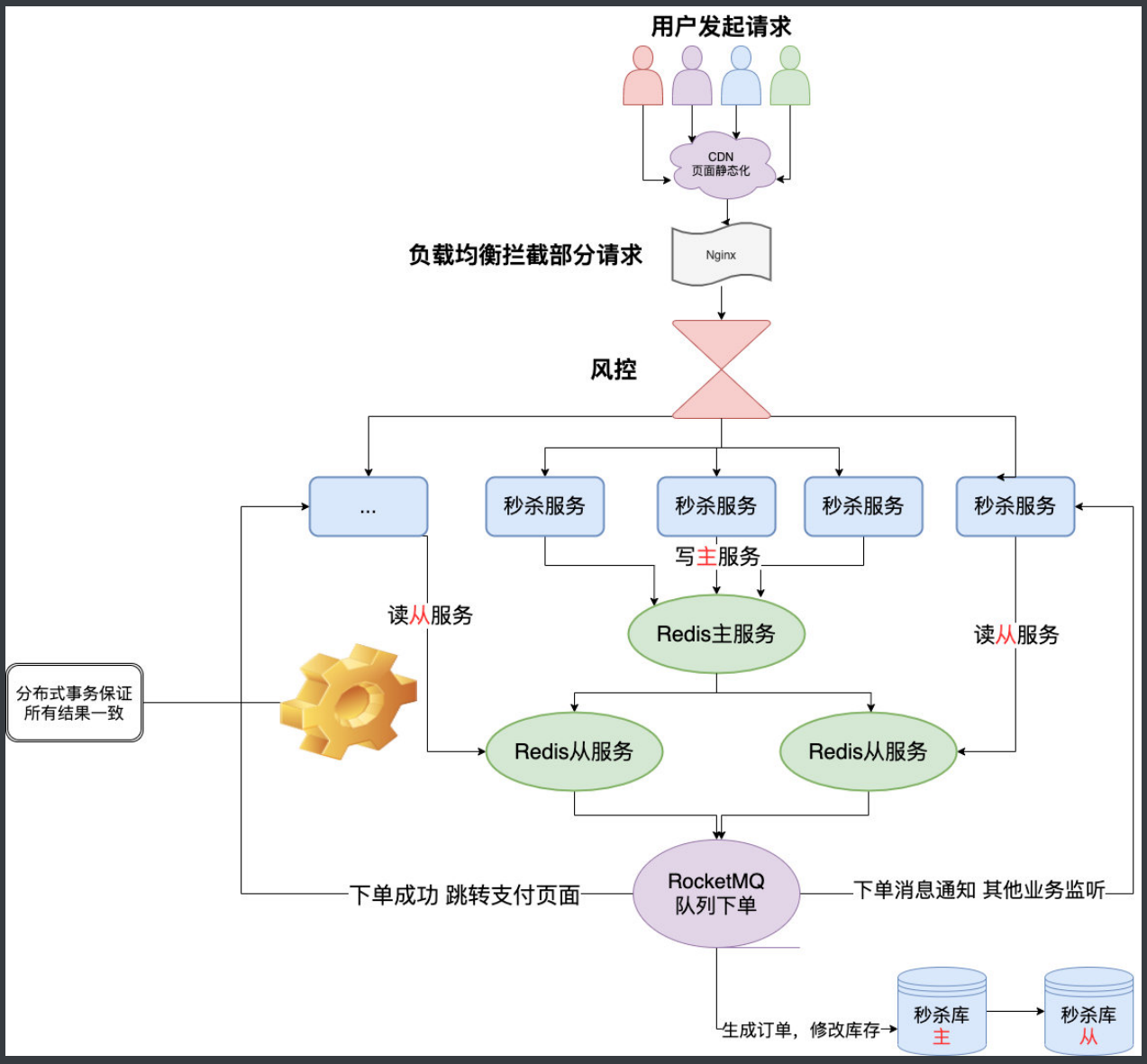
> 点赞再看，养成习惯，微信搜一搜【**三太子敖丙**】关注这个互联网苟且偷生的工具人。
>
> 本文 **GitHub** https://github.com/JavaFamily 已收录，有一线大厂面试完整考点、资料以及我的系列文章。

背景

我之前写过一个秒杀系统的文章不过有些许瑕疵，所以我准备在之前的基础上进行二次创作，不过让我决心二创秒杀系统的原因是我最近面试了很多读者，动不动就是秒杀系统把我整蒙蔽了，我懵的主要是秒杀系统的细节大家都不知道，甚至不知道电商公司一个秒杀系统的组成部分。

我之前在某电商公司就是做电商活动的，所以这样的场景和很多解决方案我是比较清楚的，那我就从我自身去带着大家看看一个秒杀的设计细节以及中间各种解决方案的利弊，以下就是我设计的秒杀系统，几乎涵盖了市面上所有秒杀的实现细节：



