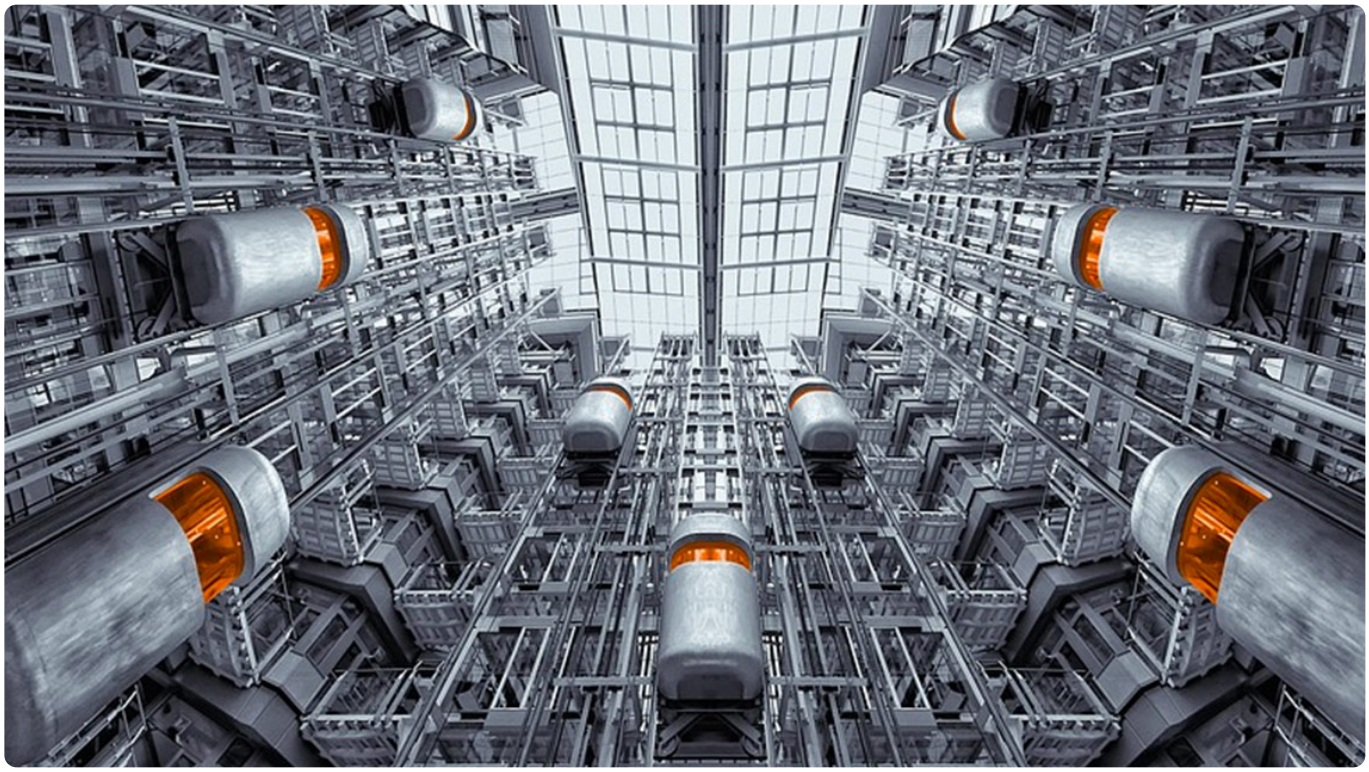
01 | 设计秒杀系统时应该注意的5个架构原则

2018-10-01 许令波

如何设计一个秒杀系统

进入课程 >



讲述：秣明
时长 12:08　大小 5.56M

说起秒杀，我想你肯定不陌生，这两年，从双十一购物到春节抢红包，再到 12306 抢火车票，“秒杀”的场景处处可见。简单来说，秒杀就是在同一个时刻有大量的请求争抢购买同一个商品并完成交易的过程，用技术的行话来说就是大量的并发读和并发写。

