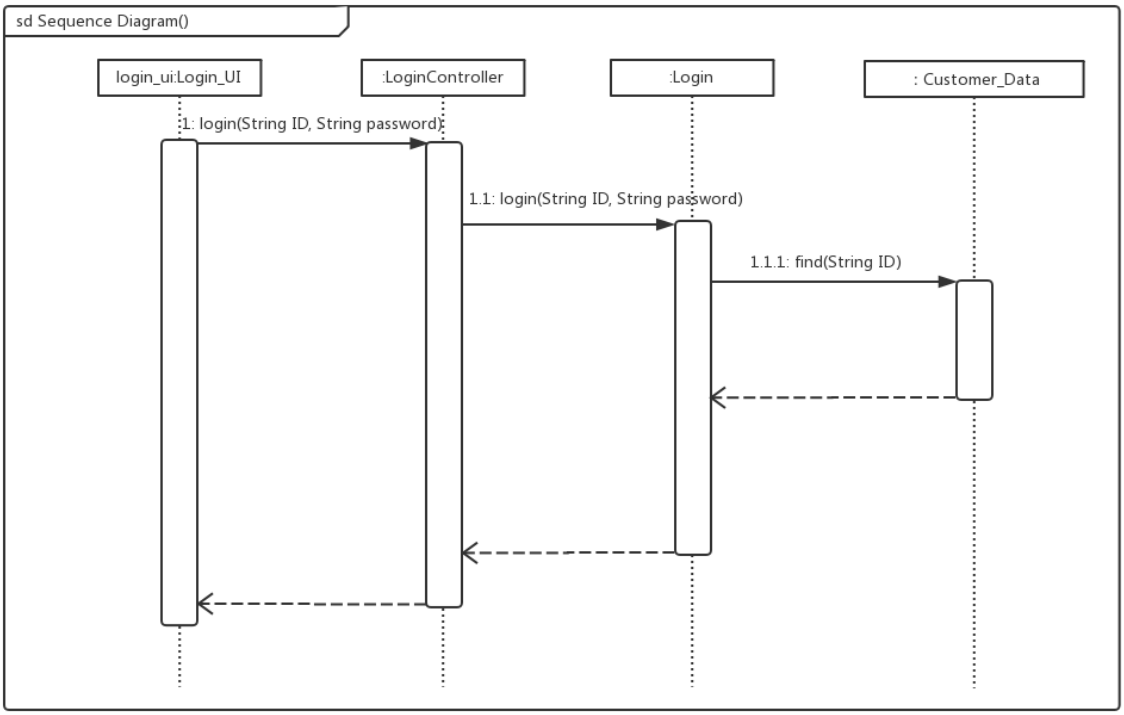


图 4.1.5 用户登录的顺序图

如图 4.1.6 所示的状态图描述了 Login 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 login 方法被 UI 调用，Login 进入 Logged in 状态。

图 4.1.6 Login 对象状态图

（5）Login_bl 模块的设计原理

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

4.2 Customer_bl 模块

（1）模块概述

Customer_bl 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。Customer_bl 模块的职责及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，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 Customer_blservice、Customer_data_service 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 CustomerSignUpController，CustomerInfoChangeController，CreditRecordController，ReservedHotelController，这样 Controller 将会将用户管理相关的业务逻辑职责和逻辑控制委托给 Customer_bl 对象。

Customer_bl 模块的设计如图 4.2.1 所示。

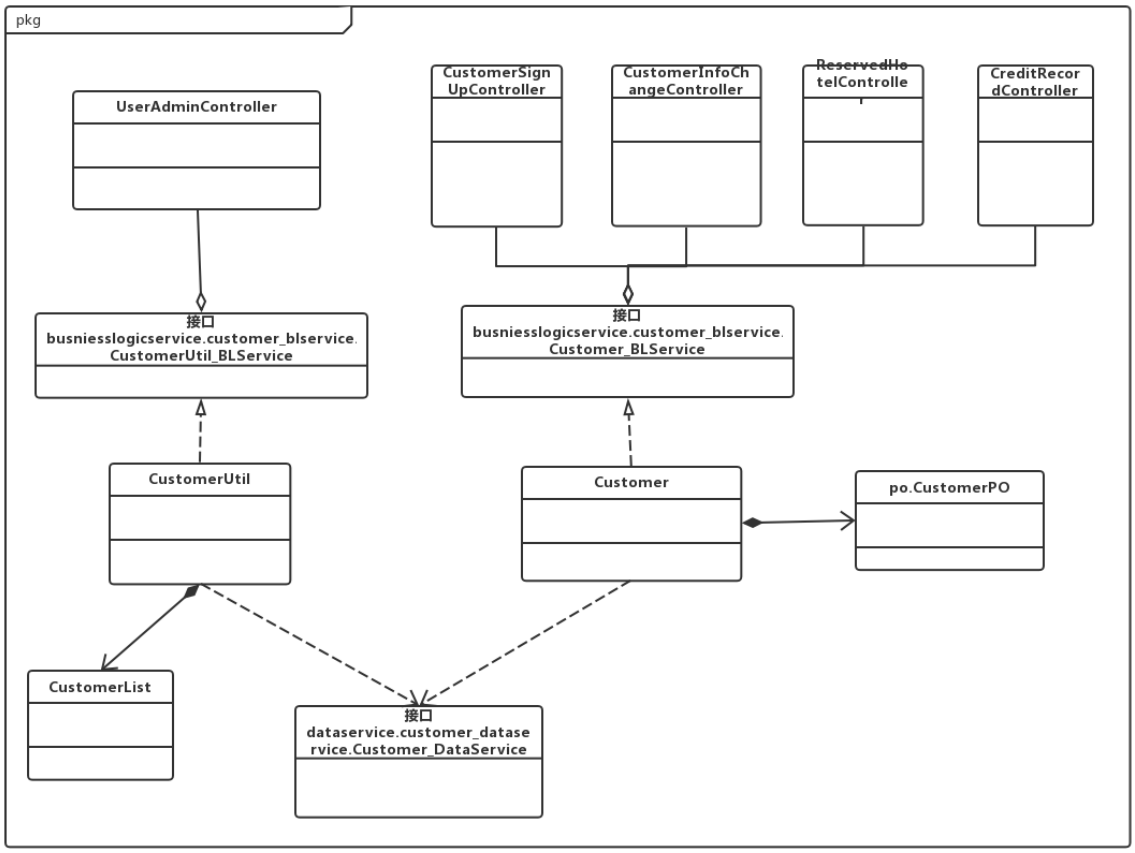


图 4.2.1 Customer_bl 模块各个类的设计

Customer_bl 模块各个类的职责如表 4.2.2 所示。

表 4.2.2 Customer_bl 模块各个类的职责

模块	职责
CustomerSignUpController	负责处理客户注册的功能
CustomerInfoChangeController	负责处理客户维护个人信息的功能
CreditRecordController	负责处理客户查看信用记录的功能
ReservedHotelController	负责处理客户查看预定过的酒店的功能
CustomerUtil	负责批量处理客户
Customer	负责处理有关客户的功能

（3）模块内部类的接口规范

CustomerSignUpController 的接口规范如表 4.2.3 所示。

表 4.2.3 CustomerSignUpController 的接口规范

提供的服务（供接口）

CustomerSignUpController.signUp	语法	public ResultMessage signUp(CustomerVO customerVO)
	前置条件	已创建一个 Customer 领域对象和一个 CustomerUtil 领域对象，且必要信息全都按规范填写
	后置条件	调用 Customer 领域对象的 signUp 方法，调用 CustomerUtil 领域对象的 getSingle 方法
需要的服务（需接口）		
服务名	服务	
Customer.signUp	客户进行注册	
CustomerUtil.getSingle	查找对应客户	

CustomerInfoChangeController 的接口规范如表 4.2.4 所示。

表 4.2.4 CustomerInfoChangeController 的接口规范

提供的服务（供接口）		
CustomerInfoChangeController.changeInfo	语法	public ResultMessage changeInfo(CustomerVO customerVO)
	前置条件	已创建一个 Customer 领域对象，且客户已登录，修改的信息格式符合规范
	后置条件	调用 Customer 领域对象的 changeInfo 方法
CustomerInfoChangeController.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1, String newPw2);
	前置条件	已创建一个 Customer 领域对象，且客户已登录
	后置条件	调用 Customer 领域对象的 changePassword 方法
需要的服务（需接口）		
服务名	服务	

Customer.changeInfo	客户维护个人信息
Customer.changePassword	客户修改密码

CreditRecordController 的接口规范如表 4.2.5 所示。

表 4.2.5 CreditRecordController 的接口规范

提供的服务（供接口）		
CreditRecordController.getCreditRecord	语法	public CreditRecordVO getCreditRecord (CustomerVO customerVO)
	前置条件	已创建一个 Customer 领域对象
	后置条件	调用 Customer 领域对象的 getCreditRecord 方法
CreditRecordController.addCreditRecord	语法	public ResultMessage addCreditRecord(String ID, CreditRecordVO vo)
	前置条件	已创建一个 Customer 领域对象
	后置条件	调用 Customer 领域对象的 addCreditRecord 方法
需要的服务（需接口）		
服务名	服务	
Customer.getCreditRecord	提供对应客户的信用记录	

ReservedHotelController 的接口规范如表 4.2.6 所示。

表 4.2.6 ReservedHotelController 的接口规范

提供的服务（供接口）		
ReservedHotelController.getHistoryHotel	语法	public List<HotelVO> getHistoryHotel (CustomerVO customerVO)；
	前置条件	已创建一个 Customer 领域对象
	后置条件	调用 Customer 领域对象的 getHistoryHotel 方法

需要的服务（需接口）	
服务名	服务
Customer.getHistoryHotel	提供对应客户的预定过的酒店

CustomerUtil 的接口规范如表 4.2.7 所示。

表 4.2.7 CustomerUtil 的接口规范

提供的服务（供接口）		
CustomerUtil.getSingle	语法	public CustomerVO getSingle(String ID)
	前置条件	发起根据 ID 查找单个客户
	后置条件	返回该 ID 的 Customer
CustomerUtil.getByName	语法	public List<CustomerVO> getByName(String name)
	前置条件	发起根据姓名查找单个客户
	后置条件	返回改姓名的 Customer
CustomerUtil.getAll	语法	public List<CustomerVO> getAll()
	前置条件	发起查找所有客户
	后置条件	返回所有的 Customer 信息

需要的服务（需接口）	
服务名	服务
Customer_data_service.findByID(String ID)	根据 ID 查找用户
Customer_data_service.findByID(String ID)	根据姓名查找用户
Customer_data_service.	返回所有用户

Find()	

Customer 的接口规范如表 4.2.8 所示。

表 4.2.8 Customer 的接口规范

提供的服务（供接口）		
Customer.signUp	语法	public ResultMessage signUp(CustomerVO customerVO)
	前置条件	发起客户注册
	后置条件	查找是否存在相应的 ID、手机、邮箱，并返回注册的结果，如果注册成功，则新增持久化对象
Customer.changeInfo	语法	public ResultMessage changeInfo(CustomerVO customerVO)
	前置条件	发起客户维护个人信息
	后置条件	返回修改是否成功的结果，如果成功，则修改信息
Customer.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1, String newPw2)
	前置条件	发起客户修改密码
	后置条件	返回修改是否成功的结果
Customer.getCreditRecord	语法	public CreditRecordVO getCreditRecord (CustomerVO customerVO)
	前置条件	发起查看信用记录
	后置条件	返回该用户的信用记录
Customer.getHistoryHotel	语法	public List<HotelVO> getHistoryHotel

		(CustomerVO customerVO)
	前置条件	发起查看预订过的酒店
	后置条件	返回该用户的预定过的酒店
Customer.getCredit	语法	public double getCredit(CustomerVO customerVO)
	前置条件	发起查看某个客户的当前信用值
	后置条件	返回该用户的当前信用值
Customer.addCreditRecord	语法	public ResultMessage addCreditRecord(String ID, CreditRecordVO vo)
	前置条件	发起增加一条信用记录
	后置条件	返回增加信用记录的结果，如果成功，则增加一条信用记录

需要的服务（需接口）	
服务名	服务
Customer_data_service.add(CustomerPO po)	新增单一持久化对象
Customer_data_service.modify(CustomerPO po)	修改单一持久化对象
Customer_Data_Service.getReservedHotel(String user_id)	获取用户预订过的酒店名列表
Customer_Data_Service.addCreditRecord(CreditR	新增信用记录对象

ecordPO crPO)	

（4）Customer_bl 模块的动态模型

图 4.2.8 表明了酒店管理系统中，当客户注册时，客户逻辑处理的相关对象之间的协作。

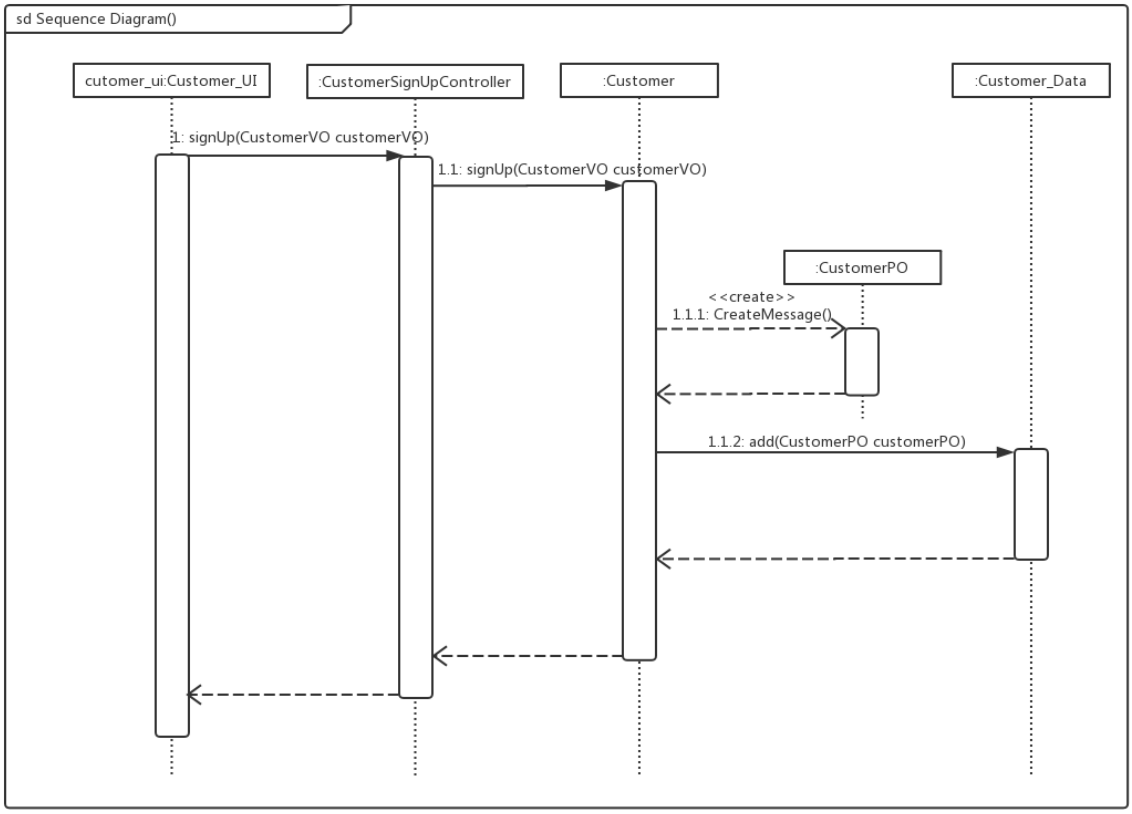


图 4.2.8 客户注册的顺序图

图 4.2.9 表明了酒店管理系统中，当客户维护个人信息时，客户逻辑处理的相关对象之间的协作。

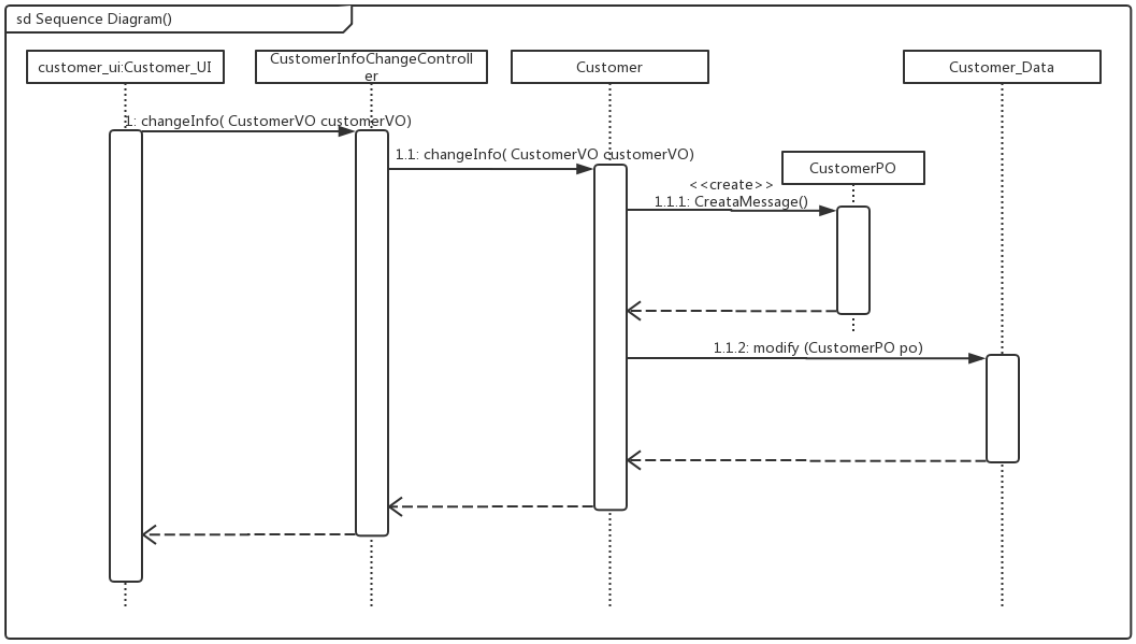


图 4.2.9 客户维护个人信息的顺序图

如图 4.2.10 示的状态图描述了 Customer 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 signUp 方法被 UI 调用，Customer 进入 signUp 状态。随着 getSingle 方法被 UI 调用，Customer 进入 find 状态，随着 getAll 方法被 UI 调用，Customer 进入 getAll 状态，随着 changeInfo、changePassword 方法被 UI 调用，Customer 进入 modify 状态等。

