**畅购电商系统开发**

**第13天**

传智播客.黑马程序员.深圳

# 学习目标

* 秒杀业务分析
* 秒杀商品压入Redis缓存
* Spring定时任务了解
* 秒杀商品频道页实现
* 秒杀商品详情页实现
* 下单实现

# 秒杀业务分析

## 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。

秒杀商品通常有两种限制：库存限制、时间限制。

需求：

（1）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、秒杀时段等信息

（2）秒杀频道首页列出秒杀商品（进行中的）点击秒杀商品图片跳转到秒杀商品详细页。

（3）商品详细页显示秒杀商品信息，点击立即抢购实现秒杀下单，下单时扣减库存。当库存为0或不在活动期范围内时无法秒杀。

（4）秒杀下单成功，直接跳转到支付页面（微信扫码），支付成功，跳转到成功页，填写收货地址、电话、收件人等信息，完成订单。

（5）当用户秒杀下单5分钟内未支付，取消预订单，调用微信支付的关闭订单接口，恢复库存。

## 表结构说明

秒杀商品信息表

**CREATE** **TABLE** `tb\_seckill\_goods` (

`id` bigint(20) **NOT** NULL AUTO\_INCREMENT,

`sup\_id` bigint(20) **DEFAULT** NULL COMMENT 'spu ID',

`sku\_id` bigint(20) **DEFAULT** NULL COMMENT 'sku ID',

`name` varchar(100) **DEFAULT** NULL COMMENT '标题',

`small\_pic` varchar(150) **DEFAULT** NULL COMMENT '商品图片',

`price` decimal(10,2) **DEFAULT** NULL COMMENT '原价格',

`cost\_price` decimal(10,2) **DEFAULT** NULL COMMENT '秒杀价格',

`create\_time` datetime **DEFAULT** NULL COMMENT '添加日期',

`check\_time` datetime **DEFAULT** NULL COMMENT '审核日期',

`status` char(1) **DEFAULT** NULL COMMENT '审核状态，0未审核，1审核通过，2审核不通过',

`start\_time` datetime **DEFAULT** NULL COMMENT '开始时间',

`end\_time` datetime **DEFAULT** NULL COMMENT '结束时间',

`num` int(11) **DEFAULT** NULL COMMENT '秒杀商品数',

`stock\_count` int(11) **DEFAULT** NULL COMMENT '剩余库存数',

`introduction` varchar(2000) **DEFAULT** NULL COMMENT '描述',

**PRIMARY** **KEY** (`id`)

) **ENGINE**=**InnoDB** AUTO\_INCREMENT=4 **DEFAULT** **CHARSET**=utf8;

秒杀订单表

**CREATE** **TABLE** `tb\_seckill\_order` (

`id` bigint(20) **NOT** NULL COMMENT '主键',

`seckill\_id` bigint(20) **DEFAULT** NULL COMMENT '秒杀商品ID',

`money` decimal(10,2) **DEFAULT** NULL COMMENT '支付金额',

`user\_id` varchar(50) **DEFAULT** NULL COMMENT '用户',

`create\_time` datetime **DEFAULT** NULL COMMENT '创建时间',

`pay\_time` datetime **DEFAULT** NULL COMMENT '支付时间',

`status` char(1) **DEFAULT** NULL COMMENT '状态，0未支付，1已支付',

`receiver\_address` varchar(200) **DEFAULT** NULL COMMENT '收货人地址',

`receiver\_mobile` varchar(20) **DEFAULT** NULL COMMENT '收货人电话',

`receiver` varchar(20) **DEFAULT** NULL COMMENT '收货人',

`transaction\_id` varchar(30) **DEFAULT** NULL COMMENT '交易流水',

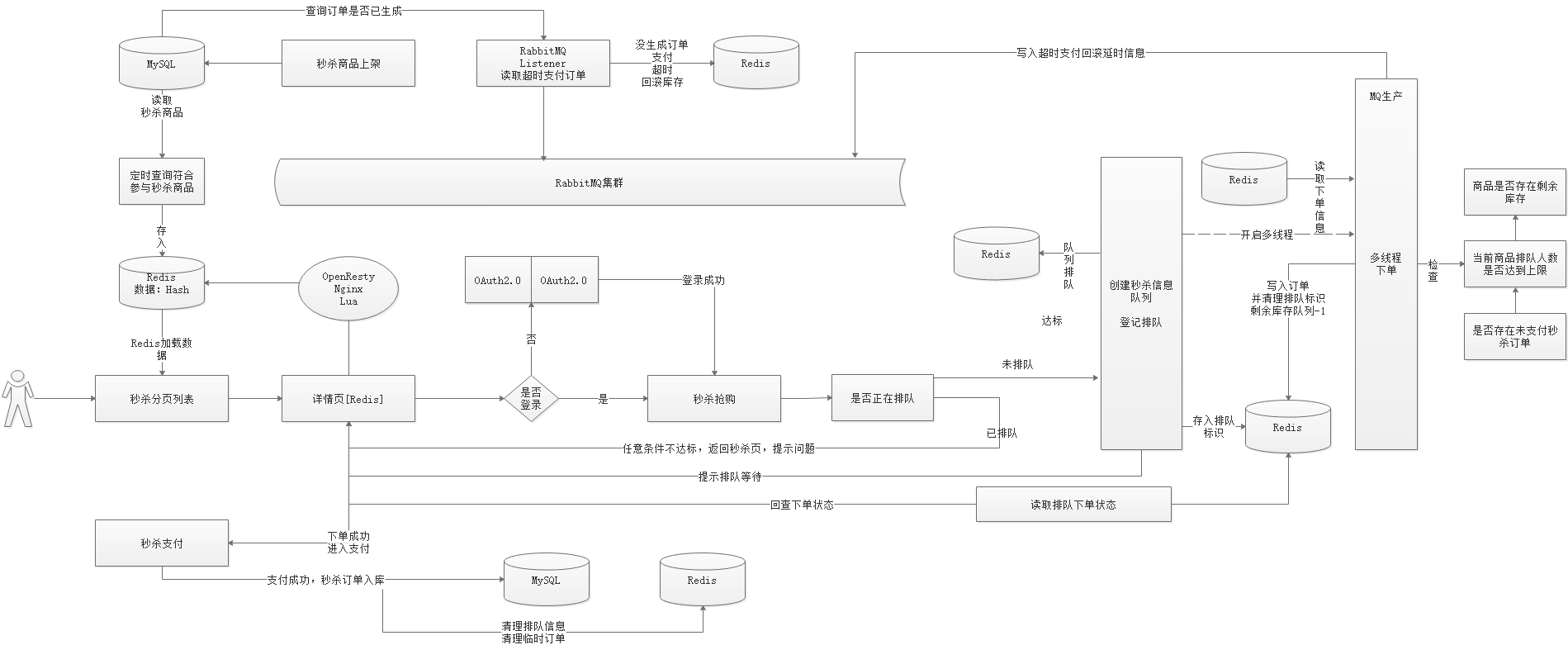
**PRIMARY** **KEY** (`id`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8;

