Day03

# 课程安排

* 秒杀实现思路分析
* 秒杀频道首页功能
* 秒杀商品详细页功能
* 秒杀下单功能
* 解决下单并发产生的订单异常问题
* 解决高并发下用户下单排队和超限问题

# 秒杀业务分析

## 需求分析

所谓“秒杀”，就是网络[卖家](https://baike.baidu.com/item/%E5%8D%96%E5%AE%B6" \t "https://baike.baidu.com/item/%E7%A7%92%E6%9D%80/_blank)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。

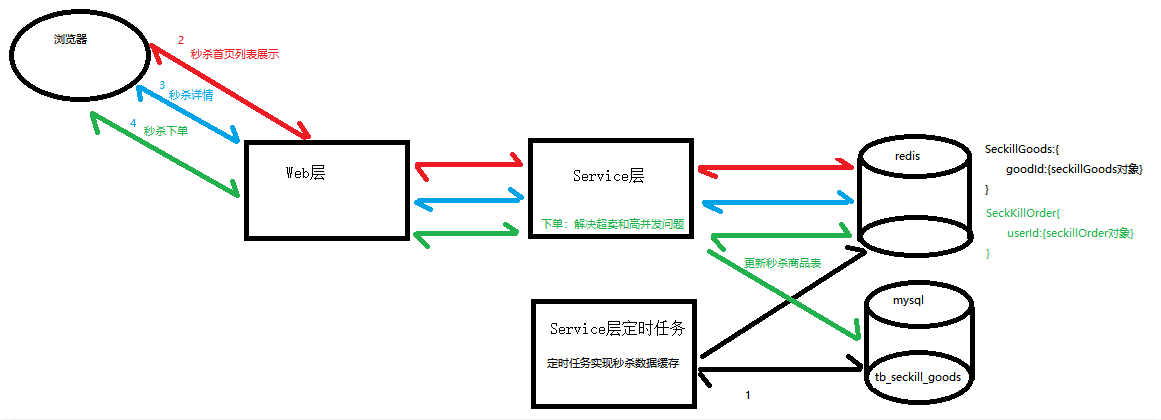
秒杀商品通常有两种限制：库存限制、时间限制。

需求：

1. 商家（pyg\_shop\_web）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（pyg\_manager\_web）审核秒杀申请
3. 秒杀频道首页（pyg\_seckill\_web）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（pyg\_seckill\_web）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（pyg\_seckill\_web/pyg\_seckill\_service），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。

## 秒杀实现思路

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。

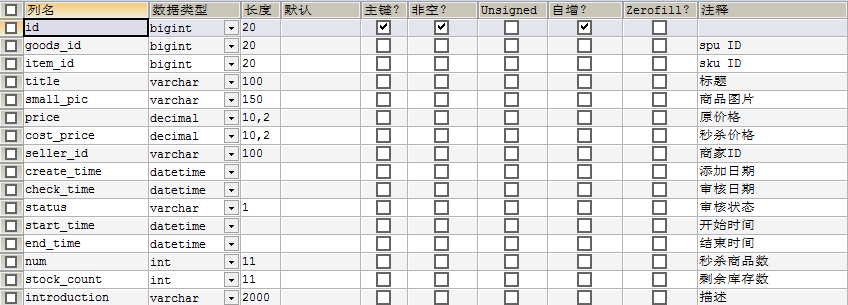


首先通过定时任务将数据库中的数据导入到redis中存起来，以redis中hash结构存储的，key是名，value也是一个 键值对（key是秒杀商品的id,value是秒杀对象），下单成功后需要将订单对象保存到redis中然后在去更新秒杀商品的数据库。

# 环境准备

## 数据库环境

Tb\_seckill\_goods 秒杀商品表



Tb\_seckill\_order 秒杀订单表



-- ----------------------------

-- Table structure for `tb\_seckill\_goods`

-- ----------------------------

DROP TABLE IF EXISTS `tb\_seckill\_goods`;

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` varchar(1) DEFAULT NULL COMMENT '审核状态',

`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=6 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of tb\_seckill\_goods

-- ----------------------------

INSERT INTO `tb\_seckill\_goods` VALUES (1, 149187842867960, NULL, '秒杀精品女装', 'http://img.mp.itc.cn/upload/20160804/6881885758bb42e09bff6e3d60d18230\_th.jpg', 100.00, 0.01, 'qiandu', NULL, '2017-10-14 21:07:51', '1', '2017-10-14 18:07:27', '2017-10-14 18:07:31', 10, 5, NULL);

INSERT INTO `tb\_seckill\_goods` VALUES (2, 149187842867953, NULL, '轻轻奶茶', 'http://sem.g3img.com/site/50021489/image/c2\_20190411232047\_66099.jpg', 10.00, 0.01, 'yijia', NULL, NULL, '1', '2017-10-12 18:24:18', '2017-10-28 18:24:20', 10, 5, '清仓打折');

INSERT INTO `tb\_seckill\_goods` VALUES (3, 11, NULL, '11', 'http://i2.sinaimg.cn/ty/2014/0326/U5295P6DT20140326155117.jpg', 44.00, 0.03, NULL, NULL, NULL, '1', '2017-1-1 00:00:00', '2017-12-1 00:00:00', 10, 2, NULL);

INSERT INTO `tb\_seckill\_goods` VALUES (4, 2, NULL, '测试', 'http://www.cnr.cn/junshi/ztl/leifeng/smlf/201202/W020120226838451234901.jpg', 10.00, 0.01, 'qiandu', '2017-10-14 19:18:18', NULL, '0', '2017-11-11 00:00:00', '2017-11-11 23:59:59', 100, 99, NULL);

INSERT INTO `tb\_seckill\_goods` VALUES (5, NULL, NULL, '羽绒服', 'http://img14.360buyimg.com/popWaterMark/g13/M03/0A/1D/rBEhU1Kmlr8IAAAAAATBCejgYvoAAGmMAC0zhIABMEh349.jpg', 100.00, 0.02, 'qiandu', '2017-10-15 09:50:52', '2017-10-15 10:06:27', '1', '2017-10-10 00:00:00', '2017-11-11 23:59:59', 10, 10, '清仓打折');

-- ----------------------------

-- Table structure for `tb\_seckill\_order`

-- ----------------------------

DROP TABLE IF EXISTS `tb\_seckill\_order`;

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` varchar(1) DEFAULT NULL COMMENT '状态',

`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;

-- ----------------------------

-- Records of tb\_seckill\_order

-- ----------------------------

INSERT INTO `tb\_seckill\_order` VALUES ('919473120379723776', null, '0.02', 'lijialong', 'qiandu', '2017-10-15 16:00:49', '2017-10-15 16:03:36', '1', null, null, null, '4200000013201710158227452548');

INSERT INTO `tb\_seckill\_order` VALUES ('919474775091339264', null, '0.02', 'lijialong', 'qiandu', '2017-10-15 16:07:24', '2017-10-15 16:07:58', '1', null, null, null, '4200000007201710158230411417');

INSERT INTO `tb\_seckill\_order` VALUES ('919497114331951104', '2', '0.01', null, 'yijia', '2017-10-15 17:36:10', '2017-10-15 17:37:35', '1', null, null, null, '4200000004201710158248971034');

INSERT INTO `tb\_seckill\_order` VALUES ('919497943340302336', '2', '0.01', null, 'yijia', '2017-10-15 17:39:27', '2017-10-15 17:39:49', '1', null, null, null, '4200000011201710158245347392');

