影院管理系统（CMS）软件详细设计文档

文档修改历史

修改人员	日期	修改原因	版本号
Agent_67	2019.5.25	初始化	v0.0
Agent_67	2019.6.2	第一次整合	v1.0
Agent_67	2019.6.17	加入类设计图和顺序图	v2.0

1. 引言

1.1 编制目的

本报告详细完成对连续商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
CMS	影院管理系统	Cinema Management System

1.3 参考资料

1. IEEE标准。
2. 影院管理系统用例文档。
3. 影院管理系统需求规格说明文档。
4. 影院管理系统体系结构文档。
5. 骆斌，丁二玉，刘钦 - 软件工程与计算 . 卷二 , 软件开发的技术基础 : Software engineering and computing . Volume Ⅱ , Fundamentals of software development technology。

2. 产品描述

参考影院管理系统用例文档和影院管理系统需求规格说明文档中对产品的概括描述。

3. 系统结构设计概述

参考影院管理系统体系结构文档中对体系结构设计的描述。

4. 结构视角

4.1 业务逻辑层的分解

业务逻辑层的开发包图参见软件体系结构文档图2。

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功能需求及相关非功能需求。

userbl模块的职责及接口参见软件体系结构文档表10。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加userdataservice.UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的UserController，这样UserController会将对用户登录注册登出的逻辑处理委托给UserBL。

userbl模块的设计如图1所示。

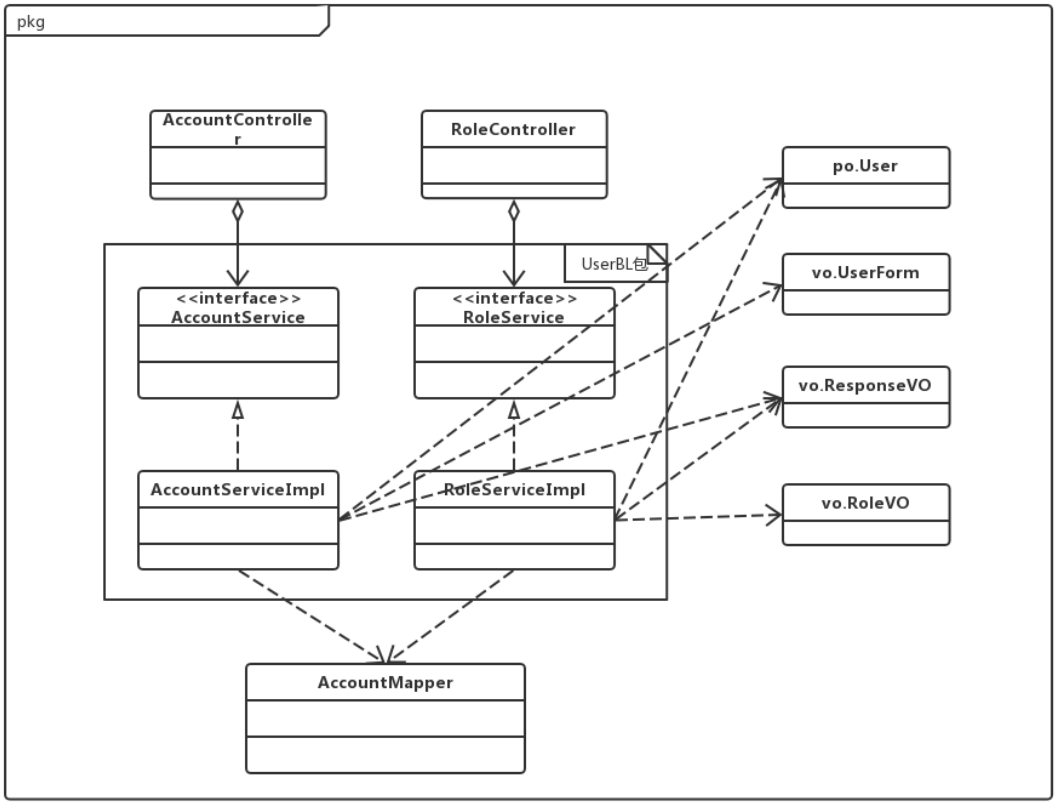


图1 userbl模块各个类的设计

userbl模块各个类的职责如表1所示。

表1 userbl模块各个类的职责

模块	职责
AccountService	负责实现用户登录注册登出的服务
RoleService	负责实现影院角色的添加修改删除的服务

(3) 模块内部类的接口规范

AccountService的接口规范如表2所示。

表2 AccountService模块的接口规范

提供的服务（供接口）		
AccountService.login	**语法**	public User login(UserForm userForm)
	前置条件	userName符合输入条件
	后置条件	根据username查找是否存在相应的User，如果username和password相匹配，则建立session会话，最后返回登录验证的结果
AccountService.registerAccount	**语法**	public ResponseVO registerAccount(UserForm userForm)
	前置条件	userFrom符合输入条件
	后置条件	根据UserForm查找是否已经存在对应的username，如果没有，则增加一条新的用户账号记录，最后返回注册的结果
AccountService.checkPassword	**语法**	public ResponseVO checkPassword(User user, String rawPassword)
	前置条件	rawPassword符合输入条件
	后置条件	根据用户输入的密码返回验证结果
AccountService.editPassword	**语法**	public ResponseVO editPassword(UserForm userForm)
	前置条件	用户账号存在
	后置条件	根据userForm查找用户的账号并修改账号密码，返回修改的结果

需要的服务（需接口）

服务名	服务
AccountMapper.createNewAccount(String username, String password, Integer auth)	注册账号
AccountMapper.getAccountByName(String username)	根据用户名查找账号
AccountMapper.updatePassword(UserForm userForm)	修改密码

RoleService的接口规范如表3所示。

表3 RoleService模块的接口规范

提供的服务（供接口）

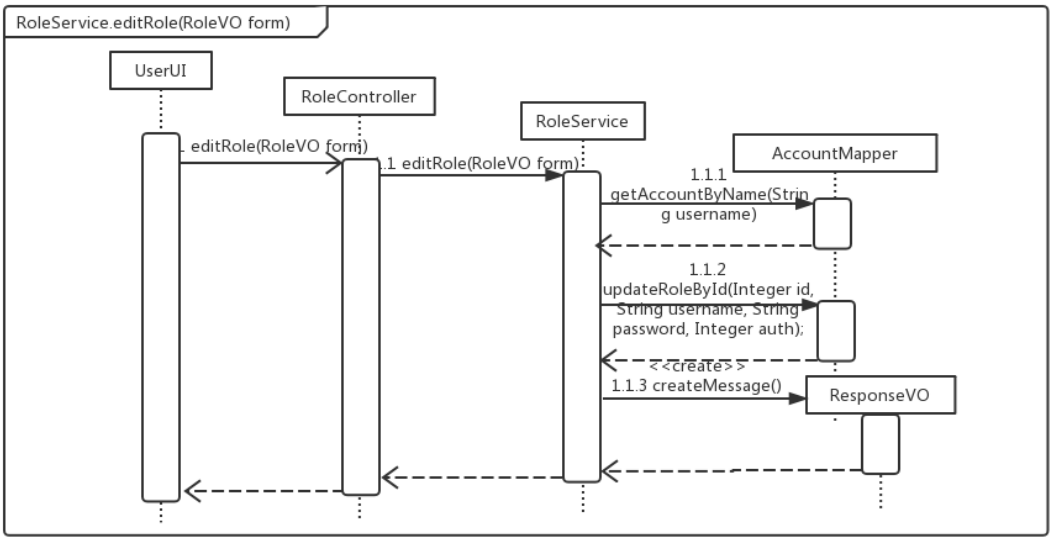
RoleService.getAdaptableRoles	**语法**	public ResponseVO getAdaptableRoles(Integer level)
	前置条件	输入的等级是有效等级
	后置条件	如果等级是有效的，根据等级查找并返回该等级可修改的影院角色
RoleService.deleteRole	**语法**	public ResponseVO deleteRole(Integer id)
	前置条件	id符合输入条件
	后置条件	如果该id是有效的，根据id查找并删除该影院角色，返回删除结果
RoleService.editRole	**语法**	public ResponseVO editRole(RoleVO form)
	前置条件	from符合输入条件
	后置条件	修改影院角色并返回修改结果
RoleService.addRole	**语法**	public ResponseVO addRole(RoleVO form)
	前置条件	from符合输入条件
	后置条件	根据from添加影院角色，并返回添加结果

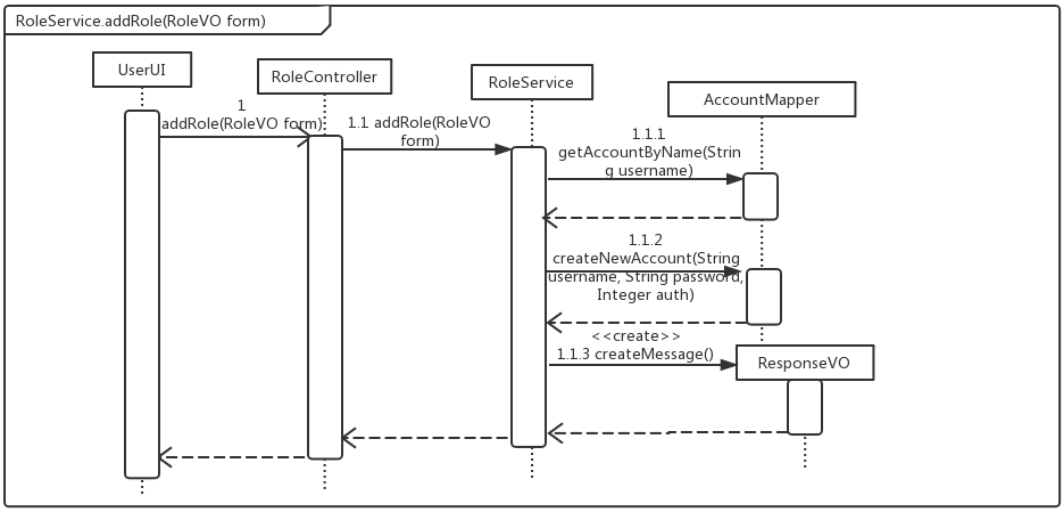
需要的服务（需接口）

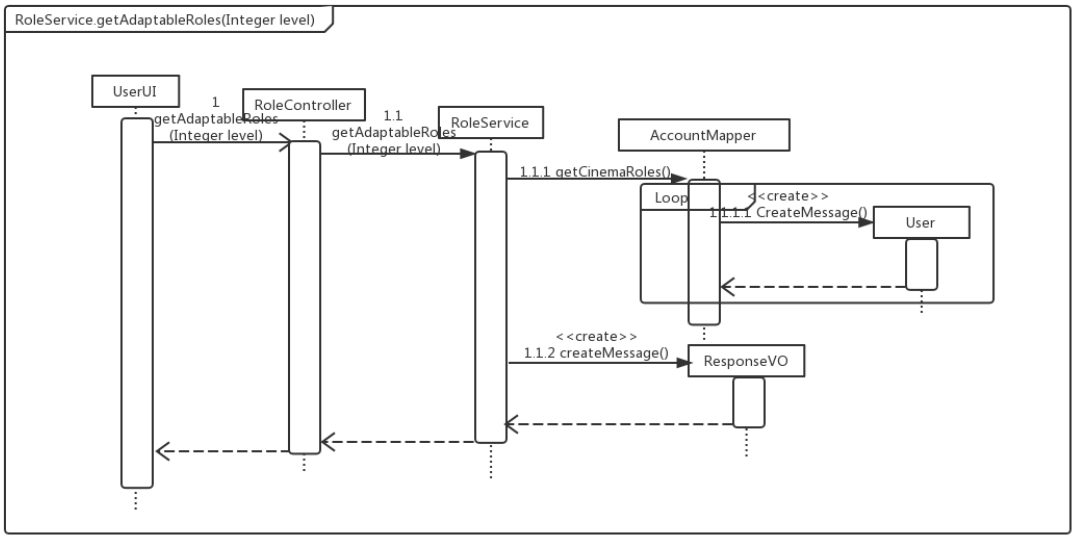
服务名	服务
AccountMapper.getCinemaRoles()	获取全部影院角色
AccountMapper.deleteRoleById(Integer id)	删除影院角色
AccountMapper.updateRoleById(Integer id, String username, String password, Integer auth)	更新角色信息

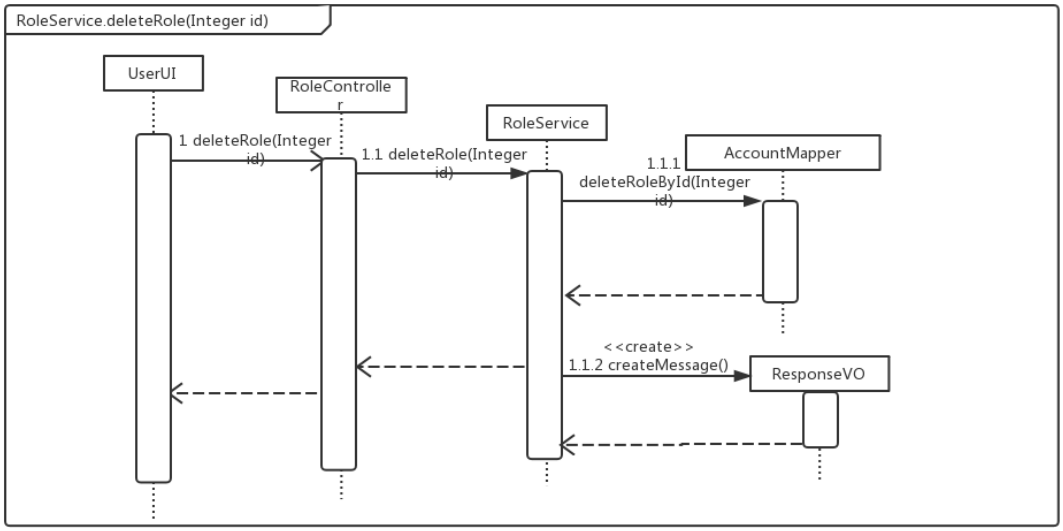
(4) 业务逻辑层的动态模型

图1~4是影院管理系统中UserBL进行业务逻辑处理时的顺序图。









由于UserBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，最后封装成ResponseVO返回给上层。

