1.项目所用技术

前台：

使用vue.js 构建 webapp

后台使用 ：

springboot

maven做版本控制

bootstrap + freemarker + jquery

前后台是用restfull接口进行关联

postman做测试工具

实体类用lombok进行封装

日志使用logback 或者 slf4j

mvc 架构模式进行架构本着多拆分，多封装的原则进行操作本项目

数据库:

使用mysql做数据存储

使用springboot+ jpa 和 springboot + MyBatis 进行数据库连接

缓存方面：

使用redis数据库

分布式锁

分布式session

消息推动方面： 使用 WebSocket

微信端操作：

微信扫描登录 模板消息推送 微信支付和退款。

2.开发思路

整个项目开发分为三部分。

用户端（查看商品信息，订单，秒杀）

微信（微信授权，模板消息推送,微信支付，微信退款）

商户端（管理系统。订单管理，商品管理，商户登入登出）

3.本人负责开发的部分

用户端：

商品信息的展示：

先建一个dataobject包使用@Entity注解把数据库中的表映射成对象）。

建一个vo类进行结果的返回，然后创建resultvo类进行结果的装配以及提示查询是否成功。

创建dao层和service层，dao层和service层每写一个方法都需要进行测试，测试采用slf的和断言。然后再创建controller进行接口的实现，方法的返回值为resultvo.然后利用posterman进行接口测试。在整个逻辑中抛出的异常都是自定义异常，所有异常都封装到枚举类中。

订单：

**包括创建订单：**

订单创建需要传递参数，所以我们建一个orderform类进行参数的传递。但是传递之前我们需要用jsr303校验是否有空值。然后创建controller方法，首先判断jsr303是否有误，如果有误则提示传递参数异常然后返回给前台。如果没有异常则把orderform转换为订单传输类orderdto（包括ordermaster信息和orderdetil信息），将orderdto传入service层进行订单创建的准备工作。由于订单创建完成之后要写入数据库所以要加入事务的操作。利用keyUtil工具类进行订单id的生成，然后利用查询orderdto中是否有商品存在，如果存在商品则创建orderid，并将属性进行赋值，然后计算订单的总价，定义支付状态的枚举类和订单状态的枚举类，最后将订单信息进行赋值写入数据库，订单创建完成。然后再进行库存的减少操作，库存减少只用到了商品的id和购买数量，所以再创建一个购物车传输类进行封装，然后把orderdto中的商品id和数量值赋给购物车，进行库存减少的操作。

**查询订单：**

分为查询单个订单和查询订单列表，直接调用相对应的方法即可，注意的是，查询订单列表用到了分页查询，由于page参数类型不一样所以建了一个数据类型转化的类里面用到了lamda表达书进行数据筛选。

取消订单：

首先判断订单状态，修改订单状态，返还库存，用户付完款的还要进行退款操作。

完结订单：

判断订单状态，修改订单状态

支付订单：

判断订单状态，判断支付状态，修改支付状态。

（取消，完结，支付都需要加事务）

秒杀：

为了减少对数据库的访问，通过redis实现了缓存的优化；并通过rabbitMQ消息中间件来接收大量的并发请求；实现异步和销峰，然后再慢慢更新数据库。然后用Jmeter测试优化前后的QPS.(每秒查询率)。

Reids,Jemter,RabbitMQ的相关配置是部署在虚拟机上运行。（项目经理已经部署好配置和环境）。

秒杀系统页与其他业务的最大区别在于秒杀的瞬间1，系统的并发量会非常大2，并发量大的同时，网络的流量也会瞬间变大

解决并发量大的问题减少对DB的访问，如何减少对DB的访问，秒杀活动开始之前把商品的goodsId和数量加载到Redis缓存。服务端收到请求的时候，首先预减一下redis的数量，如果数量减到0随后的访问直接返回秒杀失败。这是解决只有一个秒杀商品或者少数秒杀商品，如果同时有大量的商品参与秒杀这种解决方案还存在不合理，DB的压力还是会非常大。这时候我们可以用RabbitMQ消息队列，做一个缓存，然后再慢慢的更新数据库，这样做以后前端用户的请求可能不会立即得到相应是成功还是失败，很可能会得到一个排队中的返回值，这个时候需要客户端再去服务端做轮询，因为我们不能一定保证秒杀成功了。当服务端出队，生成订单之后，把用户id和商品goodsid写进缓存中，来应对客户端的轮询。解决第二个问题做页面静态化（也就是前后端的分离）；把静态页面直接缓存到用户浏览器，所需要的数据从服务器接口动态获取，再加上CDN,一般不会有大问题。

同时我还做了一个按钮控制：也就是在秒杀时间开始之前，按钮置灰，只有时间到了才能点击。具体就是前端配合，定时去请求你的后端服务器，获取最新的北京时间，到时间再给按钮可用状态。另外为了前端限流，我设计了不能一致点击秒杀按钮，必须3秒后才能重新点击。