## Web环境

本示例技术架构采用SSM+angularjs实现，持久层使用Mybatis逆向工程生成。

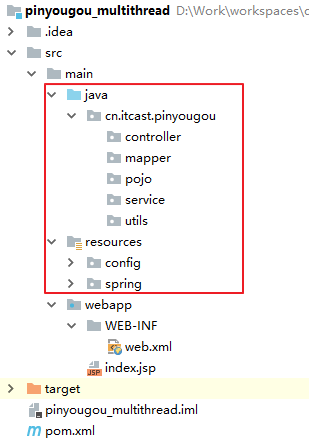
本示例假设阅读者有SSM和angularjs基础，如果没有，请自行学习。

### 创建web项目，引入jar包依赖

*<?***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"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>cn.itcast</**groupId**>  
 <**artifactId**>pinyougou\_multithread</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>war</**packaging**>  
  
 <**name**>pinyougou\_multithread Maven Webapp</**name**>  
 *<!--* ***FIXME change it to the project's website*** *-->* <**url**>http://www.example.com</**url**>  
  
 *<!-- 集中定义依赖版本号 -->* <**properties**>  
 <**junit.version**>4.12</**junit.version**>  
 <**spring.version**>4.2.4.RELEASE</**spring.version**>  
 <**pagehelper.version**>4.0.0</**pagehelper.version**>  
 <**servlet-api.version**>2.5</**servlet-api.version**>  
 <**dubbo.version**>2.8.4</**dubbo.version**>  
 <**zookeeper.version**>3.4.7</**zookeeper.version**>  
 <**zkclient.version**>0.1</**zkclient.version**>  
 <**mybatis.version**>3.2.8</**mybatis.version**>  
 <**mybatis.spring.version**>1.2.2</**mybatis.spring.version**>  
 <**mybatis.paginator.version**>1.2.15</**mybatis.paginator.version**>  
 <**mysql.version**>5.1.32</**mysql.version**>  
 <**druid.version**>1.0.9</**druid.version**>  
 <**commons-fileupload.version**>1.3.1</**commons-fileupload.version**>  
 <**freemarker.version**>2.3.23</**freemarker.version**>  
 <**activemq.version**>5.11.2</**activemq.version**>  
 <**security.version**>3.2.3.RELEASE</**security.version**>  
 <**solrj.version**>4.10.3</**solrj.version**>  
 <**ik.version**>2012\_u6</**ik.version**>  
 </**properties**>  
  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.quartz-scheduler</**groupId**>  
 <**artifactId**>quartz</**artifactId**>  
 <**version**>2.2.3</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.quartz-scheduler</**groupId**>  
 <**artifactId**>quartz-jobs</**artifactId**>  
 <**version**>2.2.3</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.aliyun</**groupId**>  
 <**artifactId**>aliyun-java-sdk-core</**artifactId**>  
 <**version**>3.2.5</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.aliyun</**groupId**>  
 <**artifactId**>aliyun-java-sdk-dysmsapi</**artifactId**>  
 <**version**>1.0.0-SNAPSHOT</**version**>  
 </**dependency**>  
 *<!-- Spring -->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-beans</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-aspects</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jms</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context-support</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.9</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.28</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>javassist</**groupId**>  
 <**artifactId**>javassist</**artifactId**>  
 <**version**>3.11.0.GA</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>commons-codec</**groupId**>  
 <**artifactId**>commons-codec</**artifactId**>  
 <**version**>1.10</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>servlet-api</**artifactId**>  
 <**version**>2.5</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
 *<!-- Mybatis -->* <**dependency**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis</**artifactId**>  
 <**version**>${mybatis.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis-spring</**artifactId**>  
 <**version**>${mybatis.spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.github.miemiedev</**groupId**>  
 <**artifactId**>mybatis-paginator</**artifactId**>  
 <**version**>${mybatis.paginator.version}</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 *<!-- MySql -->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>${mysql.version}</**version**>  
 </**dependency**>  
 *<!-- 连接池 -->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>${druid.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>commons-collections</**groupId**>  
 <**artifactId**>commons-collections</**artifactId**>  
 <**version**>3.2.1</**version**>  
 </**dependency**>  
 *<!-- 缓存 -->* <**dependency**>  
 <**groupId**>redis.clients</**groupId**>  
 <**artifactId**>jedis</**artifactId**>  
 <**version**>2.8.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.data</**groupId**>  
 <**artifactId**>spring-data-redis</**artifactId**>  
 <**version**>1.7.2.RELEASE</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.github.penggle</**groupId**>  
 <**artifactId**>kaptcha</**artifactId**>  
 <**version**>2.3.2</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>javax.servlet-api</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>dom4j</**groupId**>  
 <**artifactId**>dom4j</**artifactId**>  
 <**version**>1.6.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>xml-apis</**groupId**>  
 <**artifactId**>xml-apis</**artifactId**>  
 <**version**>1.4.01</**version**>  
 </**dependency**>  
 </**dependencies**>  
  
 <**build**>  
  
 <**plugins**>  
 *<!-- java编译插件 -->* <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**version**>3.2</**version**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 <**encoding**>UTF-8</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 <**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**configuration**>  
 <**port**>8080</**port**>  
 <**path**>/</**path**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
  
 </**build**>  
</**project**>

### 配置SSM整合环境

项目结构：



配置web.xml

**<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" *>***<**web-app**>  
 <**display-name**>Archetype Created Web Application</**display-name**>  
 <**context-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/applicationContext-\*.xml</**param-value**>  
 </**context-param**>  
 <**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**>  
 </**filter**>  
 <**filter-mapping**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
 <**listener**>  
 <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  
 </**listener**>  
 <**servlet**>  
 <**servlet-name**>dispatcherServlet</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/springmvc.xml</**param-value**>  
 </**init-param**>  
 </**servlet**>  
 <**servlet-mapping**>  
 <**servlet-name**>dispatcherServlet</**servlet-name**>  
 <**url-pattern**>/</**url-pattern**>  
 </**servlet-mapping**>  
 <**welcome-file-list**>  
 <**welcome-file**>seckill\_index.html</**welcome-file**>  
 </**welcome-file-list**>  
</**web-app**>

配置spring/springmvc.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:context="http://www.springframework.org/schema/context"  
 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/mvc  
http://www.springframework.org/schema/mvc/spring-mvc.xsd  
http://www.springframework.org/schema/context  
http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 *<!-- 扫描Controller -->* <**context:component-scan base-package="cn.itcast.pinyougou.controller"**/>  
 *<!-- mvc注解驱动 -->* <**mvc:annotation-driven**>  
 <**mvc:message-converters**>  
 <**bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  
 <**property name="supportedMediaTypes" value="application/json"**/>  
 <**property name="features"**>  
 <**array**>  
 <**value**>WriteMapNullValue</**value**>  
 <**value**>WriteDateUseDateFormat</**value**>  
 </**array**>  
 </**property**>  
 </**bean**>  
 </**mvc:message-converters**>  
 </**mvc:annotation-driven**>  
 *<!-- 静态资源处理 -->* <**mvc:default-servlet-handler**/>  
</**beans**>

