需求规格说明书及设计文档

项目场景

> *企业新产品进行线上发布销售，利用营销活动引流时，短时间内用户请求过多，流量暴增导致系统挂掉及出现商品超卖等现象。为了解决这一问题，企业要求定制设计一种“消息队列”进行流量削峰，模拟真实营销活动场景和现实需求进行压力测试。*

本团队自行设计并实现一种“消息队列”来解决商品营销活动场景中突发流量过大容易宕机的问题。该消息队列项目命名为Flash-MQ。消息队列开发完成后，为了验证消息队列方案的可行性并测试该消息队列的性能我们开发了一套基于SpringBoot的前后端分离商城系统，下文简称“商城”。该系统支持普通商品购买及秒杀商品购买，两类商品的购买均对接了Flash-MQ。

功能

商城部分

用户/管理员共有功能

1. 用户功能：
    a. 包括用户注册和登录
    b. 完善个人信息
    c. 修改密码/找回密码
2. 商品功能：
    a. 浏览商品信息，包括商品名称、价格、图片、详情等
    b. 若浏览的商品为秒杀商品可查看秒杀的规则及该商品现有订单的名单公示。
    c. 可购买普通商品和秒杀商品
    d. 可根据商品分类查看商品
3. 订单功能：
    a. 可查看本人所拥有的所有订单
    b. 可对本人拥有的订单进行付款或取消的操作

4. 人体工学功能：
    a. 前端统一现代化设计风格
    b. 前端采用非线性动画
    c. 前端支持暗黑模式（背景颜色反转）防止夜间刺眼

管理员功能

1. 仪表盘功能：
    a. 显示订单总数、订单总额、商品总数、用户总数等常用统计数据
2. 订单管理功能：
    a. 全部订单统计列表，包含订单ID、商品ID、商品名称、订单时间、折扣、应付/实付、用户ID、用户名、手机号、状态等
    b. 查看该订单的用户快照、商品快照
    c. 删除订单
3. 商品管理功能：
    a. 全部商品信息列表，包含商品ID、名称、价格、分类、总库存、现有库存、备注、是否上架等
    b. 增删改查商品
    c. 上下架商品
4. 营销管理功能：
    a. 全部营销活动列表，包含活动ID、对应商品ID\商品名、开始时间、结束时间、已使用次数等
    b. 增删改查活动
    c. 可自定义开始/结束时间
    d. 可自定义秒杀优惠人数排名范围及折扣
5. 用户管理功能：
    a. 全部用户信息列表，包含用户ID、用户名、手机号、邮箱、生日、是否为管理员等
    b. 增删改查用户

消息队列部分

1. 实现消息队列的基本功能，包括PRODUCER-COMSUMER、PUBLISH-SUBSCRIBE
2. 实现JMS 2.0 的大部分特性
3. 支持 JMS 的基本消息类型
4. 支持 AUTO_ACKNOWLEDGE、CLIENT_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE 三种 SessionMode
5. 非持久化的消息通过 Redis 的 publish 特性发送

6. 所有的 Queue 和 Topic 都可以被多个消费者监听，不支持互斥消费行为，所以 createSharedConsumer 的 API 用于创建监听非持久化消息的消息消费者了
7. 支持Java语言
8. 单机吞吐量万级
9. 消息丢失接近0
10. 消息重复可控
11. 高可用性，支持分布式部署
12. 可持久化，无需降低性能即可储存消息。
13. 文档完备（兼容JMS2.0的官方文档）

技术--商城部分

后端技术栈

- SpringBoot ：项目基础框架
- Mysql：关系型数据库服务
- JDBC+Mybatis：持久层服务框架
- Redis：非关系型数据库
- Redisson：Redis通信框架
- FastJSON：内部通信及序列化工具
- Maven ：项目构建系统
- Flash-MQ(自研消息队列)：消息队列
- Nginx：HTTP和反向代理web服务器

前端技术栈

- html +CSS +Javascript ES6：HTML5语言
- Vue：渐进式框架
- Vue Router：VUE路由插件
- VueX：VUE状态管理
- Axios ：一个基于 promise 的 HTTP 库，用于GET/POST请求
- Node.js+webpack：项目构建工具

技术介绍

技术规格及应用介绍

本项目前端基于Vue渐进式框架开发，基本遵守HTML开发规范、Vue-Cli开发规范及Webpack开发规范；后端基于SpringBoot框架开发，基本遵守JAVA开发规范及SpringBoot开发建议。下面将从HTTP请求开始介绍本项目的技术特点。

1. 通过HTTP协议访问本商城，通过Nginx集群（比赛场景为Nginx单机）均衡负载返回页面
2. 页面内通过Axios提供的AJAX方式GET/POST请求页面内的数据
3. Nginx接收到请求地址及参数通过反向代理规则`^.+api/?(.*)$`指向localhost:8000 JAVA后端部分
4. 请求进入网关部分，检查该访问是否拥有对应权限，无权限则直接返回鉴权失败，有权限则继续
5. 进入对应Controller并对应调用Service
6. Service在数据方面优先调用Redis内缓存，Redis缓存数据类型大部分为Hash、小部分为String。若Redis内无对应缓存则调用Dao层访问Mysql数据库，获取数据后存入Redis缓存后返回数据到Service
7. Service经过业务逻辑后将结果返回Controller
8. Controller将接收到的结果包装成Result类，序列化为JSON后写入Response并返回