正文

首先设计一个系统之前，我们需要先确认我们的业务场景是怎么样子的，我就**带着大家一起假设一个场景**好吧。

我们现场要卖1000件下面这个**婴儿纸尿裤**，然后我们根据以往这样秒杀活动的数据经验来看，目测来抢这100件纸尿裤的人足足有10万人。（南极人打钱！）

你一听，完了呀，这我们的服务器哪里顶得住啊！说真的直接打DB肯定挂，但是别急嘛，有**暖男**敖丙在，任何系统我们开始设计之前我们都应该去思考**会出现哪些问题**？这里我罗列了几个非常经典的问题：

问题

高并发：

是的**高并发**这个是我们想都不用想的一个点，一瞬间这么多人进来这不是高并发什么时候是呢？

是吧，秒杀的特点就是这样**时间极短**、**瞬间用户量大**。

正常的店铺营销都是用极低的价格配合上短信、APP的精准推送，吸引特别多的用户来参与这场秒杀，**爽了商家苦了开发呀**。

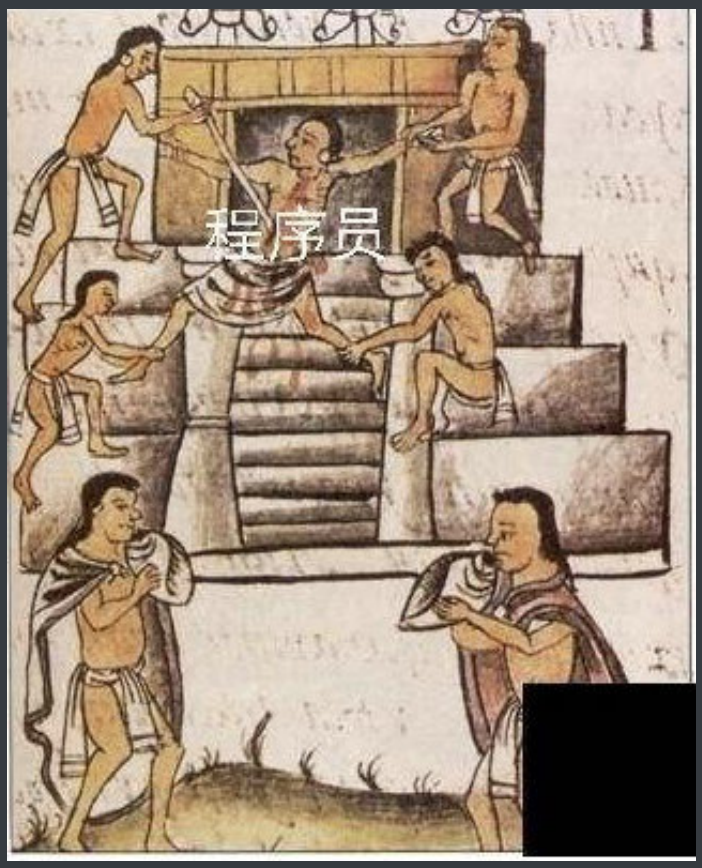
秒杀大家都知道如果真的营销到位，价格诱人，几十万的流量我觉得完全不是问题，那单机的**Redis**我感觉3-4W的QPS还是能顶得住的，但是再高了就没办法了，那这个数据随便搞个热销商品的秒杀可能都不止了。

大量的请求进来，我们需要考虑的点就很多了，**缓存雪崩**，**缓存击穿**，**缓存穿透**这些我之前提到的点都是有可能发生的，出现问题打挂DB那就很难受了，活动失败用户体验差，活动人气没了，最后背锅的还是**开发**。

超卖：

但凡是个秒杀，都怕**超卖**，我这里举例的只是尿不湿，要是换成100个MacBook Pro，商家的预算经费卖100个可以赚点还可以造势，结果你写错程序多卖出去200个，你不发货用户**投诉你**，平台**封你店**，你发货就**血亏**，你怎么办？（没事看了敖丙的文章直接不怕）