配置spring/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:aop="http://www.springframework.org/schema/aop"  
 xmlns:tx="http://www.springframework.org/schema/tx"  
 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/tx  
http://www.springframework.org/schema/tx/spring-tx.xsd  
http://www.springframework.org/schema/aop  
http://www.springframework.org/schema/aop/spring-aop.xsd  
http://www.springframework.org/schema/context  
http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 <**context:property-placeholder location="classpath:config/\*.properties"**/>  
  
 <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"**>  
 <**property name="driverClassName" value="${jdbc.driver}"**/>  
 <**property name="url" value="${jdbc.url}"**/>  
 <**property name="username" value="${jdbc.username}"**/>  
 <**property name="password" value="${jdbc.password}"**/>  
 </**bean**>  
 <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  
 <**property name="dataSource" ref="dataSource"**/>  
 <**property name="typeAliasesPackage" value="cn.itcast.pinyougou.pojo"**/>  
 </**bean**>  
 <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
 <**property name="basePackage" value="cn.itcast.pinyougou.mapper"**/>  
 </**bean**>  
 <**context:component-scan base-package="cn.itcast.pinyougou.service"**/>  
 <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dataSource"**/>  
 </**bean**>  
 <**tx:annotation-driven**/>  
</**beans**>

配置spring/applicationContext-redis.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:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 *<!-- redis 相关配置 -->* <**bean id="poolConfig" class="redis.clients.jedis.JedisPoolConfig"**>  
 <**property name="maxIdle" value="${redis.maxIdle}"** />  
 <**property name="maxWaitMillis" value="${redis.maxWait}"** />  
 <**property name="testOnBorrow" value="${redis.testOnBorrow}"** />  
 </**bean**>  
 <**bean id="jedisConnectionFactory"  
 class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"  
 p:host-name="${redis.host}" p:port="${redis.port}" p:password="${redis.pass}"  
 p:pool-config-ref="poolConfig"**/>  
 <**bean id="redisTemplate"  
 class="org.springframework.data.redis.core.RedisTemplate"**>  
 <**property name="connectionFactory" ref="jedisConnectionFactory"** />  
 </**bean**>  
</**beans**>

配置config/jdbc.properties

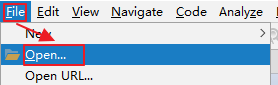
**jdbc.driver**=**com.mysql.jdbc.Driver  
jdbc.url**=**jdbc:mysql://localhost:3306/pinyougou  
jdbc.username**=**root  
jdbc.password**=**admin**

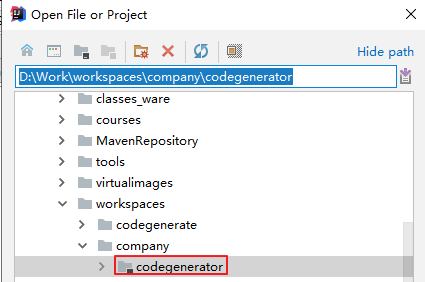
配置config/redis-config.properties

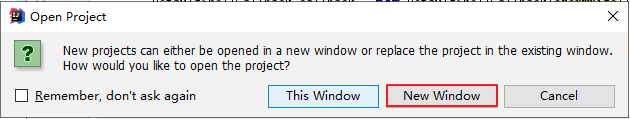
**redis.host**=**localhost  
redis.port**=**6379  
redis.pass**=  
**redis.database**=**0  
redis.maxIdle**=**300  
redis.maxWait**=**3000  
redis.testOnBorrow**=**true**

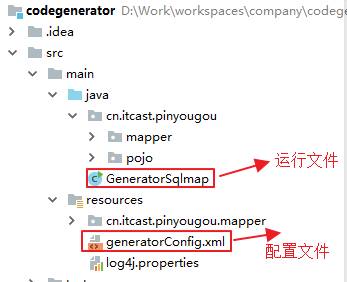
### 生成持久层代码

使用资料中提供的codegenerator项目，生成持久层代码





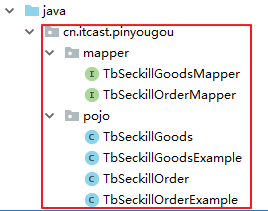


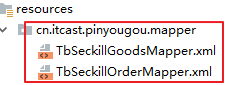


通过图中的配置文件配置数据库和生成目录；

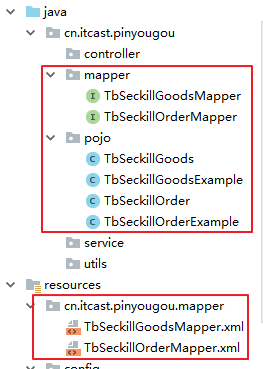
通过执行图中运行文件的main函数执行生成；

途中的mapper和pojo即为生成的代码





将代码拷贝到pinyougou\_multithread项目下

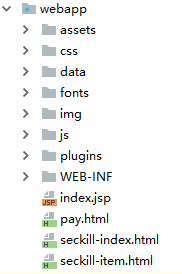


为pojo下的实体类添加Serializable接口





### 引入静态资源文件



# 秒杀商品导入缓存

秒杀查询压力是非常大的，我们可以在秒杀之前把秒杀商品存入到Redis缓存中，页面每次列表查询的时候直接从Redis缓存中取，这样会大大减轻MySQL数据库的压力。我们可以创建一个定时任务工程，每天秒杀的前一天运行并加载MySQL数据库数据到Redis缓存。

## Quartz概述

### Quartz介绍和下载

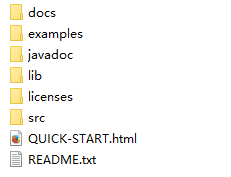
Quartz是OpenSymphony开源组织在Job scheduling领域又一个开源项目，它可以与J2EE与J2SE应用程序相结合也可以单独使用。Quartz可以用来创建简单或为运行十个，百个，甚至是好几万个Jobs这样复杂的程序。Jobs可以做成标准的Java组件或 EJBs。Quartz的最新版本为Quartz 2.2.3。