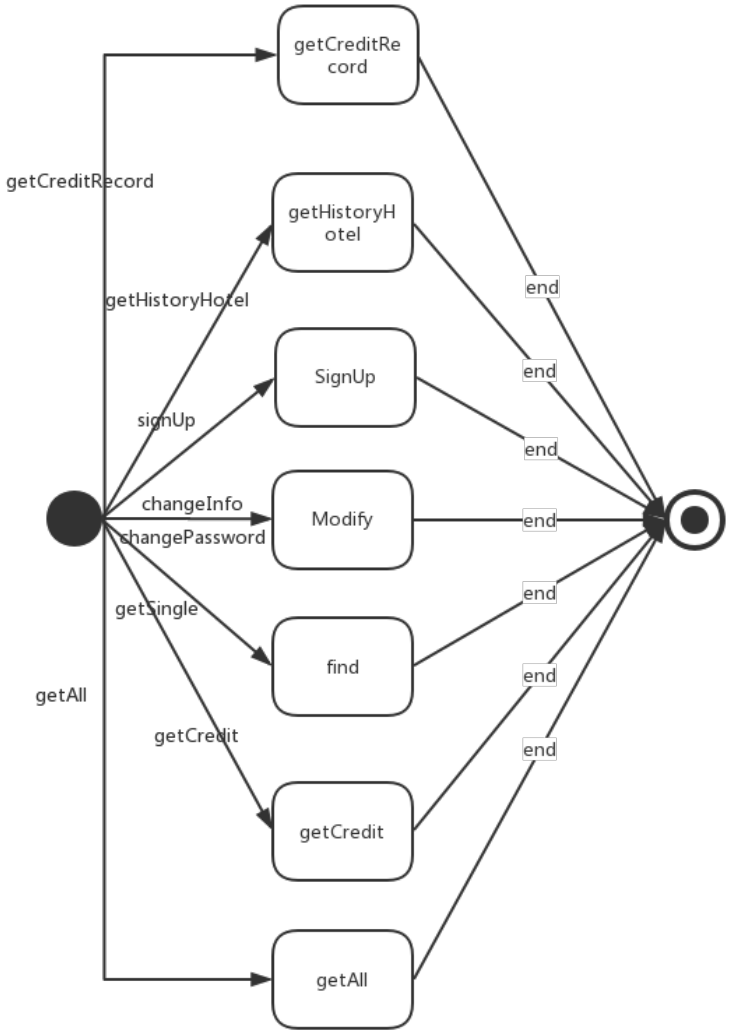


图 4.2.10 Customer 对象状态图

（5）Customer_bl 模块的设计原理

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

4.3 Clerk_bl 模块

（1）模块概述

Clerk_bl 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。Clerk_bl 模块的

职责及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，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 Clerk_blservice、Clerk_data_service 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 ClerkController，这样 ClerkController 将会将用户管理相关的业务逻辑职责和逻辑控制委托给 Clerk_bl 对象。

Clerk_bl 模块的设计如图 4.2.1 所示。

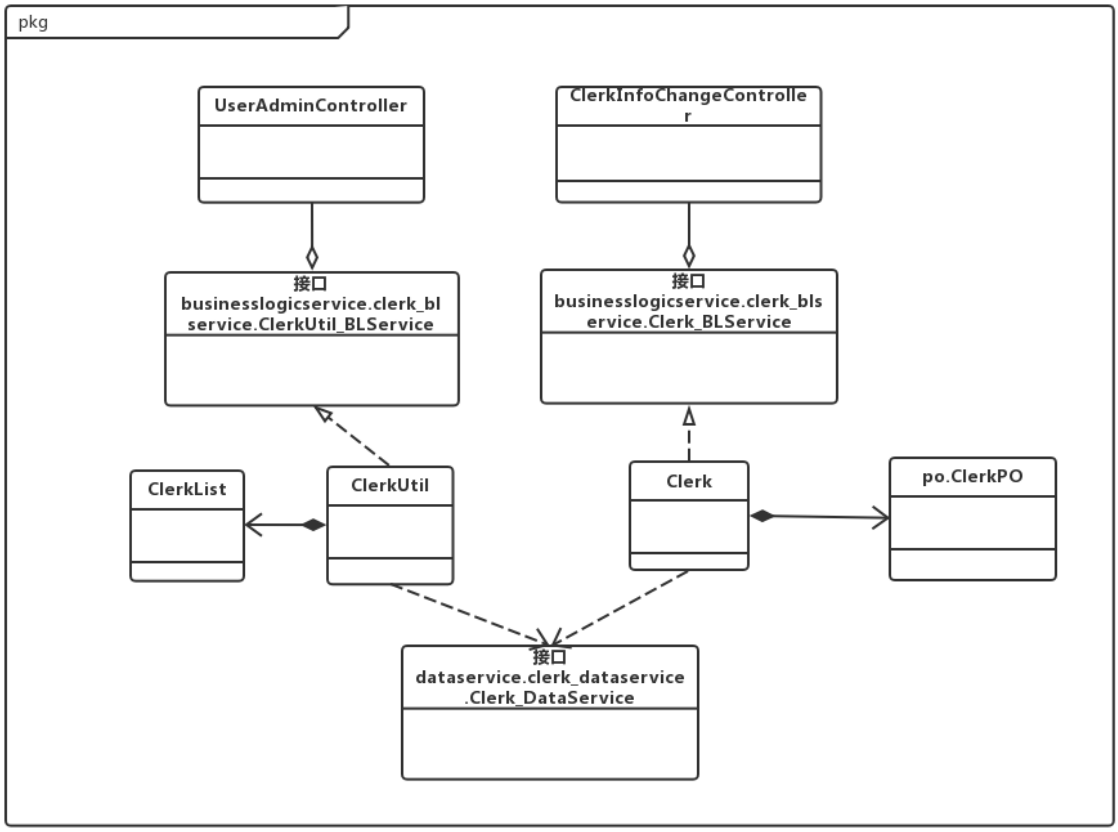


图 4.3.1 Clerk_bl 模块各个类的设计

Clerk_bl 模块各个类的职责如表 4.3.2 所示。

表 4.3.2 Clerk_bl 模块各个类的职责

类	职责
ClerkInfoChangeController	负责实处理酒店工作人员维护个人信息的功能
ClerkUtil	负责批量处理酒店工作人员
Clerk	负责处理有关酒店工作人员的功能

（3）模块内部类的接口规范

ClerkInfoChangeController 的接口规范如表 4.3.3 所示。

表 4.3.3 ClerkInfoChangeController 的接口规范

提供的服务（供接口）		
ClerkInfoChangeController.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1. String newPw2)
	前置条件	已创建一个 Clerk 领域对象，且酒店工作人员已登录
	后置条件	调用 Clerk 领域对象的 changePassword 方法
ClerkInfoChangeController.changeInfo	语法	public ResultMessage changeInfo(ClerkVO clerkVO)
	前置条件	已创建一个 Clerk 领域对象，且酒店工作人员已登录，修改的信息格式符合规范
	后置条件	调用 Clerk 领域对象的 changeInfo 方法
需要的服务（需接口）		
服务名	服务	
Clerk.changeInfo	酒店工作人员维护个人信息	
Clerk.changePassword	酒店工作人员修改密码	

ClerkUtil 的接口规范如表 4.3.4 所示。

表 4.3.4 ClerkUtil 的接口规范

提供的服务（供接口）		
ClerkUtil.getSingle	语法	public ClerkVO getSingle(String ID)
	前置条件	发起根据 ID 查找单个用户
	后置条件	返回该 ID 的 Clerk 信息
ClerkUtil.getByName	语法	public List<ClerkVO> getByName(String name)
	前置条件	发起根据姓名查找单个用户
	后置条件	返回该 name 的 Clerk 信息
ClerkUtil.getAll	语法	public List<ClerkVO> getAll()
	前置条件	发起查找所有酒店工作人员
	后置条件	返回所有的 Clerk 信息
需要的服务（需接口）		
服务名	服务	
Clerk_Data_Service.findBy ClerkID (String ID)	根据 ID 查找工作人员	
Clerk_Data_Service.findBy Clerk (String name)	根据姓名查找工作人员	
Clerk_Data_Service.find ()	返回所有工作人员	

Clerk 的接口规范如表 4.3.5 所示。

表 4.3.5 Clerk 的接口规范

提供的服务（供接口）		
Clerk.changeInfo	语法	public ResultMessage changeInfo(ClerkVO clerkVO);

	前置条件	发起酒店工作人员维护个人信息
	后置条件	返回修改是否成功的结果
Clerk.changePassword	语法	public ResultMessage changePassword(String ID, String oldPW, String newPW1. String newPW2);
	前置条件	发起酒店工作人员修改密码
	后置条件	更改酒店工作人員密碼
Clerk.addClerk	语法	pulic ResultMessage addClerk(ClerkVO clerkVO)
	前置条件	发起增加一个酒店工作人员
	后置条件	返回添加工作人员的操作结果
Clerk.deleteClerk	语法	public ResultMessage deleteClerk(ClerkVO clerkVO)
	前置条件	发起删除某个工作人员
	后置条件	返回删除工作人员的操作结果
需要的服务（需接口）		
服务名	服务	
Clerk_Data_Service.modify(ClerkPO po)	修改单一持久化对象	
Clerk_Data_Service.add(ClerkPO clerkPO)	新增单一持久化对象	
Clerk_Data_Service.delete(ClerkPO clerkPO)	删除对应工作人员	

（4）Clerk_bl 模块的动态模型

图 4.3.6 表明了酒店管理系统中，酒店工作人员修改密码时，酒店工作人员逻辑处理的相关对象之间的协作。

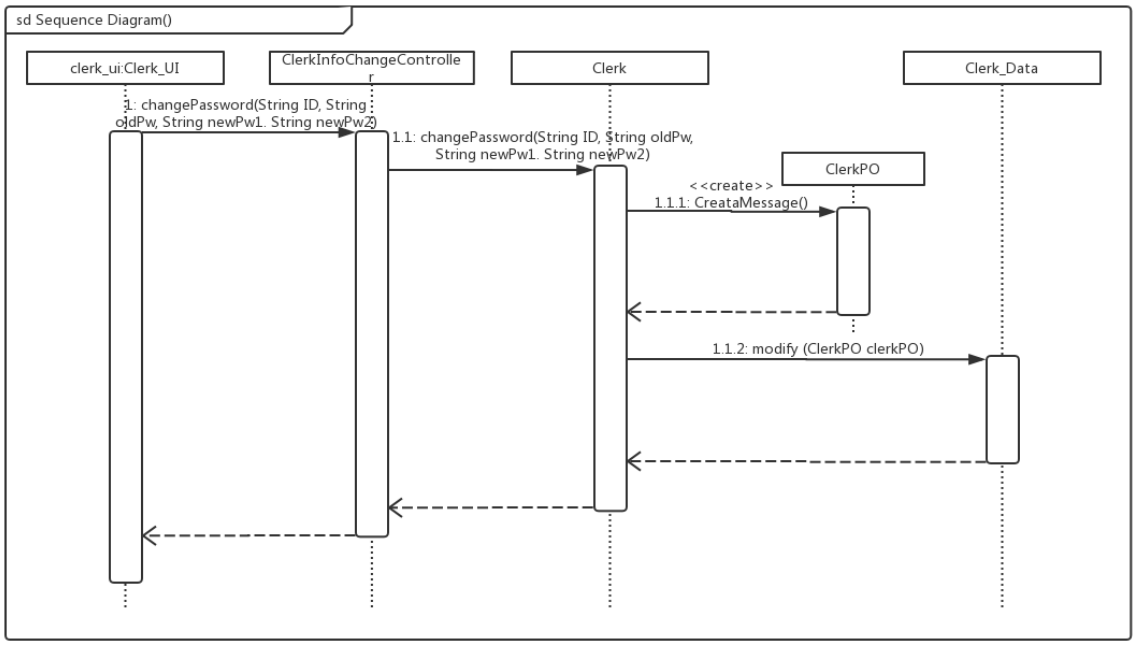


图 4.3.6 酒店工作人员修改密码的顺序图

如图 4.3.7 所示的状态图描述了 Clerk 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 getSingleByID、getSingleByName 方法被 ui 调用，Clerk 进入 find 状态，随着 getAll 方法被 UI 调用，Clerk 进入 getAll 状态，随着 changeInfo、changePassword 方法被 UI 调用，Clerk 进入 modify 状态。

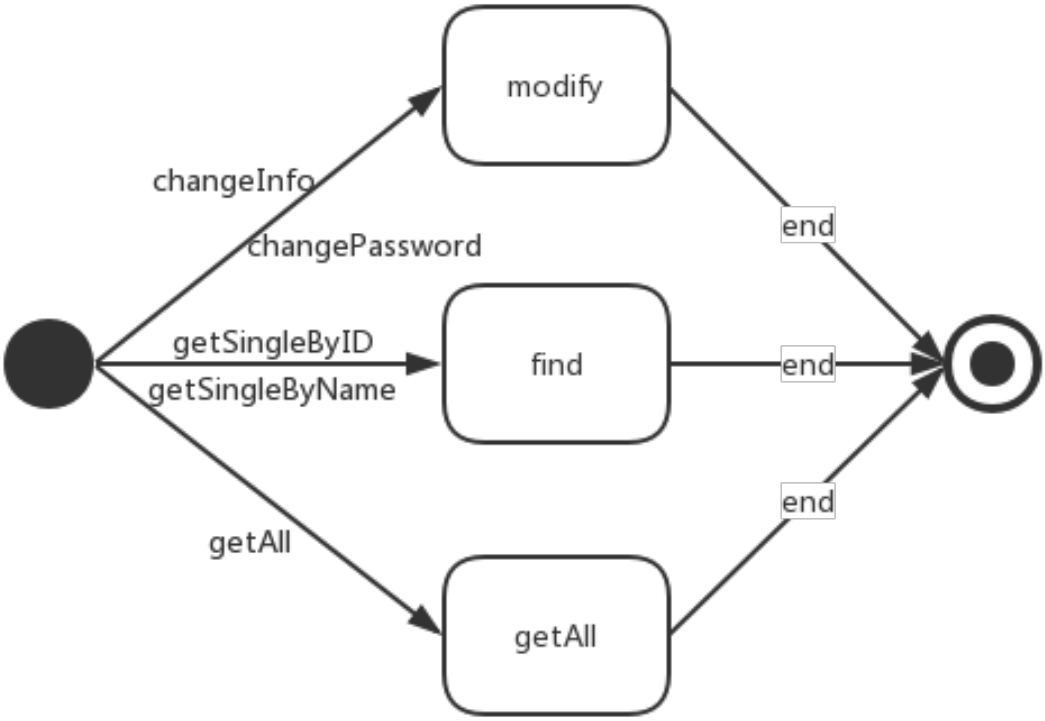


图 4.3.7 Clerk 对象状态图

（5）Clerk_bl 模块的设计原理

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

4.4 Order_bl 模块

（1）模块概述

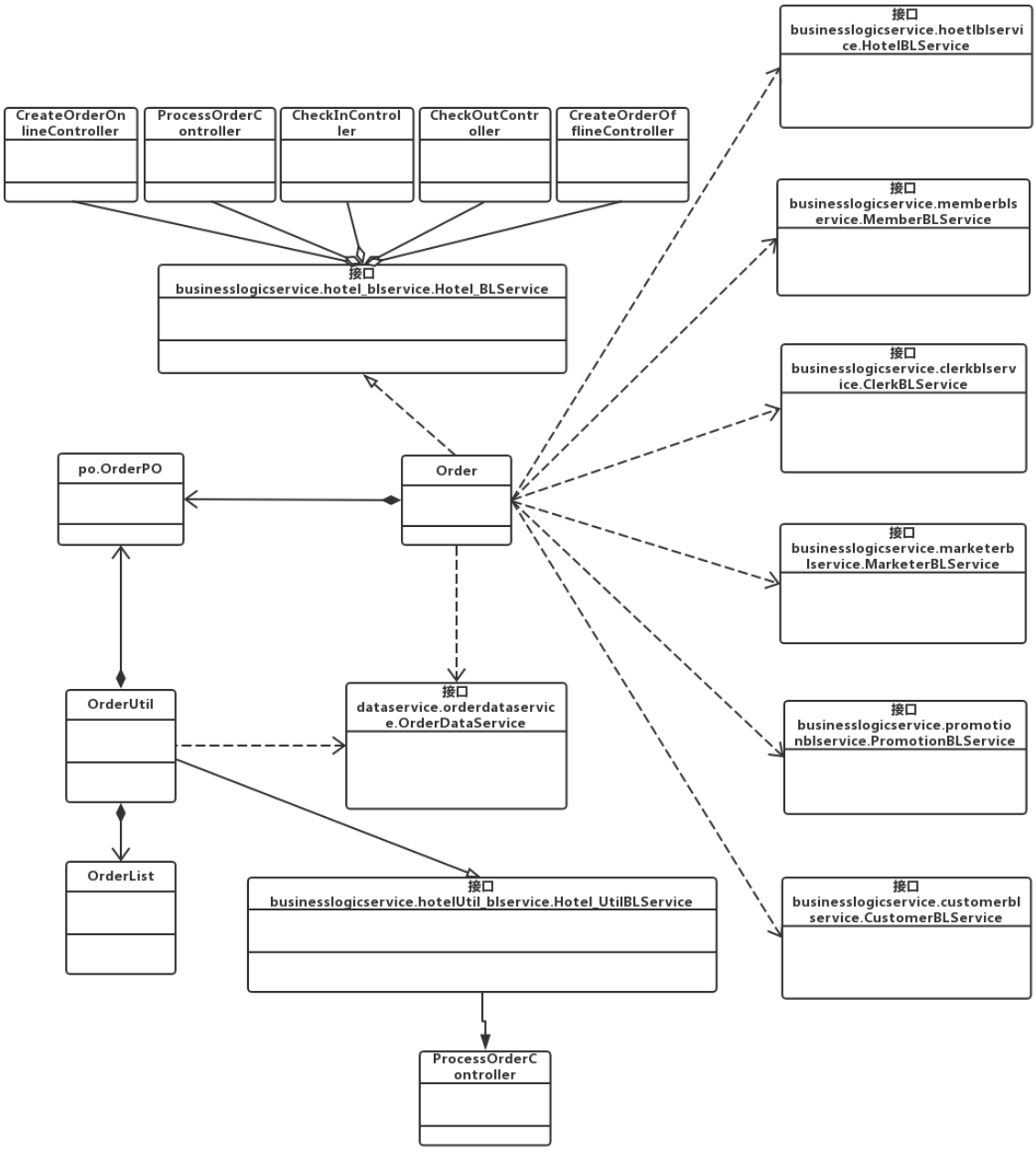
Order_bl 模块承担的需求参见需求规格说明文档功能需求及相关非功能需求。

Order_bl 模块的职责及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，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 Order_blservice、Order_data_service,OrderUtil_blservice 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 CreateOrderOnlineController,CreateOrderOfflineController，ProcessOrderController,CheckInController,CheckOutController, 这样 CreateOrderOnlineController 将会将线上创建订单相关的业务逻辑职责和逻辑控制委托给 Order_bl 对象，CreateOrderOfflineController 将会将线下创建订单相关的业务逻辑职责和逻辑控制委托给 Order_bl 对象，ProcessOrderController 将会将处理订单相关的业务逻辑职责和逻辑控制委托给 Order_bl 对象，CheckInController 将会将用户入住的业务逻辑职责和逻辑控制委托给 Order_bl 对象，CheckOutController 将会将用户退房相关的业务逻辑职责和逻辑控制委托给 Order_bl 对象

