场景

我们现在要卖100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，我们在开始之前应该先思考下会出现哪些问题？

问题

* 高并发
* 超卖
* 恶意请求
* 链接暴露
* 数据库

如何解决这些问题？

服务单一职责

现在的设计都是微服务的设计思想，分布式的部署方式。

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。

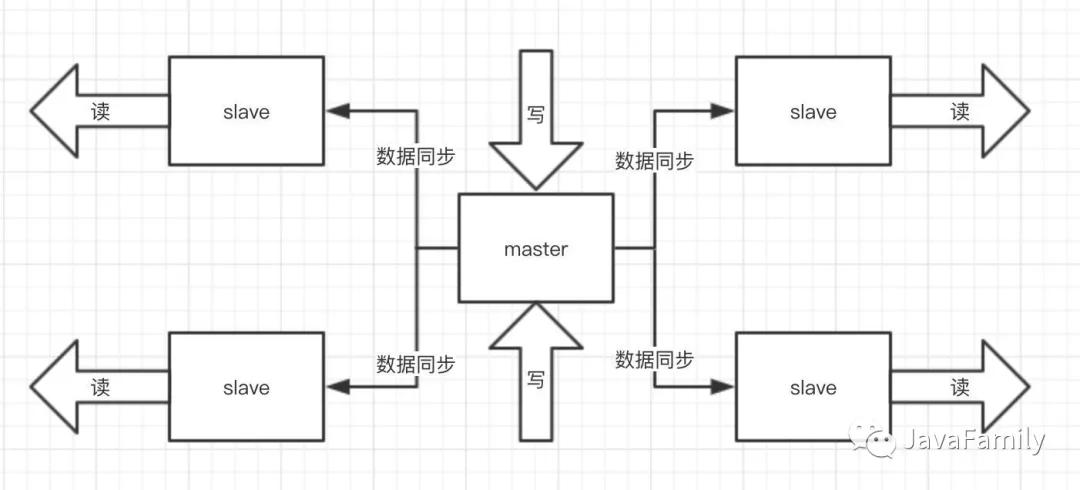
我知道url了，那我通过程序不断获取最新的北京时间，可以达到毫秒级别的，我就在00毫秒的时候请求，我敢说绝对比你人工点的成功率大太多了，而且我可以一毫秒发送N次请求，搞不好你卖100个产品我全拿了。

这种情况如何避免？

简单，把**URL动态化**，就连写代码的人都不知道，你就通过MD5之类的加密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。

Redis集群

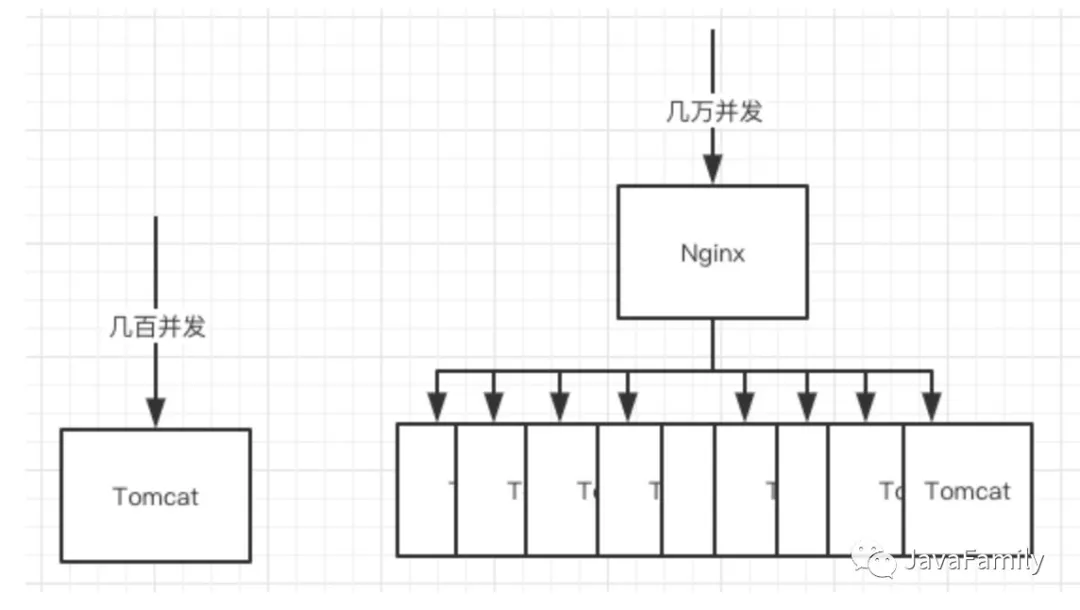
之前不是说单机的Redis顶不住嘛，那简单多找几个兄弟啊，秒杀本来就是读多写少，那你们是不是瞬间想起来我之前跟你们提到过的，Redis集群，主从同步、读写分离，我们还搞点哨兵，开启持久化直接无敌高可用！



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。

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，所以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，减少真正秒杀时服务器的压力。

按钮控制

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，不然一样在开始之后一直点的。

**限流**

限流这里我觉得应该分为**前端限流**和**后端限流**。

**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。

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，撇开性能因素，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。

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，然后等秒杀结束了，再异步地去修改库存就好了。

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。

这里我就不画图了，我本来想画图的，想了半天我觉得语言可能更好表达一点。

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，只有一个是真的抢到了，别的都是超卖的。咋办？

Lua

之前的文章就简单的提到了他，我今天就多一点篇幅说一下吧。

Lua 脚本功能是 Reids在 2.6 版本的最大亮点，通过内嵌对 Lua 环境的支持， Redis 解决了长久以来不能高效地处理 CAS （check-and-set）命令的缺点， 并且可以通过组合使用多个命令， 轻松实现以前很难实现或者不能高效实现的模式。

**Lua脚本类似Redis事务，有一定的原子性，不会被其他命令插队，可以完成一些Redis事务性的操作。**这点是关键。

知道原理了，我们就写一个脚本把判断库存扣减库存的操作都写在一个脚本丢给Redis去做，那到0了后面的都Return False了是吧，一旦失败了你修改一个开关，直接挡住所有的请求，然后再做后面的事情嘛。

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。

削峰填谷

一说到这个名词，很多小伙伴就知道了，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。

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！

你可以把它放到消息队列，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。

总结