那最后只能**杀个开发祭天**解气了，秒杀的价格本来就低了，基本上都是不怎么赚钱的，超卖了就恐怖了呀，所以超卖也是很关键的一个点。



恶意请求：

你这么低的价格，假如我抢到了，我转手卖掉我不是**血赚**？就算我不卖我也不亏啊，那用户知道，你知道，别的别有用心的人（黑客、黄牛...）肯定也知道的。

那简单啊，我知道你什么时候抢，我搞个几十台机器搞点脚本，我也模拟出来十几万个人左右的请求，那我是不是意味着我基本上有80%的成功率了。

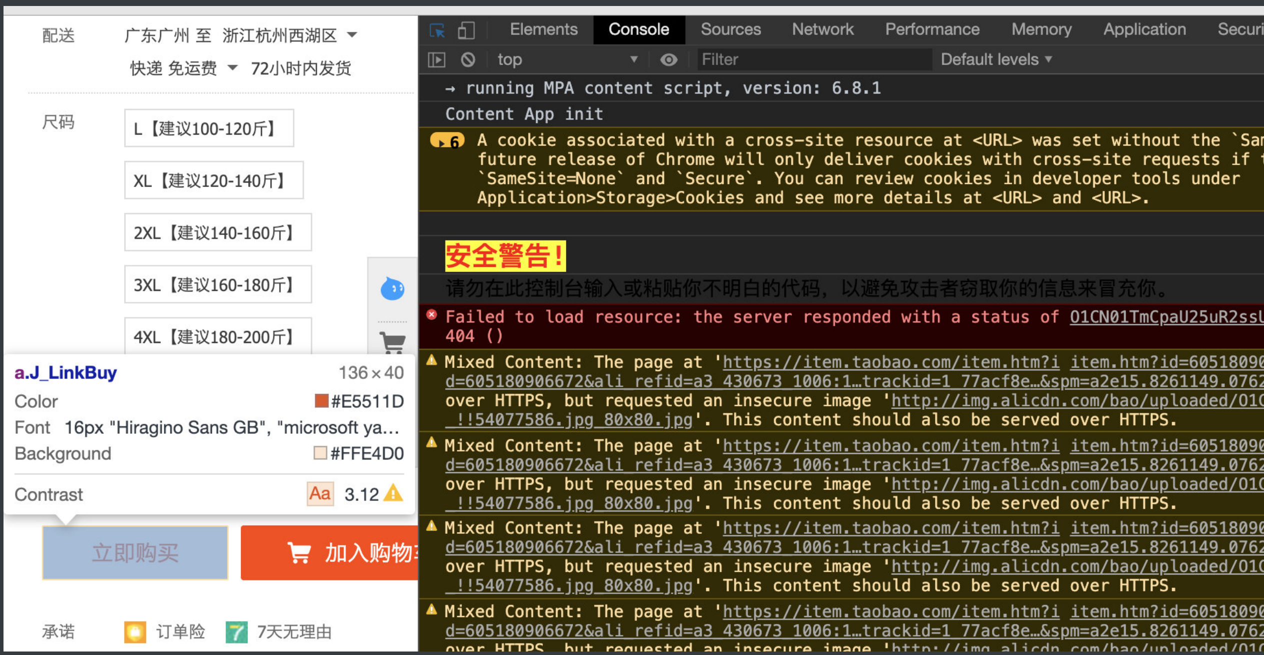
真实情况可能远远不止，因为机器请求的速度比人的手速往往快太多了，在贵州的敖丙我每年回家抢高铁票都是**秒光**的，我也不知道有没有黄牛的功劳，我要Diss你，黄牛。杰伦演唱会门票抢不到，我也Diss你。

Tip：科普下，小道消息了解到的，黄牛的抢票系统，比国内很多小公司的系统还吊很多，架构设计都是顶级的，我用**顶配的服务**加上**顶配的架构设计**，你还想看演唱会？还想回家？

不过不用黄牛我回家都难，我们云贵川跟我一样要回家过年的仔太多了555！

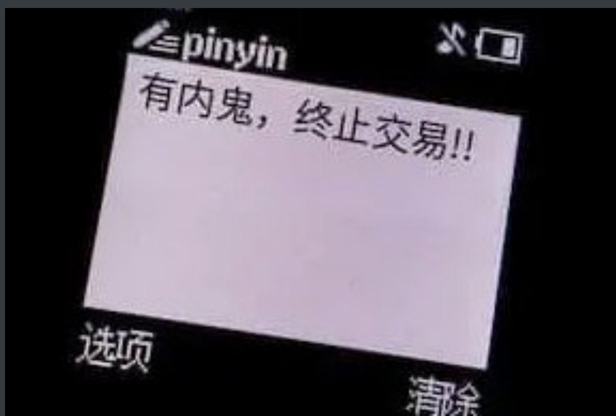
链接暴露：

前面几个问题大家可能都很好理解，一看到这个有的小伙伴可能会比较疑惑，啥是**链接暴露**呀？



相信是个开发同学都对这个画面一点都不陌生吧，懂点行的仔都可以打开谷歌的**开发者模式**，然后看看你的网页代码，有的就有URL，但是我写VUE的时候是事件触发然后去调用文件里面的接口看源码看不到，但是我可以点击一下**查看你的请求地址**啊，不过你好像可以对按钮在秒杀前置灰。

不管怎么样子都有危险，撇开外面的所有的东西你都挡住了，你卖这个东西实在便宜得过分，有诱惑力，你能保证**开发不动心**？开发知道地址，在秒杀的时候自己提前请求。。。（开发：怎么TM又是我）



数据库：

每秒上万甚至十几万的**QPS**（每秒请求数）直接打到**数据库**，基本上都要把库打挂掉，而且你服务不单单是做秒杀的还涉及其他的业务，你没做**降级、限流、熔断**啥的，别的一起挂，小公司的话可能**全站崩溃404**。