   此处的Result成员包含：

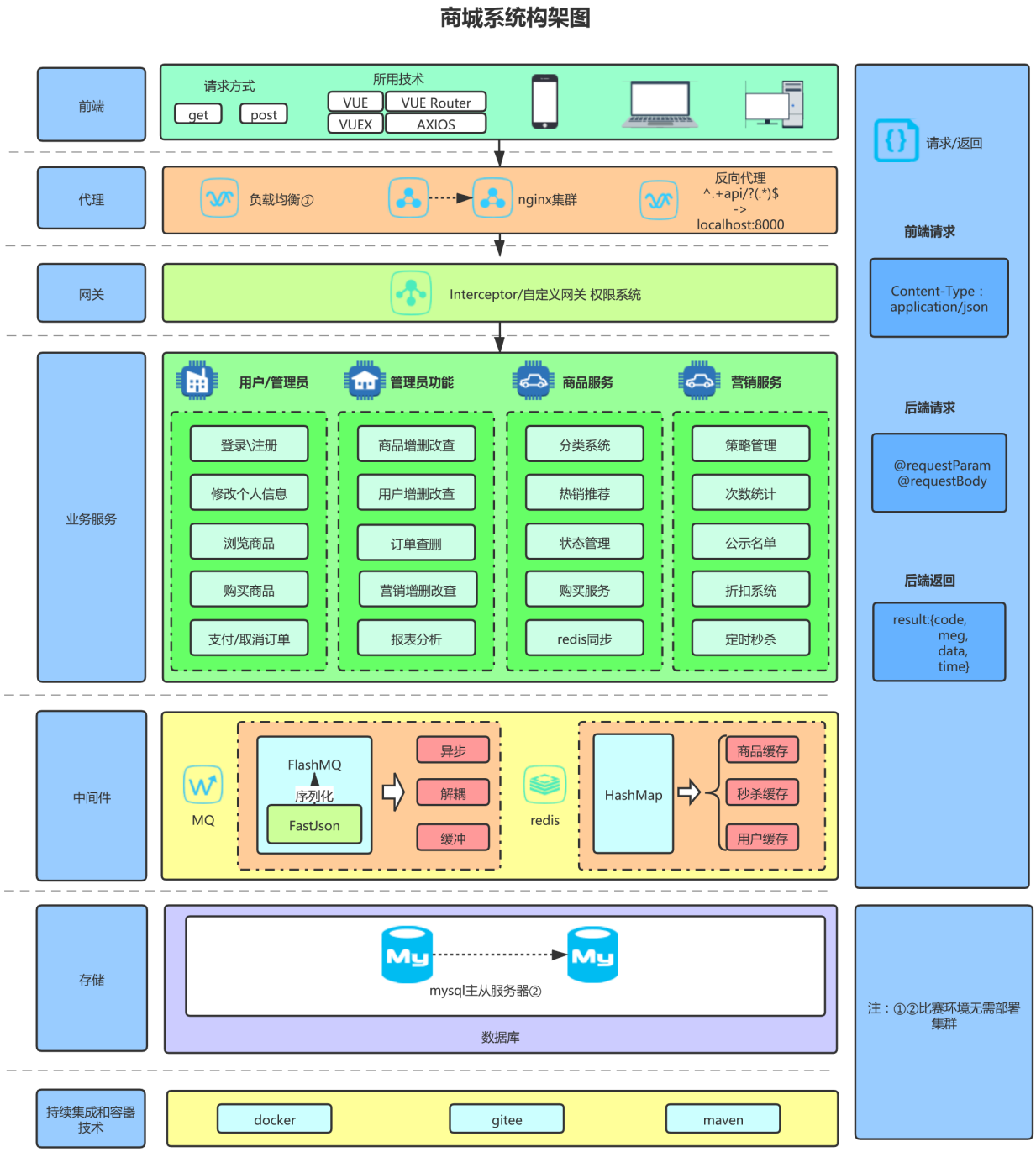
   - code：响应代码
   - meg：响应消息
   - data：响应数据
   - time：响应时间戳
9. 页面接收到数据保存至Vue-Data/VueX并渐进式显示到页面

需值得注意的是：

-

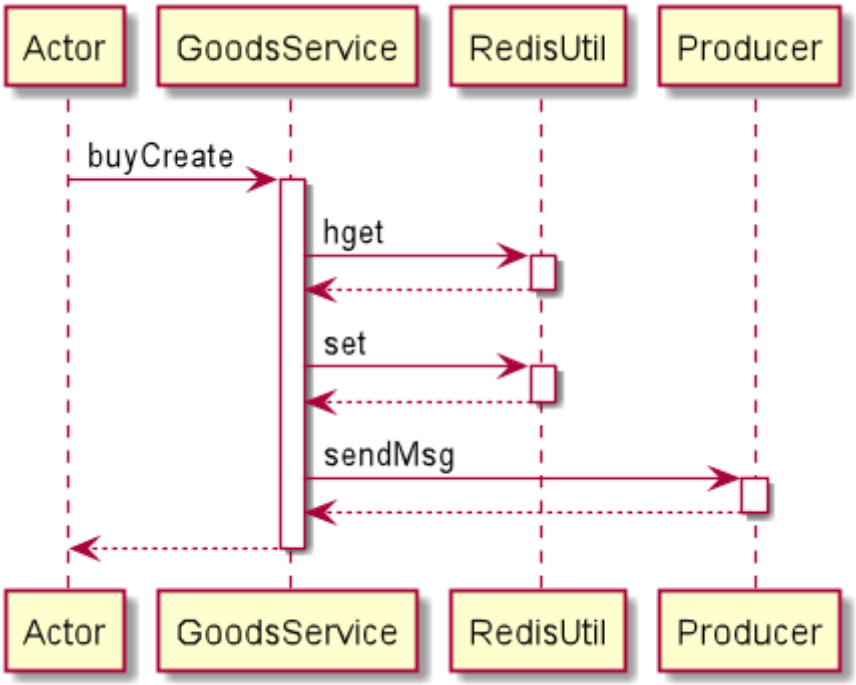
本项目中的购买服务接入了Flash-MQ（自研消息队列）若该请求为购买请求则会将购买服务所需的参数通过FastJSON序列化为JSON，将此JSON发送至购买服务相关队列，由购买服务监听器监听购买服务相关队列进行购买请求操作。同时购买服务将生成一串UUID字符串返回至前端，前端通过该UUID将自动轮询本次的订单生成状态。Flash-MQ拥有低延迟、无丢失、无重复、高可用性、可持续性、可集群等特点可轻松应对流量波峰。

