第1天-秒杀

学习目标

目标1：秒杀业务分析

目标2：秒杀商品压入Redis缓存

目标3：Spring定时任务了解

目标4：秒杀商品频道页实现

目标5：秒杀商品详情页实现

目标6：倒计时实现

目标7：通用跳转控制实现

目标8：下单实现

第1章 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）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。
（4）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。
（5）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。
（6）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,
```

```
  `goods_id` bigint(20) DEFAULT NULL COMMENT 'spu ID',
  `item_id` bigint(20) DEFAULT NULL COMMENT 'sku ID',
  `title` 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',
  `seller_id` varchar(100) DEFAULT NULL COMMENT '商家ID',
  `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;
```

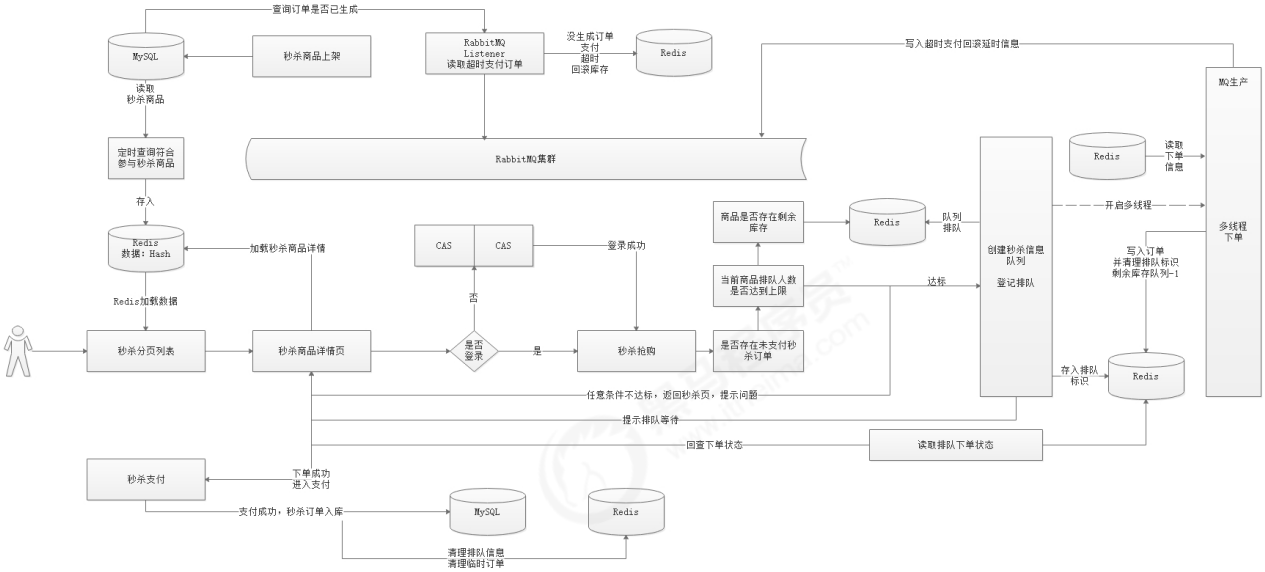
秒杀订单表

```
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',
  `seller_id` varchar(50) DEFAULT NULL COMMENT '商家',
  `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;
```

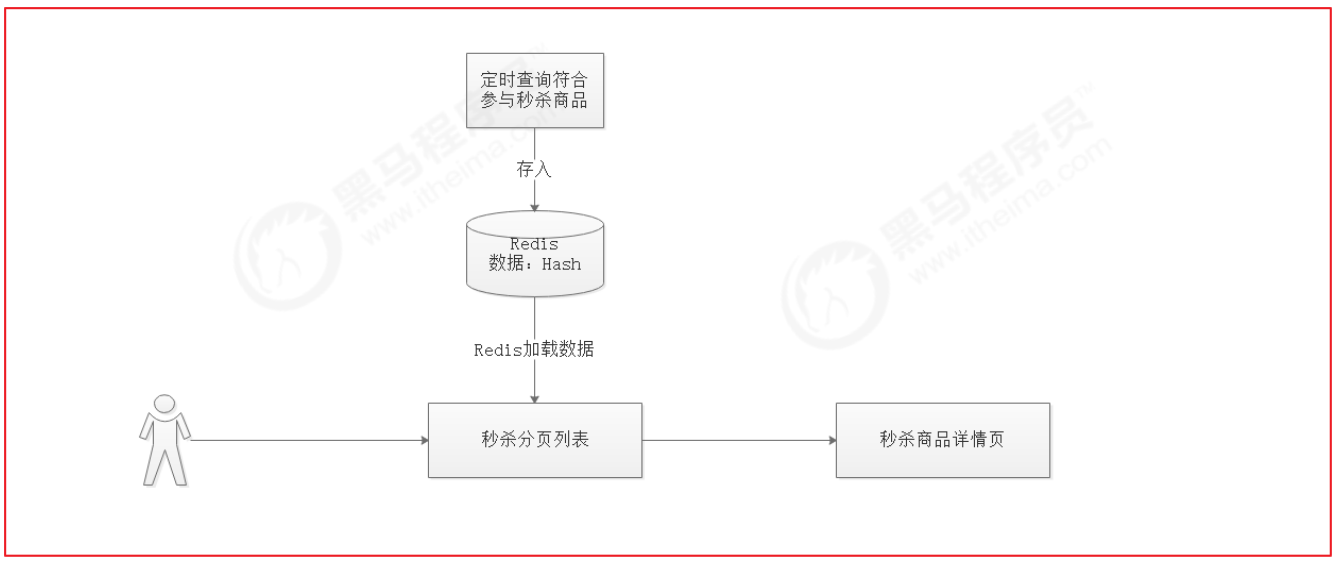
1.3 秒杀需求分析

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：



第2章 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()获取结婚数据。

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。

2.1.1 搭建秒杀服务工程

搭建qingcheng_service_seckill，作为秒杀工程的服务提供工程。

2.1.1.1 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>qingcheng_parent</artifactId>
        <groupId>com.qingcheng</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>qingcheng_service_seckill</artifactId>
    <packaging>war</packaging>

    <!--依赖包-->
    <dependencies>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_common_service</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.amqp</groupId>
            <artifactId>spring-rabbit</artifactId>
            <version>2.1.4.RELEASE</version>
        </dependency>
    </dependencies>

</project>
```

2.1.1.2 web.xml

web.xml配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <!-- 加载spring容器 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:applicationContext*.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
</web-app>
```

2.1.1.3 applicationContext-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">

    <!--包扫描-->
    <context:component-scan base-package="com.qingcheng" />

    <!--雪花ID生成器-->
    <bean id="idWorker" class="com.qingcheng.util.IdWorker">
        <constructor-arg index="0" value="2"></constructor-arg>
        <constructor-arg index="1" value="2"></constructor-arg>
    </bean>

</beans>
```

2.1.1.4 properties配置文件

db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/qingcheng_seckill?characterEncoding=UTF8
jdbc.username=root
jdbc.password=123456
```

dubbo.properties