反正不管你秒杀怎么挂，你别把别的搞挂了对吧，搞挂了就不是杀一个程序员能搞定的。

程序员：我TM好难啊！

问题都列出来了，那怎么设计，怎么解决这些问题就是接下去要考虑的了，我们对症下药。

我会从我设计的秒杀系统从上到下去给大家介绍我们正常电商秒杀系统在每一层做了些什么，每一层存在的问题，难点等。

我们从前端开始：

前端

秒杀系统普遍都是商城网页、H5、APP、小程序这几项。

在前端这一层其实我们可以做的事情有很多，如果用node去做，甚至能直接处理掉整个秒杀，但是node其实应该属于后端，所以我不讨论node Service了。

资源静态化：

秒杀一般都是特定的商品还有页面模板，现在一般都是前后端分离的，页面一般都是不会经过后端的，但是前端也要自己的服务器啊，那就把能提前放入**cdn服务器**的东西都放进去，反正把所有能提升效率的步骤都做一下，减少真正秒杀时候服务器的压力。

秒杀链接加盐：

我们上面说了链接要是提前暴露出去可能有人直接访问url就提前秒杀了，那又有小伙伴要说了我做个时间的校验就好了呀，那我告诉你，知道链接的地址比起页面人工点击的还是有**很大优势**。

我知道url了，那我通过程序不断获取最新的北京时间，可以达到**毫秒级别**的，我就在00毫秒的时候请求，我敢说绝对比你人工点的成功率大太多了，而且我可以一毫秒发送N次请求，搞不好你卖100个产品我全拿了。

那这种情况怎么避免？

简单，把**URL动态化**，就连写代码的人都不知道，你就通过MD5之类的摘要算法加密随机的字符串去做url，然后通过前端代码获取url后台校验才能通过。

这个只能防止一部分没耐心继续破解下去的黑客，有耐心的人研究出来还是能破解，在电商场景存在很多这样的羊毛党，那怎么做呢?

后面我会说。

限流：

限流这里我觉得应该分为**前端限流**和**后端限流**。

物理控制：

大家有没有发现没到秒杀前，一般按钮都是**置灰**的，只有时间到了，才能点击。

这是因为怕大家在时间快到的最后几秒秒疯狂请求服务器，然后还没到秒杀的时候基本上服务器就挂了。

这个时候就需要前端的配合，定时去请求你的后端服务器，获取最新的北京时间，到时间点再给按钮可用状态。

按钮可以点击之后也得给他置灰几秒，不然他一样在开始之后一直点的。

你敢说你们秒杀的时候不是这样的?

前端限流：这个很简单，一般秒杀不会让你一直点的，一般都是点击一下或者两下然后几秒之后才可以继续点击，这也是保护服务器的一种手段。

后端限流：秒杀的时候肯定是涉及到后续的订单生成和支付等操作，但是都只是成功的幸运儿才会走到那一步，那一旦100个产品卖光了，return了一个false，前端直接秒杀结束，然后你后端也关闭后续无效请求的介入了。

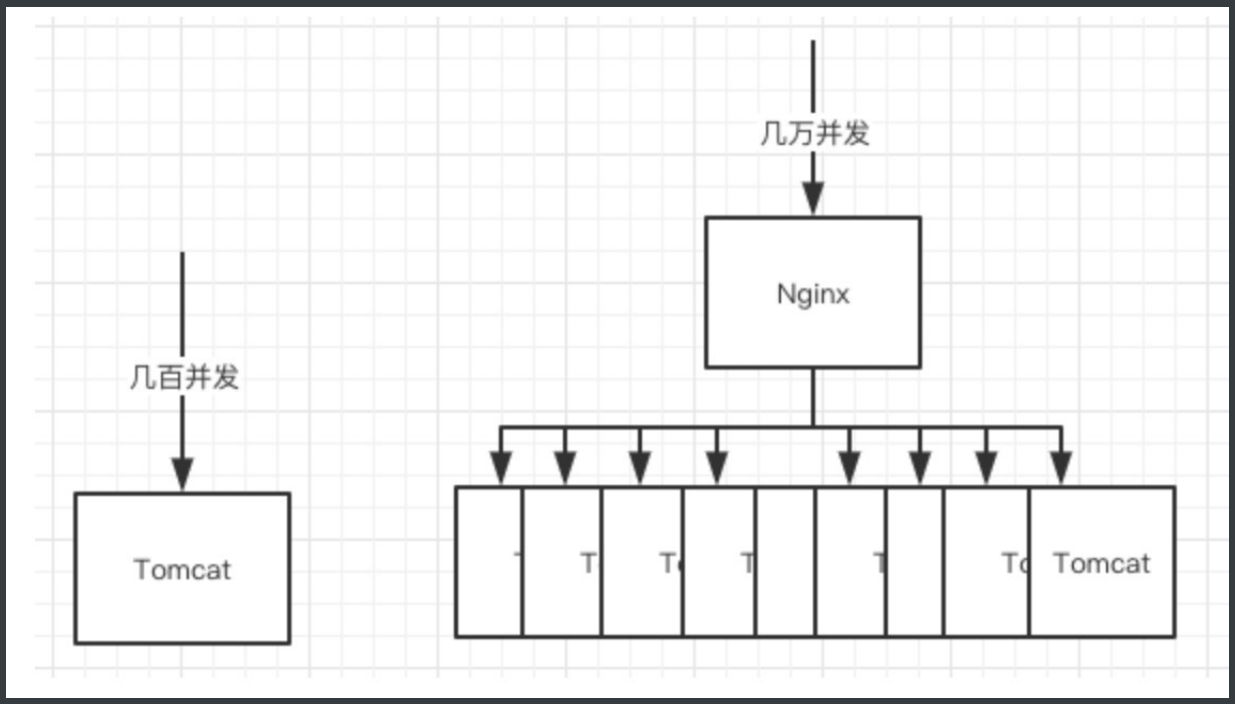
Tip：真正的限流还会有限流组件的加入例如：阿里的Sentinel、Hystrix等。我这里就不展开了，就说一下物理的限流。

