

尚品汇商城复习

版本：V 1.0

秒杀模块

一、秒杀业务分析

1、需求分析

所谓“秒杀”，就是商家发布一些超低价格的商品，所有买家在同一时间网上抢购的一种销售方式。通俗一点讲就是商家为促销等目的组织的网上限时抢购活动。由于商品价格低廉，往往一上架就被抢购一空，有时只用一秒钟。

秒杀商品通常有三种限制：库存限制、时间限制、购买量限制。

(1) 库存限制：商家只拿出限量的商品来秒杀。比如某商品实际库存是 200 件，但只拿出 50 件来参与秒杀。我们可以将其称为“秒杀库存”。

(2) 时间限制：通常秒杀都是有特定的时间段，只能在设定时间段进行秒杀活动；

(3) 购买量限制：同一个商品只允许用户最多购买几件。比如某手机限购 1 件。张某第一次买个 1 件，那么在该次秒杀活动中就不能再次抢购

需求：

(1) 商家提交秒杀商品申请，录入秒杀商品数据，主要包括：商品标题、原价、秒杀价、商品图片、介绍等信息

(2) 运营商审核秒杀申请

(3) 秒杀频道首页列出当天的秒杀商品，点击秒杀商品图片跳转到秒杀商品详情页。

(4) 商品详情页显示秒杀商品信息，点击立即抢购进入秒杀，抢购成功时预减库存。当库存为 0 或不在活动期范围内时无法秒杀。

(5) 秒杀成功，进入下单页填写收货地址、电话、收件人等信息，完成下订单，然后跳转到支付页面，支付成功，跳转到成功页，完成秒杀。

(6) 当用户秒杀下单 30 分钟内未支付，取消订单，调用支付宝的关闭订单接口。

2、秒杀功能分析

列表页



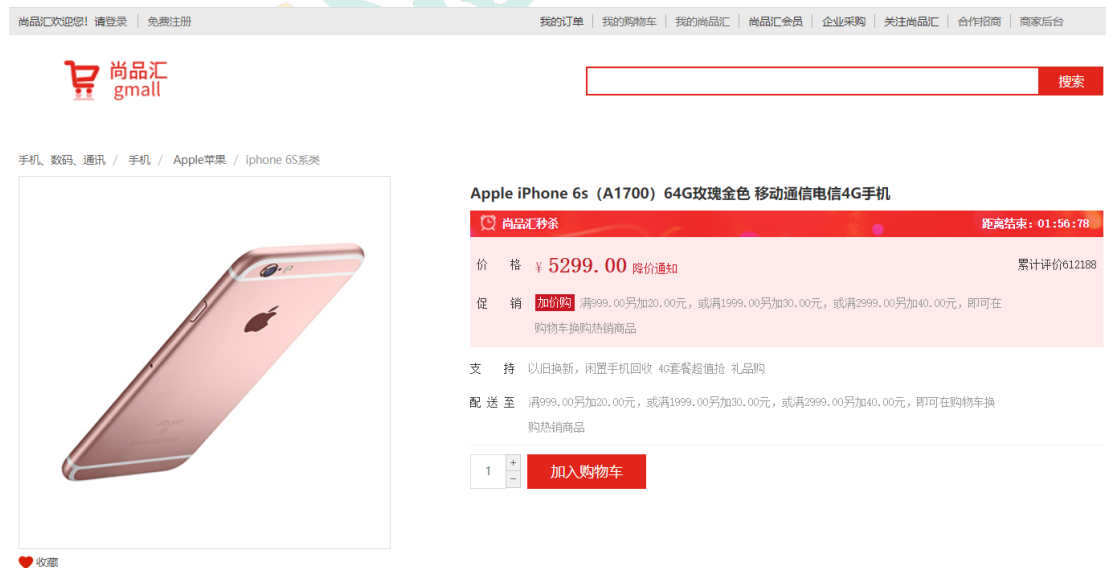
Apple苹果iPhone 6s 32G金色 移动联通电信4G手机
¥6088 ~~¥6988~~
已售87%
剩余 29件
立即抢购

Apple苹果iPhone 6s 32G金色 移动联通电信4G手机
¥6088 ~~¥6988~~
已售87%
剩余 29件
立即抢购

Apple苹果iPhone 6s 32G金色 移动联通电信4G手机
¥6088 ~~¥6988~~
已售87%
剩余 29件
立即抢购

Apple苹果iPhone 6s 32G金色 移动联通电信4G手机
¥6088 ~~¥6988~~
已售87%
剩余 29件
立即抢购

详情页



尚品汇欢迎您! 请登录 | 免费注册

我的订单 | 我的购物车 | 我的尚品汇 | 尚品汇会员 | 企业采购 | 关注尚品汇 | 合作招商 | 商家后台

手机、数码、通讯 / 手机 / Apple苹果 / iPhone 6S系列

Apple iPhone 6s (A1700) 64G玫瑰金色 移动通信电信4G手机

尚品汇秒杀 距离结束: 01:56:78

价 格 ¥ 5299.00 降价通知 累计评价612188

促 销 加价购 满999.00另加20.00元, 或满1999.00另加30.00元, 或满2999.00另加40.00元, 即可在购物车换购热销商品

支 持 以旧换新, 闲置手机回收 4G套餐超值抢 礼品购

配 送 至 满999.00另加20.00元, 或满1999.00另加30.00元, 或满2999.00另加40.00元, 即可在购物车换购热销商品

1 + - 加入购物车

❤ 收藏

排队页



搜索

排队中...

抢购失败

抢购成功 去下单

抢购成功 我的订单

个

下单页

收件人信息

张三	北京市昌平区宏福科技园综合楼6层 15010658793	默认地址
李四	北京市昌平区宏福科技园综合楼5层 13590909098	
王五	北京市昌平区宏福科技园综合楼3层 18012340987	

支付方式



在线支付	货到付款
------	------

送货清单

配送方式:

天天快递	配送时间: 预计8月10日 (周三) 09:00-15:00送达
------	----------------------------------

商品清单:

	Apple iPhone 6s (A1700) 64G 玫瑰金色 移动联通电信4G手机硅胶透明防摔软壳 本色系列	¥ 5399.00	X1	有货
	7天无理由退货			
	Apple iPhone 6s (A1700) 64G 玫瑰金色 移动联通电信4G手机硅胶透明防摔软壳 本色系列	¥ 5399.00	X1	有货
	7天无理由退货			

买家留言:

发票信息

普通发票 (电子) 个人 明细

使用优惠/抵用

1件商品, 总商品金额 ¥5399.00

返现: 0.00

运费: 0.00

应付金额: ¥5399.00

寄送至: 北京市昌平区宏福科技园综合楼6层 收货人: 张三 15010658793

提交订单

支付页

尚品汇欢迎您! 请登录 | 免费注册
我的订单 | 我的购物车 | 我的尚品汇 | 尚品汇会员 | 企业采购 | 关注尚品汇 | 合作招商 | 商家后台



 订单提交成功, 请您及时付款, 以便尽快为您发货~~

请您在提交订单4小时之内完成支付, 超时订单会自动取消。订单号: 145687
应付金额: **¥17,654**

重要说明:

- 尚品汇商城支付平台目前支持**支付宝**支付方式。
- 其它支付渠道正在调试中, 敬请期待。
- 为了保证您的购物支付流程顺利完成, 请保存以下支付宝信息。

支付宝账户信息: (很重要, 请保存!!!)
