```
dubbo.port=20888
dubbo.application=seckill
```

2.1.2 定时任务配置

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)开启定时任务注解驱动
2)在定时任务类的指定方法上加上@Scheduled开启定时任务
3)定时任务表达式：使用cron属性来配置定时任务执行时间
```

2.1.2.1 开启定时任务注解驱动

在秒杀服务工程中添加applicationContext-timer.xml配置文件，并开启定时任务注解驱动，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/task
       http://www.springframework.org/schema/task/spring-task.xsd">

    <!--开启注解驱动-->
    <task:annotation-driven executor="seckillExecutor" scheduler="seckillScheduler"/>
    <task:executor id="seckillExecutor" pool-size="10"/>
    <task:scheduler id="seckillScheduler" pool-size="10"/>
</beans>
```

2.1.2.2 定时任务方法配置

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

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

2.1.2.3 定时任务常用时间表达式(了解)

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

/ 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号是周末，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
```

2.2.3 秒杀商品压入缓存实现

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，。

2.2.3.1 创建Dao操作类

创建一个com.qingcheng.dao.SeckillGoodsMapper接口，并让该接口继承Mapper，代码如下：

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

2.2.3.2 数据压入到Redis缓存

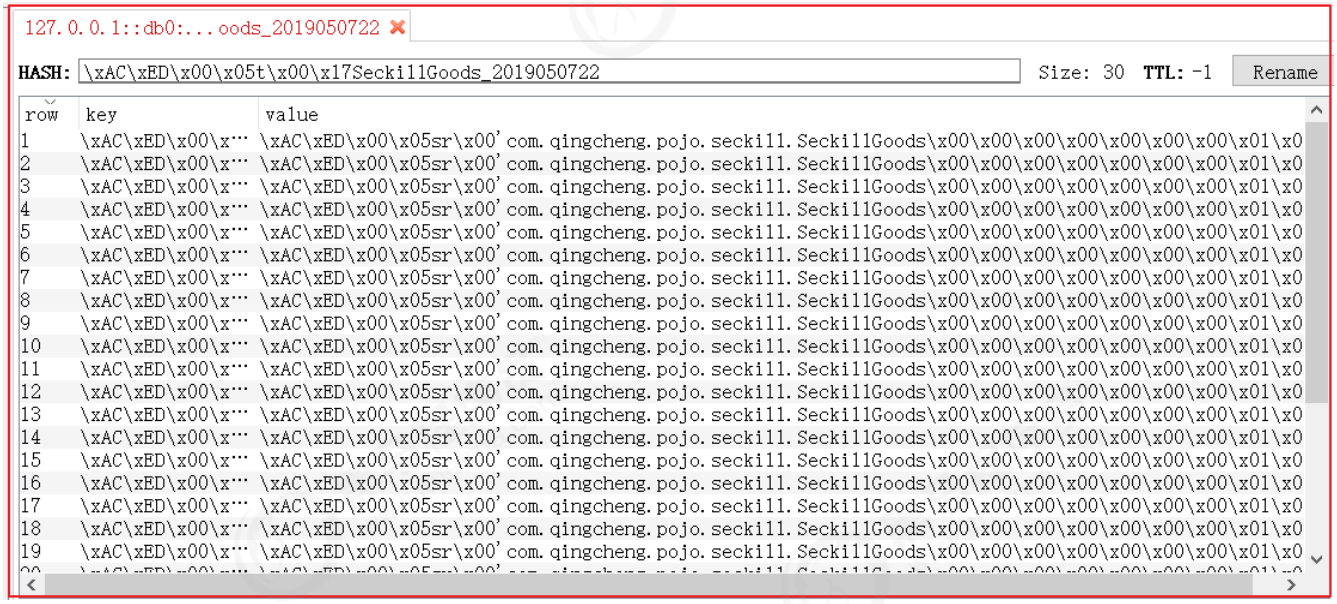
修改com.qingcheng.timer.SeckillGoodsPushTask类的loadGoodsPushRedis方法，按照如上步骤实现秒杀商品压入缓存：

```
@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.date2Str(startTime);

            //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
```

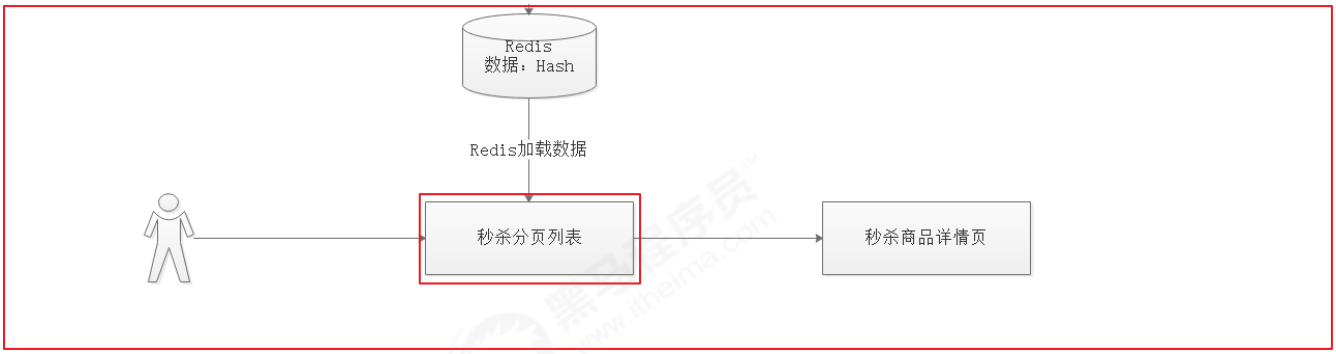
```
            for (SeckillGoods seckillGood : seckillGoods) {

redisTemplate.boundHashOps("SeckillGoods_"+extName).put(seckillGood.getId(),seckillGood);
            }
        }
    }
}
```

测试

运行起来，并通过RedisDesktopManager可以查看到如下数据,说明数据存入到Redis缓存定时操作成功。



第3章 秒杀商品-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）

3.1 业务层实现

在qingcheng_interface创建指定接口，在qingcheng_service_seckill实现该接口，用来实现秒杀商品频道页数据的显示操作。

3.1.1 业务接口层

在qingcheng_interface下创建com.qingcheng.service.seckill.SeckillGoodsService接口，并添加查询方法

```
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);

}
```

3.1.2 业务层实现类

创建com.qingcheng.service.impl.SeckillGoodsServiceImpl，实现SeckillGoodsService接口，并实现订单列表查询方法，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 表现层

需要搭建一个表现层工程，用于调用Service层获取秒杀商品列表数据。

3.2.1 表现层工程搭建

搭建一个web工程，名叫qingcheng_web_seckill，用于实现秒杀频道页显示。

3.2.1.1 pom.xml

pom.xml配置如下：

```
<?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>qingcheng_parent</artifactId>
        <groupId>com.qingcheng</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <packaging>war</packaging>
    <artifactId>qingcheng_web_seckill</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.qingcheng</groupId>
            <artifactId>qingcheng_common_cas</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

3.2.1.2 web.xml

web.xml配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <!-- 解决post乱码 -->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>utf-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
```

```
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 指定加载的配置文件 ，通过参数contextConfigLocation加载-->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath*:applicationContext*.xml</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>seckill-index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

配置dubbo.properties

