

# 目录

- 1. 秒杀接口优化思路
- 2. 清晰框图解析
- 3. 代码中我们如何实现
  - 3.1 库存预加载到Redis中
  - 3.2 开始秒杀，预减库存
  - 3.3 加入消息队列中（Direct Exchange）
  - 3.4 消息发送过程
  - 3.5 消息出队处理
    - 3.5.1 秒杀方法
  - 3.6 与前端进行交互的秒杀结果
    - 3.6.1 getMiaoshaResult方法

---

## 1. 秒杀接口优化思路

重点我们是要减少对数据库的访问

- 1. 系统初始化时，将秒杀商品库存加载到Redis中
- 2. 收到请求，在Redis中预减库存，库存不足时，直接返回秒杀失败
- 3. 秒杀成功，将订单压入消息队列，返回前端消息“排队中”（像12306的买票）
- 4. 消息出队，生成订单，减少库存
- 5. 客户端在以上过程执行过程中，将一直轮询是否秒杀成功

---

## 2. 清晰框图解析

这个方法会在程序启动的时候执行，我们在这方法中，实现预先加载库存到Redis的功能

实现InitializingBean接口，  
重写afterPropertiesSet()方法

开始秒杀

收到秒杀请求，在Redis中预减库存

秒杀成功

将该秒杀请求压入消息队列，  
其中包含了user和goodsId属性，  
用来创建订单

消息出队列

出队将消息进行消费，减少库存，  
创建秒杀订单

OK

## 3. 代码中我们如何实现

### 3.1 库存预加载到Redis中

这里我们是通过实现 `InitializingBean` 接口，重写其中 `afterProperties` 方法 达成的

```
1 public class MiaoshaController implements InitializingBean {  
2     @Override  
3     ...  
}
```

```

4      public void afterPropertiesSet() throws Exception {
5          //系统启动的时候，就将数据存入Redis
6
7          //加载所有秒杀商品
8          List<GoodsVo> goodsVos = goodsService.listGoodsVo();
9          if(goodsVos == null)
10             return;
11          //存入Redis中，各秒杀商品的数量
12          for (GoodsVo good : goodsVos){
13              redisService.set(GoodsKey.miaoshaGoodsStockPrefix,""+good.getId(),good.getSto
14              map.put(good.getId(),false);
15          }
16      }
17
18      .....
19  }

```

1. 我们先从数据库中将秒杀商品的信息读取出来，再一个一个加载到缓存中
2. 注意一下其中有一个map，它添加了对应Id-false的键值对，它表示的是该商品没有被秒杀完，用于下文中，当商品秒杀完，阻止其对redis服务的访问（后文还会提到）

## 3.2 开始秒杀，预减库存

```

1      //user不能为空，空了去登陆
2      if(user == null){
3          return Result.error(CodeMsg.SESSION_ERROR);
4      }
5
6      //HashMap内存标记，减少Redis访问时间
7      boolean over = map.get(goodsId);
8      if(over)
9          return Result.error(CodeMsg.MIAO_SHA_OVER);
10
11      //收到请求，预减库存
12      Long count = redisService.decr(GoodsKey.miaoshaGoodsStockPrefix, "" + goodsId);
13      if(count <= 0){
14          map.put(goodsId,true);
15          return Result.error(CodeMsg.MIAO_SHA_OVER);
16      }

```

1. 首先用户不能为空

2. 这里我们又看见了map，它写在了Redis服务前边，当商品秒杀完毕的时候，这样就能防止它再去访问Redis服务了
3. 预减库存，库存小于0的时候就返回秒杀失败

### 3.3 加入消息队列中 (Direct Exchange)

```
1      //判断是否已经秒杀过了
2      MiaoshaOrder miaoshaOrder = orderService.selectMiaoshaOrderByUserIdGoodsId(user.g
3      if(miaoshaOrder != null)
4          return Result.error(CodeMsg.REPEATE_MIAOSHA);
5
6      //加入消息队列
7      MiaoshaMessage miaoshaMessage = new MiaoshaMessage();
8      miaoshaMessage.setGoodsId(goodsId);
9      miaoshaMessage.setMiaoShaUser(user);
10     mqSender.sendMiaoshaMessage(miaoshaMessage);
```