- 支付帐号: 11111111
- 密码: 111111
- 支付密码: 111111

支付平台




3、数据库表

秒杀商品表 seckill_goods

名称	类型	空	默认值	其他	备注
索引 (1)					
 主索引	id			unique	
字段 (15)					
 id	bigint(20)	否	<auto_increment>		
 spu_id	bigint(20)	是	<空>		spu_id
 sku_id	bigint(20)	是	<空>		sku_id
 sku_name	varchar(100)	是	<空>		标题
 sku_default_img	varchar(150)	是	<空>		商品图片
 price	numeric(10,2)	是	<空>		原价格
 cost_price	numeric(10,2)	是	<空>		秒杀价格
 create_time	datetime	是	<空>		添加日期
 check_time	datetime	是	<空>		审核日期
 status	varchar(1)	是	<空>		审核状态
 start_time	datetime	是	<空>		开始时间
 end_time	datetime	是	<空>		结束时间
 num	int(11)	是	<空>		秒杀商品数
 stock_count	int(11)	是	<空>		剩余库存数
 sku_desc	varchar(2000)	是	<空>		描述

4、秒杀实现思路

(1) 秒杀的商品要提前放入到 redis 中 (缓存预热), 什么时间放入? 凌晨放入当天的秒杀商品数据。

(3) 用户提交秒杀请求, 将秒杀商品与用户 id 关联发送给 mq, 然后返回, 秒杀页面通过轮询接口查看是否秒杀成功

(4) 我们秒杀只是为了获取一个秒杀资格，获取秒杀资格就可以到下单页下订单，后续业务与正常订单一样

(5) 下单我们需要注意的问题:

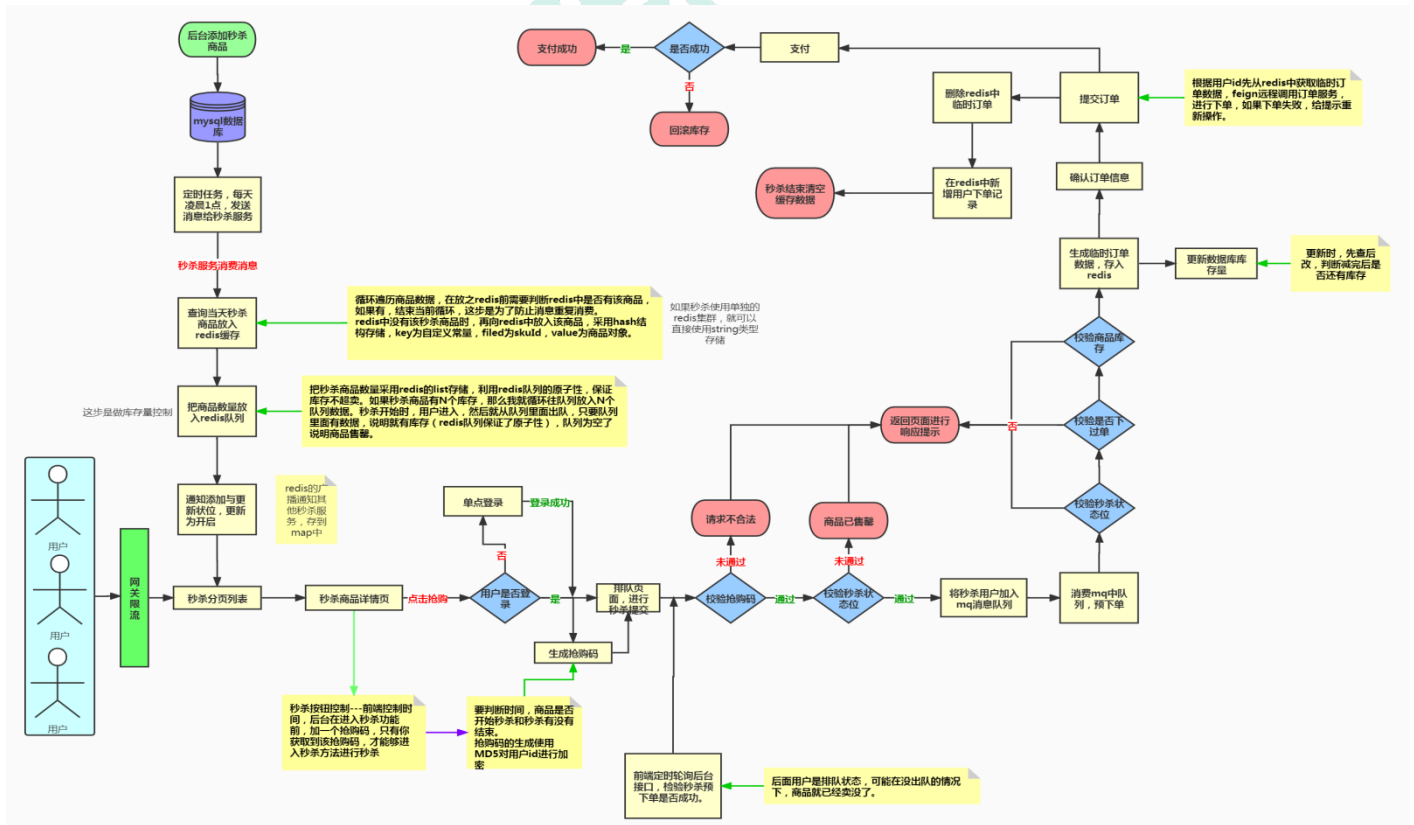
状态位如何同步到集群中的其他节点?

如何控制一个用户只下一个订单？

如何控制库存超卖?

如何控制访问压力?