不管是哪一门语言，并发都是程序员们最为头疼的部分。同样，对于一个软件而言也是这样，你可以很快增删改查做出一个秒杀系统，但是要让它支持高并发访问就没那么容易了。比如说，如何让系统面对百万级的请求流量不出故障？如何保证高并发情况下数据的一致性写？完全靠堆服务器来解决吗？这显然不是最好的解决方案。

在我看来，**秒杀系统本质上就是一个满足大并发、高性能和高可用的分布式系统**。今天，我们就来聊聊，如何在满足一个良好架构的分布式系统基础上，针对秒杀这种业务做到极致的性能改进。

架构原则：“4 要 1 不要”

如果你是一个架构师，你首先要勾勒出一个轮廓，想一想如何构建一个超大流量并发读写、高性能，以及高可用的系统，这其中有哪些要素需要考虑。我把这些要素总结为“4 要 1 不要”。

1. 数据要尽量少

所谓“数据要尽量少”，首先是指用户请求的数据能少就少。请求的数据包括上传给系统的数据和系统返回给用户的数据（通常就是网页）。

为啥“数据要尽量少”呢？因为首先这些数据在网络上传输需要时间，其次不管是请求数据还是返回数据都需要服务器做处理，而服务器在写网络时通常都要做压缩和字符编码，这些都非常消耗 CPU，所以减少传输的数据量可以显著减少 CPU 的使用。例如，我们可以简化秒杀页面的大小，去掉不必要的页面装修效果，等等。

其次，“数据要尽量少”还要求系统依赖的数据能少就少，包括系统完成某些业务逻辑需要读取和保存的数据，这些数据一般是和后台服务以及数据库打交道的。调用其他服务会涉及数据的序列化和反序列化，而这也是 CPU 的一大杀手，同样也会增加延时。而且，数据库本身也容易成为一个瓶颈，所以和数据库打交道越少越好，数据越简单、越小则越好。

2. 请求数要尽量少

用户请求的页面返回后，浏览器渲染这个页面还要包含其他的额外请求，比如说，这个页面依赖的 CSS/JavaScript、图片，以及 Ajax 请求等等都定义为“额外请求”，这些额外请求应该尽量少。因为浏览器每发出一个请求都多少会有一些消耗，例如建立连接要做三次握手，有的时候有页面依赖或者连接数限制，一些请求（例如 JavaScript）还需要串行加载等。另外，如果不同请求的域名不一样的话，还涉及这些域名的 DNS 解析，可能会耗时更久。所以你要记住的是，减少请求数可以显著减少以上这些因素导致的资源消耗。

例如，减少请求数最常用的一个实践就是合并 CSS 和 JavaScript 文件，把多个 JavaScript 文件合并成一个文件，在 URL 中用逗号隔开（https://g.xxx.com/tm/xx-b/4.0.94/mods/??module-preview/index.xtpl.js,module-jhs/index.xtpl.js,module-focus/index.xtpl.js）。这种方式在服务端仍然是单个文件各自存放，只是服务端会有一个组件解析这个 URL，然后动态把这些文件合并起来一起返回。

3. 路径要尽量短

所谓“路径”，就是用户发出请求到返回数据这个过程中，需求经过的中间的节点数。

通常，这些节点可以表示为一个系统或者一个新的 Socket 连接（比如代理服务器只是创建一个新的 Socket 连接来转发请求）。每经过一个节点，一般都会产生一个新的 Socket 连接。

然而，每增加一个连接都会增加新的不确定性。从概率统计上来说，假如一次请求经过 5 个节点，每个节点的可用性是 99.9% 的话，那么整个请求的可用性是：99.9% 的 5 次方，约等于 99.5%。

所以缩短请求路径不仅可以增加可用性，同样可以有效提升性能（减少中间节点可以减少数据的序列化与反序列化），并减少延时（可以减少网络传输耗时）。

要缩短访问路径有一种办法，就是多个相互强依赖的应用合并部署在一起，把远程过程调用（RPC）变成 JVM 内部之间的方法调用。在《大型网站技术架构演进与性能优化》一书中，我也有一章介绍了这种技术的详细实现。

