第14章 秒杀

学习目标

- 秒杀业务分析

```
1.什么是秒杀
2.业务流程
3.秒杀实现的流程->架构分析流程->重点
```

- 秒杀商品压入Redis缓存

```
秒杀商品存入到Redis来提升访问速度
1.秒杀列表数据
2.秒杀详情页数据
```

- Spring定时任务了解-定时将秒杀商品存入到Redis中

```
定时将秒杀商品存入到Redis缓存
```

- 秒杀商品频道页实现-秒杀商品列表页
- 秒杀商品详情页实现
- 下单实现(普通下单)
- 多线程异步抢单实现-队列削峰-重点

1 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。
```

1.2 表结构说明

秒杀商品信息表

```
CREATE TABLE `tb_seckill_goods` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `spu_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',8
  `end_time` datetime DEFAULT NULL COMMENT '结束时间',10
  `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;

--开始时间增加
UPDATE tb_seckill_goods c set c.start_time = DATE_ADD(c.start_time, INTERVAL 1 MONTH) ;
--结束时间增加
UPDATE tb_seckill_goods c set c.end_time = DATE_ADD(c.end_time, INTERVAL 1 MONTH) ;
```

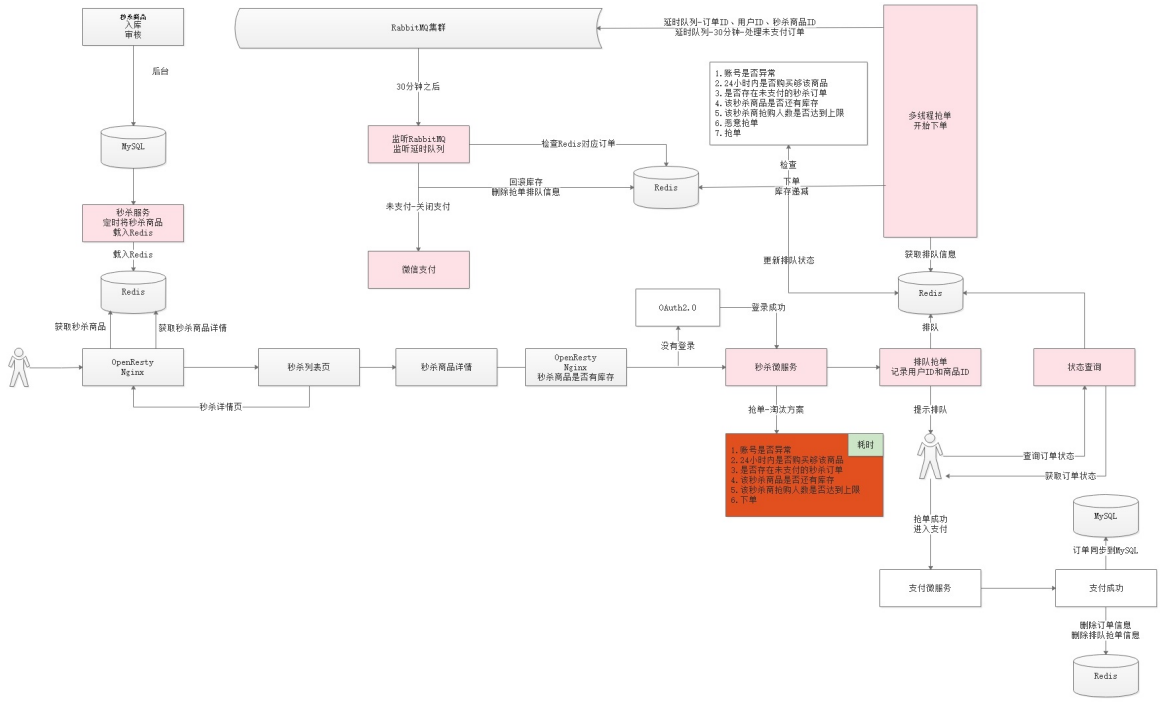
秒杀订单表

