

# 尚品汇商城

## 一、秒杀业务分析

### 1.1 需求分析

所谓“秒杀”，就是网络卖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。通俗一点讲就是网络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”。商家赔本赚吆喝，图啥？人气！

(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： **B2B2C**

(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细页。

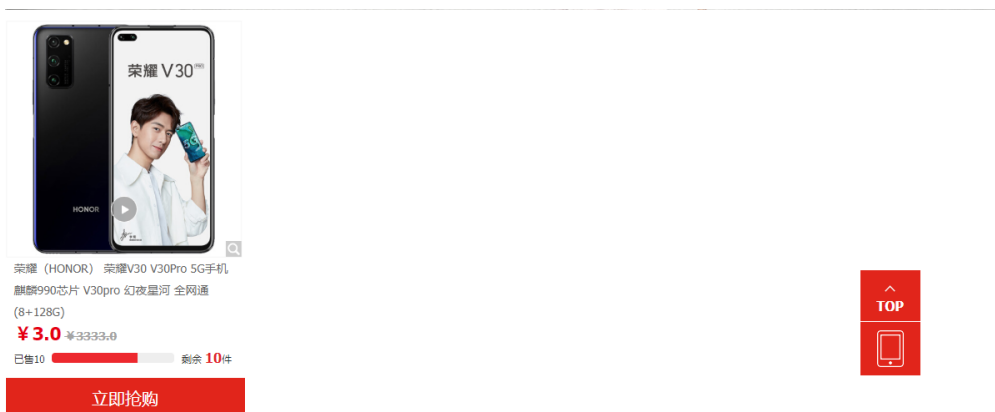
(4) 商品详细页显示秒杀商品信息，点击立即抢购进入秒杀，抢购成功时预减库存。当库存为 0 或不在活动期范围内时无法秒杀。

(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，调用微信支付或支付宝的关闭订单接口。

## 1.2 秒杀功能分析

### 列表页



### 详情页



### 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



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:

建议留言前先与商家沟通确认

## 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](#)



搜索

✓ 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



## 1.3 数据库表

秒杀商品表 seckill\_goods

名称	类型	空	默认值	其他	备注
<b>索引 (1)</b>					
 <b>主索引</b>	id			unique	
<b>字段 (15)</b>					
 <b>id</b>	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

## 1.4 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。

(2) 状态位控制访问请求，何为状态位？就是我们在内存中保存一个状态，当抢购开始时状态为 1，可以抢购，当库存为 0 时，状态位 0，不能抢购；状态位的好处，他是在内存中判断，压力很小，可以阻止很多不必要的请求

(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？

如何控制库存超买？

如何控制访问压力？

这些问题，我们都在后续陆续讲到

## 二、搭建秒杀模块

我们先把秒杀模块搭建好，秒杀一共有三个模块，秒杀微服务模块 service-activity，负责封装秒杀全部服务端业务；秒杀前端模块 web-all，负责前端显示业务；service-activity-client api 接口模块

### 2.1 搭建 service-activity 模块

#### 2.1.1 搭建 service-activity

搭建方式如 service-order

#### 2.1.2 修改 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service</artifactId>
    <version>1.0</version>
  </parent>
```

```
<version>1.0</version>
<artifactId>service-activity</artifactId>
<packaging>jar</packaging>
<name>service-activity</name>
<description>service-activity</description>

<dependencies>
  <dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-user-client</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-product-client</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-order-client</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>rabbit-util</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>

<build>
  <finalName>service-activity</finalName>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

### 2.1.3 添加配置

bootstrap.properties

```
spring.application.name=service-activity
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

## 2.1.4 启动类

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

@SpringBootApplication
@ComponentScan({"com.atguigu.gmall"})
@EnableDiscoveryClient
@EnableFeignClients(basePackages= {"com.atguigu.gmall"})
public class ServiceActivityApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceActivityApplication.class, args);
    }

}
```

## 2.2 搭建 service-activity-client 模块

### 2.2.1 搭建 service-activity-client

搭建方式如 service-order-client

### 2.2.2 修改 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```



```
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-client</artifactId>
    <version>1.0</version>
  </parent>

  <artifactId>service-activity-client</artifactId>
  <version>1.0</version>

  <packaging>jar</packaging>
  <name>service-activity-client</name>
  <description>service-activity-client</description>

</project>
```