4. 依赖要尽量少

所谓依赖，指的是要完成一次用户请求必须依赖的系统或者服务，这里的依赖指的是强依赖。

举个例子，比如说你要展示秒杀页面，而这个页面必须强依赖商品信息、用户信息，还有其他如优惠券、成交列表等这些对秒杀不是非要不可的信息（弱依赖），这些弱依赖在紧急情况下就可以去掉。

要减少依赖，我们可以给系统进行分级，比如 0 级系统、1 级系统、2 级系统、3 级系统，0 级系统如果是最重要的系统，那么 0 级系统强依赖的系统也同样是最重要的系统，以此类推。

注意，0 级系统要尽量减少对 1 级系统的强依赖，防止重要的系统被不重要的系统拖垮。例如支付系统是 0 级系统，而优惠券是 1 级系统的话，在极端情况下可以把优惠券给降级，防止支付系统被优惠券这个 1 级系统给拖垮。

5. 不要有单点

系统中的单点可以说是系统架构上的一个大忌，因为单点意味着没有备份，风险不可控，我们设计分布式系统最重要的原则就是“消除单点”。

那如何避免单点呢？我认为关键点是避免将服务的状态和机器绑定，即把服务无状态化，这样服务就可以在机器中随意移动。

如何那把服务的状态和机器解耦呢？这里也有很多实现方式。例如把和机器相关的配置动态化，这些参数可以通过配置中心来动态推送，在服务启动时动态拉取下来，我们在这些配置中心设置一些规则来方便地改变这些映射关系。

应用无状态化是有效避免单点的一种方式，但是像存储服务本身很难无状态化，因为数据要存储在磁盘上，本身就要和机器绑定，那么这种场景一般要通过冗余多个备份的方式来解决单点问题。

前面介绍了这些设计上的一些原则，但是你有没有发现，我一直说的是“尽量”而不是“绝对”？

我想你肯定会问是不是请求最少就一定最好，我的答案是“不一定”。我们曾经把有些 CSS 内联进页面里，这样做可以减少依赖一个 CSS 的请求从而加快首页的渲染，但是同样也增大了页面的大小，又不符合“数据要尽量少”的原则，这种情况下我们为了提升首屏的渲染速度，只把首屏的 HTML 依赖的 CSS 内联进来，其他 CSS 仍然放到文件中作为依赖加载，尽量实现首屏的打开速度与整个页面加载性能的平衡。

所以说，**架构是一种平衡的艺术，而最好的架构一旦脱离了它所适应的场景，一切都将是空谈。**我希望你记住的是，这里所说的几点都只是一个个方向，你应该尽量往这些方向上去努力，但也要考虑平衡其他因素。

不同场景下的不同架构案例

前面我说了一些架构上的原则，那么针对“秒杀”这个场景，怎样才是一个好的架构呢？下面我以淘宝早期秒杀系统架构的演进为主线，来帮你梳理不同的请求体量下，我认为的最佳秒杀系统架构。

如果你想快速搭建一个简单的秒杀系统，只需要把你的商品购买页面增加一个“定时上架”功能，仅在秒杀开始时才让用户看到购买按钮，当商品的库存卖完了也就结束了。这就是当时第一个版本的秒杀系统实现方式。

但随着请求量的加大（比如从 1w/s 到了 10w/s 的量级），这个简单的架构很快就遇到了瓶颈，因此需要做架构改造来提升系统性能。这些架构改造包括：

1. 把秒杀系统独立出来单独打造一个系统，这样可以有针对性地做优化，例如这个独立出来的系统就减少了店铺装修的功能，减少了页面的复杂度；
2. 在系统部署上也独立做一个机器集群，这样秒杀的大流量就不会影响到正常的商品购买集群的机器负载；
3. 将热点数据（如库存数据）单独放到一个缓存系统中，以提高“读性能”；
4. 增加秒杀答题，防止有秒杀器抢单。