Order_bl 模块的设计如图 4.4.1 所示。



Order 模块各个类的职责如表 4.4.2 所示。

表 4.4.2 Order 模块各个类的职责

模块	职责
Order	订单的领域模型对象，负责处理订单方面的功能
CreateOrderOnlineController	线上创建订单的控制器，负责调用创建订单方法
ProcessOrderController	处理订单的控制器，负责调用创建订单方法
CreateOrderOfflineController.	线下创建订单的控制器，负责调用创建订单方法
CheckInController	用户入住的控制器，负责调用执行订单方法
CheckOutController	用户退房的控制器，负责调用结束订单方法
OrderUtil	负责得到、搜索订单，以及订单的排序方法

（3）模块内部类的接口规范

CreateOrderOnlineController 的接口规范如表 4.4.3 所示。

表 4.4.3 CreateOrderOnlineController 的接口规范

提供的服务（供接口）		
CreateOrderOnlineController.createOrder	语法	public ResultMessage createOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，并且输入的订单信息正确
	后置条件	调用 Order 邻域对象的 createOrder()方法，调用 Customer 邻域对象的 getCredit()方法，调用 Hotel 邻域对象的 changeReservedRoom()方法，调用 HotelUtil 邻域对象的 getDailyRommInfo(),getRoom 方法，调用 Promotion 邻域对象的 getAll()方法，调用 Hotel 领域对象的 addToListOfHotelReservedByCustomer 方法
需要的服务（供接口）		
Order.createOrder	新建订单	
Customer.getCredit	获取用户信用值	
Hotel.changeReservedRoom	更改预定过的房间数量	
HotelUtil.getDailyRoomInfo	获取酒店当日房间信息	
HotelUtil.getRoom	获取酒店房间信息	

Promotion.getAll	获取所有促销策略
Hotel. addToListOfHotelReserve dByCustomer	将酒店添加至客户的已预订列表

ProcessOrderController 的接口规范如表 4.4.4 所示。

表 4.4.4 ProcessOrderController 的接口规范

提供的服务（供接口）		
ProcessOrderController.c ancelOrder	语法	public ResultMessage cancelOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，并且订单状态为未执行
	后置条件	调用 Order 邻域对象的 cancelOrder()方法，调用 Customer 邻域对象的 addCredit()方法，调用 Hotel 邻域对象的 changeAvailableRoom(),changeReservedRoom()方法
ProcessOrderController.s etAbnormal	语法	public ResultMessage setAbnormal(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，用户未按时入住
	后置条件	调用 Order 邻域对象的 setAbnormal()方法，调用 Customer 邻域对象的 addCredit()方法，调用 Hotel 邻域对象的 changeAvailableRoom(),changeReservedRoom()

		方法
ProcessOrderController.renewOrder	语法	public ResultMessage renewOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，订单状态为异常，用户已申诉
	后置条件	调用 Order 邻域对象的 renewOrder()方法，调用 Customer 邻域对象的 addCredit()方法
ProcessOrderController.getOrderByStatus	语法	public List<OrderVO> getOrderByStatus(OrderSatus status)
	前置条件	已创建一个 OrderUtil 对象
	后置条件	调用 OrderUtil 对象的 searchOrderByStatus 方法
ProcessOrderController.getOrderByCustomerName	语法	public List<OrderVO> getOrderByCustomerName (String customerName)
	前置条件	已创建一个 OrderUtil 对象
	后置条件	调用 OrderUtil 对象的 searchOrderByCustomerName 方法
ProcessOrderController.getOrderByHotelID	语法	public List<OrderVO> getOrderByHotelName(String hotelID)
	前置条件	已创建一个 OrderUtil 对象
	后置条件	调用 Order_Util 对象的 searchOrderByHotelID 方法
需要的服务（供接口）		

Order.cancelOrder	取消订单
Order.setAbnormal	设为异常订单
Order.renewOrder	恢复异常订单
Order.changeStatus	更改订单状态
OrderUtil.getOrderByStatus	按订单状态查找订单
OrderUtil.getOrderByCustomerName	按客户名称查找订单
OrderUtil.getOrderByHotelID	按酒店 ID 查找订单
Customer.addCreditRecord	增加用户信用记录
Hotel.changeReservedRoom	更改预定过的房间数量
Hotel.changeAvailableRoom	更改空闲房间数量

CheckInController 的接口规范如表 4.4.5 所示。

表 4.4.5 CheckInController 的接口规范

提供的服务（供接口）		
CheckInController.executeOrder	语法	public ResultMessage executeOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，用户已入住，并且订单信息

		输入正确
	后置条件	调用 Order 邻域对象的 executeOrder()方法，调用 Hotel 邻域对象的 changeOccupiedroom()方法，调用 Customer 邻域对象的 addCredit()方法
需要的服务（供接口）		
Order.executeOrder	执行订单	
Hotel.changeOccupiedRoom	更改已入住的房间数量	
Customer.addCreditRecord	增加用户信用记录	

CheckOutController 的接口规范如表 4.4.6 所示。

表 4.4.6 CheckOutController 的接口规范

提供的服务（供接口）		
CheckOutController.endOrder	语法	public ResultMessage endOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，用户已退房，并且订单信息输入正确
	后置条件	调用 Order 邻域对象的 endOrder()方法，调用 Hotel 邻域对象的 changeOccupiedRoom(),changeReservedRoom()方法
需要的服务（供接口）		

Order.endOrder	结束订单
Hotel.changeOccupiedRoom	更改已入住的房间数量
Hotel.changeReservedRoom	更改预定过的房间数量
Hotel.changeAvailableRoom	更改空闲房间数量

CreateOrderOfflineController 的接口规范如表 4.4.7 所示。

表 4.4.7 CreateOrderOfflineController 的接口规范

提供的服务（供接口）		
CreateOrderOfflineController.createOrder	语法	public ResultMessage createOrder(OrderVO orderVO)
	前置条件	已有一个 Order 邻域对象，并且输入的订单信息正确
	后置条件	调用 Order 邻域对象的 createOrder()方法，调用 HotelUtil 邻域对象的 getDailyRommInfo(),getRoom 方法，调用 Hotel 邻域对象的 changeOccupiedRoom(),changeAvailableRoom() 方法
需要的服务（供接口）		
Order.createOrder	新建订单	
HotelUtil.getDailyRoomInfo	获取酒店当日房间信息	

HotelUtil.getRoom	获取酒店房间信息
Hotel.changeOccupiedRoom	更改已入住的房间数量
Hotel.changeAvailableRoom	更改空闲房间数量

OrderUtil 的接口规范如表 4.4.8 所示。

表 4.4.8 OrderUtil 的接口规范

提供的服务（供接口）		
OrderUtil.getOrder	语法	public OrderVO getSingle(String ID)
	前置条件	无
	后置条件	根据订单号获取该订单
OrderUtil.getOrdersByCustomerID	语法	public List<OrderVO> getOrdersByCustomerID (String customerID)
	前置条件	无
	后置条件	用户 ID 存在，返回该用户的订单
OrderUtil.getOrdersByCustomerName	语法	public List<OrderVO> get getOrdersByCustomerID (String customerName)
	前置条件	无
	后置条件	用户名字存在，返回该用户的订单
OrderUtil.getOrdersByHotelID	语法	public List<OrderVO> getOrdersByHotelID (String customerID)
	前置条件	无

	后置条件	酒店 ID 存在，返回该用户的订单
OrderUtil.sortByTime	语法	public List<OrderVO> sortByTime(List<OrderVO>)
	前置条件	订单对象列表存在
	后置条件	返回排序后的订单列表
OrderUtil.getOrderByStatus	语法	public List<OrderVO> getOrderByStatus(OrderSatus status)
	前置条件	无
	后置条件	输入的酒店状态符合规范，返回订单列表
提供的服务（供接口）		
Order_Data_Service.find(String CustomerName)	按照用户名字查找订单	
Order_Data_Service.findByHotelID (String HotelID)	按照酒店 ID 查找订单	
Order_Data_Service.findByOrderStatus(OrderStatus status)	按照订单状态查找订单	
Order_Data_Service.findCustomerIDAndOrderStatus(String customerID, OrderStatus status)	按照用户 ID 及订单状态查找订单	
Order_Data_Service.find	获得所有订单	

Order 的接口规范如表 4.4.9 所示。

表 4.4.9 Order 的接口规范

提供的服务（供接口）		
Order.createOrder	语法	public ResultMessage createOrder(OrderVO orderVO)
	前置条件	启动一个订单回合
	后置条件	在一个订单回合中，创建订单
Order.cancelOrder	语法	public ResultMessage cancelOrder(OrderVO orderVO)
	前置条件	订单处于未执行状态
	后置条件	撤销订单
Order.executeOrder	语法	public ResultMessage executeOrder(OrderVO orderVO)
	前置条件	用户按时入住
	后置条件	由酒店工作人员执行订单，更新订单信息
Order.endOrder	语法	public ResultMessage endOrder(OrderVO orderVO)
	前置条件	用户退房，订单处于已执行状态
	后置条件	由酒店工作人员结束订单，更新订单信息
Order.setAbnormal	语法	public void setAbnormal(OrderVO orderVO)
	前置条件	用户超过最晚入住时间未入住
	后置条件	自动设为异常订单

Order.renewOrder	语法	public void renewOrder(OrderVO orderVO)
	前置条件	用户的订单为异常状态，申诉成功
	后置条件	由网站营销人员恢复异常订单，改为已撤销状态
Order.changeStatus	语法	public void changeStatus (OrderVO orderVO)
	前置条件	启动一个订单回合
	后置条件	在一个订单回合中，更改订单状态
需要的服务（需接口）		
服务名	服务	
Order_Data_Service.add(OrderPo po)	新增单一持久化对象	
Order_Data_Service.changeStatus（ OrderPO po, OrderCondition condition ）	更改订单状态	
Order_Data_Service.getStatus(OrderPO po)	获取订单状态	
Order_Data_Service.Update(OrderPO po)	更新订单信息	
Order_Data_Service.find(String CustomerName)	按照用户名字查找订单	
Order_Data_Service.findByHotelID (String HotelID)	按照酒店 ID 查找订单	