业务流程图：



开发步骤：

0. 做个定时任务将要秒杀的商品放入消息队列

1. 消费消息队列先将秒杀商品放入缓存

1.1 将秒杀商品信息放入缓存中 hash 数据结构中

1.2 放入一个 list 数据来存储商品的数据量

1.3 利用缓存的订阅与发布功能，来更新状态位

意义：看商品是否售罄！

2. 页面显示秒杀商品以及商品详情

2.1 通过缓存查询所有的秒杀商品

2.2 通过商品 Id 查询秒杀的商品详情

2.3 秒杀详情中的秒杀按钮要设定一个下单码 {防止用户直接进入秒杀

业务}

3. 进入秒杀

3.1 获取秒杀的下单码进行校验！

3.2 判断状态位

3.3 将用户下单请求放入到 mq，是为了防止高并发

3.4 消费下单的 mq 消息，再次验证状态位，用户是否已经下单，判断库存，保存预下单的用户 Id 以及商品 Id，

并将真正下单数据放入缓存，并更新数据商品的库存数

4. 检查抢购状态

4.1 根据用在缓存中是否有 key{用户 key、用户 key 对应的商品 key}，以及状态位，是否已经下过订单

5. 下订单

5.1 直接从缓存中获取下单数据，并显示下单列表页面！

5.2 提交订单

6. 秒杀活动结束后清空缓存数据

6.1 商品数据

6.2 用户数据

6.3 订单数据

二、秒杀商品导入缓存

缓存数据实现思路：前面的业务中我们把定时任务写在了 service-task 模块中，为了统一管理我们的定时任务，在秒杀业务中也是一样，为了减少 service-task 模块的耦合度，我们可以在定时任务模块只发送 mq 消息，需要执行定时任务的模块监听该消息即可，这样有利于我们后期动态控制，例如：每天凌晨一点我们发送定时任务信息到 mq 交换机，如果秒杀业务凌晨一点需要导入数据到缓存，那么秒杀业务绑定队列到交换机就可以了，其他业务也是一样，这样就做到了很好的扩展。

上面提到我们要控制库存数量，不能超卖，那么如何控制呢？在这里我们提供一种解决方案，那就我们在导入商品缓存数据时，同时将商品库存信息导入队列，利用 redis 队列的原子性，保证库存不超卖

库存加入队列实施方案

- 1, 如果秒杀商品有 N 个库存，那么我就循环往队列放入 N 个队列数据
- 2, 秒杀开始时，用户进入，然后就从队列里面出队，只要队列里面有数据，说明就有库存（redis 队列保证了原子性），队列为空了说明商品售罄

1、编写定时任务

在 service-task 模块发送消息

编写定时任务

```
/**
 * 每天凌晨 1 点执行
 */
```

```
//@Scheduled(cron = "0/30 * * * * ?")
@Scheduled(cron = "0 0 1 * * ?")
public void task1() {
    rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_TASK,
MqConst.ROUTING_TASK_1, "");
}
```

2、监听定时任务信息

在 service-activity 模块绑定与监听消息，处理缓存逻辑，更新状态位

2.1、数据导入缓存

监听消息

```
package com.atguigu.gmall.activity.receiver;
@Component
public class SeckillReceiver {
    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;

    @RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = MqConst.QUEUE_TASK_1),
        exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_TASK),
        key = {MqConst.ROUTING_TASK_1}
    ))
    public void importItemToRedis(Message message, Channel channel) throws
IOException {
        QueryWrapper<SeckillGoods> queryWrapper = new QueryWrapper<>();
        // 查询审核状态1 并且库存数量大于0, 当天的商品
        queryWrapper.eq("status", 1).gt("stock_count", 0);
        queryWrapper.eq("DATE_FORMAT(start_time, '%Y-%m-%d')",
DateUtil.formatDate(new Date()));
        List<SeckillGoods> list =
seckillGoodsMapper.selectList(queryWrapper);
        // 将集合数据放入缓存中
        if (list!=null && list.size()>0){
            for (SeckillGoods seckillGoods : list) {
                // 使用hash 数据类型保存商品
                // key = seckill:goods field = skuId
                // 判断缓存中是否有当前key
                Boolean flag =
redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).hasKey(seckillGoods.ge
tSkuId().toString());
                if (flag){
```



```

        // 当前商品已经在缓存中有了! 所以不需要在放入缓存!
        continue;
    }
    // 商品id为field, 对象为value 放入缓存 key =
    seckill:goods field = skuId value=商品字符串
    redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGoods.getSkuId().toString(), seckillGoods);
    // hset(seckill:goods,1,{"skuNum 10"})
    // hset(seckill:goods,2,{"skuNum 10"})
    // 根据每一个商品的数量把商品按队列的形式放进redis中
    for (Integer i = 0; i < seckillGoods.getStockCount(); i++) {
        // key = seckill:stock:skuId
        // lpush key value

        redisTemplate.boundListOps(RedisConst.SECKILL_STOCK_PREFIX+seckillGoods.getSkuId()).leftPush(seckillGoods.getSkuId().toString());
    }
    // 手动确认接收消息成功
    channel.basicAck(message.getMessageProperties().getDeliveryTag(),
    false);
    }
}
}

```

2.2、更新状态位

由于我们的秒杀服务是要集群部署 service-activity 的, 我们面临一个问题? RabbitMQ 如何实现对同一个应用的多个节点进行广播呢?

RabbitMQ 只能对绑定到交换机上面的不同队列实现广播, 对于同一队列的消费者他们存在竞争关系, 同一个消息只会被同一个队列下其中一个消费者接收, 达不到广播效果;

我们目前的需求是定时任务发送消息, 我们将秒杀商品导入缓存, 同事更新集群的状态位, 既然 RabbitMQ 达不到广播的效果, 我们就放弃吗? 当然不是, 我们想到一种解决方案, 通过 redis 的发布订阅模式来通知其他兄弟节点, 这不问题就解决了吗?

过程大致如下

应用启动, 多个节点监听同一个队列 (此时多个节点是竞争关系, 一条消息只会发到其中一个节点上)

消息生产者发送消息, 同一条消息只被其中一个节点收到

收到消息的节点通过 redis 的发布订阅模式来通知其他兄弟节点

接下来配置 redis 发布与订阅

2.2.1、redis 发布与订阅实现

```
package com.atguigu.gmall.activity.redis;
@Configuration
public class RedisChannelConfig {

    /**
     * docker exec -it bc92 redis-cli
     * subscribe seckillpush // 订阅 接收消息
     * publish seckillpush admin // 发布消息
     */
    /**
     * 注入订阅主题
     * @param connectionFactory redis 链接工厂
     * @param listenerAdapter 消息监听适配器
     * @return 订阅主题对象
     */
    @Bean
    RedisMessageListenerContainer container(RedisConnectionFactory
connectionFactory,
                                           MessageListenerAdapter
listenerAdapter) {
        RedisMessageListenerContainer container = new
RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        //订阅主题
        container.addMessageListener(listenerAdapter, new
PatternTopic("seckillpush"));
        //这个container 可以添加多个 messageListener
        return container;
    }

    /**
     * 返回消息监听器
     * @param receiver 创建接收消息对象
     * @return
     */
    @Bean
    MessageListenerAdapter listenerAdapter(MessageReceive receiver) {
        //这个地方 是给 messageListenerAdapter 传入一个消息接受的处理器，利用反
射的方法调用 "receiveMessage"
        //也有好几个重载方法，这边默认调用处理器的方法 叫 handleMessage 可以自己
到源码里面看
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }

    @Bean //注入操作数据的template
    StringRedisTemplate template(RedisConnectionFactory connectionFactory)
{
        return new StringRedisTemplate(connectionFactory);
    }
}
```

```
}  
  
}  
  
package com.atguigu.gmall.activity.redis;  
@Component  
public class MessageReceive {  
  
    /**接收消息的方法*/  
    public void receiveMessage(String message){  
        System.out.println("-----收到消息了message: "+message);  
        if(!StringUtils.isEmpty(message)) {  
            /*  
            消息格式  
            skuId:0 表示没有商品  
            skuId:1 表示有商品  
            */  
  
            // 因为传递过来的数据为 “” 6:1” ”  
  
            message = message.replaceAll("\\\"", "");  
            String[] split = StringUtils.split(message, ":");  
            if (split == null || split.length == 2) {  
                CacheHelper.put(split[0], split[1]);  
            }  
        }  
    }  
}
```

