2019
版

治愈系 Java 工程师面试指导课程

[Part10・电商项目业务和技术点话术]

[本教程梳理了电商项目的业务和技术话术，已经开发过程中的高频面试题；建议大家将目录结构打开，对照目录结构做一个复习的、提纲准备面试。]

1 项目的架构话术

这个项目采用的 soa 分布式架构模式，soa 是一种面向服务的架构模式。soa 架构模式特点就是将整个服务分为服务层和视图层，服务层每一个服务对应一个 war 包，视图层也是单独运行的工程，服务层和视图层就涉及到了通信问题，通信过程采用了 dubbo 框架。

Dubbo 是一个分布式架构的服务框架，一般结合 maven 模块进行开发。传统的单架构项目，不利于项目维护和升级；通过 maven 开发就可以把一个单架构项目拆成一个一个小模块，可以是 war 包，也可以是 jar 包，jar 包可以直接被 war 包使用，war 可以分布式部署，但是部署之后 war 包之间交互成了问题，我们之前做法是，两个 war 包交互时，一般采用 webservice 和 HTTPClient，但是在这种分布式架构中，多个 war 之间交互会非常频繁，如果采用 webservice 和 HTTPClient，架构师可能自己都搞不清楚他们之间的依赖关系了。而 dubbo 解决了这个问题，有了 dubbo 架构之后，多个 war 之间不直接进行交互，而是统一由 dubbo 的注册中心 zookeeper 进行交互，无论是发布服务还是调用服务，都通过 dubbo 实现，并且提供了 web 页面来监控管理各个接口的调用。使用时首先要搭建 dubbo 的注册中心 zookeeper，然后下载 dubbo 服务治理工程 dubbo_admin.war 包，解压并修改他配置文件 dubbo.properties，设置用户名，密码和注册中心 zookeeper 的地址 ip，端口。部署到 tomcat 的 webapp 中，启动 tomcat 后，在浏览器用 ip+port 访问就可以看到注册中心 zookeeper 的管理页面了。接着在我们要发布或者调用服务的项目 pom.xml 中导入 dubbo 相关 jar 包，加载配置文件，把要发布的服务在注册中心 zookeeper 注册，或者从 zookeeper 中调用服务，在 dubbo 架构中，service 的接口是共享的，service 的实现类是发布服务工程独有的，只需要发布服务的工厂把他的接口注册到 zookeeper 中，其他工程就在 zookeeper 可以调用了。）

2 电商商品管理详细话术

2.1 类目管理

类目包括后台类目树、前台类目树、前台类目与后台类目的映射。

后台类目与创建商品相关，每个商品在后台添加时均需选择相应的后台类目。后台类目树一般采用三级经典三级联动的方式经行表设计(这里需要注意的是:需要在类目表里添加一个判断是否是叶子类目,如果是叶子类目则不能继续添加分类)，这里采用三级分类因为过多可能导致冗杂，管理起来也不方便。类目树的最后一级类目称为叶子类目。 比如:食品为一级类目，饮料为二级类目，纯牛奶则是三级类目，也即叶子类目。【里主要介绍:关于三级分类自关联表的设计】

2.1 规格模板维护

对于商品模板信息是为了商家后台上传了一个新商品时能够约束一下这个商品的一些描述信息，就像京东上商品信息，详情展示时，每个分类会统一的模板，比如主板信息，外观信息。对于模板有两种实现思路

第一种实现思路是设计多张表结构,比如有一个商品表，还得有一个商品类别表，商品组表，商品键表，商品模板表。其中商品组表中是对应的每类商品的模板，商品键表是商品模板中又有好多小分类，比如外观下还有尺寸，重量这些具体信息，商品模板表，设计用来表示具体的商品在对应键下的具体信息。这种思路在我一开始开发的时候，用过，但是其中涉及的表过多，对数据库压力蛮大的，后来又换了一种思路。

第二种实现思路是使用 Json 形成模板方式,在涉及数据库表时，商品表，类别表这些要有，然后设计一个分类模板表，就是每一种分类都有一个 json 形式的模板，然后设计了一个商品模板信息表，就是每一个商品都有对应的模板信息。

这样在后台进行商品管理的时候，可以请求出分类内容，然后选中其中一个分类，对这个分类添加模板信息，然后以 json 形式添加到数据库中，比如手机对应的模板应该有外观，外观下有分类 颜色 尺寸，还有配置，配置下有机身存储和缓存这些信息。当添加商品时，会根据其具体的分类信息，请求下对应的模板，根据模板中显示的 key 值，让商家或者后台管理员添加对应的商品信息,比如对手机模板，将对应的机身存储和缓存信息填上，提交到后台以 json 的形式进行存储，这样商品的信息就全了。

2.2 模板管理

运营商的后台通过一个模板的管理来关联到叶子类目,品牌,属性等来创建一个基本的属性模板,这个模板里包含了模板 ID,分类,品牌,规格,扩展属性等字段。通过模板可以看出他有哪些品牌,哪些属性等信息;

2.3 新增商品

2.3.1 表设计

在表设计方面,关于商品添加我们用到了 10 左右的表,在表的设计结构上,我们具体做的时候主要采用了 SKU 和 SPU 表设计,比如说:"品牌表,规格表,模板表,分类表等"分类表采用了三级分类。商品模板表:是商品添加的重中之重,在模板表中商家自定义商品的扩展属性,规格,品牌等。

做商品添加这一块,表不多是因为我们把重点放在表结构上,维护关联关系的使用的是 json,在分类表关联了模板 ID,当添加商品时,就直接关联了商品有哪些品牌,规格,哪些扩展属性等;

SPU(标准产品单位)表关联的是商品的基本信息,SKU(库存量最小单位)示关联的是商品的规格属性详细信息;在这当中 SPU 和 SKU 属于一对多的关系;为了提高数据库的查询效率，我们将 spub 表做合理拆分，将数据量比较大的字段都放到 spu 描述表中比如商品描述、图片地址、包装描述等

2.3.2 商品添加的业务流程

在电商中商品添加业务逻辑比较复杂,在这一块我们用到一个大页面,在页面中进行了复杂的数据操作
商品添加的大页面中,因为商品信息比较多,我们将页面分了四个页签,每一个页签存储不同的信息;

2.3.2.1 第一个页签:

装的是商品的基本信息,商品的分类,品牌的选择,商品的价格和商品的详细介绍:在这一页面中,商品的分类我们使用了 angularJS 的 watch 来监听数据的变化用来实现分类的三级联动,品牌采用下拉框;在这个页签中,重点用到了富文本编辑器 kindeditor.它支持文本编辑和上传图片;但是因为我们这个项目是一个分布式项目,上传图片时候,如果像以往一样,把图片上传到 tomcat 里面,有可能其他服务器访问不到;所以我们用到一个技术 fastDfs; fastDfs 是一个轻量级分布式文件系统,它对文件进行管理,上传下载,解决了大容量储存和负载均衡问题;

2.3.2.2 fastDfs 运行机制

首先 Tracker 会定期维护上传状态信息和库的信息;client 相当于一个 jar 包,用 client 发送请求,请求发送到 Tracker 服务器,Tracker 会返回一个储存地址(ip 和端口),client 拿到信息后再往 Storage 群中存放,保存完后会生成文件的 id 还有存放地址,最后我们接到的文件存储的地址,把地址存到数据库中;
生成规则:group1(代表哪个组)/磁盘虚拟目录/文件的名后缀;

2.3.2.3 第二个页签:

商品 SKU 属性的图片上传,把不同颜色的商品图片上传到 fastDFS 中,路径生成到 SKU 属性表中,用来做商品 SKU 信息图片展示;

2.3.2.4 第三个页签:

商品的扩展信息,商家可以自定义商品的属性,然后存放到商品详情表中,作为商品扩展属性展示;

2.3.2.5 第四个页签:

商品的 SKU 属性生成页面:首先有一个启动规格的选择,有些物品不需要启动规格,当选择启动规格时候,就需要进行生成 SKU 属性:

- ✓ 根据第一个页签选择的分类读出模板从模板表中查询出该分类的规格及属性,然后在页面上循环展示规格属性的复选框;
- ✓ 在页面选中复选框属性生成一个规格集合;
- ✓ 初始化一个 SKU 属性集合(数量,价格,状态,规格),循环选中的规格集合,得到不同的规格对象,深克隆创建一个新的 SKU 属性对象,返回放到集合中,在页面生成 SKU 信息表格;

做完成四个页签:把数据提交到后台,判断是否启动规格,如果没有启动规格,后台默认生成规格属性,如果启用规格,前台操作,进行规格添加;当我们完成商品添加之后,通过商品主表里的状态字段改变确定是否可以上架;状态大致可分为新增,审核,待上架,审核未通过,已上架五个状态

3 短信微服务

3.1 短信微服务

因为我们项目是一个分布式的系统，有好多 web 工程都需要使用发送短信的功能，比如说注册、秒杀、支付还有很多场景都用到了发短信的功能，我们采用的架构是 SOA 面向服务的架构，而短信不涉及具体业务，而是一个通用的功能，所以我们项目中,是采用 SpringBoot 和 ActiveMQ 实现的一个微服务抽取出所有发送短信的功能。短信我们使用的是阿里大于提供的短信服务.