4.1.2 salesbl模块

(1) 模块概述
salesbl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。

salesbl模块的职责及接口参见软件体系结构文档表24。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加salesdataservice.SalesDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的SalesController，这样SalesController会将操作优惠活动相关的逻辑处理委托给SalesBL。

salesbl模块的设计如图3所示。

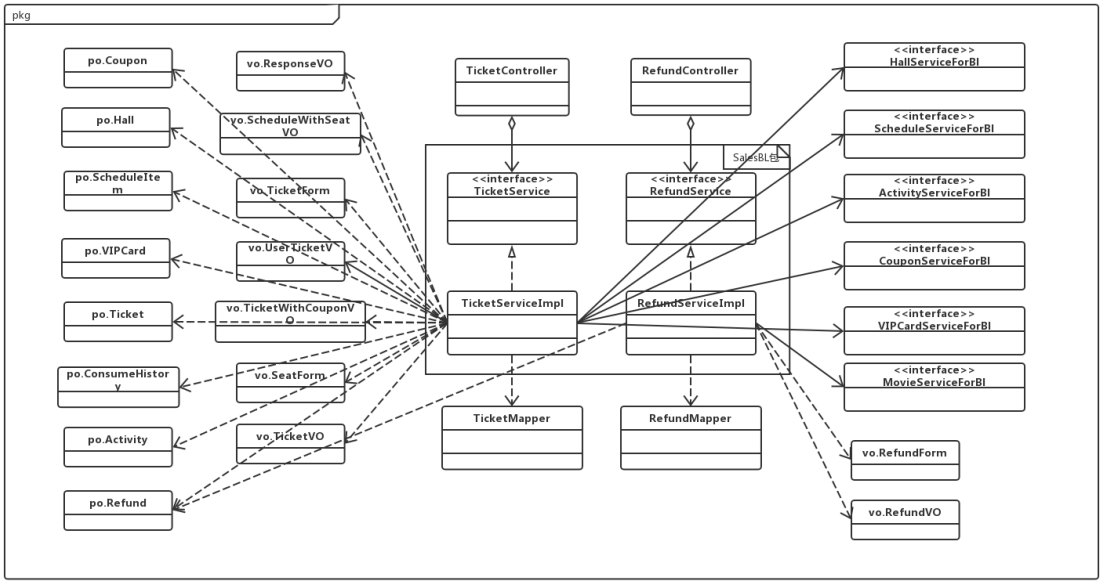


图3 salesbl模块各个类的设计

salesbl模块各个类的职责如表4所示。

表4 salesbl模块各个类的职责

模块	职责
TicketService	负责实现操作电影票的服务
RefundService	负责实现退票功能所需要的服务

(3) 模块内部类的接口规范
TicketService的接口规范如表5

表5 TicketService模块的接口规范

提供的服务（供接口）		
TicketService.completeByVIPCard	**语法**	public ResponseVO completeByVIPCard(List id, int couponId)
	前置条件	id不为空
	后置条件	根据ticket查找是否存在相应的会员卡，根据couponId查找是否存在相应的优惠券，存在则完成购买电影票流程，返回购买结果
TicketService.addTicket	**语法**	public ResponseVO addTicket(TicketForm ticketForm)
	前置条件	ticketFrom符合条件
	后置条件	根据ticketFrom锁座，返回锁座结果
TicketService.completeTicket	**语法**	public ResponseVO completeTicket(List id, int couponId)
	前置条件	id不为空
	后置条件	根据ticketAndCouponVO查找并购买电影票，并返回购票结果
TicketService.getTicketByUser	**语法**	public ResponseVO getTicketByUser(int userId)
	前置条件	userId有效
	后置条件	根据userId查找并返回用户买过的电影票

TicketService.getBySchedule	**语法**	public ResponseVO getBySchedule(int scheduleId)
	前置条件	scheduleId是有效id
	后置条件	根据scheduleId查找并返回该排片的锁座情况
TicketService.cancelTicket	**语法**	public ResponseVO cancelTicket(List id)
	前置条件	id不为空
	后置条件	根据ticketId查找并删除电影票，返回删除结果
TicketService.issueTicket	**语法**	public ResponseVO issueTicket(int id)
	前置条件	id为有效电影票id
	后置条件	根据ticketId查找并修改电影票状态为出票，返回出票结果

需要的服务（需接口）

服务名	服务
TicketMapper.insertTicket(Ticket ticket)	添加单张电影票
TicketMapper.insertTickets(List tickets)	添加多张电影票
TicketMapper.deleteTicket(int ticketId)	删除电影票
TicketMapper.selectTicketByUser(int userId)	获得用户买过的票
TicketMapper.deleteLockedTicket(int userId, int scheduleId)	删除被锁座位的票
TicketMapper.updateTicketState(int ticketId, int state)	更新电影票状态
TicketMapper.updateTicketActualPay(int ticketId, double actualPay)	更新票的实际支付金额
TicketMapper.selectTicketsBySchedule(int scheduleId)	根据排片获取电影票
TicketMapper.selectTicketByScheduleIdAndSeatint scheduleId, int columnIndex, int rowIndex)	根据排片和座位获取电影票
TicketMapper.selectTicketById(int id)	根据id获取电影票
ConsumeService.addConsumeHistory(Integer userId, Double money, Double discount, String consumeType, Integer type, Integer contentId)	添加消费记录
ScheduleServiceForBl.getScheduleItemById(int id)	根据id查找排片信息
HallServiceForBl.getHallById(int id)	根据id查找影厅
CouponServiceForBl.getCoupon(int couponId)	根据id获取优惠券
CouponServiceForBl.issueCoupon(int couponId,int userId)	根据userId和couponId新增一条优惠券信息
CouponServiceForBl.getCouponsByUser(int userId)	根据userId获取优惠券
ActivityServiceForBl.getActivitiesByMovie(int movieId)	根据movieId获取优惠活动
ActivityServiceForBl.getActivityList()	获取所有优惠活动
VIPServiceForBl.getCardByUserId(int userId)	根据userId获取会员卡
VIPServiceForBl.payByCard(int id, double balance)	使用会员卡支付
RefundServiceForBl.getRefundByMovieId(int movieId)	根据movieId获取退票策略

RefundService的接口规范如表5

表5 RefundService模块的接口规范

提供的服务（供接口）

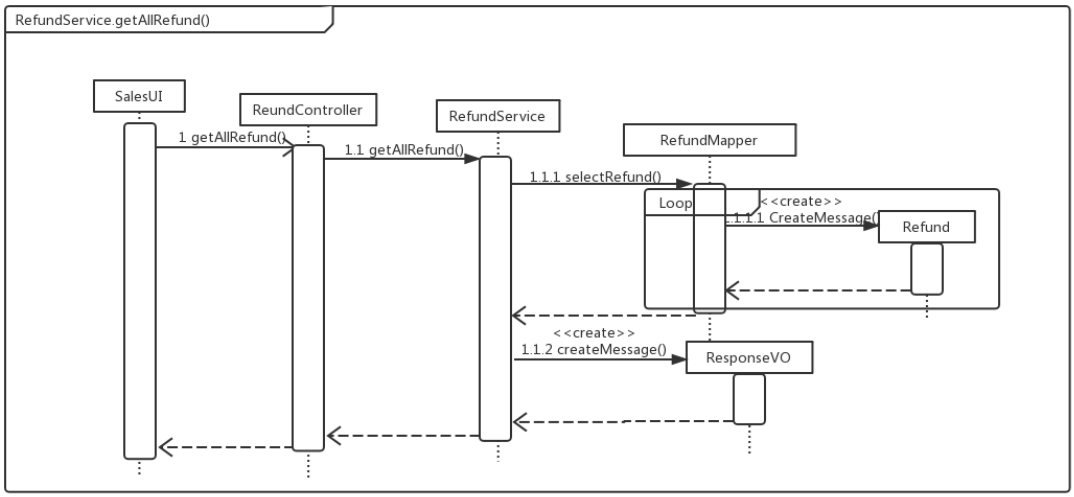
RefundService.getAllRefund	**语法**	public ResponseVO getAllRefund()
	前置条件	无
	后置条件	获取并返回所有退票策略
RefundService.publishRefund	**语法**	public ResponseVO publishRefund(RefundForm refundForm)
	前置条件	refundFrom符合输入条件
	后置条件	根据refundFrom新增一条退票策略，返回新增结果
RefundService.deleteRefund	**语法**	public ResponseVO deleteRefund(int refundId)
	前置条件	refundId存在
	后置条件	根据refundId查找并删除退票策略，返回删除结果
RefundService.getRefundById	**语法**	public ResponseVO getRefundById(int refundId)
	前置条件	refundId存在
	后置条件	根据refundId查找并返回退票策略

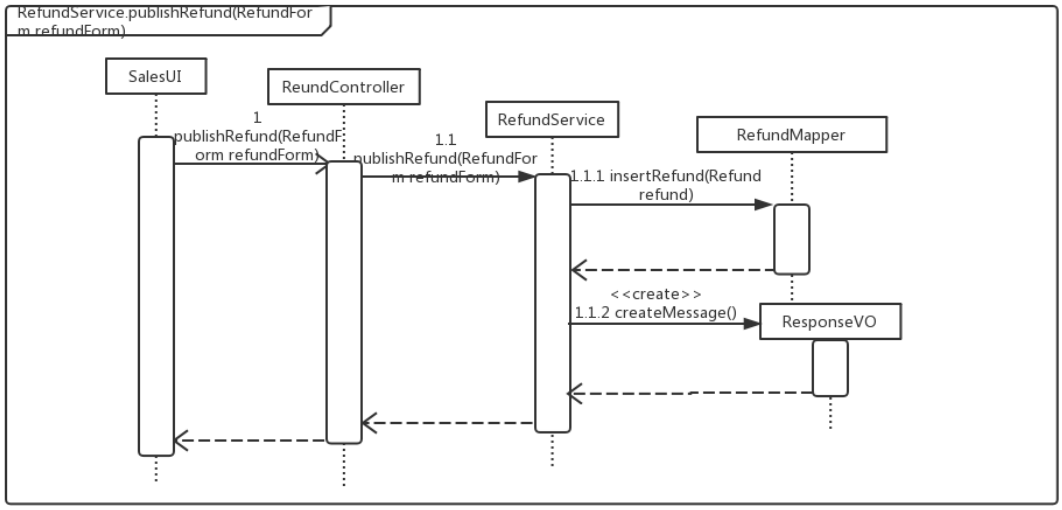
需要的服务（需接口）

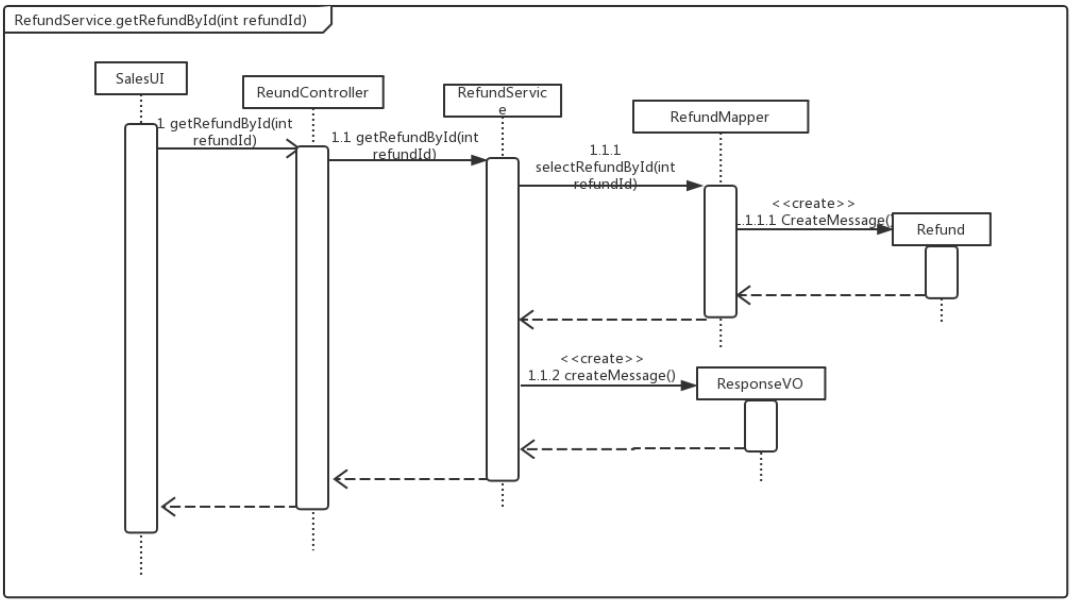
服务名	**服务**
RefundMapper.selectRefund()	获得所有退票策略
RefundMapper.insertRefund(Refund refund)	新增退票策略
RefundMapper.deleteRefundById(int refundId)	根据id删除一条退票策略
RefundMapper.selectRefundByMovieId(int movieId)	根据电影获取退票策略
RefundMapper.selectRefundById(int refundId)	根据id获取退票策略
MovieServiceForBl.getMovieById(int id)	根据id查找电影

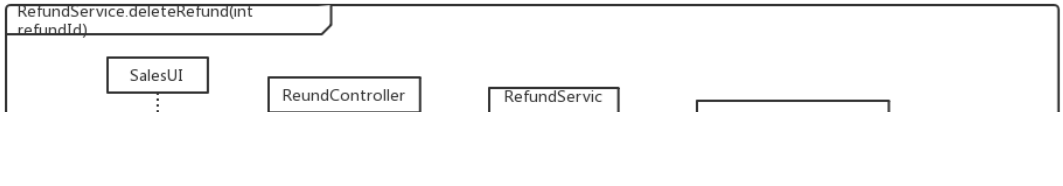
(4) 业务逻辑层的动态模型

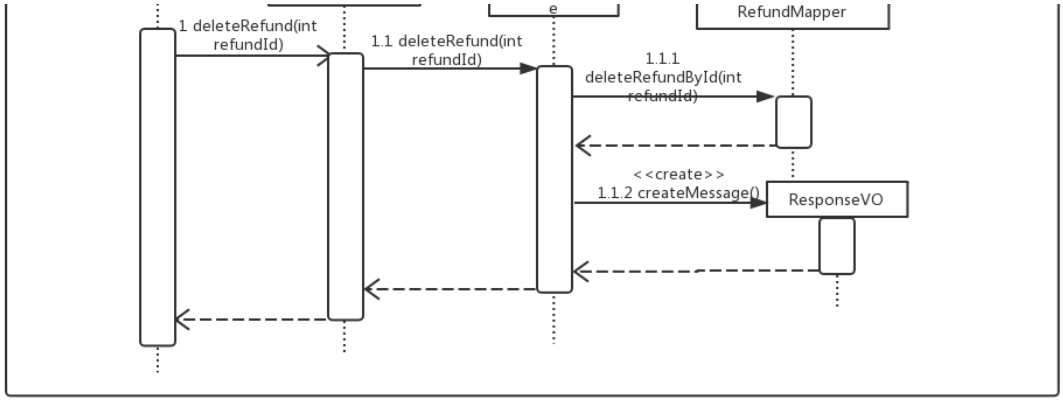
图5~8是影院管理系统中SalesBL进行业务逻辑处理时的顺序图。


RefundService.getAllRefund()
SalesUI
ReundController
RefundService
RefundMapper
1 getAllRefund()
1.1 getAllRefund()
1.1.1 selectRefund()
Loop
<<create>>
1.1.1 CreateMessage()
Refund
<<create>>
1.1.2 createMessage()
ResponseVO


RefundService.publishRefund(RefundFor
m refundForm)
SalesUI
ReundController
RefundService
RefundMapper
1
publishRefund(RefundF
orm refundForm)
1.1
publishRefund(RefundFor
m refundForm)
1.1.1 insertRefund(Refund
refund)
<<create>>
1.1.2 createMessage()
ResponseVO


RefundService.getRefundById(int refundId)
SalesUI
ReundController
RefundServic
e
RefundMapper
1 getRefundById(int
refundId)
1.1 getRefundById(int
refundId)
1.1.1
selectRefundById(int
refundId)
<<create>>
1.1.1 CreateMessage()
Refund
<<create>>
1.1.2 createMessage()
ResponseVO


RefundService.deleteRefund(int
refundId)
SalesUI
ReundController
RefundServic



由于SalesBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，最后封装成ResponseVO返回给上层。