我们卖1000件商品，请求有10W，我们不需要把十万都放进来，你可以放1W请求进来，然后再进行操作，因为秒杀对于用户本身就是黑盒的，所以你怎么做的他们是没感知的，至于为啥放1W进来，而不是刚好1000，是因为会丢掉一些薅羊毛的用户，至于怎么判断，后面的风控阶段我会说。

Nginx：

Nginx大家想必都不陌生了吧，这玩意是**高性能的web服务器**，并发也随便顶几万不是梦，但是我们的**Tomcat**只能顶几百的并发呀，那简单呀**负载均衡**嘛，一台服务几百，那就多搞点，在秒杀的时候多租点**流量机**。

Tip：据我所知国内某大厂就是在去年春节活动期间租光了亚洲所有的服务器，小公司也很喜欢在双十一期间买流量机来顶住压力。



这样一对比是不是觉得你的集群能顶很多了。

恶意请求拦截也需要用到它，一般单个用户请求次数太夸张，不像人为的请求在网关那一层就得拦截掉了，不然请求多了他抢不抢得到是一回事，服务器压力上去了，可能占用网络带宽或者把**服务器打崩、缓存击穿**等等。

风控

我可以明确的告诉大家，前面的所有措施还是拦不住很多羊毛党，因为他们是专业的团队，他们可以注册很多账号来薅你的羊毛，而且不用机器请求，就用群控，操作几乎跟真实用户一模一样。

那怎么办，是不是无解了?

这个时候就需要风控同学的介入了，在请求到达后端之前，风控可以根据账号行为分析出这个账号机器人的概率大不大，我现在负责公司的某些特殊系统，每个用户的行为都是会送到我们大数据团队进行分析处理，给你打上对应标签的。

那黑客其实也有办法：**养号**

他们去黑市买真实用户有过很多记录的账号，买到了还不闲着，帮他们去购物啥的，让系统无法识别他们是黑号还是真实用户的号。

怎么办?

通杀！是的没有办法，只能通杀了，通杀的意思就是，我们通过风管分析出来这个用户是真实用户的概率没有其他用户概率大，那就认为他是机器了，丢弃他的请求。

之前的限流我们放进来10000个请求，但是我们真正的库存只有1000个，那我们就算出最有可能是真实用户的1000人进行秒杀，丢弃其他请求，因为秒杀本来就是黑盒操作的，用户层面是无感知的，这样设计能让真实的用户买到东西，还可以减少自己被薅羊毛的概率。

风控可以说是流量进入的最后一道门槛了，所以很多公司的风控是很强的，蚂蚁金服的风控大家如果了解过就知道了，你的资金在支付宝被盗了，他们是能做到全款补偿是有原因的。

后端

服务单一职责：

设计个能抗住高并发的系统，我觉得还是得**单一职责**。

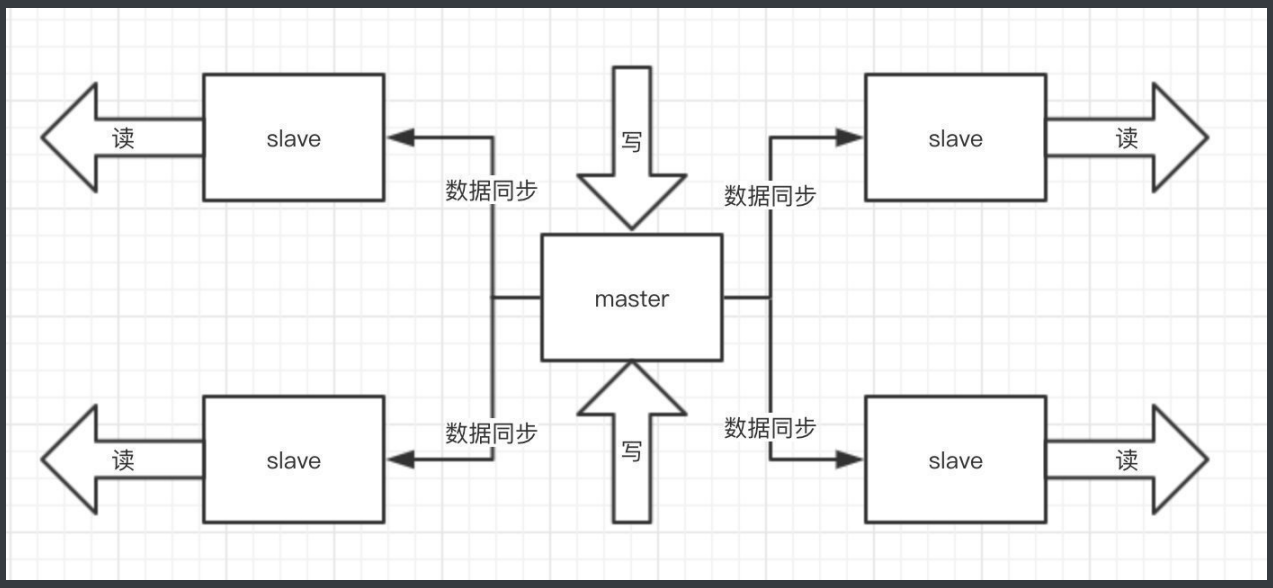
什么意思呢，大家都知道现在设计都是**微服务的设计思想**，然后再用**分布式的部署方式**。

