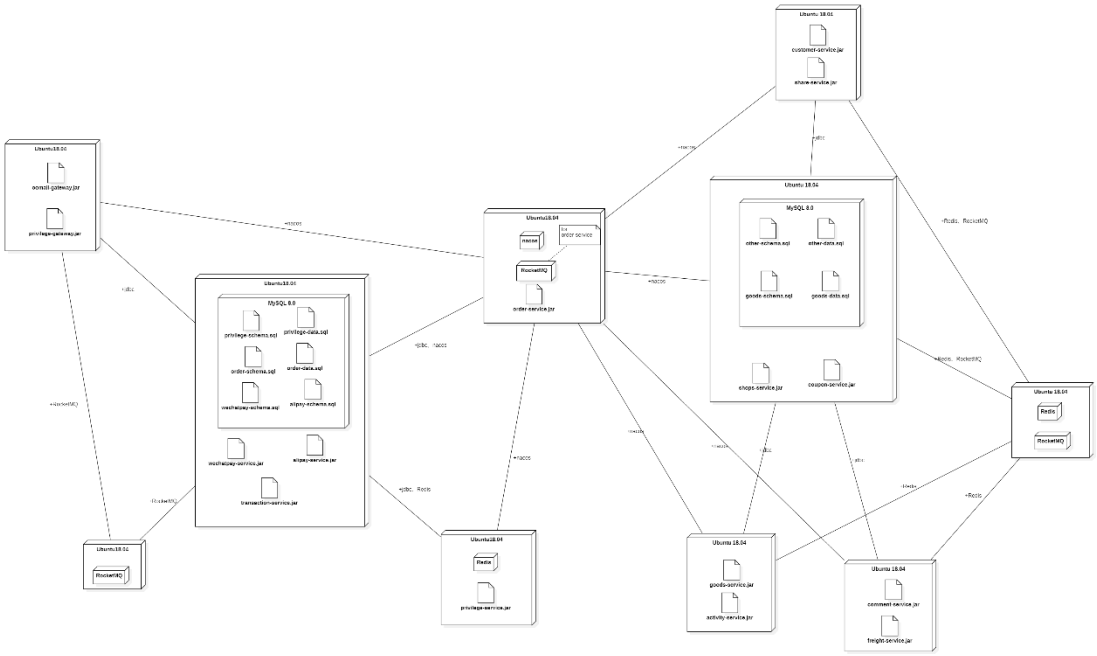
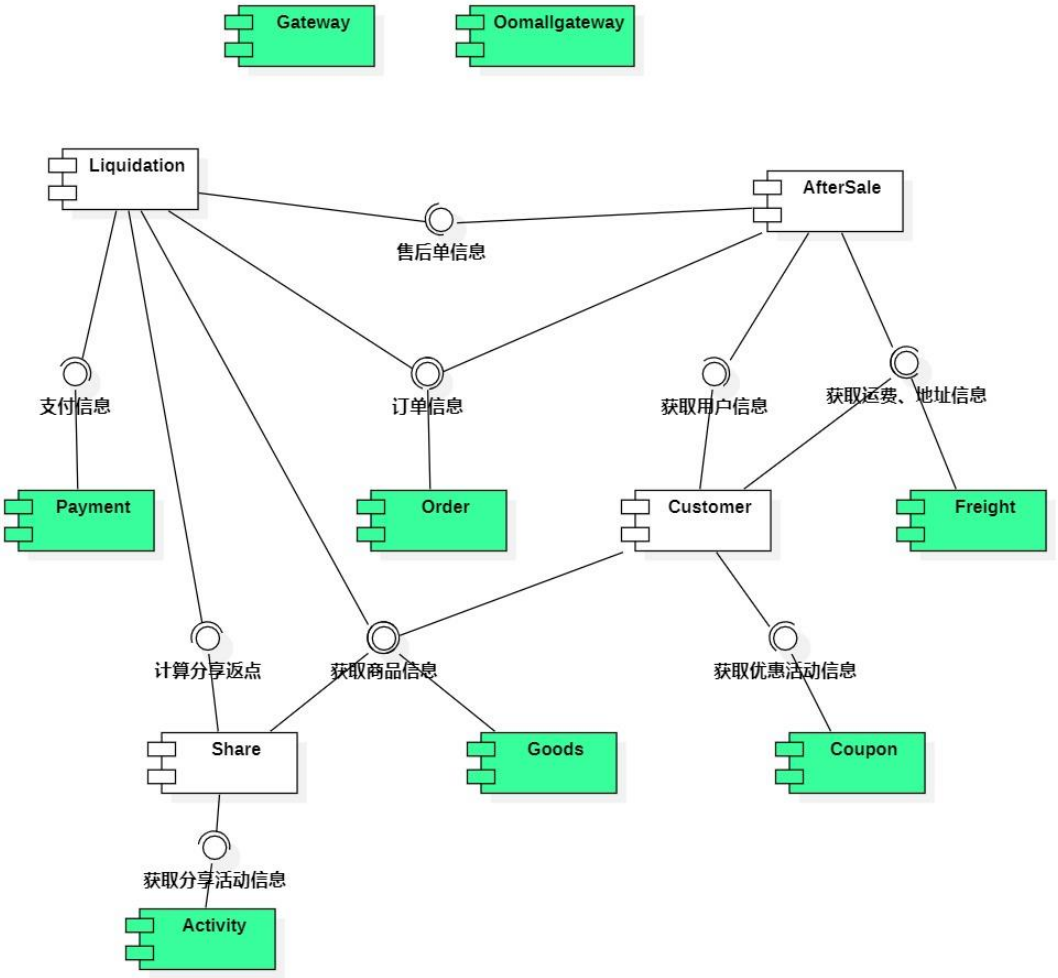
1-7 详细设计