```
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;
```

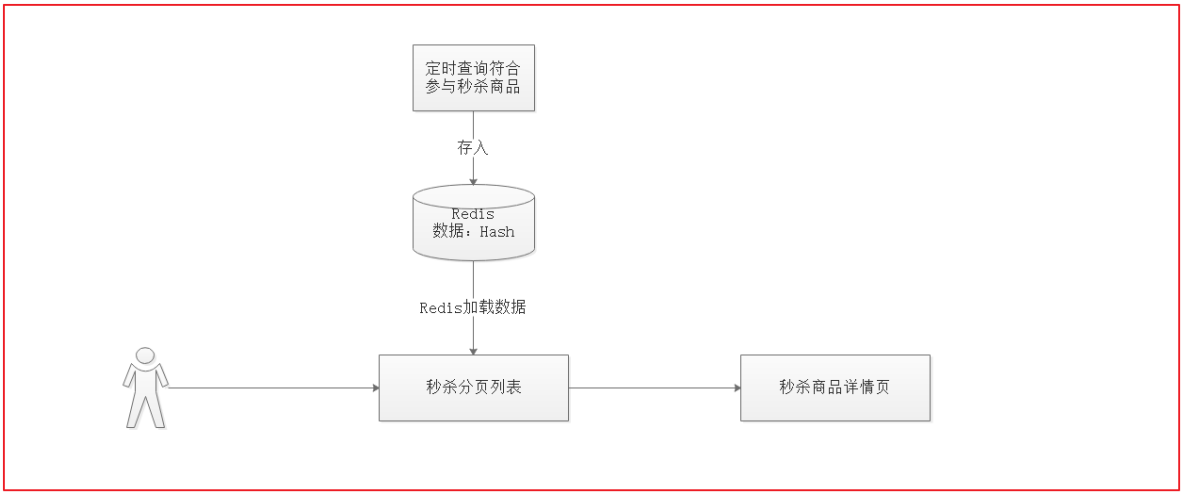
1.3 秒杀需求分析

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：



2 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。

2.1 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。

搭建changgou-service-seckill，作为秒杀工程的服务提供工程。

(1)pom.xml依赖

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
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>
    <description>秒杀微服务</description>
    <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>
```

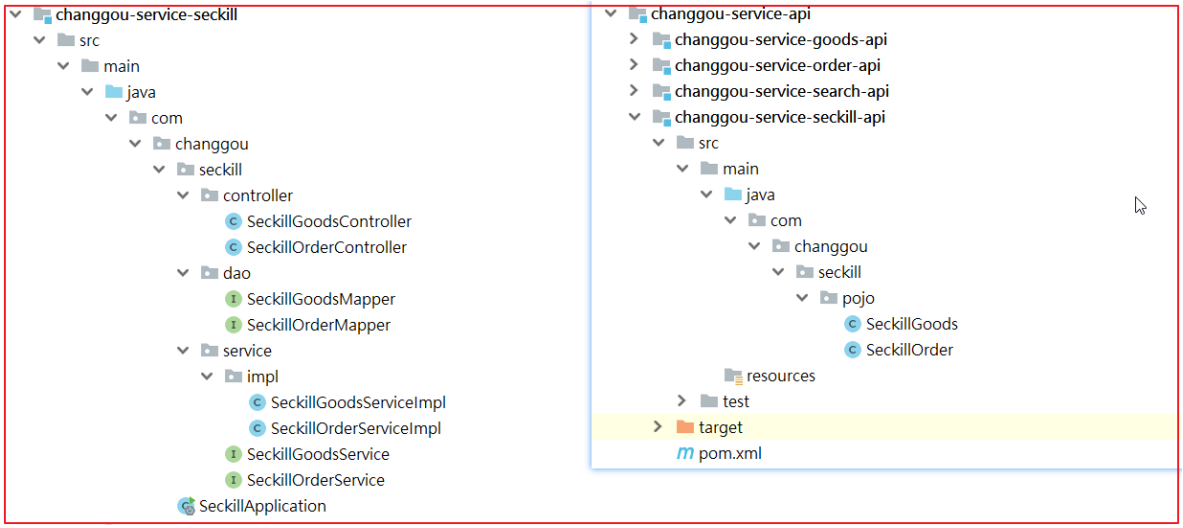
(2) application.yml配置

```
server:
  port: 18091
spring:
  application:
    name: seckill
  datasource:
    driver-class-name: com.mysql.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
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:7001/eureka
  instance:
    prefer-ip-address: true
```

(3) 导入生成文件

将生成的Dao文件和Pojo文件导入到工程中，如下图：



(4) 启动类配置

```
@SpringBootApplication
@EnableEurekaClient
@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(1,1);
    }
}
```

2.2 定时任务

一会儿我们采用Spring的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。

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
```

2.2.1 定时任务方法配置

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

```
@Component
public class SeckillGoodsPushTask {

    /****
     * 每30秒执行一次
     */
    @Scheduled(cron = "0/30 * * * * ?")
    public void loadGoodsPushRedis(){
        System.out.println("task demo");
    }
}
```

2.2.2 定时任务常用时间表达式

CronTrigger配置完整格式为： [秒][分] [小时][日] [月][周] [年]

序号	说明	是否必填	允许填写的值	允许的通配符
1	秒	是	0-59	, - * /
2	分	是	0-59	, - * /
3	小时	是	0-23	, - * /
4	日	是	1-31	, - * ? / L W
5	月	是	1-12或JAN-DEC	, - * /
6	周	是	1-7或SUN-SAT	, - * ? / L W
7	年	否	empty 或1970-2099	, - * /

使用说明：

```
通配符说明：
* 表示所有值. 例如:在分的字段上设置 "*",表示每一分钟都会触发。

? 表示不指定值。使用的场景为不需要关心当前设置这个字段的值。
```

```
例如:要在每月的10号触发一个操作，但不关心是周几，所以需要周位置的那个字段设置为"?" 具体设置为 0 0 0 10 * ?