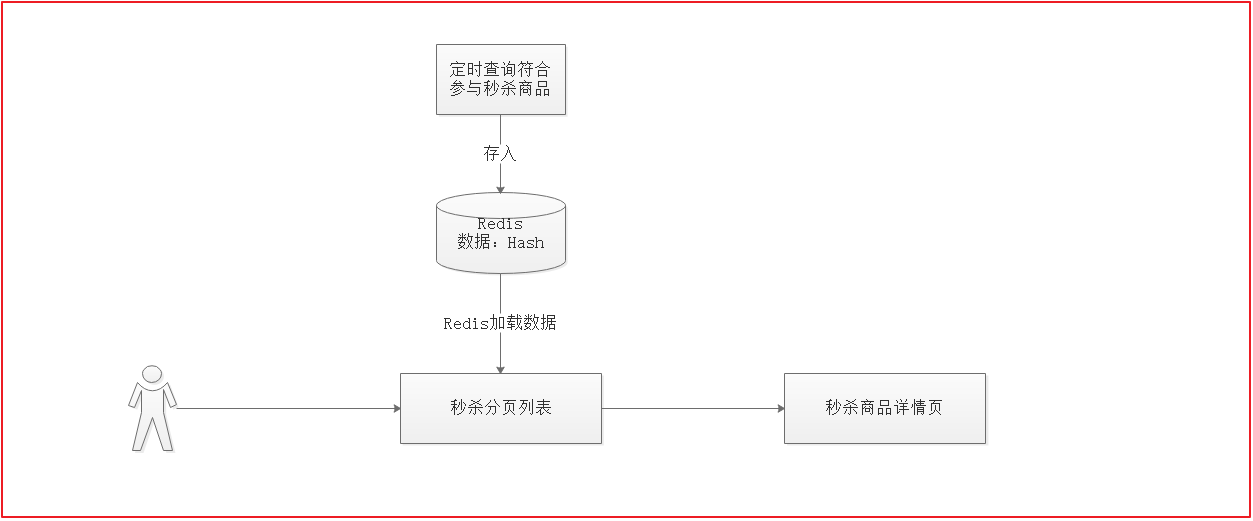
## 秒杀需求分析

秒杀技术实现核心思想是运用缓存减少数据库瞬间的访问压力！读取商品详细信息时运用缓存，当用户点击抢购时减少缓存中的库存数量，当库存数为0时或活动期结束时，同步到数据库。 产生的秒杀预订单也不会立刻写到数据库中，而是先写到缓存，当用户付款成功后再写入数据库。

当然，上面实现的思路只是一种最简单的方式，并未考虑其中一些问题，例如并发状况容易产生的问题。我们看看下面这张思路更严谨的图：



# 秒杀商品压入缓存



我们这里秒杀商品列表和秒杀商品详情都是从Redis中取出来的，所以我们首先要将符合参与秒杀的商品定时查询出来，并将数据存入到Redis缓存中。

数据存储类型我们可以选择Hash类型。

秒杀分页列表这里可以通过获取redisTemplate.boundHashOps(key).values()获取结果数据。

秒杀商品详情，可以通过redisTemplate.boundHashOps(key).get(key)获取详情。

## 秒杀服务工程

我们将商品数据压入到Reids缓存，可以在秒杀工程的服务工程中完成，可以按照如下步骤实现：

**1.查询活动没结束的所有秒杀商品**

**1)状态必须为审核通过 status**=1

2)商品库存个数>0

3)活动没有结束 endTime>=now()

4)在Redis中没有该商品的缓存

5)执行查询获取对应的结果集

2.将活动没有结束的秒杀商品入库

我们首先搭建一个秒杀服务工程，然后按照上面步骤实现。

### 创建工程

搭建changgou\_service\_seckill\_api与changgou\_service\_seckill，作为秒杀工程的服务提供工程。

api工程无特别依赖，service依赖如下：

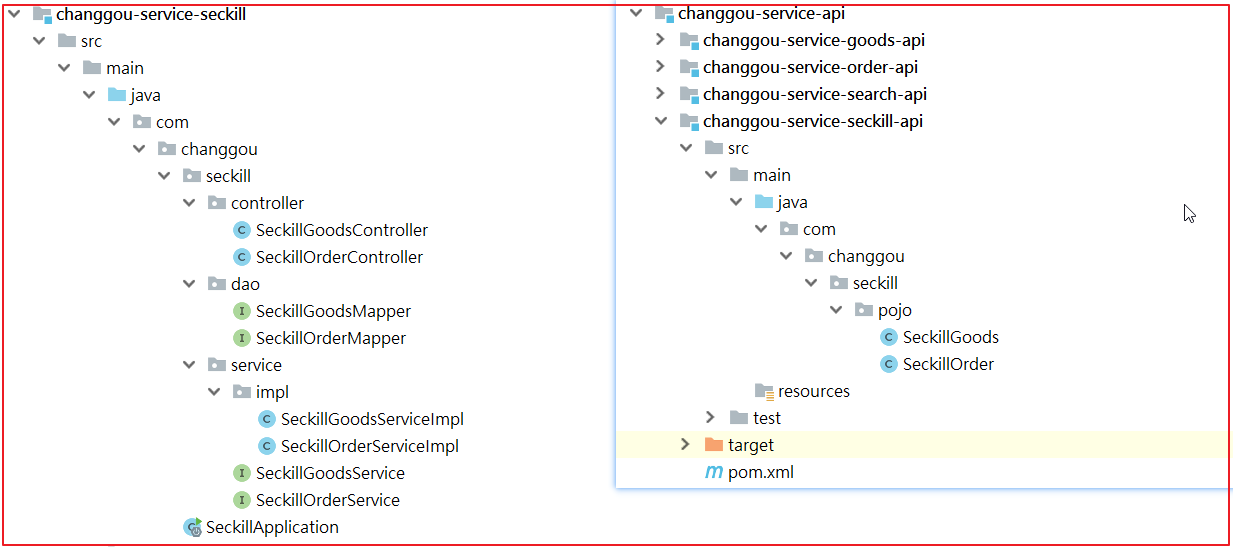
*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_service</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_service\_seckill</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_seckill\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

### application.yml配置

**server**:  
 **port**: 18091  
**spring**:  
 **application**:  
 **name**: seckill  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_seckill?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
 **rabbitmq**:  
 **host**: 192.168.211.132 *#mq的服务器地址* **username**: guest *#账号* **password**: guest *#密码* **main**:  
 **allow-bean-definition-overriding**: **true  
 redis**:  
 **host**: 192.168.211.132  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true***#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **timeout**:  
 *#如果enabled设置为false，则请求超时交给ribbon控制* **enabled**: true  
 **isolation**:  
 **thread**:  
 **timeoutInMilliseconds**: 10000  
 **strategy**: SEMAPHORE

### 准备生成代码

将生成的service、Dao、 Pojo文件导入到工程中，如下图：



### 启动引导类

@SpringBootApplication  
@EnableEurekaClient  
@EnableFeignClients  
@MapperScan(basePackages = {**"com.changgou.seckill.dao"**})  
@EnableScheduling  
**public class** SeckillApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SeckillApplication.**class**,args);  
 }  
  
 @Bean  
 **public** IdWorker idWorker(){  
 **return new** IdWorker(0,2);  
 }  
}

## 定时任务

一会儿我们采用SpringTask定时任务定时将符合参与秒杀的商品查询出来再存入到Redis缓存，所以这里需要使用到定时任务。至于SpringTask的课程我们在SpringBoot课程中已有讲解，大家可以回看相应课程内容。

这里我们了解下定时任务相关的配置,配置步骤如下：

1)在定时任务类的指定方法上加上@Scheduled开启定时任务

2)定时任务表达式：使用cron属性来配置定时任务执行时间

### 定时任务方法配置

创建com.changgou.seckill.task.SeckillGoodsPushTask类，并在类中加上定时任务执行方法，代码如下：

@Component  
**public class** SeckillGoodsPushTask {  
 */\*\*  
 \* 30秒执行一次  
 \*/* @Scheduled(cron = **"0/30 \* \* \* \* \*"**)  
 **public void** loadGoodsPushRedis(){  
 System.***out***.println(**"定时任务被调度了..."**);  
 }  
}