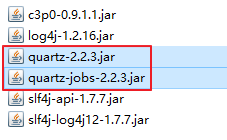
官网：<http://www.quartz-scheduler.org/>

下载开发包：



解压：



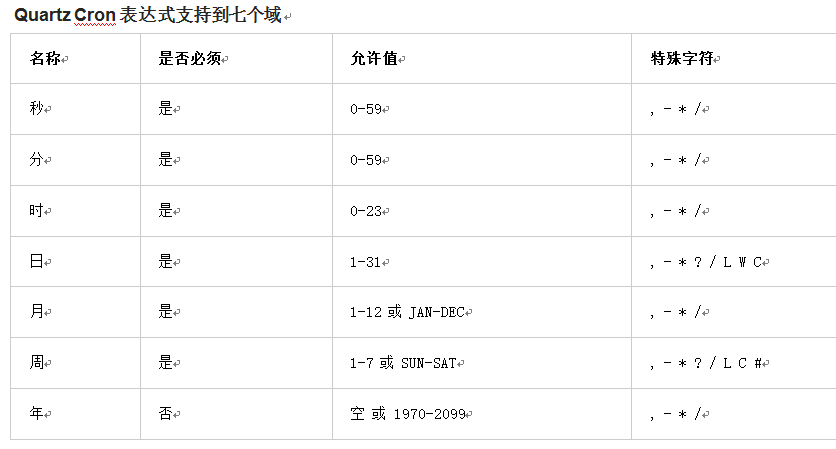


### Quartz执行流程

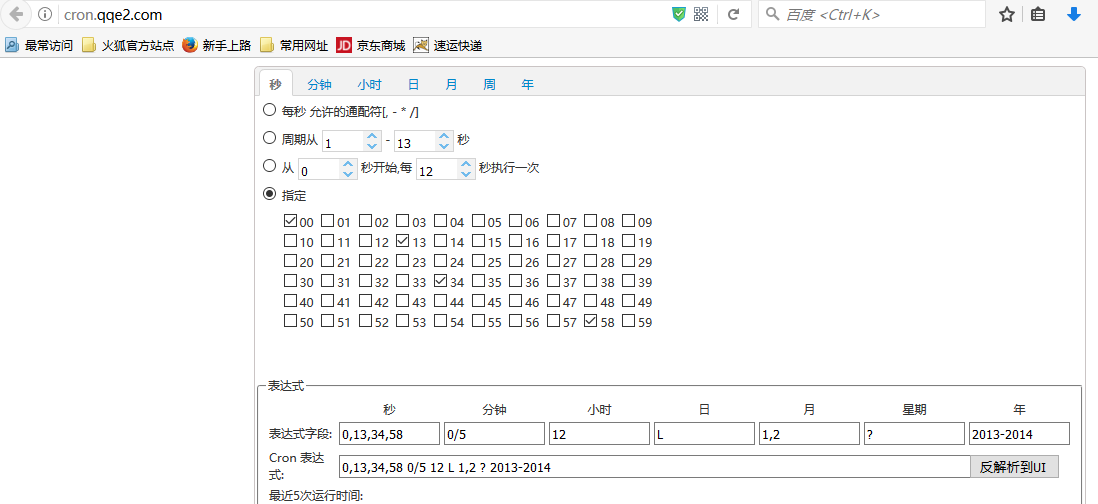
* **1、org.quartz.Job**   
  它是一个抽象接口，表示一个工作，也就是我们要执行的具体内容，他只定义了一个几口方法：   
  void execute(JobExecutionContext context)   
  作用等同Spring的：   
  **org.springframework.scheduling.quartz.QuartzJobBean**
* **2、org.quartz.JobDetail**   
  JobDetail表示一个具体的可执行的调度程序，Job是这个可执行程调度程序所要执行的内容，它包含了这个任务调度的方案和策略。   
  他的实现类：   
  **org.quartz.impl.JobDetailImpl**   
  作用等同Spring：   
  **org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean**
* **3、org.quartz.Trigger**   
  它是一个抽象接口，表示一个调度参数的配置，通过配置它，来告诉调度容器什么时候去调用JobDetail。   
  他的两个实现类:   
  **org.quartz.impl.triggers.SimpleTriggerImpl**   
  **org.quartz.impl.triggers.CronTriggerImpl**   
  等同于Spring的:   
  **org.springframework.scheduling.quartz.SimpleTriggerBean**   
  **org.springframework.scheduling.quartz.CronTriggerBean**   
  前者只支持按照一定频度调用任务，如每隔30分钟运行一次。   
  后者既支持按照一定频度调用任务，又支持定时任务。
* **4、org.quartz.Scheduler**   
  代表一个调度容器，一个调度容器中可以注册多个JobDetail和Trigger。当Trigger与JobDetail组合，就可以被Scheduler容器调度了。它的方法有start()、shutdown()等方法，负责管理整个调度作业。   
  等同Spring的： **org.springframework.scheduling.quartz.SchedulerFactoryBean**

### Cron表达式

七子表达式



表达式在线生成器：



### 入门案例

本案例基于quartz和spring整合应用

#### 第一步：创建maven工程，引入依赖

<**dependency**>  
 <**groupId**>org.quartz-scheduler</**groupId**>  
 <**artifactId**>quartz</**artifactId**>  
 <**version**>2.2.3</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.quartz-scheduler</**groupId**>  
 <**artifactId**>quartz-jobs</**artifactId**>  
 <**version**>2.2.3</**version**>  
</**dependency**>

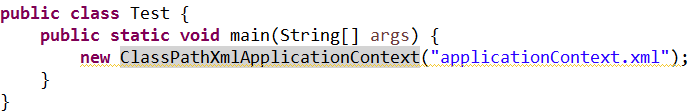
#### 第二步：创建一个自定义Job

**package** com.pinyougou.quartz.task;  
  
**import** org.springframework.scheduling.annotation.Scheduled;  
**import** org.springframework.stereotype.Component;  
  
**import** java.util.Date;  
  
@Component  
**public class** MyTask {  
  
 @Scheduled(cron = **"0/10 \* \* \* \* ?"**)  
 **public void** excTask(){  
 System.***out***.println(**"定时任务执行，执行时间是："**+**new** Date());  
 }  
}

#### 第三步：提供spring配置文件，配置定时任务

*<?***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:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 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/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task.xsd"**>  
  
 *<!--包扫描-->* <**context:component-scan base-package="com.pinyougou.quartz.task"**/>  
  
 *<!--开启注解驱动-->* <**task:annotation-driven**/>  
  
 </**beans**>

#### 第四步：加载上面的spring文件，创建spring工厂



## 秒杀商品导入Redis

### 创建定时任务类

SeckillGoodsToRedisTask.java

**package** cn.itcast.pinyougou.task;  
  
**import** cn.itcast.pinyougou.mapper.TbSeckillGoodsMapper;  
**import** cn.itcast.pinyougou.pojo.TbSeckillGoods;  
**import** cn.itcast.pinyougou.pojo.TbSeckillGoodsExample;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.data.redis.core.RedisTemplate;  
**import** org.springframework.scheduling.annotation.Scheduled;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.Date;  
**import** java.util.List;  
  
@Service  
**public class** SeckillGoodsToRedisTask {  
 @Autowired  
 **private** TbSeckillGoodsMapper **seckillGoodsMapper**;  
  
 @Autowired  
 **private** RedisTemplate **redisTemplate**;  
  
 */\*\*\*  
 \* 每年双十一启动秒杀  
 \* 将商品数据全部跟新到索引库  
 \*/* @Scheduled(cron = **"30 \* \* \* \* ?"**) *//我们这里测试数据每分钟30秒执行* **public void** startSeckill(){  
 TbSeckillGoodsExample example = **new** TbSeckillGoodsExample();  
 TbSeckillGoodsExample.Criteria criteria = example.createCriteria();  
 *//库存数量>0* criteria.andStockCountGreaterThan(0);  
 *//活动开始时间 <=当前时间< 活动结束时间* Date date = **new** Date();  
 criteria.andStartTimeLessThanOrEqualTo(date); *//活动开始时间<=now()* criteria.andEndTimeGreaterThan(date); *//活动结束时间>now()  
  
 //批量查询所有缓存数据，增加到Redis缓存中* List<TbSeckillGoods> goods = **seckillGoodsMapper**.selectByExample(example);  
  
 *//将商品数据加入到缓存中* **for** (TbSeckillGoods good : goods) {  
 *//秒杀商品信息加入缓存* **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).put(good.getId(),good);  
 }  
 }  
}

创建applicationContext-task.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:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 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/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/task  
 http://www.springframework.org/schema/task/spring-task.xsd"**>  
  
 *<!--包扫描-->* <**context:component-scan base-package="cn.itcast.pinyougou.task"**/>  
  