此时的系统架构变成了下图这个样子。最重要的就是，秒杀详情成为了一个独立的新系统，另外核心的一些数据放到了缓存（Cache）中，其他的关联系统也都以独立集群的方式进行部署。

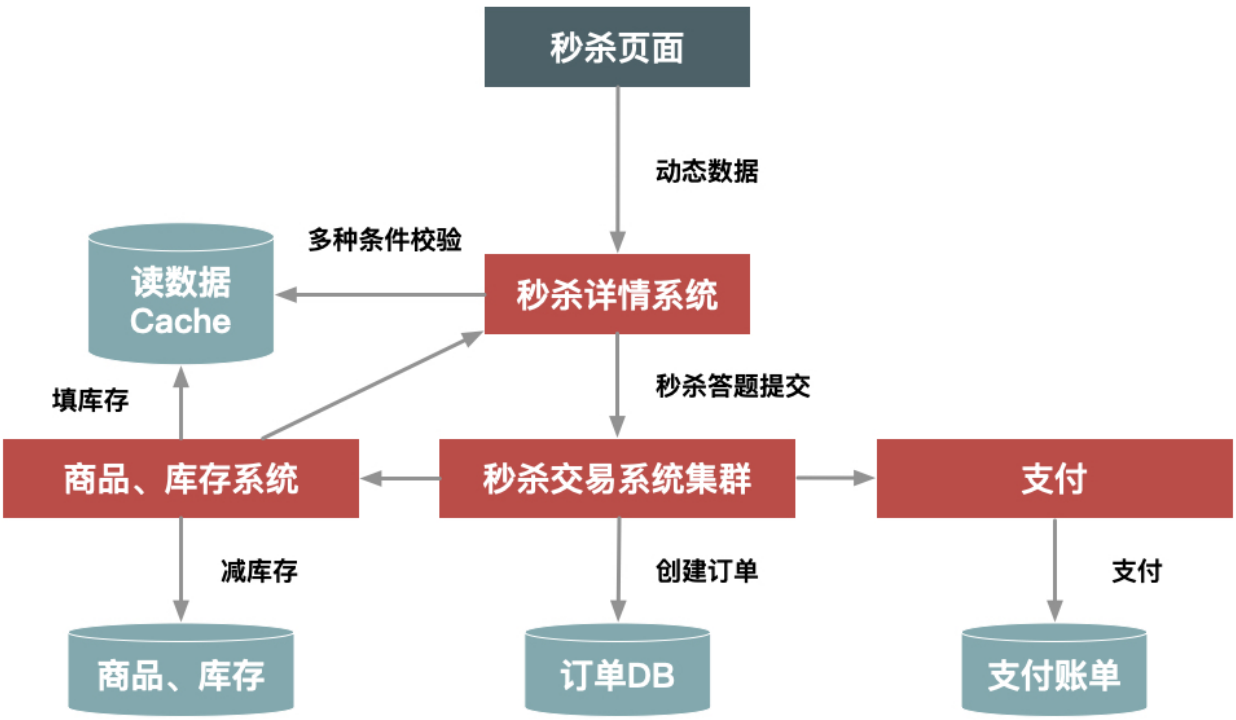


图 1 改造后的系统架构

然而这个架构仍然支持不了超过 100w/s 的请求量，所以为了进一步提升秒杀系统的性能，我们又对架构做进一步升级，比如：

1. 对页面进行彻底的动静分离，使得用户秒杀时不需要刷新整个页面，而只需要点击抢宝按钮，借此把页面刷新的数据降到最少；
2. 在服务端对秒杀商品进行本地缓存，不需要再调用依赖系统的后台服务获取数据，甚至不需要去公共的缓存集群中查询数据，这样不仅可以减少系统调用，而且能够避免压垮公共缓存集群。
3. 增加系统限流保护，防止最坏情况发生。

经过这些优化，系统架构变成了下图中的样子。在这里，我们对页面进行了进一步的静态化，秒杀过程中不需要刷新整个页面，而只需要向服务端请求很少的动态数据。而且，最关键的详情和交易系统都增加了本地缓存，来提前缓存秒杀商品的信息，热点数据库也做了独立部署，等等。