秒杀用到的技术：

1. 使用分布式session。解决分布式的多台服务器sessiion丢失的情况，利用redis缓存使用一个redis服务器专门存放用户的session信息，这样就不会出现用户seesion丢失的情况（每次需要session，从缓存中取就行）。每次登录都会生成一个tokken并把它加到Cookie中去，在跳转不同页面的时候，会把token对应的user从redis中取出。
2. 使用redis做缓存提高访问速度和并发量，减少数据库的压力，利用内存标记减少redis的访问。 用户信息缓存（分布式的session），商品信息的缓存，商品库存的缓存，订单的缓存，页面的缓存。由于用了大量的缓存会造成无法识别不同模块的缓存的问题。（key值重复，如何辨别是不同模块的key）。我做了一个抽象类，定义了BaseKey前缀以及缓存的过期时间从而将缓存Key进行封装，让不同的模块继承他，这样每次存入一个模块的缓存的时候，加上这个缓存特定的前缀，还可以统一制定不同的过期时间。
3. 页面静态化（前后端的分离），目的是加快页面的加载速度，将商品的详情页和订单的详情页做成静态页面（HTML）放在客户端的浏览器,数据的加载只需通过ajax请求。减少对后台服务器的访问，
4. 消息队列完成异步下单，提升用户体验，销峰和限流

秒杀活动开始前，把商品的id和数量加载到redis中，

后端收到秒杀请求，redis预检库存，如果库存为0，后面的请求直接返回秒杀失败。

判断这个订单形成没有，判断是否秒杀到了，避免一个用户秒杀多个商品，

库存充足，而且没用重复秒杀，将秒杀请求封装入队列，同时返回给前台一个返回值，表示在队列中。

前台收到返回值之后，显示在排队中，再根据商品id进行论讯。

后端RabbirMQ监听秒杀队列这个通道，如果有消息过来，判断数据库的库存，判断是否重复秒杀，然后执行秒杀事务（减库存，写入秒杀订单消息）。

前端根据商品id的论讯请求接口。查看是否生成了订单，返回结构-1，秒杀失败，返回结果0排队中，返回>0秒杀成功。

秒杀中遇到可能遇到的问题以及解决方案

1，大量的缓存会遇到缓存穿透，缓存雪崩，缓存一致性

解决缓存穿透，对不存在的数据缓存Null,设定过期时间。

缓存雪崩（同一时间缓存大面积失效，或者缓存服务器宕机），对于同一时间缓存大面积失效设置合理的过期时间来实现，对于缓存服务器宕机，可以使用分布式缓存，分布式缓存每一个节点只缓存部分数据，当某个节点宕机，其他节点的缓存仍然可以使用

缓存的一致性数据库更新的同时立即去更新缓存，在读取缓存之前，先判断是不是最新的，如果不是最新的，先更新数据库，然后把缓存里对应的数据失效掉。再去读缓存，缓存中没有，就去读数据库，同时把返回的数据写到缓存中。

2，大量使用缓存，对缓存服务器也有很大压力，如何减少对redis的访问。

在redis预减库存的时候，内存中维护一个isOvermap作为一个内存标记，当没有库存的时候，将其设置为true,每次秒杀业务访问redis之前，查一下map标记，如果是true，说明没有库存，直接返回，无需再去访问redis服务器

3，在高并发请求的业务场景，大量请求来不及处理，甚至出现请求堆积。

消息队列，用来异步处理请求。每次请求过来先不处理请求，放入消息队列，然后在后台布置一个监听器，分别监听不同业务的消息队列，当有秒杀消息来的时候，才进行秒杀业务逻辑，连接数据库。

4，怎么保证一个用户不能重复下单

在秒杀订单表中建立一个唯一索引（索引是用用户id与商品id），使得第一个记录可以插入，第二个出错，然后通过事务回滚，防止一个用户同时发出多个请求，秒杀到多个商品。

5.怎么解决超卖现象

最简单的方法，更新数据库库存的时候，进行库存条件限制，在减库存的方法里sql要多加一个stock\_count>0.使用数据库的特性来保证超卖问题，只有stock——count大于0的时候在取读stock\_count然后进行减一操作。

6，假如见了库存的用户没有支付，库存怎么还原继续参加抢购

设定一个最长付款时间比如15分钟，后台有个定时任务，论寻超过15分钟待付款的订单（数据库里判断订单状态），然后关闭订单，恢复库存。

商户端：

商户的登录和登出：

微信扫码登录（开放平台），需要传入两个参数openappid和openappsecret,调用controller里面的方法登陆页面，并且在url里返回openid。

1. 判断Openid和数据库中的openid是否一致
2. 设置tokken到redis中（获取tokken使用UUID，redis中存储数据一定要设置过期时间）
3. 设置tokken到cookie中。登录成功

Controller中所有调用的方法都是sdk第三方提供的，可以直接在pom中引入相关依赖即可

登出：

把redis和cookie删除即可。

本模块使用了aop进行身份验证（登录拦截），如果验证不通过返回登陆页面。

订单管理：

订单查询：

使用[www.ibootstrap.cn](http://www.ibootstrap.cn)生成订单列表展示的页面。然后传入参数商家的openid和分页查询的参数，进行分页展示。

每条订单后面有两个超链接分别是订单详情和取消订单。都是直接调用orderservice中的相关方法即可。

如果订单状态是新订单，会有完结订单和取消订单的操作。也都是直接调用orderservice中的相关方法。

商品管理：

类目管理：

包括类目的修改和新增，判断categoryid是否为空，为空则新增，不为空则修改。

商品管理：

商品列表：

直接调用productservice里面的方法即可。

商品的上下架：

在controller中写两个方法，对商品的状态进行修改即可。

商品的新曾和修改

判断productid是否为空，为空则新增，不为空则修改。