著作权归https://pdai.tech所有。 链接：<https://www.nenggz.com/md/arch/arch-example-seckill.html>

# 秒杀小结

**秒杀特点：**

短时间内，大量用户涌入，集中读和写有限的库存。

**解决方案：**

层层拦截，将请求尽量拦截在系统上游，避免将锁冲落到数据库上。

* 第一层：客户端优化

产品层面，用户点击“查询”或者“购票”后，按钮置灰，禁止用户重复提交请求； JS层面，限制用户在x秒之内只能提交一次请求，比如微信摇一摇抢红包。 基本可以拦截80%的请求。

* 第二层：站点层面的请求拦截（nginx层，写流控模块）

怎么防止程序员写for循环调用，有去重依据么? IP? cookie-id? …想复杂了，这类业务都需要登录，用uid即可。在站点层面，对uid进行请求计数和去重，甚至不需要统一存储计数，直接站点层内存存储（这样计数会不准，但最简单，比如guava本地缓存）。一个uid，5秒只准透过1个请求，这样又能拦住99%的for循环请求。 对于5s内的无效请求，统一返回错误提示或错误页面。

这个方式拦住了写for循环发HTTP请求的程序员，有些高端程序员（黑客）控制了10w个肉鸡，手里有10w个uid，同时发请求（先不考虑实名制的问题，小米抢手机不需要实名制），这下怎么办，站点层按照uid限流拦不住了。

* 第三层：服务层拦截

方案一：写请求放到队列中，每次只透有限的写请求到数据层，如果成功了再放下一批，直到库存不够，队列里的写请求全部返回“已售完”。

方案二：或采用漏斗机制，只放一倍的流量进来，多余的返回“已售完”，把写压力转换成读压力。 读请求，用cache，redis单机可以抗10W QPS,用异步线程定时更新缓存里的库存值。

还有提示“模糊化”，比如火车余票查询，票剩了58张，还是26张，你真的关注么，其实我们只关心有票和无票。

* 第四层：数据库层

浏览器拦截了80%，站点层拦截了99.9%并做了页面缓存，服务层又做了写请求队列与数据缓存，每次透到数据库层的请求都是可控的。 db基本就没什么压力了，通过自身锁机制来控制，避免出现超卖。

**总结：**

1. 尽量将请求拦截在系统上游（越上游越好）；
2. 读多写少的多使用缓存（缓存抗读压力）；

其他要点：

1. 热点隔离

* 业务隔离。把秒杀做成一种营销活动，卖家要参加秒杀这种营销活动需要单独报名，从技术上来说，卖家报名后对我们来说就是已知热点，当真正开始时我们可以提前做好预热。
* 系统隔离。系统隔离更多是运行时的隔离，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，目的也是让请求落到不同的集群中。
* 数据隔离。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%。

1. 动静分离
2. 基于时间分片削峰
3. 数据分层校验

**参考资料：**

**秒杀业务架构优化之路 沈剑**

<http://www.infoq.com/cn/articles/flash-deal-architecture-optimization>

秒杀系统架构分析与实战

https://www.jianshu.com/p/df4fbecb1a4b

<https://my.oschina.net/xianggao/blog/524943>

淘宝大秒系统设计详解

https://www.cnblogs.com/jifeng/p/5264268.html

https://blog.csdn.net/Luomingkui1109/article/details/77432192