```
dubbo.application=seckill-web
```

3.2.1.3 引入静态资源

将css、data、fonts、img、js、seckill-item.html、seckill-index.html引入到工程的webapp目录下。

3.3 频道页实现分析

频道页这里需要显示不同的时间段的商品信息，然后点击不同时间段，查询对应的商品数据，实现过程大致分为2个步骤：

1)时间菜单
2)加载对应菜单数据

3.4.1 时间菜单分析

每2个小时就会切换一次抢购活动，所以商品发布的时候，我们建议将时间定格在2小时内抢购，每次发布商品的时候，商品抢购开始时间和结束时间是这2小时的边界。

每2小时会有一批商品参与抢购，所以我们可以将24小时切分为12个菜单，每个菜单都是个2小时的时间段，当前选中的时间菜单需要根据当前时间判断，判断当前时间属于哪个秒杀时间段，然后将该时间段作为选中的第1个时间菜单。

3.4.2 加载对应时间段数据

进入页面后，到后台查询时间段菜单，然后根据第1个菜单的时间段时间去后台Redis中查询秒杀商品集合，并显示到页面，页面每次点击切换不同时间段菜单的时候，都将时间段传入到后台，后台根据时间段获取对应的秒杀商品集合。

3.5 时间菜单实现

时间菜单显示，先运算出每2小时一个抢购，就需要实现12个菜单，可以先计算出每个时间的临界值，然后根据当前时间判断需要显示12个时间段菜单中的哪个菜单，再在该时间菜单的基础之上往后挪4个菜单，一个显示5个菜单。

3.5.1 时间菜单计算

修改DateUtil，在该类中添加一个获取时间菜单运算的方法，返回每个时间段菜单的开始时间，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 = new ArrayList<Date>();
    //循环12次
    Date date = toDayStartHour(new Date()); //凌晨
    for (int i = 0; i <12 ; 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));
    }

    //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;
}
```

编写控制器，调用上面的方法，获取时间菜单,在com.qingcheng.controller.SeckillGoodsController添加如下方法：

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

3.5.2 加载时间菜单

```
<script>
    var app = new Vue({
        el: '#app',
```

```
        data: {
            dateMenus: [],
        },
        methods: {
            //获取时间菜单
            loadMenus: function () {
                //请求后台，获取时间菜单
                axios.get('/seckill/goods/menus.do').then(function (response) {
                    //将时间菜单存入到一个变量中
                    app.dateMenus = response.data;
                })
            },
        },
        created: function () {
            //加载菜单
            this.loadMenus();
        }
    })
</script>
```

页面循环显示时间菜单

```
<!--秒杀时间-->
<div class="sectime">
    <div v-for="item in dateMenus">
        <div class="time-clock">{{item}}</div>
        <div class="time-state-on">
            <span class="on-text">快抢中</span>
            <span class="on-over">距离结束：2019年5月7日15:43:05</span>
        </div>
    </div>
</div>
```

3.5.3 时间格式化

2019-05-07 16:00:00 | 2019-05-07 18:00:00 | 2019-05-07 20:00:00 | 2019-05-07 22:00:00 | 2019-05-08 00:00:00

上面菜单循环输出后，会出现如上图效果，时间格式全部不对，我们需要引入一个moment.min.js来格式化时间。

引入moment.min.js

```
<script src="/js/moment.min.js"></script>
```

在js中添加一个过滤器,代码如下：

```
//过滤器
Vue.filter("dateFilter", function(date, formatPattern){
    return moment(date).format(formatPattern || "YYYY-MM-DD HH:mm:ss");
});
```

页面输出

```
<div class="time-clock">{{item | dateFilter('HH:mm')}}</div>
```

效果如下：

3.5.4 选中实现

我们需要要实现的是如下效果，让第一个菜单选中，并加载第一个菜单对应的数据。

我们可以先定义一个ctime=0，用来记录当前选中的菜单下标，因为默认第一个选中，第一个下标为0，所以初始值为0，每次点击对应菜单的时候，将被点击的菜单的下标值赋值给ctime,然后在每个菜单上判断，下标=ctime则让该菜单选中。

定义ctime=0

```
data: {
    dateMenus: [],
    ctime:0,      //当前时间菜单选中的下标
},
```

页面样式控制：

```
<!--秒杀时间-->
<div class="sectime">
    <div v-for="(item,index) in dateMenus" :class="['item-time',index==ctime? 'active':'']" @click="ctime=index;">
        <div class="time-clock">{{item | dateFilter('HH:mm')}}</div>
        <div class="time-state-on">
            <span class="on-text">快抢中</span>
            <span class="on-over">距离结束：2019年5月8日15:55:18</span>
        </div>
    </div>
</div>
```

上图代码如下：

```
<div v-for="(item,index) in dateMenus" :class="['item-time',index==ctime? 'active':'']"
@click="ctime=index;">
    <div class="time-clock">{{item | dateFilter('HH:mm')}}</div>
    <div class="time-state-on">
        <span class="on-text">快抢中</span>
        <span class="on-over">距离结束：2019年5月8日15:55:18</span>
    </div>
</div>
```

3.5.5 倒计时时间

倒计时这里，第一个是距离结束时间倒计时，后面的4个都是距离开始倒计时，每个倒计时其实就是2个时差，分析如下：

```
第1个时差：第2个抢购开始时间-当前时间，距离结束时间
第2个时差：第2个抢购开始时间-当前时间，距离开始时间
第3个时差：第3个抢购开始时间-当前时间，距离开始时间
第4个时差：第4个抢购开始时间-当前时间，距离开始时间
第5个时差：第5个抢购开始时间-当前时间，距离开始时间
```