 *<!--开启注解驱动-->* <**task:annotation-driven**/>  
  
</**beans**>

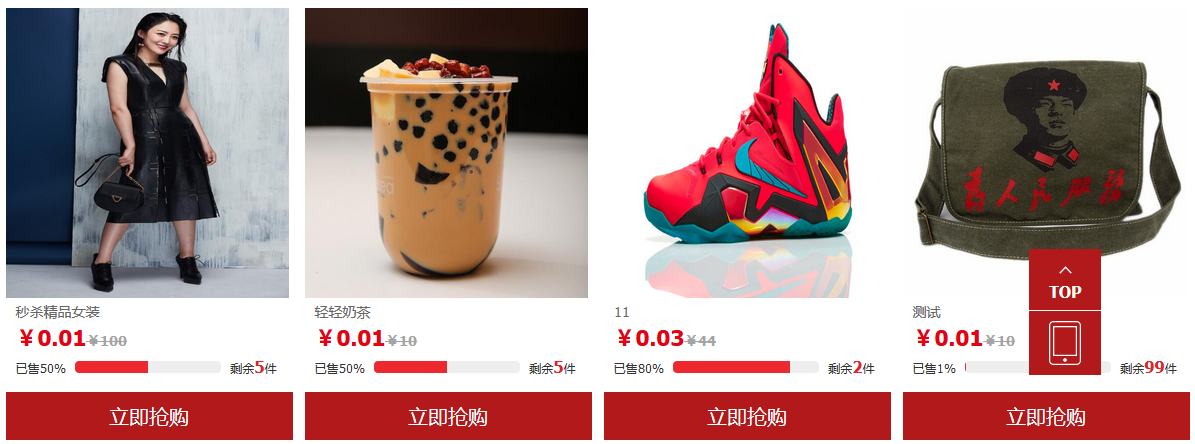
创建测试类：

**package** cn.itcast.pinyougou.test.task;  
  
**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.springframework.test.context.ContextConfiguration;  
**import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
  
**import** java.io.IOException;  
  
@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(**"classpath\*:spring/applicationContext-\*.xml"**)  
**public class** SeckillGoodsToRedisTest {  
  
 @Test  
 **public void** importToRedis() **throws** InterruptedException {  
 **while**(**true**){  
 **try** {  
 System.***in***.read();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

# 品优购-秒杀频道首页

## 需求分析

秒杀频道首页，显示正在秒杀的商品（已经开始，未结束的商品）



## 前端代码实现

### 修改seckill-index.html,引入js

|  |
| --- |
| <**script src="/plugins/angularjs/angular.min.js"**></**script**> |

### 绑定指令

|  |
| --- |
| <**body ng-app="pyg" ng-controller="seckillGoodsController" ng-init="findAll()"**> |

### 创建模块和控制器

<**script**>  
 **var *app*** = **angular**.**module**(**'pyg'**,[]);  
 ***app***.controller(**'seckillGoodsController'**, **function** ($scope, $http) {  
 $scope.findAll = **function** () {  
 $http.get(**'seckillGoods/findAll'**).success(**function** (res) {  
 $scope.**list** = res;  
 });  
 }  
 });  
</**script**>

### 绑定列表展示

<**ul class="seckill" id="seckill"**>  
 <**li class="seckill-item" ng-repeat="item in list"**>  
 <**div class="pic" onclick="*location***.href=**'seckill-item.html'"**>  
 <**img src="{{*item*.smallPic}}" alt='' width="283" height="290"** >  
 </**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)/***item***.num\*100).toFixed(0)}}%</**div**>  
 <**div class='progress'**>  
 <**div class='sui-progress progress-danger'**><**span style='width**: {{((***item***.**num-*item***.**stockCount**)/***item***.**num**\*100).**toFixed**(0)}}%;**' 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' href='/seckill-item.html#?id={{*item*.id}}'**>立即抢购</**a**>  
 </**li**>  
  
</**ul**>

## 后端代码

### 创建Controller类

**package** cn.itcast.pinyougou.controller;  
  
**import** cn.itcast.pinyougou.pojo.TbSeckillGoods;  
**import** cn.itcast.pinyougou.service.SeckillGoodsService;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** javax.annotation.Resource;  
**import** java.util.List;  
  
@RestController  
@RequestMapping(**"/seckillGoods"**)  
**public class** SeckillGoodsController {  
  
 @Resource  
 **private** SeckillGoodsService **seckillGoodsService**;  
  
 @RequestMapping(**"/findAll"**)  
 **public** List<TbSeckillGoods> findAll(){  
 **return seckillGoodsService**.findAll();  
 }  
}

### 创建Service接口和类

SeckillGoodsService.java

**package** cn.itcast.pinyougou.service;  
  
**import** cn.itcast.pinyougou.pojo.TbSeckillGoods;  
  
**import** java.util.List;  
  
**public interface** SeckillGoodsService {  
 List<TbSeckillGoods> findAll();  
}

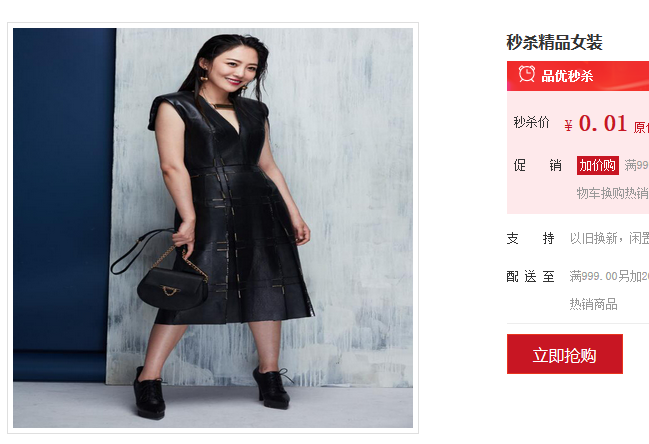
SeckillGoodsServiceImpl.java

**package** cn.itcast.pinyougou.service.impl;  
  
**import** cn.itcast.pinyougou.mapper.TbSeckillGoodsMapper;  
**import** cn.itcast.pinyougou.pojo.TbSeckillGoods;  
**import** cn.itcast.pinyougou.service.SeckillGoodsService;  
**import** org.springframework.stereotype.Service;  
**import** org.springframework.transaction.annotation.Transactional;  
  
**import** javax.annotation.Resource;  
**import** java.util.List;  
  
@Service  
@Transactional  
**public class** SeckillGoodsServiceImpl **implements** SeckillGoodsService {  
  
 @Resource  
 **private** TbSeckillGoodsMapper **seckillGoodsMapper**;  
  
 @Override  
 **public** List<TbSeckillGoods> findAll() {  
 **return seckillGoodsMapper**.selectByExample(**null**);  
 }  
}

# 品优购-秒杀详细页

## 需求分析

商品详细页显示秒杀商品信息。



## 显示详细页信息

### 前端代码

修改seckill-index.html

<**div class="pic" ng-click="jumpToItem(item.id)"**>

指令

|  |
| --- |
| <**body ng-app="pyg" ng-controller="seckillGoodsController" ng-init="findOne()"**> |

引入js文件

<**script src="/plugins/angularjs/angular.min.js"**></**script**>

创建模块和控制器

<**script**>  
 **var *app*** = **angular**.**module**(**'pyg'**,[]);  
 ***app***.controller(**'seckillGoodsController'**, **function** ($scope, $http, $location) {  
 $scope.findOne = **function** () {  
 $http.get(**'seckillGoods/findOne/'**+$location.search().**id**).success(**function** (res) {  
 $scope.**item** = res;  
 });  
 }  
 });  
</**script**>