### 定时任务常用时间表达式

Cron表达式是一个字符串，字符串以5或6个空格隔开，分为6或7个域，每一个域代表一个含义，Cron有如下两种语法格式：

（1）Seconds Minutes Hours DayofMonth Month DayofWeek Year

（2）Seconds Minutes Hours DayofMonth Month DayofWeek

cron从左到右（用空格隔开）：秒 分 小时 月份中的日期 月份 星期中的日期 年份

**各字段的含义**

|  |  |  |
| --- | --- | --- |
|  | | |
| 字段 | 允许值 | 允许的特殊字符 |
| 秒（Seconds） | 0~59的整数 | , - \* /    四个字符 |
| 分（*Minutes*） | 0~59的整数 | , - \* /    四个字符 |
| 小时（*Hours*） | 0~23的整数 | , - \* /    四个字符 |
| 日期（*DayofMonth*） | 1~31的整数（但是你需要考虑你月的天数） | ,- \* ? / L W C     八个字符 |
| 月份（*Month*） | 1~12的整数或者 JAN-DEC | , - \* /    四个字符 |
| 星期（*DayofWeek*） | 1~7的整数或者 SUN-SAT （1=SUN） | , - \* ? / L C # 八个字符 |
| 年(可选，留空)（*Year*） | 1970~2099 | , - \* /    四个字符 |

**注意事项：**

每一个域都使用数字，但还可以出现如下特殊字符，它们的含义是：

（1）\*：表示匹配该域的任意值。假如在Minutes域使用\*, 即表示每分钟都会触发事件。

（2）?：只能用在DayofMonth和DayofWeek两个域。它也匹配域的任意值，但实际不会。因为DayofMonth和DayofWeek会相互影响。例如想在每月的20日触发调度，不管20日到底是星期几，则只能使用如下写法： 13 13 15 20 \* ?, 其中最后一位只能用？，而不能使用\*，如果使用\*表示不管星期几都会触发，实际上并不是这样。

（3）-：表示范围。例如在Minutes域使用5-20，表示从5分到20分钟每分钟触发一次

（4）/：表示起始时间开始触发，然后每隔固定时间触发一次。例如在Minutes域使用5/20,则意味着5分钟触发一次，而25，45等分别触发一次.

（5）,：表示列出枚举值。例如：在Minutes域使用5,20，则意味着在5和20分每分钟触发一次。

（6）L：表示最后，只能出现在DayofWeek和DayofMonth域。如果在DayofWeek域使用5L,意味着在最后的一个星期四触发。

（7）W:表示有效工作日(周一到周五),只能出现在DayofMonth域，系统将在离指定日期的最近的有效工作日触发事件。例如：在 DayofMonth使用5W，如果5日是星期六，则将在最近的工作日：星期五，即4日触发。如果5日是星期天，则在6日(周一)触发；如果5日在星期一到星期五中的一天，则就在5日触发。另外一点，W的最近寻找不会跨过月份 。

（8）LW:这两个字符可以连用，表示在某个月最后一个工作日，即最后一个星期五。

（9）#:用于确定每个月第几个星期几，只能出现在DayofMonth域。例如在4#2，表示某月的第二个星期三。

## 秒杀商品压入缓存实现

### 数据检索条件分析

按照2.1中的几个步骤实现将秒杀商品从数据库中查询出来，并存入到Redis缓存

**1.查询活动没结束的所有秒杀商品**

**1)计算秒杀时间段**

**2)状态必须为审核通过 status**=1

3)商品库存个数>0

4)活动没有结束 endTime>=now()

5)在Redis中没有该商品的缓存

6)执行查询获取对应的结果集

2.将活动没有结束的秒杀商品入库

上面这里会涉及到时间操作，所以这里提前准备了一个时间工具包DateUtil。

### 时间菜单分析



我们将商品数据从数据库中查询出来，并存入Redis缓存，但页面每次显示的时候，只显示当前正在秒杀以及往后延时2个小时、4个小时、6个小时、8个小时的秒杀商品数据。我们要做的第一个事是计算出秒杀时间菜单，这个菜单是从后台获取的。

这个时间菜单的计算我们来分析下，可以先求出当前时间的凌晨，然后每2个小时后作为下一个抢购的开始时间，这样可以分出12个抢购时间段,如下：

00:00-02:00

02:00-04:00

04:00-06:00

06:00-08:00

08:00-10:00

10:00-12:00

12:00-14:00

14:00-16:00

16:00-18:00

18:00-20:00

20:00-22:00

22:00-00:00

而现实的菜单只需要计算出当前时间在哪个时间段范围，该时间段范围就属于正在秒杀的时间段，而后面即将开始的秒杀时间段的计算也就出来了，可以在当前时间段基础之上+2小时、+4小时、+6小时、+8小时。

关于时间菜单的运算，在common工程的DateUtil里已经实现，代码如下：

**public class** DateUtil {  
  
 *//时间格式* **public static final** String ***PATTERN\_YYYYMMDDHH*** = **"yyyyMMddHH"**;  
 **public static final** String ***PATTERN\_YYYY\_MM\_DDHHMM*** = **"yyyy-MM-dd HH:mm"**;

//省略其它代码………..  
  
 */\*\*\*  
 \* 获取时间菜单  
 \** ***@return*** *\*/* **public static** List<Date> getDateMenus(){  
 *//定义一个List<Date>集合，存储所有时间段* List<Date> dates = *getDates*(12);  
 *//判断当前时间属于哪个时间范围* Date now = **new** Date();  
 **for** (Date cdate : dates) {  
 *//开始时间<=当前时间<开始时间+2小时* **if**(cdate.getTime()<=now.getTime() && now.getTime()<*addDateHour*(cdate,2).getTime()){  
 now = cdate;  
 **break**;  
 }  
 }  
  
 *//当前需要显示的时间菜单* List<Date> dateMenus = **new** ArrayList<Date>();  
 **for** (**int** i = 0; i <5 ; i++) {  
 dateMenus.add(*addDateHour*(now,i\*2));  
 }  
 **return** dateMenus;  
 }  
  
 */\*\*\*  
 \* 时间转成yyyyMMddHH  
 \** ***@param date*** *\** ***@param pattern*** *\** ***@return*** *\*/* **public static** String data2str(Date date, String pattern){  
 SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(pattern);  
 **return** simpleDateFormat.format(date);  
 }  
  
 **public static void** main(String[] args) {  
 List<Date> dateMenus = *getDateMenus*();  
 **for** (Date date : dateMenus) {  
 System.***out***.println(*data2str*(date, **"HH"**));  
 }  
 }  
}

### 查询秒杀商品导入Reids

我们可以写个定时任务，查询从当前时间开始，往后延续4个时间菜单间隔，也就是一共只查询5个时间段抢购商品数据，并压入缓存，实现代码如下：

修改SeckillGoodsPushTask的loadGoodsPushRedis方法，代码如下：

@Autowired  
**private** SeckillGoodsMapper **seckillGoodsMapper**;  
@Autowired  
**private** RedisTemplate **redisTemplate**;  
  
*/\*\*  
 \* 30秒执行一次  
 \*/*@Scheduled(cron = **"0/5 \* \* \* \* \*"**)  
**public void** loadGoodsPushRedis() {  
 System.***out***.println(**"定时任务被调度了..."**);  
 *//获取时间段集合* List<Date> dateMenus = DateUtil.*getDateMenus*();  
 *//循环时间段* **for** (Date startTime : dateMenus) {  
 *// namespace = SeckillGoods\_20195712* String extName = DateUtil.*data2str*(startTime, DateUtil.***PATTERN\_YYYYMMDDHH***);  
  
 *//根据时间段数据查询对应的秒杀商品数据* Example example = **new** Example(SeckillGoods.**class**);  
 Example.Criteria criteria = example.createCriteria();  
 *// 1)商品必须审核通过 status=1* criteria.andEqualTo(**"status"**, **"1"**);  
 *// 2)库存>0* criteria.andGreaterThan(**"stockCount"**, 0);  
 *// 3)开始时间<=活动开始时间* criteria.andGreaterThanOrEqualTo(**"startTime"**, startTime);  
 *// 4)活动结束时间<开始时间+2小时* criteria.andLessThan(**"endTime"**, DateUtil.*addDateHour*(startTime, 2));  
 *// 5)排除之前已经加载到Redis缓存中的商品数据* Set keys = **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + extName).keys();  
 **if** (keys != **null** && keys.size() > 0) {  
 criteria.andNotIn(**"id"**, keys);  
 }  
 *//查询数据* List<SeckillGoods> seckillGoods = **seckillGoodsMapper**.selectByExample(example);  
  