- 表示区间。例如 在小时上设置 "10-12",表示 10,11,12点都会触发。

, 表示指定多个值，例如在周字段上设置 "MON,WED,FRI" 表示周一，周三和周五触发  12,14,19

/ 用于递增触发。如在秒上面设置"5/15" 表示从5秒开始，每增15秒触发(5,20,35,50)。 在月字段上设置'1/3'所示每月1号开始，每隔三天触发一次。

L 表示最后的意思。在日字段设置上，表示当月的最后一天(依据当前月份，如果是二月还会依据是否是润年[leap]), 在周字段上表示星期六，相当于"7"或"SAT"。如果在"L"前加上数字，则表示该数据的最后一个。例如在周字段上设置"6L"这样的格式,则表示“本月最后一个星期五"

W 表示离指定日期的最近那个工作日(周一至周五). 例如在日字段上设置"15W"，表示离每月15号最近的那个工作日触发。如果15号正好是周六，则找最近的周五(14号)触发, 如果15号是周未，则找最近的下周一(16号)触发.如果15号正好在工作日(周一至周五)，则就在该天触发。如果指定格式为"1W",它则表示每月1号往后最近的工作日触发。如果1号正是周六，则将在3号下周一触发。(注，"W"前只能设置具体的数字,不允许区间"-").

# 序号(表示每月的第几个周几)，例如在周字段上设置"6#3"表示在每月的第三个周六.注意如果指定"#5",正好第五周没有周六，则不会触发该配置(用在母亲节和父亲节再合适不过了) ；
```

常用表达式

```
0 0 10,14,16 * * ? 每天上午10点，下午2点，4点
0 0/30 9-17 * * ? 朝九晚五工作时间内每半小时
0 0 12 ? * WED 表示每个星期三中午12点
"0 0 12 * * ?" 每天中午12点触发
"0 15 10 ? * *" 每天上午10:15触发
"0 15 10 * * ?" 每天上午10:15触发
"0 15 10 * * ? *" 每天上午10:15触发
"0 15 10 * * ? 2005" 2005年的每天上午10:15触发
"0 * 14 * * ?" 在每天下午2点到下午2:59期间的每1分钟触发
"0 0/5 14 * * ?" 在每天下午2点到下午2:55期间的每5分钟触发
"0 0/5 14,18 * * ?" 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发
"0 0-5 14 * * ?" 在每天下午2点到下午2:05期间的每1分钟触发
"0 10,44 14 ? 3 WED" 每年三月的星期三的下午2:10和2:44触发
"0 15 10 ? * MON-FRI" 周一至周五的上午10:15触发
"0 15 10 15 * ?" 每月15日上午10:15触发
"0 15 10 L * ?" 每月最后一日的上午10:15触发
"0 15 10 ? * 6L" 每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6L 2002-2005" 2002年至2005年的每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6#3" 每月的第三个星期五上午10:15触发
```

2.3 秒杀商品压入缓存实现

2.3.1 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。

2.3.2 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，在给出的DateUtil包里已经实现，代码如下：