CacheHelper 类本地缓存类

```
package com.atguigu.gmall.activity.util;  
  
/**  
 * 系统缓存类  
 */  
public class CacheHelper {  
  
    /**  
     * 缓存容器  
     */  
    private final static Map<String, Object> cacheMap = new  
    ConcurrentHashMap<String, Object>();
```

```
/**
 * 加入缓存
 *
 * @param key
 * @param cacheObject
 */
public static void put(String key, Object cacheObject) {
    cacheMap.put(key, cacheObject);
}

/**
 * 获取缓存
 *
 * @param key
 * @return
 */
public static Object get(String key) {
    return cacheMap.get(key);
}

/**
 * 清除缓存
 *
 * @param key
 * @return
 */
public static void remove(String key) {
    cacheMap.remove(key);
}

public static synchronized void removeAll() {
    cacheMap.clear();
}
}
```

说明:

- 1, RedisChannelConfig 类配置 redis 监听的主题和消息处理器
- 2, MessageReceive 类为消息处理器, 消息 message 为: 商品 id 与状态位, 如: 1:1 表示商品 id 为 1, 状态位为 1

2.2.2、redis 发布消息

监听已经配置好, 接下来我就发布消息, 更改秒杀监听器{ SeckillReceiver }, 如下

完整代码如下

```
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(value = MqConst.QUEUE_TASK_1, durable =
"true"),
    exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_TASK,
type = ExchangeTypes.DIRECT, durable = "true"),
    key = {MqConst.ROUTING_TASK_1}
))
public void importItemToRedis(Message message, Channel channel)
throws IOException {
    //Log.info("importItemToRedis:");

    QueryWrapper<SeckillGoods> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("status", 1);
    queryWrapper.gt("stock_count", 0);
    // 当天的秒杀商品导入缓存
    queryWrapper.eq("DATE_FORMAT(start_time,'%Y-%m-%d')",
DateUtil.formatDate(new Date()));

    List<SeckillGoods> list =
seckillGoodsMapper.selectList(queryWrapper);

    // 把数据放在 redis 中
    for (SeckillGoods seckillGoods : list) {
        if
(redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).hasKey(seckill
Goods.getSkuId().toString()))
            continue;

        redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGood
s.getSkuId().toString(), seckillGoods);

        // 根据每一个商品的数量把商品按队列的形式放进 redis 中
        for (int i = 0; i < seckillGoods.getStockCount(); i++) {

            redisTemplate.boundListOps(RedisConst.SECKILL_STOCK_PREFIX +
seckillGoods.getSkuId()).leftPush(seckillGoods.getSkuId().toString()
);
        }

        // 通知添加与更新状态位, 更新为开启
        redisTemplate.convertAndSend("seckillpush",
seckillGoods.getSkuId()+":1");
    }

    channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false);
}
```

说明：到目前我们就实现了商品信息导入缓存，同时更新状态位的工作

三、秒杀列表与详情

1、封装秒杀列表与详情接口

实现类

```
package com.atguigu.gmall.activity.service.impl;

/**
 * 服务实现层
 *
 * @author Administrator
 */
@Service
@Transactional
public class SeckillGoodsServiceImpl implements SeckillGoodsService {

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 查询全部
     */
    @Override
    public List<SeckillGoods> findAll() {
        List<SeckillGoods> seckillGoodsList =
        redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).values();
        return seckillGoodsList;
    }

    /**
     * 根据 ID 获取实体
     *
     * @param id
     * @return
     */
    @Override
    public SeckillGoods getSeckillGoods(Long id) {
        return (SeckillGoods)
```

```
redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).get(id.toString());
    }
}

SeckillGoodsController

package com.atguigu.gmall.activity.controller;

/**
 * controller
 */
@RestController
@RequestMapping("/api/activity/seckill")
public class SeckillGoodsController {

    @Autowired
    private SeckillGoodsService seckillGoodsService;

    @Autowired
    private UserFeignClient userFeignClient;

    @Autowired
    private ProductFeignClient productFeignClient;

    /**
     * 返回全部列表
     */
    @GetMapping("/findAll")
    public Result findAll() {
        return Result.ok(seckillGoodsService.findAll());
    }

    /**
     * 获取实体
     */
    @GetMapping("/getSeckillGoods/{skuId}")
    public Result getSeckillGoods(@PathVariable("skuId") Long skuId)
    {
        return Result.ok(seckillGoodsService.getSeckillGoods(skuId));
    }
}
```

2、页面渲染

2.1、列表页

在 web-all 项目中添加控制器

```
package com.atguigu.gmall.item.controller;

/**
 * 秒杀
 *
 */
@Controller
public class SeckillController {

    @Autowired
    private ActivityFeignClient activityFeignClient;

    /**
     * 秒杀列表
     * @param model
     * @return
     */
    @GetMapping("seckill.html")
    public String index(Model model) {
        Result result = activityFeignClient.findAll();
        model.addAttribute("list", result.getData());
        return "seckill/index";
    }
}
```

列表

页面资源： \templates\seckill\index.html

```
<div class="goods-list" id="item">
    <ul class="seckill" id="seckill">
        <li class="seckill-item" th:each="item: ${list}">
            <div class="pic" th:@click="|detail(${item.skuId})|">
                
            </div>
            <div class="intro">
                <span th:text="${item.skuName}">手机</span>
            </div>
        </li>
    </ul>
</div>
```



```

        <div class='price'>
            <b class='sec-price' th:text="' ¥ ' + ${item.costPrice}">
                ¥0</b>
            <b class='ever-price' th:text="' ¥ ' + ${item.price}">
                ¥0</b>
        </div>
        <div class='num'>
            <div th:text="'已售' + ${item.num}">已售 1</div>
            <div class='progress'>
                <div class='sui-progress progress-danger'>
                    <span style='width: 70%;' class='bar'></span>
                </div>
            </div>
            <div>剩余
            <b class='owned' th:text="${item.stockCount}">0</b>件</div>
        </div>
        <a class='sui-btn btn-block btn-buy'
            th:href="'/seckill/' + ${item.skuId} + '.html'" target='_blank'>立即抢购
        </a>
    </li>
</ul>
</div>

```

2.2、详情页

说明：

- 1, 为了减轻访问压力，秒杀详情我们可以生成静态页面
- 2, 立即购买，该按钮我们要加以控制，该按钮就是一个链接，页面只是控制能不能点击，一般用户可以绕过去，直接点击秒杀下单，所以我们要加以控制，在秒杀没有开始前，不能进入秒杀页面