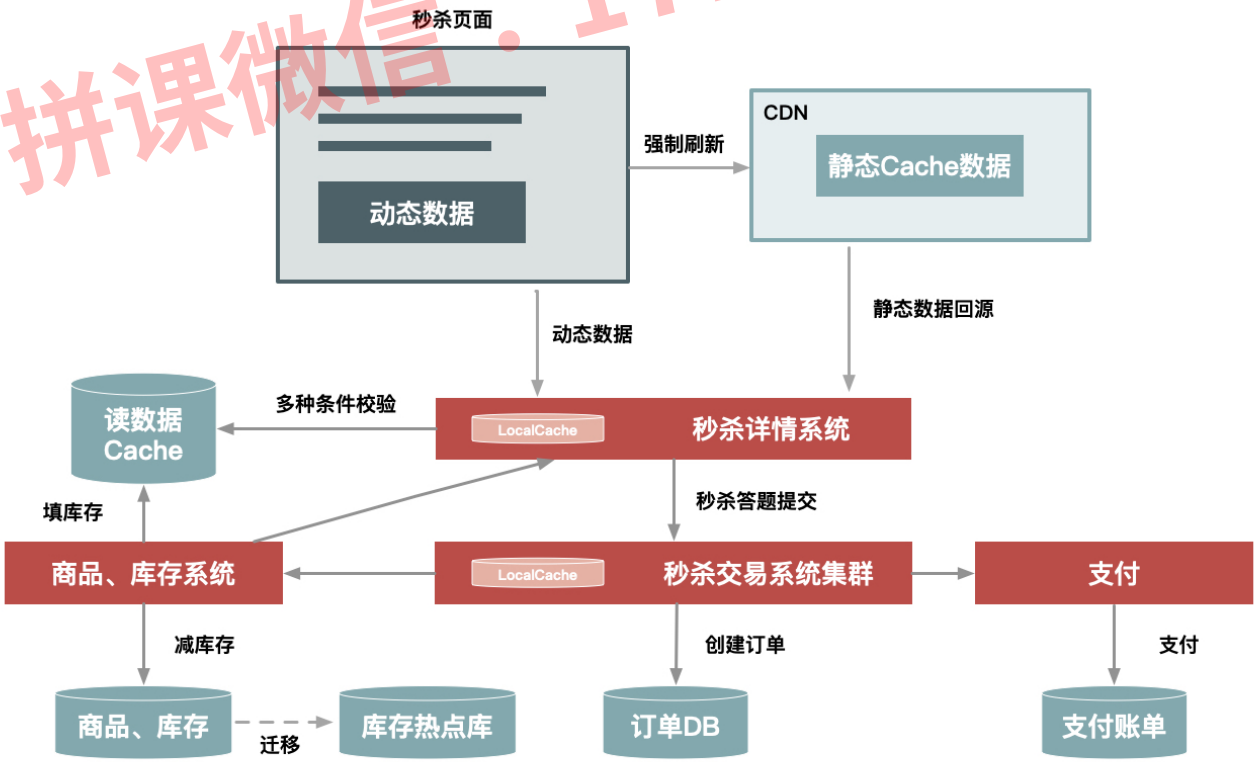


图 2 进一步改造后的系统架构

从前面的几次升级来看，其实越到后面需要定制的地方越多，也就是越“不通用”。例如，把秒杀商品缓存在每台机器的内存中，这种方式显然不适合太多的商品同时进行秒杀的情况，因为单机的内存始终有限。所以要取得极致的性能，就要在其他地方（比如，通用性、易用性、成本等方面）有所牺牲。

总结

来让我们回顾下前面的内容，我首先介绍了构建大并发、高性能、高可用系统中几种通用的优化思路，并抽象总结为“4 要 1 不要”原则，也就是：数据要尽量少、请求数要尽量少、路径要尽量短、依赖要尽量少，以及不要有单点。当然，这几点是你要努力的方向，具体操作时还是要密切结合实际的场景和具体条件来进行。