3.2 业务上

我们在注册页面填写用户注册的相关信息，其中一个重要的信息就是手机号，当用户填写完手机号后前台页面采用 ajax 异步的方式调用后台 controller 接口进行短信验证码的发送,后台短信 controller 接口中主要做了以下几件事儿：1、在后台生成短信验证码 2、将手机号为 key,生成的验证码为 value 存放到 redis 中 3、然后通过消息中间件将需要发送的短信信息发送我们搭建的 sms 短信微服务的这个服务器上进行短信的发送,这个就是我们短信的发送一个业务流程。

当短信发送成功后,在用户发送注册请求的时候首先获取到前台输入的验证码,然后根据手机号到 redis 中获取生成的验证码,两者进行比较.如果比较结果一致那么可以继续注册,如果比较不一致的话.那么取消注册.

我们用到了我们搭建的 springBoot 微服务 sms 的短信接口,还有 JMS 中间件 activeMQ 的消息队列,服务端的话会用到用户中心这一端,短息微服务的这一端,并在这俩端搭建 AMQ 消息队列进行通信,表呢注册时用到了一个用户表,

3.3 技术

阿里的大于短信微服务接口 ,对中小型的项目来说,不仅完美的支持,还融合了三大运营商的通信能力.符合我们现在项目的一个支持。springBoot 是一个轻量级的服务框架,因为我们的阿里的大于也是一个轻量级的服务,相对于 spring 来说,太多的配置文件比较繁琐并不适合我们来做短信服务这个端口,所以呢.我们就用了 springBoot；activeAMQ 这一块呢,是消息中间件,是用来管理消息队列的一个服务,activeAMQ,有俩种模式,一种一对一,点对点的模式,另一种呢是一对多的一种模式,一个人发送,有多个人消费

3.4 具体的细节

当我们的后台发送短信的方法被调用的时候。首先,我们用判断手机是否正确的正则工具类判断手机号是否正确,如果手机号不正确的话,直接返回到前台,并把错误信息告诉用户；如果手机号正确的话,我们将要发送的短信信息发送到 mq 中

3.4.1 生成验证码

首先,我们会生成一个 6 位数的验证码,在生成验证码的时候我们开始直接用随机数*6 位来生成验证码,这样生成的验证码会有缺失，后来这里我们自己写了一个生成 6 位验证码的方法。我们采用循环六次的方式，每次从 0-9 中随机产生一个数字，然后通过 stringbuffer 将 6 个验证码进行拼接后台 toString 后返回

3.4.2 Redis 存验证码和发送 mq

我们用的是 RedisTemplate 来操作 redis,使用的 hash 这种数据结构。我们用的是 hset 的这个方式存的值,给了一个 key 为固定值，feildkey 是手机号，feildvalue 是生成的验证码的值,并设置了失效时间【设置失效时间的用处非常大的，为了防止前台页面一直调用发送验证码的接口。我们在后台做了一次控制，就是每次前台调用发送验证码的接口时都先从 redis 中取一遍验证码，如果能取出来，说明该用户在有效时间内已经调用过了这个接口了。那么我们不重新生成验证码，并将结果返回给前台】

然后就是利用 jmsTemplate 来把们的验证码发送 sms 服务器，我们把要发送的信息放到 MapMessage 里,这里的参数,有待发送人的电话号码,有阿里大于的模板钥匙,和我们生成的验证码,还有一个是我们要用的模板类型.

3.4.3 收验证码

sms 短信微服务用 AMQ 来接收我们刚刚发出的消息.在 sms 里呢,我们通过 @JmsListener 这个注解来监听发生的消息，在收到消息后我们把消息用短信服务工具类调用阿里大于的短信接口进行短信的发送。这个短息工具类需要配置 accessKeyId accessKeySecret。我们放入的数据，有待发送人的电话,使用的短信签名,短信模板,还有就是模板变量的值。设置好这些后,进行发送

4 门户展示模块

门户界面这块，主要有商品分类展示，热门商品展示,广告位信息展示等功能。由于用户访问主界面会比较多，频繁的查询 mysql 数据库一个是造成服务器过多的压力，另外一个是查询效率比较低。为了减轻 MySQL 压力，我们使用到了 Redis 数据库。当然像热门商品，广告位，这些信息，是会不定时产生变化，我们需要将变化的数据从 mysql 数据库同步到 redis 数据库。当我们删除修改的时候，可以直接删除 redis 中内容，或者也给它设置一个生命周期，这个当时设置的是一天的生命周期时间。当后台更新广告位信息时，会同时操作 redis 中的数据，避免展示过时信息.

门户界面还有在商品类别查询的，比较复杂的是用到了一个递归封装 json,一个类别下边，有一个子类别，子类别下边还有可能有一个子类别，所以在设计商品类别表的时候,商品类别都有一个父 id，并且有一个 is_parent 来标识当前节点是否是父节点，当进行数据库查询封装 json 时，会判断当前是否是 parent 节点，如果是，进行递归查询，封装成一个 json，封装成 jsonp 可以传输的数据格式，完成跨域请求。

5 单点登录（Redis\Cookie\CAS）

5.1 Redis+cookie 解决方案

单点登录的主要原理就是在每次登录成功以后生成一个唯一不可重复的令牌 token，我们就简单的用了一个随机的 UUID 来生成 token。当用户登录成功后用生成的 token 做 key，登录的用户对象信息转成 json 字符串后作为 value 放到 redis 里边【使用的是 hash 这种数据结构】。然后再把这个 token 回写给浏览器的 cookie 中。这样，当别的模块再访问后台的时候只需要将 cookie 中的 token 传给服务端，服务端就能根据 token 到 Redis 中获取用户信息，如果能获取到说明登录过，如果获取不到说明没有登录过，从而实现单点登录的功能。

再来说一下 cookie 保存时的一些细节。因为我们的系统在部署时，各个模块都是通过 Nginx 映射到同一个一级域名下的，cookie 只要把他的作用域设置成一级域名，就可以在所有同一个一级域名下的模块中共享。所以我们以字符串 “sso”为 key 把随机生成的 token 为 value 写入到 cookie 里边，设置有效期 30 分钟；

然后做了一个统一校验 token 的接口；各个模块在自己的拦截器里边，调用此 token 校验接口，验证是否登录。统一校验 token 的接口的主要流程是，首先从 cookie 里边获取到 token，然后通过 token 到 redis 里边获取用户信息。这两个过程，任何一个失败，都直接返回 null，则说明没有登录，拦截到统一的登录页面，并把进入拦截到的 url 放入 cookie 里边，方便登录成功以后获取这个 url，进行原路径跳转，而不是每次登录都进入首页，提高用户的体验度。如果成功，就 cookie 和 token 的值从新设置一遍(这个是为了刷新有效期)；这样就实现了多个模块只需要登录一次就可以的流程。

还有就是注销，注销也是调用统一的注销接口，注销时需要首先从 cookie 中获取 token，根据 token 删除 redis 中的用户信息，然后在删除 cookie 中的 token。

5.2 Cas 解决方案

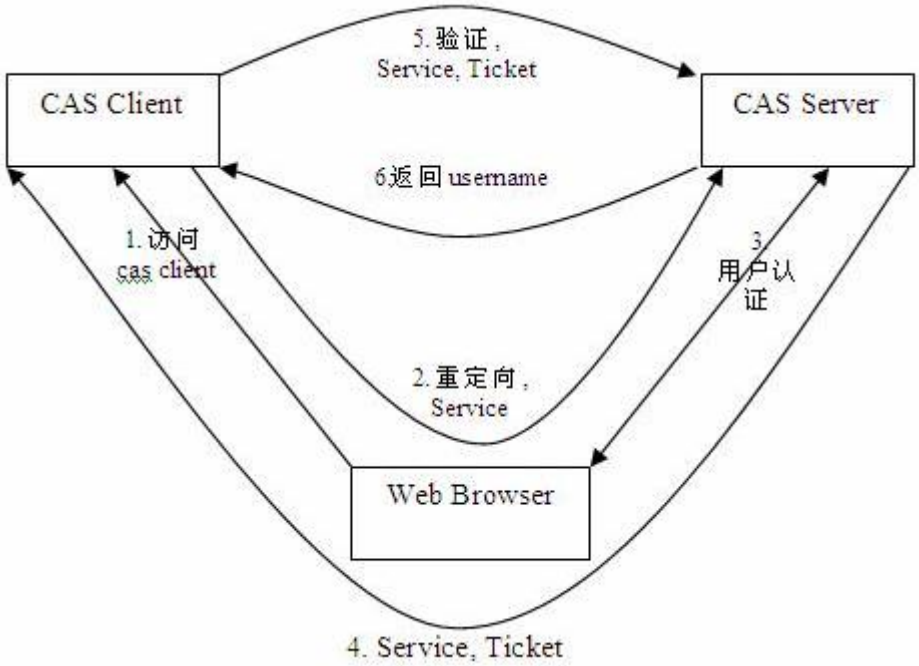
CAS 是 Yale 大学发起的一个开源项目，旨在为 Web 应用系统提供一种可靠的单点登录方法，CAS 在 2004 年 12 月正式成为 JA-SIG 的一个项目。CAS 具有以下特点：

【1】开源的企业级单点登录解决方案。

【2】CAS Server 为需要独立部署的 Web 应用。

【3】CAS Client 支持非常多的客户端(这里指单点登录系统中的各个 Web 应用)，包括 Java, .Net, PHP, Perl, Apache, uPortal, Ruby 等。

从结构上看，CAS 包含两个部分： CAS Server 和 CAS Client。CAS Server 需要独立部署，主要负责对用户的认证工作；CAS Client 负责处理对客户端受保护资源的访问请求，需要登录时，重定向到 CAS Server。下图是 CAS 最基本的协议过程：