架构图

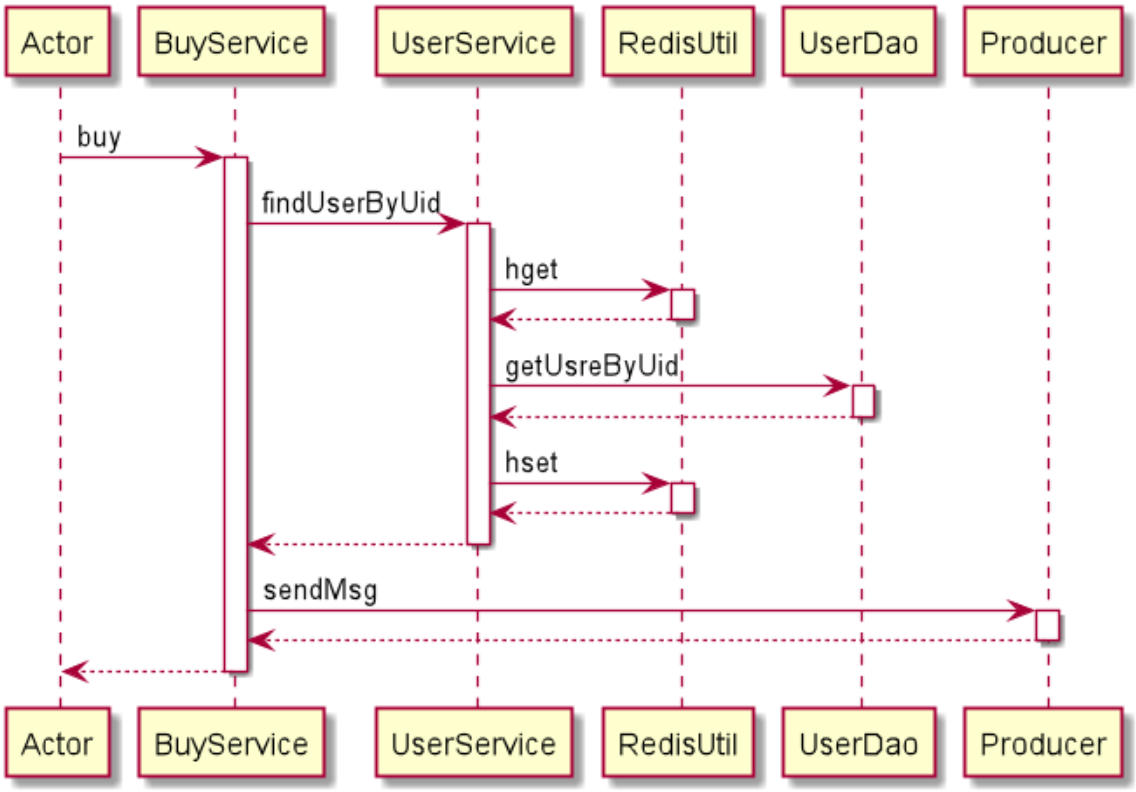


购买服务时序图（按方法执行顺序排序）

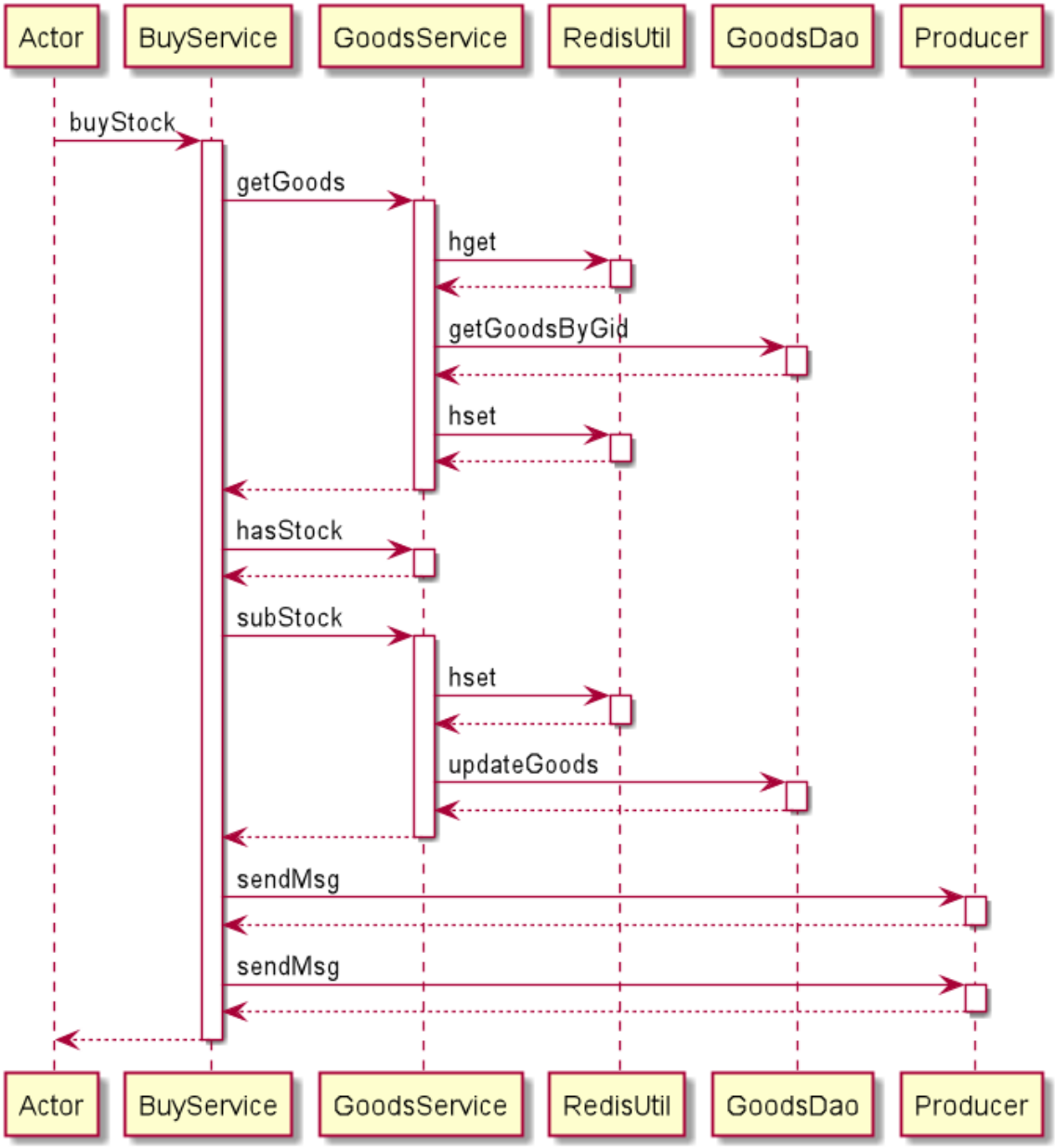
1. buyCreate



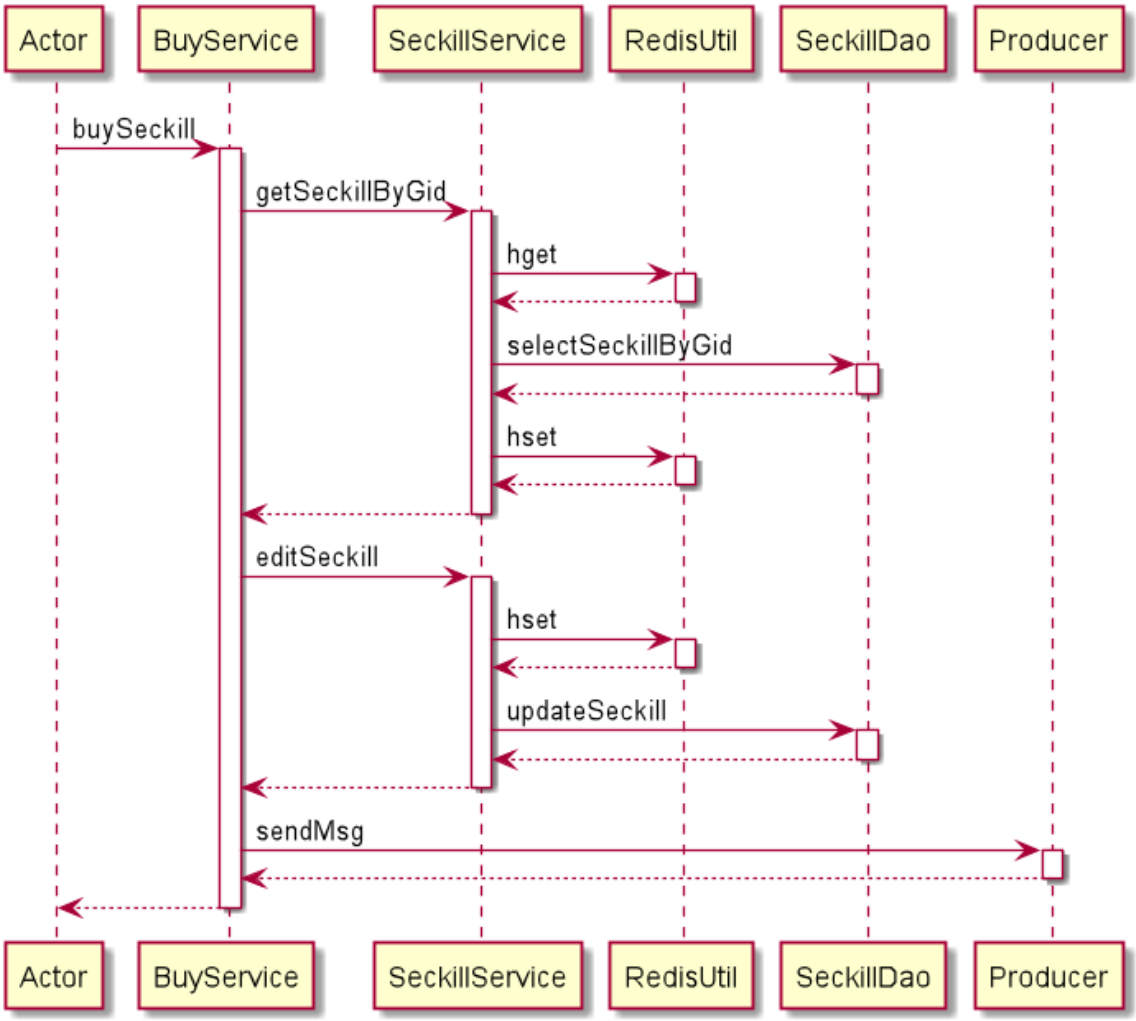
2. buy



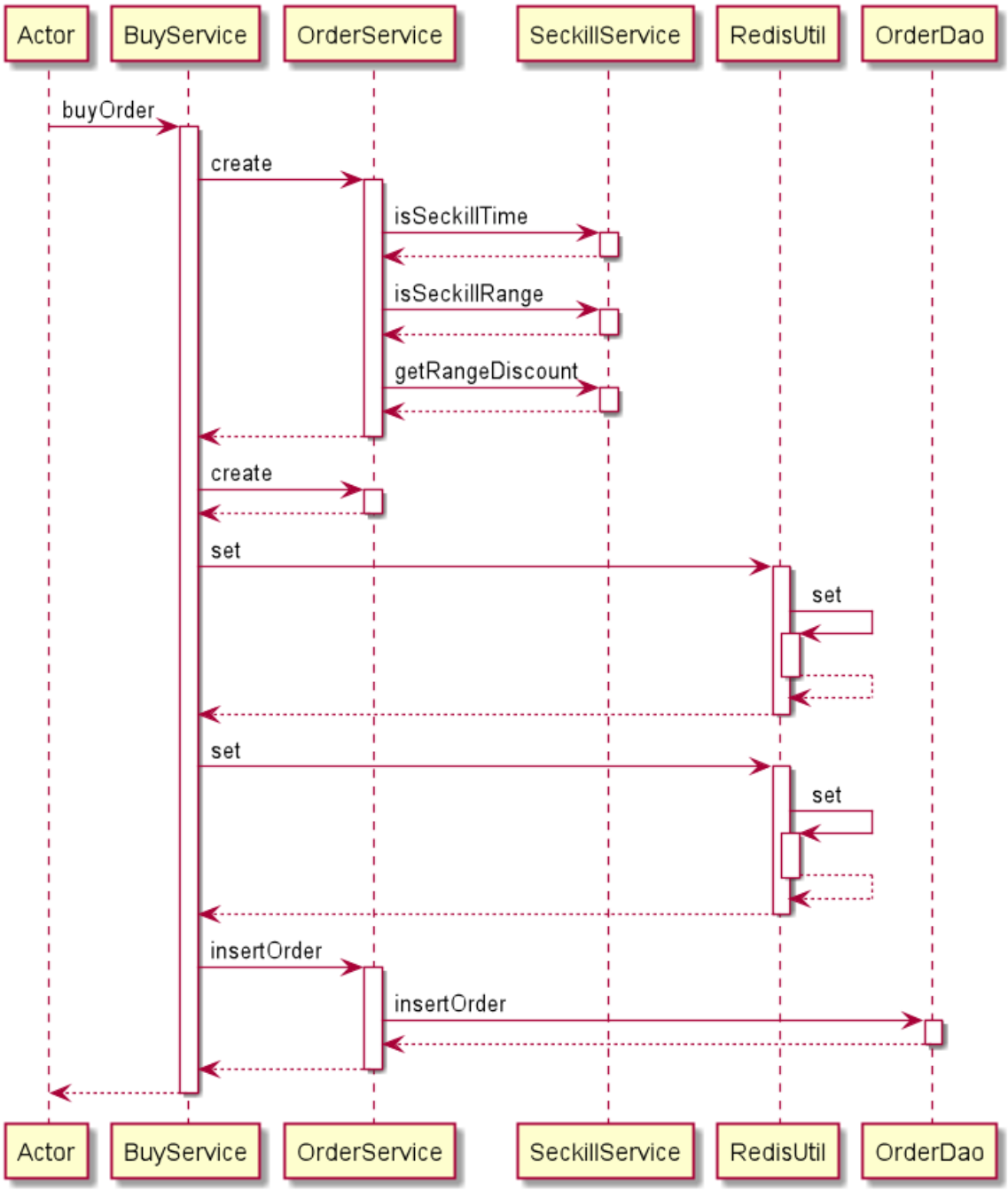
3. buyStock


Actor
BuyService
GoodsService
RedisUtil
GoodsDao
Producer
buyStock
getGoods
hget
getGoodsByGid
hset
hasStock
subStock
hset
updateGoods
sendMsg
sendMsg
Actor
BuyService
GoodsService
RedisUtil
GoodsDao
Producer