3.5.5.1 倒计时实现

我们可以先定义5个变量，然后每过1秒中，让5个变量时间都减1秒实现倒计时操作。

周期执行函数用法如下：

```
window.setInterval(function(){//要做的事},1000);
```

结束执行周期函数用法如下：

```
window.clearInterval(timers);
```

倒计时实现代码如下：

```
<script>
    //过滤器
    Vue.filter("dateFilter", function(date, formatPattern){
        return moment(date).format(formatPattern || "YYYY-MM-DD HH:mm:ss");
    });

    var app = new Vue({
        el: '#app',
        data: {
            dateMenus: [],
            ctime:0,      //当前时间菜单选中的下标
            alltimes:[],//每个倒计时的时间差.单位：毫秒
        },
        methods: {
            //获取时间菜单
            loadMenus: function () {
                //请求后台，获取时间菜单
                axios.get('/seckill/goods/menus.do').then(function (response) {
                    //将时间菜单存入到一个变量中
                    app.dateMenus = response.data;

                    //循环所有时间菜单
                    for(var i=0;i<app.dateMenus.length;i++){
                        //运算每个时间菜单倒计时时间差
                        if(i==0){
                            app.$set(app.alltimes,i,new Date(app.dateMenus[(i+1)]).getTime()-new Date().getTime());
                        }else{
                            app.$set(app.alltimes,i,new Date(app.dateMenus[i]).getTime()-new Date().getTime());
                        }
                    }

                    //周期执行函数 window.setInterval(function(){执行操作},1000)
                    let timers = window.setInterval(function () {
                        for(var i=0;i<app.alltimes.length;i++){
                            //时间递减
                            app.$set(app.alltimes,i,app.alltimes[i]-1000);

                            if(app.alltimes[i]<=0){
                                //停止倒计时
                                window.clearInterval(timers);
                                //当任意一个时间<=0的时候，需要重新刷新菜单，并刷新对应的数据
                                app.loadMenus();
                            }
                        }
                    },1000);
                })
            },
        },
        created: function () {
            //加载菜单
            this.loadMenus();
        }
    })
</script>
```

定义5个倒计时时间差

第1个时差：第2个抢购开始时间-当前时间，距离结束时间

其他的时差：第N个抢购开始时间-当前时间，距离开始时间

上图代码如下：

```
<script>
    //过滤器
    Vue.filter("dateFilter", function(date, formatPattern){
        return moment(date).format(formatPattern || "YYYY-MM-DD HH:mm:ss");
    });
```

```
    var app = new Vue({
        el: '#app',
        data: {
            dateMenus: [],
            ctime:0,     //当前时间菜单选中的下标
            alltimes:[],//每个倒计时的时间差.单位：毫秒
        },
        methods: {
            //获取时间菜单
            loadMenus: function () {
                //请求后台，获取时间菜单
                axios.get('/seckill/goods/menus.do').then(function (response) {
                    //将时间菜单存入到一个变量中
                    app.dateMenus = response.data;

                    //循环所有时间菜单
                    for(var i=0;i<app.dateMenus.length;i++){
                        //运算每个时间菜单倒计时时间差
                        if(i==0){
                            app.$set(app.alltimes,i,new
Date(app.dateMenus[(i+1)]).getTime()-new Date().getTime());
                        }else{
                            app.$set(app.alltimes,i,new Date(app.dateMenus[i]).getTime()-new
Date().getTime());
                        }
                    }

                    //周期执行函数 window.setInterval(function(){执行操作},1000)
                    let timers = window.setInterval(function () {
                        for(var i=0;i<app.alltimes.length;i++){
                            //时间递减
                            app.$set(app.alltimes,i,app.alltimes[i]-1000);

                            if(app.alltimes[i]<=0){
                                //停止倒计时
                                window.clearInterval(timers);
                                //当任意一个时间<=0的时候，需要重新刷新菜单，并刷新对应的数据
                                app.loadMenus();
                            }
                        }
                    },1000);

                })
            },
        },
        created: function () {
            //加载菜单
            this.loadMenus();
        }
    })
</script>
```

页面根据对应的菜单下标，到倒计时时差数组中根据下标取数据，而只有第1个菜单属于距离结束时间倒计时，其他的都是距离开始倒计时，如下图：

```
<div class="time-state-on">
    <span class="on-text" v-if="index==0">快抢中</span>
    <span class="on-over" v-if="index==0">距离结束：{{alltimes[index]}}</span>

    <span class="on-text" v-if="index>0">即将开始</span>
    <span class="on-over" v-if="index>0">距离开始：{{alltimes[index]}}</span>
</div>
```

上图代码如下：

```
<div class="time-state-on">
    <span class="on-text" v-if="index==0">快抢中</span>
    <span class="on-over" v-if="index==0">距离结束：{{alltimes[index]}}</span>

    <span class="on-text" v-if="index>0">即将开始</span>
    <span class="on-over" v-if="index>0">距离开始：{{alltimes[index]}}</span>
</div>
```

效果如下：

3.5.5.2 时间换算

上面实现了秒杀倒计时，但是我们要的效果是将毫秒转成时、分、秒，而不是毫秒递减，而将毫秒时差转成时分秒，可以先判断下有多少小时，余数再判断有多少分钟，剩下的余数还有多少秒即可，实现代码如下：

```
//将毫秒转换成时分秒
timedown:function(num) {
    var oneSecond = 1000;
    var oneMinute=oneSecond*60;
    var oneHour=oneMinute*60
    //小时
    var hours =Math.floor(num/oneHour);
    //分钟
    var minutes=Math.floor((num%oneHour)/oneMinute);
    //秒
    var seconds=Math.floor((num%oneMinute)/oneSecond);
    //拼接时间格式
    var str = hours+':'+minutes+':'+seconds;
    return str;
},
```

页面只需要将时间差值传入该方法即可，代码如下：

```
<div class="time-state-on">
    <span class="on-text" v-if="index==0">快抢中</span>
    <span class="on-over" v-if="index==0">距离结束：{{timedown(alltimes[index])}}</span>

    <span class="on-text" v-if="index>0">即将开始</span>
    <span class="on-over" v-if="index>0">距离开始：{{timedown(alltimes[index])}}</span>
</div>
```

上图代码如下：

```
{{timedown(alltimes[index])}}
```

效果如下：

3.6 秒杀时间段商品加载

秒杀时间段商品加载时根据商品每个时间段去加载对应的商品信息，在之前的定时任务已经将对应的商品信息存入到了Redis中，我们可以根据秒杀时间菜单的开始时间去Redis中查询数据。

3.6.1 控制层实现

创建com.qingcheng.controller.SeckillGoodsController，用于实现人机交互，获取秒杀商品列表,添加list方法，如下代码：

```
/****
 * 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 2019-5-7 16:00，可以调用DateUtils，将它转成2019050716格式
 */
@RequestMapping(value = "/list")
public List<SeckillGoods> list(String time){
    //调用Service查询数据
    return seckillGoodsService.list(DateUtil.formatStr(time));
}
```

3.6.2 页面对接

添加一个goodslist存储当前时间菜单对应的秒杀商品集合，如下：

```
data: {
    goodslist: [],      存储秒杀商品集合
    dateMenus: [],
    ctime:0,      //当前时间菜单选中的下标
    alltimes:[],//每个倒计时的时间差.单位：毫秒
},
```

上图代码如下：

```
goodslist: [],
```

编写一个方法，查询对应时间菜单的秒杀商品，代码如下：

```
//加载当前时间对应的秒杀商品
searchList:function (time) {
    axios.get('/seckill/goods/list.do?time='+time).then(function (response) {
        app.goodslist=response.data;
    })
},
```

页面可以在每次加载菜单的时候，将第1个菜单的其实时间传入到后台，来查询对应的数据，代码如下：

```
//获取时间菜单
loadMenus: function () {
    //请求后台，获取时间菜单
    axios.get('/seckill/goods/menus.do').then(function (response) {
        //将时间菜单存入到一个变量中
        app.dateMenus = response.data;

        //查询当前时间段对应的秒杀商品
        app.searchList(app.dateMenus[0]);

        ///...略
    })
},
```