然后，我给出了实际构建秒杀系统时，根据不同级别的流量，由简单到复杂打造的几种系统架构，希望能供你参考。当然，这里面我没有说具体的解决方案，比如缓存用什么、页面静态化用什么，因为这些对于架构来说并不重要，作为架构师，你应该时刻提醒自己主线是什么。

说了这么多，总体上我希望给你一个方向，就是想构建大并发、高性能、高可用的系统应该从哪几个方向上去努力，然后在不同性能要求的情况下系统架构应该从哪几个方面去做取舍。同时你也要明白，越追求极致性能，系统定制开发就会越多，同时系统的通用性也就会越差。

最后，欢迎你在评论区和我分享你在设计秒杀系统时的一些经验和思考，你的经验对我们这个专栏来说也很重要。

上一篇 开篇词 | 秒杀系统架构设计都有哪些关键点？

下一篇 02 | 如何才能做好动静分离？有哪些方案可选？

精选留言 (77)

写留言

黄金的太阳
2018-10-01

179

希望老师可以分别从1W QPS，10W QPS ，100WQPS在架构升级前遇到的性能瓶颈做为讲解入口点，为什么这样设计之后就能解决问题的方式，为什么切分点是1万，10万和100万，瓶颈的分析方式等等，感觉效果更好，否则看到一堆架构，但并不清楚为什么要这样做还是很难平移到自己的系统设计中，一点拙见，希望老师能够解惑

展开

作者回复: 架构升级的逻辑要具体问题具体分析的
例如秒杀的场景来说，不同QPS量级下瓶颈也会不一样，10w级别可能瓶颈就在数据读取上，通过增加缓存一般就能解决，如果要到100w那么，可能服务端的网络可能都是瓶颈，所以要把大部分的静态数据放到cdn上甚至缓存在浏览器里

所以要做架构升级，还是主要要分析在预估的QPS下，整个系统的瓶颈会在什么地方，要针对这起瓶颈来重新设计架构方案

Mr.钧
2018-10-02

51

高并发系统的几大方向
1.请求数据尽量少，从而减少cpu消耗
2.访问路径尽量短，减少节点消耗
3.强依赖尽量少，减少加载时间
4.不要有单点，要有备份...
展开

作者回复: 👍

Edward
2018-10-01

22

数据缓存在机器内存中的话，集群内如何实现多台机器数据一致性？

作者回复: 在内存的数据是静态数据，不会更新，没有一致性问题

小喵喵
2018-10-01

17

1 .本地cache用什么实现好呢？
2. 通过什么方式往本地cache 写数据呢？
3. 秒杀系统的及时性非常高，把库存写进cache ，怎么及时更新呢？

作者回复: 1.本地cache一般就是用内存实现，用java集合类型就行
2.用订阅的方式，在初始化时加载到内存
3.有两种方法，一是定时更新取3秒，二是，主动更新，数据库字段更新后发消息更新缓存，这个需要用到一个组件阿里叫metaq就是就是数据库字段更新会产生一条消息。另外cache里库存不需要100%和数据库一致，这个在后面的文章也有介绍

王永旺
2018-10-01

11

单点的概念是啥意思，不太明白

展开

作者回复: 单点就是没有备份，挂了系统没法正常服务了

王虹凯
2018-10-02

8

能不能给一些你认为的关键知识点在文章最后加一些对这些点的外链。这样可以更详细，当然更具系统性。 极客时间也可以考虑这样的一个功能，整个当前专栏共用这些链接。

当然，作者不加这些也没问题，只是觉得那样是不是更权威些。不过要求作者太多了！

展开

作者回复: 你想了解那块内容，我可以单独发给你😊

酱了个油
2018-10-09

7

库存不会放在localcache，localcache只放静态数据。
库存是放在独立的缓存系统里，如redis，库存是采用主动失效的方式来失效缓存

展开

作者回复: 👍

1024
2018-11-25

6

阅读5分钟，留言中回复看了半个多小时；感觉从读者讨论中收益更大，许神加油！

作者回复: 😊

李海凡
2018-10-05

6

想了解系统性能和架构升级背后的逻辑是什么，这样设计系统解决了具体什么瓶颈，例如您举的例子中，qps只能到10w、为什么只能到10w,瓶颈在什么地方，如何分析，如何解决