4.1.3 statisticsbl模块

(1)模块概述

statisticsbl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。

statisticsbl模块的职责及接口参见软件体系结构文档表Y。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加statisticsdataservice.MovieLikeDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的StatisticsController，这样StatisticsController会将电影统计数据相关的逻辑处理委托给MovieLikeBL和StatisticsBL。

statisticsbl模块的设计如图9所示。

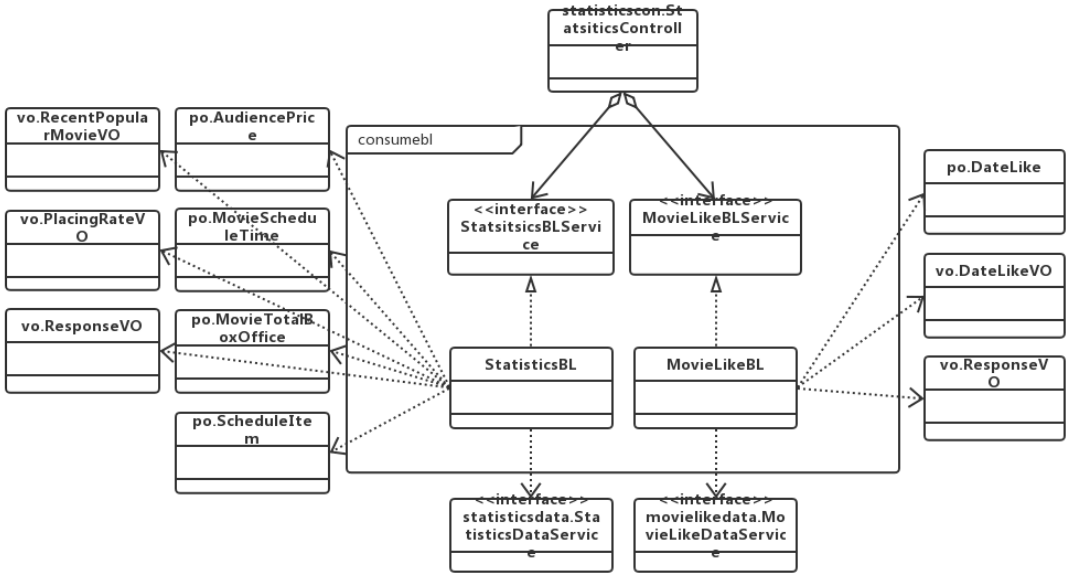


图9 statisticsbl模块各个类的设计

statisticsbl模块各个类的职责如表6所示。

表6 statisticsbl模块各个类的职责

模块	职责
MovieLikeBL	负责实现想看电影相关的服务
StatisticsBL	负责实现获取电影统计数据所需要的服务

(3) 模块内部类的接口规范

MovieLikeBL和StatisticsBL的接口规范如表7~8所示。

表7 MovieLikeBL模块的接口规范

提供的服务（供接口）		
MovieLikeBL.likeMovie	**语法**	ResponseVO likeMovie(int userId, int movieId)
	前置条件	无
	后置条件	标记电影为想看
MovieLikeBL.unLikeMovie	**语法**	ResponseVO unlikeMovie(int userId, int movieId)
	前置条件	无
	后置条件	取消电影为想看
MovieLikeBL.getCountOfLikes	**语法**	ResponseVO getCountOfLikes(int movieId)
	前置条件	无
	后置条件	统计想看电影的人数
MovieLikeBL.getLikeNumsGroupByDate	**语法**	ResponseVO getLikeNumsGroupByDate(int movieId)
	前置条件	无
	后置条件	获得电影每日的想看人数

需要的服务（需接口）

服务名	服务
MovieLikeDataService.insertOneLike(int movieId,int userId)	插入一条想看记录
MovieLikeDataService.deleteOneLike(int movieId,int userId)	删除一条想看记录
MovieLikeDataService.selectLikeNums(int movieId)	根据movieId查看电影的想看人数
MovieLikeDataService.selectLikeMovie(int movieId,int userId)	根据movieId和userId查找特定想看记录
MovieLikeDataService.selectMovieScheduleTimes(Date date, Date nextDate)	查询date当天每部电影的排片次数
MovieLikeDataService.getDateLikeNum(int movieId)	根据movieId获取按日期统计的想看人数

表8 StatisticsBL模块的接口规范

提供的服务（供接口）

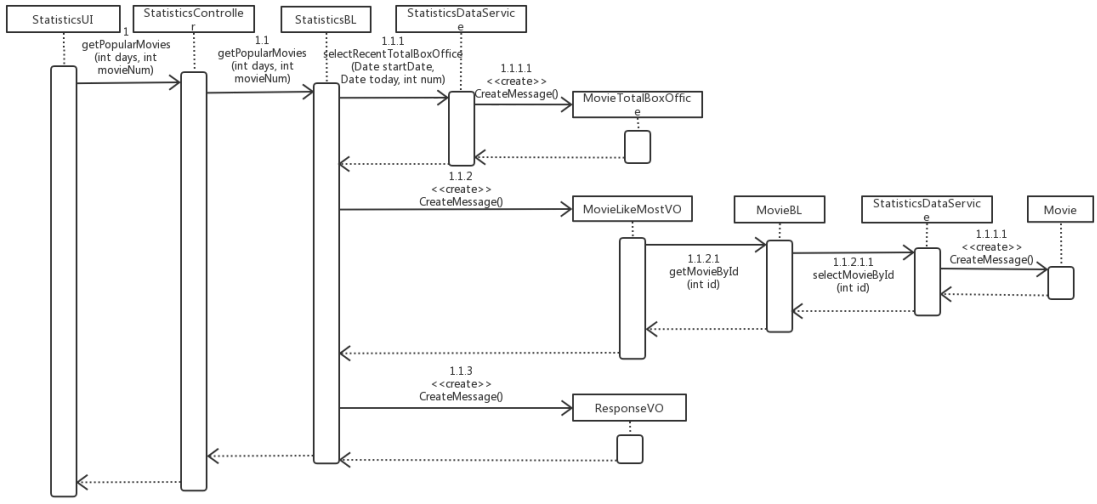
StatisticsBL.getScheduleRateByDate	**语法**	ResponseVO getScheduleRateByDate(Date date)
	前置条件	无
	后置条件	获取某日各影片排片率统计数据
StatisticsBL.getTotalBoxOffice	**语法**	ResponseVO getTotalBoxOffice()
	前置条件	无
	后置条件	获取所有电影的累计票房(降序排序，且包含已下架的电影)
StatisticsBL.getAudiencePriceSevenDays	**语法**	ResponseVO getAudiencePriceSevenDays()
	前置条件	无
	后置条件	获得过去7天内每天客单价
StatisticsBL.getMoviePlacingRateByDate	**语法**	ResponseVO getMoviePlacingRateByDate(Date date)
	前置条件	无
	后置条件	获取所有电影某天的上座率
StatisticsBL.getPopularMovies	**语法**	ResponseVO getPopularMovies(int days, int movieNum)
	前置条件	无
	后置条件	获取最近days天内，票房最高的的movieNum个电影

需要的服务（需接口）

服务名	服务
StatisticsDataService.selectMovieScheduleTimes(Date date, Date nextDate)	查询date当天每部电影的排片次数
StatisticsDataService.getDateLikeNum(int movieId)	根据movieId获取按日期统计的想看人数
StatisticsDataService.selectMovieTotalBoxOffice()	查询所有电影的总票房（包括已经下架的，降序排列）
StatisticsDataService.selectAudiencePrice(Date date,Date nextDate)	查询date当天每个客户的购票金额
StatisticsDataService.selectPlacingRate(Date date,Date nextDate)	查询date当天各电影上座率
StatisticsDataService.selectRecentTotalBoxOffice(Date startDate,Date today,int num)	查询startDate到today日期内票房最高的num部电影

(4) 业务逻辑层的动态模型

图10是影院管理系统中StatisticsBL进行业务逻辑处理时的顺序图。



由于MovieLikeBL和StatisticsBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.4 consumebl模块

(1)模块概述

consumebl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。

consumebl模块的职责及接口参见软件体系结构文档表Y。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加consumedataservice.ConsumeDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的ConsumeController，这样ConsumeController会将电影统计数据相关的逻辑处理委托给ConsumeBL。

consumebl模块的设计如图11所示。

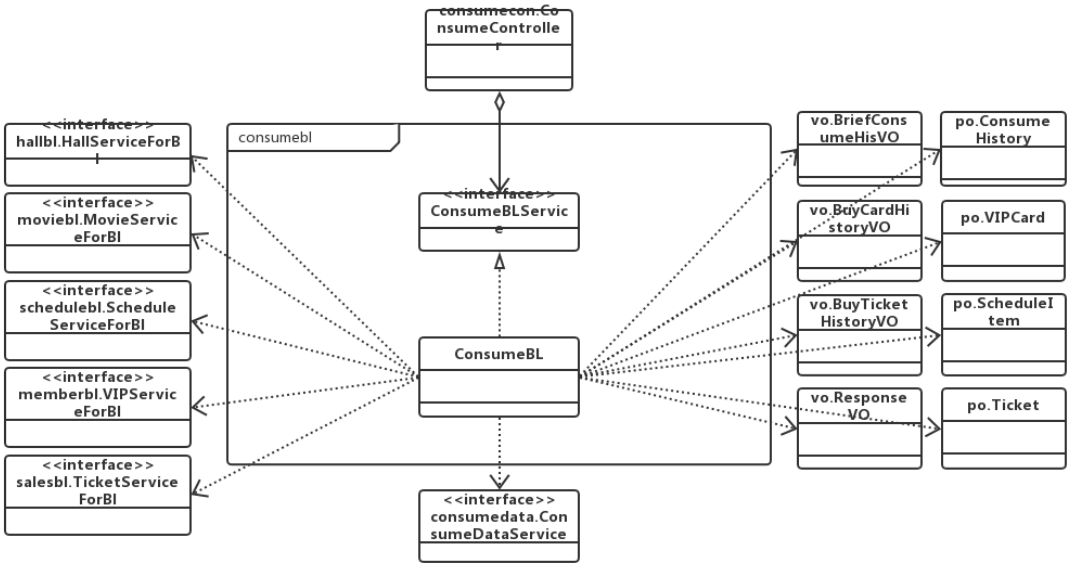


图11 consumebl模块各个类的设计

consumebl模块各个类的职责如表9所示。

表9consumebl模块各个类的职责

模块	职责
ConsumeBL	负责实现充值消费记录相关的服务

(3) 模块内部类的接口规范

ConsumeBL的接口规范如表10所示。

表10 ConsumeBL模块的接口规范

提供的服务（供接口）		
ConSumeBL.getAllTopUpHistory	**语法**	ResponseVO getAllTopUpHistory(Integer userId)
	前置条件	无
	后置条件	获取用户全部的充值记录
ConSumeBL.getBriefConsumeHis	**语法**	ResponseVO getBriefConsumeHis(Integer userId)
	前置条件	无
	后置条件	获取用户简略消费记录信息
ConSumeBL.getConsumeHisDetail	**语法**	ResponseVO getConsumeHisDetail(Integer id)
	前置条件	无
	后置条件	获取用户消费记录详细信息
ConSumeBL.addTopUpHistory	**语法**	ResponseVO addTopUpHistory(Integer userId, Double money, Double discount, Double balance, Timestamp time)
	前置条件	无

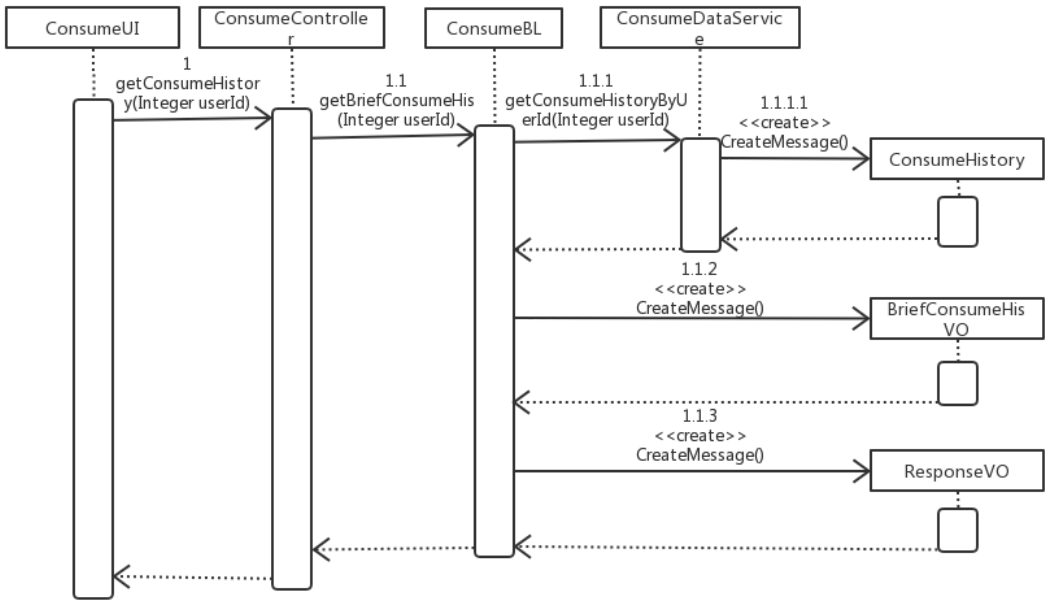
	后置条件	添加充值记录
ConSumeBL.addConsumeHistory	**语法**	ResponseVO addConsumeHistory(Integer userId, Double money, Double discount, String consumeType,Integer type, Integer contentId)
	前置条件	无
	后置条件	添加消费记录
ConSumeBL.getConsumeQualifiedUsers	**语法**	ResponseVO getConsumeQualifiedUsers(Double totalConsume)
	前置条件	无
	后置条件	获取消费总额满一定值的用户信息