 System.***out***.println(extName + **"时段导入商品个数为："** + seckillGoods.size());  
 *//将秒杀商品数据存入到Redis缓存* **for** (SeckillGoods seckillGood : seckillGoods) {  
 **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + extName).put(seckillGood.getId(), seckillGood);  
 }  
 }  
}

备注：已准备好的数据库中有9W多条秒杀商品数据，而且最大end\_time为2020年1月28，如果当前时间超过了2020-01-28，需要把数据库的日期一下，才能查询到数据。

MySQL 为日期增加一个时间间隔：date\_add()

now()       //now函数为获取当前时间

select date\_add(now(), interval 1 day); - 加1天

select date\_add(now(), interval 1 hour); -加1小时

select date\_add(now(), interval 1 minute); - 加1分钟

select date\_add(now(), interval 1 second); -加1秒

select date\_add(now(), interval 1 microsecond);-加1毫秒

select date\_add(now(), interval 1 week);-加1周

select date\_add(now(), interval 1 month);-加1月

select date\_add(now(), interval 1 quarter);-加1季

select date\_add(now(), interval 1 year);-加1年

-- 测试sql

update tb\_seckill\_goods set end\_time = date\_add(end\_time,interval 1 day) where id = 1131814837488324608

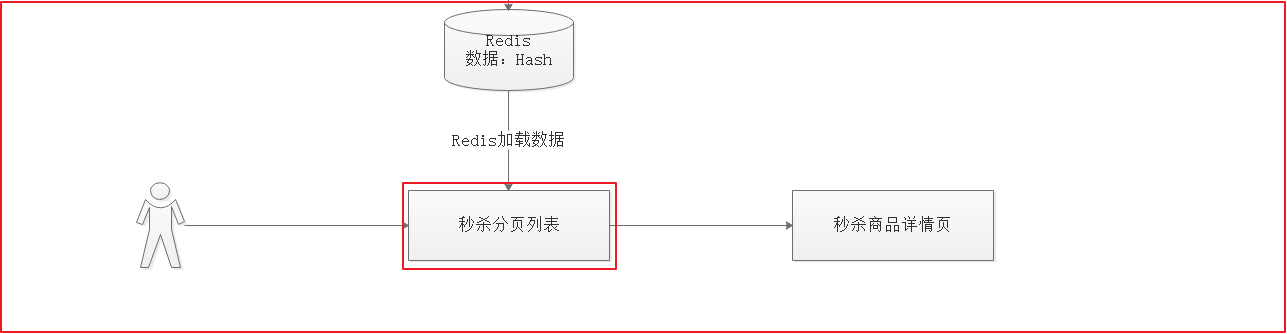
MySQL adddate(), addtime()函数，可以用date\_add() 来替代。

2. MySQL 为日期减去一个时间间隔：date\_sub()

MySQL date\_sub() 日期时间函数 和date\_add() 用法一致。

MySQL 中subdate(),subtime()函数，建议，用date\_sub()来替代。

# 秒杀频道页



秒杀频道首页，显示正在秒杀的和未开始秒杀的商品（已经开始或者还没开始，未结束的秒杀商品）

## 秒杀时间菜单



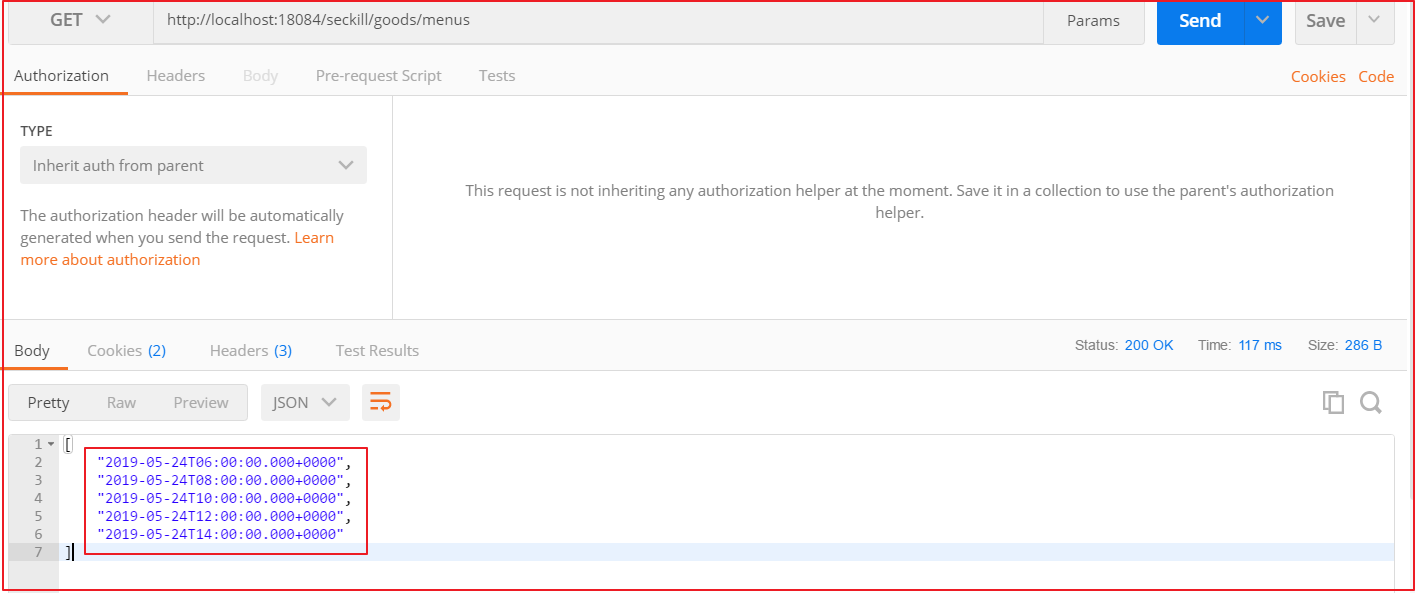
如上图，时间菜单需要根据当前时间动态加载，时间菜单的计算上面功能中已经实现，在DateUtil工具包中。我们只需要将时间菜单获取，然后响应到页面，页面根据对应的数据显示即可。

修改com.changgou.seckill.controller.SeckillGoodsController，并添加菜单获取方法，代码如下：

*/\*\*\*\*\*  
 \* 获取时间菜单  
 \* URLL:/seckillGoods/menus  
 \*/*@RequestMapping(value = **"/menus"**)  
**public** List<Date> dateMenus(){  
 List<Date> dateMenus = DateUtil.*getDateMenus*();  
 **return** dateMenus;  
}

使用Postman测试，效果如下：

<http://localhost:18091/seckillGoods/menus>



## 秒杀频道页



秒杀频道页是指将对应时区的秒杀商品从Reids缓存中查询出来，并到页面显示。对应时区秒杀商品存储的时候以Hash类型进行了存储，key=SeckillGoods\_2019010112，value=每个商品详情。

每次用户在前端点击对应时间菜单的时候，可以将时间菜单的开始时间以yyyyMMddHH格式提交到后台，后台根据时间格式查询出对应时区秒杀商品信息。

### 业务层

修改com.changgou.seckill.service.SeckillGoodsService,添加根据时区查询秒杀商品的方法，代码如下：

*/\*\*\*  
 \* 获取指定时间对应的秒杀商品列表  
 \** ***@param key*** *\*/*List<SeckillGoods> list(String key);