```
/***
 * 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;
}

/***
 * 指定时间往后N个时间间隔
 * @param hours
 * @return
 */
public static List<Date> getDates(int hours) {
    List<Date> dates = new ArrayList<Date>();
    //循环12次
    Date date = toDayStartHour(new Date()); //凌晨
    for (int i = 0; i <hours ; i++) {
        //每次递增2小时,将每次递增的时间存入到List<Date>集合中
        dates.add(addDateHour(date,i*2));
    }
    return dates;
}
```

2.3.3 查询秒杀商品导入Reids

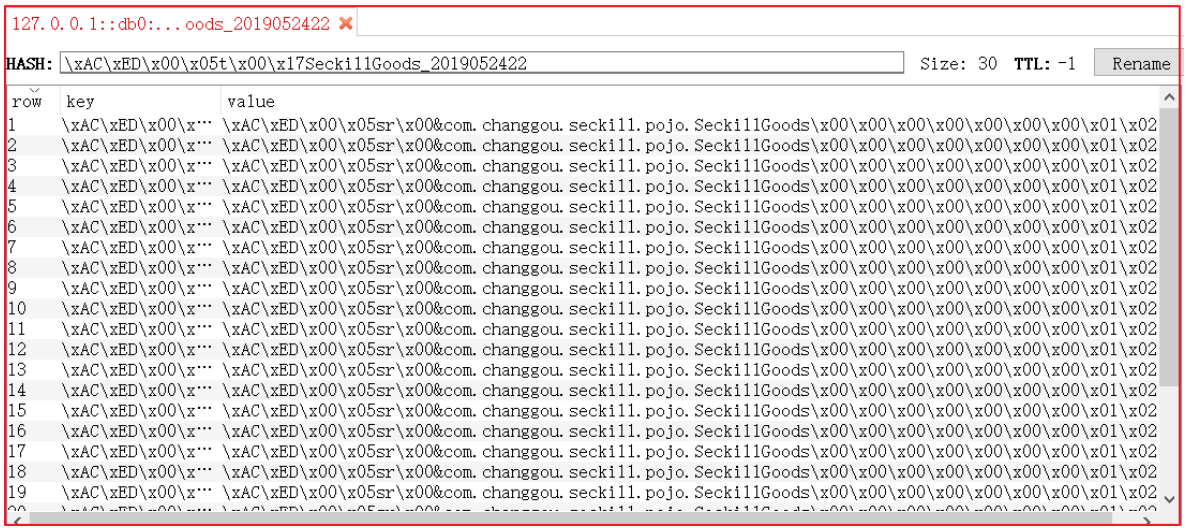
我们可以写个定时任务，查询从当前时间开始，往后延续4个时间菜单间隔，也就是一共只查询5个时间段抢购商品数据，并压入缓存，实现代码如下：

修改SeckillGoodsPushTask的loadGoodsPushRedis方法，代码如下：

```
@Component
public class SeckillGoodsPushTask {

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;

    @Autowired
    private RedisTemplate redisTemplate;

    /****
     * 定时任务方法
     * 0/30 * * * * ?:从每分钟的第0秒开始执行，每过30秒执行一次
     */
    @Scheduled(cron = "0/30 * * * * ?")
    public void loadGoodsPushRedis(){
        //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);

            //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);
            }
        }
    }
}
```

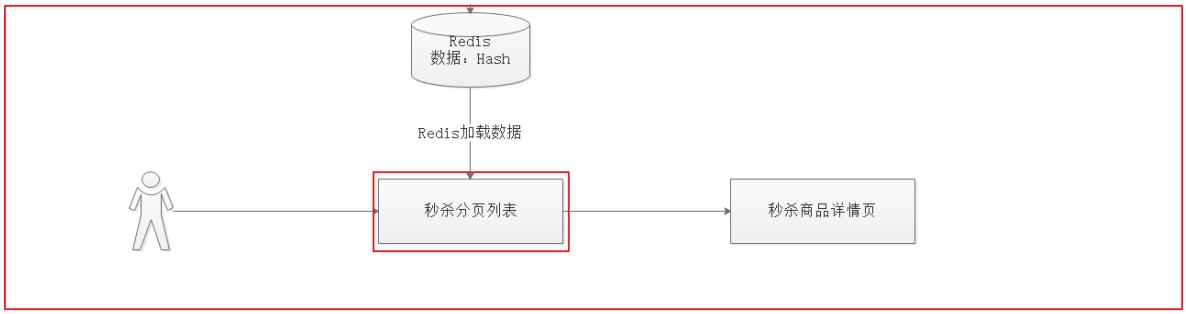
Redis数据如下：



```
```

作业：使用elastic-job做定时任务

3 秒杀频道页[列表页]



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）

3.1 秒杀时间菜单

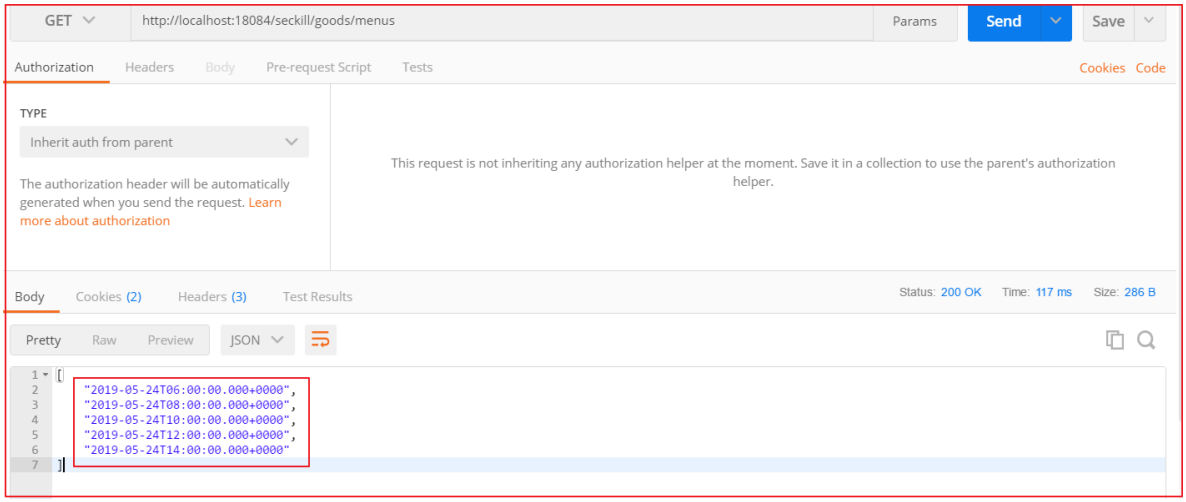
如上图，时间菜单需要根据当前时间动态加载，时间菜单的计算上面功能中已经实现，在DateUtil工具包中。我们只需要将时间菜单获取，然后响应到页面，页面根据对应的数据显示即可。

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