Order_Data_Service.findByOrderStatus(OrderStatus status)	按照订单状态查找订单

（4）Order_bl 模块的动态模型

图 4.4.10 表明了酒店管理系统中，创建订单时，相关对象之间的协作。

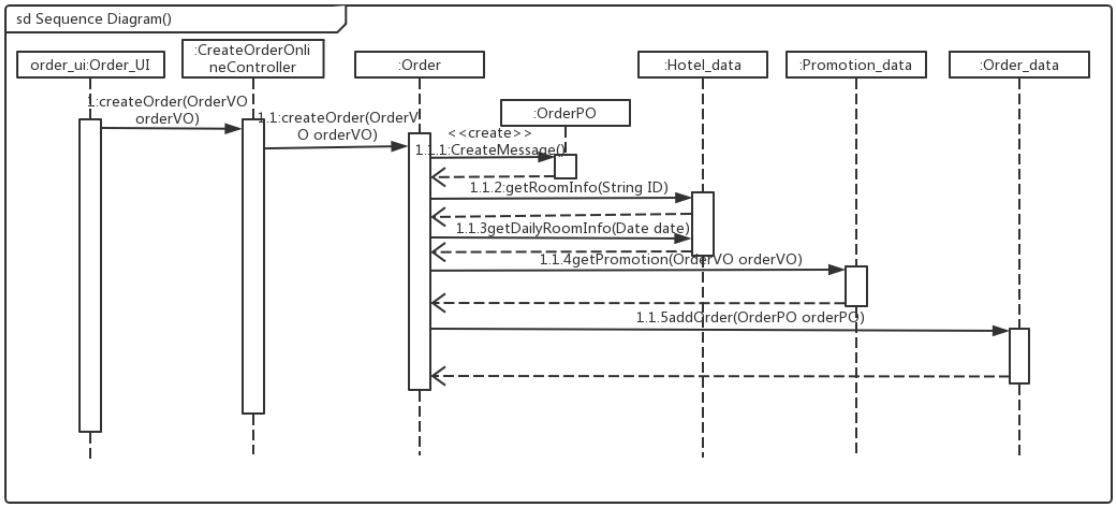


图 4.4.10 创建订单的顺序图

图 4.4.11 表明了酒店管理系统中，撤销订单时，相关对象之间的协作。

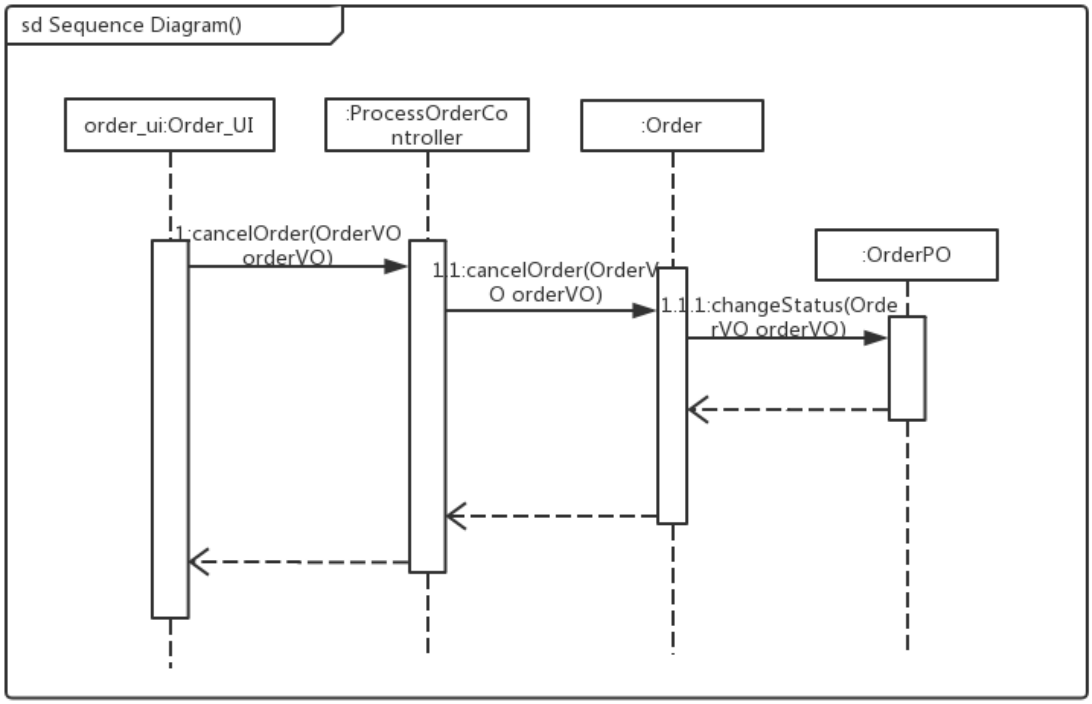


图 4.4.11 撤销订单的顺序图

图 4.4.12 表明了酒店管理系统中，执行订单时，相关对象之间的协作。

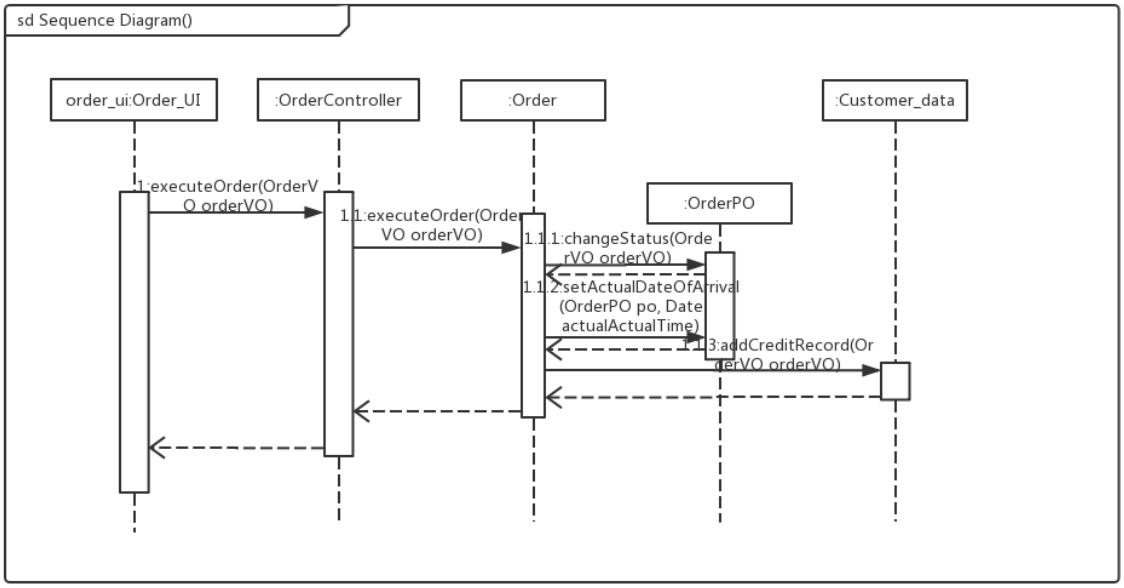


图 4.4.12 执行订单的顺序图

图 4.4.13 表明了酒店管理系统中，设为异常订单时，相关对象之间的协作

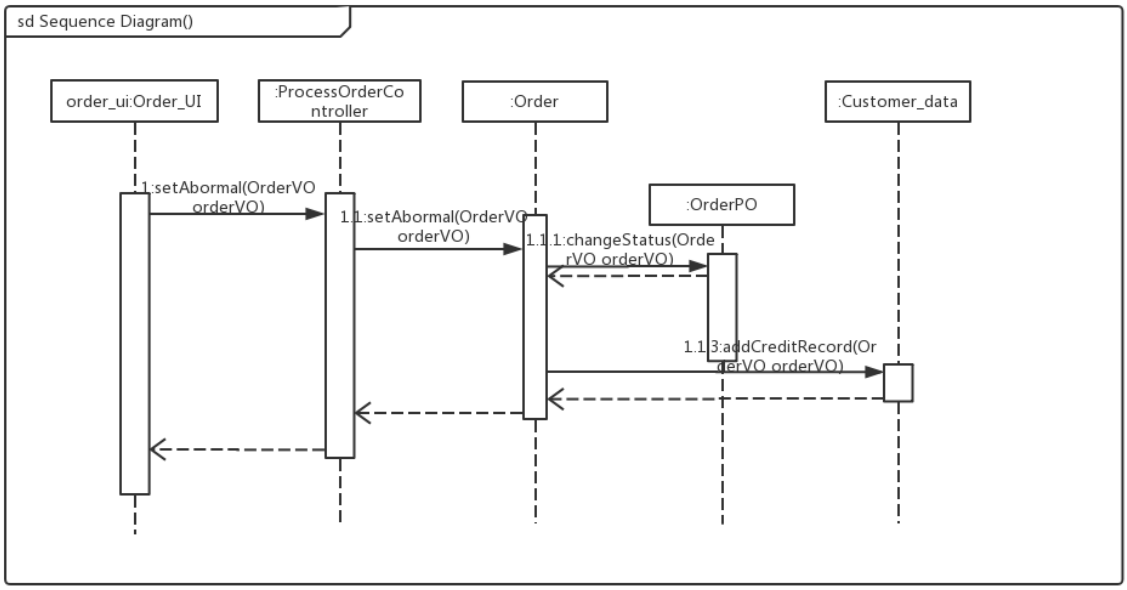


图 4.4.13 设为异常订单的顺序图

如图 4.4.14 所示的状态图描述了 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。随着 createOrder 方法被 ui 调用，Order 进入 unexecuted 状态，随着 executeOrder 方法被 UI 调用 ,Order 进入 executed 状态 ,随着 endOrder 方法被 UI 调用 ,Order 进入 finished 状态。

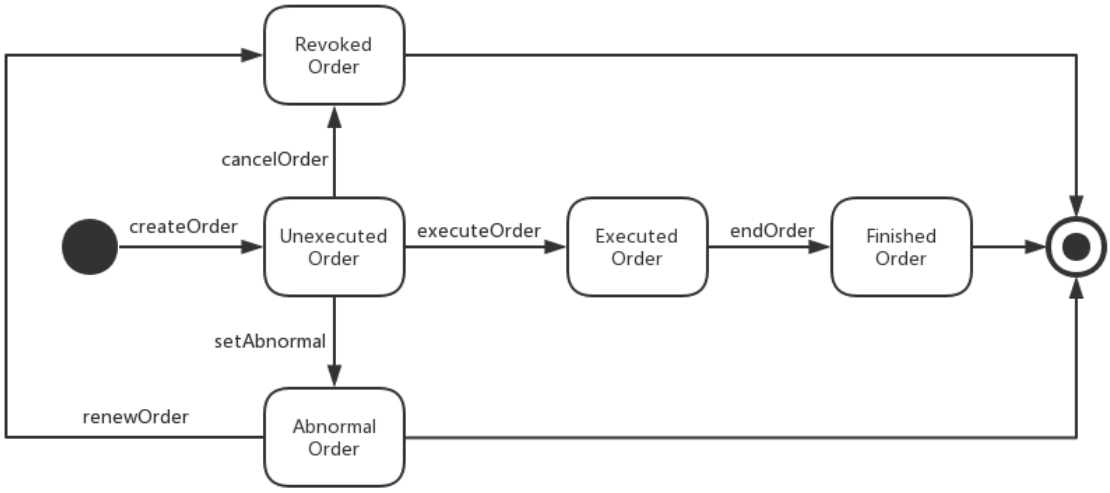


图 4.4.14 订单模块的状态图

（5）Order_bl 模块的设计原理

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

4.5 Member_bl 模块

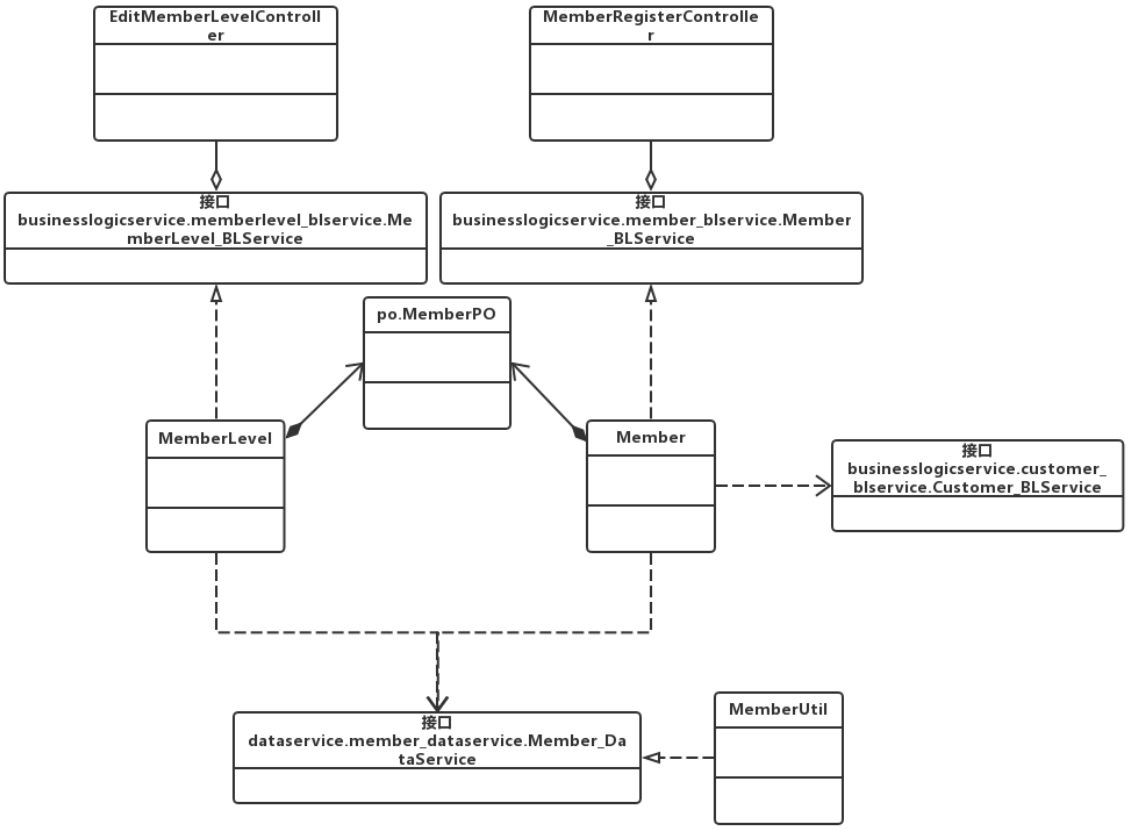
（1）模块概述

Member_bl 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。Member_bl 模块的职责及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，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 Member_blservice、Member_data_service,MemberLevel_blservice 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 MemberRegisterController,EditMemberLevelController ,这样 MemberRegisterController 将会将注册会员的业务逻辑职责和逻辑控制委托给 Member_bl 对象，这样 EditMemberLevelController 将会将会员等级制度的业务逻辑职责和逻辑控制委托给 Member_bl 对象

Member_bl 模块的设计如图 4.5.1 所示。



Member 模块各个类的职责如表 4.5.2 所示。

表 4.5.2 Order 模块各个类的职责

模块	职责
Member	会员的领域模型对象，负责处理会员方面的功能
MemberRegisterController	负责实现对于会员注册界面所需的方法
MemberLevel	会员等级的领域模型对象，负责处理会员等级制度方面的功能
EditMemberLevelController	负责实现对于会员等级制度界面所需的方法
MemberUtil	负责获取会员信息

（3）模块内部类的接口规范

MemberRegisterController 的接口规范如表 4.5.3 所示。

表 4.5.3 MemberRegisterController 的接口规范

提供的服务（供接口）		
MemberController.signUp	语法	public ResultMessage signUp(MemberVO

		memberVO, CustomerVO customerVO)
	前置条件	已有一个 Member 邻域对象，输入会员信息正确
	后置条件	调用 Member 邻域对象的 signUp（ ）方法，调用 Customer 邻域对象的 getCredit（ ）方法
需要的服务（需接口）		
服务名	服务	
Member.signUp	注册会员	
Customer.getCredit	获取用户信用值	

Member 的接口规范如表 4.5.4 所示。

表 4.5.4 Member 的接口规范

提供的服务（供接口）		
Member.signUp	语法	public ResultMessage signUp(MemberVO memberVO, CustomerVO customerVO)
	前置条件	用户信用值足以注册会员
	后置条件	已输入会员信息
需要的服务（需接口）		
服务名	服务	
Member_Data_Service.add	新增单一持久化对象	

ProcessMemberController 的接口规范如表 4.5.5 所示。

表 4.5.5 ProcessMemberController 的接口规范

提供的服务（供接口）

ProcessMemberController.upGrade	语法	public void upGrade(MemberVO memberVO, CustomerVO customerVO)
	前置条件	已有一个 Member 邻域对象，信用值足够升级
	后置条件	调用 Member 邻域对象的 upGrade（ ）方法
ProcessMemberController.deGrade	语法	public void deGrade(MemberVO memberVO, CustomerVO customerVO)
	前置条件	已有一个 Member 邻域对象，信用值不够当前等级
	后置条件	调用 Member 邻域对象的 deGrade（ ）方法
需要的服务（需接口）		
服务名	服务	
Member.upGrade	会员等级上升	
Member.deGrade	会员等级下降	
Customer.getCredit	获取用户信用值	

EditMemberLevelController 的接口规范如表 4.5.6 所示。

表 4.5.6 EditMemberLevelController 的接口规范

提供的服务（供接口）		
MemberLevelController.addMemberLevel	语法	public ResultMessage addMemberLevel(MemberLevelVO memberLevelVO)
	前置条件	已创建一个 memberlevel 领域对象，并且输入符合输入规则
	后置条件	调用 MemberLevel 领域对象的 addMemberLevel 方

		法
MemberLevelController. modifyMemberLevel	语法	public ResultMessage modifyMemberLevel(MemberLevelVO memberLevelVO)
	前置条件	已创建一个 memberlevel 领域对象，并且输入符合输入规则
	后置条件	调用 MemberLevel 领域对象的 modifyMemberLevel 方法
需要的服务（需接口）		
服务名	服务	
MemberLevel.addMemberLevel	添加会员等级制度	
MemberLevel.modifyMemberLevel	修改会员等级制度	

MemberLevel 的接口规范如表 4.5.7 所示。

表 4.5.7 MemberLevel 的接口规范

提供的服务（供接口）		
MemberLevel.addMemberLevel	语法	public ResultMessage addMemberLevel(MemberLevelVO memberLevelVO)
	前置条件	无
	后置条件	填写会员等级制度信息完整

MemberLevel.modifyMemberLevel	语法	public ResultMessage modifyMemberLevel(MemberLevelVO memberLevelVO)
	前置条件	无
	后置条件	填写会员等级制度信息完整
需要的服务（需接口）		
服务名	服务	
Member_Data_Service.addMemberLevel(MemberLevelPO po)	添加会员等级制度	
Member_Data_Service.modifyMemberLevel(MemberLevelPO po)	修改会员等级制度	

MemberUtil 的接口规范如表 4.5.7 所示。

表 4.5.7 MemberUtil 的接口规范

提供的服务（供接口）		
MemberUtil.getSingle	语法	public MemberVO getSingle (String ID)
	前置条件	无
	后置条件	根据用户 ID 获取会员信息
需要的服务（需接口）		
服务名	服务	
Member_Data_Service.fin	获取会员信息	

d(String ID)	

（4）Member_bl 模块的动态模型

图 4.5.8 表明了酒店管理系统中，注册会员时，相关对象之间的协作。

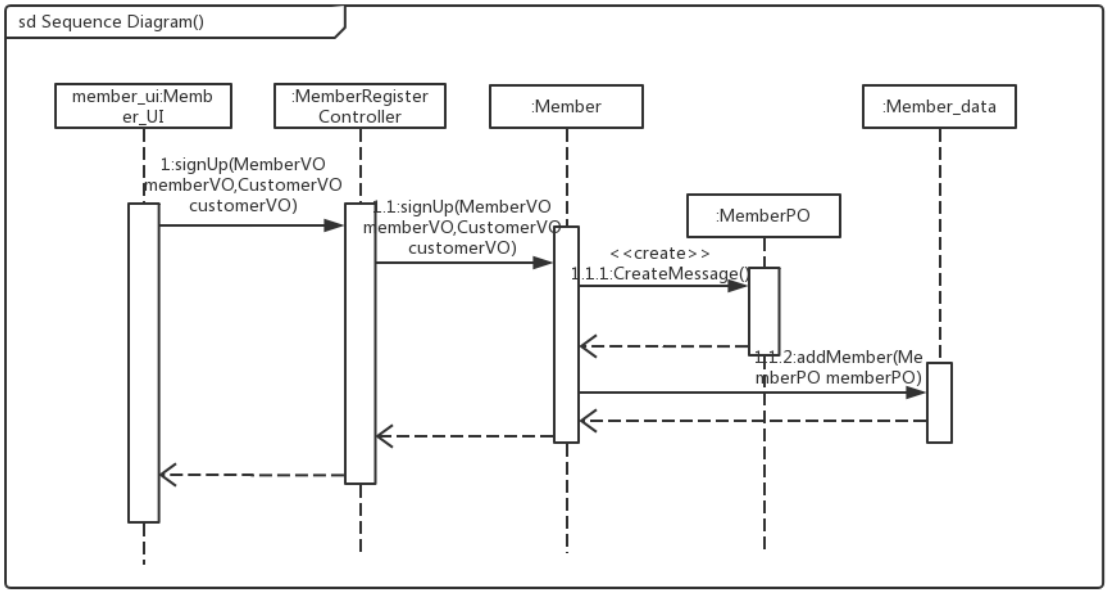


图 4.5.8 注册会员的顺序图

如图 4.5.9 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 signUp 方法被 UI 调用

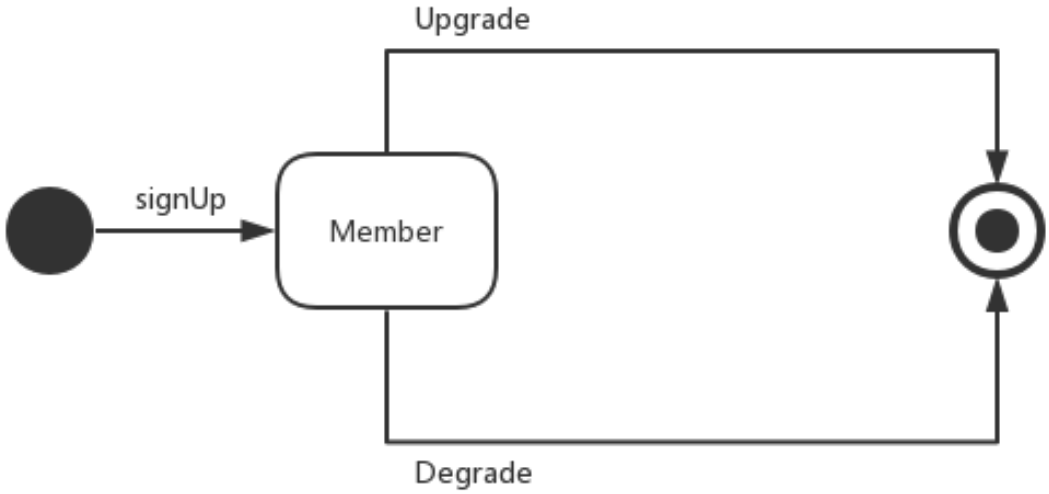


图 4.5.9 会员模块的状态图

4.6 Hotel_bl 模块

（1）模块概述

Hotel_bl 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_bl 模块的职责及接口参见软件体系结构描述文档表 5.3.2-6。