修改com.changgou.seckill.service.impl.SeckillGoodsServiceImpl，实现根据时区查询秒杀商品的方法，代码：

*/\*\*\*  
 \* Redis中根据Key获取秒杀商品列表  
 \** ***@param key*** *\** ***@return*** *\*/*@Override  
**public** List<SeckillGoods> list(String key) {  
 **return redisTemplate**.boundHashOps(**"SeckillGoods\_"**+key).values();  
}

### 控制层

修改com.changgou.seckill.controller.SeckillGoodsController，并添加秒杀商品查询方法，代码如下：

*/\*\*\*\*  
 \* URL:/seckillGoods/list  
 \* 对应时间段秒杀商品集合查询  
 \** ***@param time****:2019050716  
 \*/*@RequestMapping(value = **"/list"**)  
**public** List<SeckillGoods> list(String time){  
 *//调用Service查询数据* **return seckillGoodsService**.list(time);  
}

测试

使用Postman测试，效果如下：

<http://localhost:18091/seckillGoods/list?time=2019052414>



# 秒杀详情页

通过秒杀频道页点击请购按钮，会跳转到商品秒杀详情页，秒杀详情页需要根据商品ID查询商品详情，我们可以在频道页点击秒杀抢购的时候将ID一起传到后台，然后根据ID去Redis中查询详情信息。

## 业务层

修改com.changgou.seckill.service.SeckillGoodsService，添加如下方法实现查询秒杀商品详情,代码如下：

*/\*\*\*\*  
 \* 根据ID查询商品详情  
 \** ***@param time****:时间区间  
 \** ***@param id****:商品ID  
 \*/*SeckillGoods one(String time,Long id);

修改com.changgou.seckill.service.impl.SeckillGoodsServiceImpl，添加查询秒杀商品详情，代码如下：

*/\*\*\*\*  
 \* 根据商品ID查询商品详情  
 \** ***@param time****:时间区间  
 \** ***@param id****:商品ID  
 \*/*@Override  
**public** SeckillGoods one(String time, Long id) {  
 **return** (SeckillGoods) **redisTemplate**.boundHashOps(**"SeckillGoods\_"**+time).get(id);  
}

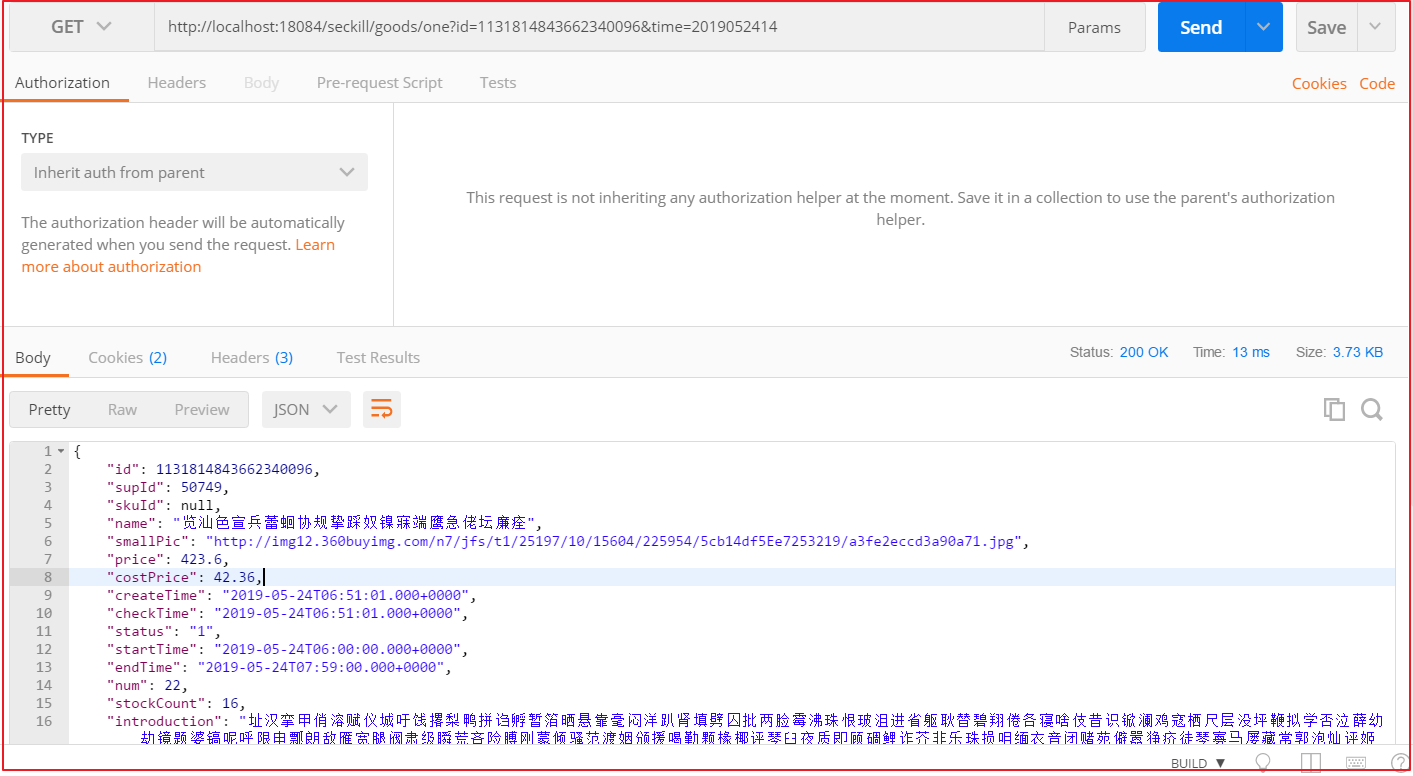
## 控制层

修改com.changgou.seckill.controller.SeckillGoodsController，添加如下方法实现查询秒杀商品详情，代码如下：

*/\*\*\*\*  
 \* URL:/seckillGoods/one  
 \* 根据ID查询商品详情  
 \** ***@param time*** *时间区间  
 \** ***@param id*** *商品id  
 \*/*@RequestMapping(value = **"/one"**)  
**public** SeckillGoods one(String time,Long id){  
 *//调用Service查询商品详情* **return seckillGoodsService**.one(time,id);  
}

使用Postman测试，效果如下：

<http://localhost:18091/seckillGoods/one?id=1131814843662340096&time=2019052414>



# 下单实现

用户下单，从控制层->Service层->Dao层，所以我们先把dao创建好，再创建service层，再创建控制层。

用户下单，为了提升下单速度，我们将订单数据存入到Redis缓存中，如果用户支付了，则将Reids缓存中的订单存入到MySQL中，并清空Redis缓存中的订单。

## 业务层

修改com.changgou.seckill.service.SeckillOrderService，并在接口中增加下单方法，代码如下：

*/\*\*\*  
 \* 添加秒杀订单  
 \** ***@param id****:商品ID  
 \** ***@param time****:时间区间  
 \** ***@param username****:用户登录名  
 \** ***@return*** *\*/***boolean** add(Long id, String time, String username);

修改com.changgou.seckill.service.impl.SeckillOrderServiceImpl实现类，并在类中添加下单实现方法，代码：

@Autowired  
**private** RedisTemplate **redisTemplate**;  
@Autowired  
**private** SeckillGoodsMapper **seckillGoodsMapper**;  
@Autowired  
**private** IdWorker **idWorker**;  
*/\*\*\*\*  
 \* 添加订单  
 \*/*@Override  
**public boolean** add(Long id, String time, String username){  
 *//1.获取商品数据* SeckillGoods goods = (SeckillGoods) **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).get(id);  
 *//2.如果没有库存，则直接抛出异常* **if** (goods == **null** || goods.getStockCount() <= 0) {  
 **throw new** RuntimeException(**"你来晚了一步，商品已抢购一空!"**);  
 }  
 *//3.如果有库存，则创建秒杀商品订单* SeckillOrder seckillOrder = **new** SeckillOrder();  
 seckillOrder.setId(**idWorker**.nextId());  
 seckillOrder.setSeckillId(id);  
 seckillOrder.setMoney(goods.getCostPrice());  
 seckillOrder.setUserId(username);  
 seckillOrder.setCreateTime(**new** Date());  
 seckillOrder.setStatus(**"0"**);  
 *//将秒杀订单存入到Redis中* **redisTemplate**.boundHashOps(**"SeckillOrder"**).put(username,seckillOrder);  
  