1. 部署图

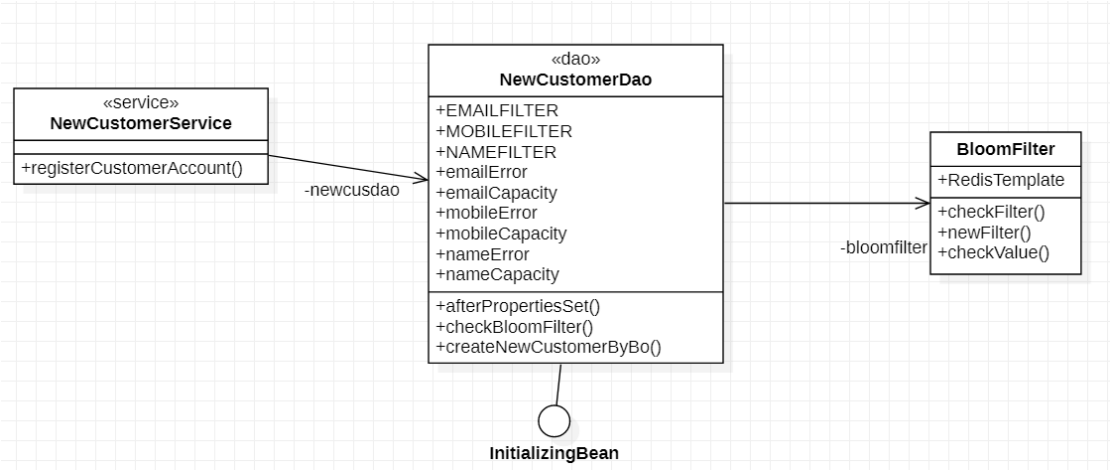


2. 组件图

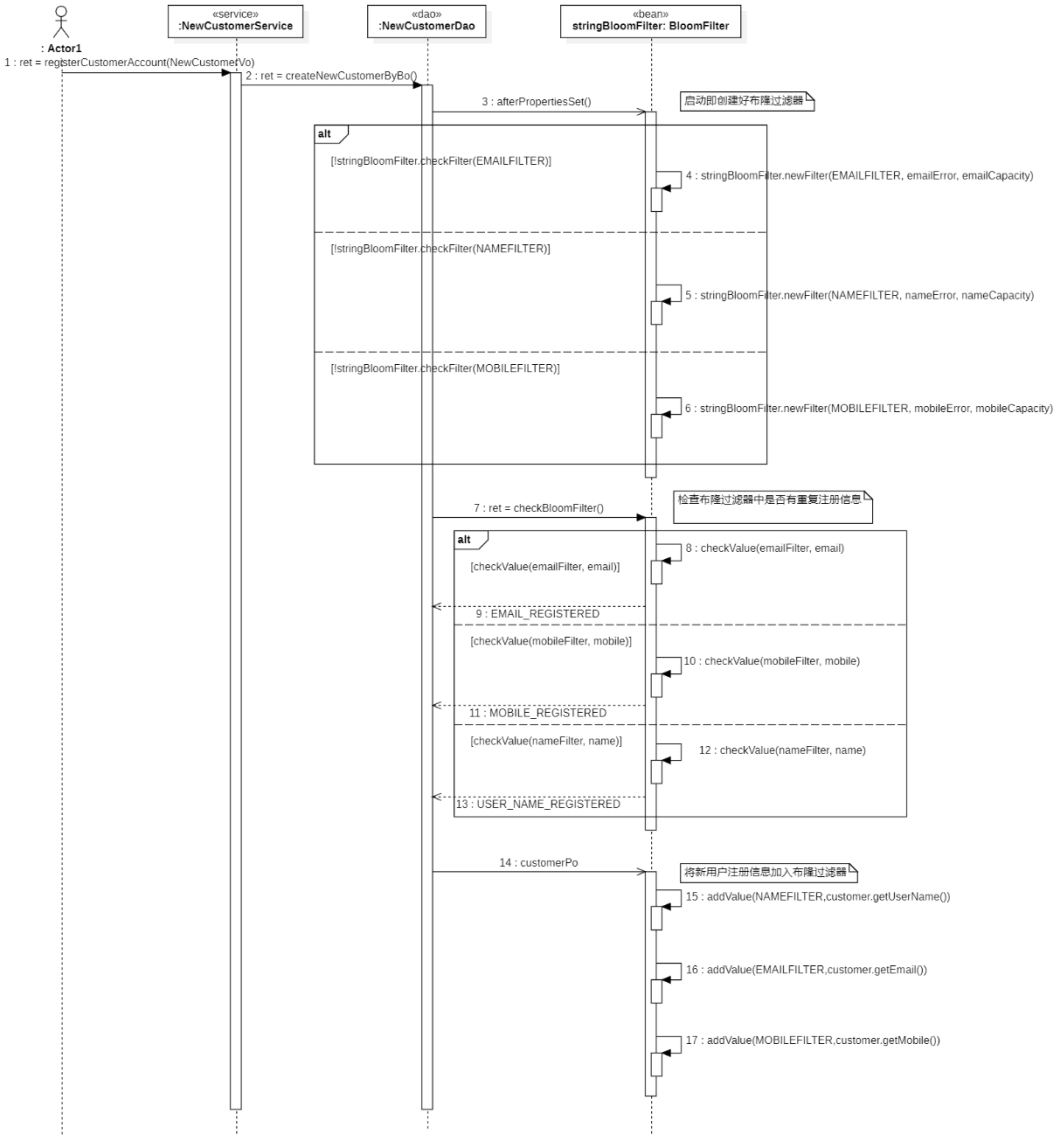


3. 用户模块详细设计

1. 类图



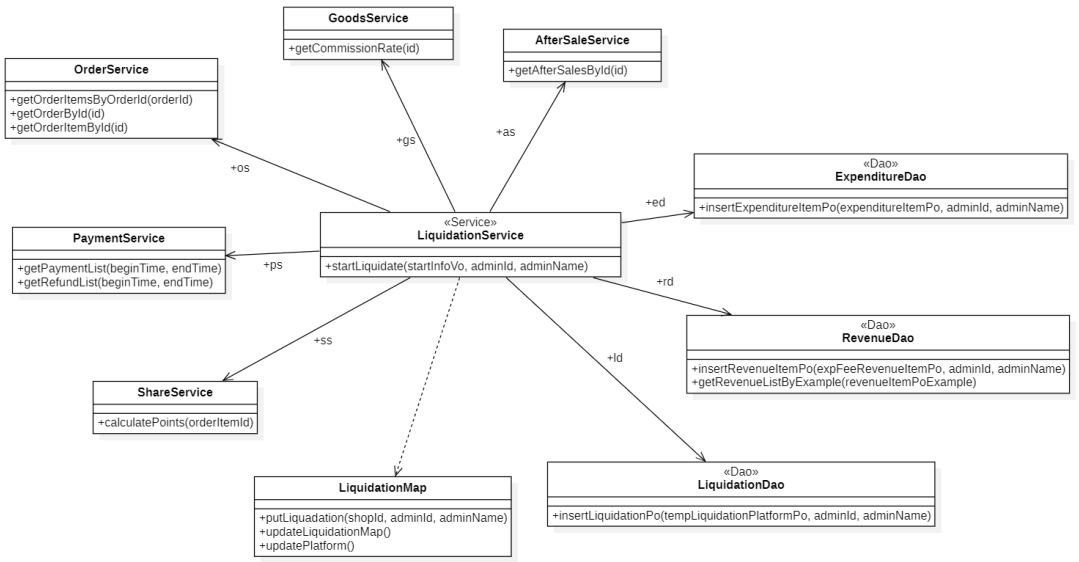
2. 时序图



用户注册用到的类不多，主要逻辑在于布隆过滤器。在系统启动时，afterPropertiesSet 方法会在 redis 中建立三个过滤器，用于检测用户用户名、邮箱、电话号码是否重复。用户发来的请求中包含了注册信息：用户名、邮箱、电话号码、真实姓名。信息到达 dao 层时，首先会由布隆过滤器去检查是否存在重复信息，如果有重复信息，则判定为注册失败，直接将对应注册失败信息返回。如果不重复，则将新的用户信息加入到布隆过滤器，以方便后续检验。之后，将新用户信息插入到数据库。

4. 清算模块详细设计

1. 类图



2. 时序图

图 1:

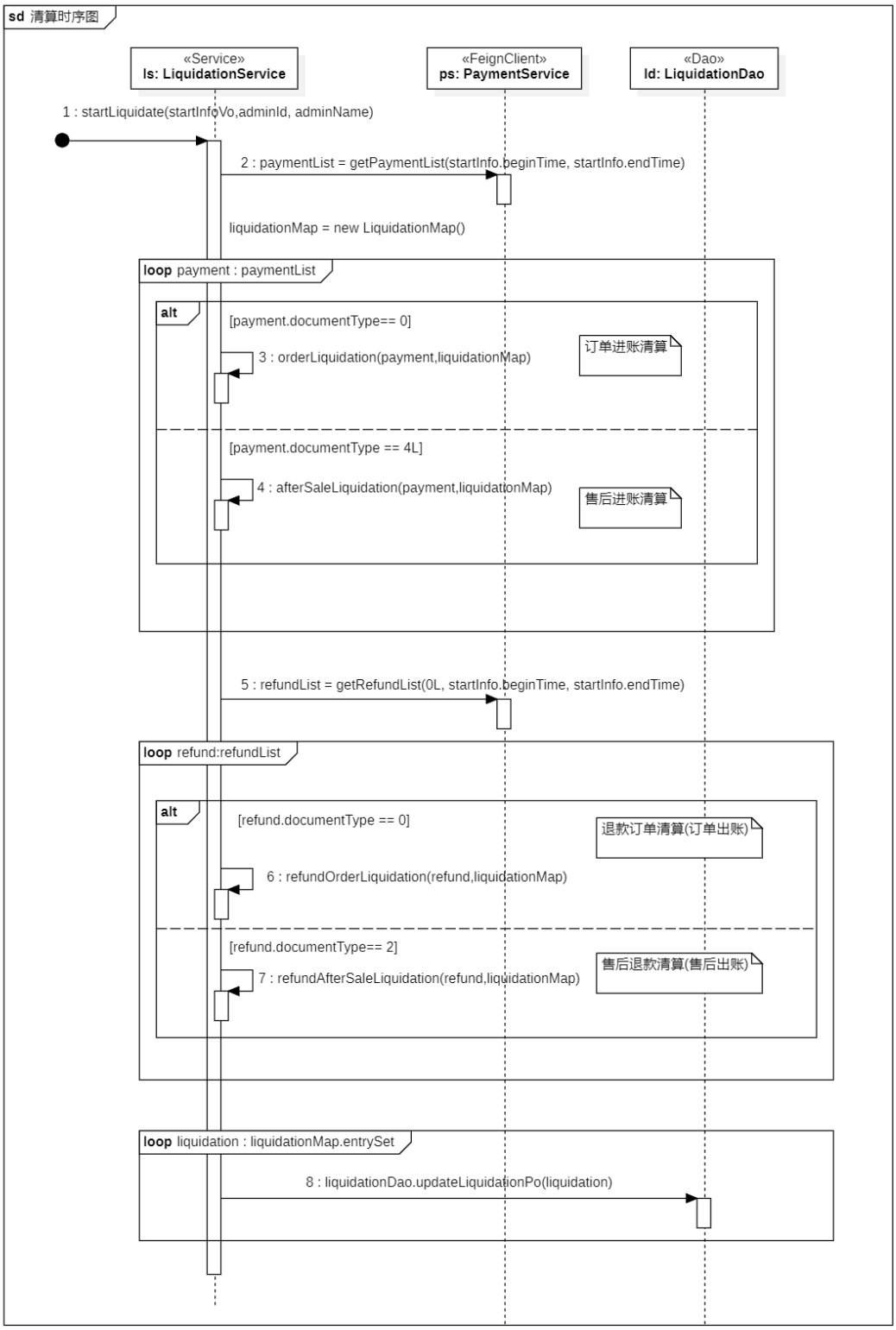
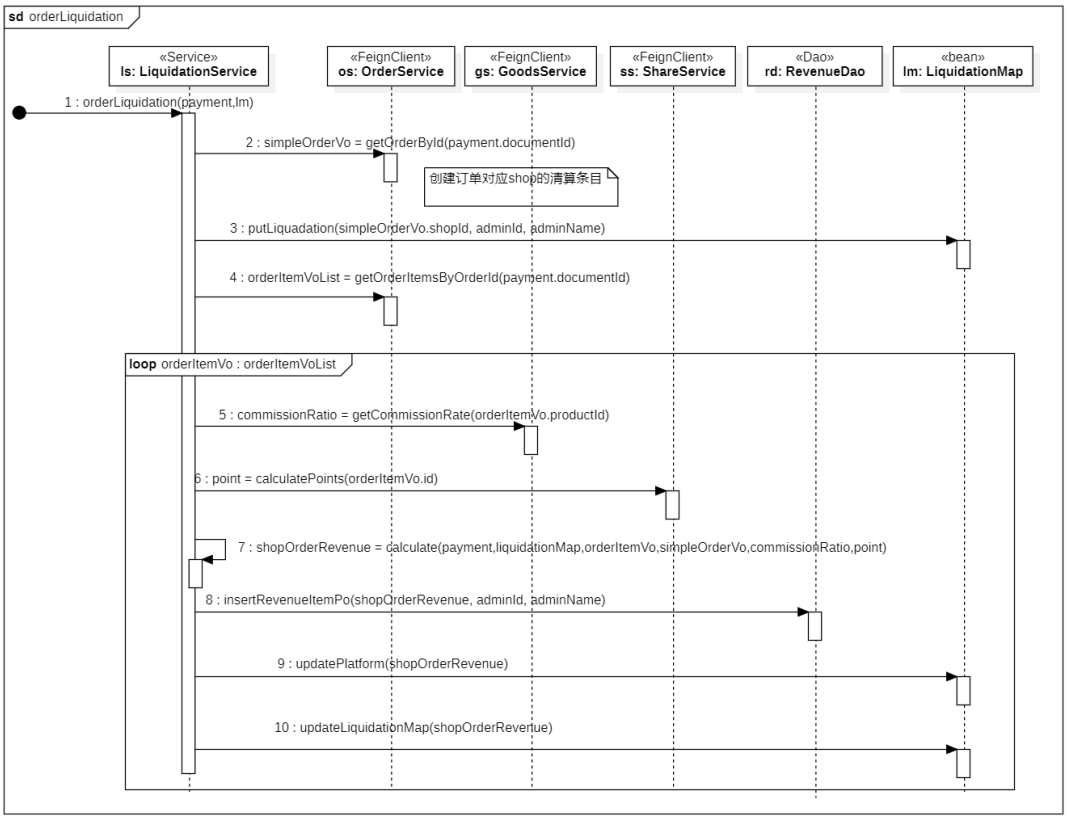

sd 清算时序图
«Service»
ls: LiquidationService
«FeignClient»
ps: PaymentService
«Dao»
ld: LiquidationDao
1 : startLiquidate(startInfoVo,adminId, adminName)
2 : paymentList = getPaymentList(startInfo.beginTime, startInfo.endTime)
liquidationMap = new LiquidationMap()
loop payment : paymentList
alt
[payment.documentType== 0]
3 : orderLiquidation(payment,liquidationMap)
订单进账清算
[payment.documentType == 4L]
4 : afterSaleLiquidation(payment,liquidationMap)
售后进账清算
5 : refundList = getRefundList(0L, startInfo.beginTime, startInfo.endTime)
loop refund:refundList
alt
[refund.documentType == 0]
退款订单清算(订单出账)
6 : refundOrderLiquidation(refund,liquidationMap)
[refund.documentType== 2]
7 : refundAfterSaleLiquidation(refund,liquidationMap)
售后退款清算(售后出账)
loop liquidation : liquidationMap.entrySet
8 : liquidationDao.updateLiquidationPo(liquidation)