（2）整体结构

根据体系结构设计采用分层风格，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businesslogicservice.hotelblservice.HotelBLService, dataservice.hoteldataservice.HotelDataService 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 HotelController、CommentHotelController、InputRoomController，这样

HotelController 将会将浏览酒店信息相关的业务逻辑职责和逻辑控制委托给 Hotel_bl 对象，CommentHotelController 将会将评价酒店相关的业务逻辑职责和逻辑控制委托给 Hotel_bl 对象，InputRoomController 将会将录入可用客房相关的业务逻辑职责和逻辑控制委托给 Hotel_bl 对象。HotelPO、CommentPO、RoomPO 和 DailyRoomInfoPO 是做为管理信息的持久化对象被添加到设计模型中的。RoomList 和 RoomLineItem 的添加是 RoomPO 的容器类。Room 保有酒店的房间信息，RoomList 封装了 RoomLineItem 的数据集合的数据结构的秘密。DailyRoomInfoList 和 DailyRoomInfoLineItem 的添加是 DailyRoomInfoPO 的容器类。DailyRoomInfo 保有酒店每天房间的实时状态数据，DailyRoomInfoList 封装了 DailyRoomInfo 的数据集合的数据结构的秘密。CoomentList 和 CommentLineItem 的添加是 CommentPO 的容器类。Comment 保有酒店的评价，CoomentList 封装了 Comment 的数据集合的数据结构的秘密。

Hotel_bl 模块的设计如 4.6.1 所示。

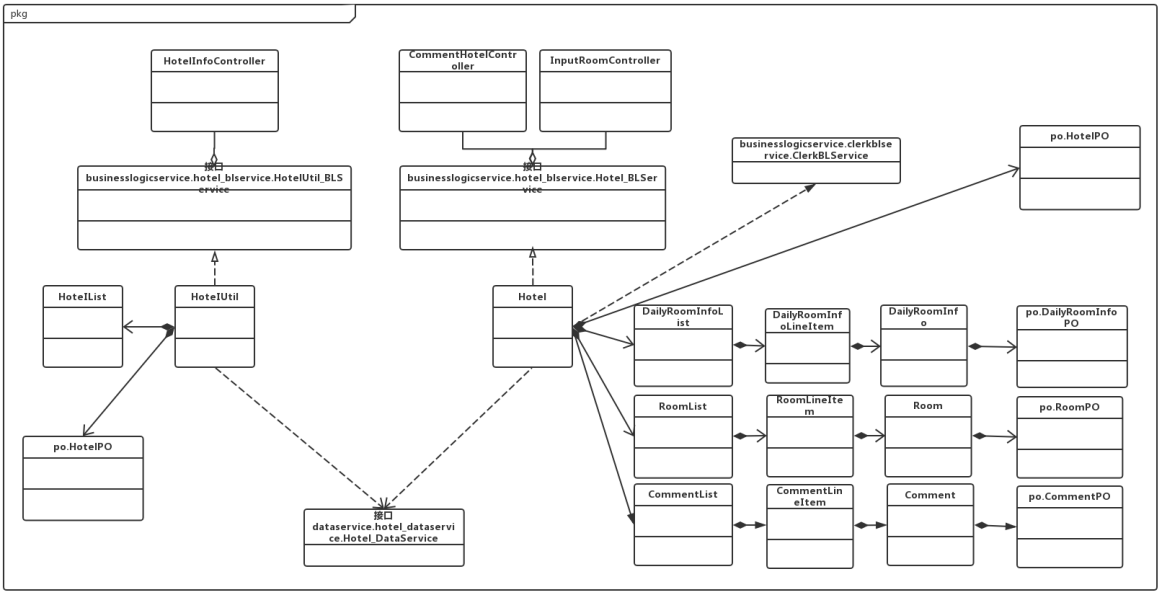


图 4.6.1 Hotel_bl 模块各个类的设计

Hotel_bl 模块各个类的职责如表 4.6.2 所示。

表 4.6.2 Hotel_bl 模块各个类的职责

模块	职责
Hotel	酒店的领域模型对象，负责酒店方面的功能
HotelUtil	负责批量处理酒店方面的功能
HotelInfoController	负责实现对于浏览酒店信息所需的方法
CommentHotelController	负责实现对于评价酒店所需的方法
InputRoomController	负责实现对于录入可用客房所需的方法

（3）模块内部类的接口规范

HotelInfoController、CommentHotelController、InputRoomController 和 Hotel 和 HotelUtil 的接口规范如表 4.6.3、4.6.4、4.6.5、4.6.6 和 4.6.7 所示。

表 4.6.3 HotelInfoController 的接口规范

HotelInfoController.sortByPrice	语法	public List<HotelVO> sortByPrice(List<HotelVO> list)
	前置条件	已创建一个 HotelUtil 领域对象
	后置条件	调用 HotelUtil 领域对象的 sortByPrice 方法
HotelInfoController.sortByStar	语法	public List<HotelVO> sortByStar(List<HotelVO> list)
	前置条件	已创建一个 HotelUtil 领域对象
	后置条件	调用 HotelUtil 领域对象的 sortByStar 方法
HotelInfoController.sortByScore	语法	public List<HotelVO> sortByScore(List<HotelVO> list)
	前置条件	已创建一个 HotelUtil 领域对象
	后置条件	调用 HotelUtil 领域对象的 sortByScore 方法
HotelInfoController.searchHotel	语法	public List<HotelVO> searchHotel (HotelVO hotelVO)

	前置条件	已创建一个 HotelUtil 领域对象
	后置条件	调用 HotelUtil 领域对象的 searchHotel 方法
需要的服务（需接口）		
服务名	服务	
HotelUtil.sortByPrice	按价格排列酒店列表	
HotelUtil.sortByStar	按星级排列酒店列表	
HotelUtil.sortByScore	按评分排列酒店列表	
HotelUtil.searchHotel	搜索酒店	

表 4.6.4 CommentHotelController 的接口规范

提供的服务（供接口）		
CommentHotelController.addComment	语法	public ResultMessage addComment(CommentVO commentVO, OrderVO orderVO)
	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 addComment 方法
需要的服务（需接口）		
服务名	服务	
Hotel.addComment	添加一个评论对象	

表 4.6.5 InputRoomController 的接口规范

提供的服务（供接口）		
InputRoomController.addRoom	语法	public ResultMessage addRoom(RoomVO roomVO)
	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 addRoom 方法 调用 Hotel 领域对象的 modifyDailyRoomInfo 方法
需要的服务（需接口）		
服务名	服务	
Hotel.addRoom	加入一个房间对象	
Hotel.modifyDailyRoomInfo	修改每日房间信息	

表 4.6.6 Hotel 的接口规范

提供的服务（供接口）		
Hotel.addRoom	语法	public ResultMessage addRoom(RoomVO roomVO)
	前置条件	启动一个添加房间回合
	后置条件	在一个销售回合中，增加酒店的房间信息
Hotel.modifyRoom	语法	public ResultMessage modifyRoom(RoomVO roomVO)
	前置条件	启动一个修改回合
	后置条件	在一个修改回合中，修改酒店的房间信息
Hotel.deleteRoom	语法	public ResultMessage deleteRoom (RoomVO roomVO)
	前置条件	启动一个删除回合
	后置条件	在一个删除回合中，删除酒店的房间信息
Hotel.changeAvailableRoom	语法	public ResultMessage changeAvailableRoom(String ID, String type, int number , DailyRoomInfoVO

		dailyRoomInfoVO)
	前置条件	已添加基本房间信息
	后置条件	返回改变空闲房间结果
Hotel.changeReservedRoom	语法	public ResultMessage changeReservedRoom(String type, int number, DailyRoomInfoVO dailyRoomInfoVO)
	前置条件	已添加基本房间信息
	后置条件	返回变化已预订房间结果
Hotel.changeOccupiedRoom	语法	public ResultMessage changeOccupiedRoom(String type, int number, DailyRoomInfoVO dailyRoomInfovo)
	前置条件	已添加基本房间信息
	后置条件	返回变化已入住房间结果
Hotel.addComment	语法	public ResultMessage addComment(CommentVO commentVO, OrderVO orderVO)
	前置条件	启动一个订单回合
	后置条件	在一个订单回合中，增加评价信息
Hotel.getDailyRoomInfo	语法	public ResultMessage getDailyRoomInfo (String ID ，Date Date)
	前置条件	已添加基本房间信息
	后置条件	返回该酒店该日期的每日房间数量信息
Hotel.getRoom	语法	public List<RoomVo> getRoom(String ID)

	前置条件	已添加基本房间信息
	后置条件	返回酒店房间信息
Hotel.addHotel	语法	public ResultMessage addHotel(HotelVO roomVO)
	前置条件	启动一个添加酒店回合
	后置条件	在数据库中添加一个酒店的信息
Hotel.modifyHotel	语法	public ResultMessage modifyHotel(HotelVO roomVO)
	前置条件	已添加该酒店
	后置条件	在数据库中修改该酒店的信息
Hotel.deleteHotel	语法	public ResultMessage deleteHotel(HotelVO roomVO)
	前置条件	已添加该酒店
	后置条件	在数据库中删除该酒店的信息
Hotel.addTo ListOfHotelReservedByCustomer	语法	public ResultMessage addToListOfHotelReservedByCustomer(HotelPO hotelPO, CustomerPO customerPO)
	前置条件	客户预订该酒店
	后置条件	将该酒店添加至客户的预订过的酒店列表
Hotel.modifyDailyRoomInfo	语法	public ResultMessage modifyDailyRoomInfo (List<DailyRoomInfoPo> list)
	前置条件	已有每日房间信息
	后置条件	修改每日房间信息

需要的服务（需接口）	
服务名	服务
Hotel_Data_Service.addRoomType（RoomPO po）	新增单一持久化对象
Hotel_Data_Service.add(HotelPO po)	新增酒店信息
Hotel_Data_Service.modify(HotelPO po)	修改酒店信息
Hotel_Data_Service.delete(HotelPO po)	删除酒店信息
Hotel_Data_Service.modifyRoomType(RoomPo po)	修改房间信息
Hotel_Data_Service.deleteRoomType(RoomPo po)	删除房间信息
Hotel_Data_Service.setDailyRoomInfo(List<DailyRoomInfoPo>)	设置房间信息
Hotel_Data_Service.addComment(CommentPO po)	增加酒店评价

Hotel_Data_Service.getDailyRoomInfo（Date date）	获取当日房间信息
Hotel_Data_Service.addToListOfHotelReservedByCustomer(HotelPO hotelPO, CustomerPO customerPO)	将酒店添加至客户预订过的酒店列表

表 4.6.7 HotelUtil 的接口规范

提供的服务（供接口）		
HotelUtil.getSingle	语法	public HotelVo getSingle(String ID)
	前置条件	已添加酒店信息
	后置条件	返回酒店信息
HotelUtil.getAll	语法	public List<HotelVo> getAll()
	前置条件	已添加酒店信息
	后置条件	返回所有酒店信息
HotelUtil.sortByPrice	语法	public List<HotelVO> sortByPrice(List<HotelVO> list)
	前置条件	已获得酒店列表
	后置条件	将酒店按照价格排序
HotelUtil.sortByStar	语法	public List<HotelVO> sortByStar(List<HotelVO> list)
	前置条件	已获得酒店列表

	后置条件	将酒店按照星级排序
HotelUtil.sortByScore	语法	public List<HotelVO> sortByScore(List<HotelVO> list)
	前置条件	已获得酒店列表
	后置条件	将酒店按照评分排序
HotelUtil.searchHotel	语法	public List<HotelVO> searchHotel (HotelVO hotelVO)
	前置条件	启动一个搜索酒店回合
	后置条件	返回所有符合搜索条件的酒店
HotelUtil.getByID	语法	public HotelVo getByID (String ID)
	前置条件	已添加酒店信息
	后置条件	返回该酒店信息
HotelUtil.getByName	语法	public HotelVo getByName (String name)
	前置条件	已添加酒店信息
	后置条件	返回该酒店信息
HotelUtil.getByAddress	语法	public HotelVo getByAddress (String address)
	前置条件	已添加酒店信息
	后置条件	返回该酒店信息
需要的服务（需接口）		
服务名	服务	
Hotel_Data_Service.getHotel(String id)	获得酒店信息	

Hotel_Data_Service.getAllHotel()	获得所有酒店
Hotel_Data_Service.find(String address, String area, Date expected_date_of_arrival, Date expected_date_of_departure， int star, double score)	查找酒店