 *//4.扣减库存* goods.setStockCount(goods.getStockCount() - 1);  
  
 *//5.判断当前商品是否还有库存* **if** (goods.getStockCount() <= 0) {  
 *//并且将商品数据同步到MySQL中* **seckillGoodsMapper**.updateByPrimaryKeySelective(goods);  
 *//如果没有库存,则清空Redis缓存中该商品* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).delete(id);  
 } **else** {  
 *//如果有库存，则将扣减库存后的goods重新放入redis* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).put(id, goods);  
 }  
 **return true**;  
}

## 控制层

修改com.changgou.seckill.controller.SeckillOrderController，添加下单方法，代码如下：

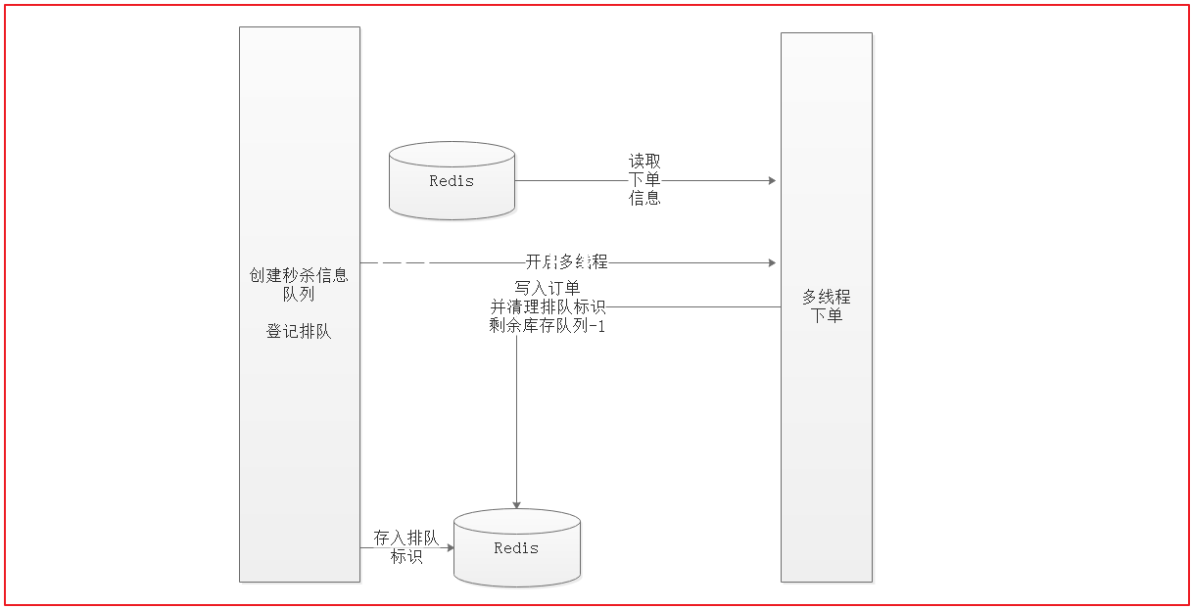
*/\*\*\*\*  
 \* URL:/seckillOrder/add  
 \* 添加订单  
 \* 匿名访问：anonymousUser  
 \** ***@param time*** *时段  
 \** ***@param id*** *商品id  
 \*/*@RequestMapping(value = **"add"**)  
**public** Result add(String time, Long id){  
 **try** {  
 *//用户登录名  
 //String username = TokenDecode.getUserInfo().get("username");* String username = **"zhangsan"**;  
 *//调用Service增加订单* **boolean** flag = **seckillOrderService**.add(id, time, username);  
 **if**(flag){  
 *//抢单成功* **return new** Result(**true**,StatusCode.***OK***,**"抢单成功！"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return new** Result(**true**,StatusCode.***ERROR***,**"服务器繁忙，请稍后再试!"**);  
}

问题分析:

上述功能完成了秒杀抢单操作，但没有解决并发相关的问题，例如并发、超卖现象，这块甚至有可能产生雪崩问题。

# 多线程抢单

## 实现思路分析



在审视秒杀中，操作一般都是比较复杂的，而且并发量特别高，比如，检查当前账号操作是否已经秒杀过该商品，检查该账号是否存在存在刷单行为，记录用户操作日志等。

下订单这里，我们一般采用多线程下单，但多线程中我们又需要保证用户抢单的公平性，也就是先抢先下单。我们可以这样实现，用户进入秒杀抢单，如果用户复合抢单资格，只需要记录用户抢单数据，存入队列，多线程从队列中进行消费即可，存入队列采用左压，多线程下单采用右取的方式。

## 异步实现

要想使用Spring的异步操作，在SpringBoot中需要先开启异步操作，用@EnableAsync注解开启，然后在对应的异步方法上添加注解@Async即可。

@SpringBootApplication  
@EnableEurekaClient  
@EnableFeignClients  
@MapperScan(basePackages = {**"com.changgou.seckill.dao"**})  
@EnableScheduling  
@EnableAsync *//开启步功能***public class** SeckillApplication {  
  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SeckillApplication.**class**,args);  
 }  
  
 @Bean  
 **public** IdWorker idWorker(){  
 **return new** IdWorker(0,2);  
 }  
}

创建com.changgou.seckill.task.MultiThreadingCreateOrder类，在类中创建一个createOrder方法，并在方法上添加@Async,代码如下：

@Component  
**public class** MultiThreadingCreateOrder {  
  
 *//加入此注解标识以下方法是开启新线程执行的-异步* @Async  
 **public void** createOrder(){  
 **try** {  
 System.***out***.println(**"开始查询下单相关业务，模拟业务处理时间....."**);  
 Thread.*sleep*(2000);  
 System.***out***.println(**"完成查询下单相关业务，模拟业务处理时间....."**);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

修改秒杀抢单SeckillOrderServiceImpl代码，注入MultiThreadingCreateOrder,并调用createOrder方法，代码如下：