也就是我们下单是有个订单服务，用户登录管理等有个用户服务等等，那为啥我们不给秒杀也开个服务，我们把秒杀的代码业务逻辑放一起。

单一职责的好处就是就算秒杀没抗住，秒杀库崩了，服务挂了，也不会影响到其他的服务。（高可用）

Redis集群：

之前不是说单机的**Redis**顶不住嘛，那简单多找几个兄弟啊，秒杀本来就是读多写少，那你们是不是瞬间想起来我之前跟你们提到过的，**Redis集群**，**主从同步**、**读写分离**，我们还搞点**哨兵**，开启**持久化**直接无敌高可用！



库存预热：

秒杀的本质，就是对库存的抢夺，每个秒杀的用户来你都去数据库查询库存校验库存，然后扣减库存，撇开性能因数，你不觉得这样好繁琐，对业务开发人员都不友好，而且数据库顶不住啊。

开发：你tm总算为我着想一次了。

那怎么办?

我们都知道数据库顶不住但是他的兄弟非关系型的数据库**Redis**能顶啊！

那不简单了，我们要开始秒杀前你通过定时任务或者运维同学**提前把商品的库存加载到Redis中**去，让整个流程都在Redis里面去做，然后等秒杀介绍了，再异步的去修改库存就好了。

但是用了Redis就有一个问题了，我们上面说了我们采用**主从**，就是我们会去读取库存然后再判断然后有库存才去减库存，正常情况没问题，但是高并发的情况问题就很大了。

多品几遍！！！就比如现在库存只剩下1个了，我们高并发嘛，4个服务器一起查询了发现都是还有1个，那大家都觉得是自己抢到了，就都去扣库存，那结果就变成了-3，是的只有一个是真的抢到了，别的都是超卖的。咋办?

事务：

Redis本身是支持事务的，而且他有很多原子命令的，大家也可以用LUA，还可以用他的管道，乐观锁他也知支持。

限流&降级&熔断&隔离：

这个为啥要做呢，不怕一万就怕万一，万一你真的顶不住了，**限流**，顶不住就挡一部分出去但是不能说不行，**降级**，降级了还是被打挂了，**熔断**，至少不要影响别的系统，**隔离**，你本身就独立的，但是你会调用其他的系统嘛，你快不行了你别拖累兄弟们啊。

消息队列（削峰填谷）：

一说到这个名词，很多小伙伴就知道了，对的**MQ**，你买东西少了你直接100个请求改库我觉得没问题，但是万一秒杀一万个，10万个呢？服务器挂了，**程序员又要背锅的**。

秒杀就是这种瞬间流量很高，但是平时又没有流量的场景，那消息队列完全契合这样的场景了呀，削峰填谷。

Tip：**可能小伙伴说我们业务达不到这个量级，没必要。但是我想说我们写代码，就不应该写出有逻辑漏洞的代码，至少以后公司体量上去了，别人一看居然不用改代码，一看代码作者是敖丙？有点东西！**

你可以把它放消息队列，然后一点点消费去改库存就好了嘛，不过单个商品其实一次修改就够了，我这里说的是**某个点多个商品**一起秒杀的场景，像极了双十一零点。

数据库

数据库用MySQL只要连接池设置合理一般问题是不大的，不过一般大公司不缺钱而且秒杀这样的活动十分频繁，我之前所在的公司就是这样秒杀特卖这样的场景一直都是不间断的。

单独给秒杀建立一个数据库，为秒杀服务，表的设计也是竟可能的简单点，现在的互联网架构部署都是**分库**的。

至于表就看大家怎么设计了，该设置索引的地方还是要设置索引的，建完后记得用**explain**看看**SQL**的执行计划。（不了解的小伙伴也没事，MySQL章节去康康）

分布式事务

这为啥我不放在后端而放到最后来讲呢?