上图代码如下：

```
//查询当前时间段对应的秒杀商品
app.searchList(app.dateMenus[0]);
```

页面参数绑定

修改seckill-index.html页面，实现参数数据的绑定，如下：

```
<!--商品列表-->
<div class="goods-list" id="app1">
    <ul class="seckill" id="seckill">
        <li class="seckill-item" v-for="item in goodslist" :key="item">
            <div class="pic">
                <img :src="item.smallPic" alt=''>
            </div>
            <div class="intro">
                <span>{{item.title}}</span>
            </div>
            <div class='price'>
                <b class='sec-price'>￥{{item.costPrice}}</b>
                <b class='ever-price'>￥{{item.price}}</b>
            </div>
            <div class='num'>
                <div>已售{{item.num-item.stockCount}}</div>
                <div class='progress'>
                    <div class='sui-progress progress-danger'>
                        <span :style="'width:'+((item.num-item.stockCount)/item.num)*100+'%'" class='bar'></span>
                    </div>
                </div>
                <div>剩余
                    <b class='owned'>{{item.stockCount}}</b>件
                </div>
            </div>
            <a class='sui-btn btn-block btn-buy' target='_blank'>立即抢购</a>
        </li>
    </ul>
</div>
```

上图代码如下:

```
<!--商品列表-->
<div class="goods-list" id="app1">
    <ul class="seckill" id="seckill">
        <li class="seckill-item" v-for="item in goodslist" :key="item">
            <div class="pic">
                <img :src="item.smallPic" alt=''>
            </div>
            <div class="intro">
                <span>{{item.title}}</span>
            </div>
            <div class='price'>
                <b class='sec-price'>￥{{item.costPrice}}</b>
                <b class='ever-price'>￥{{item.price}}</b>
            </div>
            <div class='num'>
                <div>已售{{item.num-item.stockCount}}</div>
                <div class='progress'>
                    <div class='sui-progress progress-danger'>
                        <span :style="'width:'+((item.num-
item.stockCount)/item.num)*100+'%'" class='bar'></span>
                    </div>
                </div>
                <div>剩余
                    <b class='owned'>{{item.stockCount}}</b>件
                </div>
            </div>
            <a class='sui-btn btn-block btn-buy' target='_blank'>立即抢购</a>
```

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

3.7 时间菜单数据切换

每次点击不同时间菜单的时候，将对应的时间菜单的开始时间传入到searchList方法中，实现数据搜索，代码如下：

```
<!--秒杀时间-->
<div class="sectime">
    <div v-for="(item,index) in dateMenus"
         :class="['item-time',index==ctime? 'active':'']"
         @click="ctime=index;searchList(item)">
        <div class="time-clock">{{item | dateFilter('HH:mm')}}</div>
        <div class="time-state-on">
            <span class="on-text" v-if="index==0">快抢中</span>
            <span class="on-over" v-if="index==0">距离结束：{{timedown(alltimes[index])}}</span>

            <span class="on-text" v-if="index>0">即将开始</span>
            <span class="on-over" v-if="index>0">距离开始：{{timedown(alltimes[index])}}</span>
        </div>
    </div>
</div>
```

3.8 详情页跳转

跳转到详情页的时候，需要根据商品ID以及商品秒杀开始时间查询商品详情，所以每次需要将商品ID和开始时间传入到后台去，时间还需要转换成yyyyMMddHH格式，我们可以先定义一个方法，实现地址栏拼接，代码如下：

```
//点击跳转，获取跳转地址  startTime:YYYYMMDDHH
targetHref:function (startTime,id) {
    return "/seckill-item.html?time="+moment(startTime).format("YYYYMMDDHH")+"&id="+id;
}
```

页面点击抢购的时候，href调用上面写好的方法，代码如下：

```
<a class='sui-btn btn-block btn-buy' :href="targetHref(item.startTime,item.id)"
target='_blank'>立即抢购</a>
```

效果如下：

第4章 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。

4.2 业务层

4.2.1 业务层接口方法添加

修改qingcheng_interface工程的com.qingcheng.service.seckill.SeckillGoodsService接口，添加一个根据ID查询详情方法one()，代码如下：

```
public interface SeckillGoodsService {

    //...略

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

4.2.2 业务层实现类

修改qingcheng_service_seckill工程的com.qingcheng.service.impl.SeckillGoodsServiceImpl类，在类中添加一个one()方法，代码如下：

```
@Service
public class SeckillGoodsServiceImpl implements SeckillGoodsService {

    //...略

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

4.3 控制层

修改qingcheng_web_seckill工程，在该工程中的com.qingcheng.controller.SeckillGoodsController类中添加一个one()方法，用于调用Service层实现根据ID查询商品详情，代码如下：

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

    //...略

    /****
     * 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,Long id){
        //调用Service查询商品详情
        return seckillGoodsService.one(time,id);
    }
}
```

4.4 页面绑定

4.4.1 js配置

在页面编写js代码，先定义一个entity对象，用于接收从后台查询的商品详情信息，然后再定义一个getOne方法，用于去后台查询数据，代码如下：

```
<script>
    var app = new Vue({
        el: '#app',
        data: {
            entity: {},         //存储商品详情
        },
        methods: {
            //获取ID查询数据
            getOne: function () {
                //获取地址栏time参数
                let time = getQueryString("time");
                //获取地址栏id参数
                let id = getQueryString("id");

                //调用后台查询数据
                axios.get('/seckill/goods/one.do?time=' + time + '&id=' + id).then(function
(response) {
                    app.entity = response.data;
                })
            },
        },
```

```
        created: function () {
            //初始化调用
            this.getOne();
        }
    });
</script>
```

注意：上面用到了getQueryString方法，获取地址栏的指定参数，我们需要在页面引入util.js工具包。

```
<script type="text/javascript" src="/js/util.js"></script>
```

4.4.2 页面元素绑定

将数据从entity中绑定输出到页面中，代码如下：

图片：

```
<span class="jqzoom">
    <img :jqimg="entity.smallPic" :src="entity.smallPic"/>
</span>
```

商品标题：