用表达式显示标题

|  |
| --- |
| <h4>{{item.title}}</h4> |

图片

|  |
| --- |
| <span class=*"jqzoom"*><img jqimg=*"{{item.smallPic}}"* src=*"{{item.smallPic}}"* **style="width**:400**px**;**height**:400**px"**  /></span> |

价格

|  |
| --- |
| <div class=*"fl price"*><i>¥</i>  <em>{{item.costPrice}}</em>  <span>原价：{{item.price}}</span>  </div> |

介绍

|  |
| --- |
| <div class=*"intro-detail"*>{{item.introduction}}</div> |

### 后端代码

SeckillGoodsController.java

@RequestMapping(**"/findOne/{id}"**)  
**public** TbSeckillGoods findOne(@PathVariable(**"id"**) Long id){  
 **return seckillGoodsService**.findOne(id);  
}

SeckillGoodsService.java

TbSeckillGoods findOne(Long id);

SeckillGoodsServiceImpl.java

@Override  
**public** TbSeckillGoods findOne(Long id) {  
 **return seckillGoodsMapper**.selectByPrimaryKey(id);  
}

## 秒杀倒计时效果

### $interval服务简介（用来实现秒杀倒计时）

在AngularJS中$interval服务用来处理间歇性处理一些事情

格式为：

|  |
| --- |
| $interval(执行的函数,间隔的毫秒数,运行次数); |

运行次数可以缺省，如果缺省则无限循环执行

取消执行用cancel方法

|  |
| --- |
| $interval.cancel(time); |

### 秒杀倒计时

修改seckillGoodsController.js

***app***.controller(**'seckillGoodsController'**, **function** ($scope, $http, $location, $interval) {  
 $scope.findOne = **function** () {  
$http.get(**'seckillGoods/findOne/'**+$location.search().**id**).success(**function** (res) {  
 $scope.**item** = res;  
 *//计算出剩余时间* **var** endTime = **new *Date***($scope.**item**.endTime).getTime();  
 **var** nowTime = **new *Date***().getTime();  
  
 *//剩余时间* $scope.**secondes** =***Math***.floor( (endTime-nowTime)/1000 );  
  
 **var** time =$interval(**function** () {  
 **if**($scope.**secondes**>0){  
 *//时间递减* $scope.**secondes**=$scope.**secondes**-1;  
 *//时间格式化* $scope.**timeString**=$scope.convertTime2String($scope.**secondes**);  
 }**else**{  
 *//结束时间递减* $interval.cancel(time);  
 }  
 },1000);  
 });  
 }  
 *//时间计算转换* $scope.convertTime2String=**function** (allseconds) {  
 *//计算天数* **var** days = ***Math***.floor(allseconds/(60\*60\*24));  
  
 *//小时* **var** hours =***Math***.floor( (allseconds-(days\*60\*60\*24))/(60\*60) );  
  
 *//分钟* **var** minutes = ***Math***.floor( (allseconds-(days\*60\*60\*24)-(hours\*60\*60))/60 );  
  
 *//秒* **var** seconds = (allseconds-(days\*60\*60\*24)-(hours\*60\*60)-(minutes\*60)).toFixed(0);  
  
 **if**(seconds < 10){  
 seconds = **"0"**+seconds;  
 }  
 *//拼接时间* **var** timString=**""**;  
 **if**(days>0){  
 timString=days+**"天:"**;  
 }  
 **return** timString+=hours+**"小时:"**+minutes+**"分钟:"**+seconds+**"秒"**;  
 }  
 });

修改页面seckill-item.html ，显示time的值

|  |
| --- |
| <span class=*"overtime"*> 距离结束：{{timeString}}</span> |

# 品优购-秒杀下单

## 需求分析

商品详细页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。

秒杀下单业务流程（从立即抢购开始下单）：

1），从redis服务器中获取入库的秒杀商品

2），判断商品是否存在，或是是商品库存是否小于等于0直接结束。

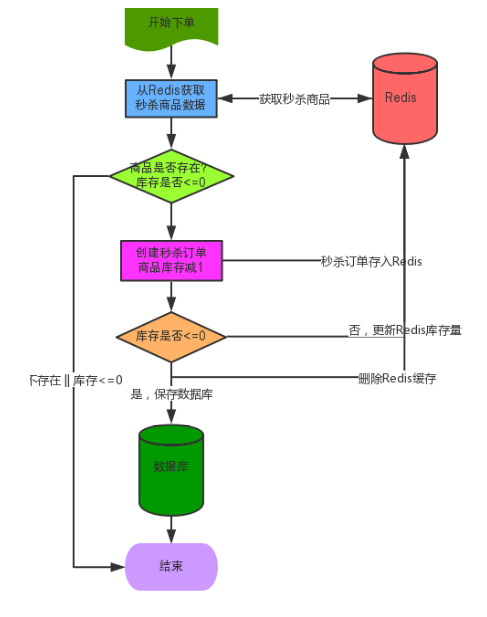
3），如果秒杀商品存在，创建秒杀订单

4），把新增订单存储在redis服务器中（后期可以用多线程将redis中的订单保存到数据库）

5），把存储在redis中入库的商品库存减一（数据库会每30s执行一下）

6），判断库存是否小于0,卖完需要同步数据库（小于0的话，说名卖完了，删除redis缓存，同时更新数据库）

7），否则同步redis购物车数量



## 前端代码

修改seckill-item.html

|  |
| --- |
| <**a href="javascript:void(0)" ng-click="saveOrder()" target="\_self" class="sui-btn btn-danger addshopcar"**>立即抢购</**a**> |

修改控制器：

$scope.saveOrder = **function** () {  
$http.get(**'seckillGoods/saveOrder/'**+$scope.**item**.**id**).success(**function** (res) {  
 *alert*(res.**message**);  
 **if**(res.**success**){  
 ***location***.href=**'pay.html'**;  
 }  
});

}

## 后端代码

### 控制层

SeckillOrderController.java

|  |
| --- |
| @RequestMapping(**"/saveOrder/{id}"**) **public** Result saveOrder(@PathVariable(**"id"**) Long id){  String userId = **"jiuwenlong"**;*//本示例未实现登录功能，假设登录用户是jiuwenlong* **return seckillGoodsService**.saveOrder(id, userId); } |

### 服务接口层

SeckillOrderService.java

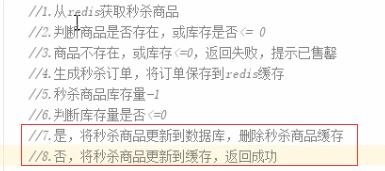
|  |
| --- |
| Result saveOrder(Long id, String userId); |

### 服务实现层

Spring配置文件配置IdWorker

<**bean id="idWorker" class="cn.itcast.pinyougou.utils.IdWorker"**>  
 <**constructor-arg index="0" value="0"**/>  
 <**constructor-arg index="1" value="0"**/>  
</**bean**>