## 2.3 添加依赖，配置网关

### 2.3.1 在 web-all 中引入依赖

```
<dependency>
  <groupId>com.atguigu.gmall</groupId>
  <artifactId>service-activity-client</artifactId>
  <version>1.0</version>
</dependency>
```

### 2.3.2 在网关项目中配置秒杀服务，域名

```
- id: web-activity
  uri: lb://web-all
  predicates:
    - Host=activity.gmall.com

- id: service-activity
  uri: lb://service-activity
  predicates:
    - Path=/*/activity/** # 路径匹配
```

## 三、秒杀商品导入缓存

缓存数据实现思路：service-task 模块统一管理我们的定时任务，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{list}，**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，只有队列里面有数据，说明就一点有库存（redis 队列保证了原子性），队列为空了说明商品售罄**

### 3.1 编写定时任务

在 service-task 模块发送消息

#### 3.1.1 搭建 service-task 服务

搭建方式如 service-mq

#### 3.1.2 修改配置 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>com.atguigu.gmall</groupId>
  <artifactId>service</artifactId>
  <version>1.0</version>
</parent>

<artifactId>service-task</artifactId>
<version>1.0</version>

<packaging>jar</packaging>
<name>service-task</name>
<description>service-task</description>

<dependencies>
  <!--rabbitmq 消息队列-->
  <dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>rabbit-util</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>

<build>
  <finalName>service-task</finalName>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

说明：引入依赖

### 3.1.3 添加配置文件以及启动类

bootstrap.properties

```
spring.application.name=service-task
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
```

```
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

启动类

```
package com.atguigu.gmall.task;

@SpringBootApplication(exclude =
DataSourceAutoConfiguration.class)//取消数据源自动配置
@ComponentScan({"com.atguigu.gmall"})
@EnableDiscoveryClient
public class ServiceTaskApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceTaskApplication.class, args);
    }

}
```

### 3.1.4 添加定时任务

定义凌晨一点 mq 相关常量

```
/**
 * 定时任务
 */
public static final String EXCHANGE_DIRECT_TASK =
"exchange.direct.task";
public static final String ROUTING_TASK_1 = "seckill.task.1";
//队列
public static final String QUEUE_TASK_1 = "queue.task.1";
```