SSO 单点登录访问流程主要有以下步骤：

1. 访问服务：SSO 客户端发送请求访问应用系统提供的服务资源。
2. 定向认证：SSO 客户端会重定向用户请求到 SSO 服务器。
3. 用户认证：用户身份认证。
4. 发放票据：SSO 服务器会产生一个随机的 Service Ticket。
5. 验证票据：SSO 服务器验证票据 Service Ticket 的合法性，验证通过后，允许客户端访问服务。
6. 传输用户信息：SSO 服务器验证票据通过后，传输用户认证结果信息给客户端。

5.3 第三方登录

第三方登录，我的理解就是基于用户在第三方平台上已有的账号和密码来快速完成己方应用的登录或者注册的功能。遵循一个 Oauth2.0 国际通用协议，允许用户在不提供用户名和密码的情况下，让第三方应用访问一些资

源。使用第三方登录时，我们不需要用户再次输入用户名和密码，而是直接通过一个唯一 openid 来进行授权登录。对于普通用户来说，如果能用 QQ、微信、百度、新浪这些平台的账号一键注册登录各个平台，无疑会方便很多。对于我们的应用来说，通过授权，借助 QQ、微信这些用户量比较大的第三方平台增强自己的知名度也非常划算。

我们的平台集成了 QQ、微信、百度、新浪四种第三方登录方式，实现的方式都是类似的。首先去各大开放平台进行注册成为开发者，并创建应用，填写回调地址，获取 appid（应用唯一的识别标志）、appkey（给应用分配的密钥），（名称可能不一样）；下载 api 文档和 sdk 开发工具包；就可以开始开发了。首先在我们网站的登录页面根据 api 集成第三方登录的 logo 图标，并给与点击事件，当用户点击此图标时，发送请求，直接跳转到第三方平台的登录页面，第三方平台也会自动检测电脑是否有已登录的账号。登录成功以后，第三方平台会自动调用我们传递的回调地址，并传递回一个 code 参数；我们拿到 code 以后，再次调用第三方 api 提供的接口，传入 code、app_id、appkey 等参数，调用获取 access_token 的接口(接口调用，有第三方提供的 sdk 包，直接导入 jar 包，根据 api 文档，传递参数调用方法就可以，我们没必要太过关心第三方平台是用 webservic 接口或 httpclient 接口。)。获取到 access_token 同时，会获取到 openid，拿到 openid 以后，就相当于拿到了登录授权。用 openid 去自己的用户表中查找是否与对应的用户，如果有，就直接查出用户信息，创建自己的 session 就可以了。如果没有，则新创建一个用户，把 openid 放进去。如果还需要其他信息，可以通过 openid 再次调用第三方平台的接口获取用户信息，如果用户信息还是不够，可以创建完用户以后再次跳转一个页面，让用户不全信息。信息补全以后，创建 session，完成登录。这样一个第三方登录就完成了。

6 搜索模块（Solr）

Solr 引擎 Apache 旗下的开源搜索平台。基于 lucence（全文搜索引擎）。**特点：**企业级，快速，高度可扩展，构建复杂，性能高。支持并行 sql 查询。几乎可以适用任何编程语言。（并行：同一时间段，多个线程同时执行；并发：同一时间点，同时执行。）也可用于存储，存储方式是以文档格式存储，存储在【solrhome\collection1\data\tlog 文件夹下】。类似于 NoSql 数据库，是一种非关系数据库存储和处理技术。

使用时，Lucene 官方仅提供了核心实现的 jar 包，我们只能通过 api 文档，自己实现创建索引和查询的逻辑，并进行优化处理，开发比较繁琐，但比较基础，更灵活，做文档的检索时可能相对来说好点。而 Solr 是 Apache 开发的一个开源 web 工程，他帮助我们实现了创建索引，查询等操作，经内部进行了一定的优化，使用 Restful 风格对外提供了接口。并且提供了 solrj 的 sdk 工具包，方便 java 开发者直接使用。

在项目中只需要官方下载到 solr 工程的 war 包，根据我们实际情况，修改 solrhome，solrconfig.xml 和 schemama.xml 文件，简单的配置一下使用的数据库和创建索引的 sql 和字段，然后把 war 部署。可以通过他提供的页面进行创建和查询索引等操作，也可以在项目中继承 solrj 的 jar 包，直接调用他的方法，进行创建索引和查询操作。

Solr 支持高亮显示搜索关键字，支持匹配权重配比（无特殊排序要求时，默认根据查询相关度来进行排序，可以在配置文件中配置，通过权重配置相关度）、支持分页、支持自动分词等。我们一般都采用 IK 分词器，代替他默认的标准分词器。因为 IK 对中文的支持比较好。如果想要支持拼音，还可以使用它的拼音分词器。

Solr 也支持集群部署，官方提供了 solrCloud 的部署方案，一般通过 solrcloud+zookeeper 实现，zookeeper 在 solr 集群中主要有两大作用：1.负载均衡；2.集中式配置管理。需要注意的是 zookeeper 一般采用选举机制，所以一般都是 2n+1（奇数）台。

另外，由于 solr 查询时，不是直接查询数据库，而是开始时，先把数据库中的数据同步到索引文件中，查询时，直接查询索引文件。索引数据库修改时，需要同时同步到 solr 的索引文件中，也就是创建索引。创建索引分为全量创建和增量创建。可以通过配置文件实现自动增量更新，自动增加更新可定时将数据库中的数据导入到 solr 索引库中，并可定时重建索引库，合并小索引文件为大的索引文件。

除 solr 外，最近又出了一个搜索引擎叫 ElasticSearch，我简单的了解了一下，他和 solr 一样，也是基于 Lucene 的，rest 风格，分布式搜索引擎。solr 的特点是：查询快，但更新索引时慢（插入删除慢），对用于电商等提前录入好数据，之后查询特别多的应用；而 ElasticSearch 简称 ES，建立索引快，查询相对慢点。所以我们采用了 solr 来做搜索引擎。

6.1 Solr 的运行

Solr 的服务器搭建有两种方式，一种是使用 solr 自带的 web 容器运行。由源码包下的 example 文件夹下的 start.jar 直接 cmd 启动，默认端口号 8983。一种是将 war 包直接放在 tomcat 下运行，端口号随意。

6.1.1 Solrhome 的组成

- bin 文件夹,一般为空
- solr.xml(不是必须的)基础配置文件,链接设置，端口号等
- collection1：默认生成的 solr 核心文件夹
  - 配置文件夹 conf
    a)最主要的是 schema.xml 可在此文件中配置域
    b)最大堆内存，及最小堆内存默认 512
    c)编码格式
    d)默认的查询格式
  - 数据的存储文件夹：data\tlog
    数据以文档的格式存储，且带索引
  - core.propertites 文件
    配置核心的名字 collection1

6.1.2 介绍管理页面

Solr 有自己的管理页面：地址+端口号/solr/#，切记这个#号
a)添加新的核心（添加到 solrhome 文件夹下，跟 collection1 同级）
b)Documents 选项，可以上传文件，搜索资源等，
c)Files：核心文件
d)Query：可以对存储在 solr 中的数据进行增删改查的操作
e)选择核心的下拉框，此处可以选择要操作的核心。

6.1.3 Schma.xml 的配置介绍

schema.xml 配置文件是 solr 的最主要的配置文件，位于 solrhome 的 collection1\conf\schema.xml

- 在 schma.xml 中配置 FileType(重要是配置分词器)
  官方提供的中文分词器：smartcn；通用的中文分词器：Mm 分词器、IK 分词器。IK 分词器。对中文更为友好。
  - 将分词器的 jar 包存放在 solr 工程的 web-inf/lib 目录下
  - 在 solr 工程下创建 web-inf/classes 文件夹，存入 IK 分词器的扩展词典、停用词词典，配置文件

- schema.xml 中配置 FieldType，调用 IK 分词器【fileType：用来配置引用文件， name，class，以及源码包】

◆ 在 schma.xml 中配置 File 域

File 域分为普通域、复制域、动态域等多种形式，File 域中可以配置如下的属性

- Name:域名（实体类中字段上的注解名）
- Type：类型（字段类型）,需要中文分词的话，配置成我们上面配置的 IK 分词器
- Index：是否索引
- Stored：是否存储

6.2 项目中的应用

我们的网站的首页。用户进来一般有两种情况，无目的的随意看看和有目的的商品搜索，而第二个就是我们搜索引擎的存在的意义。在首页中，一般会有一个搜索框，当用户在搜索框内输入关键字并点击搜索，则进入商品搜索页面。可以说首页的这个搜索框就是我们整个搜索模块的入口。

在搜索页面中，我们会根据用户的关键字进行查询，整个页面中展示了包括关键字所涉及的所有分类，以及默认分类下的所有商品品牌、规格、规格属性，还有所有涉及的商品的列表。用户根据需求选择相应的属性，得到一个细化过后的 spu 属性，点击这个 spu 属性，则进入 freemarker 的静态页获取具体的 sku 商品。