4. buySeckill



5. buyOrder



类图

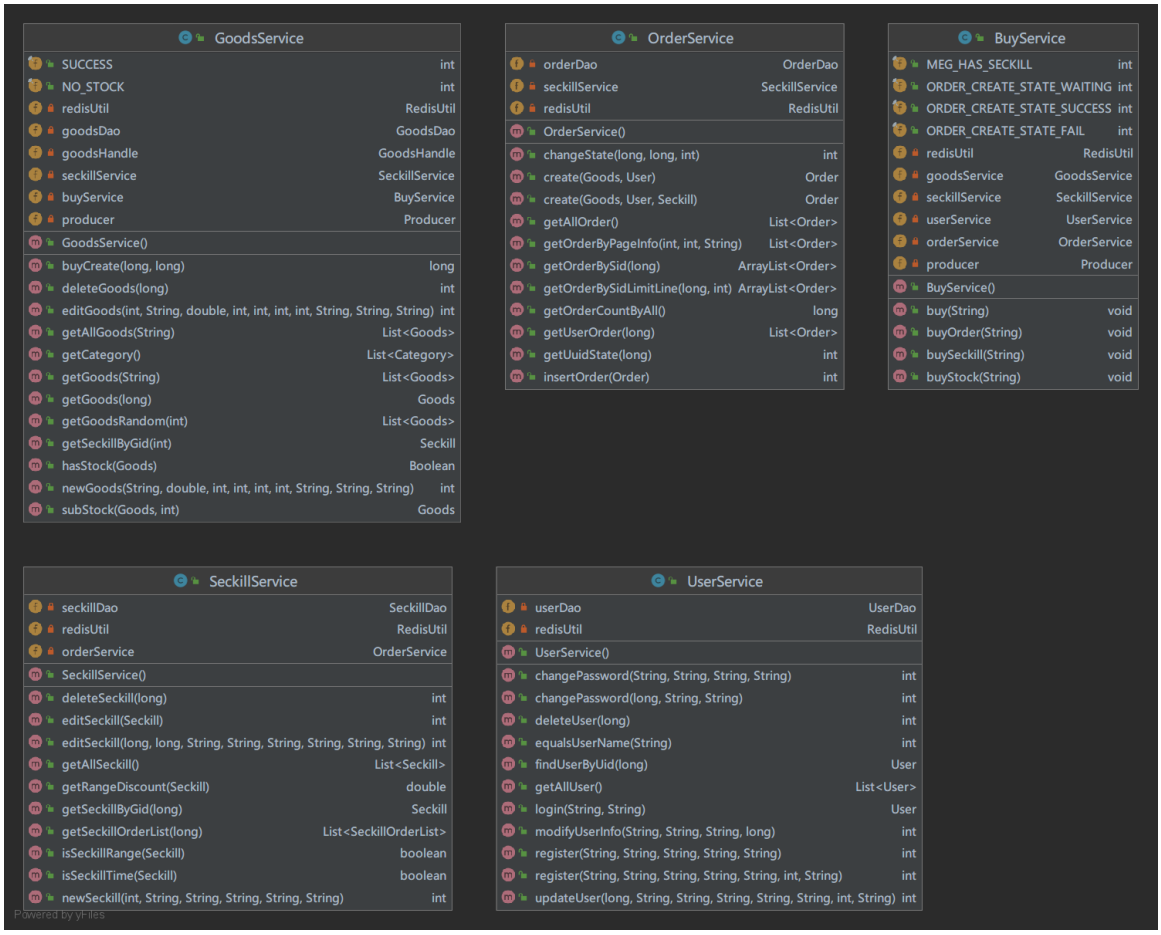
1. interceptor

```
LoginInterceptor
  LoginInterceptor()
  afterCompletion(HttpServletRequest, HttpServletResponse, Object, Exception)  void
  postHandle(HttpServletRequest, HttpServletResponse, Object, ModelAndView)  void
  preHandle(HttpServletRequest, HttpServletResponse, Object)  boolean

AdminInterceptor
  AdminInterceptor()
  afterCompletion(HttpServletRequest, HttpServletResponse, Object, Exception)  void
  postHandle(HttpServletRequest, HttpServletResponse, Object, ModelAndView)  void
  preHandle(HttpServletRequest, HttpServletResponse, Object)  boolean

AdminRequired
LoginRequired
Powered by yFiles
```

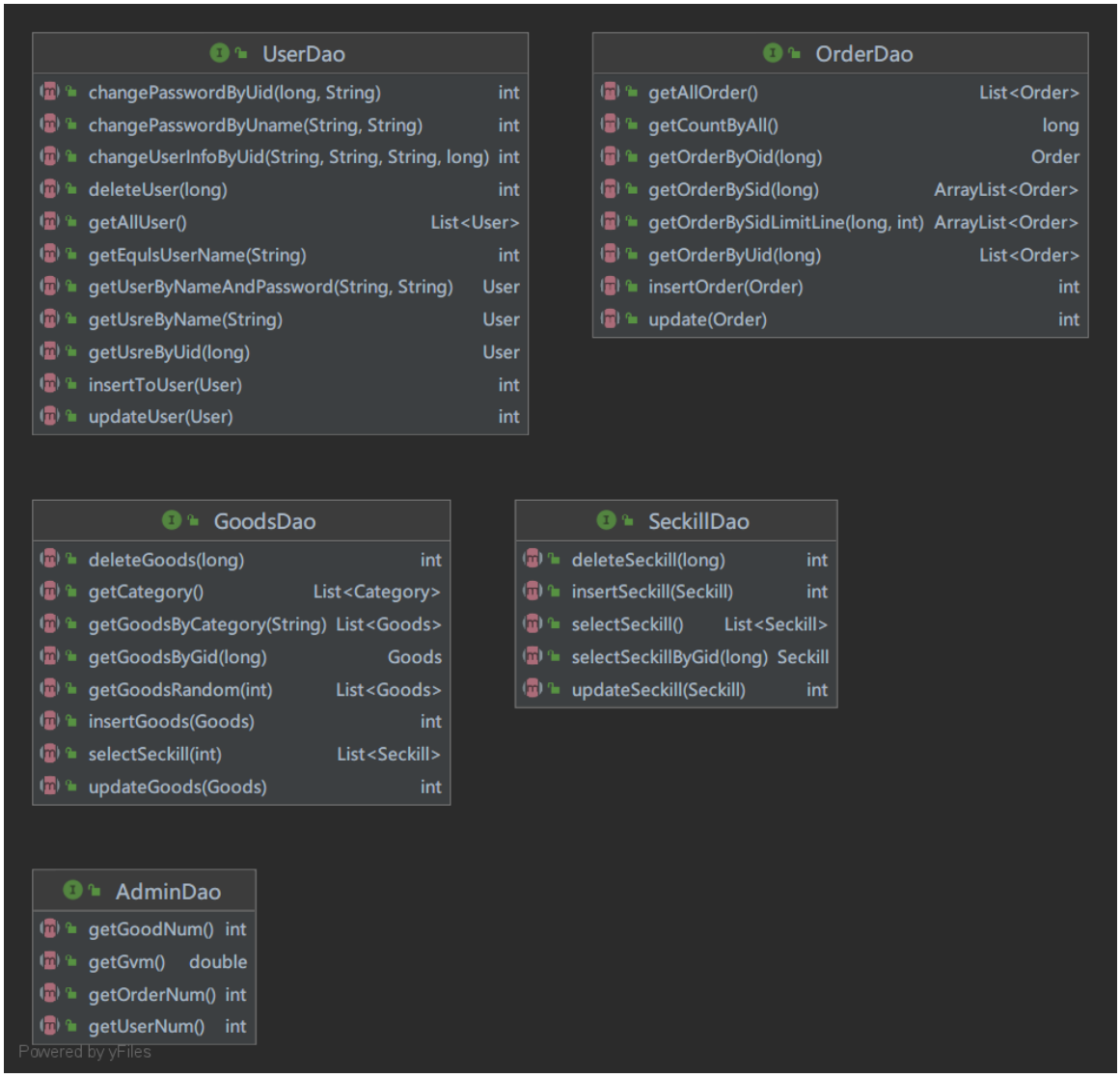
2. controller

```
UserController
  UserController()
  deleteUser(String)  Result
  editPassword(HttpServletRequest, String, String)  Result
  editUser(String, String, String, String, String, String, String, String)  Result
  forgetPassword(String, String, String, String)  Result
  getAllUser()  Result
  login(HttpServletRequest, String, String)  Result
  logout(HttpServletRequest)  Result
  look(HttpServletRequest)  Result
  modifyUserInfo(HttpServletRequest, String, String, String)  Result
  newUser(String, String, String, String, String, String, String)  Result
  register(String, String, String, String, String)  Result
  test()  Result

GoodsController
  GoodsController()
  buy(long, HttpSession)  Result
  deleteGoods(String)  Result
  editGoods(String, String, String, String, String, String, String, String, String, String)  Result
  getAllGoods(String)  Result
  getCategory()  Result
  getGoods(String)  Result
  getGoods(long)  Result
  getGoodsRandom(int)  Result
  getSeckillByGid(String)  Result
  newGoods(String, double, int, int, int, int, String, String, String)  Result
  notNull(String, String, String, String, String, String, String, String)  boolean

OrderController
  OrderController()
  closeOrder(long, HttpSession)  Result
  getAllOrder()  Result
  getOrderByPageInfo(int, int, String)  Result
  getOrderCountByAll()  Result
  getUserOrder(HttpSession)  Result
  getUuidState(long)  Result
  pay(long, HttpSession)  Result

SeckillController
  SeckillController()
  deleteSeckill(long)  Result
  editSeckill(String, String, String, String, String, String, String, String)  Result
  getAllSecikill()  Result
  getSeckillOrderList(long)  Result
  newSecikill(int, String, String, String, String, String)  Result

AdminController
  AdminController()
  getInfo()  Result
Powered by yFiles
```