2.2.1、详情页

在 SeckillController 添加方法

```

/**
 * 秒杀详情
 * @param skuId
 * @param model
 * @return
 */

```

```
@GetMapping("seckill/{skuId}.html")
public String getItem(@PathVariable Long skuId, Model model){
    // 通过 skuId 查询 skuInfo
    Result result = activityFeignClient.getSeckillGoods(skuId);
    model.addAttribute("item", result.getData());
    return "seckill/item";
}
```

详情页面

页面资源： \templates\seckill\item.html

基本信息渲染

```
<div class="product-info">
    <div class="f1 preview-wrap">
        <!-- 放大镜效果 -->
        <div class="zoom">
            <!-- 默认第一个预览 -->
            <div id="preview" class="spec-preview">
                <span class="jqzoom"></span>
            </div>
        </div>
    </div>
    <div class="fr itemInfo-wrap">
        <div class="sku-name">
            <h4 th:text="${item.skuName}">三星</h4>
        </div>
        <div class="news">
            <span>品优秒杀</span>
            <span class="overtime">{{timeTitle}}: {{timeString}}</span>
        </div>
        <div class="summary">
            <div class="summary-wrap">
                <div class="f1 title">
                    <i>秒杀价</i>
                </div>
                <div class="f1 price">
                    <i>¥</i>
                    <em th:text="${item.costPrice}">0</em>
                    <span th:text="'原价: '+${item.price}">原价: 0</span>
                </div>
                <div class="fr remark">

```

```

        剩余库存: <span th:text="${item.stockCount}">0</span>
    </div>
</div>
<div class="summary-wrap">
    <div class="fl title">
        <i>促    销</i>
    </div>
    <div class="fl fix-width">
        <i class="red-bg">加价购</i>
        <em class="t-gray">满 999.00 另加 20.00 元, 或满 1999.00
另加 30.00 元, 或满 2999.00 另加 40.00 元, 即可在购物车换购热销商品</em>
    </div>
</div>
</div>
<div class="support">
    <div class="summary-wrap">
        <div class="fl title">
            <i>支    持</i>
        </div>
        <div class="fl fix-width">
            <em class="t-gray">以旧换新, 闲置手机回收 4G 套餐超值抢 礼
品购</em>
        </div>
    </div>
</div>
<div class="summary-wrap">
    <div class="fl title">
        <i>配 送 至</i>
    </div>
    <div class="fl fix-width">
        <em class="t-gray">满 999.00 另加 20.00 元, 或满 1999.00
另加 30.00 元, 或满 2999.00 另加 40.00 元, 即可在购物车换购热销商品</em>
    </div>
</div>
</div>
<div class="clearfix choose">

    <div class="summary-wrap">
        <div class="fl title">

        </div>
        <div class="fl">
            <ul class="btn-choose unstyled">
                <li>
                    <a href="javascript:" v-if="isBuy"
@click="queue()" class="sui-btn btn-danger addshopcar">立即抢购</a>
                    <a href="javascript:" v-if="!isBuy" class="sui-
btn btn-danger addshopcar" disabled="disabled">立即抢购</a>
                </li>
            </ul>
        </div>
    </div>
</div>

```

```
</div>
</div>
</div>
```

倒计时处理

思路：页面初始化时，拿到商品秒杀开始时间和结束时间等信息，实现距离开始时间和活动倒计时。

活动未开始时，显示距离开始时间倒计时；

活动开始后，显示活动结束时间倒计时。

倒计时代码片段

```
init() {
  // debugger
  // 计算出剩余时间
  var startTime = new Date(this.data.startTime).getTime();
  var endTime = new Date(this.data.endTime).getTime();
  var nowTime = new Date().getTime();

  var secondes = 0;
  // 还未开始抢购
  if(startTime > nowTime) {
    this.timeTitle = '距离开始'
    secondes = Math.floor((startTime - nowTime) / 1000);
  }
  if(nowTime > startTime && nowTime < endTime) {
    this.isBuy = true
    this.timeTitle = '距离结束'
    secondes = Math.floor((endTime - nowTime) / 1000);
  }
  if(nowTime > endTime) {
    this.timeTitle = '抢购结束'
    secondes = 0;
  }

  const timer = setInterval(() => {
    secondes = secondes - 1
    this.timeString = this.convertTimeString(secondes)
  }, 1000);
  // 通过$once 来监听定时器，在beforeDestroy 可以被清除。
  this.$once('hook:beforeDestroy', () => {
    clearInterval(timer);
  })
}
```

```
},
```

时间转换方法

```
convertTimeString(allseconds) {  
    if(allseconds <= 0) return '00:00:00'  
    // 计算天数  
    var days = Math.floor(allseconds / (60 * 60 * 24));  
    // 小时  
    var hours = Math.floor((allseconds - (days * 60 * 60 * 24)) / (60 * 60));  
    // 分钟  
    var minutes = Math.floor((allseconds - (days * 60 * 60 * 24) - (hours * 60 * 60)) / 60);  
    // 秒  
    var seconds = allseconds - (days * 60 * 60 * 24) - (hours * 60 * 60) - (minutes * 60);  
  
    // 拼接时间  
    var timString = "";  
    if (days > 0) {  
        timString = days + "天:";  
    }  
    return timString += hours + ":" + minutes + ":" + seconds;  
}
```

2.2.2、秒杀按钮控制

- 1, 我们通过前面页面时间控制
- 2, 通过服务器端控制, 如何控制呢?

在进入秒杀功能前, 我们加一个下单码, 只有你获取到该下单码, 才能够进入秒杀方法进行秒杀

获取秒杀码

```
SeckillGoodsController  
  
/**  
 * 获取下单码  
 * @param skuId  
 * @return  
 */  
@GetMapping("auth/getSeckillSkuIdStr/{skuId}")  
public Result getSeckillSkuIdStr(@PathVariable("skuId") Long skuId,
```

```
HttpServletRequest request) {
    String userId = AuthContextHolder.getUserId(request);
    SeckillGoods seckillGoods =
    seckillGoodsService.getSeckillGoods(skuId);
    if (null != seckillGoods) {
        Date curTime = new Date();
        if (DateUtil.dateCompare(seckillGoods.getStartTime(),
            curTime) && DateUtil.dateCompare(curTime,
            seckillGoods.getEndTime())) {
            // 可以动态生成, 放在 redis 缓存
            String skuIdStr = MD5.encrypt(userId);
            return Result.ok(skuIdStr);
        }
    }
    return Result.fail().message("获取下单码失败");
}
```

说明：只有在商品秒杀时间范围内，才能获取下单码，这样我们就有效控制了用户非法秒杀，下单码我们可以根据业务自定义规则，目前我们定义为当前用户 id MD5 加密。

前端页面

页面获取下单码，进入秒杀场景

```
queue() {
    debugger
    seckill.getSeckillSkuIdStr(this.skuId).then(response => {
        var skuIdStr = response.data.data
        window.location.href =
        '/seckill/queue.html?skuId='+this.skuId+'&skuIdStr='+skuIdStr
    })
},
```