6.2.1 综合需求

关键字高亮查询、区间查询、分组查询、排序查询、分页查询：（详细代码自己去项目中找，这里只介绍方法）

（1） 当用户输入关键字搜索后，除了显示列表结果外，还应该显示通过这个关键字搜索到的记录都有哪些商品分类。

（2）根据第一个商品分类查询对应的模板，根据模板查询出品牌列表

（3）根据第一个商品分类查询对应的模板，根据模板查询出规格列表

（4）当用户点击搜索面板的商品分类时，显示按照这个关键字查询结果的基础上，筛选此分类的结果。

（5）当用户点击搜索面板的品牌时，显示在以上结果的基础上，筛选此品牌的结果

（6）当用户点击搜索面板的规格时，显示在以上结果的基础上，筛选此规格的结果

（7）当用户点击价格区间时，显示在以上结果的基础上，按价格进行筛选的结果

6.2.2 索引库的初始化

由于 solr 是从 solr 索引库查询的，而我们工程中的数据存在 mysql 数据库中的，因此项目安装完 solr 服务器时候我们做了一次 solr 库的初始化工作。为此我们特意写了一个 solr 工具类，查询所有的 sku 信息，然后使用 springDataSolr 将 sku 表中的所有数据查出来同步到 solr 索引库中。

第一步：Schma.xml 中配置需要进行索引的域，并配置了 IK 分词器

第二步：pojo 类和 solr 索引库建立联系，主要是增加@ID 和@Flied 注解

第三步：从数据库 sku 表中查询所有的 sku 信息使用 springdatesolr 同步到 solr 索引库中

6.2.1 添加商品更新索引库

当商家后台新增了新的商品信息时，运营商后台审核通过以后会更新索引库。开始我们采用的同步更新的方式，

运营商审核通过后就直接同步到 solr 索引库中，但是使用的过程中发现这种方式用户体验很不好，因为同步到 solr 索引库会很慢，html 页面迟迟没有响应。后来我们改成了多线程异步的方式同步的索引库，但是需要我们手动维护多线程。最后我们引入了 activemq 消息中间件完成了同步 solr 索引库的操作

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现。说白了他就是一个消息中间件。主要用于系统之间的解耦。系统与系统之间的通信不用直接的调用，可以用 MQ 进行通信。它对消息的传递类型有两种形式，一种是 queue，点对点的的形式，即一个生产者对应一个消费者。一种是发布与订阅的模式，即一个生产者产生消息并进行发送后，可以由多个消费者进行接收。他的特点是再服务器端端不能持久化。

我在项目中的应用是将 ActiveMQ 与 Spring 进行了整合，来进行添加商品信息时与搜索库进行同步。

具体使用方法是

- 第一步：引入相关的 jar 包比如说 jms 的，
- 第二步：在 xml 文件中配置 ActiveMQ 整合 Spring

在生产者的 xml 文件中配置 JMS 服务厂商提供的 它相当于一个实现类；spring 对象 ConnectionFactory 的封装它相当于是一个接口；

配置 JMSTemplate，使用 JMStemplate 对象来发送消息； 配置消息的 Destination 对象 在我的项目里由于我是做添加商品时进行缓存同步，所以我吧目的地配置成了 topic 形式，来保证能被多个消费者使用。

- 第三步： 在添加完商品的业务中在成功添加完商品后，使用注入到 Spring 中的 jmsTemplate 来发送一个商品添加信息。这时候消息队列里就有了我们发送的信息。
- 第四步我们开始配置消费者接受信息：

  首先在配置文件中配置监听，写一个负责监听的类来接收生产者发送过来的信息

  我在项目中取得是商品的 id 根据 id 去查询数据库，查询出来数据后，创建文档对象中添加域

  将文档对象写入索引库提交。这样我们就完成了使用消息队列完成了缓存同步。

7 购物车

在未登录的情况下，因为没有用户信息，所以这时候如果想要存储购物车信息，只能保存在浏览器客户端。有两种选择 cookie 和 localstorage，无论是 cookie 还是 localstorage 都是以 key-value 的方式存储，我们用“cartList”这个固定字符串作为 key，把购物车列表转成 json 作为 value 将购物车信息保存到 cookie 中。

登录以后，购物车列表信息保存在 redis 里。我们使用了 Hasm 类型的数据格式缓存的购物车信息。存储时 key 为 “cartList”这个固定的字符串,fieldKey “固定前缀_用户 id”。fieldValue 值为购物车列表数据；当给购物车存放一个商品或者取出一个商品时，可以直接用“固定前缀_用户 id”获取购物车列表信息，返回一个 List，然后迭代 List，获取到购物车中所有的商品信息，在进行增删购物车中购物项操作。

另外，由于用户未登录时，购物车信息只能保存在客户端，不能和用户建立联系，所以如果有其他人借用电脑，购物车信息就可能混乱，这个是不可避免的，如果要考虑这点，最好的方式，就是用户不登录时，不让他操作购物车。还有就是，如果未登录时，保存了购物车信息，在登录成功后，一定要把购物车信息合并给登录用户

补充一下 cookie 和 localstrorage 的只是。cookie，是每个浏览器都有的，既可以通过 js 操作，也可以随着请求传递到服务器端，在 Java 里边通过 request 和 reponse 对其进行操作，可以设置 cookie 的有效期、作用域、作用路径等。但是缺点是有可能被禁用，不能跨域，存储量小。localstorage 是 html5 的本地存储，存储量可以达到 5M，本身也不支持跨域，但可以借助其他方式解决。但是他只能通过 js 操作，不能随着请求传递到后台用 Java

操作

8 秒杀

首先商家填写完商品，运营商审核，审核通过后，保存到搜索服务和页面静态化服务，关于秒杀的静态页面，我们根据数据库中商品的活动开始时间，进行倒计时。然后做了各种校验，比如秒杀的时候根据商品 id 到 redis 中查询，商品的活动开始时间是否大于或等于服务器的当前时间，如果大于等于提示活动尚未开始。还有做商品的状态校验，如果商品状态是已下架或者无效，提示非法请求。

超卖限制，商品的信息保存到 redis 中去，redis 的 key 是商品 id，value 是库存数量，利用 redis 事务的 watch 机制，来控制库存。

成功买到商品的用户同 mq 异步更新到数据库下订单，扣减库存，发送短信。我们秒杀的流量是一秒钟 8w 左右，我们有 40 台服务器，做了 nginx 和 tomcat 的集群，dubbo 服务集群，redis 集群,zookeeper 集群如果服务宕机，我们 linux 上有定时任务，每分钟检测服务，自动重启

9 订单和支付

我们支持多种支付方式，包括支付宝、块钱、易宝、京东、微信、银联等。我负责的是京东支付和微信支付，我就调一个微信支付简单说一下，首先调用微信的统一下单 api ，传入订单号，金额，币种，回调的地址，签名。接口调用成功后，给我们返回一个 codeurl，利用 google 的 zxing 包，生成二维码。

支付成功后，微信进行回调，我们根据微信回调的参数进行，校验，比如签名是否正确。订单金额是否一致。支付状态是否为未支付。

9.1 订单管理

订单表的设计，主要包括订单表和订单详情表，订单表主要包含订单的主要信息，比如订单的编号、总额、数量、状态、收货人信息等。其中收货人信息要冗余到订单表中。

订单详情表和订单表是多对一关系，订单详情表主要计量订单中的商品的详细信息，这些信息也要冗余进来，不能通过 id 进行简单的关联，因为订单一旦生成，这些信息一般不会再允许改变。

订单在用户结算购物车时生成，生成订单时，应该对库存进行一次校验，防止超卖；如果同时购买多个商家的商品，在结算购物车时需要进行分单，同时生成多张订单。在用户中心，每个用户都可以看到并跟踪自己的订单，进行支付、申请退货、确认收货、评价等操作。

支付日志表 主要目的是为了记录订单的支付相关信息，并和支付宝等支付平台进行对账操作

商家后台可以看到商家自己的所有订单，进行确认发货操作。而在运营管理平台，可以监控所有的订单，但是不能进行操作。

订单的状态主要包括：待支付、待发货、已发货、已完成、已取消；

9.2 订单重复提交

9.2.1 产生的原因

1、在网络延迟的情况下让用户有时间点击多次提交按钮导致表单重复提交。
2、表单提交后用户点击浏览器的刷新导致表单重复提交
3、用户提交表单后，点击浏览器的【后退】按钮回退到表单页面后进行再次提交

9.2.2 客户端处理。

- 设置一个标识，让他只能提交一次
- 表单提交之后，将提交按钮设置为不可用，让用户没有机会点击第二次提交按钮

9.2.3 Token(令牌)验证

- 在订单确认页面的初始化方法中，我们需要向后台申请交易 ID，并使用 UUID 生成一个 token，生成后存入 redis 中，并设置失效时间
- 在创建订单的方法中，需要前台带着 token 参数来，后台处理时会检查 token 在 redis 中存在不存在，如果存在进行创建订单的处理并且删除 token，如果不存在说明有可能是重复提交也有可能是时间太长失效了

9.3 订单失效的处理

- 第一个版本，使用定时器扫库
  我们使用 Spring Task 定时器 ，每隔 30 分钟就扫一遍数据库，看那些订单已经过了支付超时时间还没有进行支付，那么就调用支付宝的取消支付接口取消支付，并把这个订单的状态修改为超时未支付。
- 第二个版本，使用 rabbitMQ 的延迟队列
  我们需要一直访问数据库，查询哪些订单到 45 分钟了，还有时间间隔方面，都不太好把控（比较浪费系统资源）。 我们选用的是另一种方按，rabbitMQ 延时队列的方式，使用 rabbitMQ 的 延时队列 ， rabbitMQ 在创建队列的时候 可以给这个队列设置死信交换器 和 死信路由 ，如果一个信息在指定时间内 没有人消费，会变成死信重新将消息投递至死信交换器 及 死信路由 。我们做延时队列就是利用这个机制，我们创建了两个队列一个队列用于投放订单消息并设置一个失效时间（45 分钟）这个队列我们不会消费， 45 分之后该队列中的消息 会变成死信， 会重新路由到我们的另一个订单失效处理的队列中， 在那个 队列里 我们会判断 该订单是否为待支付状态，如果是待支付状态会把订单状态改为失效。