```
@RestController
@CrossOrigin
@RequestMapping(value = "/seckill/goods")
public class SeckillGoodsController {

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

使用Postman测试，效果如下：

http://localhost:18091/seckill/goods/menus



3.2 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。

3.2.1 业务层

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

```
public interface SeckillGoodsService {

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

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

```
@Service
public class SeckillGoodsServiceImpl implements SeckillGoodsService {

    @Autowired
    private RedisTemplate redisTemplate;

    /***
     * 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();
    }
}
```

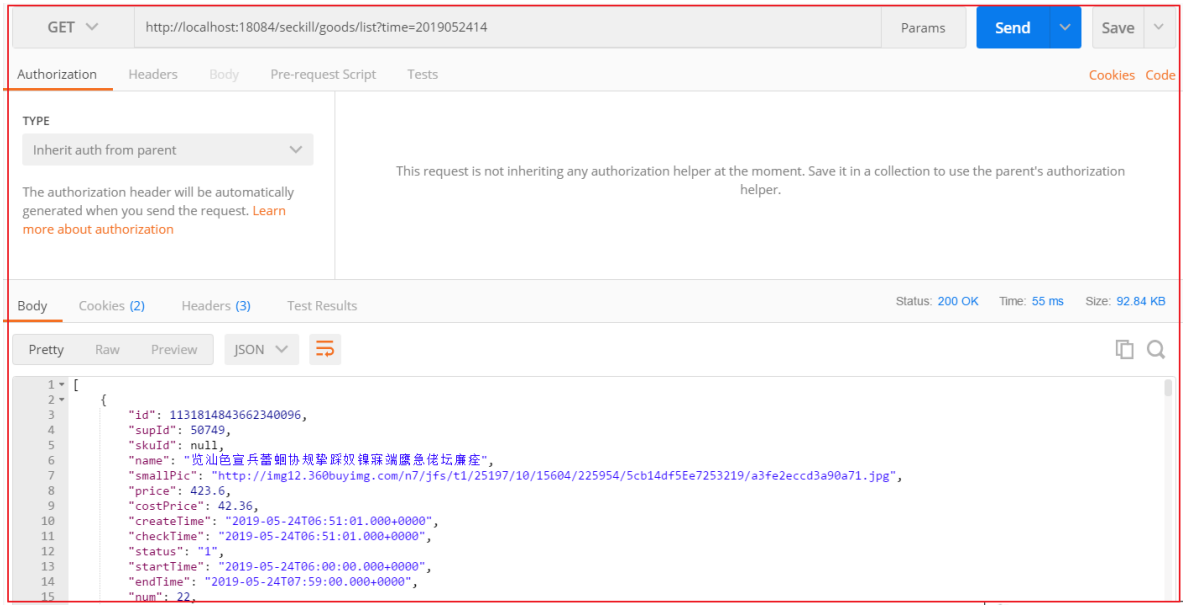
3.2.2 控制层

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

```
@Autowired
private SeckillGoodsService seckillGoodsService;