前端 js 完整代码如下

```
<script src="/js/api/seckill.js"></script>
<script th:inline="javascript">
    var item = new Vue({
        el: '#item',

        data: {
            skuId: [[${item.skuId}]],
            data: [[${item}]],
        }
    })
```

```

        timeTitle: '距离开始',
        timeString: '00:00:00',
        isBuy: false
    },

    created() {
        this.init()
    },

    methods: {
        init() {
            // debugger
            // 计算出剩余时间
            var startTime = new Date(this.data.startTime).getTime();
            var endTime = new Date(this.data.endTime).getTime();
            var nowTime = new Date().getTime();

            var secondes = 0;
            // 还未开始抢购
            if(startTime > nowTime) {
                this.timeTitle = '距离开始'
                secondes = Math.floor((startTime - nowTime) / 1000);
            }
            if(nowTime > startTime && nowTime < endTime) {
                this.isBuy = true
                this.timeTitle = '距离结束'
                secondes = Math.floor((endTime - nowTime) / 1000);
            }
            if(nowTime > endTime) {
                this.timeTitle = '抢购结束'
                secondes = 0;
            }

            const timer = setInterval(() => {
                secondes = secondes - 1
                this.timeString = this.convertTimeString(secondes)
            }, 1000);
            // 通过$once 来监听定时器，在 beforeDestroy 钩子可以被清除。
            this.$once('hook:beforeDestroy', () => {
                clearInterval(timer);
            })
        },

        queue() {
            debugger
            seckill.getSeckillSkuIdStr(this.skuId).then(response
=> {
                var skuIdStr = response.data.data
                window.location.href =
                '/seckill/queue.html?skuId='+this.skuId+'&skuIdStr='+skuIdStr
            })
        },
    },

```

```
        convertTimeString(allseconds) {
            if(allseconds <= 0) return '00:00:00'
            // 计算天数
            var days = Math.floor(allseconds / (60 * 60 * 24));
            // 小时
            var hours = Math.floor((allseconds - (days * 60 * 60 * 24)) /
(60 * 60));
            // 分钟
            var minutes = Math.floor((allseconds - (days * 60 * 60 * 24)
- (hours * 60 * 60)) / 60);
            // 秒
            var seconds = allseconds - (days * 60 * 60 * 24) - (hours
* 60 * 60) - (minutes * 60);

            //拼接时间
            var timString = "";
            if (days > 0) {
                timString = days + "天:";
            }
            return timString += hours + ":" + minutes + ":" +
seconds;
        }
    })
</script>
```

3.3、进入秒杀

```
SeckillController

/**
 * 秒杀排队
 * @param skuId
 * @param skuIdStr
 * @param request
 * @return
 */
@GetMapping("seckill/queue.html")
public String queue(@RequestParam(name = "skuId") Long skuId,
                    @RequestParam(name = "skuIdStr") String skuIdStr,
                    HttpServletRequest request){
    request.setAttribute("skuId", skuId);
    request.setAttribute("skuIdStr", skuIdStr);
    return "seckill/queue";
}
```


页面资源: \templates\seckill\queue.html

Js 部分

```
<script src="/js/api/seckill.js"></script>
<script th:inline="javascript">
    var item = new Vue({
        el: '#item',

        data: {
            skuId: [[${skuId}]],
            skuIdStr: [[${skuIdStr}]],
            data: {},
            show: 1,
            code: 211,
            message: '',
            isCheckedOrder: false
        },

        mounted() {
            const timer = setInterval(() => {
                if(this.code != 211) {
                    clearInterval(timer);
                }
                this.checkOrder()
            }, 3000);
        }
    });
</script>
```

```
// 通过$once 来监听定时器，在 beforeDestroy 钩子可以被清除。
this.$once('hook:beforeDestroy', () => {
  clearInterval(timer);
})
},

created() {
  this.saveOrder();
},

methods: {
  saveOrder() {
    seckill.seckillOrder(this.skuId,
this.skuIdStr).then(response => {
    debugger
    console.log(JSON.stringify(response))
    if(response.data.code == 200) {
      this.isCheckOrder = true
    } else {
      this.show = 2
      this.message = response.data.message
    }
  })
},

  checkOrder() {
    if(!this.isCheckOrder) return

    seckill.checkOrder(this.skuId).then(response => {
      debugger
      this.data = response.data.data
      this.code = response.data.code
      console.log(JSON.stringify(this.data))
      //排队中
      if(response.data.code == 211) {
        this.show = 1
      } else {
        //秒杀成功
        if(response.data.code == 215) {
          this.show = 3
          this.message = response.data.message
        } else {
          if(response.data.code == 218) {
            this.show = 4
            this.message = response.data.message
          } else {
            this.show = 2
            this.message = response.data.message
          }
        }
      }
    })
  }
}
```

```
        })
    }
})
</script>
```

说明：该页面直接通过 controller 返回页面，进入页面后显示排队中，然后通过异步执行秒杀下单，提交成功，页面通过轮询后台方法查询秒杀状态

四、秒杀业务

秒杀的主要目的就是获取一个下单资格，拥有下单资格就可以去下单支付，获取下单资格后的流程就与正常下单流程一样，只是没有购物车这一步，总结起来就是，秒杀根据库存获取下单资格，拥有下单资格进入下单页面（选择地址，支付方式，提交订单，然后支付订单）

步骤：

- 1，校验下单码，只有正确获得下单码的请求才是合法请求

- 2，校验状态位 state，

State 为 null，说明请求非法；

State 为 0 说明已经售罄；

State 为 1，说明可以抢购

状态位的好处，他是在内存中判断，效率极高，如果售罄，直接就返回了，不会给服务器造成太大压力

- 3，前面条件都成立，将秒杀用户加入队列，然后直接返回

- 4，前端轮询秒杀状态，查询秒杀结果

1、秒杀下单

SeckillGoodsController 添加方法

```
/**
 * 根据用户和商品 ID 实现秒杀下单
 *
 * @param skuId
 * @return
 */
@PostMapping("auth/seckillOrder/{skuId}")
public Result seckillOrder(@PathVariable("skuId") Long skuId,
    HttpServletRequest request) throws Exception {
    // 校验下单码 (抢购码规则可以自定义)
    String userId = AuthContextHolder.getUserId(request);
    String skuIdStr = request.getParameter("skuIdStr");
    if (!skuIdStr.equals(MD5.encrypt(userId))) {
        // 请求不合法
        return Result.build(null, ResultCodeEnum.SECKILL_ILLEGAL);
    }
    // 产品标识, 1: 可以秒杀 0: 秒杀结束
    String state = (String) CacheHelper.get(skuId.toString());
    if (StringUtils.isEmpty(state)) {
        // 请求不合法
        return Result.build(null, ResultCodeEnum.SECKILL_ILLEGAL);
    }
    if ("1".equals(state)) {
        // 用户记录
        UserRecode userRecode = new UserRecode();
        userRecode.setUserId(userId);
        userRecode.setSkuId(skuId);

        rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_SECKILL_USER,
            MqConst.ROUTING_SECKILL_USER, userRecode);
    } else {
        // 已售罄
        return Result.build(null, ResultCodeEnum.SECKILL_FINISH);
    }
    return Result.ok();
}
```