9.4 支付宝支付

要使用支付宝支付，必须要有支付宝的企业号，在企业号的后台可以创建一个应用(企业号的申请需要公司的相关材料,如组织号，营业执照等);有了应用后可以查看应用的 APPID

我们还需要使用支付宝提供的 RSA 签名工具生成一对密钥（商户私钥和商户公钥）商户私钥是要用在项目中，支付宝支付的配置上面，商户公钥上传到支付宝后台的应用上后，会得到支付宝公钥，这个公钥也要用到项目中
1、有一个应用 ID，我的 APPID，收款账号即是我的 APPID 对应支付宝账号

2、商户私钥

3、商户公钥生成一个支护宝公钥

4、Notify_url：修改订单状态，是支付成功还是没支付成功

5、Return_url:用户跟支付宝交互，用户把钱付出去了，在支付宝哪里支付完之后，这时支付宝需要一个页面跳转的

9.5 商户没有收到支付结果的通知问题

用户在我们的支付成功，支付成功一定收到支付宝的通知。在这个通知之前是会回调我们提供的接口。支付宝回调的过程中，没有调通我们的支付接口。 【支付平台不健壮，就是服务不稳定导致的】 用户马上投诉，解决办法，客服后台，根据用户提供的订单，查询支付的信息（这个查询我们是直接调用 支付包接口提供的查询功能进行查询的，如果存在的话就是我们网站的问题，这我们的代码会修改订单的状态）当然我们还有一个对账系统，这个对账体统只要做的是，将我们支付平台的订单到第三方支付进行查询，看是否有支付记录。

9.6 安全问题

签名机制 参数信息（秘钥）一-> 根据加密的算法进行加密 ~>生成加密的签名 ——>进行 传递 ---> 我会根据秘钥+传过来的参数，在进行一次加密也会生成一个签名，传过来的签名和我自己生成的签名进行对比，如果一直说明数据没有修改，如果不一致，说明数据修改了。

9.7 异步订单状态修改(分布式事务)

使用开源框架 TX-LCN 分布式事务框架 5.0 版本对代码的侵入比较少，而且提供了 完整的 SpringCloud 和 dubbo 注解解决方案，项目组引入该框架的成本非常低，并且有专门的公司对这个开源项目进行维护。

该框架主要有两大核心部分 TM 事务的协调管理器需要独立运行 (在源码中有该工程，并且整合了 docker 只需要改下配置文件中 mysql、redis 的地址即可运行)