```
<h4>{{entity.title}}</h4>
```

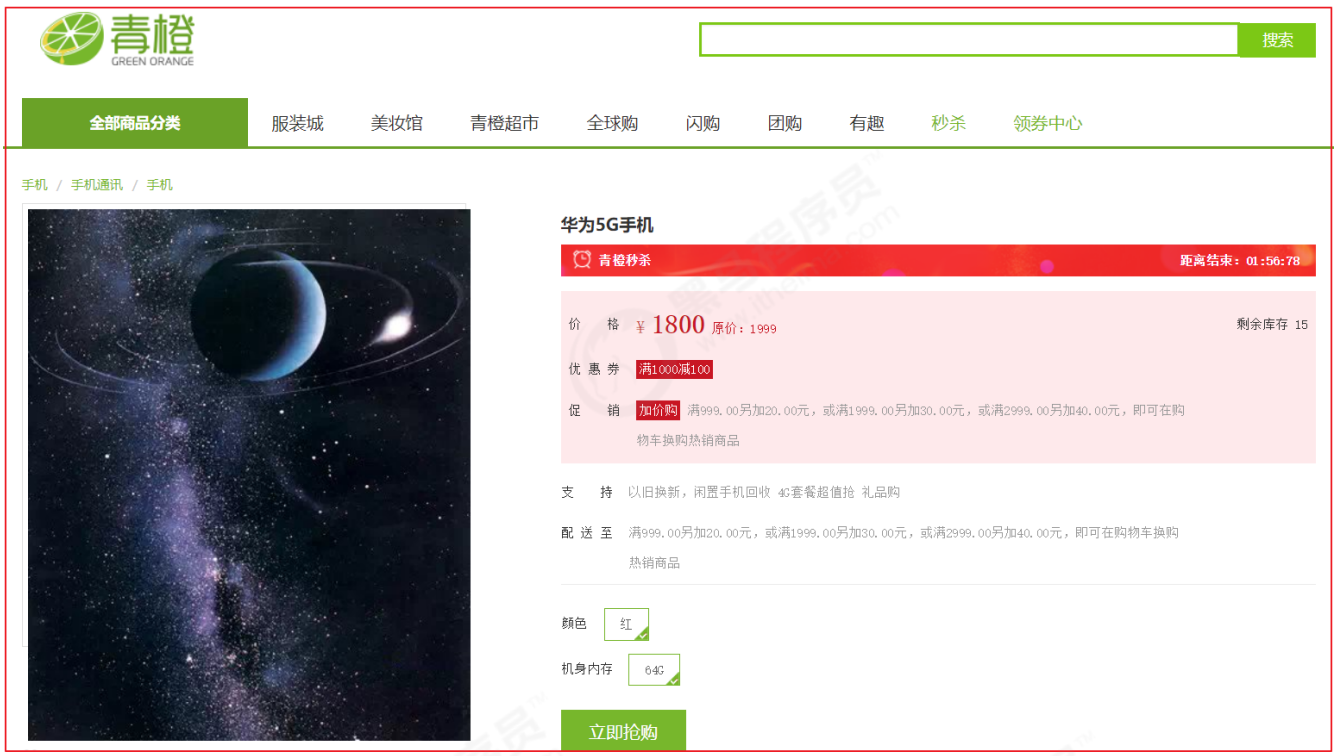
商品价格和库存:

```
<div class="summary-wrap">
    <div class="fl title">
        <i>价    格</i>
    </div>
    <div class="fl price">
        <i>¥</i>
        <em>{{entity.costPrice}}</em>
        <span>原价: {{entity.price}}</span>
    </div>
    <div class="fr remark">
        <i>剩余库存</i>
        <em>{{entity.stockCount}}</em>
    </div>
</div>
```

商品描述：

```
<div id="one" class="tab-pane active">
    <div class="intro-detail" v-html="entity.introduction">
    </div>
</div>
```

测试效果如下：



4.5 倒计时实现

在秒杀页面有一个倒计时功能，这个功能其实只需要我们计算当前时间距离活动结束时间拥有多少天、小时、分钟、秒，我们下面提供一段js代码，实现了时间单位换算，可以将毫秒换成天-小时-分钟-秒。该方法只需要将2个时间的差值传入即可。代码如下：

```
//将毫秒转换成天时分秒
timedown:function(num) {
    var oneSecond = 1000;
    var oneMinute=oneSecond*60;
    var oneHour=oneMinute*60
    var oneDay=oneHour*24;
    //天数
    var days =Math.floor(num/oneDay);
    //小时
    var hours =Math.floor((num%oneDay)/oneHour);
    //分钟
    var minutes=Math.floor((num%oneHour)/oneMinute);
    //秒
    var seconds=Math.floor((num%oneMinute)/oneSecond);
    //拼接时间格式
    var str = days+'天'+hours+'时'+minutes+'分'+seconds+'秒';
    return str;
}
```

我们可以将上述代码添加到seckill-item.html的js页面里面，计算时间差。

setInterval: 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

我们可以利用setInterval来实现倒计时功能，只需要定义一个方法，将一个总的时间实现每秒减去秒，然后调用上面时间换算的方法，并定义一个变量startTime接收换算好的时间，即可实现倒计时。

我们定义一个timeCalculate方法，在该方法中，分别传入starttimes和endtimes2个参数，这2个参数分别用于运算是距离开始时间还是距离结束时间的时间差值，实现代码如下：

```
//倒计时运算
timeCalculate:function (starttimes,endtimes) {
    //提示当前是距离开始还是距离结束
    let prefix="距离结束:";

    //时间差,单位：毫秒
    let timeremain=9999999;

    //获取当前时间
    let nowtimes = new Date().getTime();

    //判断是距离开始还是距离结束
    if(starttimes>nowtimes){
        prefix="距离开始:";
    }
    timeremain=endtimes-nowtimes;

    let clock = window.setInterval(function () {
        //时间递减
        timeremain=timeremain-1000;
        //调用时间单位换算方法
        app.startTime =prefix + app.timedown(timeremain);

        if(timeremain<1000){
            //倒计时距离开始切换到距离结束
            if(starttimes>nowtimes){
                //停止递减
                window.clearInterval(clock);
                //再次运算
                app.timeCalculate(starttimes,endtimes);
            }else{
                //商品活动结束
                window.clearInterval(clock);
            }
        }
    },1000)
},
```

我们需要在将商品详情查询完毕后，求出开始时间和结束时间，然后将时间传入到上述的那个方法，代码如下：

```
getOne: function () {
    //获取地址栏time参数
    let time = getQueryString("time");
    //获取地址栏id参数
    let id = getQueryString("id");

    //调用后台查询数据
    axios.get('/seckill/goods/one.do?time=' + time + '&id=' + id).then(function (response) {
        app.entity = response.data;

        //商品开始时间
        let starttimes = new Date(app.entity.startTime).getTime();

        //商品下架时间
        let endtimes = new Date(app.entity.endTime).getTime();

        //开始倒计时
        app.timeCalculate(starttimes,endtimes);
    })
},
```

完整实现代码如下：

```
<script>
    var app = new Vue({
        el: '#app',
        data: {
            entity: {},          //存储商品详情
            startTime:'',        //倒计时显示
            message:''           //显示错误信息
        },
        methods: {
            //获取ID查询数据
            getOne: function () {
                //获取地址栏time参数
                let time = getQueryString("time");
                //获取地址栏id参数
                let id = getQueryString("id");

                //调用后台查询数据
                axios.get('/seckill/goods/one.do?time=' + time + '&id=' + id).then(function (response) {
                    app.entity = response.data;

                    //商品开始时间
                    let starttimes = new Date(app.entity.startTime).getTime();

                    //商品下架时间
                    let endtimes = new Date(app.entity.endTime).getTime();

                    //开始倒计时
                    app.timeCalculate(starttimes,endtimes);
                })
            },