（4）Hotel_bl 模块的动态模型

图 4.6.8 表明了变更房间状态的顺序图

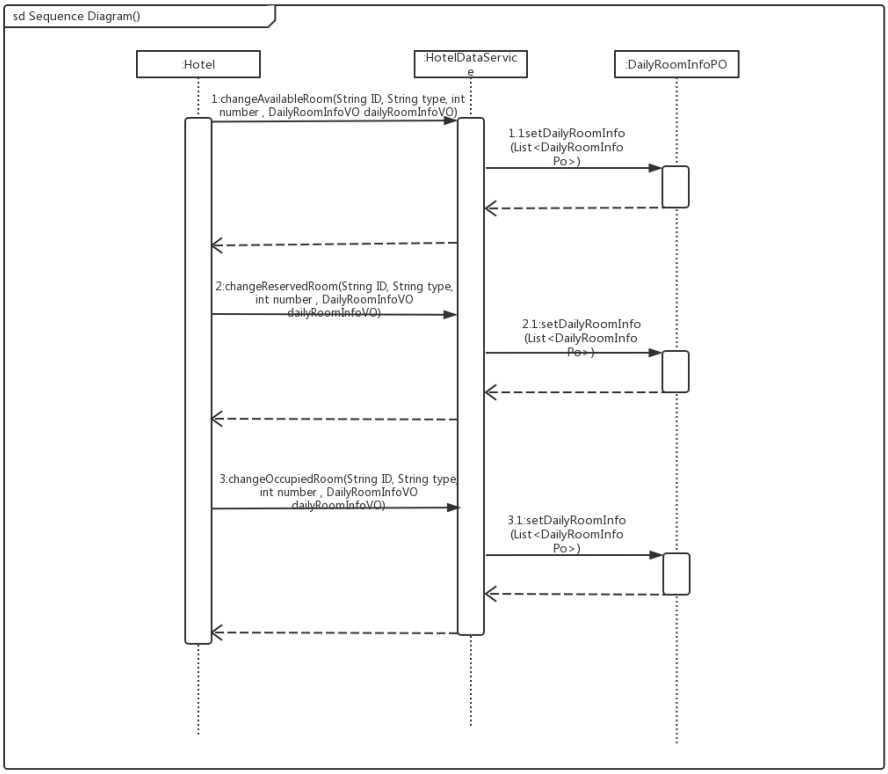


图 4.6.8 变更房间状态的顺序图

图 4.6.9 表明了添加酒店房间的顺序图

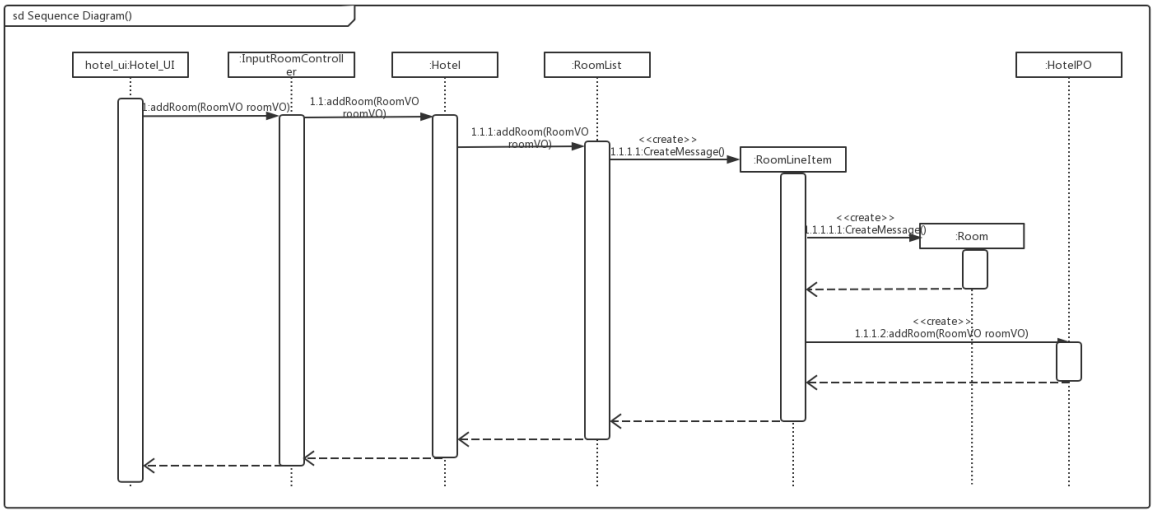


图 4.6.9 添加酒店房间的顺序图

图 4.6.10 表明了添加酒店评论的顺序图

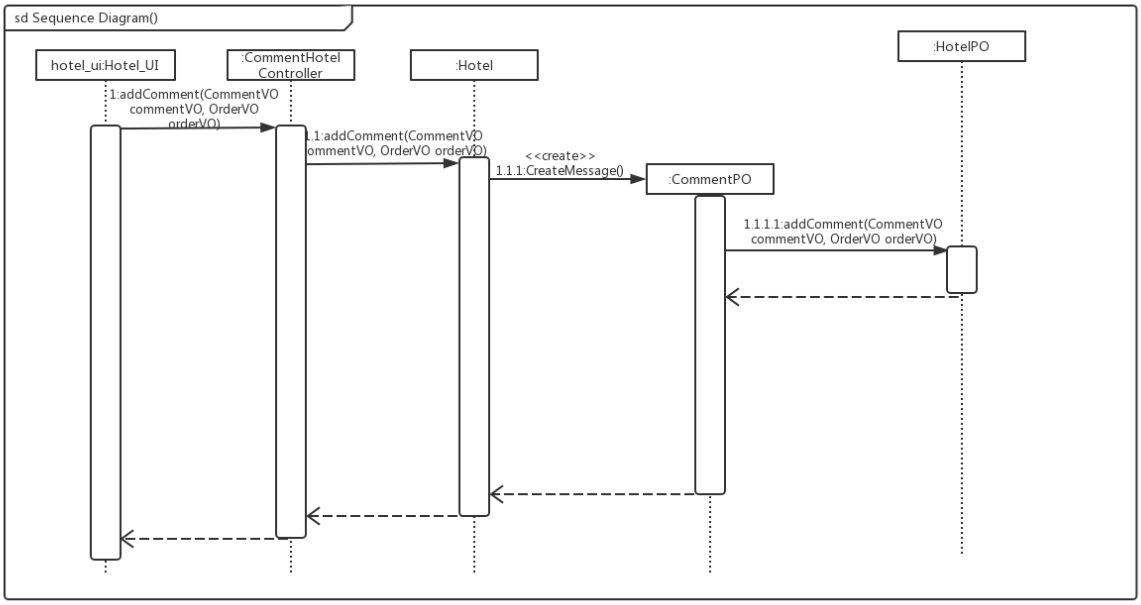


图 4.6.10 添加酒店评论的顺序图

图 4.6.11 表明了获得房间信息的顺序图

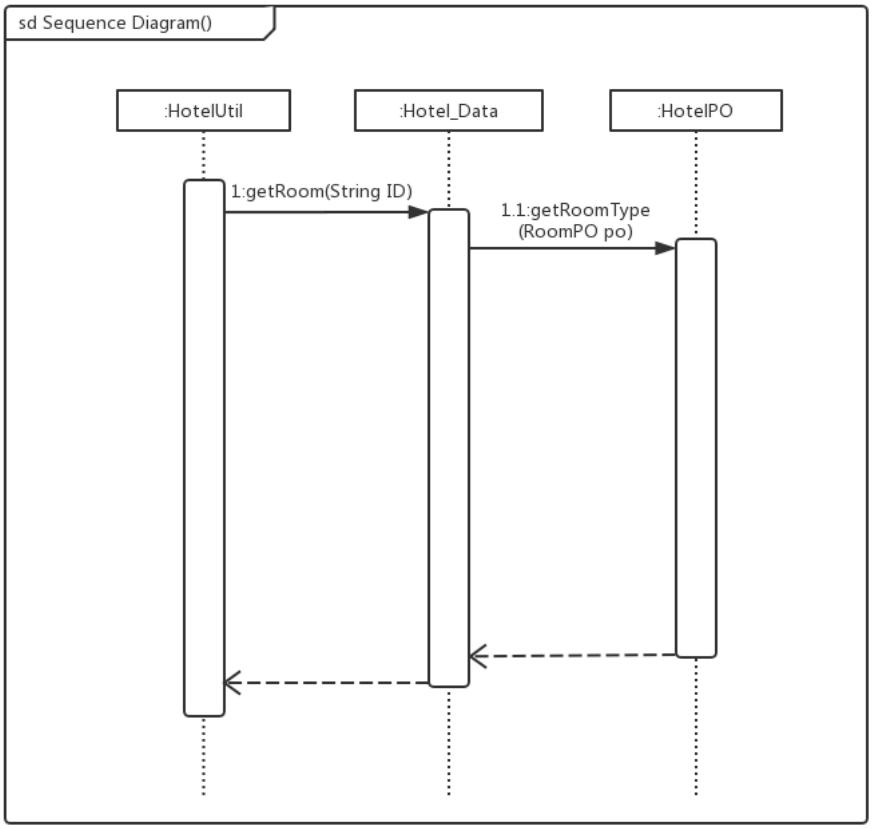


图 4.6.11 获得房间信息的顺序图

图 4.6.12 表明了获得酒店信息的顺序图

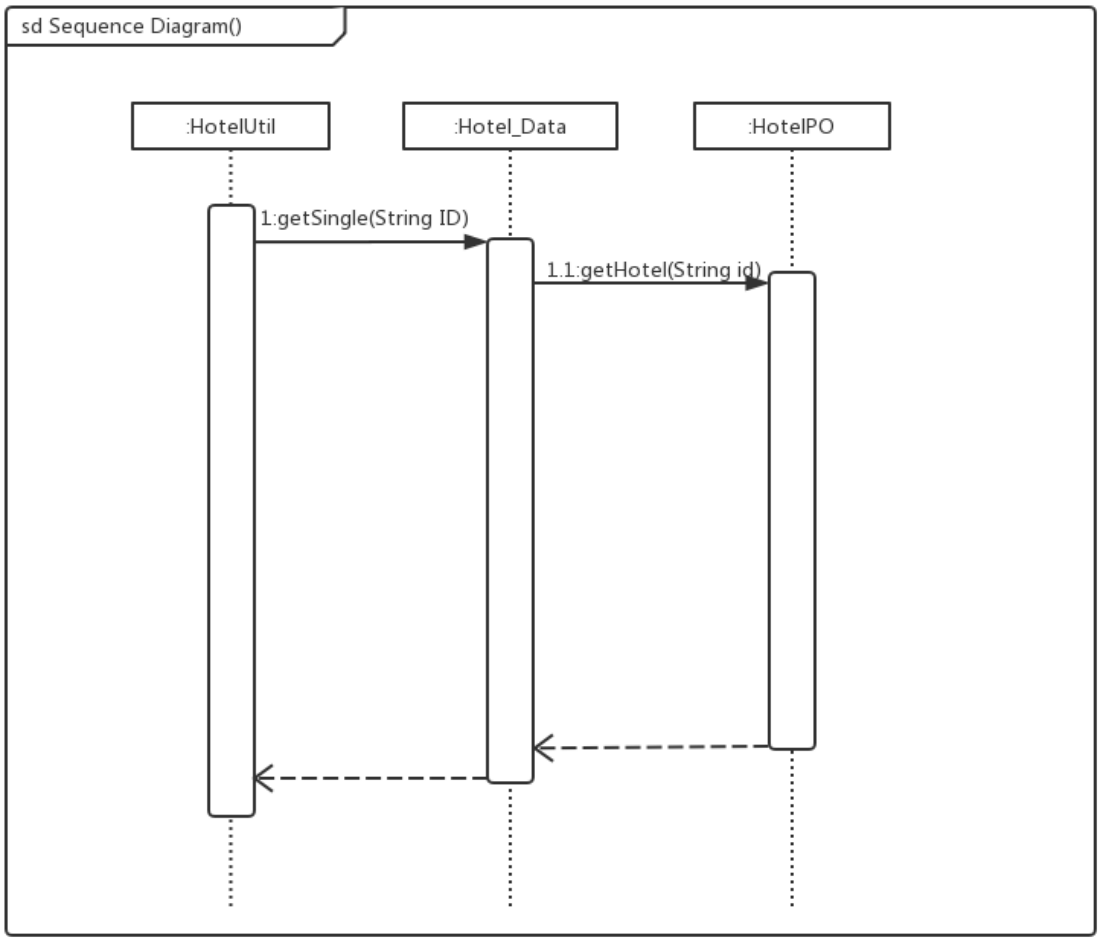


图 4.6.12 获得酒店信息的顺序图

如图 4.6.13 所示的状态图描述了 Hotel 对象的生存期间的状态序列、引起转移的时间，以及因状态转移而伴随的动作。随着 addComment 方法被 UI 调用，Hotel 进入 Comment 状态，之后通过评论添加进入 CommentLineItem 状态。随着 addRoom 方法被 UI 调用，Hotel 进入 Room 状态，之后通过评论添加进入 RoomLineItem 状态。随着 getDailyLineItem 方法被 UI 调用，Hotel 进入 DailyLineItem 状态，之后通过评论添加进入 DailyLineItemLineItem 状态。随着 getSingle 方法被 UI 调用，Hotel 进入 SingleRoom 状态。随着 getAll 方法被 UI 调用，Hotel 进入 AllRoom 状态。

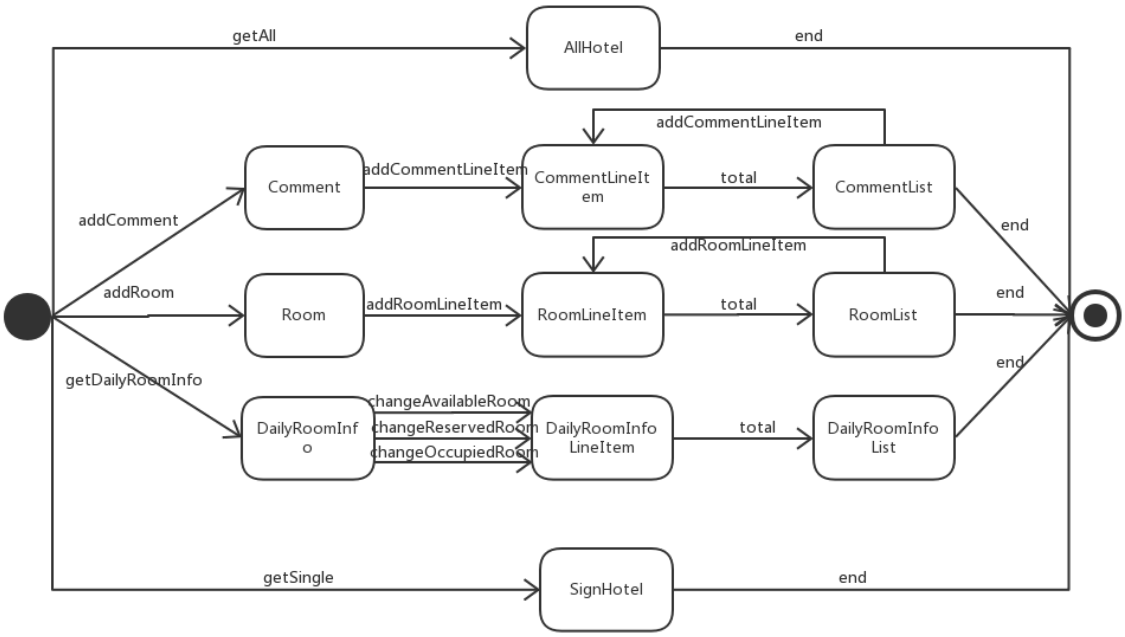


图 4.6.13 酒店对象状态图

（5）Hotel_bl 模块的设计原理

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。

4.7 Promotion_bl 模块

（1）模块概述

Hotel_bl 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_bl 模块的职责及接口参见软件体系结构描述文档表 5.3.2-6。

（2）整体结构

根据体系结构设计采用分层风格，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businesslogicservice.promotionblservice.PromotionBLService, dataservice.promotiondataservice.PromotionDataService 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 PromotionController，这样 PromotionController 将会将用户管理相关的业务逻辑职责和逻辑控制委托给 Promotion_bl 对象。PromotionPO 是做为管理信息的持久化对象被添加到设计模型中的。

Promotion_bl 模块的设计如图 4.7.1 所示。

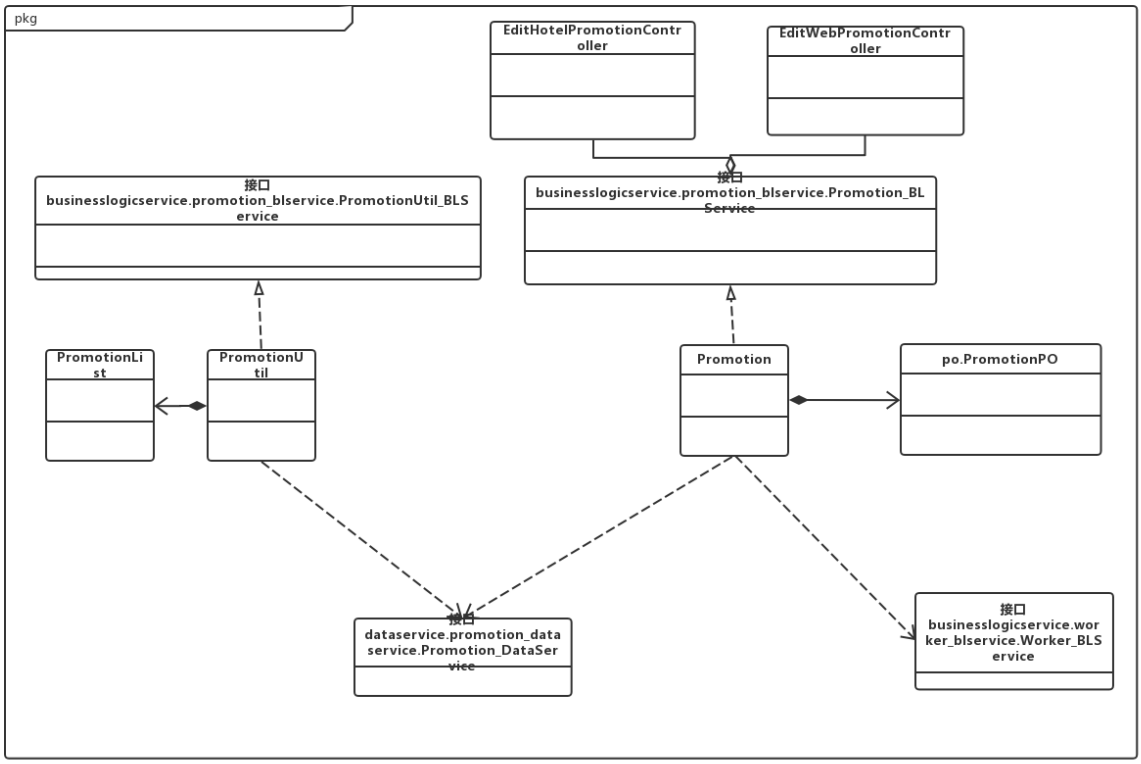


图 4.7.1 Promotion_bl 模块各个类的设计

Promotion_bl 模块各个类的职责如表 4.7.2 所示。

表 4.7.2 Promotion_bl 模块各个类的职责

模块	职责

EditWebPromotionController	负责实现对于制定网站营销策略所需的方法
EditHotelPromotionController	负责实现对于制定酒店营销策略所需的方法
Promotion	促销策略的领域模型对象，负责促销策略方面的功能
PrmotionUtil	负责对促销策略进行批量处理

（3）模块内部类的接口规范

EditHotelPromotionController 、 EditWebPromotionController 、 Promotion 和 PrmotionUtil 的接口规范如表 4.7.3、4.7.4、4.7.5 和 4.7.6 所示。

表 4.7.3 EditPromotionController 的接口规范

提供的服务（供接口）		
EditPromotionController.addHotelPromotion	语法	public ResultMessage addHotelPromotion (PromtionVO promotionVO)
	前置条件	已创建一个 Promotion 领域对象，并且输入符合输入规则
	后置条件	调用 Promotion 领域对象的 addHotelPromotion 方法
EditPromotionController.addWebPromotion	语法	public ResultMessage addWebPromotion (PromtionVO promotionVO)
	前置条件	已创建一个 Promotion 领域对象，并且输入符合输入规则
	后置条件	调用 Promotion 领域对象的 addWebPromotion 方法
EditPromotionController.modifyPromotion	语法	public ResultMessage modifyPromotion (PromtionVO promotionVO)
	前置条件	已创建一个 Promotion 领域对象，并且输入符合输入

		规则
	后置条件	调用 Promotion 领域对象的 modifyPromotion 方法
EditPromotionController.deletePromotion	语法	public ResultMessage deletePromotion (String promotionID)
	前置条件	已创建一个 Promotion 领域对象，并且输入符合输入规则
	后置条件	调用 Promotion 领域对象的 deletePromotion 方法
需要的服务（需接口）		
服务名	服务	
Promotion.addWebPromotion(PromtionVO promotionVO)	添加一个网站促销策略对象	
Promotion.addHotelPromotion(PromtionVO promotionVO)	添加一个酒店促销策略对象	
Promotion.modifyPromotion(PromtionVO promotionVO)	修改促销策略	
Promotion.deletePromotion(PromtionVO	删除促销策略	

promotionVO)	

表 4.7.4 Promotion 的接口规范

提供的服务（供接口）		
Promotion.addHotelPromotion	语法	public ResultMessage addHotelPromotion(PromtionVO promotionVO)
	前置条件	启动一个促销回合
	后置条件	在一个促销回合中，添加新的促销策略信息
Promotion.addWebPromotion	语法	public ResultMessage addWebPromotion(PromtionVO promotionVO)
	前置条件	启动一个促销回合
	后置条件	在一个促销回合中，添加新的促销策略信息
Promotion.modifyPromotion	语法	public ResultMessage addWebPromotion(PromtionVO promotionVO)
	前置条件	启动一个促销回合
	后置条件	在一个促销回合中，修改促销策略信息
Promotion.deletePromotion	语法	public ResultMessage addWebPromotion(String promotionID)
	前置条件	已添加该促销策略
	后置条件	删除促销策略信息
需要的服务（需接口）		
服务名	服务	

Promotion_Data_Service.add	新增单一持久化对象
Promotion_Data_Service.modify	修改单一持久化对象
Promotion_Data_Service.delete	删除单一持久化对象

表 4.7.6PromotionUtil 的接口规范

提供的服务（供接口）		
Promotion.getAll	语法	public List<PromotionVO> getAll (Date date)
	前置条件	已存在促销策略
	后置条件	返回当前时间内所有促销策略
Promotion.getSingle	语法	public PromotionVO getSingle(String promotionID)
	前置条件	已存在促销策略
	后置条件	返回具体的促销策略
需要的服务（需接口）		
服务名	服务	
Promotion_Data_Service.getSingle(String promotionID)	获取某个促销策略	
Promotion_Data_Service.getAll()	获取所有促销策略	