```
package com.atguigu.gmall.task.scheduled;
```

```
@Component
@EnableScheduling
@Slf4j
public class ScheduledTask {
```

```
@Autowired
private RabbitService rabbitService;
/**
 * 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, "");
}
}
```

### cron 表达式各占位符解释:

**{秒数} {分钟} {小时} {日期} {月份} {星期} {年份(可为空)}**

{秒数}{分钟} ==> 允许值范围: 0~59 ,不允许为空值, 若值不合法, 调度器将抛出 SchedulerException 异常

"\*" 代表每隔 1 秒钟触发;

"," 代表在指定的秒数触发, 比如" 0,15,45" 代表 0 秒、15 秒和 45 秒时触发任务

"-" 代表在指定的范围内触发, 比如" 25-45" 代表从 25 秒开始触发到 45 秒结束触发, 每隔 1 秒触发 1 次

"/" 代表触发步进(step), "/" 前面的值代表初始值( " "等同" 0" ), 后面的值代表偏移量, 比如" 0/20" 或者" /20" 代表从 0 秒钟开始, 每隔 20 秒钟触发 1 次, 即 0 秒触发 1 次, 20 秒触发 1 次, 40 秒触发 1 次; " 5/20" 代表 5 秒触发 1 次, 25 秒触发 1 次, 45 秒触发 1 次; " 10-45/20" 代表在[10,45]内步进 20 秒命中的时间点触发, 即 10 秒触发 1 次, 30 秒触发 1 次

{小时} ==> 允许值范围: 0~23 ,不允许为空值, 若值不合法, 调度器将抛出 SchedulerException 异常,占位符和秒数一样

{日期} ==> 允许值范围: 1~31 ,不允许为空值, 若值不合法, 调度器将抛出 SchedulerException 异常

{星期} ==> 允许值范围: 1~7 (SUN-SAT),1 代表星期天(一星期的第一天), 以此类推, 7 代表星期六(一星期的最后一天), 不允许为空值, 若值不合法, 调度器将抛出 SchedulerException 异常

{年份} ==> 允许值范围: 1970~2099 ,允许为空, 若值不合法, 调度器将抛出

## SchedulerException 异常

### 注意：日期和星期的问题

"?"与{日期}互斥，即意味着若明确指定{日期}触发，则表示{星期}无意义，以免引起冲突和混乱

常用实例：

"30 \* \* \* \* ?" 每半分钟触发任务

"30 10 \* \* \* ?" 每小时的 10 分 30 秒触发任务

"30 10 1 \* \* ?" 每天 1 点 10 分 30 秒触发任务

"30 10 1 20 \* ?" 每月 20 号 1 点 10 分 30 秒触发任务

"30 10 1 20 10 ? \*" 每年 10 月 20 号 1 点 10 分 30 秒触发任务

"30 10 1 20 10 ? 2011" 2011 年 10 月 20 号 1 点 10 分 30 秒触发任务

"30 10 1 ? 10 \* 2011" 2011 年 10 月每天 1 点 10 分 30 秒触发任务

"30 10 1 ? 10 SUN 2011" 2011 年 10 月每周日 1 点 10 分 30 秒触发任务

"15,30,45 \* \* \* \* ?" 每 15 秒，30 秒，45 秒时触发任务

"15-45 \* \* \* \* ?" 15 到 45 秒内，每秒都触发任务

"15/5 \* \* \* \* ?" 每分钟的每 15 秒开始触发，每隔 5 秒触发一次

"15-30/5 \* \* \* \* ?" 每分钟的 15 秒到 30 秒之间开始触发，每隔 5 秒触发一次

"0 0/3 \* \* \* ?" 每小时的第 0 分 0 秒开始，每三分钟触发一次

"0 15 10 ? \* MON-FRI" 星期一到星期五的 10 点 15 分 0 秒触发任务

"0 15 10 L \* ?" 每个月最后一天的 10 点 15 分 0 秒触发任务

"0 15 10 LW \* ?" 每个月最后一个工作日的 10 点 15 分 0 秒触发任务

"0 15 10 ? \* 5L" 每个月最后一个星期四的 10 点 15 分 0 秒触发任务

"0 15 10 ? \* 5#3" 每个月第三周的星期四的 10 点 15 分 0 秒触发任务

## 3.2 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

### 3.2.1 数据导入缓存

#### 3.2.1.1 在 service-util 的 RedisConst 类中定义常量

```
// 秒杀商品前缀
public static final String SECKILL_GOODS = "seckill:goods";
public static final String SECKILL_ORDERS = "seckill:orders";
public static final String SECKILL_ORDERS_USERS = "seckill:orders:users";
public static final String SECKILL_STOCK_PREFIX = "seckill:stock:";
public static final String SECKILL_USER = "seckill:user:";
// 用户锁定时间 单位：秒
public static final int SECKILL_TIMEOUT = 60 * 60;
```

#### 3.2.1.2 创建秒杀商品实体与 Mapper

```
package com.atguigu.gmall.model.activity;

@Data
@ApiModel(description = "SeckillGoods")
@TableName("seckill_goods")
public class SeckillGoods extends BaseEntity {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "spu ID")
    @TableField("spu_id")
    private Long spuId;

    @ApiModelProperty(value = "sku ID")
    @TableField("sku_id")
    private Long skuId;

    @ApiModelProperty(value = "标题")
    @TableField("sku_name")
    private String skuName;
```

```
@ApiModelProperty(value = "商品图片")
@TableField("sku_default_img")
private String skuDefaultImg;

ApiModelProperty(value = "原价格")
@TableField("price")
private BigDecimal price;

ApiModelProperty(value = "秒杀价格")
@TableField("cost_price")
private BigDecimal costPrice;

ApiModelProperty(value = "添加日期")
@TableField("create_time")
private Date createTime;

ApiModelProperty(value = "审核日期")
@TableField("check_time")
private Date checkTime;

ApiModelProperty(value = "审核状态")
@TableField("status")
private String status;