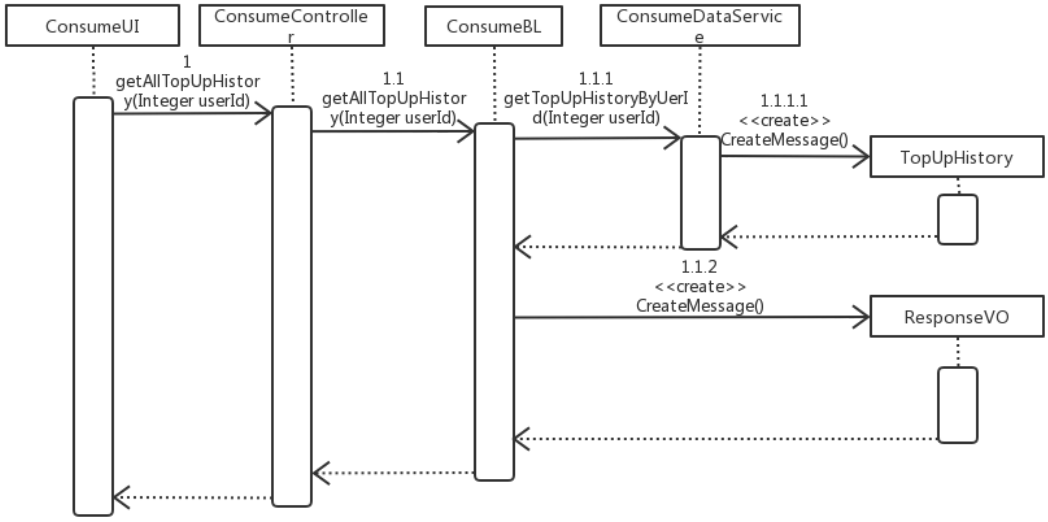
需要的服务（需接口）

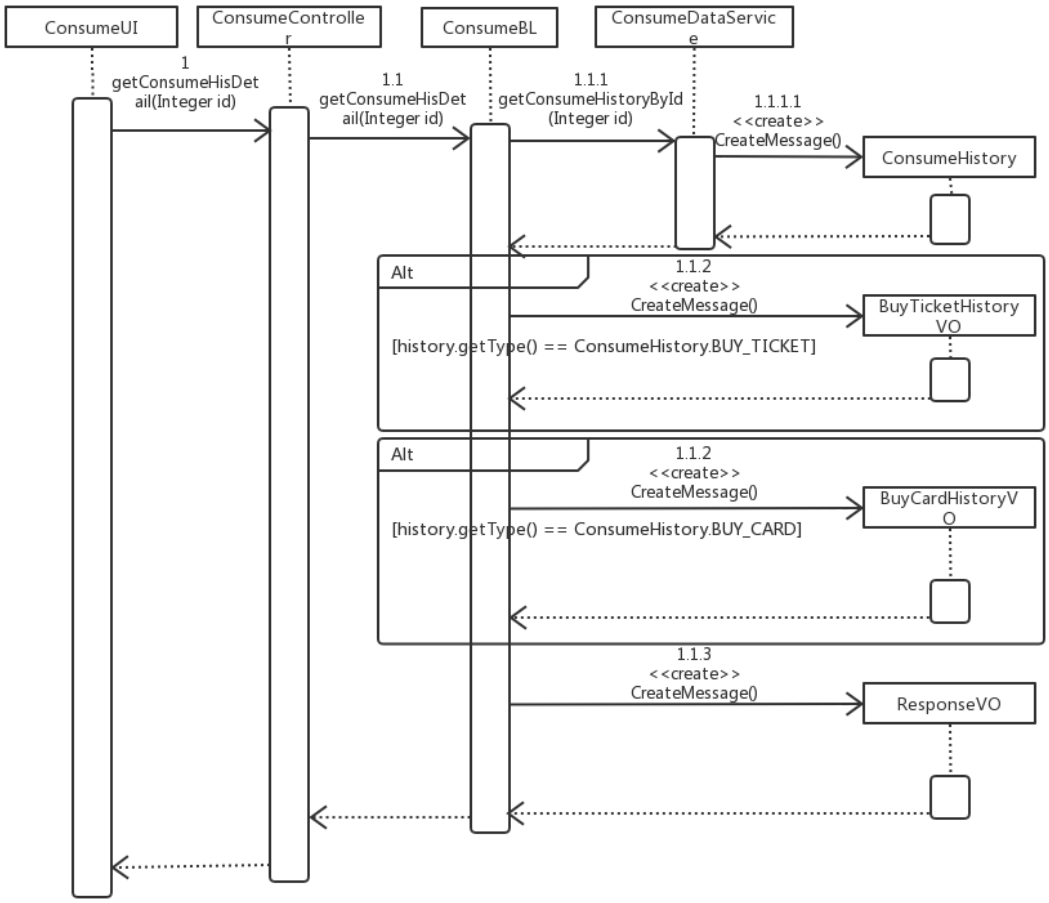
服务名	服务
ConsumeDateService.getTopUpHistoryByUserId(Integer userId)	根据userId获取此用户的充值记录列表，若无记录则返回空
ConsumeDateService.getConsumeHistoryByUserId(Integer userId)	根据userId获取此用户的消费记录列表，若无记录则返回空
ConsumeDateService.getConsumeHistoryById(Integer id)	根据id获取消费记录
ConsumeDateService.insertTopUpHistory(Integer userId, Double money,Double discount,Double balance,Timestamp time)	插入新的充值记录
ConsumeDateService.insertConsumeHistory(Integer userId, Double money,Double discount,String consumeType,Integer type,Integer contentId)	插入新的消费记录
ConsumeDateService.selectConsumeQulifiedUsers(Double totalConsume)	获得消费总额大于totalConsume的用户
HallServiceForBl.getHallById(int id)	获得指定影厅信息
MovieServiceForBl.getMovieById(int id)	获得指定电影信息
ScheduleServiceForBl.getScheduleItemById(int id)	获得指定排片信息
VipService.getCardById(int id)	获得指定会员卡信息
TicketServiceForBl.getTicketById(int id)	获得指定电影票信息

(4) 业务逻辑层的动态模型

图12~14是影院管理系统中ConsumeBL进行业务逻辑处理时的顺序图。







由于ConsumeBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.5 activitybl模块

(1) 模块概述

activitybl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。

activitybl模块的职责及接口参见软件体系结构文档。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加activitydataservice.ActivityDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的ActivityController，这样ActivityController会将操作优惠活动相关的逻辑处理委托给ActivityBL。

activitybl模块的设计如图15所示。

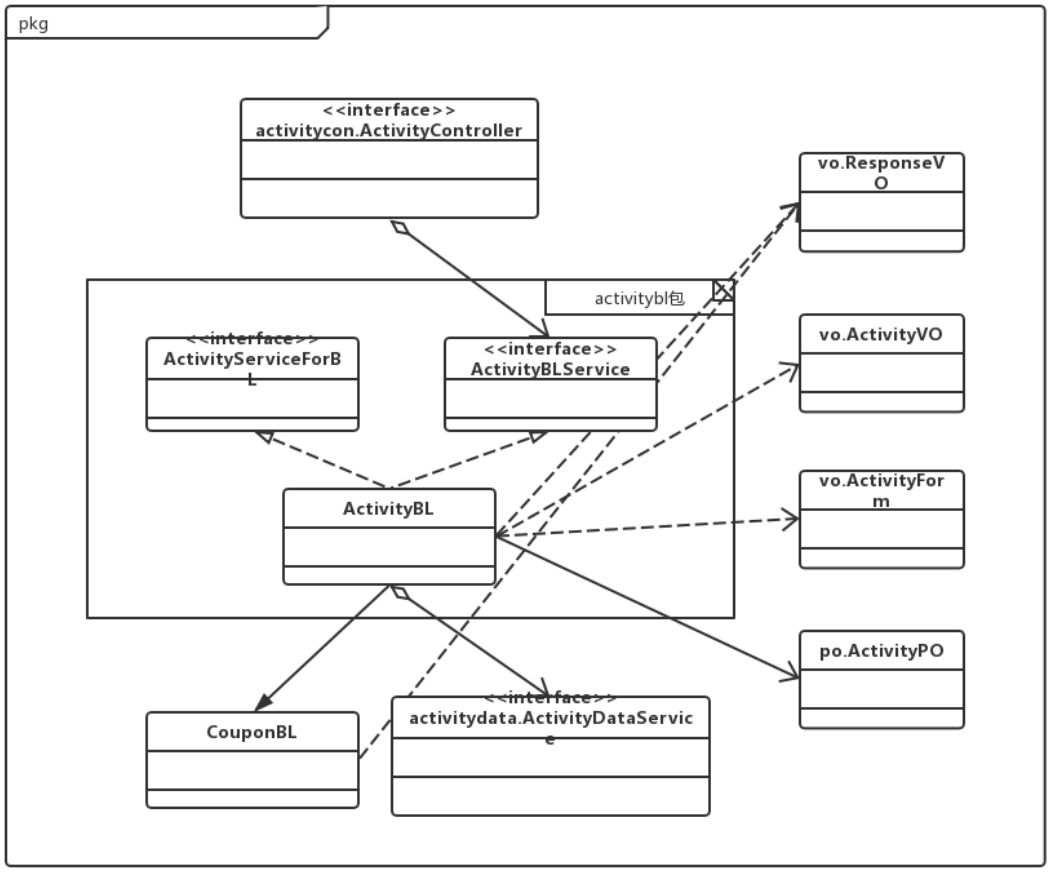


图15 activitybl模块各个类的设计

activitybl模块各个类的职责如表11所示。

表11 activitybl模块各个类的职责

模块	职责
ActivityBL	负责实现操作优惠活动的服务

(3) 模块内部类的接口规范

ActivityBL的接口规范如表12所示。

表12 ActivityBL的接口规范

提供的服务（供接口）

ActivityBL.publishActivity	**语法**	public ResponseVO publishActivity(ActivityForm activityForm)
	前置条件	activityForm格式正确
	后置条件	发布一个优惠活动
ActivityBL.getActivities	**语法**	public ResponseVO getActivities()
	前置条件	无
	后置条件	获取所有优惠活动信息
ActivityBL.deleteActivity	**语法**	public ResponseVO deleteActivity(int activityId)
	前置条件	activityId格式正确
	后置条件	删除优惠活动
ActivityBL.getActivityById	**语法**	public ResponseVO getActivityById(int activityId)
	前置条件	activityId格式正确
	后置条件	获取指定id的优惠活动

需要的服务（需接口）

服务名	**服务**
ActivityDataService.insertActivity(Activity activity)	在数据库中插入一个优惠活动
ActivityDataService.insertActivityAndMovie(int activityId,List<Integer> movieId)	在数据库中插入一条活动和电影的关系
ActivityDataService.selectActivities()	返回所有优惠活动信息
ActivityDataService.selectActivitiesByMovie(int movieId)	返回特定电影的优惠活动信息
ActivityDataService.selectById(int id)	获取指定id的优惠活动
ActivityDataService.deleteActivityById(int id)	删除指定的优惠活动
ActivityDataService.deleteActivityAndMovie(int id)	根据活动id删除活动和电影关系

(4) 业务逻辑层的动态模型

图16~19是影院管理系统中ActivityBL进行业务逻辑处理时的顺序图。

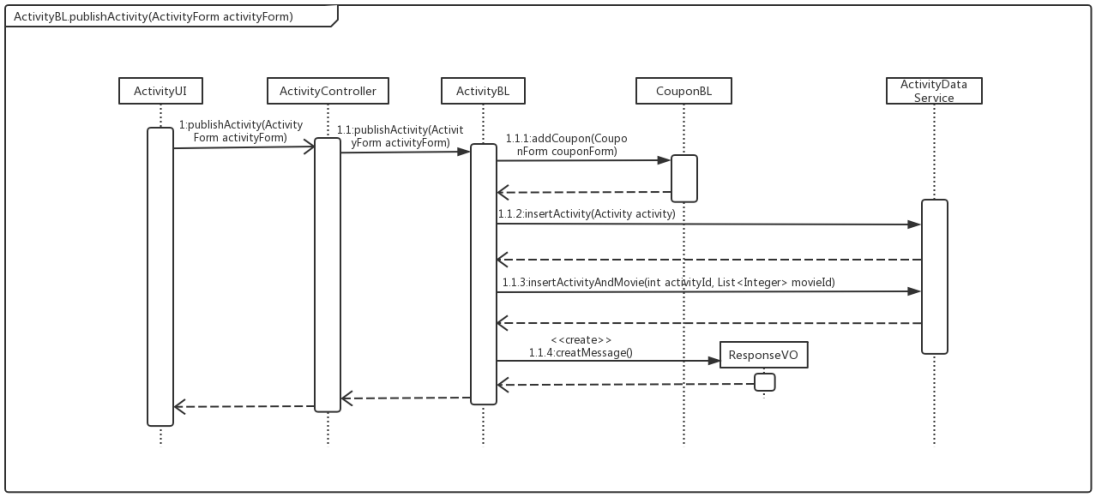


图16 发布优惠活动的顺序图

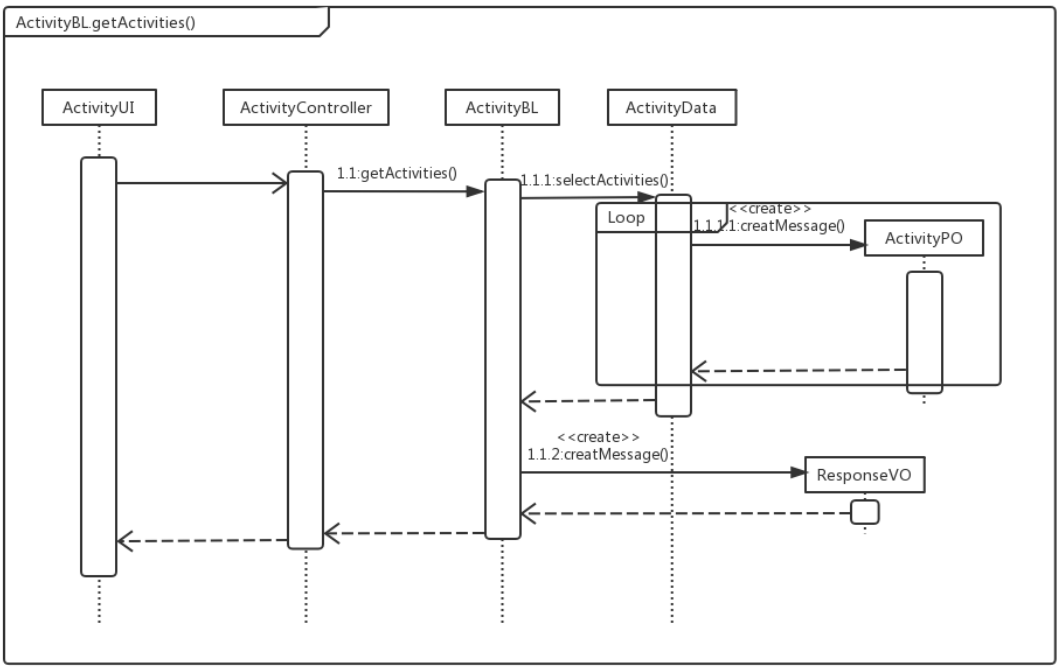


图17 获取所有优惠活动的顺序图

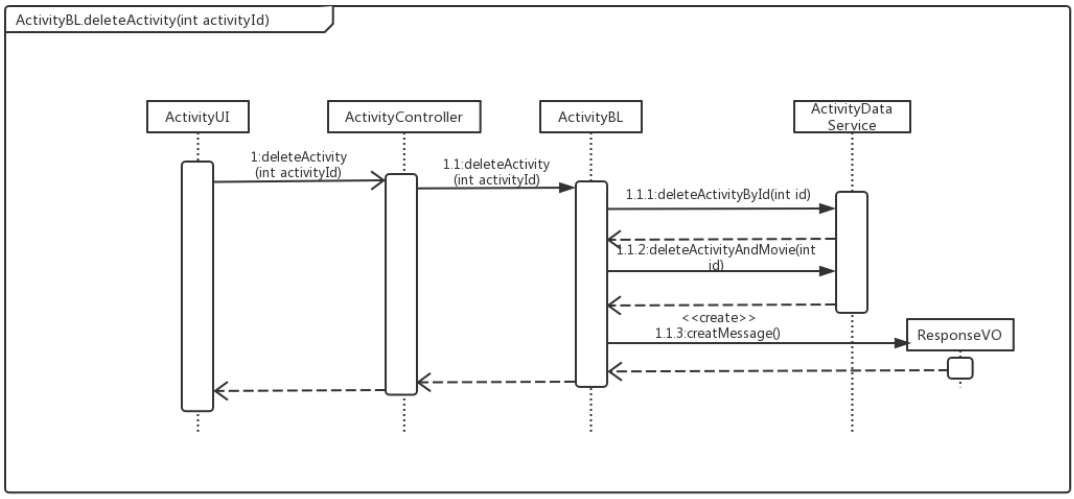


图18 删除优惠活动的顺序图

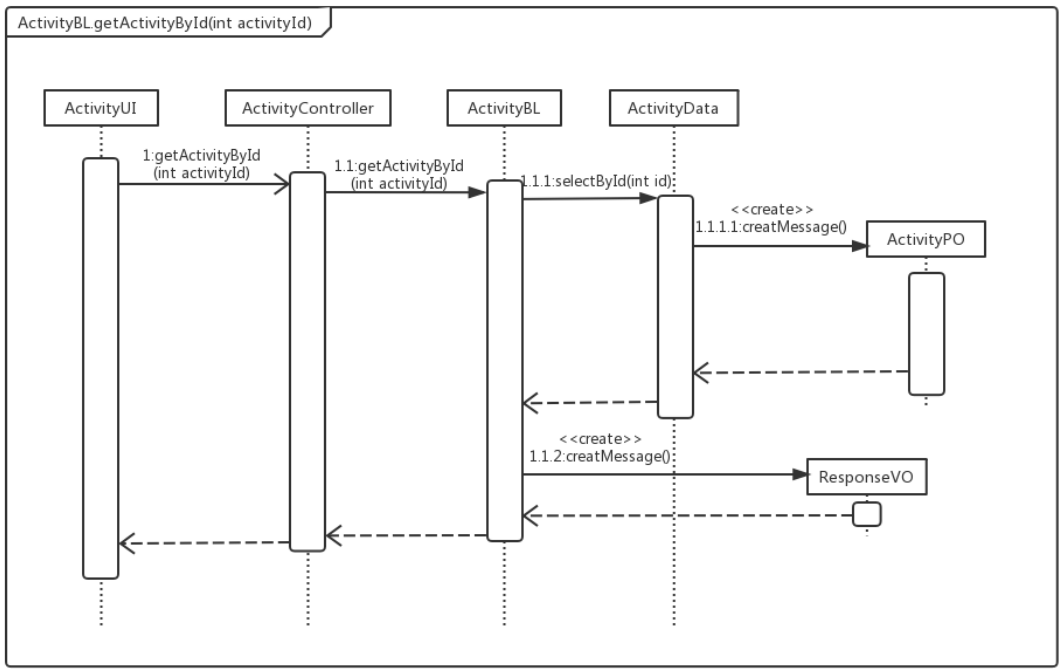


图19 获取优惠活动的顺序图

由于ActivityBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.6 cardbl模块

(1) 模块概述

cardbl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。

cardbl模块的职责及接口参见软件体系结构文档表x。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加carddataservice.CardDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的CardController，这样CardController会将操作优惠活动相关的逻辑处理委托给ActivityBL。