图 2:

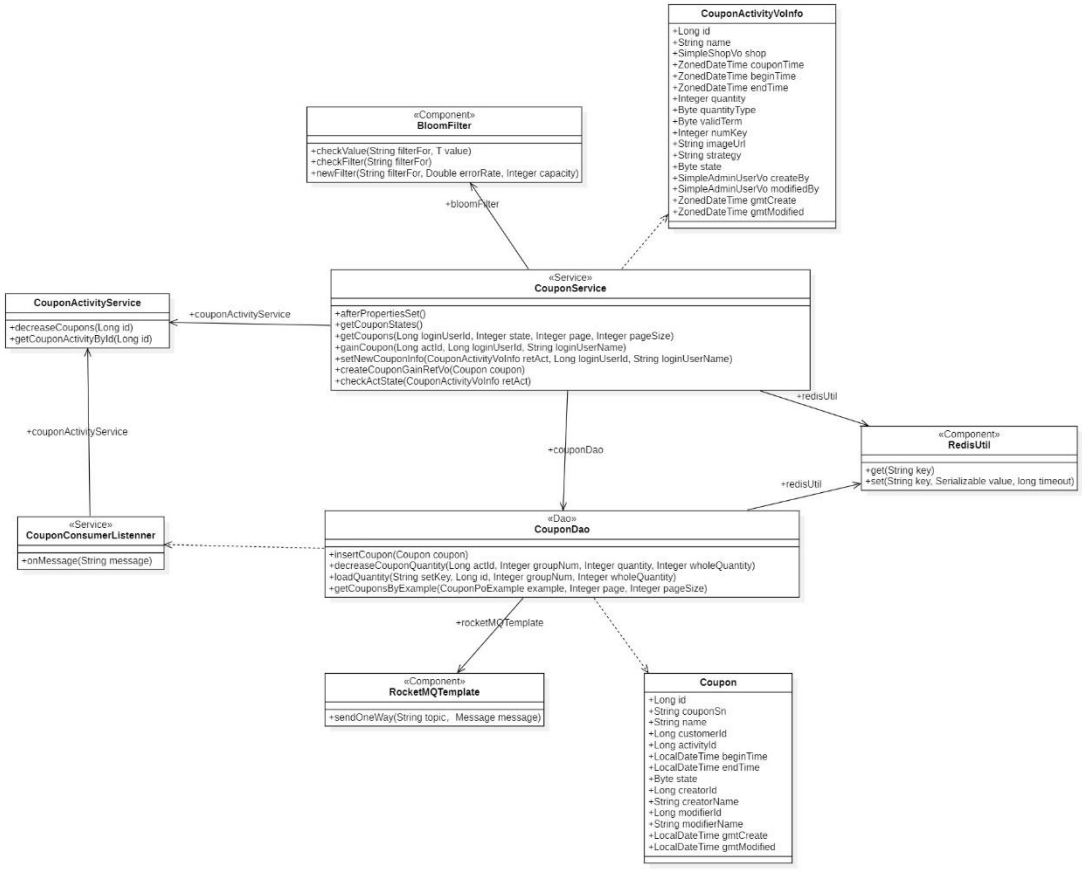


需要清算的主要为两大类：支付 Payment 与退款 Refund，每一类又分为订单与售后两个小类，开始清算方法按照小类划分为四个：订单支付、售后支付、订单退款、售后退款。时序图 2 展示了订单支付清算的流程，其他三类大同小异。