ApiModelProperty(value = "开始时间")
@TableField("start_time")
private Date startTime;

ApiModelProperty(value = "结束时间")
@TableField("end_time")
private Date endTime;

ApiModelProperty(value = "秒杀商品数")
@TableField("num")
private Integer num;

ApiModelProperty(value = "剩余库存数")
@TableField("stock_count")
private Integer stockCount;

ApiModelProperty(value = "描述")
@TableField("sku_desc")
private String skuDesc;
}

package com.atguigu.gmall.activity.mapper;

import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface SeckillGoodsMapper extends BaseMapper<SeckillGoods>
{
}
```



```
}
```

### 3.2.1.3 监听消息

导入工具包{redis,util}到 service-activity 项目中！

```
package com.atguigu.gmall.activity.receiver;

@Component
public class SeckillReceiver {
    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;

    @SneakyThrows
    @RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = MqConst.QUEUE_TASK_1,durable =
"true",autoDelete = "false"),
        exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_TASK),
        key = {MqConst.ROUTING_TASK_1}
    ))
    public void importToRedis(Message message, Channel channel){
        try {
            // 将当天的秒杀商品放入缓存! 通过mapper 执行sql 语句!
            // 条件当天, 剩余库存>0, 审核状态 = 1
            QueryWrapper<SeckillGoods> seckillGoodsQueryWrapper = new
            QueryWrapper<>();
            seckillGoodsQueryWrapper.eq("status","1").gt("stock_count",0);
            // select DATE_FORMAT(start_time,'%Y-%m-%d') from
            seckill_goods; yyyy-mm-dd
            seckillGoodsQueryWrapper.eq("DATE_FORMAT(start_time,'%Y-%m-
            %d')", DateUtil.formatDate(new Date()));
            // 获取到当天秒杀的商品列表!
            List<SeckillGoods> seckillGoodsList =
            seckillGoodsMapper.selectList(seckillGoodsQueryWrapper);

            // 将seckillGoodsList 这个集合数据放入缓存!
            for (SeckillGoods seckillGoods : seckillGoodsList) {
                // 考虑使用哪种数据类型, 以及缓存的 key! 使用 hash! hset key
                field value hget key field
                // 定义 key = SECKILL_GOODS field = skuId value =
                seckillGoods
                // 判断当前缓存key 中是否有 秒杀商品的 skuId
                Boolean flag =
                redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).hasKey(seckillGoo
                ds.getSkuId().toString());
                // 判断
```

```

        if (flag){
            // 表示缓存中已经当前的商品了。
            continue;
        }
        // 没有就放入缓存!

        redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGoods.
            getSkuId().toString(),seckillGoods);
        // 将每个商品对应的库存剩余数,放入 redis-list 集合中!
        for (Integer i = 0; i < seckillGoods.getStockCount(); i++) {
            // 放入 list key = seckill:stock:skuId;
            String key = RedisConst.SECKILL_STOCK_PREFIX+seckillGoods.getSkuId();
            redisTemplate.opsForList().leftPush(key,seckillGoods.getSkuId().toStri
                ng());
            //
            redisTemplate.boundListOps(key).leftPush(seckillGoods.getSkuId());
        }

        // 秒杀商品在初始化的时候: 状态位初始化 1
        // publish seckillpush 46:1 | 后续业务如果说商品被秒杀完
        // 了! publish seckillpush 46:0

        redisTemplate.convertAndSend("seckillpush",seckillGoods.getSkuId()+":1");
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 手动确认消息

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

```



### 3.2.2 更新状态位

由于我们的秒杀服务时集群部署 service-activity 的, 我们面临一个问题? RabbitMQ 如何实现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，我们很容易就想到一种解决方案，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

### 3.2.2.1 redis 发布与订阅实现

```
package com.atguigu.gmall.activity.redis;

@Configuration
public class RedisChannelConfig {

    /**
     * docker exec -it e222dac4e559 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

### 3.2.2.2 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

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

    redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGoods.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( key: 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( channel: "seckillpush", message: seckillGoods.getSkuId()+"1");
}