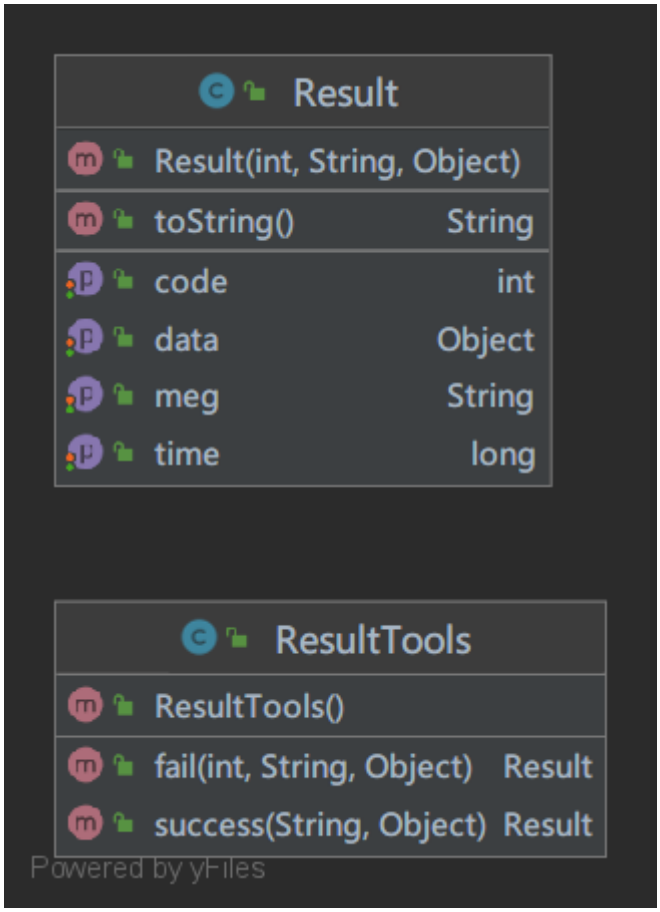
3. service


GoodsService
SUCCESS int
NO_STOCK int
redisUtil RedisUtil
goodsDao GoodsDao
goodsHandle GoodsHandle
seckillService SeckillService
buyService BuyService
producer Producer
GoodsService()
buyCreate(long, long) long
deleteGoods(long) int
editGoods(int, String, double, int, int, int, int, String, String, String) int
getAllGoods(String) List<Goods>
getCategory() List<Category>
getGoods(String) List<Goods>
getGoods(long) Goods
getGoodsRandom(int) List<Goods>
getSeckillByGid(int) Seckill
hasStock(Goods) Boolean
newGoods(String, double, int, int, int, int, String, String, String) int
subStock(Goods, int) Goods
OrderService
orderDao OrderDao
seckillService SeckillService
redisUtil RedisUtil
OrderService()
changeState(long, long, int) int
create(Goods, User) Order
create(Goods, User, Seckill) Order
getAllOrder() List<Order>
getOrderByPageInfo(int, int, String) List<Order>
getOrderBySid(long) ArrayList<Order>
getOrderBySidLimitLine(long, int) ArrayList<Order>
getOrderCountByAll() long
getUserOrder(long) List<Order>
getUuidState(long) int
insertOrder(Order) int
BuyService
MEG_HAS_SECKILL int
ORDER_CREATE_STATE_WAITING int
ORDER_CREATE_STATE_SUCCESS int
ORDER_CREATE_STATE_FAIL int
redisUtil RedisUtil
goodsService GoodsService
seckillService SeckillService
userService UserService
orderService OrderService
producer Producer
BuyService()
buy(String) void
buyOrder(String) void
buySeckill(String) void
buyStock(String) void
SeckillService
seckillDao SeckillDao
redisUtil RedisUtil
orderService OrderService
SeckillService()
deleteSeckill(long) int
editSeckill(Seckill) int
editSeckill(long, long, String, String, String, String, String, String) int
getAllSeckill() List<Seckill>
getRangeDiscount(Seckill) double
getSeckillByGid(long) Seckill
getSeckillOrderList(long) List<SeckillOrderList>
isSeckillRange(Seckill) boolean
isSeckillTime(Seckill) boolean
newSeckill(int, String, String, String, String, String) int
UserService
userDao UserDao
redisUtil RedisUtil
UserService()
changePassword(String, String, String, String) int
changePassword(long, String, String) int
deleteUser(long) int
equalsUserName(String) int
findUserByUid(long) User
getAllUser() List<User>
login(String, String) User
modifyUserInfo(String, String, String, long) int
register(String, String, String, String, String) int
register(String, String, String, String, String, int, String) int
updateUser(long, String, String, String, String, String, int, String) int
Powered by yFiles