- 所有参与分布式事务的 微服务项目 需要引入客户端依赖

```
<dependency>
    <groupId>com.codingapi.txlcn</groupId>
    <artifactId>txlcn-txmsg-netty</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

- 在启动类当中，使用@EnableDistributedTransaction 注解开启分布式事务
- 在配置文件中配置好 Tx-manager 的连接地址
- 在需要进行 事务管理的 service 方法中

如果要使用分布式事务加上 @LcnTransation；如果要使用 TCC 的模式需要加上@TCCTransation 注解并且在该注解指定 confirm 对应的方法以及 cancel 对应的方法

9.1 第三方支付

B2C 电商的支付，一般由于支付金额比较小，支付比较频繁，所以一般采用第三方支付，常用的第三方支付有：支付宝、微信、聚合支付、付钱啦等。他们的原理都差不多。都是在点击支付时，直接调用第三方支付接口，传入 appid、appsecret、订单编号、订单金额、回调 url，直接跳转到第三方支付页面，接下来的支付过程，我们都

不需要管，支付成功以后，第三方支付平台会直接回调我们的 url。给我们返回：状态码、订单编号、支付流水号三个参数。我们首先根据订单编号，找到我们的订单，把支付流水号和状态码更新到我们的订单里边。回调 url，一般有两种，一种用同步 get 方法回调，一种用异步的类似 ajax 方法回调，同步方法回调，一般是成功以后才会回调，并且只回调一次，回调成功以后我们可以直接跳转到我们的支付成功页面、异步方法回调，一般要求我们返回一个 success 字符串，第三方平台如果没有接受到 success，就会认为没有调用成功，他会重复多次调用。比如支付宝会在 25 小时之内，调用 8 次；一般情况下第三方支付都采用第二种方式，因为比较安全，但支付宝是同时采用了两种。

我之前接触过一个 B2B 的电商，他们由于交易金额比较大，第三方支付无法实现，所以是直接和银行对接。大体上是，首先平台和银行签订合同，银行为平台开设一个总账号，当企业在平台注册以后，平台会为企业调用银行接口，创建一个子账号，这个子张号是挂在总账号下边的，也是一个在银行实际存在的账号，但是，只能通过外部银行卡给里边转账，而不能给外部银行卡转出。可以在子行号直接互相转账。

10 ActiveMQ

以前我们做电商项目的时候由于用户的并发量不是太高，而且用 dubbo 对服务进行监控和管理，没有发现什么问题。后来并发量上来了以后就有反馈回来，说用户在下单后的响应提示有点慢，开始的时候对系统检查没发现什么问题。后来考虑到可能是由于并发量变高导致的模块间的响应变慢了。

所以在最近的一个电商项目中就用到了 ActiveMQ，比如购物车模块，购物车有个特点就是，用户在购物车选择相应的商品进行提交订单后，就需要将生成订单的商品从购物车删除，由于我们的系统是分布式的，用户提交订单成功后，我们的购物车系统是不知道的。所以这时候订单系统就要通知我们购物车系统，其实有多种方式可以实现：一种是我们这边开发 dubbo 一个服务接口，订单提交成功后，订单直接调用我们的服务接口，完成下单后的购物车的删除功能。但是最后发现和订单交互的系统越来越多，比如：下单后还要通知，优惠券，优惠码，积分，库存，完成相应的扣除功能。这样给订单系统去通知的时候就比较麻烦了，增加了系统的压力，而且也需要维护很多地址，最后我们就修改成 mq 提高了系统的可维护性和可用性，从而提高下单的效率。在扫单系统中过期的订单需要系统取消，取消的时候需要调用优惠券，积分，库存系统进行还资源的过程，可以使用发布订阅。

11 权限管理

11.1 RBAC 模型

前期我们使用的 RBAC 模型进行的权限管理

我负责系统管理模块， 系统管理主要是对用户角色和权限的管理，不同角色的人登录应该看到不同的权限和内容，权限通常有 3,5,7 张表甚至更多来完成，我们当时用了五张，包括员工信息表、角色信息表、权限信息表和两张中间关系表：一张员工角色关系表，一张角色权限关系表。可以对用户赋角色，然后角色赋权限，权限表里存着不同的权限的 url，当用户登录时，从 session 中获取用户 id，通过用户 id 获取用户的所有角色 id，再通过角色 id 获取所有权限 url。把当前用户的权限信息用 ztree 以树形结构展示，就实现了不同的人登录展示不同的权限树。（当然，在这里可能有一些安全问题，就是用户直接在浏览器输入非法权限的 url 执行非法操作的问题，我们采用 filter 拦截器在用户执行操作时先判断该用户有无该权限的 url，在有权限的情况下才放行，否则提示非法操作，并且终止该操作）。

11.2 springscurity 安全框架

springscurity 一个能够为基于 Spring 的企业应用系统提供声明式的安全訪问控制解决方式的安全框架。应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。 spring security 的主要核心功能为 认证和授权，所有的架构也是基于这两个核心功能去实现的。

11.2.1 框架原理

众所周知 想要对对 Web 资源进行保护，最好的办法莫过于 Filter，要想对方法调用进行保护，最好的办法莫过于 AOP。所以 springSecurity 在我们进行用户认证以及授予权限的时候，通过各种各样的拦截器来控制权限的访问，从而实现安全。

11.2.2 框架的核心组件

SecurityContextHolder：提供对 SecurityContext 的访问
SecurityContext,：持有 Authentication 对象和其他可能需要的信息
AuthenticationManager 其中可以包含多个 AuthenticationProvider
ProviderManager 对象为 AuthenticationManager 接口的实现类
AuthenticationProvider 主要用来进行认证操作的类 调用其中的 authenticate()方法去进行认证操作
Authentication：Spring Security 方式的认证主体
GrantedAuthority：对认证主题的应用层面的授权，含当前用户的权限信息，通常使用角色表示
UserDetails：构建 Authentication 对象必须的信息，可以自定义，可能需要访问 DB 得到
UserDetailsService：通过 username 构建 UserDetails 对象，通过 loadUserByUsername 根据 userName 获取 UserDetail 对象 （可以在这里基于自身业务进行自定义的实现 如通过数据库，xml,缓存获取等）

11.2.3 Springscurity 的实现步骤

参考代码实现，主要进行如下几个部分阐述
Web.xml
Springscurity 的配置文件
userDetailServiceImpl 的实现

12 实现页面静态化业务场景

12.1 版本 1

我们在做某项目时，涉及到程序访问的性能问题，这时候我们想到可以通过静态化来提高用户访问时候的性能，所以我们就采用了 freemarker 模板引擎，考虑到页面也是要有动态的变化的，所以我们采用 spring 定时器在每天晚上 2 点钟的时候定时再次生成 html 静态页面，考虑发布时候的性能问题，我们又采取线程池技术，让多个线程同时发布，从而缩减发布时间。

12.2 版本 2

首页静态化的功能。由于任何用户访问时，首先会访问到我们的首页，所以很多东西都希望能在首页展示，但是放的东西多了，就会加载很慢。给用户的体验度很不好。所以我们在项目启动时，直接把首页需要的数据查询出来，通过 freemarker 模板生成静态的 html 页面，之后用户访问时，都去访问这个静态页面，这样就不需要频繁访问数据库，减轻了数据库压力，提高了用户体验度。但是缺点是，数据库数据数据变换了以后，数据无法实时更新。我们一般通过定时器的方式，每天凌晨重新生成。

还有就是热销商品的商品详情页面也做了静态化处理，首页我们是通过定时器每天凌晨一点，去重新生成；商品详情我们是在商品信息修改以后，给定时器表（id,业务 id、type，remark）中推送一条信息，到第二天凌晨一点时，定时任务扫描表，发现有需要重新生成的页面，就调用 freemarker 模板，重新生成 html 页面，以商品 id 作为 html 页面名称，然后删除任务表中的数据。为了预防大量静态页面给服务器造成压力，我们把 html 页面直接生成到 Nginx 的静态页面服务器上。访问时，不用经过 Tomcat，直接通过 Nginx 访问。

12.3 版本 3

生成商品详情页时我们也是通过 mq 的方式异步生成的静态页面，逻辑和 solr 的索引库同步一样

13 电商项目其他功能

13.1 电商如何处理高并发

为了解决高并发的问题，首先我们想到的是通过分流的方式和加快系统吞吐量和业务的优化的形式解决高并发的

在分流方面我们使用的是 Nginx 做负载均衡和反向代理

在系统的吞吐量上我们使用的 redis 进行缓存处理

在文件的存储上可以使用 fastdfs 可存储。

页面则可以采用 freemarker 静态化技术对页面静态化。

有一些必须和关系型数据库打交道的业务可以采用分库分表的形式，来提高数据库的效率。方法有很多的，这就要根据系统的压力和场景来选择自己网站适合的方式就可以了。

13.2 客户区域热点图统计

网站访问统计采用 tomcat 日志记录+百度地图，我们在 tomcat 中配置了日志记录功能，可以记录访问者的相关信息，我们配置的是每个小时生成一个日志文件。定时器每天晚上 12 点利用多线程分析访问日志，并且将访问信息保存到数据库中的访问日志表中，然后调用百度地图，百度地图可以根据 IP 地址转换成经纬度在地图上展示出来。

13.3 库存预警功能

13.4 销售报表统计功能

14 Angular.js

AngularJS 是一个 JavaScript 框架。它是一个以 JavaScript 编写的库。和他类似的还有 vue.js, AngularJS 它更适用于开发 CRUD 应用，即数据操作比较多的应用。为了实现这些，AngularJs 引入了一些非常棒的特性，包括模板机制、数据绑定、指令、依赖注入、路由等。数据绑定可能是 AngularJS 最酷最实用的特性。通过数据不模板的绑定，能够让我们摆脱繁琐的 DOM 操作，而将注意力集中在业务逻辑上。AngularJS 内置了很多指令用来控制模板，如 ng-repeat，ng-class 等，也有很多指令来帮我们完成业务逻辑。还有 AngularJS 服务啊路由之类的功能也都是非常实用的。AngularJS 丌依赖（也丌妨碍）任何其他的框架，我们甚至可以基于其它的框架来开发 AngularJS 应用。在我们之前的一个项目中也用到了 AngularJS，通过 AngularJS，我们实现了前后端分离开发，前端使用路由，页面的性能会有很大的提升，同时也会减少后端的压力，页面跳转可以需要经过后端，后端只负责提供数据做为展示。

15 实际开发过程中遇到的其他问题

15.1 项目中用的到哪些管理工具

项目版本管理工具：git(gitLab)或者 svn
项目进度管理工具：禅道或者腾讯的 tapd
项目依赖管理和构建工具：Maven
项目发布工具：jekins
项目代码审核工具：gerrit

15.2 项目有没有做过压测

有的。测试人员做压力测试的工作，主要测试的几个指标：接口响应时间要求是 500 毫秒以下，系统在极限（cpu, 磁盘，内存）情况下每秒处理的查询次数是 100qps。测试那边使用 jmeter 来进行测试。

15.3 项目迭代、敏捷式开发

互联网项目都是快速上线之后在继续完善，这个时候会由产品经理不断的提出新的需求，在原来的基础上做迭代开发。敏捷开发就是实现快速开发，分布式系统的好处也在于此，根据不同的需求，只需要对不同的模块进行快速开发测试上线即可。

15.4 前后端数据交互是通过什么技术交互的,如何联调？

因为我们是前后端分离的项目，所以前端都是通过 ajax 异步请求实现的。我们使用的是 json 数据进行交互。一般来讲我们前后端的开发是并行的，当需求确认之后，会由后端根据需求完成相关的接口文档的编写，然后前后端同时根据接口文档进行开发。
我会把自己分支的发布到测试环境，前端访问测试接口完成调试。

15.5 有没有做过 APP 接口开发？App 接口有做安全处理吗？