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

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

## 四、秒杀列表与详情

### 4.1 封装秒杀列表与详情接口

#### 4.1.1 封装接口

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

public interface SeckillGoodsService {

    /**
     * 返回全部列表
     * @return
     */
    List<SeckillGoods> findAll();

    /**
     * 根据 ID 获取实体
     * @param id
     * @return
     */
    SeckillGoods getSeckillGoods(Long id);
}
```

#### 4.1.2 完成实现类

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

@Service
public class SeckillGoodsServiceImpl implements SeckillGoodsService {

    @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());
}
}
```

### 4.1.3 完成控制器

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

@RestController
@RequestMapping("/api/activity/seckill")
public class SeckillGoodsApiController {

    @Autowired
    private SeckillGoodsService seckillGoodsService;

    @Autowired
    private UserFeignClient userFeignClient;

    @Autowired
    private ProductFeignClient productFeignClient;

    /**
     * 返回全部列表
     *
     * @return
     */
    @GetMapping("/findAll")
    public Result findAll() {
```

```
        return Result.ok(seckillGoodsService.findAll());
    }

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

## 4.2 在 service-activity-client 模块添加接口

```
package com.atguigu.gmall.activity.client;

@FeignClient(value = "service-activity", fallback =
ActivityDegradeFeignClient.class)
public interface ActivityFeignClient {

    /**
     * 返回全部列表
     *
     * @return
     */
    @GetMapping("/api/activity/seckill/findAll")
    Result findAll();

    /**
     * 获取实体
     *
     * @param skuId
     * @return
     */
    @GetMapping("/api/activity/seckill/getSeckillGoods/{skuId}")
    Result getSeckillGoods(@PathVariable("skuId") Long skuId);
}

package com.atguigu.gmall.cart.client.impl;
```

```
@Component
public class ActivityDegradeFeignClient implements
ActivityFeignClient {

    @Override
    public Result findAll() {
        return Result.fail();
    }

    @Override
    public Result getSeckillGoods(Long skuId) {
        return Result.fail();
    }
}
```

## 4.3 页面渲染

### 4.3.1 在 web-all 中编写控制器

在 web-all 项目中添加控制器

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

@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>
      <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 ' + ${item.num}">已售 1</div>
        <div class="progress">
          <div class="sui-progress progress-danger">
            <span style="width: 70%;" class="bar"></span>
          </div>
          <div>剩余
            <b class="owned" th:text="${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>
```

## 4.3.2 秒杀详情页面功能介绍

说明：

1，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

### 4.3.2.1 web-all 添加商品详情控制器

SeckillController

```
@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";
}
```

### 4.3.2.2 详情页面介绍

#### 4.3.2.2.1 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>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

        <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>

```

```
</li>
</ul>
</div>
</div>
</div>
</div>
</div>
```

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

  const timer = setInterval(() => {
    seconds = seconds - 1
    this.timeString = this.convertTimeString(seconds)
  }, 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);  
})  
},
```

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;  
}
```

#### 4.3.2.3 秒杀按钮控制

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

##### 4.3.2.3.1 获取下单码

SeckillGoodsApiController



```
@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。

#### 4.3.2.3.2 前端页面

页面获取下单码, 进入秒杀场景

```
queue() {
    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() {
            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>
```

### 4.3.3 编写排队控制器

SeckillController

```
@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

页面资源： \templates\seckill\queue.html

石

石

```
<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);
            // 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

## 五、整合秒杀业务

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，说明非法请求；

State 为 0 说明已经售罄；

State 为 1，说明可以抢购

状态位是在内存中判断，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

### 5.1 秒杀下单

#### 5.1.1 添加 mq 常量 MqConst 类

```
* 秒杀
*/
public static final String EXCHANGE_DIRECT_SECKILL_USER =
"exchange.direct.seckill.user";
public static final String ROUTING_SECKILL_USER = "seckill.user";
// 队列
public static final String QUEUE_SECKILL_USER =
"queue.seckill.user";
```

### 5.1.2 定义实体 UserRecode

记录哪个用户要购买哪个商品!