2、秒杀下单监听

思路:

1, 首先判断产品状态位, 我们前面不是已经判断过了吗? 因为产品可能随时售罄, mq 队列里面可能堆积了十万数据, 但是已经售罄了, 那么后续流程就没有必要再走了;

2, 判断用户是否已经下过订单, 这个地方就是控制用户重复下单, 同一个用户只能抢购一个下单资格, 怎么控制呢? 很简单, 我们可以利用 setnx 控制用户, 当用户第一次进来时, 返回 true, 可以抢购, 以后进入返回 false, 直接返回, 过期时间可以根据业务自定义, 这样用户这一段就控制住了

3, 获取队列中的商品, 如果能够获取, 则商品有库存, 可以下单, 如果获取的商品 id 为空, 则商品售罄, 商品售罄我们要第一时间通知兄弟节点, 更新状态位, 所以在这里发送 redis 广播

4, 将订单记录放入 redis 缓存, 说明用户已经获得下单资格, 秒杀成功

SeckillReceiver 类添加监听方法

```
@Autowired
private SeckillGoodsService seckillGoodsService;

/**
 * 秒杀用户加入队列
 *
 * @param message
 * @param channel
 * @throws IOException
 */
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(value = MqConst.QUEUE_SECKILL_USER, durable =
"true"),
    exchange = @Exchange(value =
MqConst.EXCHANGE_DIRECT_SECKILL_USER, type = ExchangeTypes.DIRECT,
durable = "true"),
    key = {MqConst.ROUTING_SECKILL_USER}
))
public void seckill(UserRecode userRecode, Message message, Channel
channel) throws IOException {
    if (null != userRecode) {
        //Log.info("paySuccess:" +
JSONObject.toJSONString(userRecode));
        //预下单
        seckillGoodsService.seckillOrder(userRecode.getSkuId(),
userRecode.getUserId());
        //确认收到消息

channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false);
    }
}
```

预下单接口

实现类

```
/**
 * 创建订单
 * @param skuId
 * @param userId
 */
@Override
public void seckillOrder(Long skuId, String userId) {
    // 产品状态位, 1: 可以秒杀 0: 秒杀结束
    String state = (String) CacheHelper.get(skuId.toString());
    if("0".equals(state)) {
        // 已售罄
        return;
    }
    // 判断用户是否下过单
    boolean isExist =
    redisTemplate.opsForValue().setIfAbsent(RedisConst.SECKILL_USER +
    userId, skuId, RedisConst.SECKILL__TIMEOUT, TimeUnit.SECONDS);
    if (!isExist) {
        return;
    }

    // 获取队列中的商品, 如果能够获取, 则商品存在, 可以下单
    String goodsId = (String)
    redisTemplate.boundListOps(RedisConst.SECKILL_STOCK_PREFIX +
    skuId).rightPop();
    if (StringUtils.isEmpty(goodsId)) {
        // 商品售罄, 更新状态位
        redisTemplate.convertAndSend("seckillpush", skuId+"0");
        // 已售罄
        return;
    }
    // 订单记录
    OrderRecode orderRecode = new OrderRecode();
    orderRecode.setUserId(userId);
    orderRecode.setSeckillGoods(this.getSeckillGoods(skuId));
    orderRecode.setNum(1);
    // 生成下单码
    orderRecode.setOrderStr(MD5.encrypt(userId+skuId));
    // 订单数据存入 Redis
    redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).put(orderRecode
    .getUserId(), orderRecode);

    // 更新库存
```

```
this.updateStockCount(orderRecode.getSeckillGoods().getSkuId());
}

package com.atguigu.gmall.model.activity;
@Data
public class OrderRecode implements Serializable {

    private static final long serialVersionUID = 1L;

    private String userId;

    private SeckillGoods seckillGoods;

    private Integer num;

    private String orderStr;
}

/**
 * 更新库存
 * @param skuId
 */
private void updateStockCount(Long skuId) {
    //更新库存，批量更新，用于页面显示，以实际扣减库存为准
    Long stockCount =
    redisTemplate.boundListOps(RedisConst.SECKILL_STOCK_PREFIX +
    skuId).size();
    if (stockCount % 2 == 0) {
        //商品卖完，同步数据库
        SeckillGoods seckillGoods = this.getSeckillGoods(skuId);
        seckillGoods.setStockCount(stockCount.intValue());
        seckillGoodsMapper.updateById(seckillGoods);
        //更新缓存
        redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGoods
        .getSkuId().toString(), seckillGoods);
    }
}
```

3、页面轮询接口

该接口判断用户秒杀状态

SeckillGoodsService 接口

```
/**
 * 根据用户 ID 查看订单信息
 * @param userId
 * @return
 */
@Override
public Result checkOrder(Long skuId, String userId) {
    // 用户在缓存中存在, 有机会秒杀到商品
    boolean isExist = redisTemplate.hasKey(RedisConst.SECKILL_USER + userId);
    if (isExist) {
        // 判断用户是否正在排队
        // 判断用户是否下单
        boolean isHasKey = redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).hasKey(userId);
        if (isHasKey) {
            // 抢单成功
            OrderRecode orderRecode = (OrderRecode) redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).get(userId);
            // 秒杀成功!
            return Result.build(orderRecode, ResultCodeEnum.SECKILL_SUCCESS);
        }

        // 判断是否下单
        boolean isExistOrder = redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS_USERS).hasKey(userId);
        if (isExistOrder) {
            String orderId = (String) redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS_USERS).get(userId);
            return Result.build(orderId, ResultCodeEnum.SECKILL_ORDER_SUCCESS);
        }

        String state = (String) CacheHelper.get(skuId.toString());
        if ("0".equals(state)) {
            // 已售罄 抢单失败
            return Result.build(null, ResultCodeEnum.SECKILL_FAIL);
        }
        // 正在排队中
        return Result.build(null, ResultCodeEnum.SECKILL_RUN);
    }
}
```

SeckillGoodsController

```
/**
 * 查询秒杀状态
 * @return
 */
@GetMapping(value = "auth/checkOrder/{skuId}")
public Result checkOrder(@PathVariable("skuId") Long skuId,
    HttpServletRequest request) {
```


}

该页面有四种状态：

- 抢购成功，页面显示去下单，跳转下单确认页面

</div>

5、下单页面

填写并核对订单信息

收件人信息

张三

北京市昌平区宏福科技园综合楼6层 15010658793

默认地址

李四

北京市昌平区宏福科技园综合楼5层 13590909098

王五

北京市昌平区宏福科技园综合楼3层 18012340987

支付方式

在线支付

货到付款

送货清单

配送方式:

天天快递

配送时间: 预计8月10日 (周二) 09:00-15:00送达

商品清单:



Apple iPhone 6s (A1700) 64G 玫瑰金色 移动联通电信4G手机硅胶透明防摔软壳 本色系列

¥5399.00

X1

有货

7天无理由退货



Apple iPhone 6s (A1700) 64G 玫瑰金色 移动联通电信4G手机硅胶透明防摔软壳 本色系列

¥5399.00

X1

有货

7天无理由退货

买家留言:

建议留言前先与商家沟通确认

我们已经把下单信息记录到 redis 缓存中，所以接下来我们要组装下单页数据

5.1、下单页数据接口封装

Service-activity 模块

```
SeckillGoodsController
@Autowired
private RedisTemplate redisTemplate;

/**
 * 秒杀确认订单
 * @param request
 * @return
 */
@GetMapping("auth/trade")
public Result trade(HttpServletRequest request) {
    // 获取到用户Id
    String userId = AuthContextHolder.getUserId(request);

    // 先得到用户想要购买的商品！
}
```

```
OrderRecode orderRecode = (OrderRecode)
redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).get(userId);
if (null == orderRecode) {
    return Result.fail().message("非法操作");
}
SeckillGoods seckillGoods = orderRecode.getSeckillGoods();

// 获取用户地址
List<UserAddress> userAddressList =
userFeignClient.findUserAddressListByUserId(userId);

// 声明一个集合来存储订单明细
ArrayList<OrderDetail> detailArrayList = new ArrayList<>();
OrderDetail orderDetail = new OrderDetail();
orderDetail.setSkuId(seckillGoods.getSkuId());
orderDetail.setSkuName(seckillGoods.getSkuName());
orderDetail.setImgUrl(seckillGoods.getSkuDefaultImg());
orderDetail.setSkuNum(orderRecode.getNum());
orderDetail.setOrderPrice(seckillGoods.getCostPrice());
// 添加到集合
detailArrayList.add(orderDetail);

// 计算总金额
OrderInfo orderInfo = new OrderInfo();
orderInfo.setOrderDetailList(detailArrayList);
orderInfo.sumTotalAmount();

Map<String, Object> result = new HashMap<>();
result.put("userAddressList", userAddressList);
result.put("detailArrayList", detailArrayList);
// 保存总金额
result.put("totalAmount", orderInfo.getTotalAmount());
return Result.ok(result);
}
```

5.2、web-all 调用接口

SeckillController

```
/**
 * 确认订单
 * @param model
 * @return
 */
```

```
@GetMapping("seckill/trade.html")
public String trade(Model model) {
    Result<Map<String, Object>> result =
    activityFeignClient.trade();
    if(result.isOk()) {
        model.addAllAttributes(result.getData());
        return "seckill/trade";
    } else {
        model.addAttribute("message",result.getMessage());
        return "seckill/fail";
    }
}
```

页面资源： \templates\seckill\trade.html; \templates\seckill\fail.html

5.2、下单确认页面

该页面与正常下单页面类似，只是下单提交接口不一样，因为秒杀下单不需要正常下单的各种判断，因此我们要在订单服务提供一个秒杀下单接口，直接下单

Service-order 模块提供秒杀下单接口

```
OrderApiController

/**
 * 秒杀提交订单，秒杀订单不需要做前置判断，直接下单
 * @param orderInfo
 * @return
 */
@PostMapping("inner/seckill/submitOrder")
public Long submitOrder(@RequestBody OrderInfo orderInfo) {
    Long orderId = orderService.saveOrderInfo(orderInfo);
    return orderId;
}
```

Service-activity 模块秒杀下单

```
SeckillGoodsController

@Autowired
private OrderFeignClient orderFeignClient;

/**
 * 秒杀提交订单
 *
 * @param orderInfo
 * @return
 */
@PostMapping("auth/submitOrder")
```

```
public Result submitOrder(@RequestBody OrderInfo orderInfo,
    HttpServletRequest request) {
    String userId = AuthContextHolder.getUserId(request);

    OrderRecode orderRecode = (OrderRecode)
    redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).get(userId);
    if (null == orderRecode) {
        return Result.fail().message("非法操作");
    }

    orderInfo.setUserId(Long.parseLong(userId));

    Long orderId = orderFeignClient.submitOrder(orderInfo);
    if (null == orderId) {
        return Result.fail().message("下单失败，请重新操作");
    }
    // 删除下单信息

    redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).delete(userId);
    ;
    // 下单记录

    redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS_USERS).put(userId, orderId.toString());

    return Result.ok(orderId);
}
```

说明：下单成功后，后续流程与正常订单一致

6、秒杀结束清空 redis 缓存

秒杀过程中我们写入了大量 redis 缓存，我们可以在秒杀结束或每天固定时间清楚缓存，释放缓存空间；

实现思路：假如根据业务，我们确定每天 18 点所有秒杀业务结束，那么我们编写定时任务，每天 18 点发送 mq 消息，service-activity 模块监听消息清理缓存

Service-task 发送消息

6.1、编写定时任务发送消息

```
/**
 * 每天下午 18 点执行
```

```
*/
//@Scheduled(cron = "0/35 * * * * ?")
@Scheduled(cron = "0 0 18 * * ?")
public void task18() {
    Log.info("task18");
    rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_TASK,
MqConst.ROUTING_TASK_18, "");
}
```

6.3、接收消息并处理

Service-activity 接收消息

```
SeckillReceiver

/**
 * 秒杀结束清空缓存
 *
 * @param message
 * @param channel
 * @throws IOException
 */
@Resource
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(value = MqConst.QUEUE_TASK_18, durable = "true"),
    exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_TASK, type =
ExchangeTypes.DIRECT, durable = "true"),
    key = {MqConst.ROUTING_TASK_18}
))
public void clearRedis(Message message, Channel channel) throws IOException
{
    // 活动结束清空缓存
    QueryWrapper<SeckillGoods> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("status", 1);
    queryWrapper.le("end_time", new Date());
    List<SeckillGoods> list = seckillGoodsMapper.selectList(queryWrapper);
    // 清空缓存
    for (SeckillGoods seckillGoods : list) {
        redisTemplate.delete(RedisConst.SECKILL_STOCK_PREFIX +
seckillGoods.getSkuId());
    }
    redisTemplate.delete(RedisConst.SECKILL_GOODS);
    redisTemplate.delete(RedisConst.SECKILL_ORDERS);
    redisTemplate.delete(RedisConst.SECKILL_ORDERS_USERS);
    // 将状态更新为结束
    SeckillGoods seckillGoodsUp = new SeckillGoods();
    seckillGoodsUp.setStatus("2");
    seckillGoodsMapper.update(seckillGoodsUp, queryWrapper);
    // 手动确认
    channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);
}
```

说明：清空 redis 缓存，同时更改秒杀商品活动结束