cardbl模块的设计如图20所示。

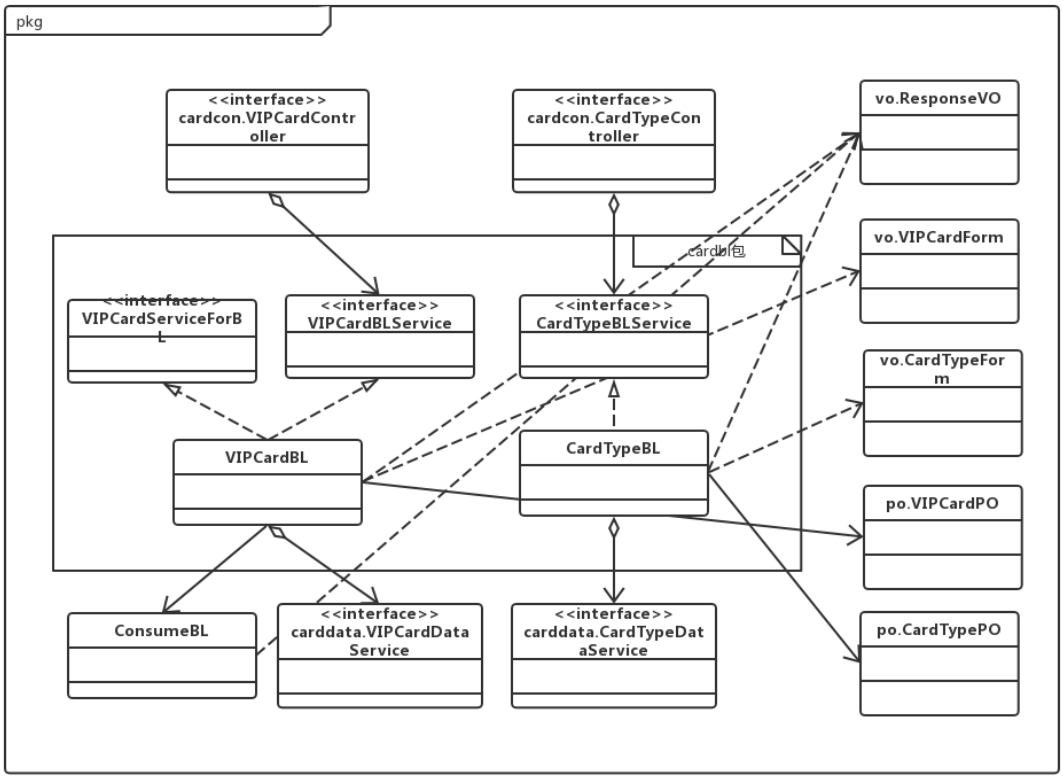


图20 cardbl模块各个类的设计

cardbl模块各个类的职责如表13所示。

表13 cardbl模块各个类的职责

模块	职责
CardTypeBL	负责实现操作会员卡类型的服务
VIPCardBL	负责实现操作用户会员卡的服务

(3) 模块内部类的接口规范

CardTypeBL和VIPCardBL的接口规范如表14~15所示。

表14 CardTypeBL的接口规范

提供的服务（供接口）

CardTypeBL.getCards	**语法**	public ResponseVO getCards()
	前置条件	无
	后置条件	获取所有会员卡类型
CardTypeBL.publishCard	**语法**	public ResponseVO publishCard(CardTypeForm cardTypeForm)
	前置条件	cardTypeForm信息格式正确
	后置条件	发布会员卡类型
CardTypeBL.deleteCard	**语法**	public ResponseVO deleteCard(int cardId)
	前置条件	cardId格式正确
	后置条件	删除会员卡类型
CardTypeBL.updateCard	**语法**	public ResponseVO updateCard(int cardId, CardTypeForm cardTypeForm)
	前置条件	cardId和cardTypeForm格式正确
	后置条件	更新会员卡类型信息

需要的服务（需接口）

服务名	**服务**
CardTypeDataService.selectAllCards()	获取所有会员卡类型
CardTypeDataService.insertOneCard(CardType cardType)	在数据库中插入会员卡类型
CardTypeDataService.deleteCardById(int cardId)	在数据库中删除会员卡类型，将会员卡类型状态设为0
CardTypeDataService.updateCardById(int cardId, CardType cardType)	更新会员卡类型信息

表15 VIPCardBL的接口规范

提供的服务（供接口）

VIPCardBL.addVIPCard	**语法**	public ResponseVO addVIPCard(int userId, int cardTypeId)
	前置条件	userId和cardTypeId格式正确
	后置条件	为指定用户增加会员卡
VIPCardBL.getCardByUserId	**语法**	public ResponseVO getCardByUserId(int userId)
	前置条件	userId格式正确
	后置条件	获取某用户的有效会员卡
VIPCardBL.charge	**语法**	public ResponseVO charge(VIPCardForm vipCardForm)
	前置条件	vipCardForm格式正确
	后置条件	为指定会员卡充值制定金额
VIPCardBL.changeVIPCard	**语法**	public ResponseVO changeVIPCard(int cardId, int cardTypeId)
	前置条件	cardId和carTypeId格式真确
	后置条件	更换指定用户的会员卡类型

需要的服务（需接口）

服务名	**服务**
VIPCardDataService.insertOneCard(VIPCard vipCard)	为指定用户增加会员卡
VIPCardDataService.selectCardByUserId(int userId)	获取某用户的有效会员卡
VIPCardDataService.updateCardBalance(int id, double balance)	为指定会员卡更新余额
VIPCardDataService.selectCardById(int id)	获取指定id的会员卡
VIPCardDataService.deleteCardById(int cardId)	从数据库中删除指定id的会员卡，将状态设置为0