```
@Data
public class UserRecode implements Serializable {

    private static final long serialVersionUID = 1L;

    private Long skuId;

    private String userId;
}
```

### 5.1.3 编写控制器

SeckillGoodsApiController

```
@Autowired
private RabbitService rabbitService;

/**
 * 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();
}
```

## 5.2 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, 这样用户这一段咱们就控制住了

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
- 5, 秒杀成功要更新库存

### 5.2.1 SeckillReceiver 添加监听方法

```
@Autowired
private SeckillGoodsService seckillGoodsService;

// 监听用户与商品的消息!
@sneakyThrows
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(value = MqConst.QUEUE_SECKILL_USER,durable =
"true",autoDelete = "false"),
    exchange = @Exchange(value =
MqConst.EXCHANGE_DIRECT_SECKILL_USER),
    key = {MqConst.ROUTING_SECKILL_USER}
))
public void seckillUser(UserRecode userRecode,Message message,Channel
channel){
    try {
        // 判断接收过来的数据
        if (userRecode!=null){
            // 预下单处理!

            seckillGoodsService.seckillOrder(userRecode.getSkuId(),userRecode.getUs
erId());
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 手动确认

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

### 5.2.2 预下单接口

```
SeckillGoodsService 接口

/**
 * 根据用户和商品 ID 实现秒杀下单
```

```
* @param skuId
* @param userId
*/
void seckillOrder(Long skuId, String userId);
```

## 5.2.3 实现类

秒杀订单实体类

```
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;
}
```

```
/**
 * 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);
// 生成订单单码
orderRecode.setOrderStr(MD5.encrypt(userId+skuId));

// 订单数据存入 Reids
redisTemplate.boundHashOps(RedisConst.SECKILL_ORDERS).put(orderRecode
.getUserId(), orderRecode);

// 更新库存
this.updateStockCount(orderRecode.getSeckillGoods().getSkuId());
}
```

## 5.2.4 更新库存

```
// 表示更新mysql -- redis 的库存数据!
public void updateStockCount(Long skuId) {
    // 加锁!
    Lock lock = new ReentrantLock();
    // 上锁
    lock.lock();
    try {
        // 获取到存储库存剩余数!
        // key = seckill:stock:46
        String stockKey = RedisConst.SECKILL_STOCK_PREFIX + skuId;
        //
        redisTemplate.opsForList().leftPush(key, seckillGoods.getSkuId());
        Long count = redisTemplate.boundListOps(stockKey).size();
        // 减少库存数! 方式一减少压力!
    }
}
```

```
        if (count%2==0){
            // 开始更新数据!
            SeckillGoods seckillGoods = this.getSeckillGoods(skuId);
            // 赋值剩余库存数!
            seckillGoods.setStockCount(count.intValue());
            // 更新的数据库!
            seckillGoodsMapper.updateById(seckillGoods);
            // 更新缓存!

            redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).put(seckillGoods.getSkuId().toString(),seckillGoods);
        }
    } finally {
        // 解锁!
        lock.unlock();
    }
}
```

## 5.3 页面轮询接口

思路:

1. 判断用户是否在缓存中存在
2. 判断用户是否抢单成功
3. 判断用户是否下过订单
4. 判断状态位

### 5.3.1 接口

SeckillGoodsService 接口

```
/**
 * 根据商品 id 与用户 ID 查看订单信息
 * @param skuId
 * @param userId
 * @return
 */
Result checkOrder(Long skuId, String userId);
```

## 5.3.2 实现类

```
@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);
    }
}
```

## 5.3.3 控制器

SeckillGoodsApiController

```
@GetMapping(value = "auth/checkOrder/{skuId}")
public Result checkOrder(@PathVariable("skuId") Long skuId,
    HttpServletRequest request) {
```

}

## 5.4 轮询排队页面

该页面有四种状态：

- 1, 排队中
- 2, 各种提示 (非法、已售罄等)
- 3, 抢购成功, 去下单
- 4, 抢购成功, 已下单, 显示我的订单

抢购成功，页面显示去下单，跳转下单确认页面

[illegible]

## 5.5 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

### 5.5.1 下单页数据数据接口

SeckillGoodsApiController

```
@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);
}
```

## 5.5.2 service-activity-client 添加接口

ActivityFeignClient

```
/**
 * 秒杀确认订单
 * @return
 */
@GetMapping("/api/activity/seckill/auth/trade")
```



```
Result<Map<String, Object>> trade();
```

```
ActivityDegradeFeignClient
```

```
@Override
```

```
public Result<Map<String, Object>> trade() {  
    return Result.fail();  
}
```