因为上面的任何一步都是可能出错的，而且我们是在不同的服务里面出错的，那就涉及分布式事务了，但是分布式事务大家想的是一定要成功什么的那就不对了，还是那句话，几个请求丢了就丢了，要保证时效和服务的可用可靠。

所以**TCC**和**最终一致性**其实不是很适合，TCC开发成本很大，所有接口都要写三次，因为涉及TCC的三个阶段。

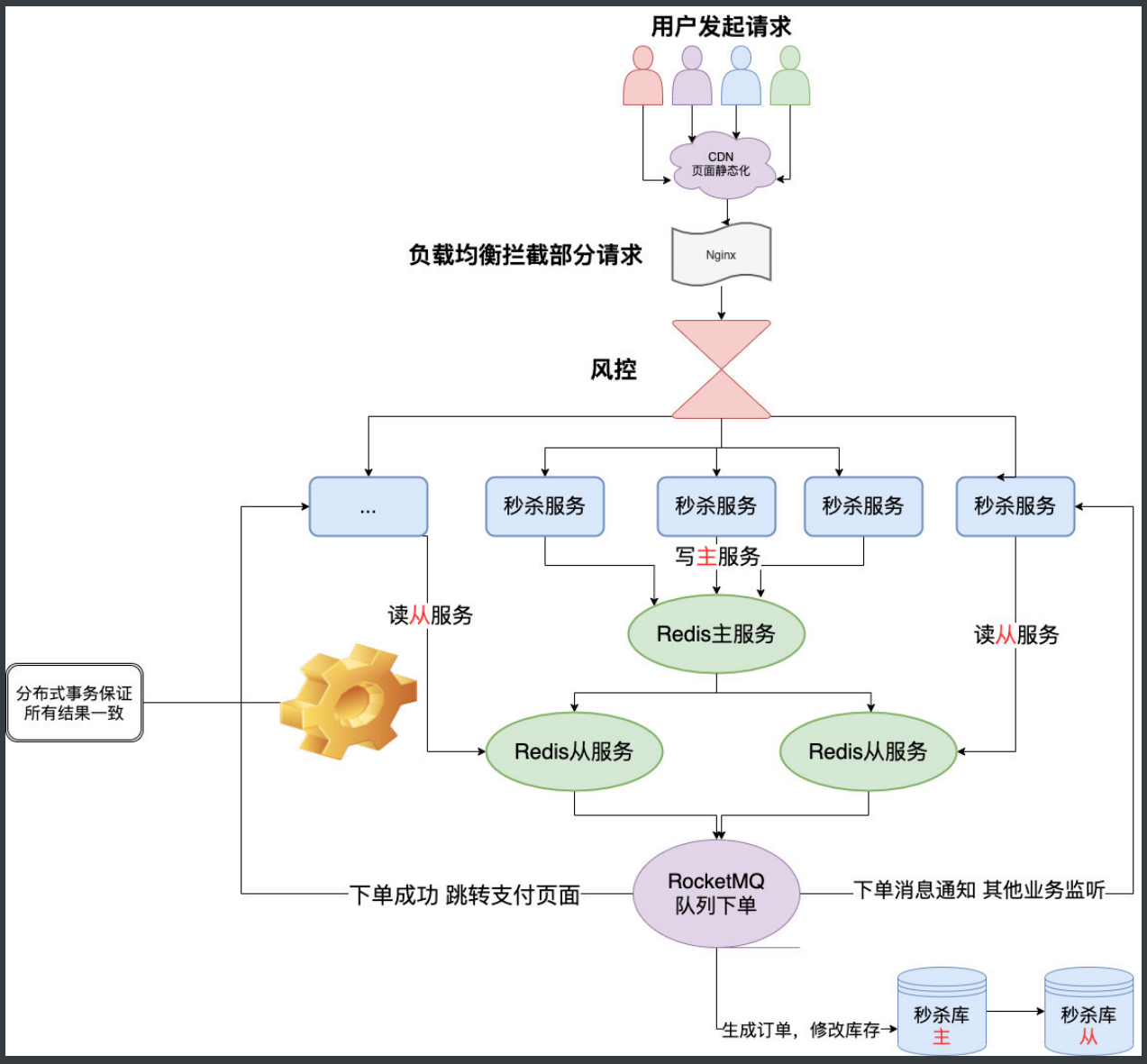
最终一致性基本上都是靠轮训的操作去保证一个操作一定成功，那时效性就大打折扣了。

大家觉得不那么可靠的**两段式（2PC）**和**三段式（3PC）**就派上用场了，他们不一定能保证数据最终一致，但是效率上还算ok。

总结

到这里我想我已经基本上把该考虑的点还有对应的解决方案也都说了一下，不知道还有没有没考虑到的，但是就算没考虑到我想我这个设计，应该也能撑住一个完整的秒杀流程。

最后大家再看看这个秒杀系统或许会有新的感悟，是不是一个系统真的没有大家想的那么简单，而且我还是有漏掉的细节，这是一定的。



秒杀这章我脑细胞死了很多，考虑了很多个点，最后还是出来了，忍不住给自己**点赞**！

总结

我们玩归玩，闹归闹，别拿面试开玩笑。

秒杀不一定是每个同学都会问到的，至少肯定没**Redis基础**那样常问，但是一旦问到，大家一定要回答到点上。