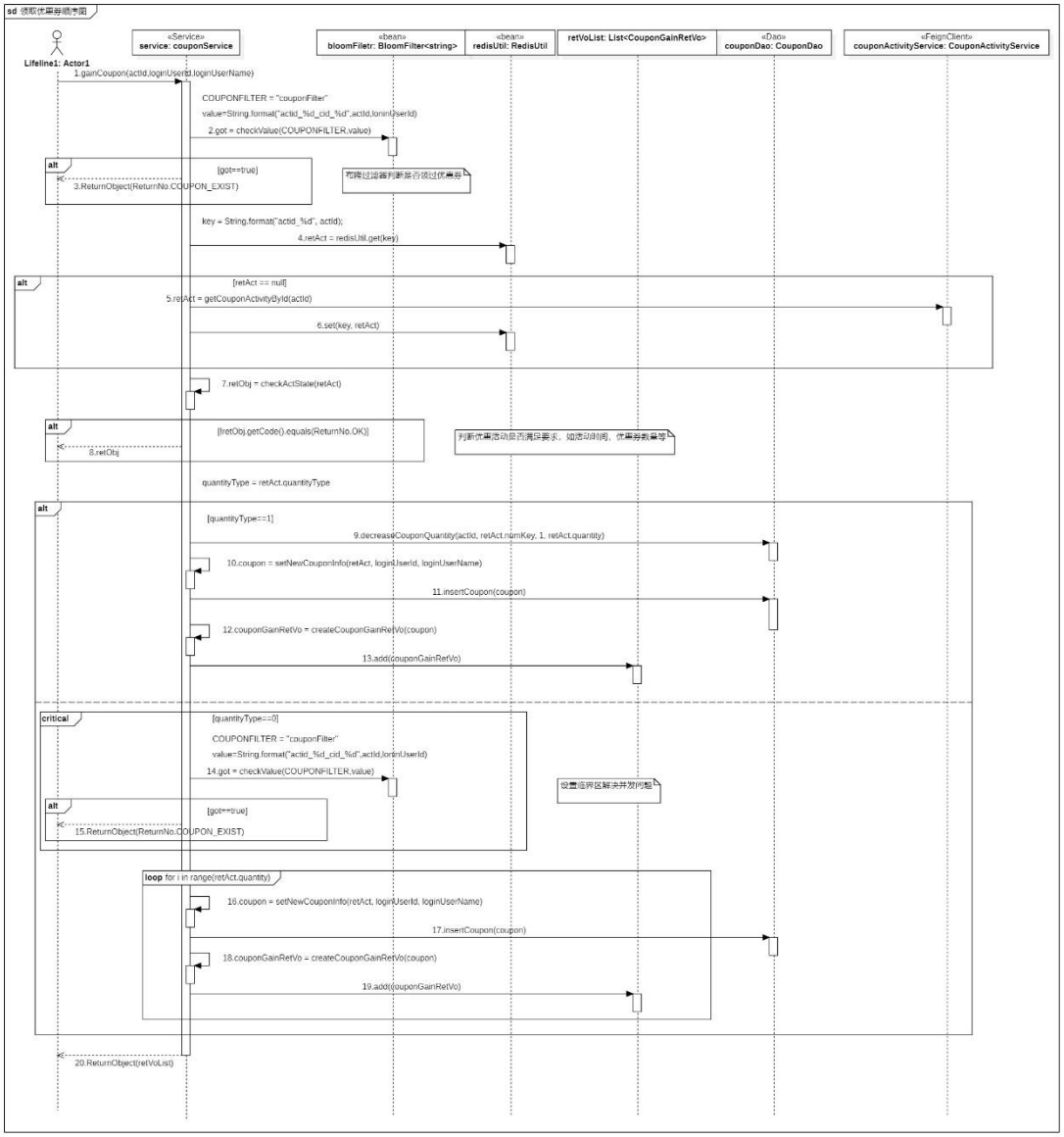
方法中用 LiquidationMap 来存储店铺的 id 以及它本次的清算结果，LiquidationMap 可用进账 Revenue 或者出账 Expenditure 进行更新。每个 orderItem 清算完成之后，可用对应的 Revenue、Expenditure 更新平台收入、佣金、运费等。