有的，其实 App 接口和 web 接口并没有太大的区别，只是 app 接口在程序异常的情况下也需要返回对应的状态码，我们统一使用一个 response 对象，这个对象有三个属性：状态码(status)：标识处理结果状态，消息描述(msg)：异常的情况下，描述异常问题，数据（data）:处理成功之后返回值数据。
有做安全处理，我们 app 端和后端都会对数据进行加解密，后端的加解密操作是封装在架构的底层的，是通过拦截器实现的，具体细节不是特别清楚了。还有就是需要使用用户的 token 进行鉴权。

15.6 开发中需求中有不明确的问题该怎么处理？

首先自己会确认好是否需求有问题，如果需求确实有问题会和技术经理沟通。然后在和项目经理进行需求确认和更改。

15.7 项目中有没有用到分库分表？

有的，我们的库是根据业务拆分的，不同的业务会有自己的库。需要数据交互的话都是通过调用 dubbo 服务来实现，同时我们通过配置 mysql 实现了读写分离。
分表的话，一般来讲有垂直拆分和水平拆分两种，这个可以通过一个 mycat 的框架来实现。但是我们因为数据量不大。所以还没有做。但是我了解到的像公司的积分模块那边的实现的话是通过每个月一张表的方式来进行分表的，我觉得也是水平分表的一种方式。查询积分的消费和获取的时候都是通过月份来查询，只支持半年的查询。

15.8 如何理解高可用和高并发？

高可用是指系统能过做到大部分时间都正常运行，一般我们要求 99.9%的时间都可用。
高并发是指系统可以承受同一时间大量的用户请求。
一个项目刚开始的时候是为了实现基本功能，随着版本和功能的迭代，大数据和高并发成了软件设计必须考虑的问题!本质很简单，一个是慢，一个是等。两者是相互关联的，因为慢，所以要等，因为等，所以慢，解决了慢，也就解决了等，解决了等，也就解决了慢。

关键是如何解决慢和等，核心一个是短，一个是少，一个是分流,最后一个是集群/横向扩张/读写分离/建立主从。
短是指路径要短:
典型的 mvc 结构是请求->controller->model->dao->view，然后把页面返回给用户。要想短的话，
1，页面静态化- 用户可以直接获取页面，不用走那么多流程，比较适用于页面不频繁更新。
2，使用缓存- 第一次获取数据从数据库准提取，然后保存在缓存中，以后就可以直接从缓存提取数据。不过需要有机制维持缓存和数据库的一致性。
3，使用储存过程-那些处理一次请求需要多次访问数据库的操作，可以把操作整合到储存过程，这样只要一次数据库访问就可以了。
4，批量读取 - 高并发情况下，可以把多个请求的查询合并到一次进行，以减少数据库的访问次数
5，延迟修改 - 高并发情况下，可以把多次修改请求，先保存在缓存中，然后定时将缓存中的数据保存到数据库中，风险是可能会断电丢失缓存中的数据，
6, 使用索引 - 索引可以看作是特殊的缓存，尽量使用索引就要求 where 字句中精确的给出索引列的值。
少是指查询的数据要少:
1，分表 - 把本来同一张表的内容，可以按照地区，类别等分成多张表，很简单的一个思路，但是要尽量避免分出来的多表关联查询。
2，分离活跃数据 - 例如登录用户业务，注册用户很多，但是活跃的登录用户很少，可以把活跃用户专门保存一张表，查询是先查询活跃表，没有的话再查总表，这也类似与缓存啦。
3，分块 - 数据库层面的优化，对程序是透明的，查询大数据只用找到相应块就行。
分流三种:
1，集群 - 将并发请求分配到不同的服务器上，可以是业务服务器，也可以是数据库服务器。
2，分布式 - 分布式是把单次请求的多项业务逻辑分配到多个服务器上，这样可以同步处理很多逻辑，一般使用与特别复杂的业务请求。
3，CDN - 在域名解析层面的分流，例如将华南地区的用户请求分配到华南的服务器，华中地区的用户请求分配到华中的服务器。
横向扩展
大型网站为了软解大量的并发访问，除了在网站实现分布式负载均衡，远远不够。到了数据业务层、数据访问层，如果还是传统的数据结构，
或者只是单单靠一台服务器扛，如此多的数据库连接操作，数据库必然会崩溃，数据丢失的话，后果更是不堪设想。这时候，我们会考虑如
何减少数据库的联接，一方面采用优秀的代码框架，进行代码的优化，采用优秀的数据缓存技术如：memcached,如果资金丰厚的话，必然会
想到假设服务器群，来分担主数据库的压力。Ok 切入今天主题，利用 MySQL 主从配置，实现读写分离，减轻数据库压力。

15.9 Spu 和 Sku 分别是什么意思，订单编号是怎么生成的?

sKU 是库存量单位,区分单品,.另外还有 SPU,是标准化的产品单元,区分品种,例如 iphon8 这是一个 SPU， 苹果 8 (64G 黑色 移动) 这就是一个 SKU。
我们订单号是用时间戳+推特的那个分布式 id 生成器实现的

15.10 能自己搭建一个项目框架吗？

能，springBoot 项目搭建还是比较简单的，SSM 项目搭建比较麻烦点，但是自己也搭建过，查找资料搭建起来问题不大。无非是相关 jar 包的引入和配置信息的配置，springmvc 的配置信息，spring 的配置信息以及 Mybatis 的配置信息。

15.11 公司有几个环境，项目有多少个工程，发布是怎么发布的？

我们上家公司有 3 个环境，分别是开发环境（dev），测试环境（test）,线上环境（pro）。项目发布采用的分布式 soa 开发，大概有 20 多个工程，发布项目使用的是自动集成化工具 jekins，直接完成项目从 gitlab 到服务器 tomcat 的发布。

15.12 项目有多少个接口？

大概有几百个接口，具体多少就没数过了。

15.13 需求文档设计,接口文档设计,系统设计文档都写过哪些,怎么

写的?
接口文档也叫接口 api，三要素：url,参数，返回值。
需求文档写过吗？当然写过啊。客户在做需求调研的时候 ，我们也写过的。需求文档大概是这样写的。比如客户的那些需求呢理出来，理出来以后呢。就根据一个功能点来说吧，就拿商品添加这个功能点来说吧，首先有个流程图要画出来，就用那个 viso，有开始结束标签，中间有判断，首先开始，点击进入到登陆到我们的系统，然后点击新增商品的这个链接，然后有那么一个操作符，下一步点击这个商品分类，这是一个操作符，接下来把这个商品分类显示出来，等等一系列的这个操作，我们要在需求文档里面写出来。流程图画完了，我们再说这个过程，过程呢是分一二三四五六七八这样的步骤写下来的，写完这些步骤以后呢，还有产品得截图，什么样得截图呢？就是输出画面一，输出画面二这样得截图。
系统设计的写法是：首先要归纳商品中有哪些功能点，也有系统设计的流程图，这个流程图呢就不是需求文档那个页面逻辑的流程图了。是代码的流程图。里面有规定到我们具体使用的哪个方法，类都要写清楚了。里面包括了有哪些字段。比如商品中设计到十几个字段，那么这十几个字段都得写。以及涉及到的数据库表，数据字段都要写清楚了。但这是我们原来传统的项目这样写。现在要做的是敏捷是开发，客户那边直接派人来我们公司驻扎，一些需求啥的我们和客户现场去定，定好之后就马上开发，叠加几个功能点测试，测试没问题的话就很快发布了。
现在我们用这个现代化的管理工具 swagger 管理接口文档，比如 springmvc 里面有一些 swagger 第三方 api 文档，他可以生成记录文档的信息，就是前台能够显示出来的页面的展示，类似接口文档的 url,参数，返回值类型，这个参数干什么的那个干什么的。

15.14 有没有自己设计的模块？数据库？

当然有，简单的一些功能模块就自己完成设计了，但是如果涉及的其他模块交互比较多

的时候，会和领导同事沟通以下，确保数据库建模没有问题。

15.15 你这个项目的用户量大概有多大?

要么回答还没有上线，要么回答：以电商项目为例,我们这个项目上线以后,用户量大概也就十来万，每天访问量也就一万左右 PV,并发量也就 1000 多。

15.16 平时自己会上什么网站学习？

非常多，主要有 github,掘金社区，力扣网也会看看。

15.17 说一下你们公司怎么用的 maven?

Maven 是一个项目管理构建工具，主要负责对相关 jar 包进行管理，也可以通过 maven 命令完成项目的构建（编译，测试，打包）。我们公司 maven 有自己的私服，当完成一些 jar 项目的开发之后，需要通过 deploy 命令发布到私服中，方便其他同事引用。
Maven 常用的命令有 clean、compare、test、package、install、deploy 等等。

15.18 说一下 maven 中的 jar 包冲突如何解决？

一般我们公司的 jar 包版本都会统一，尽量避免同一项目中使用不同版本的 jar 包，从而导致冲突问题。如果因为依赖某个 jar 包，而 jar 包内的依赖与外部依赖产生冲突可以使用 exclude 标签排除掉指定 jar 包。

15.19 说一下 git 用的是什么作为远程仓库的，以及常用命令和解决

冲突的办法？
我们是使用自己搭建的 gitlab 作为 git 仓库的。项目拉取和拉取都是对 gitlab 操作。
Git 常用命令：拉取项目 git clone， 添加到本地库 git add 提交到本地库 git commit
创建分支：git branch <分支名>
拉取分支：git checkout -b <分支名>
提交代码：git push 更新代码：git pull
分支合并：分支的合并-merge 首先使用 git checkout master 命令切换到 master 分支上。然后使用 git merge bugfix01 命令将 bugfix01 分支的修改合入到 master 分支上，在合入成功后会将合入后的新文件进行提交，此刻会有一个新的 commit 号
如何解决 git 冲突：
当合并代码的时候会遇到代码冲突的问题，此时 git 会提示我们冲突文件以及冲突代码在哪里，根据提示手动对冲突的代码进行处理，然后再合并。

15.20 maven 业务场景

maven 业务场景 maven 是一个项目管理工具，其核心特点就是通过 maven 可以进行包的依赖管理，保证 jar 包版本的一致性，以及可以使多个项目共享 jar 包，从而能够在开发大型 j2ee 应用的时候，减小项目的大小，，maven 根据“约定优于配置”的特性，可以对其项目的编译打包部署进行了 更为抽象的封装，使得自己不需要像 ant 那样进行详细配置文件的编写，直接使用 系统预定好的 mvn clean,compile,test,package 等命令进行项目的操作。于是我就 在 XX 项目中采用了 maven,为了保证团队中的成员能够节省下载 jar 包所需要的时间， 于是我就采用 nexus 搭建了在局域网内的 maven 私服，然后通过配置 settings.xml 中 建立 mirror 镜像，将所有下载 jar 包的请求都转发到 maven 私服上，之后通过在 pom.xml 即(project object model)中配置项目所依赖的 jar 包，从而达到在构建项目的时候， 先从本地仓库中查找，如果不存在从内部私服查找，如果不存在最后再从外网 central 服务器查找的机制，达到了节省下载带宽，提高开发效率，以及 jar 包重用的目的。 咱们使用 maven 是如何使用的： Profiles maven 的常用命令 mvn eclipse:clean eclipse:eclipse -Dwtpversion=2.0 mvn clean package maven 的生命周期是独立的，但是生命周期下的阶段是相互关联并且延续的。maven 的生命周期 clean(清理):clean;default(默认):compile,test,packageinstall;site(站点)

15.21 接口

常用的接口开发方式主要有 Httpclient、Webservice、hession、ESB、dubbo 几种。(restful 风格接口的一种规范，(把所有的请求归纳为 4 大类、get（查）、post（改）、detete（删）、put（增））

1、 Httpclient 本身就是 restful 分格 （模仿浏览器客户端获取数据的一种方式。用 get （获取）、post （提交)），使用比较简单，现在应用的比较多。

a) 非 REST 风格的 url：localhost:8080/springmvc?name=小白&password=123;

b) REST 风格的 url：localhost:8080/springmvc/小白/123；(参数用@ PathVariable 接收)