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

使用Postman测试，效果如下：

http://localhost:18084/seckill/goods/list?time=2019052414



作业：使用Lua脚本+nginx实现查询，效率更高。

4 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。

4.1 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
 * @return
 */
@Override
public SeckillGoods one(String time, Long id) {
    return (SeckillGoods)
redisTemplate.boundHashOps("SeckillGoods_"+time).get(id);
}
```

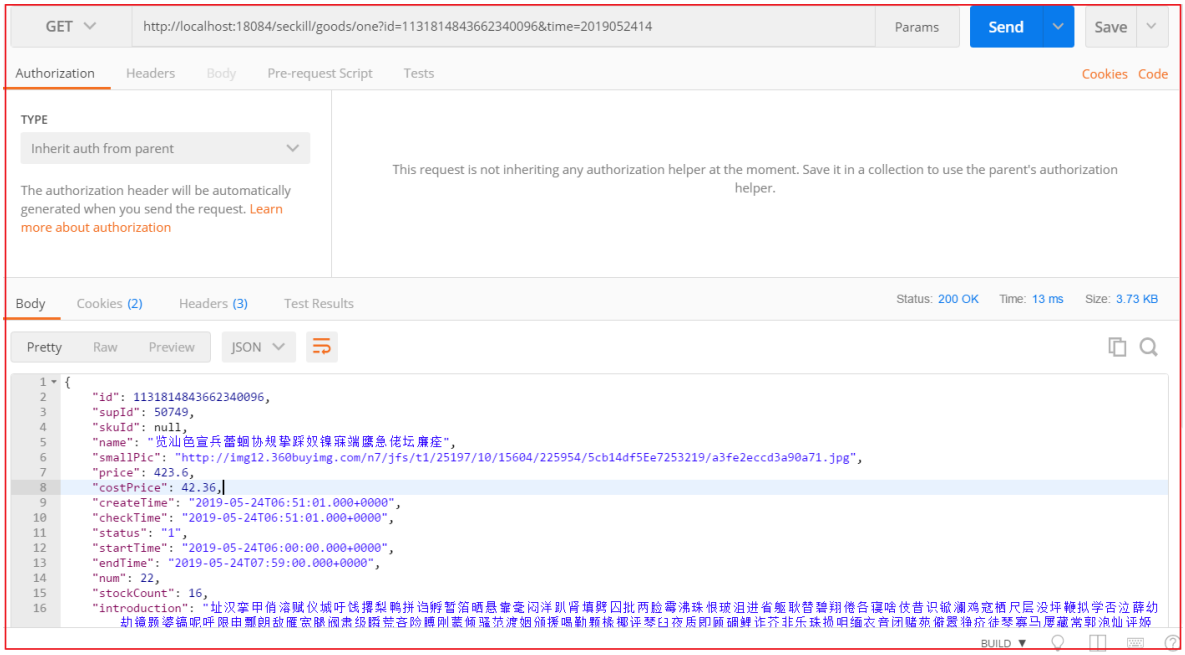
4.2 控制层

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:/seckill/goods/one
 * 根据ID查询商品详情
 * 调用Service查询商品详情
 * @param time
 * @param id
 */
@RequestMapping(value = "/one")
public SeckillGoods one(String time,String id){
    //调用Service查询商品详情
    return seckillGoodsService.one(time,id);
}
```

使用Postman测试，效果如下：

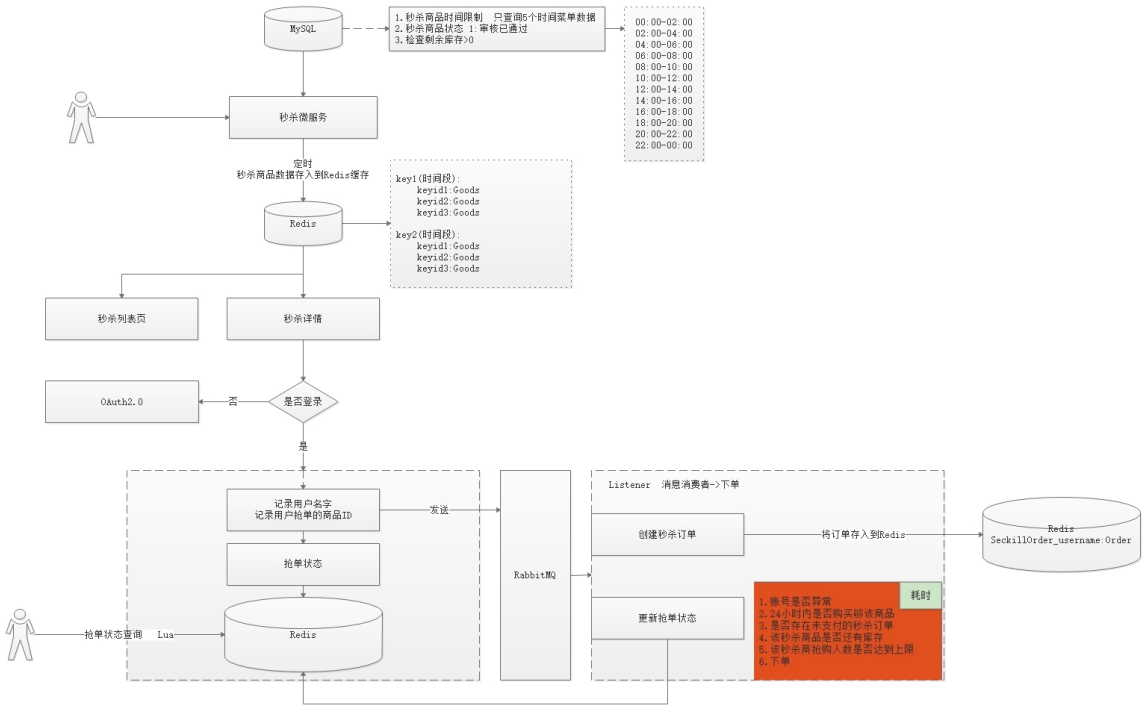
http://localhost:18084/seckill/goods/one?id=1131814843662340096&time=2019052414