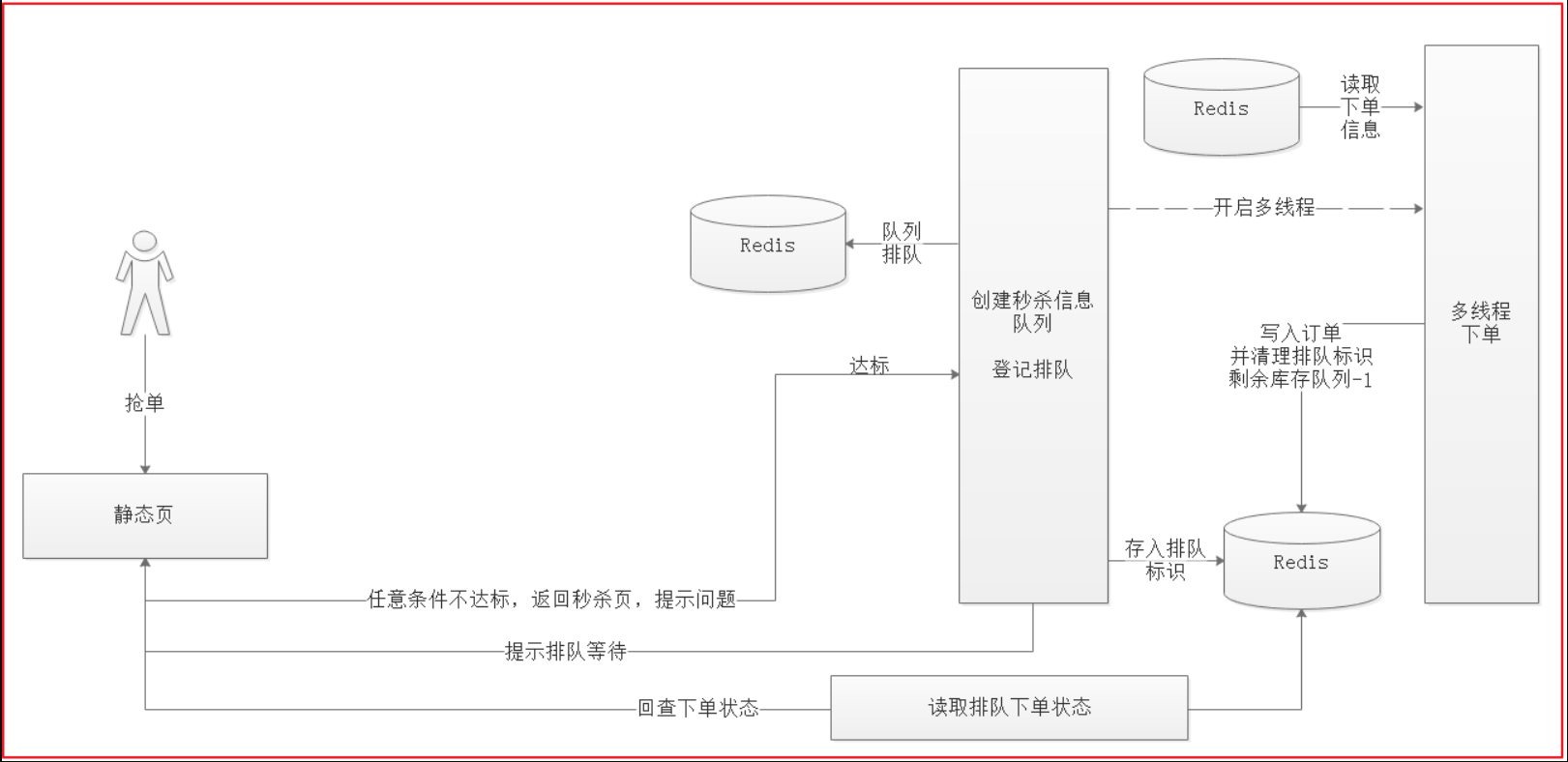
@Autowired  
**private** MultiThreadingCreateOrder **multiThreadingCreateOrder**;  
*/\*\*\*\*  
 \* 添加订单  
 \*/*@Override  
**public boolean** add(Long id, String time, String username){  
 System.***out***.println(**"进入了抢购业务方法，开始异步调用..."**);  
 *//调用异步测试* **multiThreadingCreateOrder**.createOrder();  
 System.***out***.println(**"进入了抢购业务方法，结束异步调用..."**);  
  
 *//1.省略其它代码…………*

**return true**;  
}

测试，重新下个单，可以看到控制台输出的流程是异步调用的。



## 多线程抢单



用户每次下单的时候，我们都让他们先进行排队，然后采用多线程的方式创建订单，排队我们可以采用Redis的队列实现，多线程下单我们可以采用Spring的异步实现。

### 多线程下单

将之前下单的代码全部挪到多线程的方法中，com.changgou.seckill.service.impl.SeckillOrderServiceImpl类的方法值负责调用即可，代码如下：

@Autowired  
**private** MultiThreadingCreateOrder **multiThreadingCreateOrder**;  
*/\*\*\*\*  
 \* 添加订单  
 \*/*@Override  
**public boolean** add(Long id, String time, String username){  
 *//System.out.println("进入了抢购业务方法，开始异步调用...");  
 //调用异步测试* **multiThreadingCreateOrder**.createOrder();  
 *//System.out.println("进入了抢购业务方法，结束异步调用...");* **return true**;  
}

多线程下单代码如下：

@Component  
**public class** MultiThreadingCreateOrder {  
  
 @Autowired  
 **private** RedisTemplate **redisTemplate**;  
 @SuppressWarnings(**"SpringJavaInjectionPointsAutowiringInspection"**)  
 @Autowired  
 **private** SeckillGoodsMapper **seckillGoodsMapper**;  
 @Autowired  
 **private** IdWorker **idWorker**;  
  
 *//加入此注解标识以下方法是开启新线程执行的-异步* @Async  
 **public void** createOrder(){  
 */\*try {  
 System.out.println("开始查询下单相关业务，模拟业务处理时间.....");  
 Thread.sleep(2000);  
 System.out.println("完成查询下单相关业务，模拟业务处理时间.....");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }\*/*

*//便于测试我们这里的参数先写死  
 //时间区间* String time = **"2019052510"**;  
 *//用户登录名* String username=**"zhangsan"**;  
 *//用户抢购商品* Long id = 1131814847898587136L;  
  
 *//1.获取商品数据* SeckillGoods goods = (SeckillGoods) **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).get(id);  
  
 *//2.如果没有库存，则直接抛出异常* **if** (goods == **null** || goods.getStockCount() <= 0) {  
 **throw new** RuntimeException(**"你来晚了一步，商品已抢购一空!"**);  
 }  
 *//3.如果有库存，则创建秒杀商品订单* SeckillOrder seckillOrder = **new** SeckillOrder();  
 seckillOrder.setId(**idWorker**.nextId());  
 seckillOrder.setSeckillId(id);  
 seckillOrder.setMoney(goods.getCostPrice());  
 seckillOrder.setUserId(username);  
 seckillOrder.setCreateTime(**new** Date());  
 seckillOrder.setStatus(**"0"**);  
 *//将秒杀订单存入到Redis中* **redisTemplate**.boundHashOps(**"SeckillOrder"**).put(username,seckillOrder);  
  
 *//4.扣减库存* goods.setStockCount(goods.getStockCount() - 1);  
  
 *//5.判断当前商品是否还有库存* **if** (goods.getStockCount() <= 0) {  
 *//并且将商品数据同步到MySQL中* **seckillGoodsMapper**.updateByPrimaryKeySelective(goods);  
 *//如果没有库存,则清空Redis缓存中该商品* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).delete(id);  
 } **else** {  
 *//如果有库存，则将扣减库存后的goods重新放入redis* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).put(id, goods);  
 }  
 }  
}

此时测试，是可以正常下单的，但是用户名和订单都写死了，此处需要继续优化。

### 排队下单

#### 排队信息封装

用户每次下单的时候，我们可以创建一个队列进行排队，然后采用多线程的方式创建订单，排队我们可以采用Redis的队列实现。 排队信息中需要有用户抢单的商品信息，主要包含商品ID，商品抢购时间段，用户登录名。我们可以设计个javabean，就在当前工程中创建如下：

*/\*\*  
 \* 抢购排队对象  
 \** ***@description*** *com.changgou.seckill.utils  
 \*/***public class** SeckillStatus **implements** Serializable {  
 *//秒杀用户名* **private** String **username**;  
 *//创建时间* **private** Date **createTime**;  
 *//秒杀状态 1:排队中，2:秒杀等待支付,3:支付超时，4:秒杀失败,5:支付完成* **private** Integer **status**;  
 *//秒杀的商品ID* **private** Long **goodsId**;  
 *//应付金额* **private** Float **money**;  
 *//订单号* **private** Long **orderId**;  
 *//时间段* **private** String **time**;  
  
 **public** SeckillStatus() {  
 }  
 **public** SeckillStatus(String username, Date createTime, Integer status, Long goodsId, String time) {  
 **this**.**username** = username;  
 **this**.**createTime** = createTime;  
 **this**.**status** = status;  
 **this**.**goodsId** = goodsId;  
 **this**.**time** = time;  
 }  
 *//get、set...略*}

