链滴

秒杀系统解决方案

作者：gentoo666
原文链接：https://ld246.com/article/1513577841972
来源网站：链滴

我看了二十篇左右的秒杀系统设计及解决方案的文章，从架构、产品、前端、后端四个层面分别结了一些解决方案。
<h3 id="要点总结-">要点总结：</h3>
1.架构：扩容，业务分离，数据分离
2.产品：下单按钮控制，秒杀答题削峰，简化页面设计
3.前端：限流（反作弊） 静态化
4.后端：内存 队列
<h4 id="一-秒杀一般会带来2个问题-">一、秒杀一般会带来 2 个问题：</h4>
<h5 id="1-高并发">1、高并发</h5>
比较火热的秒杀在线人数都是 10w 起的，如此之高的在线人数对于网站架构从前到后都是一种验。
<h5 id="2-超卖">2、超卖</h5>
任何商品都会有数量上限，如何避免成功下订单买到商品的人数不超过商品数量的上限，这是每抢购活动都要面临的难题。
实际上超卖问题是高并发带来的一个子问题，但是因为这个问题太过致命，所以我们把他的解决案单独拿出来说。
<h3 id="二-如何解决-">二、如何解决？</h3>
<h4 id="1-架构层面-">1.架构层面：</h4>
<h5 id="秒杀架构设计原则-">秒杀架构设计原则：</h5>

尽量将请求拦截在系统上游

读多写少的常用多使用缓存

<h5 id="扩容">扩容</h5>
说白了加机器
<h5 id="系统隔离">系统隔离</h5>
为了避免短时间内的大访问量对现有网站业务造成的冲击，可以将秒杀系统独立部署。系统隔离多是运行时的隔离，可以通过分组部署的方式和另外 99% 分开。秒杀还申请了单独的域名，目的也让请求落到不同的集群中。即使秒杀系统崩溃了，也不会对网站造成影响。
<h5 id="数据隔离">数据隔离</h5>
将即将被秒杀的热数据维护到 redis。秒杀所调用的数据大部分都是热数据，比如会启用单独 cache 集群或 MySQL 数据库来放热点数据，目前也是不想 0.01% 的数据影响另外 99.99%。
<h6 id="减库存操作">减库存操作</h6>
<h6 id="一种是拍下减库存-另外一种是付款减库存-目前采用的-拍下减库存-的方式-拍下就是一瞬的事-对用户体验会好些-">一种是拍下减库存 另外一种是付款减库存；目前采用的“拍下减库存”的式，拍下就是一瞬间的事，对用户体验会好些。</h6>
<h4 id="2-产品层面-">2.产品层面：</h4>
<h5 id="1-控制秒杀商品页面抢购按钮的可用-禁用-">1.控制秒杀商品页面抢购按钮的可用/禁用。<h5>
购买按钮只有在秒杀开始的时候才能点亮，在此之前是灰色的，显示活动未开始。
<h5 id="2-增加了秒杀答题-基于时间分片削峰">2.增加了秒杀答题，基于时间分片削峰</h5>
秒杀答题一个很重要的目的是为了防止秒杀器。还有一个重要的功能，就是把峰值的下单请求给长了，从以前的 1s 之内延长到 2~10s 左右，请求峰值基于时间分片了，这个时间的分片对服务端处并发非常重要，会减轻很大压力，另外由于请求的先后，靠后的请求自然也没有库存了，也根本到不最后的下单步骤，所以真正的并发写就非常有限了。其实这种设计思路目前也非常普遍，如支付宝的咻一咻”已及微信的摇一摇。
<h5 id="3-秒杀页面设计简化-">3.秒杀页面设计简化：</h5>
秒杀场景业务需求与一般购物不同，用户更在意的是能够抢到商品而不是用户体验。所以秒杀商页面应尽可能简单并且拍下后地址等个人信息应该使用默认信息，减轻秒杀进行时系统负载，若有更可以在秒杀结束后进行更改。

<h4 id="3-前端层面">3.前端层面</h4>
<h5 id="静态化以及页面缓存">静态化以及页面缓存</h5>
将页面能够静态的部分都静态化，并将静态页面缓存于 CDN，以及反向代理服务器，可能还要时租借服务器。 利用 页面静态化、数据静态化，反向代理 等方法可以避免 带宽和 sql 压力 ，但是之而来一个问题，页面抢单按钮也不会刷新了，可以把 js 文件单独放在 js 服务器上，由另外一台服器写 定时任务 来控制 js 推送。 另外还有一个问题，js 文件会被大部分浏览器缓存，我们可以使用 xxx js?v=随机数 的方式来避免 js 被缓存。
<h5 id="限流-反作弊-">限流（反作弊）</h5>
1.针对同一个用户 id 来实现，前端 js 控制一个客户端几秒之内只能发送同一个请求，后端校验一个 uid 在几秒之内返回同一个页面
2.针对同一个 ip 来实现，进行 ip 检测，同一个 ip 几秒之内不发送请求或者只返回同一个页面</>
3.针对多用户多 ip 来实现，依靠数据分析
4.为了避免用户直接访问下单页面 URL，需要将改 URL 动态化，即使秒杀系统的开发者也无法秒杀开始前访问下单页面的 URL。办法是在下单页面 URL 加入由服务器端生成的随机数作为参数，秒杀开始的时候才能得到。
<h4 id="4-后端层面-">4.后端层面：</h4>
<h5 id="1-加入缓存redis-">1.加入缓存 redis：</h5>
因为秒杀是典型的读多写少的场景，适合操作内存而非操作硬盘；缓存工具 redis 本身的操作是证原子性的，所以可以保证请求了 redis 的写的操作的线程安全性。
<h5 id="2-加入消息队列-利用队列进行削峰-">2.加入消息队列，利用队列进行削峰：</h5>
将用户请求放置于一个或多个队列中，队列中元素总和等于该商品库存总和，未进入队列的请求失败。利用多线程轮询分别从一个或多个队列中取出用户请求。操作 redis 进行减库存操作，成功减存之后返回成功，并将用户信息与商品信息存入另一个队列当中，进行生成订单的操作。利用两个队异步处理业务减轻秒杀高峰时期服务器负载。
<h5 id="3-程序计数器-">3.程序计数器：</h5>
队列与缓存为了保证请求 redis 的次数不超过总的库存量，利用一个程序计数器来这一点。程序数器用 JUC 包下原子类可以实现。
<h5 id="4-分布式锁">4.分布式锁</h5>
分布式情况下可以利用分布式锁来解决任务每次只能由一次服务来执行且不能重复执行。 分布锁的实现：zk、redis 分布式锁的优化：先考虑是否可以去锁，然后考虑尽可能多用乐观锁，少用悲锁。这里有一个问题，乐观锁如果每一次都会有并发冲突的话性能反而不如悲观锁，那么难道真的多乐观锁性能会比悲观锁高吗？选举考虑 ha，比如心跳检测。
<h5 id="5-分布式去锁-方案">5.分布式去锁 方案</h5>
利用集群并发加入队列，选举队列处理服务单点执行，这样可以保证并发实现和加锁一样的并发但不会影响性能。