作者回复: 架构升级的逻辑要具体问题具体分析的

例如秒杀的场景来说，不同QPS量级下瓶颈也会不一样，10w级别可能瓶颈就在数据读取上，通过增加缓存一般就能解决，如果要到100w那么，可能服务端的网络可能都是瓶颈，所以要把大部分的静态数据放到cdn上甚至缓存在浏览器里

所以要做架构升级，还是主要要分析在预估的QPS下，整个系统的瓶颈会在什么地方，要针对这起瓶颈来重新设计架构方案

olaf
2018-10-01

6

请问一下减库存的结果分别是如何更新到公共缓存和本地缓存的呢？

作者回复: 缓存的更新一般都是没命中时再填充更新

没命中的原因一种是主动失效，一种是定时失效

公用缓存主动失效较多，本地缓存，定时失效更常用

Lugyedo
2018-10-01

5

缓存中的热点数据在高并发情况下如何保证一致

展开

作者回复: 减库存一章会有介绍

潘政宇
2018-10-01

5

每经过一个节点，一般都会产生一个新的 Socket 连接。什么意思啊，不是一个tcp连接就有一个socket吗，从客户端到server就是一个tcp连接啊？

展开

作者回复: 就是每经过一个节点会产生一个新的tcp连接

broken_ope...
2019-02-15

3

老师你好，减库存操作，update count = count-1 where count>0，这种做法不会多卖吧？

作者回复: 😉

Kuer
2018-11-28

3

老师你好，刚才说要设计时候尽量做到4要1不要，1不要就是不要用单点，也就是因为没有数据备份，想问这个单点和单点登录系统的单点有什么关系吗？希望老师有时间可以解答一下。

作者回复: 这里的单点有两层含义
一是本身的系统是单点系统，比如你说的登录系统，所以其他系统需要验证登录状态时都需要调用他，那他就是一个单点，怎么解决？解决的办法就是尽量让这个系统简单，减少更改保持稳定。
另外一种是状态单点或者叫数据单点，也就是状态一旦丢失比较难恢复，这个就比较麻烦，比如保存登录状态的session保存在一台机器内存中，一旦这台机器挂掉，状态丢失就比较难恢复，这个单点就比较严重

不似旧日
2019-02-21

2

什么是秒杀:

简单来说,秒杀就是在同一个时刻有大量的请求争抢购买同一个商品并完成交易的过程,用技术的行话来说就是大量的并发读和并发写。

...

展开

一成 2

2018-11-22

读过作者的《深入分析java技术内幕》,看到作者的专栏直接订阅了

作者回复: 感谢😄

王维 2

2018-10-11

老师好，我觉得图1有个地方有问题，秒杀详情系统需要从cache中读取缓存数据，那么是不是缺少一个从读cache到秒杀详情系统的箭头→？

展开

作者回复: 嗯，看的很仔细，是要从cache读取数据的，只不过用这一个箭头表示了，应该加个说明会更好

Seven Blu... 2

2018-10-09

文中讲的原则在后续给出的架构演变例子中，并没有很好的进行说明，看完之后，并没有对此有深刻体会

作者回复: 😊，可以看看后面的文章，会有更一些详细的介绍

猎户星座 2

2018-10-01

请问秒杀的时间是怎么控制的，比如十点开始，十点十分结束。各个客户端、客户端与服务器、以及集群内的服务器之间。时间如何精确同步？

展开

作者回复: 都是以服务端的时间为准，服务端的时间同步需要依赖一个时间同步组件完成如ntp
当然当前的服务器时间同步还是有一定的时间延时，但是也不是太影响

北祐葉 2

2018-10-01

虽然还没看，但是很期待

展开

作者回复: 😉