#### 排队实现

我们可以将秒杀抢单信息存入到Redis中,这里采用List方式存储,List本身是一个队列，用户点击抢购的时候，就将用户抢购信息存入到Redis中，代码如下：

@Autowired  
**private** MultiThreadingCreateOrder **multiThreadingCreateOrder**;  
@Autowired  
**private** RedisTemplate **redisTemplate**;  
*/\*\*\*\*  
 \* 添加订单  
 \*/*@Override  
**public boolean** add(Long id, String time, String username){  
 *//System.out.println("进入了抢购业务方法，开始异步调用...");  
  
 //排队信息封装* SeckillStatus seckillStatus = **new** SeckillStatus(username, **new** Date(),1, id,time);  
 *//将秒杀抢单信息存入到Redis中,这里采用List方式存储,List本身是一个分布式队列-左进右出* **redisTemplate**.boundListOps(**"SeckillOrderQueue"**).leftPush(seckillStatus);  
  
 *//调用异步测试* **multiThreadingCreateOrder**.createOrder();  
 *//System.out.println("进入了抢购业务方法，结束异步调用...");* **return true**;  
}

多线程每次从队列中获取数据，分别获取用户名和订单商品编号以及商品秒杀时间段，进行下单操作，代码如下：

*//加入此注解标识以下方法是开启新线程执行的-异步*@Async  
**public void** createOrder(){  
 */\*try {  
 System.out.println("开始查询下单相关业务，模拟业务处理时间.....");  
 Thread.sleep(2000);  
 System.out.println("完成查询下单相关业务，模拟业务处理时间.....");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }\*/  
  
 //从队列中获取排队信息-左进右出* SeckillStatus seckillStatus = (SeckillStatus) **redisTemplate**.boundListOps(**"SeckillOrderQueue"**).rightPop();  
 *//如果有排队信息* **if**(seckillStatus != **null**){  
 *//时间区间* String time = seckillStatus.getTime();  
 *//用户登录名* String username=seckillStatus.getUsername();  
 *//用户抢购商品* Long id = seckillStatus.getGoodsId();  
  
 *//1.省略其它代码…………*   
 }  
}

### 下单状态查询

按照上面的流程，虽然可以实现用户下单异步操作，但是并不能确定下单是否成功，所以我们需要做一个页面判断，每过1秒钟查询一次下单状态,多线程下单的时候，需要修改抢单状态，支付的时候，清理抢单状态。

#### 下单更新抢单状态

用户每次点击抢购的时候，如果排队成功，则将用户抢购状态存储到Redis中，多线程抢单的时候，如果抢单成功，则更新抢单状态。

修改SeckillOrderServiceImpl的add方法，记录状态，代码如下：

@Override  
**public boolean** add(Long id, String time, String username){  
 *//System.out.println("进入了抢购业务方法，开始异步调用...");  
  
 //排队信息封装* SeckillStatus seckillStatus = **new** SeckillStatus(username, **new** Date(),1, id,time);  
 *//将秒杀抢单信息存入到Redis中,这里采用List方式存储,List本身是一个分布式队列* **redisTemplate**.boundListOps(**"SeckillOrderQueue"**).leftPush(seckillStatus);  
 *//记录用户排队信息-登记* **redisTemplate**.boundHashOps(**"UserQueueStatus"**).put(username,seckillStatus);  
 *//调用异步测试* **multiThreadingCreateOrder**.createOrder();  
 *//System.out.println("进入了抢购业务方法，结束异步调用...");* **return true**;  
}

多线程抢单更新状态，修改MultiThreadingCreateOrder的createOrder方法，代码如下：

*//加入此注解标识以下方法是开启新线程执行的-异步*@Async  
**public void** createOrder(){ *//从队列中获取排队信息-左进右出* SeckillStatus seckillStatus = (SeckillStatus) **redisTemplate**.boundListOps(**"SeckillOrderQueue"**).rightPop();  
 *//如果有排队信息* **if**(seckillStatus != **null**){  
 *//时间区间* String time = seckillStatus.getTime();  
 *//用户登录名* String username=seckillStatus.getUsername();  
 *//用户抢购商品* Long id = seckillStatus.getGoodsId();  
  
 *//1.获取商品数据* SeckillGoods goods = (SeckillGoods) **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).get(id);  
  
 *//2.如果没有库存，则直接抛出异常* **if** (goods == **null** || goods.getStockCount() <= 0) {  
 **throw new** RuntimeException(**"你来晚了一步，商品已抢购一空!"**);  
 }  
 *//3.如果有库存，则创建秒杀商品订单* SeckillOrder seckillOrder = **new** SeckillOrder();  
 seckillOrder.setId(**idWorker**.nextId());  
 seckillOrder.setSeckillId(id);  
 seckillOrder.setMoney(goods.getCostPrice());  
 seckillOrder.setUserId(username);  
 seckillOrder.setCreateTime(**new** Date());  
 seckillOrder.setStatus(**"0"**);  
 *//将秒杀订单存入到Redis中* **redisTemplate**.boundHashOps(**"SeckillOrder"**).put(username,seckillOrder);  
  
 *//4.扣减库存* goods.setStockCount(goods.getStockCount() - 1);  
  
 *//5.判断当前商品是否还有库存* **if** (goods.getStockCount() <= 0) {  
 *//并且将商品数据同步到MySQL中* **seckillGoodsMapper**.updateByPrimaryKeySelective(goods);  
 *//如果没有库存,则清空Redis缓存中该商品* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).delete(id);  
 } **else** {  
 *//如果有库存，则将扣减库存后的goods重新放入redis* **redisTemplate**.boundHashOps(**"SeckillGoods\_"** + time).put(id, goods);  
 }  
  
 *//抢单成功，更新抢单状态,排队->等待支付* seckillStatus.setStatus(2); *//1:排队中，2:秒杀等待支付,3:支付超时，4:秒杀失败,5:支付完成* seckillStatus.setOrderId(seckillOrder.getId()); *//更新订单id* seckillStatus.setMoney(**new** Float(seckillOrder.getMoney())); *//记录金额  
 //更新用户订单排队信息为等待支付* **redisTemplate**.boundHashOps(**"UserQueueStatus"**).put(username,seckillStatus);  
 }  
}

#### 后台查询抢单状态

后台提供抢单状态查询方法，**修改SeckillOrderService**，添加如下查询方法：

*/\*\*\*  
 \* 抢单状态查询  
 \** ***@param username*** *\*/*SeckillStatus queryStatus(String username);

**修改SeckillOrderServiceImpl**,添加如下实现方法：

*/\*\*\*  
 \* 抢单状态查询  
 \** ***@param username*** *\** ***@return*** *\*/*@Override  
**public** SeckillStatus queryStatus(String username) {  
 **return** (SeckillStatus) **redisTemplate**.boundHashOps(**"UserQueueStatus"**).get(username);  
}

**修改SeckillOrderController**,添加如下查询方法：

*/\*\*\*\*  
 \* 查询抢购  
 \** ***@return*** *\*/*@RequestMapping(value = **"query"**)  
**public** Result<SeckillStatus> queryStatus(){  
 *//获取用户名  
 //String username = TokenDecode.getUserInfo().get("username");* String username = **"zhangsan"**;  
  
 *//根据用户名查询用户抢购状态* SeckillStatus seckillStatus = **seckillOrderService**.queryStatus(username);  
  
 **if**(seckillStatus!=**null**){  
 **return new** Result<SeckillStatus>(**true**,seckillStatus.getStatus(),**"当前用户有抢购订单"**,seckillStatus);  
 }  
 *//NOTFOUNDERROR =20006,没有对应的抢购数据* **return new** Result(**false**,StatusCode.***NOTFOUNDERROR***,**"当前用户没有抢购订单!"**);  
}

**测试**

先清空Redis数据，重启项目，先访问<http://localhost:18091/seckillOrder/query> 可以发现没有抢购订单，

下单抢购后再次查询，就可以查询到抢购订单信息了。如果想看到抢购状态轮转，可以在多线程下单的程序中睡眠10秒来测试。