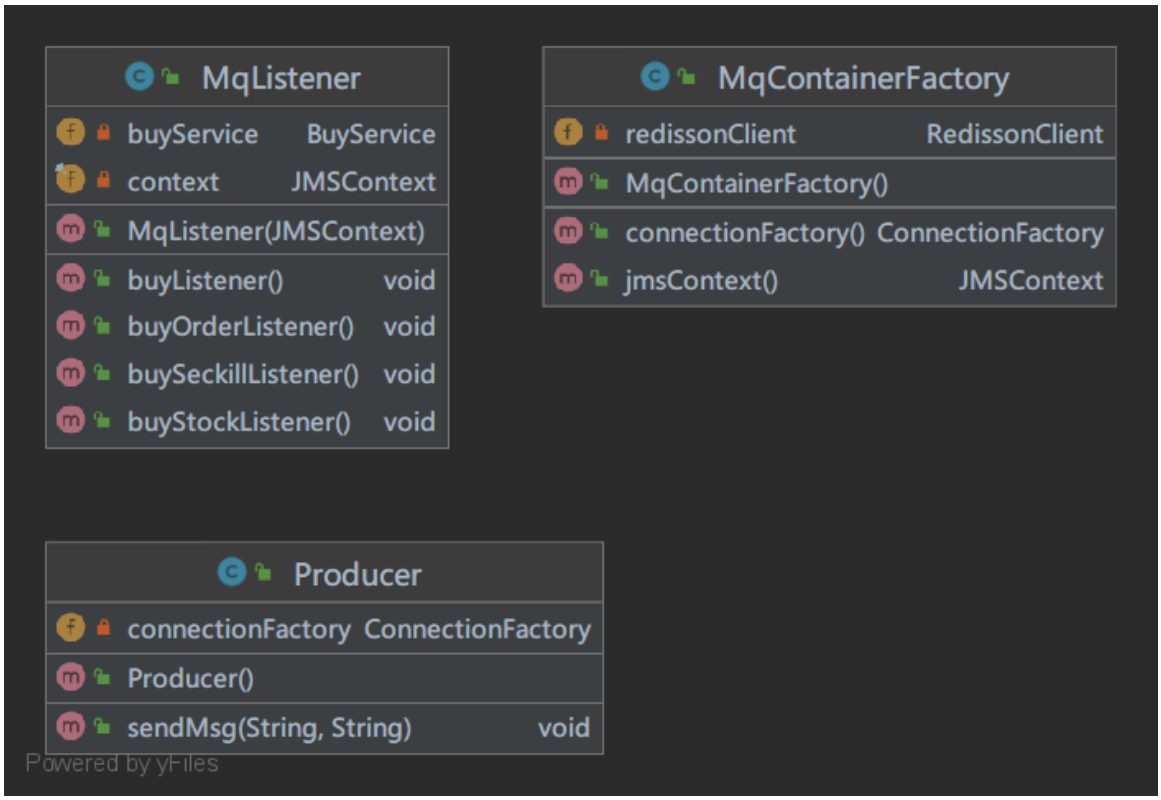
4. dao


UserDao
changePasswordByUid(long, String) int
changePasswordByUname(String, String) int
changeUserInfoByUid(String, String, String, long) int
deleteUser(long) int
getAllUser() List<User>
getEqulsUserName(String) int
getUserByNameAndPassword(String, String) User
getUsreByName(String) User
getUsreByUid(long) User
insertToUser(User) int
updateUser(User) int
OrderDao
getAllOrder() List<Order>
getCountByAll() long
getOrderByOid(long) Order
getOrderBySid(long) ArrayList<Order>
getOrderBySidLimitLine(long, int) ArrayList<Order>
getOrderByUid(long) List<Order>
insertOrder(Order) int
update(Order) int
GoodsDao
deleteGoods(long) int
getCategory() List<Category>
getGoodsByCategory(String) List<Goods>
getGoodsByGid(long) Goods
getGoodsRandom(int) List<Goods>
insertGoods(Goods) int
selectSeckill(int) List<Seckill>
updateGoods(Goods) int
SeckillDao
deleteSeckill(long) int
insertSeckill(Seckill) int
selectSeckill() List<Seckill>
selectSeckillByGid(long) Seckill
updateSeckill(Seckill) int
AdminDao
getGoodNum() int
getGvm() double
getOrderNum() int
getUserNum() int
Powered by yFiles