### 5.5.3 web-all 编写去下单控制器

```
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

### 5.5.4 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

#### 5.5.4.1 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;
}
```

#### 5.5.4.2 service-order-client 模块暴露接口

```
OrderFeignClient

/**
 * 提交秒杀订单
 * @param orderInfo
 * @return
 */
@PostMapping("/api/order/inner/seckill/submitOrder")
Long submitOrder(@RequestBody OrderInfo orderInfo);

OrderDegradeFeignClient

@Override
public Long submitOrder(OrderInfo orderInfo) {
    return null;
}
```

#### 5.5.4.3 service-activity 模块秒杀下单

```
SeckillGoodsApiController

@Autowired
private OrderFeignClient orderFeignClient;

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

    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);  
}
```

页面提交订单代码片段

```
submitOrder() {  
    seckill.submitOrder(this.order).then(response => {  
        if (response.data.code == 200) {  
            window.location.href =  
'http://payment.gmall.com/pay.html?orderId='  
response.data.data  
        } else {  
            alert(response.data.message)  
        }  
    })  
},
```

说明：下单成功后，后续流程与正常订单一致

## 5.6 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

### 5.6.1 添加常量 MqConst 类

```
/**
 * 定时任务
 */
public static final String ROUTING_TASK_18 = "seckill.task.18";
//队列
public static final String QUEUE_TASK_18 = "queue.task.18";
```

### 5.6.2 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() {

    rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_TASK,
MqConst.ROUTING_TASK_18, "");
}
```

### 5.6.3 接收消息并处理

Service-activity 接收消息

```
SeckillReceiver

// 监听删除消息!
@sneakyThrows
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(value = MqConst.QUEUE_TASK_18,durable =
"true",autoDelete = "false"),
    exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_TASK),
    key = {MqConst.ROUTING_TASK_18}
))
public void deleteRedisData(Message message, Channel channel){
    try {
        // 查询哪些商品是秒杀结束的! end_time , status = 1
        // select * from seckill_goods where status = 1 and end_time <
new Date();
        QueryWrapper<SeckillGoods> seckillGoodsQueryWrapper = new
```

```
QueryWrapper<>();
    seckillGoodsQueryWrapper.eq("status",1);
    seckillGoodsQueryWrapper.le("end_time",new Date());
    List<SeckillGoods> seckillGoodsList =
    seckillGoodsMapper.selectList(seckillGoodsQueryWrapper);

    // 对应将秒杀结束缓存中的数据删除!
    for (SeckillGoods seckillGoods : seckillGoodsList) {
        // seckill:stock:46 删除库存对应 key

    redisTemplate.delete(RedisConst.SECKILL_STOCK_PREFIX+seckillGoods.getSk
    uId());

        // 如果有多个秒杀商品的时候,
        //
    redisTemplate.boundHashOps(RedisConst.SECKILL_GOODS).delete(seckillGood
    s.getSkuId());
    }
    // 删除预热等数据! 主要针对于预热数据删除! 我们项目只针对一个商品的
    秒杀! 如果是多个秒杀商品, 则不能这样直接删除预热秒杀商品的 key!
    // 46 : 10:00 -- 10:30 | 47 : 18:10 -- 18:30
    redisTemplate.delete(RedisConst.SECKILL_GOODS);
    // 预下单
    redisTemplate.delete(RedisConst.SECKILL_ORDERS);
    // 删除真正下单数据
    redisTemplate.delete(RedisConst.SECKILL_ORDERS_USERS);

    // 修改数据库秒杀对象的状态!
    SeckillGoods seckillGoods = new SeckillGoods();
    // 1:表示审核通过, 2: 表示秒杀结束
    seckillGoods.setStatus("2");

    seckillGoodsMapper.update(seckillGoods,seckillGoodsQueryWrapper);
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 手动确认消息
    channel.basicAck(message.getMessageProperties().getDeliveryTag(),false)
    ;
}
```

说明: 情况 redis 缓存, 同时更改秒杀商品活动结束

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