（4）Promotion_bl 模块的动态模型

图 4.7.8 表明了添加酒店促销策略的顺序图

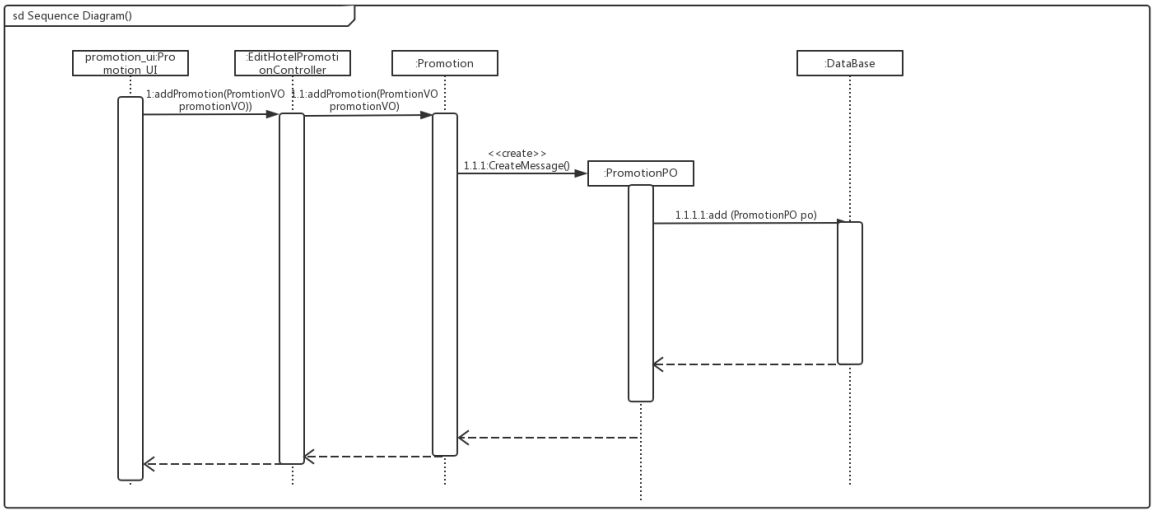


图 4.7.8 添加酒店促销策略的顺序图

图 4.7.9 表明了添加促销策略的顺序图

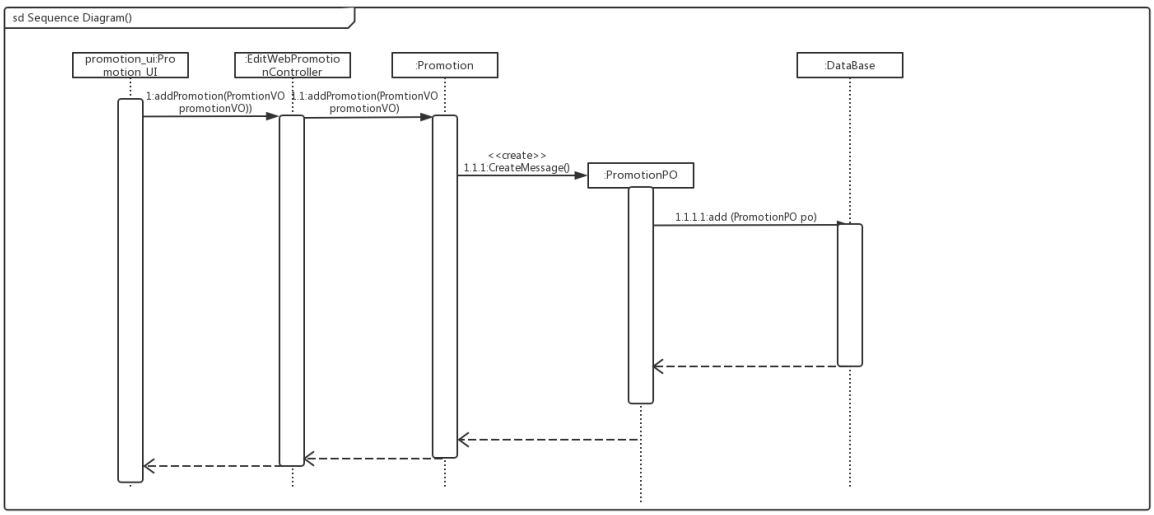


图 4.7.9 添加网站促销策略的顺序图

如图 4.7.10 所示的状态图描述了 Promotion 对象的生存期间的状态序列、引起转移的时间，以及因状态转移而伴随的动作。随着 addPromotion 方法被 UI 调用，Promotion 进入 Promotion 状态，之后通过评论添加进入 PromotionLineItem 状态。随着 getSingle 方法被 UI 调用，Promotion

进入 SinglePromotion 状态。随着 getAll 方法被 UI 调用，Hotel 进入 AllPromotion 状态。

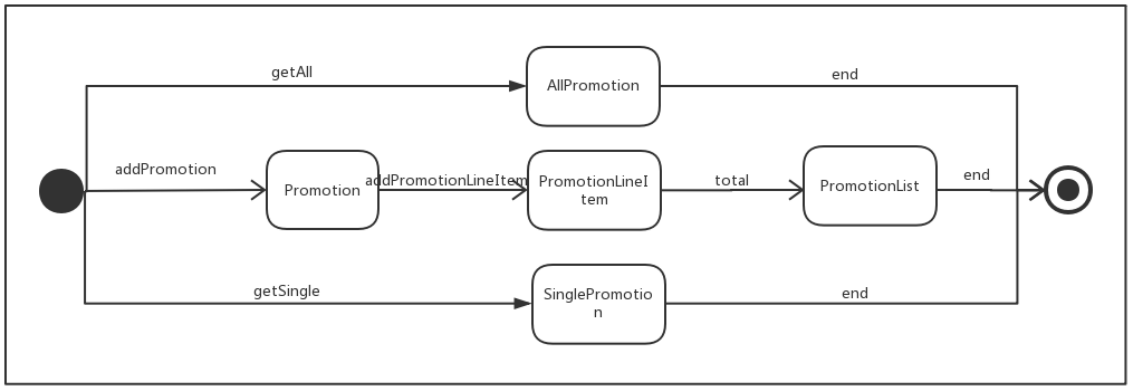


图 4.7.9 促销策略对象状态图

（5）Promotion_bl 模块的设计原理

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。

4.8 Marketer_bl 模块

（1）模块概述

Marketer_bl 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。

Marketer_bl 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businesslogicservice.markerter_blservice.MarketerBLService,

businesslogicservice.marketerblservice.MarketerUtilBLService, dataservice.marketer_dataservice.MarketerDataService 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 CreditChrgeController 和 MarketerInfoChangeController.这样 CreditChrgeController 和 MarketerInfoChangeController 将会将网站营销人员信用充值和维护个人信息相关的业务逻辑职责和逻辑控制委托给 Marketer 对象。MarketerPO 和 CreditRecordPO 是作为管理信息的持久化对象被添加到设计模型中的。

Marketer_bl 模块的设计如图 4.8.1 所示。

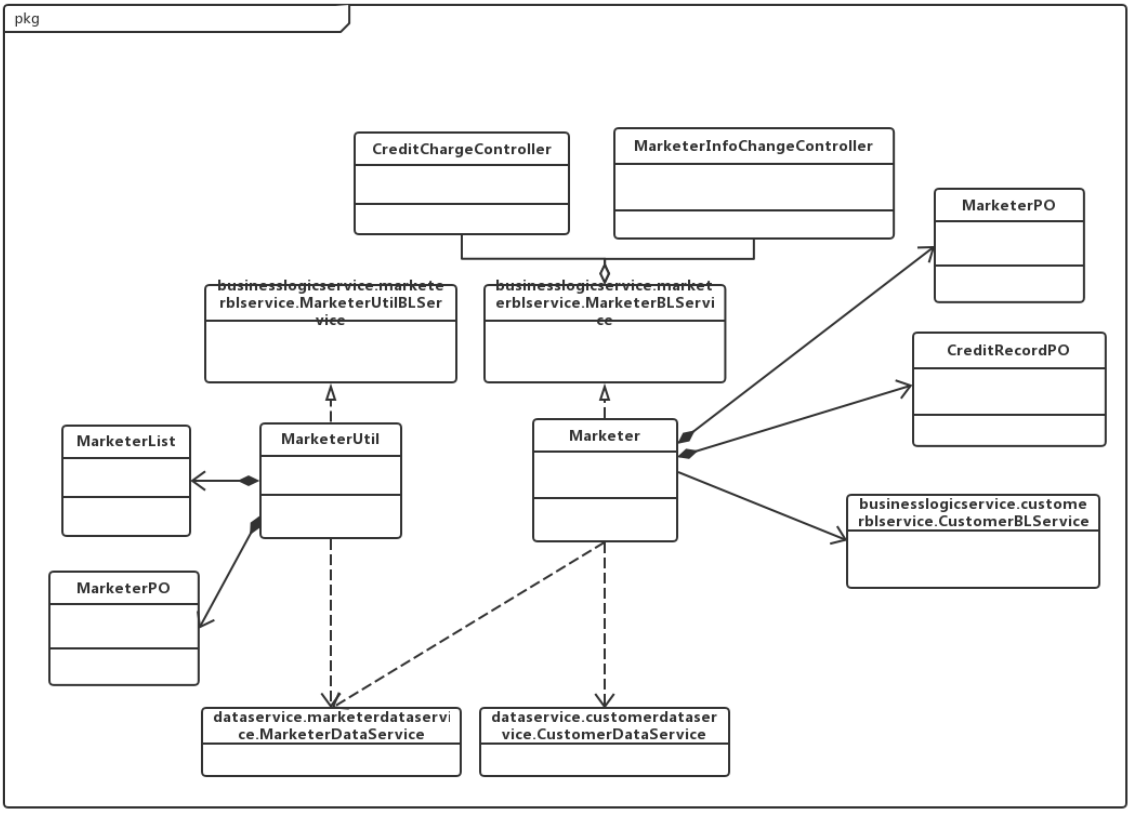


图 4.8.1 Marketer_bl 模块各个类的设计

Marketer_bl 模块各个类的职责如表 4.8.2 所示。

表 4.8.2 Marketer_bl 模块各个类的职责

模块	职责
CreditChargeController	负责实现信用充值界面所需要的服务
MarketerInfoChangeController	负责实现网站营销人员维护个人信息所需要的服务
MarketerUtil	批量处理网站营销人员信息的辅助类

Marketer	网站营销人员的领域模型对象，拥有其个人信息，可以解决为客户充值信用和更改网站营销人员个人信息的问题

（3）模块内部类的接口规范

CreditChargeContorller、MarkerterInfoChangeController、Marketer 和 MarketerUtil 的接口规范如表 4.8.3、4.8.4、4.8.5 和 4.8.6 所示。

表 4.8.3 CreditChargeController 的接口规范

提供的服务（供接口）		
CreditChargeController.creditCharge	语法	public ResultMessage creditCharge(String ID, CustomerVO vo)
	前置条件	已创建一个 Marketer 领域对象，客户信息存在，并且输入符合输入规则
	后置条件	调用 Marketer 领域对象的 creditCharge 方法 调用 Customer 领域对象的 addCredditRecord 方法
需要的服务（需接口）		
服务名	服务	
Marketer.creditCharge	增加客户的信用值	
Customer.addCreditRecord	增加客户的信用记录	

表 4.8.4 MarketerInfoChangeController 的接口规范

提供的服务（供接口）		
MarketerInfoChangeController.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1, String newPw2)

	前置条件	已创建一个 Marketer 领域对象，并且密码输入符合输入规范
	后置条件	调用 Marketer 领域对象的 changePassword 方法
MarketerInfoChangeController.changeInfo	语法	public ResultMessage changeInfo(MarketerVO marketerVO)
	前置条件	已创建一个 Marketer 领域对象，并且输入符合输入规则
	后置条件	调用 Marketer 领域对象的 changeInfo 方法
需要的服务（需接口）		
服务名	服务	
Marketer.changePassword	更改网站营销人员的密码	
Marketer.changeInfo	更改网站营销人员的个人资料	

表 4.8.5 Marketer 的接口规范

提供的服务（供接口）		
Marketer.deleteMarketer	语法	public ResultMessage deleteMarketer(MarketerVO marketerVO)
	前置条件	存在营销人员的信息
	后置条件	在数据库中删除营销人员的信息
Marketer.addMarketer	语法	public ResultMessage addMarketer(MarketerVO marketerVO)
	前置条件	不存在营销人员的信息
	后置条件	在数据库中新增营销人员信息

Marketer.changeInfo	语法	public ResultMessage changeInfo(MarketerVO marketerVO)
	前置条件	存在营销人员信息
	后置条件	更改营销人员的个人信息
Marketer.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1, String newPw2)
	前置条件	存在营销人员信息
	后置条件	更改营销人员的密码
Marketer.creditCharge	语法	public ResultMessage creditCharge(String ID, CustomerVO vo)
	前置条件	存在客户信息
	后置条件	增加客户的信用值，并增添信用记录
需要的服务（需接口）		
服务名	服务	
MarketerDataService.find(String ID)	根据 ID 进行查找单一持久化对象	
MarketerDataService.update(MarketerPO po)	更新单一持久化对象	
CustomerDataService.find(String ID)	根据 ID 进行查找单一持久化对象	
CustomerDataService.update(CustomerPO po)	更新单一持久化对象	

表 4.8.6 MarketerUtil 的接口规范

提供的服务（供接口）		
MarketerUtil.getSingle	语法	public MarketerVO getSingle(String ID)
	前置条件	启动一次营销人员搜索
	后置条件	根据 ID 搜索营销人员的信息
MarketerUtil.getByName	语法	public MarketerVO getByName(String name)
	前置条件	启动一次营销人员搜索
	后置条件	根据姓名搜索营销人员的信息
MarketerUtil.getAll	语法	public List<MarketerVO> getAll()
	前置条件	启动一次营销人员的搜索
	后置条件	得到所有营销人员的列表
需要的服务（需接口）		
服务名	服务	
MarketerDataService.findByID(String ID)	根据 ID 进行查找单一持久化对象	
Marketer.DataService.findByName(String name)	根据姓名进行查找单一持久化对象	
MarketerDataService.findAll()	查找多个持久化对象	

（4）Marketer_bl 模块的动态模型

图 4.8.7 表明了网站营销人员维护个人信息的顺序图

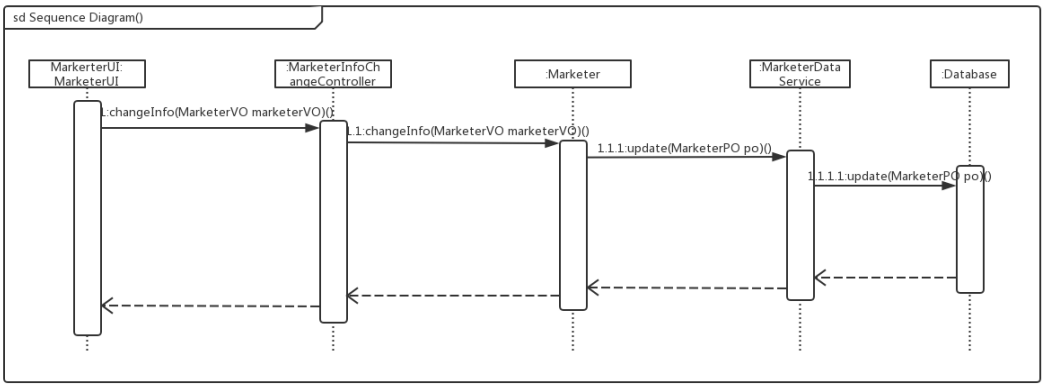


图 4.8.7 网站营销人员维护个人信息的顺序图

如图 4.8.8 所示的状态图描述了 Marketer 对象的生存期间的状态序列、引起转移的时间，以及因状态转移而伴随的动作。随着 changeInfo 方法被 UI 调用，Marketer 进入 find 状态，之后通过修改信息，进入 Modify 状态。

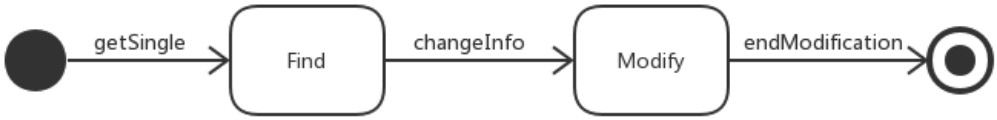


图 4.8.8 网站营销人员对象状态图

（5）Markerter_bl 模块的设计原理

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。

4.9 Manager_bl 模块

（1）模块概述

Manager_bl 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。

Manager_bl 模块的职责及接口参见软件系统结构描述文档。

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加了 businesslogicservice.managerblservice.ManagerBLService, businesslogicservice.managerblservice.ManagerUtilService 接口。业务逻辑层和数据层之间添加 dataservice.managerdataservice.ManagerDataService 接口。为了隔离业务逻辑职责和逻辑控制职责 ,我们增加了 UserAdminController, HotelAdminController, WorkerAdminController，这样会把对管理酒店和用户的业务逻辑处理委托给 Manager 对象。ManagerPO 是作为酒店管理和用户管理的持久化对象被添加到设计模型中去的。