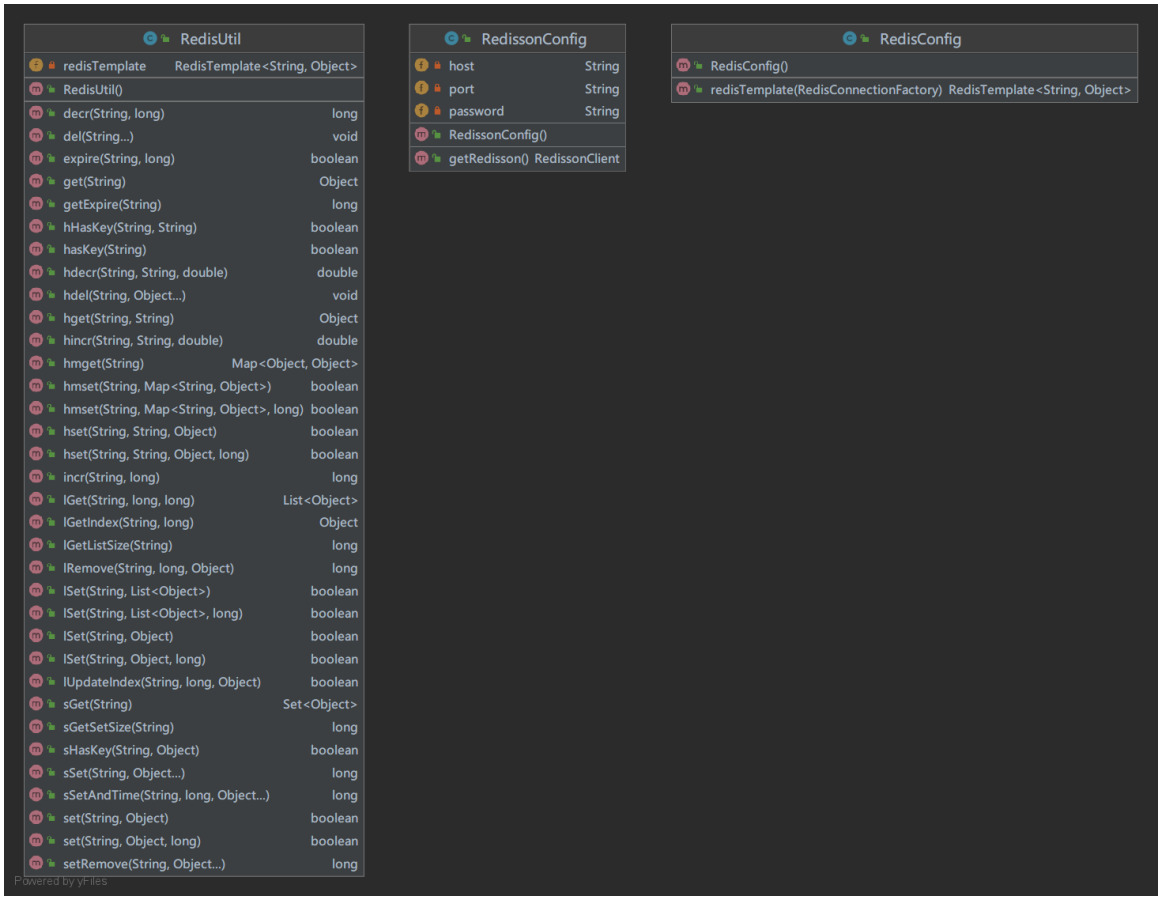
5. result


Result
Result(int, String, Object)
toString() String
code int
data Object
meg String
time long
ResultTools
ResultTools()
fail(int, String, Object) Result
success(String, Object) Result
Powered by yFiles

6. mq


MqListener
buyService BuyService
context JMSContext
MqListener(JMSContext)
buyListener() void
buyOrderListener() void
buySeckillListener() void
buyStockListener() void
MqContainerFactory
redissonClient RedissonClient
MqContainerFactory()
connectionFactory() ConnectionFactory
jmsContext() JMSContext
Producer
connectionFactory ConnectionFactory
Producer()
sendMsg(String, String) void
Powered by yFiles

7. redis


RedisUtil
redisTemplate RedisTemplate<String, Object>
RedisUtil()
decr(String, long) long
del(String...) void
expire(String, long) boolean
get(String) Object
getExpire(String) long
hHasKey(String, String) boolean
hasKey(String) boolean
hdecr(String, String, double) double
hdel(String, Object...) void
hget(String, String) Object
hincr(String, String, double) double
hmget(String) Map<Object, Object>
hmset(String, Map<String, Object>) boolean
hmset(String, Map<String, Object>, long) boolean
hset(String, String, Object) boolean
hset(String, String, Object, long) boolean
incr(String, long) long
lGet(String, long, long) List<Object>
lGetIndex(String, long) Object
lGetListSize(String) long
lRemove(String, long, Object) long
lSet(String, List<Object>) boolean
lSet(String, List<Object>, long) boolean
lSet(String, Object) boolean
lSet(String, Object, long) boolean
lUpdateIndex(String, long, Object) boolean
sGet(String) Set<Object>
sGetSetSize(String) long
sHasKey(String, Object) boolean
sSet(String, Object...) long
sSetAndTime(String, long, Object...) long
set(String, Object) boolean
set(String, Object, long) boolean
setRemove(String, Object...) long
Powered by yFiles
RedissonConfig
host String
port String
password String
RedissonConfig()
getRedisson() RedissonClient
RedisConfig
RedisConfig()
redisTemplate(RedisConnectionFactory) RedisTemplate<String, Object>

8. pojo

技术--Flash-MQ部分

基于 Redis 实现的 Java 消息服务，通过Redisson与Redis通信。通过对Redis的支持继承了Redis的大部分特性，对比赛要求场景添加了更多特性并进行了单独的适配工作。支持 JMS 2.0 的大部分特性。

支持特性

- 支持 JMS2.0 的基本消息类型
- 支持的SessionMode： AUTO_ACKNOWLEDGE、CLIENT_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE
- 通过 Redis 的 publish 特性发送非持久化的消息
- 所有的 Queue 和 Topic 都可以被多个消费者监听，不支持互斥消费行为，createSharedConsumer API 用于创建监听非持久化消息的消息消费者
- 支持Java语言
- 单机吞吐量万级
- 消息丢失接近0
- 消息重复可控
- 高可用性，支持分布式部署
- 可持久化，无需降低性能即可储存消息。
- 文档完备（兼容JMS2.0的官方文档）

不支持特性

- 不支持 JMS 1.x 消息发送/接收相关的 API
- 不支持事务
- 不支持设置 DisableMessageID，消息默认会生成一个消息ID，格式为`ID:Base64压缩后的UUID`
- 不支持 MessageSelector
- 暂不支持优先级队列
- 暂不支持延迟发送 (DeliveryDelay)

基本使用方式

1. 运行Redis
2. 使用Redission连接Redis，创建RedissonClient
3. 创建JmsConnectionFactory，参数为连接ID和RedissonClient
4. 使用JmsConnectionFactory创建JMSContext
5. 使用JMSContext创建Queue
6. 使用JMSContext创建JMSProducer
7. 调用Producer.send(Queue,"Message")方法
8. 使用JMSContext创建JMSConsumer
9. 调用JMSConsumer.receiveNoWait()方法获取Message
10. 调用Message.getBody(class)获取消息体
11. 反序列化并输出消息内容

```
  import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
```

```java
import JmsConnectionFactory;

import javax.jms.*;

public class Example {

   public static void main(String[] args) throws Exception {
      RedissonClient client = Redisson.create();
      ConnectionFactory factory = new
JmsConnectionFactory("ClientId", client);

      try (JMSContext context = factory.createContext()) {
         Queue queue = context.createQueue("Queue");
         JMSProducer producer = context.createProducer();
         producer.send(queue, "Queue Message");

         JMSConsumer consumer = context.createConsumer(queue);
         Message message = consumer.receiveNoWait();
         System.out.println(message.getBody(String.class));
      }

      client.shutdown();
      System.exit(0);
   }
}
```

关于序列化

默认使用 Java 自带的序列化工具实现序列化支持，并允许使用 Java 的 SPI 机制进行扩展。如果需要替换默认实现，通过 SPI 扩展 `Serializer` 接口即可。

更多Api使用方式

请参阅Flash-MQ源码Test测试单元目录部分/flash-mq/src/test/java/top/imzdx/flashmq

或参阅Flash-MQ的HTML/CHM格式帮助文档Flash-MQ/docs/html/index.html or ./FlashMQ-API文档.chm

或参阅JMS2.0中文文档介绍 | JMS2.0规范中文版 (yonyoucloud.com)

或参阅JMS2.0原版文档Flash-MQ/docs/JMS20.pdf