5 MQ抢单

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。



5.1 队列配置

在`changgou-service-seckill`工程中创建队列`com.changgou.seckill.mq.SeckillOrderConfig`，代码如下：

```
@Configuration
public class SeckillOrderConfig {

    /***
     * 创建一个交换机
     */
    @Bean
    public DirectExchange seckillOrderExchange(){
        return new DirectExchange("seckillOrderExchange",true,false);
    }
    /***
     * 创建一个队列
     */
    @Bean
    public Queue seckillOrderQueue(){
        return new Queue("seckillOrderQueue");
    }

    /***
     * 队列与交换机绑定
```

```
     */
    @Bean
    public Binding queueBindingExchange(Queue seckillOrderQueue, Exchange seckillOrderExchange){
        return BindingBuilder.bind(seckillOrderQueue).to(seckillOrderExchange).with("seckillOrderQueue").noargs();
    }
}
```

5.2 抢单记录封装

在`changgou-seckill-api`中封装一个对象，记录用户抢单的信息以及状态，对象信息如下：

```
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:秒杀失败,4:支付完成
    private Integer status;
    //秒杀的商品ID
    private String goodsId;
    //应付金额
    private Float money;
    //订单号
    private String orderId;
    //时间段
    private String time;

    //...get..set...
}
```

5.3 抢单实现

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

```java
public interface SeckillOrderService {

    /***
     * 秒杀抢单
     * @param username : 抢单用户
     * @param id : 商品ID
     * @param time : 时间 20200106
     */
    Boolean add(String username,String id,String time);
}
```

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

```java
@Service
public class SeckillOrderServiceImpl implements SeckillOrderService {

    @Autowired
    private SeckillOrderMapper seckillOrderMapper;

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    /***
     * 秒杀抢单
     * @param username : 抢单用户
     * @param id : 商品ID
     * @param time : 时间 20200106
     */
    @Override
    public Boolean add(String username, String id, String time) {
        //封装抢单信息
        SeckillStatus seckillStatus = new SeckillStatus();
        seckillStatus.setUsername(username);
        seckillStatus.setCreateTime(new Date());
        seckillStatus.setStatus(1);     //排队中
        seckillStatus.setGoodsId(id);   //商品ID
        seckillStatus.setTime(time);    //商品所在的key的时间后缀

        //状态信息
        String statusJson = JSON.toJSONString(seckillStatus);

        //队列削峰
        //将抢单信息发送到RabbitMQ    交换机、队列、队列与交换机绑定
        rabbitTemplate.convertAndSend("seckillOrderExchange",
"seckillOrderQueue",statusJson);

        //将抢单信息存入到Redis  key:value
        //                       key=SeckillStatus_username
```

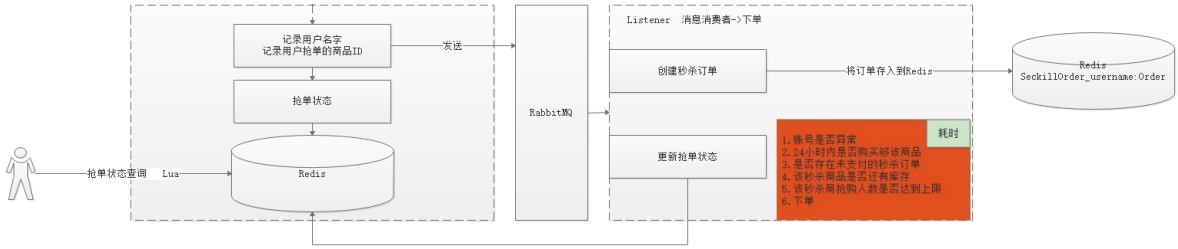
```
        //                          value=seckillStatus
        String key = "SeckillStatus_"+username;
        stringRedisTemplate.boundValueOps(key).set(statusJson);
        return true;
    }
}
```

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

```
@RestController
@CrossOrigin
@RequestMapping(value = "/seckill/order")
public class SeckillOrderController {

    @Autowired
    private SeckillOrderService seckillOrderService;

    /***
     * 秒杀抢单
     * @param id : 商品ID
     * @param time : 时间 20200106
     */
    @GetMapping(value = "/add")
    public Result add(String time,String id){
        String username="zhaoliu";
        //调用Service实现排队抢单
        Boolean bo = seckillOrderService.add(username, id, time);
        if(bo){
            return new Result(true,StatusCode.OK,"正在排队！");
        }
        return new Result(false,StatusCode.OK,"服务器繁忙！");
    }
}
```