Manager_bl 模块的设计如图 4.9.1 所示。

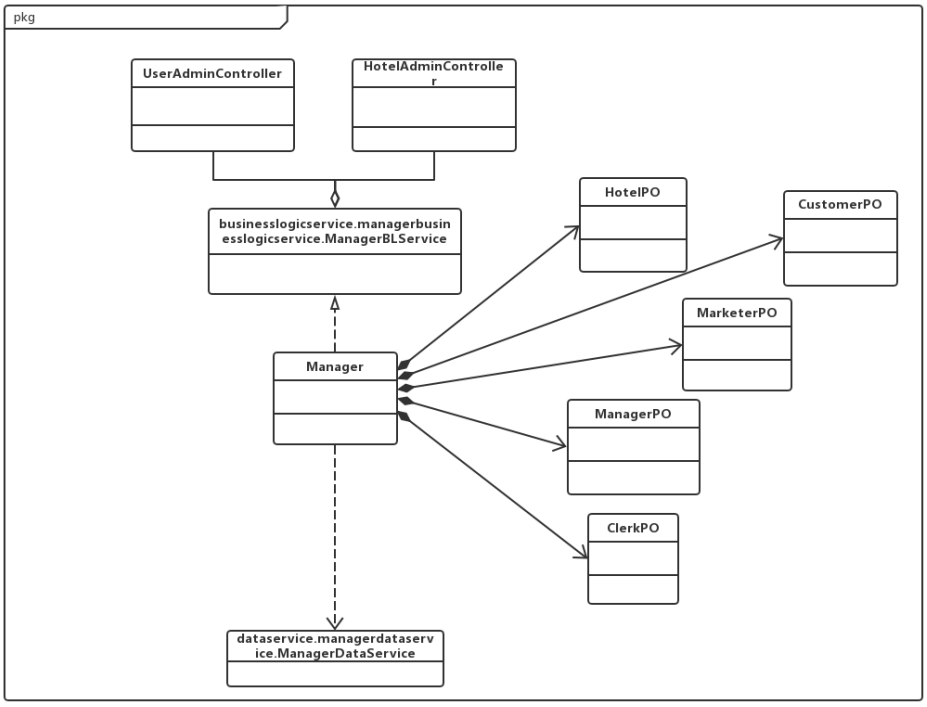


图 4.9.1 managerbl 模块各个类的设计

Manager_bl 模块各个类的职责如表 4.9.2 所示。

表 4.9.2 Manager_bl 模块各个类的职责

模块	职责
UserAdminController	负责实现对应于用户信息管理所需要的服务
HotelAdminController	负责实现对应于酒店信息管理所需要的服务
Manager	网站管理人员的领域模型对象，拥有管理人员的基本信息和密码，可以管理用户和酒店信息

（3）模块内部类的接口规范

HotelAdminController、UserAdminController、Manager 的接口规范如表 4.9.3、4.9.4、4.9.5 所示。

表 4.9.3 HotelAdminController 的接口规范

提供的服务（供接口）		
HotelAdminController.findByID	语法	public HotelVO findByID(String ID)
	前置条件	已存在一个 HotelUtil 对象 ,并且输入符合输入规则
	后置条件	调用 HotelUtil 领域对象的 getByID 方法
HotelAdminController.findByName	语法	public HotelVO findByName(String name)
	前置条件	已存在一个 HotelUtil 对象 ,并且输入符合输入规则
	后置条件	调用 HotelUtil 领域对象的 getByName 方法
HotelAdminController.findByAddress	语法	public HotelVO findByAddress(String address)
	前置条件	已存在一个 Hotel 领域对象，并且输入符合输入规则
	后置条件	调用 HotelUtil 领域对象的 getByAddress 方法
HotelAdminController.addH	语法	public ResultMessage addHotel(HotelVO vo)

otel	前置条件	已存在一个 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
HotelAdminController.deleterHotel	语法	public ResultMessage deleteHotel(HotelVO vo)
	前置条件	已存在一个 Hotel 领域对象
	后置条件	调用 Hotel 领域对象的 deleteHotel 方法
HotelAdminController.updateHotelInfo	语法	public ResultMessage updateHotelInfo (HotelVO vo)
	前置条件	已存在一个 Hotel 领域对象，并且输入符合输入规范
	后置条件	调用 Hotel 领域对象的 modifyHotel 方法
需要的服务（需接口）		
服务名	服务	
HotelUtil.getByID	根据酒店 ID 查询酒店	
HotelUtil.getByName	根据酒店名称查询酒店	
HotelUtil.getByAddress	根据酒店地址查询酒店	
Hotel.addHotel	新增酒店	
Hotel.deleteHotel	删除酒店	
Hotel.modifyHotel	更新酒店	

表 4.9.4 UserAdminController 的接口规范

提供的服务（供接口）		
UserAdminController.findAll	语法	public UserVO findAllByID(String ID)

ByID	前置条件	已存在一个 CustomerUtil、ClerkUtil 和 MarketerUtil 领域对象
	后置条件	调用 CustomerUtil 的 getSingle 方法，ClerkUtil 的 getSingle 方法和 MarketerUtil 的 getSingle 方法
UserAdminController.findAllByName	语法	public UserVO findAllByName(String name)
	前置条件	已存在一个 CustomerUtil、ClerkUtil 和 MarketerUtil 领域对象
	后置条件	调用 CustomerUtil 的 getByName 方法，ClerkUtil 的 getByName 方法和 MarketerUtil 的 getByName 方法
UserAdminController.findAllClerk	语法	public List<ClerkVO> findAllClerk()
	前置条件	已存在一个 ClerkUtil 领域对象
	后置条件	调用 ClerkUtil 的 getAll 方法
UserAdminController.findClerkByName	语法	public List<ClerkVO> findClerkByName(String name)
	前置条件	已存在一个 ClerkUtil 领域对象，并且输入符合输入规则
	后置条件	调用 ClerkUtil 的 getByName 方法
UserAdminController.findClerkByID	语法	public ClerkVO findClerkByID(String ID)
	前置条件	已存在一个 ClerkUtil 领域对象，并且输入符合输入规则

	后置条件	调用 ClerkUtil 的 getSingle 方法
UserAdminController.findAllMarketer	语法	public List<MarketerVO> findAllMarketer()
	前置条件	已存在一个 MarketerUtil 领域对象
	后置条件	调用 MarketerUtil 的 getAll 方法
UserAdminController.findMarketerByID	语法	public MarketerVO findMarketerByID (String ID)
	前置条件	已存在一个 MarketerUtil 领域对象，并且输入符合输入规则
	后置条件	调用 MarketerUtil 的 getSingle 方法
UserAdminController.findMarketerByName	语法	public List<MarketerVO> findMarketerByName (String name)
	前置条件	已存在一个 Marketer 领域对象 ,并且输入符合输入规则
	后置条件	调用 MarketerUtil 的 getByName 方法
UserAdminController.findAllCustomer	语法	public List<CustomerVO> findAllCustomer()
	前置条件	已存在一个 CustomerUtil 领域对象
	后置条件	调用 CustomerUtil 的 getAll 方法
UserAdminController.findCustomerByID	语法	public CustomerVO findCustomerByID(String ID)
	前置条件	已存在一个 CustomerUtil 领域对象 ,并且输入符合输入规则
	后置条件	调用 CustomerUtil 的 getSingle 方法

UserAdminController.findCustomerByName	语法	public List<CustomerVO> findCustomerByName (String name)
	前置条件	已存在一个 CustomerUtil 领域对象 ,并且输入符合输入规则
	后置条件	调用 CustomerUtil 的 getByName 方法
UserAdminController.addClerk	语法	public ResultMessage addClerk(ClerkVO vo)
	前置条件	已存在一个 Clerk 领域对象，酒店工作人员不存在,需要新增,酒店已存在,且无工作人员,输入符合输入规则
	后置条件	调用 Clerk 的 addClerk 方法
UserAdminController.addMarketer	语法	public ResultMessage addMarketer (MarketerVO vo)
	前置条件	已存在一个 Marketer 领域对象 ,网站营销人员不存在,需要新增,输入符合输入规则
	后置条件	调用 Marketer 的 addMarketer 方法
UserAdminController.deleteClerk	语法	public ResultMessage deleteClerk(ClerkVO vo)
	前置条件	已存在一个 Clerk 领域对象
	后置条件	调用 Clerk 的 deleteClerk 方法
UserAdminController.deleteMarketer	语法	public ResultMessage deleteMarketer (MarketerVO vo)
	前置条件	已存在一个 Marketer 领域对象
	后置条件	调用 Marketer 的 deleteMarketer 方法

UserAdminController.updateCustomerInfo	语法	public ResultMessage updateCustomerInfo (CustomerVO vo)
	前置条件	已存在一个 Customer 领域对象，输入符合输入规范
	后置条件	调用 Customer 的 changeInfo 方法
UserAdminController.updateClerkInfo	语法	public ResultMessage updateClerkInfo (ClerkVO vo)
	前置条件	已存在一个 Clerk 领域对象，输入符合输入规范
	后置条件	调用 Clerk 的 changeInfo 方法
UserAdminController.updateMarketerInfo	语法	public ResultMessage updateMarketerInfo (MarketerVO vo)
	前置条件	已存在一个 Marketer 领域对象 ,输入符合输入规范
	后置条件	调用 Marketer 的 changeInfo 方法
需要的服务（需接口）		
服务名	服务	
CustomerUtil.getSingle	根据 ID 查找客户	
ClerkUtil.getSingle	根据 ID 查找酒店工作人员	
MarketerUtil.getSingle	根据 ID 查找网站营销人员	
CustomerUtil.getByName	根据姓名查找客户	
ClerkUtil.getByName	根据姓名查找酒店工作人员	
MarketerUtil.getByName	根据姓名查找网站营销人员	
ClerkUtil.getAll	查询全体酒店工作人员	

MarketerUtil.getAll	查询全体网站营销人员
CustomerUtil.getAll	查询全体客户
Clerk.addClerk	新增酒店工作人员
Marketer.addMarketer	新增网站营销人员
Clerk.deleteClerk	删除酒店工作人员
Marketer.deleteMarketer	删除网站营销人员
Customer.changeInfo	更改客户信息
Clerk.changeInfo	更改酒店工作人员信息
Marketer.changeInfo	更改网站营销人员信息

表 4.9.5 Manager 的接口规范

提供的服务（供接口）		
Manager.changePassword	语法	public ResultMessage changePassword(String ID, String oldPw, String newPw1, String newPw2)
	前置条件	存在管理人员的信息
	后置条件	更改网站管理人员的密码
Manager.changeInfo	语法	public ResultMessage changeInfo(ManagerVO vo)
	前置条件	已存在管理人员的信息
	后置条件	更新管理人员的信息
需要的服务（需接口）		

服务名	服务
ManagerDataService.update(ManagerPO po)	更新持久化对象
MarketerDataService.find(String ID)	根据 ID 查找持久化对象

（4）Manager_bl 模块的动态模型

图 4.9.6 表明了网站管理人员新增酒店的顺序图。

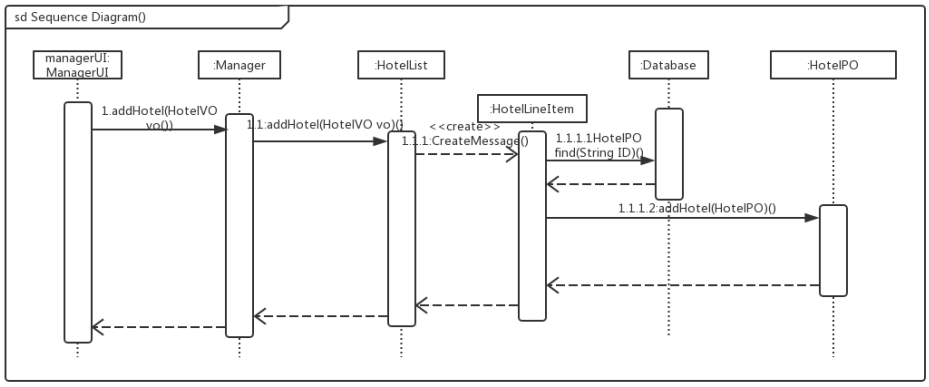


图 4.9.6 网站管理人员新增酒店的顺序图

如图 4.9.7 所示的状态图描述了 Manager 对象的生存期间的状态序列、引起转移的时间，以及因状态转移而伴随的动作。随着 changeInfo 方法被 UI 调用，Manager 进入 find 状态，之后通过修改信息，进入 Modify 状态。

图 4.9.7 网站管理人员对象状态图

（5）Manager_bl 模块的设计原理

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。

5. 依赖视角

图 5.1 和图 5.2 是客户端和服务器端各自的包之间的依赖关系。

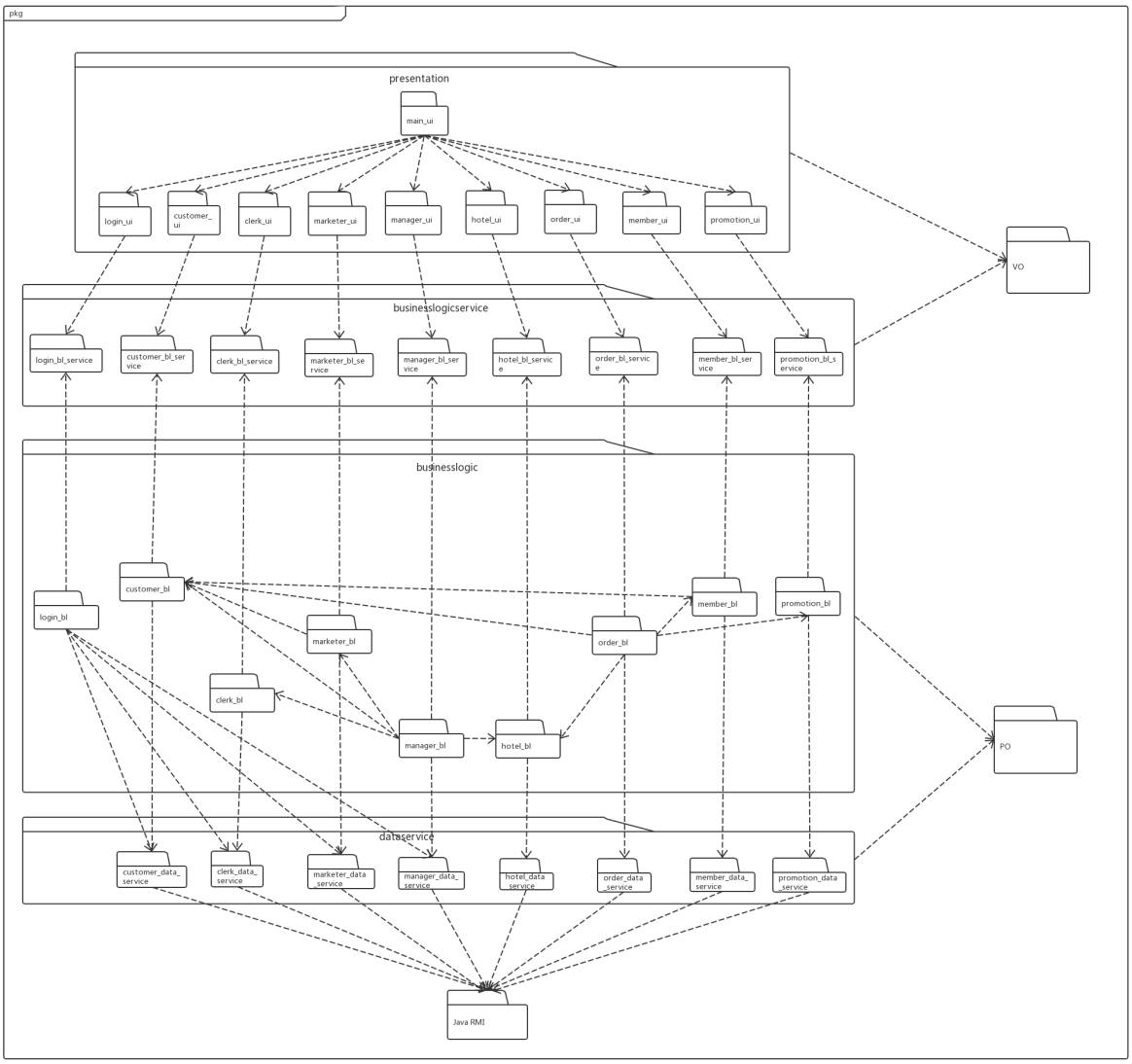


图 5.1 客户端包图

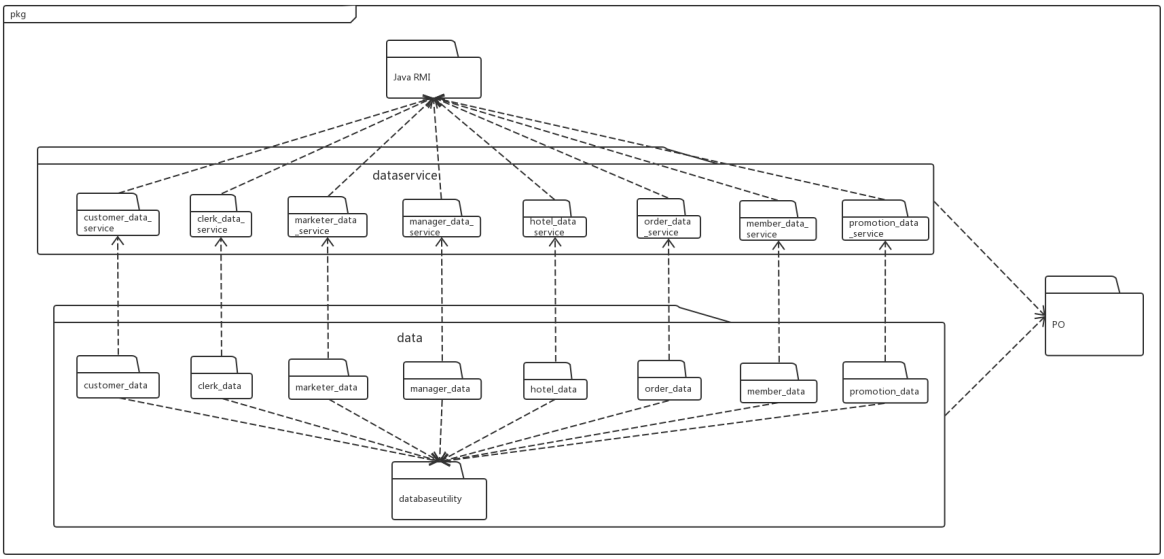

pkg
Java RMI
dataservice
customer_data_service
clerk_data_service
marketer_data_service
manager_data_service
hotel_data service
order_data_service
member_data_service
promotion_data_service
data
customer_data
clerk_data
marketer_data
manager_data
hotel_data
order_data
member_data
promotion_data
databaseutility
PO

图 5.2 服务器端包图