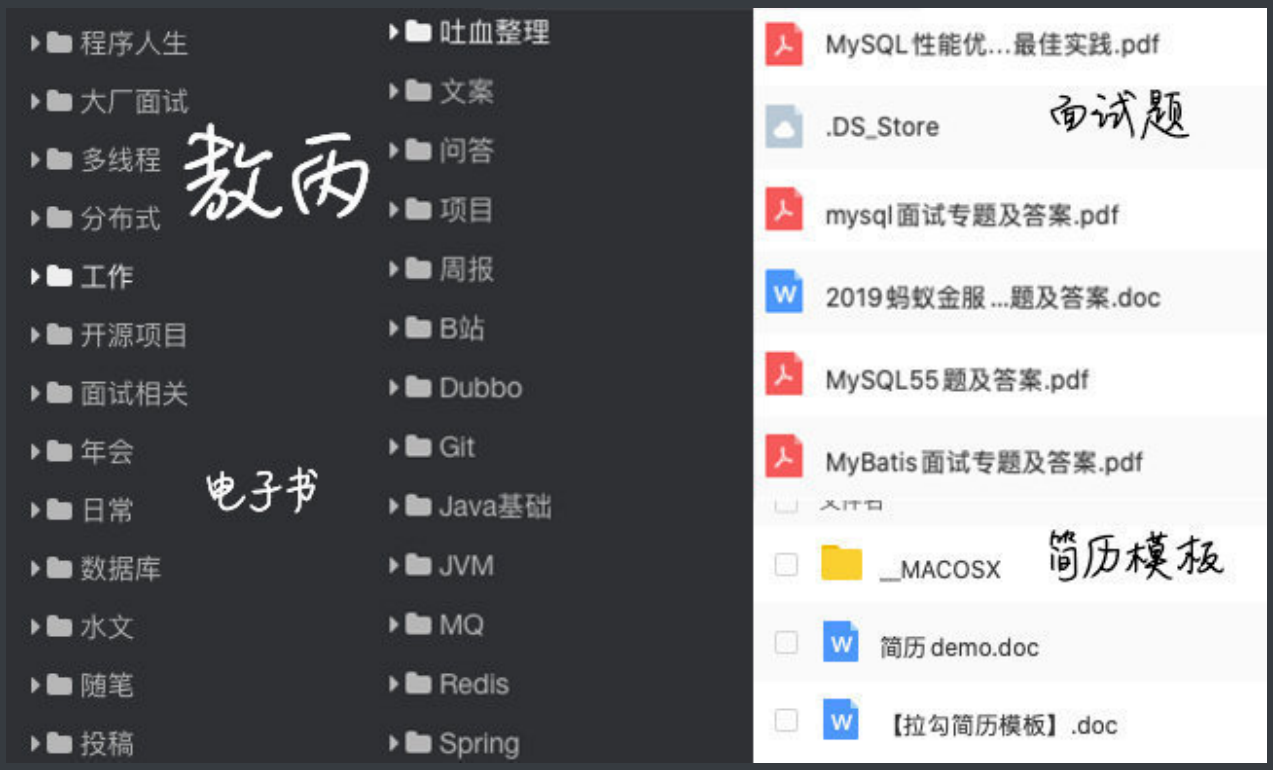
至少你得说出**可能出现的情况**，**需要注意的情况**，以及对于的**解决思路和方案**，因为这才是一个coder的基本素养，这些你不考虑你也很难去进步。

最后就是需要对整个链路比较熟悉，注意是一个完整的链路，前端怎么设计的呀，网关的作用呀，怎么**解决Redis的并发竞争**啊，**数据的同步方式**呀，**MQ的作用**啊等等，相信你会有不错的收获。

不知道这是一次成功还是失败的二创，我里面所有提到的技术细节我都写了对应的文章，大家可以关注我去历史文章看看，天色已晚，我溜了。

絮叨

另外，敖丙把自己的面试文章整理成了一本电子书，共 1630页！目录如下，还有我复习时总结的面试题以及简历模板



现在免费送给大家，在我的公众号**三太子敖丙**回复 **【888】** 即可获取。

我是敖丙，你知道的越多，你不知道的越多，我们下期见！

人才们的 **【三连】** 就是敖丙创作的最大动力，如果本篇博客有任何错误和建议，欢迎人才们留言！

> 文章持续更新，可以微信搜一搜「 **三太子敖丙** 」第一时间阅读，回复【**资料**】有我准备的一线大厂面试资料和简历模板，本文 **GitHub** https://github.com/JavaFamily 已经收录，有大厂面试完整考点，欢迎Star。