SeckillOrderServiceImpl.java实现方法



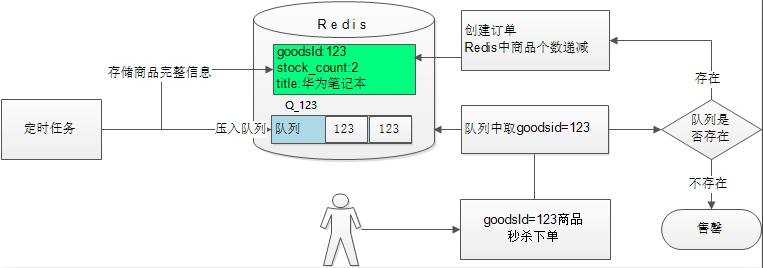
|  |
| --- |
| @Resource **private** RedisTemplate **redisTemplate**; @Resource **private** IdWorker **idWorker**;  @Override **public** Result saveOrder(Long id, String userId) {  *//1.从redis获取商品* TbSeckillGoods seckillGoods = (TbSeckillGoods) **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).get(id);  *//2.判断商品为null或库存<=0，返回商品已售罄* **if**(**null** == seckillGoods || seckillGoods.getStockCount() <= 0){  **return new** Result(**false**, **"对不起，商品已售罄，请查看其他商品！"**);  }  *//3.创建秒杀订单* TbSeckillOrder seckillOrder = **new** TbSeckillOrder();  seckillOrder.setCreateTime(**new** Date());  seckillOrder.setMoney(seckillGoods.getCostPrice());  seckillOrder.setSeckillId(**idWorker**.nextId());  seckillOrder.setSellerId(seckillGoods.getSellerId());  seckillOrder.setUserId(userId);  *//4.秒杀订单存入缓存，库存-1* **redisTemplate**.boundHashOps(TbSeckillOrder.**class**.getSimpleName()).put(userId, seckillOrder);  seckillGoods.setStockCount(seckillGoods.getStockCount() - 1);  *//5.订单存入缓存后，判断库存是否<=0（注意这里是操作后再判断缓存）* **if**(seckillGoods.getStockCount() <= 0){  *//5.1是，更新秒杀商品，保存秒杀订单，删除缓存（缓存中库存为0的时候更新数据库）* **seckillGoodsMapper**.updateByPrimaryKey(seckillGoods);  **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).delete(seckillGoods.getId());  } **else** {  *//5.2否，更新秒杀商品缓存（上面虽然库存-1是说的获取后减去1并没有更新到redis缓存中）* **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).put(id, seckillGoods);  }  //可以直接返回这个，上面的4，5，6逻辑用多线程异步去处理  **return new** Result(**true**, **"秒杀成功，请您尽快支付！"**); }  上面的代码在多线程的情况下会产生超卖的问题 |

# 超卖问题解决

上面那种方案在没有并发情况下是可以的，但秒杀一般是具备大量并发，并发时就有可能出现超卖问题。

|  |
| --- |
| *//获取商品详情* SeckillGoods seckillGoods = (SeckillGoods) **redisTemplate**.boundHashOps(SeckillGoods.**class**.getSimpleName()).get(seckillid); **if**(goodsId==**null**){  **throw new** RuntimeException(**"已售罄！"**); } |

上面保存订单的方式是先查看Redis中对应商品是否存在，如果存在且数量是否>0如果>0则下单，如果在并发情况下,如果20个人同时在执行如上查询代码这里，而此时对应商品只有一个，则会下20个单，而这20个单一定是有问题的，因为1件商品不可能同时给20个人发货。那么如何解决这种并发问题呢？我们可以用Redis队列实现。



## 数据导入Redis队列操作

修改pyg-seckill-task中SeckillTask.java

|  |
| --- |
| @Component **public class** SeckillTask {   */\*\*\*  \* 每年双十一启动秒杀  \* 将商品数据全部跟新到索引库  \*/* @Scheduled(cron = **"30 \* \* \* \* ?"**) *//每天上午10点15分出发一次* **public void** startSeckill(){  //*......略* List<SeckillGoods> goods = **seckillGoodsMapper**.selectByExample(example);   *//将商品数据加入到缓存中* **for** (SeckillGoods good : goods) {  *//秒杀商品信息加入缓存* **redisTemplate**.boundHashOps(SeckillGoods.**class**.getSimpleName()).put(good.getId(),good);   *//给每个商品加入到Redis队列中，秒杀对应商品有多少个，则加多少个ID到队列中* pushSeckillGoods(good);  }  }    */\*\*\*  \* 给每个商品加入到Redis队列中  \* 秒杀对应商品有多少个，则加多少个ID到队列中  \** ***@param goods*** *\*/* **private void** pushSeckillGoods(SeckillGoods goods){  *//库存量* Integer stockCount = goods.getStockCount();  *//循环加入Redis队列  //左压栈方式加入* **for** (**int** i = 0; i <stockCount ; i++) {  **redisTemplate**.boundListOps(SysContant.***SECKILL\_PREFIX***+goods.getId()).leftPush(goods.getId());  }  } } |

在pyg\_common下创建SysContant.***SECKILL\_PREFIX是定义的常量，这里不建议写死。***

|  |
| --- |
| *//秒杀商品前缀* **public static final** String ***SECKILL\_PREFIX***=**"SECKILL\_PREFIX\_GOODSID\_"**; |

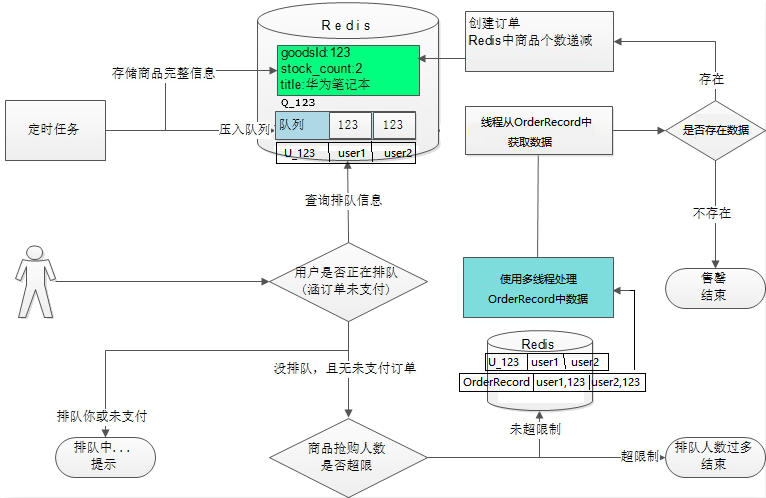
## 秒杀下单优化

修改pyg-seckill-service的SeckillOrderServiceImpl.java，加入从队列中取数据校验商品是否存在的实现过程,队列中商品存在，则继续下单，否则抛出异常提示已售罄。

|  |
| --- |
| */\*\*\*  \* 创建订单  \** ***@param seckillid*** *\** ***@param userid*** *\*/* @Override **public Result** saveOrder(Long seckillid, String userid) {  *//获取队列中的商品，如果能够获取，则商品存在，可以下单  //这样可以避免多个用户同时抢购意见商品重复下单* Long goodsId = (Long) **redisTemplate**.boundListOps(SysContant.***SECKILL\_PREFIX*** + seckillid).rightPop();  **if**(goodsId==**null**){  **return new** Result(**false**, **"对不起，商品已售罄，请查看其他商品！"**);  }   *//获取商品详情* SeckillGoods seckillGoods = (SeckillGoods) **redisTemplate**.boundHashOps(SeckillGoods.**class**.getSimpleName()).get(seckillid);  *//略....* } |