1. 在其之前我们有一个判断，判断该用户是不是重复秒杀，其实这一步是多余的，因为我们在数据库中已经建立了唯一索引，将userId和GoodsId绑定在了一起，不会生成重复的订单
2. 自定义MiaoshaMessage类，创建对象，其中加入我们想要的user和goodsId信息，并将消息发出去

### 3.4 消息发送过程

```
1      @Autowired
2      AmqpTemplate amqpTemplate;
3
4
5      public void sendMiaoshaMessage(MiaoshaMessage miaoshaMessage){
6          String msg = RedisService.beanToString(miaoshaMessage);
7          log.info("miaosha send msg:" + msg);
8          amqpTemplate.convertAndSend(MQConfig.MIAOSHA_QUEUE,msg);
9      }
```

- 用SpringBoot框架提供的AmqpTemplate实例来为我们的秒杀队列发送消息

### 3.5 消息出队处理

```
1      @RabbitListener(queues = MQConfig.MIAOSHA_QUEUE)
2      public void receiveMiaoshaMsg(String miaoshaMessage){
3          log.info("miaosha receive msg:" + miaoshaMessage);
4      }
```

```

5      MiaoshaMessage msg = RedisService.stringToBean(miaoshaMessage, MiaoshaMessage.class);
6
7      long goodsId = msg.getGoodsId();
8      MiaoshaUser miaoshaUser = msg.getMiaoshaUser();
9      GoodsVo goodsVo = goodsService.getGoodsVoByGoodsId(goodsId);
10
11     //判断库存
12     int stock = goodsVo.getStockCount();
13     if(stock < 0)
14         return;
15
16     //有库存而且没秒杀过，开始秒杀
17     miaoshaService.miaosha(miaoshaUser, goodsVo);
18 }

```

- 判断库存是否还有，有的话，向下执行秒杀

### 3.5.1 秒杀方法

```

1      @Transactional
2      public OrderInfo miaosha(MiaoshaUser user, GoodsVo goods) {
3          //库存减一
4          boolean success = goodsService.reduceStock(goods);
5
6          if(success)
7              //下订单
8              return orderService.createOrder(user, goods);
9          else{
10              setGoodsOver(goods.getId());
11              return null;
12          }
13      }

```

- 该方法我们用@Transactional注解标记，保证减库存和下订单都执行成功
- 注意其中有一个setGoodsOver()方法，它的作用是当该商品库存没有的时候，在redis中存一个标志，下面我们接着看

## 3.6 与前端进行交互的秒杀结果

```

1      /**
2       * orderId 成功
3       * -1 秒杀失败
4       * 0 继续轮询
5       * @param miaoshaUser

```

```

5      * @param goodsId
6      * @return
7      */
8      @RequestMapping(value = "/result",method = RequestMethod.GET)
9      @ResponseBody
10     public Result<Long> miaoshaResult(MiaoShaUser miaoShaUser,
11                                       @RequestParam("goodsId")long goodsId){
12         if(miaoShaUser == null)
13             return Result.error(CodeMsg.SESSION_ERROR);
14
15         long result = miaoshaService.getMiaoshaResult(miaoShaUser.getId(),goodsId);
16         return Result.success(result);
17     }

```

- 这里写了一个/result请求，前端会根据返回值，来判断秒杀的状态

### 3.6.1 getMiaoshaResult方法

```

1     public long getMiaoshaResult(long userId, long goodsId) {
2         MiaoshaOrder order = orderService.selectMiaoshaOrderByUserIdGoodsId(userId, goods
3
4         if(order != null){
5             //秒杀成功
6             return order.getOrderid();
7         }else {
8             boolean isOver = getGoodsOver(goodsId);
9             if(isOver)
10                 return -1;
11             else
12                 //继续轮询
13                 return 0;
14         }
15     }

```

- 用户在秒杀该商品的过程中，在得到秒杀结果之前，会一直进行轮询，直到返回orderId或者-1来告知秒杀成功与失败
- 该方法中，从数据库中看看能不能查询到秒杀订单信息，有说明秒杀成功，返回订单号；失败了则获取redis中的是否秒杀完的标志，跟前边setGoodsOver()相对应，这里的getGoodsOver()便是对set的值进行获取，如果没有库存了则说明秒杀失败了，否则要继续轮询了（已经秒杀到，但是订单还没有创建完成）