            //倒计时运算
            timeCalculate:function (starttimes,endtimes) {
                //提示当前是距离开始还是距离结束
                let prefix="距离结束:";

                //时间差,单位:毫秒
                let timeremain=9999999;

                //获取当前时间
                let nowtimes = new Date().getTime();

                //判断是距离开始还是距离结束
                if(starttimes>nowtimes){
                    prefix="距离开始:";
                }
                timeremain=endtimes-nowtimes;

                let clock = window.setInterval(function () {
                    //时间递减
                    timeremain=timeremain-1000;
                    //调用时间单位换算方法
                    app.startTime =prefix + app.timedown(timeremain);

                    if(timeremain<1000){
                        //倒计时距离开始切换到距离结束
                        if(starttimes>nowtimes){
                            //停止递减
                            window.clearInterval(clock);
```

准备倒计时

倒计时实现

```
                //再次运算
                app.timeCalculate(starttimes,endtimes);
            }else{
                //商品活动结束
                window.clearInterval(clock);
            }
        }
    },1000)
},

//将毫秒转换成天时分秒
timedown:function(num) {
    var oneSecond = 1000;
    var oneMinute=oneSecond*60;
    var oneHour=oneMinute*60
    var oneDay=oneHour*24;
    //天数
    var days =Math.floor(num/oneDay);
    //小时
    var hours =Math.floor((num%oneDay)/oneHour);
    //分钟
    var minutes=Math.floor((num%oneHour)/oneMinute);
    //秒
    var seconds=Math.floor((num%oneMinute)/oneSecond);
    //拼接时间格式
    var str = days+'天'+hours+'时'+minutes+'分'+seconds+'秒';
    return str;
}
},
created: function () {
    //初始化调用
    this.getOne();
}
});
</script>
```

时间单位换算

上图代码如下:

```
<script>
    var app = new Vue({
        el: '#app',
        data: {
            entity: {},          //存储商品详情
            startTime:'',      //倒计时显示
            message:''         //显示错误信息
        },
        methods: {
            //获取ID查询数据
            getOne: function () {
                //获取地址栏time参数
                let time = getQueryString("time");
                //获取地址栏id参数
                let id = getQueryString("id");

                //调用后台查询数据
```

```
                axios.get('/seckill/goods/one.do?time=' + time + '&id=' + id).then(function
(response) {
                    app.entity = response.data;

                    //商品开始时间
                    let starttimes = new Date(app.entity.startTime).getTime();

                    //商品下架时间
                    let endtimes = new Date(app.entity.endTime).getTime();

                    //开始倒计时
                    app.timeCalculate(starttimes,endtimes);
                })
            },

            //倒计时运算
            timeCalculate:function (starttimes,endtimes) {
                //提示当前是距离开始还是距离结束
                let prefix="距离结束:";

                //时间差,单位：毫秒
                let timeremain=9999999;

                //获取当前时间
                let nowtimes = new Date().getTime();

                //判断是距离开始还是距离结束
                if(starttimes>nowtimes){
                    prefix="距离开始:";
                }
                timeremain=endtimes-nowtimes;

                let clock = window.setInterval(function () {
                    //时间递减
                    timeremain=timeremain-1000;
                    //调用时间单位换算方法
                    app.startTime =prefix + app.timedown(timeremain);

                    if(timeremain<1000){
                        //倒计时距离开始切换到距离结束
                        if(starttimes>nowtimes){
                            //停止递减
                            window.clearInterval(clock);
                            //再次运算
                            app.timeCalculate(starttimes,endtimes);
                        }else{
                            //商品活动结束
                            window.clearInterval(clock);
                        }
                    }
                },1000)
            },
```

```
            //将毫秒转换成天时分秒
            timedown:function(num) {
                var oneSecond = 1000;
                var oneMinute=oneSecond*60;
                var oneHour=oneMinute*60
                var oneDay=oneHour*24;
                //天数
                var days =Math.floor(num/oneDay);
                //小时
                var hours =Math.floor((num%oneDay)/oneHour);
                //分钟
                var minutes=Math.floor((num%oneHour)/oneMinute);
                //秒
                var seconds=Math.floor((num%oneMinute)/oneSecond);
                //拼接时间格式
                var str = days+'天'+hours+'时'+minutes+'分'+seconds+'秒';
                return str;
            }
        },
        created: function () {
            //初始化调用
            this.getOne();
        }
    });
</script>
```

页面显示错误信息

```
<li style="line-height: 42.8px">
    <a href="javascript:void(0)" @click="add()" class="addshopcar">立即抢购</a>
    {{message}}
</li>
```

活动还未开始前效果：

活动开始后效果：

第5章 通用跳转

用户每次下单的时候，必须是处于登录状态，这时候我们可以使用CAS做单点登录，登录成功后再跳转到商品详情页，点击下单购买，但问题是登录成功后，再如何跳转到当前商品详情页?

用户每次向后台发送请求的时候，头文件中会有一个referer属性，referer告诉服务器我是从哪个页面链接过来的。我们可以定义一个用户无法直接访问的方法，用户每次访问该方法的时候，会携带referer，此时用户无法访问，会跳转到CAS，CAS登录后会再次访问之前要访问的方法，我们可以在后台该方法中获取该头文件属性，然后每次再跳转过来，这样就可以实现页面跳转原路返回了。

5.1 定义跳转控制器

以编写一个controller来实现跳转，创建RedirectController类，代码如下:

上图代码如下:

```
@Controller
@RequestMapping(value = "/redirect")
public class RedirectController {

    /****
     * 登录地址跳转方法
     * referer是header的一部分,
```

```
     *       当浏览器向web服务器发送请求的时候，一般会带上Referer，告诉服务器我是从哪个页面链接过来的。
     * 登录地址返回
     * @param referer
     * @return
     */
    @RequestMapping(value = "/back")
    public String redirect(@RequestHeader(value = "Referer",required = false)String referer)
{
        if(!StringUtils.isEmpty(referer)){
            return "redirect:"+referer;
        }
        return "redirect:/seckill-index.html";
    }
}
```

5.2 下单登录判断

创建一个下单控制器，并创建下单方法，让下单方法允许匿名访问，用户每次下单的时候，必须要先登录，可以在下单控制器中获取用户登录名，如果登录名为匿名用户(anonymousUser),则提示用户登录。

5.2.1 匿名配置

修改qingcheng_common_cas工程的spring-security.xml,添加匿名访问配置

```
<!--   entry-point-ref  入口点引用 -->
<http  use-expressions="false" entry-point-ref="casProcessingFilterEntryPoint">
    <intercept-url pattern="/login/*.do" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/seckill/*/*.do" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/seckill/*/*/*.do" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/**" access="ROLE_USER"/>
    <csrf disabled="true"/>
    <!-- custom-filter为过滤器， position 表示将过滤器放在指定的位置上，before表示放在指定位置之前  ，after表示放在指定的位置之后  -->
    <custom-filter ref="casAuthenticationFilter"  position="CAS_FILTER" />
    <custom-filter ref="requestSingleLogoutFilter" before="LOGOUT_FILTER"/>
    <custom-filter ref="singleLogoutFilter" before="CAS_FILTER"/>
</http>
```

上图代码如下：