# 并发问题解决

上面的方案解决了并发情况下下单操作异常问题，但其实际秒杀中大量并发情况下，这个下单过程是需要很长等待时间的，所以这里我们建议用异步和多线程实现，最好不要让程序处于阻塞状态，而是在用户一下单的时候确认用户是否符合下单条件，如果符合，则开启线程执行，执行完毕之后，用户等待查询结果即可。



每个用户在下单的时候先去看看redis中用户set（防止用户重复提交）是否有这个用户，有的话，提醒去排队或者去支付，没有的话，再去判断商品抢购人数是否超限（还是到redis中去看每个商品的队列（防止超卖）），如果队列中还有商品id说明没有超出限制，没有超出限制就将用户假如用用户set中（去记录当前这个用户下单成功了），然后将用户的下单信息存到令一个队列中。到此直接给客户返回秒杀成功。然后把订单队列交给多线程处理。

## 创建线程下订单

创建CreateOrderThread.java实现订单下单操作,在面试中常常会被问及多线程应用在项目哪里，这正好是一个很好的案例。

@Component  
**public class** CreateOrderThread **implements** Runnable {  
  
 @Resource  
 **private** RedisTemplate **redisTemplate**;  
 @Resource  
 **private** IdWorker **idWorker**;  
 @Resource  
 **private** TbSeckillGoodsMapper **seckillGoodsMapper**;  
  
 @Override  
 **public void** run() {  
 OrderRecord orderRecord = (OrderRecord) **redisTemplate**.boundListOps(OrderRecord.**class**.getSimpleName()).rightPop();  
 **if**(**null** != orderRecord){  
 Long id = orderRecord.getSeckillid();  
 String userid = orderRecord.getUserid();  
 TbSeckillGoods seckillGoods = (TbSeckillGoods) **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).get(id);  
 *//3.未售罄，创建订单，以用户id为key存入redis* TbSeckillOrder seckillOrder = **new** TbSeckillOrder();  
 seckillOrder.setId(**idWorker**.nextId());  
 seckillOrder.setSeckillId(id);  
 seckillOrder.setMoney(seckillGoods.getCostPrice()); *//秒杀价格* seckillOrder.setUserId(userid);  
 seckillOrder.setSellerId(seckillGoods.getSellerId());  
 seckillOrder.setCreateTime(**new** Date());  
 seckillOrder.setStatus(**"0"**);  
  
 **redisTemplate**.boundHashOps(TbSeckillOrder.**class**.getSimpleName()).put(userid, seckillOrder);

//减去1操作，多进程并发  
 **synchronized** (CreateOrderThread.**class**){  
 seckillGoods = (TbSeckillGoods) **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).get(id);  
 *//4.更新库存，判断库存是否售罄* seckillGoods.setStockCount(seckillGoods.getStockCount() - 1);  
 **if**(seckillGoods.getStockCount() <= 0){  
 *//5.售罄，同步秒杀商品数据库（seckillGoods），将秒杀商品从redis中删除* **seckillGoodsMapper**.updateByPrimaryKeySelective(seckillGoods);  
 **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).delete(seckillGoods.getId());  
 } **else** {  
 *//6.未售罄，更新redis中秒杀商品库存* **redisTemplate**.boundHashOps(TbSeckillGoods.**class**.getSimpleName()).put(seckillGoods.getId(), seckillGoods);  
 }  
 }  
 }  
 }  
}

## 配置线程池

|  |
| --- |
| *<!--  线程池配置 -->* <**bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor" id="executor"**>  *<!-- 核心线程数，默认为1-->* <**property name="corePoolSize" value="10"** />  *<!--最大线程数，默认为Integer.MAX\_VALUE-->* <**property name="maxPoolSize" value="50"** />  *<!--队列最大长度，一般需要设置值>=notifyScheduledMainExecutor.maxNum；默认为Integer.MAX\_VALUE-->* <**property name="queueCapacity" value="10000"** />  *<!--线程池维护线程所允许的空闲时间，默认为60s-->* <**property name="keepAliveSeconds" value="300"** />  *<!--线程池对拒绝任务（无线程可用）的处理策略，目前只支持AbortPolicy、CallerRunsPolicy；默认为后者-->* <**property name="rejectedExecutionHandler"**>  <**bean class="java.util.concurrent.ThreadPoolExecutor$CallerRunsPolicy"** />  </**property**> </**bean**> |

线程池对拒绝任务的处理策略：

CallerRunsPolicy ：

这个策略重试添加当前的任务，他会自动重复调用 execute() 方法，直到成功。

AbortPolicy ：

对拒绝任务抛弃处理，并且抛出异常。

DiscardPolicy ：

对拒绝任务直接无声抛弃，没有异常信息。

DiscardOldestPolicy ：

对拒绝任务不抛弃，而是抛弃队列里面等待最久的一个线程，然后把拒绝任务加到队列。

## 下单保存修改

修改SeckillOrderServiceImpl.java，把之前实现的保存订单修改成启动线程调用

|  |
| --- |
| @Resource **private** RedisTemplate **redisTemplate**;  @Resource//这是线程池 **private** ThreadPoolTaskExecutor **executor**;  @Resource **private** CreateOrderThread **createOrderThread**; @Override  **public void** saveOrder(Long id, String userid) {  *//1.判断用户是否在排队队列* Boolean isMember = **redisTemplate**.boundSetOps(SysConsts.***SECKILL\_USER***+id).isMember(userid);  **if**(isMember){  TbSeckillOrder seckillOrder = (TbSeckillOrder) **redisTemplate**.boundHashOps(TbSeckillOrder.**class**.getSimpleName()).get(userid);  *//1.1在排队，判断用户是否在订单队列中* **if**(**null** != seckillOrder){  *//1.1.1在订单队列，“您已抢购成功，请支付订单！”异常* **return new** Result(**false**, **"您已抢购成功，请支付订单！"**);  }  *//1.1.2不在订单队列，“您正在排队...”* **return new** Result(**false**, **"您正在排队，请耐心等待。。。"**);  }  *//2.判断商品是否售罄* Long goodsId = (Long) **redisTemplate**.boundListOps(SysConsts.***SECKILL\_PREFIX***+id).rightPop();  **if**(**null** == goodsId ){  *//2.售罄* **return new** Result(**false**, **"对不起，商品已售罄，请查看其他商品！"**);  }  **redisTemplate**.boundSetOps(SysConsts.***SECKILL\_USER***+id).add(userid);   **redisTemplate**.boundListOps(OrderRecord.**class**.getSimpleName()).leftPush(**new** OrderRecord(userid, id));  //让线程池去执行这个下单的过程  **executor**.execute(**createOrderThread**);  **return new** Result(**true**, **"秒杀成功，请您尽快支付！"**); } |
|  |

## 添加静态常量

**public static final** String ***SECKILL\_USER*** = **"SECKILL\_USER\_"**;*//保存用户id*

## 创建OrderRecord类

**package** com.pinyougou.vo;  
  
**import** java.io.Serializable;  
  
*/\*\*  
 \* 记录下单用户id和商品id  
 \*/***public class** OrderRecord **implements** Serializable{  
  
 **private** String **userId**;  
  
 **private** Long **id**;  
  
 **public** OrderRecord(String userId, Long id) {  
 **this**.**userId** = userId;  
 **this**.**id** = id;  
 }  
  
 **public** String getUserId() {  
 **return userId**;  
 }  
  
 **public void** setUserId(String userId) {  
 **this**.**userId** = userId;  
 }  
  
 **public** Long getId() {  
 **return id**;  
 }  
  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
}