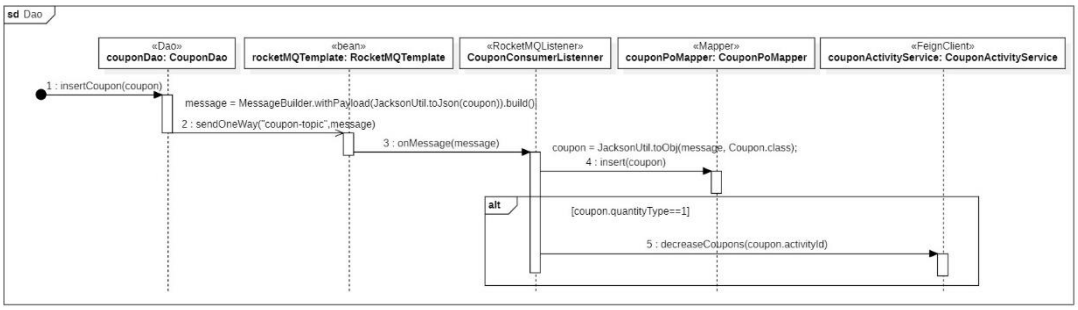
5. 领取优惠券详细设计

1. 类图


CouponActivityVoInfo
+Long id
+String name
+SimpleShopVo shop
+ZonedDateTime couponTime
+ZonedDateTime beginTime
+ZonedDateTime endTime
+Integer quantity
+Byte quantityType
+Byte validTerm
+Integer numKey
+String imageUrl
+String strategy
+Byte state
+SimpleAdminUserVo createBy
+SimpleAdminUserVo modifiedBy
+ZonedDateTime gmtCreate
+ZonedDateTime gmtModified
«Component»
BloomFilter
+checkValue(String filterFor, T value)
+checkFilter(String filterFor)
+newFilter(String filterFor, Double errorRate, Integer capacity)
+bloomFilter
«Service»
CouponService
+afterPropertiesSet()
+getCouponStates()
+getCoupons(Long loginUserId, Integer state, Integer page, Integer pageSize)
+gainCoupon(Long actId, Long loginUserId, String loginUserName)
+setNewCouponInfo(CouponActivityVoInfo retAct, Long loginUserId, String loginUserName)
+createCouponGainRetVo(Coupon coupon)
+checkActState(CouponActivityVoInfo retAct)
CouponActivityService
+decreaseCoupons(Long id)
+getCouponActivityById(Long id)
+couponActivityService
+couponActivityService
+redisUtil
«Component»
RedisUtil
+get(String key)
+set(String key, Serializable value, long timeout)
+couponDao
+redisUtil
«Service»
CouponConsumerListenner
+onMessage(String message)
«Dao»
CouponDao
+insertCoupon(Coupon coupon)
+decreaseCouponQuantity(Long actId, Integer groupNum, Integer quantity, Integer wholeQuantity)
+loadQuantity(String setKey, Long id, Integer groupNum, Integer wholeQuantity)
+getCouponsByExample(CouponPoExample example, Integer page, Integer pageSize)
+rocketMQTemplate
«Component»
RocketMQTemplate
+sendOneWay(String topic, Message message)
Coupon
+Long id
+String couponSn
+String name
+Long customerId
+Long activityId
+LocalDateTime beginTime
+LocalDateTime endTime
+Byte state
+Long creatorId
+String creatorName
+Long modifierId
+String modifierName
+LocalDateTime gmtCreate
+LocalDateTime gmtModified

2. 时序图





用户发起请求之后首先在布隆过滤器中检查是否领取过优惠券，当用户成功领取过一次优惠券之后(对于只能领取一次优惠券的活动)，能有效阻挡重复的领取请求。之后获取对应的优惠券活动，使用 redis 缓存能有效减少 feign 跨模块的请求开销。

优惠券活动分为每人数量和总数控制两种形式。对于总数控制型，生成一张优惠券插入数据库中并减少对应活动的优惠券库存；对于每人数量型，应解决线程并发的问题，所以我们在代码中设置了一个临界区，在其中串行地用布隆过滤器判断同时到达线程的用户是否领

取过优惠券，当有一个线程通过临界区之后，相同用户请求的线程都会被拒绝。之后生成 n 张优惠券并加入数据库中。

对于总数控制型的优惠活动，可利用内存并发读取的特性，对优惠券库存量在 redis 中进行分桶，实现快速减少库存，由于判断库存剩余量和减少库存量具有原子性，所以我们利用 lua 脚本将两个 redis 操作整合在一起。

Redis 中减少库存之后数据库中也应该减少对应的数量，同时也要将用户领取的优惠券放入数据库中，由于数据库读写为 IO 操作，在高并发大负载的情况下响应时间达不到要求，所以我采用了 RocketMQ 发送异步消息来提升响应时间，在 Listener 中进行数据库操作。