(4) 业务逻辑层的动态模型

图21~24是影院管理系统中CardTypeBL进行业务逻辑处理时的顺序图。

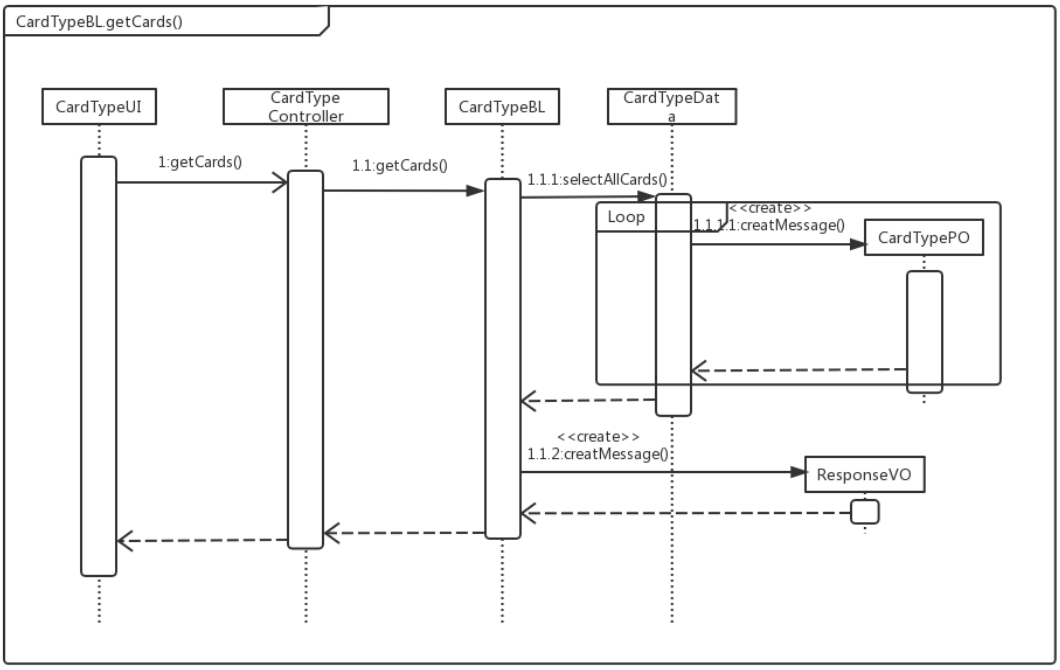


图21 获取所有会员卡类型的顺序图

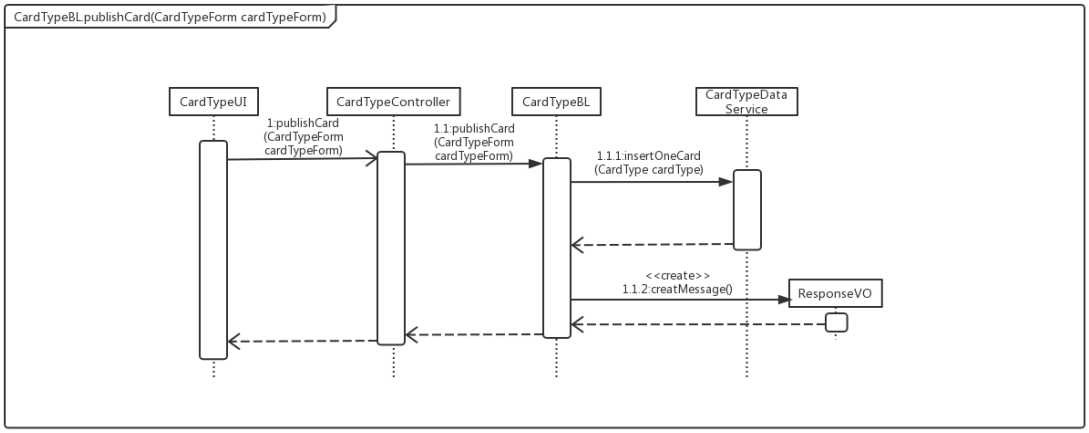


图22 发布会员卡类型的顺序图

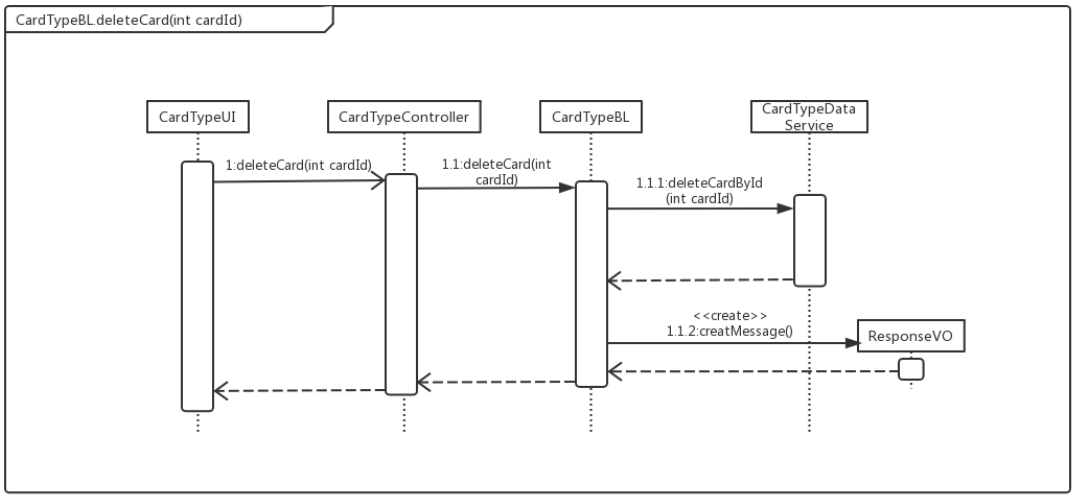


图23 删除会员卡类型的顺序图

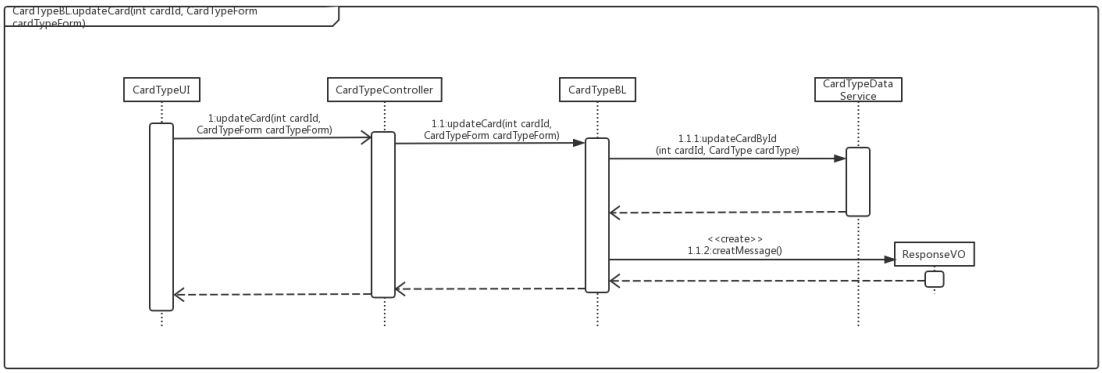


图24 更新会员卡类型的顺序图

由于CardTypeBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

图25~28是影院管理系统中VIPCardBL进行业务逻辑处理时的顺序图。

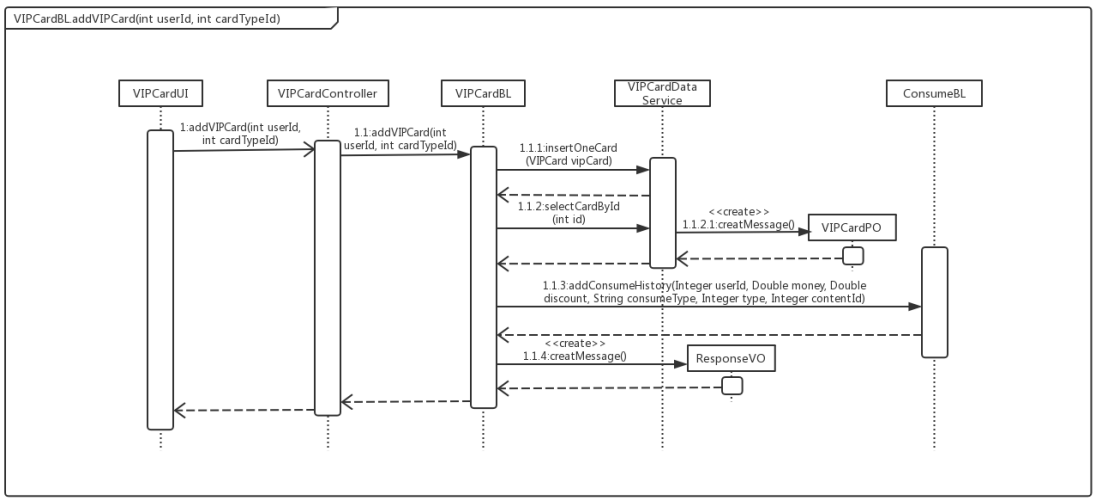


图25 为用户增加会员卡的顺序图

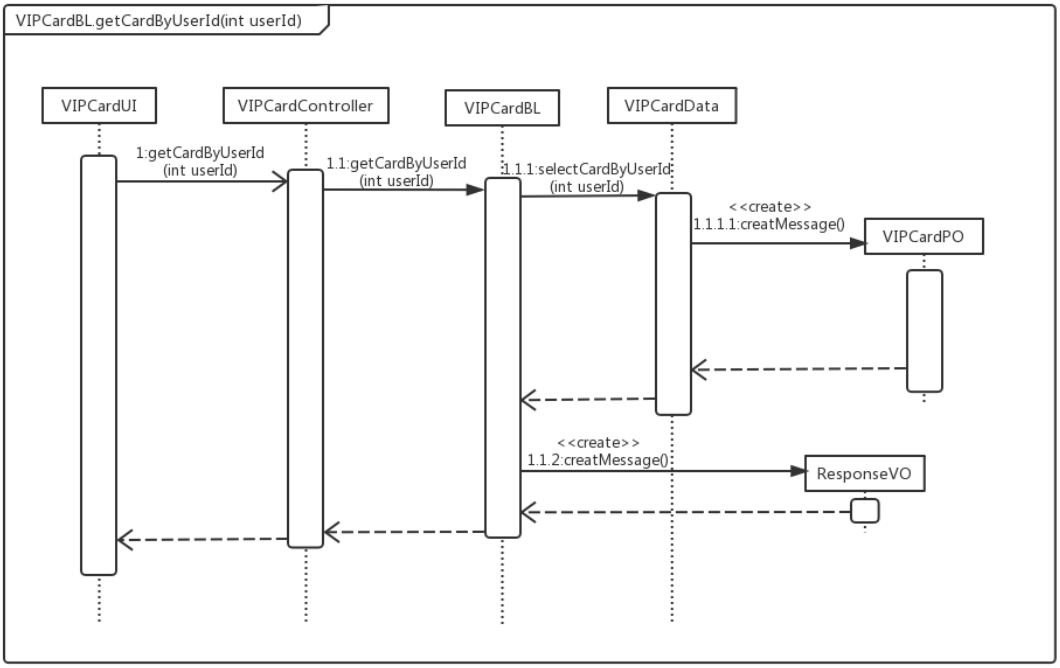


图26 获取用户会员卡的顺序图

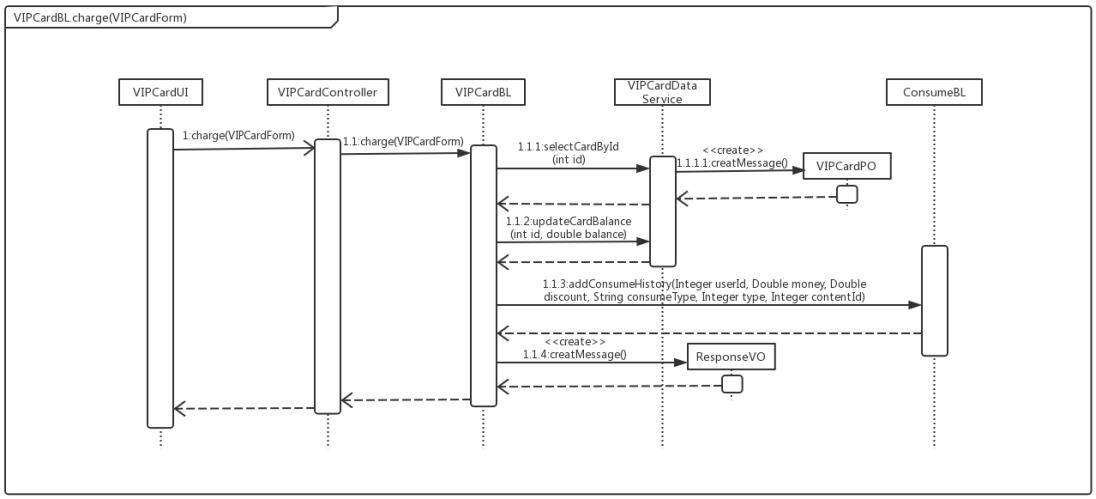


图27 充值会员卡的顺序图

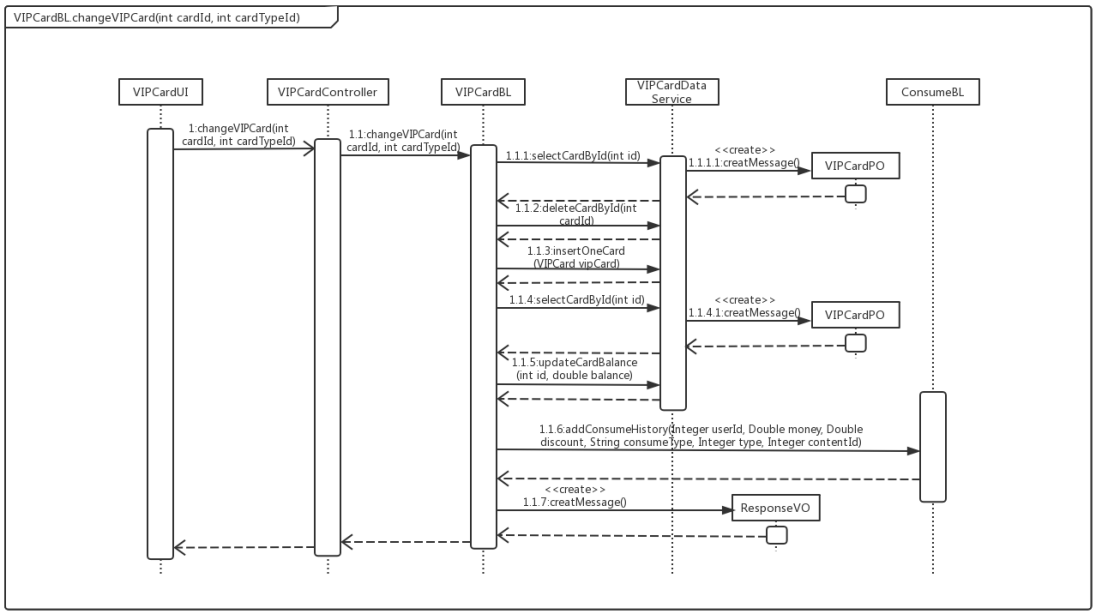


图28 更换会员卡的顺序图

由于VIPCardBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.7 couponbl模块

(1) 模块概述

couponbl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。

couponbl模块的职责及接口参见软件体系结构文档表x。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加coupondataservice.CouponDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的CouponController，这样CouponController会将操作优惠活动相关的逻辑处理委托给CouponBL。

couponbl模块的设计如图29所示。

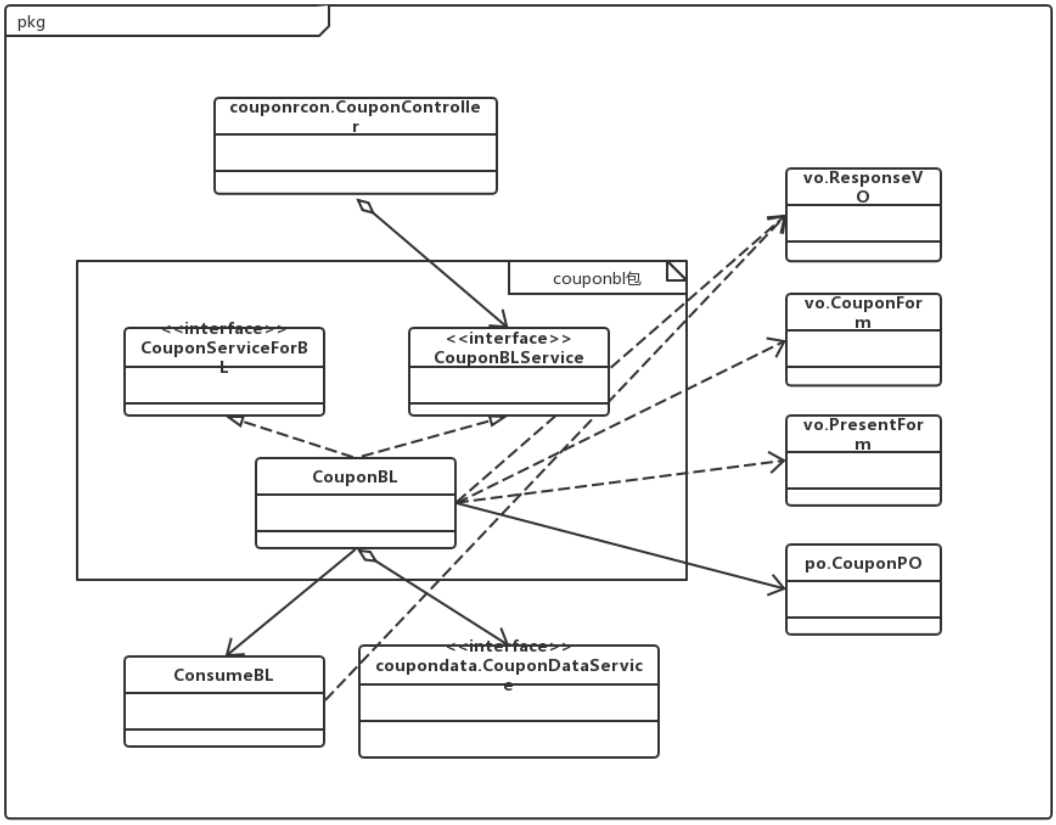


图29 couponbl模块各个类的设计

couponbl模块各个类的职责如表16所示。

表16 couponbl模块各个类的职责

模块	职责
CouponBL	负责实现操作优惠券的服务

(3) 模块内部类的接口规范

CouponBL的接口规范如表17所示。

表17 CouponBL的接口规范

提供的服务（供接口）		
CouponBL.getAllCoupons	**语法**	public ResponseVO getAllCoupons()
	前置条件	管理员准备赠送优惠券
	后置条件	获取所有优惠券
CouponBL.getUsersByConsume	**语法**	public ResponseVO getUsersByConsume(double totalConsume)
	前置条件	totalConsume符合输入规范
	后置条件	获取满足一定消费总额的用户
CouponBL.presentCoupon2User	**语法**	public ResponseVO presentCoupon2User(PresentForm presentForm)
	前置条件	presentForm符合输入规范
	后置条件	批量赠送优惠券
CouponBL.getCouponsByUser	**语法**	public ResponseVO getCouponsByUser(int userId)
	前置条件	userId存在
	后置条件	返回用户拥有的有效优惠券
CouponBL.addCoupon	**语法**	public ResponseVO addCoupon(CouponForm couponForm)
	前置条件	couponForm符合输入规范
	后置条件	发布新优惠券

需要的服务（需接口）

服务名	服务
ConsumeBL.getConsumeQualifiedUsers(double totalConsume)	获取满足一定消费总额的用户
CouponDataService.selectAllCoupons()	获取所有有效优惠券
CouponDataService.insertCouponUser(int couponId,int userId)	赠送某种优惠券给某个用户
CouponDataService.insertCoupon(Coupon coupon)	在数据库中插入优惠券
CouponDataService.selectCouponByUser(int userId)	获取用户拥有的所有有效优惠券