2、Webservice 是传统开发中最常用的接口调用方式，(有四种实现工具：cxf、xfire、axis、axis2)。我用过 cxf 方式，因为 cxf 方式和 spring 集成比较简单。开始 webservice 是基于 soap 协议的，数据格式是 xml，现在也支持 restful。

3、hession 我也就简单了解了一下，他是采用二进制协议的。相比 WebService，Hessian 更简单、快捷。使用方法当时网上看过，也做过个例子，但现在忘了。

4、ESB 是企业服务总线，是企业内部多个项目之间接口调用比较频繁时的一种解决方案；所有项目都和 ESB 交互。不需要知道谁提供的服务（数据转换，调用监控）

ESB 帮忙管理服务，进行数据转换

ESB 处理高并发时，容易出现问题。ActiveMq（消息队列 MQ）开源：mule 收费：IBM ESB

5、Dubbo 一般叫服务注册，是分布式立体架构中做的一种治理服务的方案。不负责数据转换，只管中间搭线。(支持负载均衡和分布式）dubbo 更适合于一些访问量大，高并发的项目，比如电商；

15.22 App 接口开发

1、 由于移动端 app 一般时不能直接访问数据库的，所以需要我们 Java 后台开发接口，供移动端去调用；

2、 App 接口的开发一般都是 restful 分格的接口。就是不需要跳转页面，都是直接返回需要的数据。所有的方法上都需要@ResponseBody 注解，把返回结果转换成 json。所以我们一般用@RestController 代替@controller；

3、 因为 app 接口如果获取不到想要数据，就很可能发生闪退，所以我们需要在 controller 中的所有方法中，用 try catch 捕获异常，把异常也返回。

4、 为了让用户识别异常，我们一般需要定义一组错误编码。

5、 由于接口比较多，为了方便客户端处理接口返回数据，我们定义了一个统一的返回对象类。里边主要包含三个

参数：Boolean success，String code，Object data；

6、 安全方面考虑，接口最后都要把 http 协议转换成 https（http+CA 证书）协议。

7、 在接口的拦截器里边，采用对称加密的方式，进行签名认证。 对称加密就是服务端给调用端一个 appid 和一个 appsecret；

8、 接口文档：接口说明、接口的 url，传入参数、响应参数。

9、 接口自己测试：一般在浏览器上安装一个 restclient 的插件，用这个插件就可以测试。

10、 接口的调试：方法一：手机连接公司的内外；方法二：通过 Nginx 把自己的 ip 代理到外网。

15.23 微信公众号

首先微信提供了一个官方的微信公众号平台，我们申请成功以后，可以在这个平台上直接进行配置。比如消息回复，有消息自动回复，被关注时被动回复、关键字回复等。回复的消息包括文字消息、图片消息、语音消息、视频消息等。还有就是自定义菜单，最多可以配置 3*5 个菜单，点击菜单可以回复消息，跳转链接，跳转小程序。如果想要进一步使用更复杂的功能，就需要成为开发者，调用他的 api 接口文档了。

微信公众号的接口都是 restful 风格的接口，我们可以用 httpclient 直接调用，接口有些是 json 数据格式，有些事 xml 数据格式。调用接口文档首先需要接入测试和网页授权。接入时，必须使用域名和 80 端口，在公众号平台配置好 url，发送请求，平台 url 里边接收参数，按照要求生成签名，与传过来的签名进行比较，验证签名，如果成功返回一个字符串。这样就接入成功了。Json 和 xml 的解析我们当时封装了工具类。解析方法网上都有，很简单。

15.24 Restful

Restful（ Representational State Transfer 表现层状态转化 ），是一种软件架构设计的设计风格而不是标准，只是提供了一组设计原则和约束条件。 它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。他的理念是把 每一个 URI 代表一种资源，浏览器可以通过 http 的四个关键词对这种资源进行操作。形成四种不同的请求（get、post、put、delete）；但 java 里边一般只支持 get 和 post，如果想要支持 put 和 delete，需要在 web.xml 中加一个过滤器，还需要前台传一个_method 的参数，告诉后台是 put 还是 delete 请求。个人感觉也不是很方便。我们用 restful 分格，一种是写 rest 风格的 api，一种是写 rest 风格的 url。rest 风格的 url 就是用路径传参，代替传统的？号传参。用注解@PathVariable 来接收参数。rest 风格的 api，就是把同一个资源的请求，用一个 rui 表示，用四个关键字来区分不同的请求。使得接口调用人员不会对请求的资源地址产生混淆和大量的检查方法名的麻烦，形成一个统一的接口。另外，Rest 风格的接口方法，一定是直接响应客户数据的，而不是跳转页面的，在 springmvc 中，一般我们把所有需要用到 ReponseBody 注解响应数据的方法，成为 rest 方法。他还提供了一个@RestController 的注解，用来代替@Controller+@ReponseBody;

15.25 webservice

webservice 是一种跨平台，跨语言，跨框架的接口开发和调用技术。在 java 中通常有四种技术框架分别是 xfire,cxf, axis，axis2。我们用的是 cxf，因为 cxf 使用简单，并且可以和 spring 无缝集成.webservice 使用 wsdl（web service definition language - web service 定义语言）语言，soap（简单对象）协议、xml 数据格式；webservice 服务端配置流程（没必要主动说）

首先在 web.xml 中引入 cxfServlet 核心类，指定对以/cxf 开头的 url 路径提 webservice 服务，之后我们在要发布成服务的接口的实现类上添加@webservice 注解，之后在 spring-webservice.xml 中发布 webservice 服务，通过 jaxws:endpoint 这个标签，并且 在标签配置 implementor 和 address 来表明实现服务的类，以及发布的地址，最后

在浏 览器中输入相关的 webservice 地址?wsdl 来验证服务是否发布成功。

webservice 客户端的配置（没必要主动说）

首先通过 wsdl2java 根据发布的 webservice 服务端地址的 wsdl 生成客户端调用的中 间桥梁 java 类，将生成的 java 类拷贝到客户端项目中，配置 spring-client.xml 文件， 通过 jaxws:client 定义一个 bean,并通过 address 属性指明要访问的 webservice 的服务地 址，通过 serviceClass 指明充当中间桥梁 的 服 务类，之后获取该 bean, 就可以通过它来 访问发布的 webservice 接口中的方法。

15.26 httpclient

HttpClient 字面理解就是 http 请求的客户端，他是 Apache 开发的一套 HTTP 协议的客户端编程工具包。是纯 Java 语音编写的，支持 https 协议，实现了 http 的 get、post、delete、put 等全部方法，但我们一般只用 get 和 post 方法，用 httpclient 可以直接跨域请求 http 协议的 url，并拦截返回的响应结果。我们项目里边有写好的工具类，里边封装好了他的 get 和 post 方法，传入 url 和参数，返回 String 类型数据（一般是 json 字符串或 xml 字符串），然后我们进行解析就可以了。

他的调用步骤是：

1. 创建 HttpClient 对象。
2. 创建请求方法的实例，并指定请求 URL。如果需要发送 GET 请求，创建 HttpGet 对象；如果需要发送 POST 请求，创建 HttpPost 对象。
3. 如果需要发送请求参数，可调用 HttpGet、HttpPost 共同的 setParams(HetpParams params)方法来添加请求参数；对于 HttpPost 对象而言，也可调用 setEntity(HttpEntity entity)方法来设置请求参数。
4. 调用 HttpClient 对象的 execute(HttpUriRequest request)发送请求，该方法返回一个 HttpResponse。
5. 调用 HttpResponse 的 getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头；调用 HttpResponse 的 getEntity()方法可获取 HttpEntity 对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。
6. 释放连接。无论执行方法是否成功，都必须释放连接

15.27 项目文档

1、需求规格说明书：根据客户的或者我们调研到的模糊需求，进行分析整理后获得，主要包括：项目背景、项目预期、项目的目的、项目的主要功能；

2、概要设计：根据需求规格说明书进行项目设计。项目的架构、框架、主要技术、最低服务器要求、涉及到的第三方接口或应用，主要功能模块（对每个模块进行描述）；

3、详细设计：对概设的进一步细化,目的是为了让即使没有参与需求分析和设计的开发人员，看着这个文档，就可以准确的完成开发任务。功能模块的详细设计（用户列表显示哪些字段、增删改按钮如何摆放）；数据库设计（powerdesigner 和 excel）（表、字段、字段属性、表关系）；代码规范、页面风格和规范；接口设计；在详细设计的同时，每个做 demo；

4、开发计划（project 或者 excel，具体到每个任务多少（人/天），分成几个大的里程碑（基本模块完成、整体流程实现、第一版测试版本发布、试运营））；

5、项目风险评估；（工期的风险、质量的风险、硬件或第三方的风险）；

6、日报、周报、月报，日后、周例会、月例会；

7、部署文档、使用说明书、技术文档（某个技术如何使用的）；

8、验收文档（明确哪些功能已经完成，和需求的匹配度）（内部、客户验收（上线试运行验收（支付第二笔款，第一笔款签合同时支付）），客户最终验收（运维 3 个月或 1 年之后，支付第三笔款）（352）（343））

15.28项目中遇到什么问题

- 跨域，利用 cors 解决，
- 内存溢出问题：在 jvm 参数中配置如果内存溢出导出 dump 文件，通过 java 的 jvisualvm 装载 dump 文件，对象从大到小进行排列最大的对象一定是出问题的对象，分析为什么会有这么的对象产生，然后修改代码，调大 jvm 参数。可以调节 tomcat 内存解决。还要找出代码中内存溢出的漏洞，一般是流没有关闭或者死循环；
- 数据库连接超出最大连接量：修改 mysql 配置文件的最大连接数；
- 代码编译失败、缓存：清理 Eclipse 编译缓存、清理 tomcat 缓存、清理浏览器缓存。

15.29项目的重构，代码的优化

1、 现在的项目，一般都采用迭代式开发的模式，就是在需求不是特别确定，只有大概需求时，就快速进入开发，把开发分成多个阶段，需求一部分一部分的完成。这样的开发模式，可以适应现在千变万化的互联网模式，
但是当后期项目做大以后，就会遗留很多问题。促使我们不得不去对项目进行重构，对代码进行优化。
2、 项目的重构主要是从项目的整体架构上考虑，对项目的各个模块进行解耦，降低各个模块的关联性，方便以后进行集群或升级。
3、 代码的优化，主要是针对各个具体流程节点的代码逻辑的优化，
4、 一方面从 java 的三大核心思想：封装、继承、多态方面考虑。
5、 一方面从代码思维逻辑的严谨性考虑。
6、 还有就是通用接口、工具类等的性能的优化。
7、 另外代码的简洁、可读性也非常重要，所以代码的注释和规范性也是优化必须注意的一个地方。

15.30开发组多少人

公司性质：我们是一个小的外包公司，主要得业务是电商方案服务商，在 XX 电子商务公司做电商的开发，他们有自己的大数据团队主要做数据分析，和用户行为分析，和云服务。

我们技术团队大概有 40 人，我们开发组有 13 个左右，1 个技术经理，8 个开发，1 个测试 3 个前端。用户量 100w 左右用户，每天的页面访问量 60w 左右，每秒请求 1w 左右，秒杀美秒 8w，页面访问量达到 200-500 左右。机房是租的，硬件服务器 20 几台，剩下的租的阿里云。