```
<intercept-url pattern="/seckill/*/*.do" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/seckill/*/*/*.do" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
```

5.2.2 集成SpringSecurity+CAS

修改qingcheng_web_seckill工程的web.xml，集成SpringSecurity,在web.xml中添加如下配置：

```
<!--代理过滤器-->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
```

```
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!--SpringSecurity配置文件-->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:spring-security*.xml</param-value>
</context-param>
<!--创建Spring监听器-->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

创建spring-security-seckill.xml，取消seckill-index.html和seckill-item.html页面的安全配置，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
             xmlns:beans="http://www.springframework.org/schema/beans"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="
             http://www.springframework.org/schema/beans
             http://www.springframework.org/schema/beans/spring-beans.xsd
              http://www.springframework.org/schema/security
              http://www.springframework.org/schema/security/spring-security.xsd">

    <!--放行地址-->
    <http pattern="/seckill-index.html" security="none"></http>
    <http pattern="/seckill-item.html" security="none"></http>

</beans:beans>
```

5.2.3 下单控制器创建

创建一个com.qingcheng.controller.SeckillOrderController控制器，用于实现下单操作，代码如下：

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

    @Reference
    private SeckillOrderService seckillOrderService;
```

```
    /***
     * URL:/seckill/order/add
     * 秒杀下单
     * @return
     */
    @RequestMapping(value = "/add")
    public Result add(Long id){
        try {
            //获取用户名
            String username =
SecurityContextHolder.getContext().getAuthentication().getName();

            //如果用户账号为anonymousUser，则表明用户未登录
            if(username.equalsIgnoreCase("anonymousUser")){
                //这里403错误代码表示用户没登录
                return new Result(403,"请先登录！");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return new Result(1,"秒杀下单失败！");
    }

}
```

5.2.4 页面对接

在页面添加一个下单方法，并在方法中实现下单操作，代码如下：

```
//添加订单
add:function () {
    //提示信息
    app.message='正在下单';
    //获取地址栏time参数
    let time = getQueryString("time");
    //获取地址栏id参数
    let id = getQueryString("id");

    //下单操作
    axios.get('/seckill/order/add.do?time='+time+'&id='+id).then(function (response) {
        if(response.data.code==1){
            //成功!
            app.message='抢单成功，即将进入支付!';
        }else if(response.data.code==403){
            //用户未登录,跳转到登录页
            app.message=response.data.message;

            //未登录,跳转到/back.do,携带referer，此时会被拒绝访问，跳转到CAS登录
            // CAS登录成功后，跳转到/back.do，此时跟重新跳转到referer对应的地址，也就是当前页地址
            location.href='/redirect/back.do';
        }else{
            //用户未登录
            app.message=response.data.message;
        }
    })
},
```

上图代码如下：

```
//添加订单
add:function () {
    //提示信息
    app.message='正在下单';
    //获取地址栏time参数
    let time = getQueryString("time");
    //获取地址栏id参数
    let id = getQueryString("id");

    //下单操作
    axios.get('/seckill/order/add.do?time='+time+'&id='+id).then(function (response) {
        if(response.data.code==1){
            //成功!
            app.message='抢单成功，即将进入支付!';
        }else if(response.data.code==403){
            //用户未登录,跳转到登录页
            app.message=response.data.message;

            //未登录,跳转到/back.do,携带referer，此时会被拒绝访问，跳转到CAS登录
            // CAS登录成功后，跳转到/back.do，此时跟重新跳转到referer对应的地址，也就是当前页地址
            location.href='/redirect/back.do';
        }else{
            //用户未登录
            app.message=response.data.message;
        }
    })
},
```

立即抢购 按钮添加一个点击事件，每次点击的时候，调用上面的add下单方法，代码如下：

```
<a href="javascript:void(0)" @click="add()" class="sui-btn  btn-danger addshopcar">立即抢购
</a>
```

第6章 下单

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。

6.1 Dao层创建

在qingcheng_service_seckill创建com.qingcheng.dao.SeckillOrderMapper接口代码如下：

```
public interface SeckillOrderMapper extends Mapper<SeckillOrder> {
}
```

6.2 Service层创建

6.2.1 Service接口创建

在qingcheng_interface中创建com.qingcheng.service.seckill.SeckillOrderService接口，代码如下：

```
public interface SeckillOrderService {

    /***
     * 添加秒杀订单
     * @param id:商品ID
     * @param time:商品秒杀开始时间
     * @param username:用户登录名
     * @return
     */
    Boolean add(Long id, String time, String username);
}
```

6.2.2 Service接口实现类创建

在qingcheng_service_seckill中创建com.qingcheng.service.impl.SeckillOrderServiceImpl实现类，代码如下：

```
@Service
public class SeckillOrderServiceImpl implements SeckillOrderService {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;

    @Autowired
    private IdWorker idWorker;

    /****
     * 添加订单
     * @param id
     * @param time
     * @param username
     */
    @Override
    public Boolean add(Long id, String time, String username){
        //获取商品数据
        SeckillGoods goods = (SeckillGoods) redisTemplate.boundHashOps("SeckillGoods_" +
time).get(id);

        //如果没有库存，则直接抛出异常
        if(goods==null || goods.getStockCount()<=0){
            throw new RuntimeException("已售罄!");
        }
        //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.setSellerId(goods.getSellerId());
        seckillOrder.setCreateTime(new Date());
        seckillOrder.setStatus("0");

        //将秒杀订单存入到Redis中
        redisTemplate.boundHashOps("SeckillOrder").put(username,seckillOrder);

        //库存减少
        goods.setStockCount(goods.getStockCount()-1);

        //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，则直数据重置到Reids中
            redisTemplate.boundHashOps("SeckillGoods_" + time).put(id,goods);
```

```
        }
        return true;
    }
}
```

6.3 控制层创建

在qingcheng_web_seckill中的SeckillOrderController类的add方法中完善添加订单方法，代码如下：

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

    @Reference
    private SeckillOrderService seckillOrderService;

    /****
     * URL:/seckill/order/add
     * 添加订单
     * 调用Service增加订单
     * 匿名访问：anonymousUser
     * @param time
     * @param id
     */
    @RequestMapping(value = "/add")
    public Result add(String time,Long id){
        try {
            //获取用户登录名
            String username =
SecurityContextHolder.getContext().getAuthentication().getName();

            //用户未登录
            if(username.equalsIgnoreCase("anonymousUser")){
                //403表示用户未登录
                return new Result(403,"未登录！");
            }

            //调用Service增加订单
            Boolean bo = seckillOrderService.add(id, time, username);

            if(bo){
                //抢单成功
                return new Result(0,"下单成功！");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return new Result(1,e.getMessage());
        }
        return new Result(1,"下单失败！");
    }
}
```

可以看到Reids中的下单数据如下：