(4) 业务逻辑层的动态模型

图30~34是影院管理系统中CouponBL进行业务逻辑处理时的顺序图。

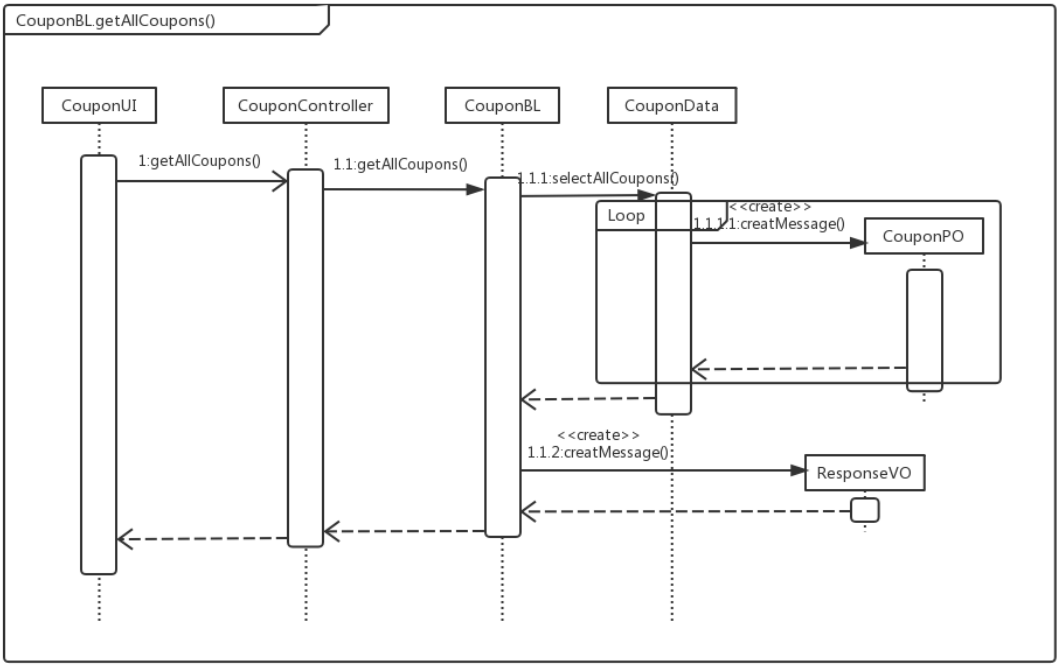


图30 获取所有优惠券的顺序图

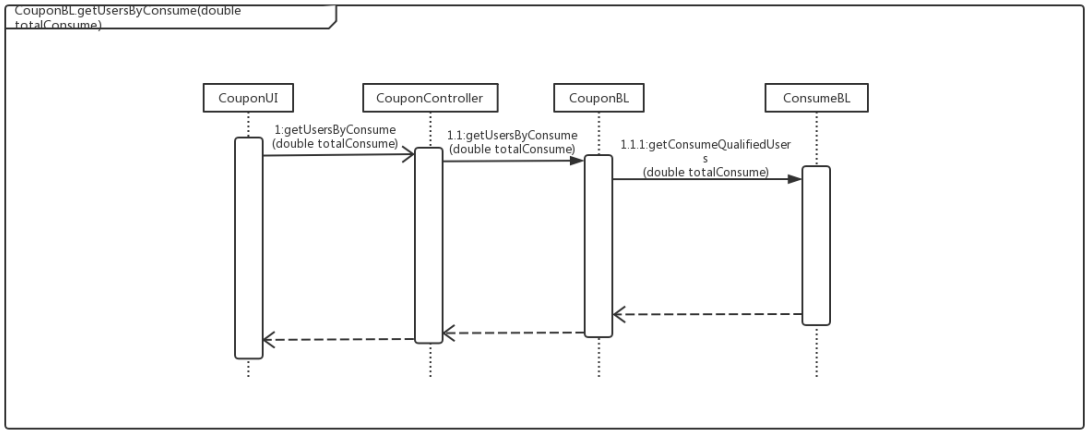


图31 获取消费满一定金额的用户的顺序图

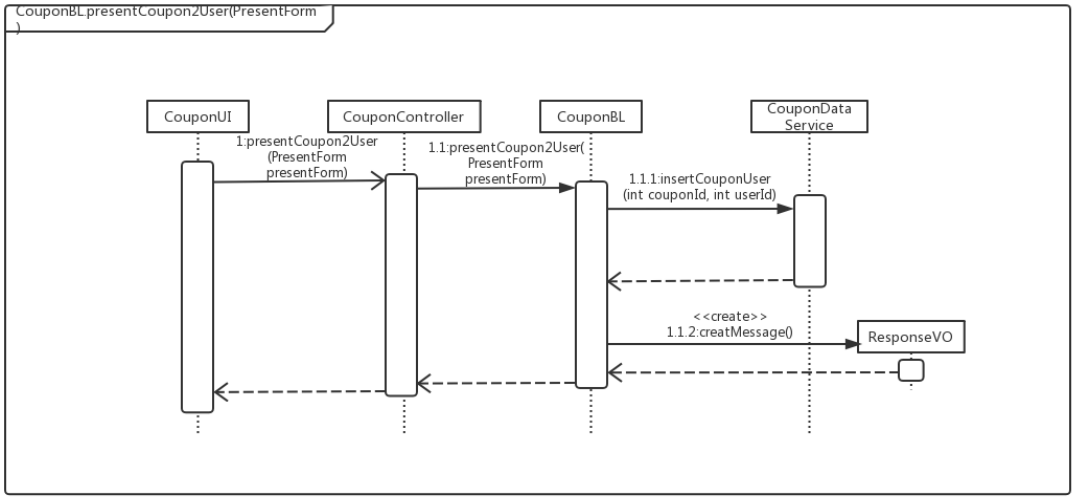


图32 赠送优惠券给用户的顺序图

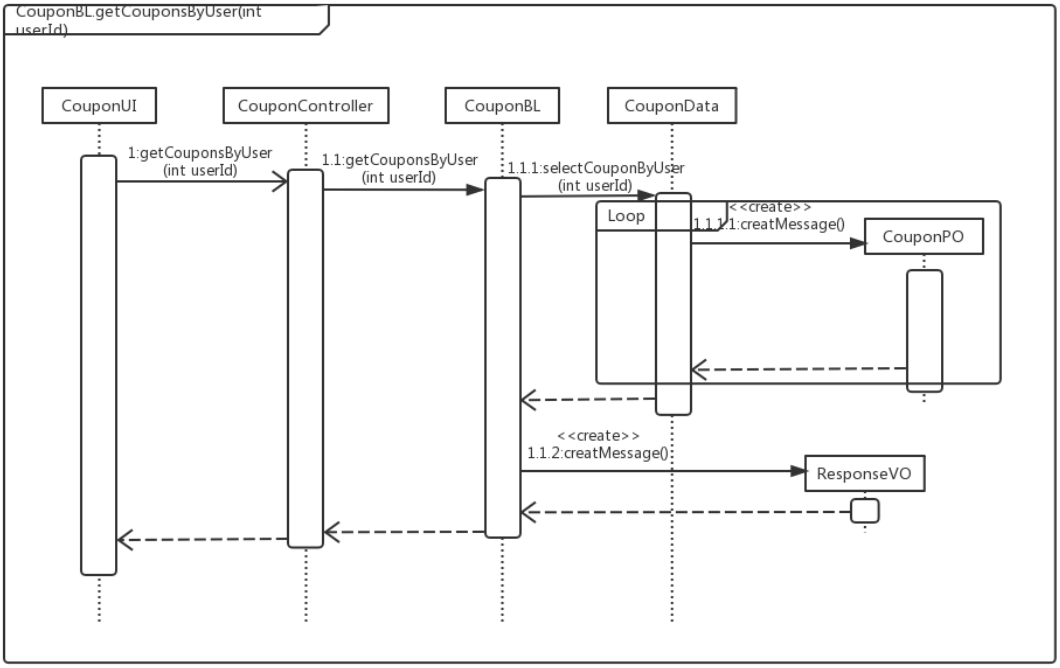


图33 获取用户优惠券的顺序图

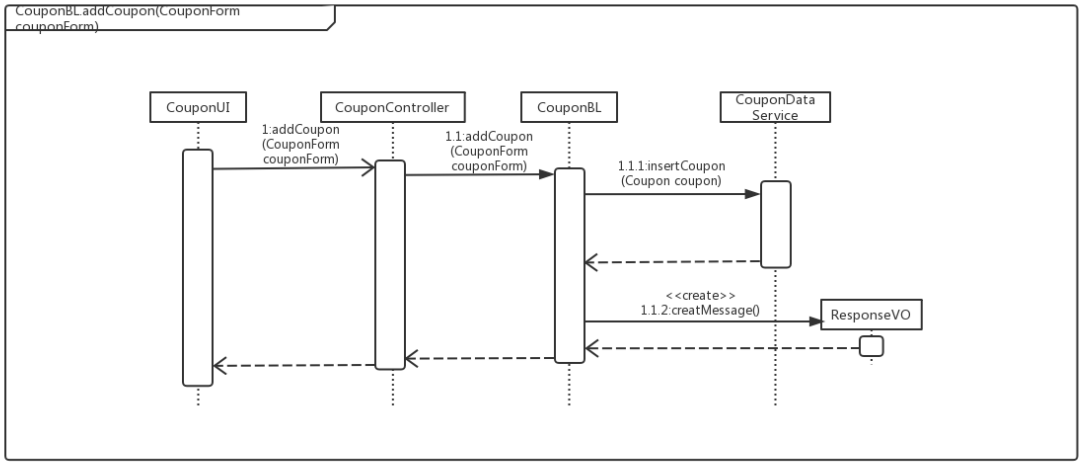


图34 添加优惠券的顺序图

由于CouponBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.8 hallbl模块

(1) 模块概述

hallbl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。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加halldataservice.HallDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的HallController，这样HallController会将操作影厅信息相关的逻辑处理委托给HallBL。

hallbl模块的设计如图35所示。

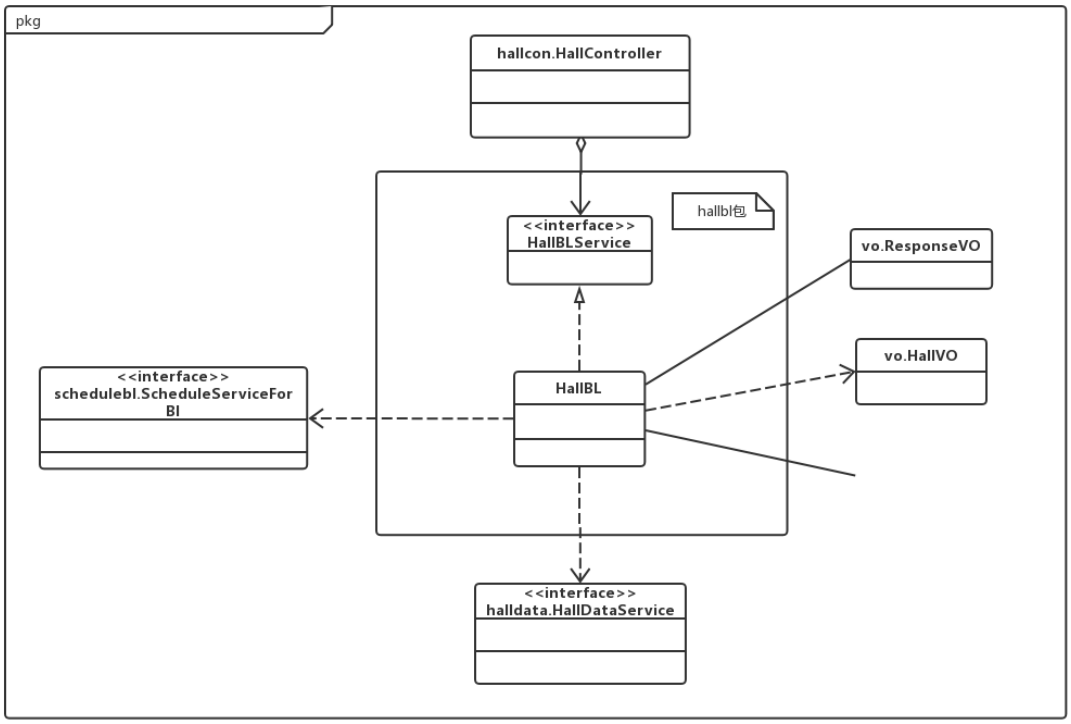


图35 hallbl模块各个类的职责

模块	职责
HallBL	负责实现影厅信息增删改查的服务

(3) 模块内部类的接口规范

HallBL的接口规范如表18所示。

表18 hallbl模块的接口规范

提供的服务（供接口）		
HallBLService.searchAllHall	**语法**	public ResponseVO searchAllHall()
	前置条件	无
	后置条件	返回所有影厅信息列表
HallBLService.addHall	**语法**	public ResponseVO addHall(HallVO hallVO)
	前置条件	hallVO不为空且符合输入规则
	后置条件	根据hallId查询是否存在对应的影厅信息记录，如果有则将对应的影厅信息删除，最后返回删除影厅信息的结果
HallBLService.updateHall	**语法**	public ResponseVO updateHall(HallVO hallVO)
	前置条件	hallVO不为空且符合输入规则
	后置条件	根据hallVO查询是否已经存在相同的hallName，如果没有则将原来对应的影厅信息更新，最后返回更新影厅信息的结果
HallBLService.removeHall	**语法**	public ResponseVO removeHall(int hallId)
	前置条件	无
	后置条件	根据hallId查询是否存在对应的影厅信息记录，如果有则将对应的影厅信息删除，最后返回删除影厅信息的结果

HallBLService.getAvailableHalls	**语法**	public ResponseVO getAvailableHalls()
	前置条件	无
	后置条件	获取所有空闲（没有排片）的影厅信息列表

需要的服务（需接口）

服务名	服务
HallDataService.selectAllHall	获取所有影厅信息列表
HallDataService.checkHallName	查询是否存在相同名字的影厅记录
HallDataService.addHall	添加影厅信息
HallDataService.updateHall	更新影厅信息
HallDataService.removeHallById	根据影厅id删除影厅信息
HallDataService.getHallsExcept	根据影厅id列表，获取影厅id不属于该id列表的影厅信息列表
HallDataService.selectHallById	根据影厅id获取影厅信息
ScheduleService.getScheduledHalls	获取当前已经有排片的影厅id列表