6 MQ消费者下单

6.1 实现思路分析



用户抢单的时候，会将抢单的商品信息存入到Reids，也会将抢单信息发送给RabbitMQ，我们可以创建消费者获取用户抢单信息，并根据抢单信息创建订单，将订单信息存入到Redis缓存。

6.2 MQ消费者下单

在`changgou-service-seckill`中添加消息监听读取消息，实现下单操作，并将订单存入到Redis缓存，代码如下：

```
@Component
@RabbitListener(queues = {"seckillOrderQueue"})
public class SeckollOrderConsumer {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    @Autowired
    private IdWorker idWorker;

    /****
     * 下单操作
     */
    @RabbitHandler
    public void readOrderMessage(String message){
        //获取抢单信息   username,id,time
        SeckillStatus seckillStatus =
JSON.parseObject(message,SeckillStatus.class);

        //有可能存在未支付的订单  key1=SeckillOrder    key2=username
        Object order =
redisTemplate.boundHashOps("SeckillOrder").get(seckillStatus.getUsername());

        if(order!=null){
            //更新抢单状态
            System.out.println("-------存在未支付订单，不允许排队抢单----");
            return;
        }
        //商品是否有库存(存在缺陷)->是否超卖
        SeckillGoods seckillGoods = (SeckillGoods)
redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).get(seck
illStatus.getGoodsId());

        if(seckillGoods!=null && seckillGoods.getStockCount()>0){
            //创建订单对象
            SeckillOrder seckillOrder = new SeckillOrder();
            seckillOrder.setId("No"+idWorker.nextId());
            seckillOrder.setSeckillId(seckillStatus.getGoodsId());
            seckillOrder.setMoney(seckillGoods.getCostPrice());
            seckillOrder.setUserId(seckillStatus.getUsername());
            seckillOrder.setCreateTime(seckillStatus.getCreateTime());
            seckillOrder.setStatus("0");    //未支付
            //将数据存入到Redis

 redisTemplate.boundHashOps("SeckillOrder").put(seckillStatus.getUsername(),
seckillOrder);

            //库存递减
            seckillGoods.setStockCount(seckillGoods.getStockCount()-1);
```

```
47
 redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).put(sec
killGoods.getId(),seckillGoods);
48
49              //更新抢单状态
50
 seckillStatus.setMoney(Float.valueOf(seckillGoods.getCostPrice()));
51              seckillStatus.setOrderId(seckillOrder.getId());
52              seckillStatus.setStatus(2); //抢单成功，  等待支付
53
 stringRedisTemplate.boundValueOps("SeckillStatus_"+seckillStatus.getUsernam
e()).set(JSON.toJSONString(seckillStatus));
54          }
55      }
56  }
```

6.3 抢单状态查询

为了避免对后台程序造成过多的，我们尽可能的将一些查询使用Lua脚本实现，将大部分请求的并发量在Nginx端就能实现控制，查询抢单状态这类操作，我们可以使用Nginx+Lua实现，避免了对后端程序造成一定压力。

在虚拟机中创建`/root/lua/read_status.lua`脚本，代码如下：

```
1  ngx.header.content_type="application/json;charset=utf8"
2  local uri_args = ngx.req.get_uri_args();
3  local name = uri_args["name"];
4
5  local redis = require("resty.redis");
6  local red = redis:new()
7  red:set_timeout(2000)
8  red:connect("192.168.211.132", 6379)
9  local rescontent=red:get("SeckillStatus_"..name);
10 ngx.say(rescontent)
11 red:close()
```

上述脚本大致思路如下：

```
1  1.用户请求携带用户名参数name
2  2.使用lua获取name参数
3  3.链接redis
4  4.使用lua根据获取的name参数取redis中获取订单状态信息
5  5.将状态信息返回页面，并关闭redis链接
```

修改虚拟机中`/usr/local/openresty/nginx/conf`的nginx.conf配置文件,在data-changgou-java.itheima.net域名配置下添加如下配置：

```
#查询用户抢单状态
location /order/status {
   add_header Access-Control-Allow-Origin *;
   add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
    content_by_lua_file /root/lua/read_status.lua;
}
```

上图代码如下：

```
#查询用户抢单状态
location /order/status {
   add_header Access-Control-Allow-Origin *;
   add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
    content_by_lua_file /root/lua/read_status.lua;
}
```

重新加载nginx配置文件访问`<http://data-changgou-java.itheima.net/order/status?name=zhaoliu>`效果如下：