(4) 业务逻辑层的动态模型

图36~40是影院管理系统中HallBL进行业务逻辑处理时的顺序图。

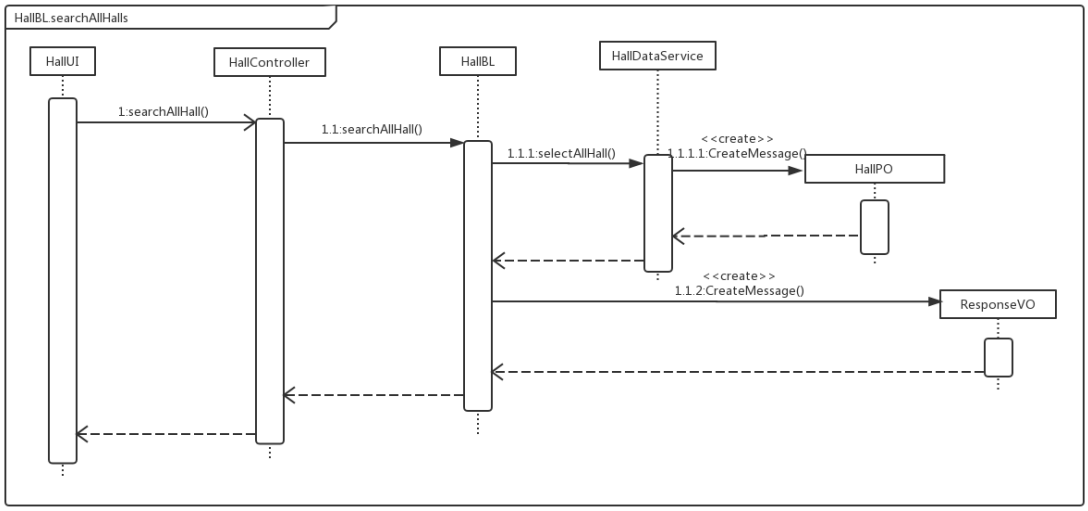


图36 获取所有影厅信息的顺序图

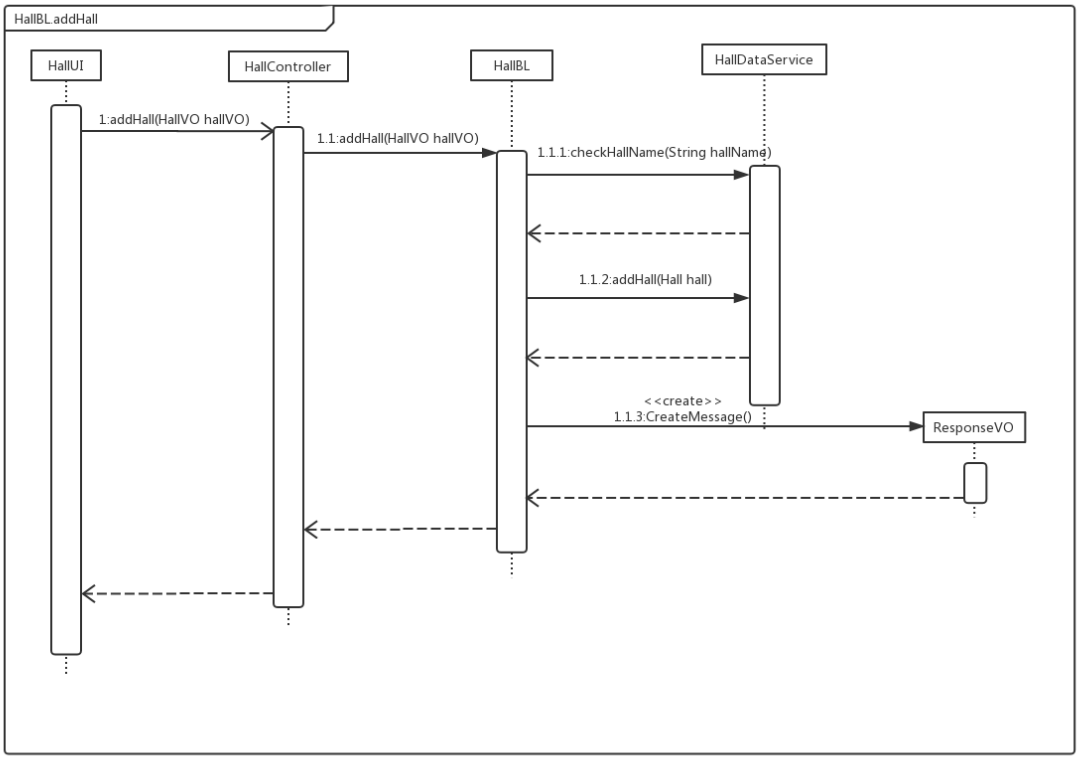


图37 添加影厅信息的顺序图

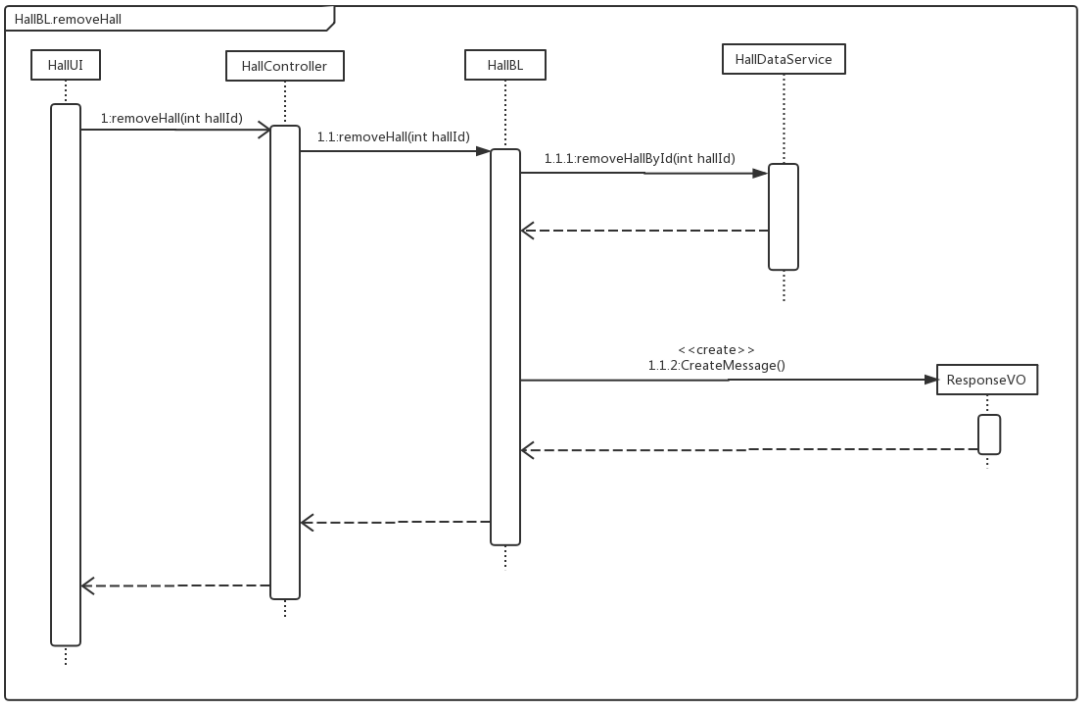


图38 删除影厅信息的顺序图

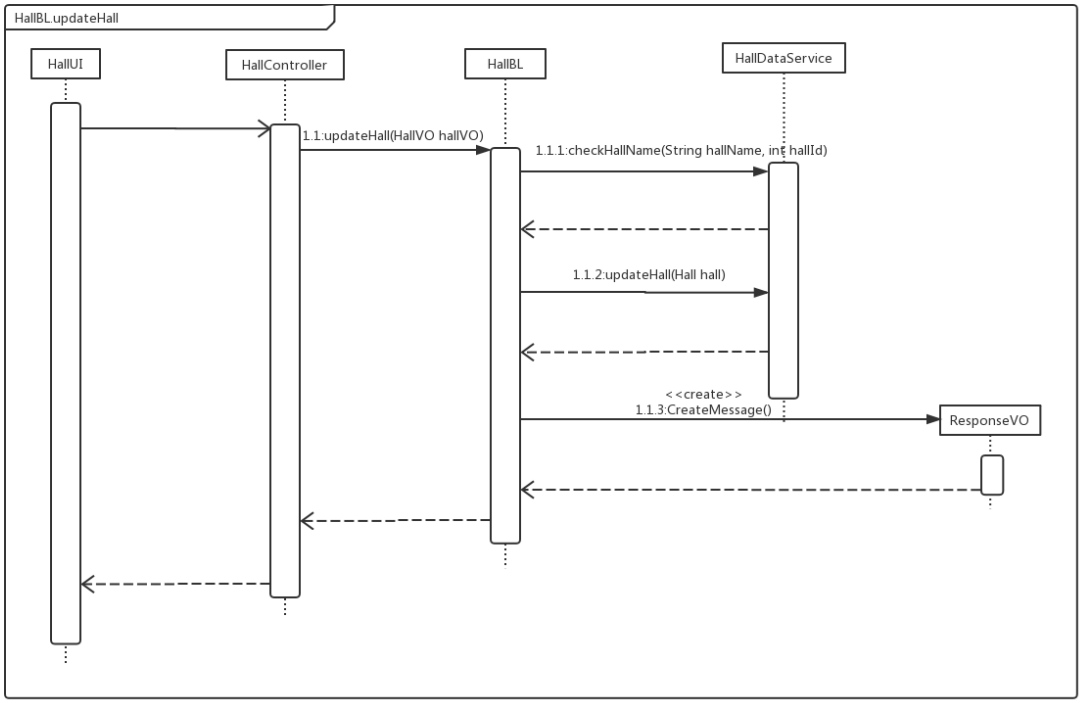


图39 更新影厅信息的顺序图

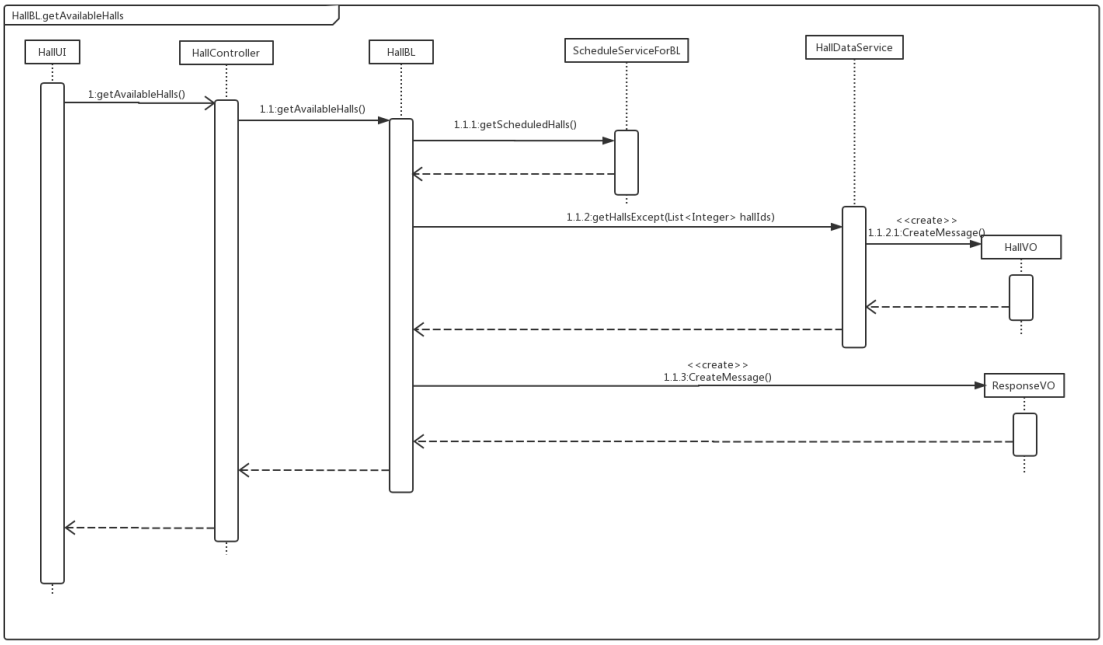


图40 获取空闲的影厅信息的顺序图

由于HallBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

4.1.9 schedulebl模块

(1) 模块概述

schedulebl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。

(2) 整体结构

根据体系结构的设计，我们将系统分为展示层、控制层、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。比如业务逻辑层和数据层之间，我们添加scheduledataservice.ScheduleDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了控制层的ScheduleController，这样ScheduleController会将操作影厅信息相关的逻辑处理委托给ScheduleBL。为了方便业务逻辑层其他模块使用排片相关的方法，我们增加了ScheduleServiceForBl，避免层间其他模块直接调用scheduledataservice.ScheduleDataService接口，减少层间的耦合，提高层内内聚

schedulebl模块的设计如图41所示。

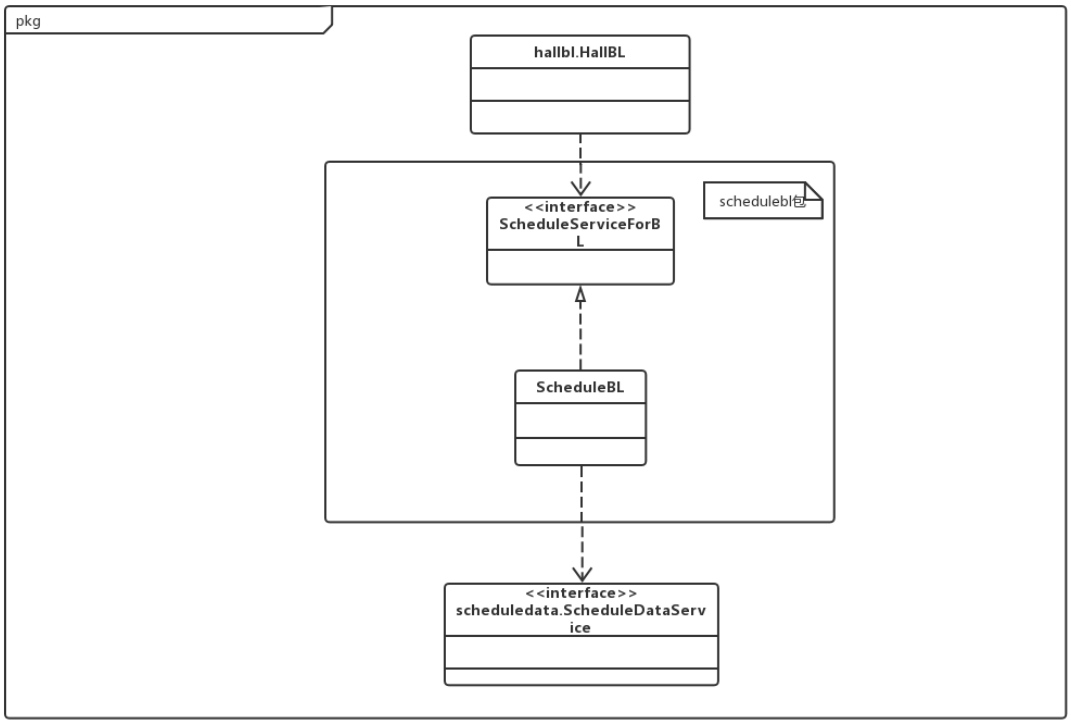


图41 schedulebl模块各个类的职责

模块	职责
ScheduleBL	负责实现排片的服务

(3) 模块内部类的接口规范

schedulebl的接口规范如表19所示。

表19 schedulebl模块的接口规范

提供的服务（供接口）

ScheduleServiceForBl.getScheduledHalls	**语法**	public List<Integer> getScheduledHalls()
	前置条件	无
	后置条件	返回所有已排片的影厅信息列表

需要的服务（需接口）

服务名	**服务**
ScheduleDataService.getHallsInSchedules	获取数据库中当前没有排片（当前时间晚于电影结束时间）的所有影厅id列表

(4) 业务逻辑层的动态模型

图42是影院管理系统中ScheduleBL进行业务逻辑处理时的顺序图。

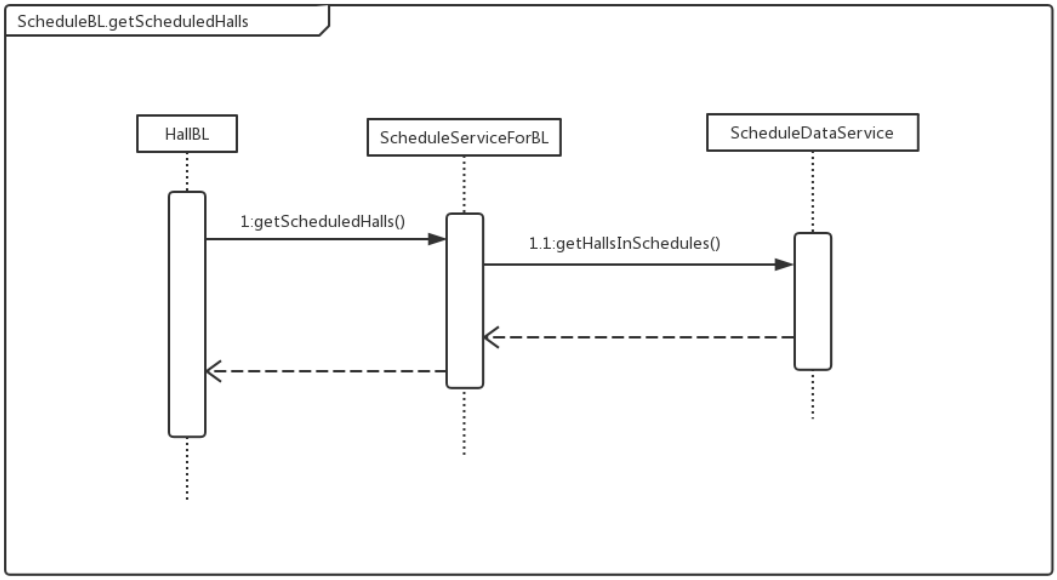


图42 获取空闲的影厅信息的顺序图

由于ScheduleBL只作为工具进行业务逻辑处理，不持有成员变量，因而在处理过程中是无状态的。

(5) 业务逻辑层的设计原理

利用SpringBoot的依赖注入，通过数据层接口获得po后，进行业务逻辑处理，将需要在页面上显示的信息封装成vo，最后封装成ResponseVO返回给上层。

5. 依赖视角

图43和图44是第三阶段客户端和服务器各自的包之间的依赖关系。

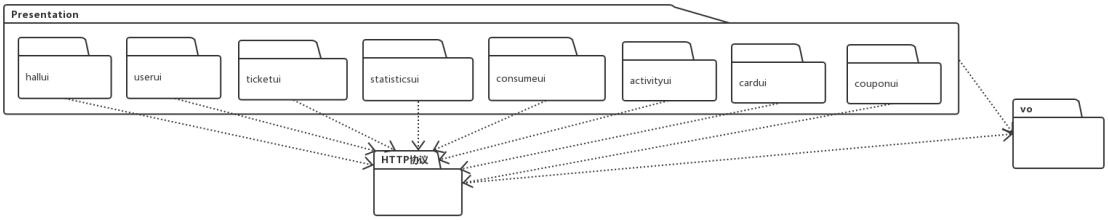


图43 客户端包图

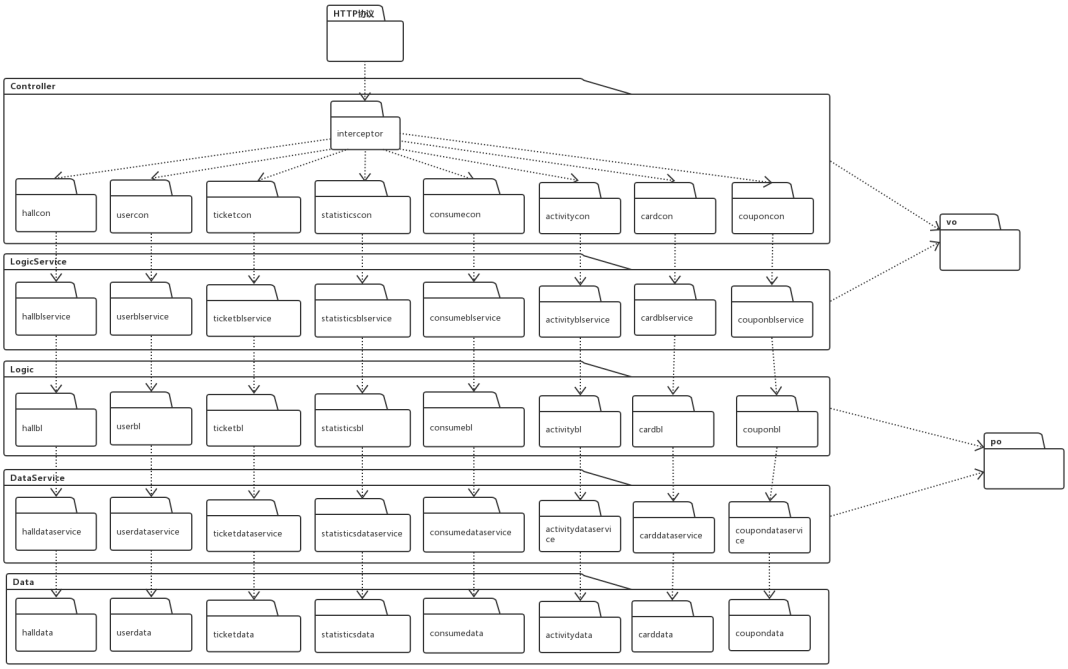


图44 服务器端包图