17 | 高性能架构案例：如何设计一个秒杀系统?

2020-03-30 王庆友

讲述：王庆友
时长 12:44 大小 11.67M

你好，我是王庆友。

在上一讲中，我和你详细介绍了打造一个高性能系统的应对策略和架构手段。那么今天，我就以 1 号店的秒杀系统为例，来具体说明如何实现一个高性能的系统。

背景和问题

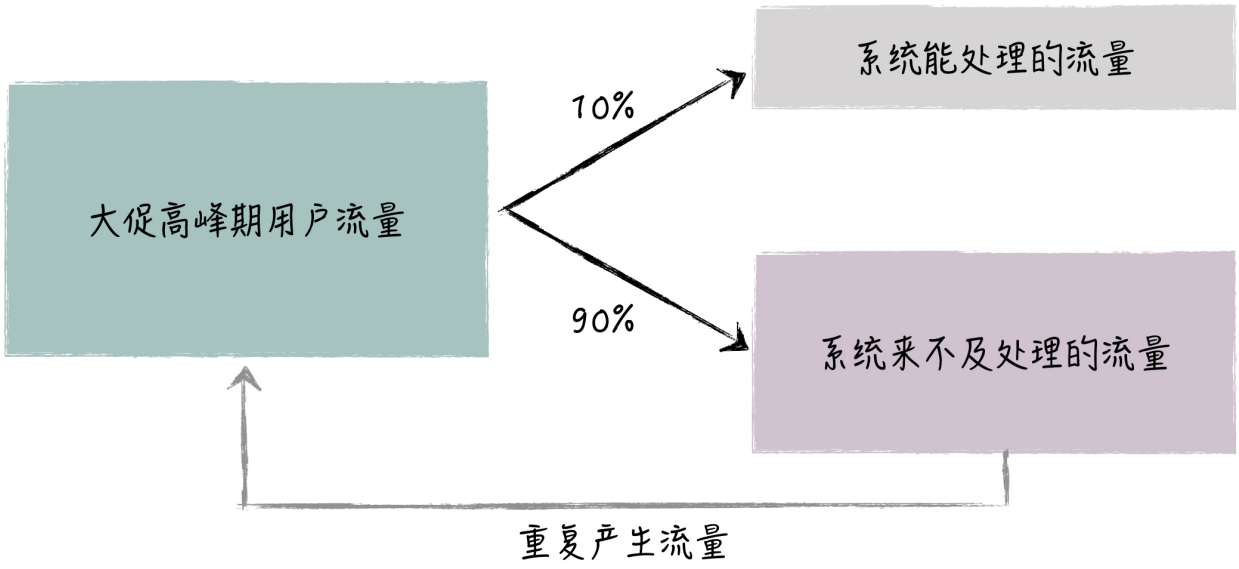
先说下背景。在 2014 年的时候，1 号店作为网上超市类电商，经常在线上举行各种大型的促销活动。比如进口牛奶促销活动，每次促销的牛奶有几十万盒，促销价格非常优惠，一般这样的促销活动会在某个整点的时间进行开卖（如上午 10 点）。对于这种质高价优并且是刚需的

商品，会有大量的用户来抢购，俗话说“手快有，手慢无”，往往短短几分钟内，所有牛奶就能售卖完毕。

这本质上是一种秒杀活动，但商品数量非常大，一瞬间会有大量的用户流量涌入，流量可以高达平时的几十倍。而且和少量商品的秒杀不同，这些都是有效流量，最终会生成订单。

而在正常情况下，系统因为资源有限，只能处理 10% 的流量，无法处理剩下的 90% 流量，瞬间高并发的流量涌入，很大程度上会引起后台系统超时报错，导致用户下单不成功。这样一来，用户就会反复刷新页面，多次尝试下单，不但用户的体验不好，而且系统的压力会更大。

最终的结果就是，系统往往由于过载，整体处理能力下降，甚至瘫痪，导致所有用户都无法购买。就像下图表示的一样，在秒杀场景下，系统会面临这样的困境：



在这种情况下，对于用户来说，能不能买到商品，拼的是体力和人品，由于体验不好，用户会逐渐对活动失去兴趣；而对于系统来说，我们需要拼命地加机器来满足峰值流量。

每次 1 号店要进行大促的时候，在活动开始前，运营和技术人员会坐在一起，大家一起来预估活动的峰值流量，然后技术人员做评估，系统的哪些节点需要加机器，以及要加多少机器。但这样的做法其实存在几个问题：

首先，我们对峰值流量的预估以及要加多少机器都是拍脑袋的，和实际出入往往很大，一旦估计少了，系统同样会面临过载的风险；

其次，为了短暂的几分钟促销，我们需要增加大量的机器，事先要做很多的运维准备工作，不但浪费资源，而且效率很低；

最为关键的是，有些处理节点，系统不是通过加机器就能扩展处理能力的，比如商品库存数据库，下单时，我们需要扣库存，而为了防止库存更新冲突，我们需要锁定库存记录，导致系统的并发处理能力有限，这个问题单靠加机器是解决不了的。

总体方案

对于这种高并发情况，看来让系统单纯地通过加机器去硬扛，是不可行的。**那么，我们有没有更好的办法，既保证用户体验，又保证系统能够轻松地应对流量挑战呢？**

我们先来深入分析下业务场景。这个秒杀活动的特点是，在短期的 1~2 分钟内，用户流量很大，但只要促销的商品卖完，流量马上恢复常态。所以，对于前端短期内这么大的下单请求，后端如果实时处理，压力会非常大，但如果把这个处理时间延长到 10 分钟，后端是可以完成下单的。那对用户来说，商品优惠的力度这么大，他们关心的是能否买到，所以会愿意多等一段时间，而不是在页面上一次次点击下单，每次系统都提示下单失败。

当然，如果我们把订单处理的时间延长了，只要我们在前台告诉用户，系统已经接受了他们的订单，并且不断同步用户订单处理的进度，用户体验的问题其实也不大。

基于这个分析，我们就可以利用**异步处理**的思路来应对秒杀活动。

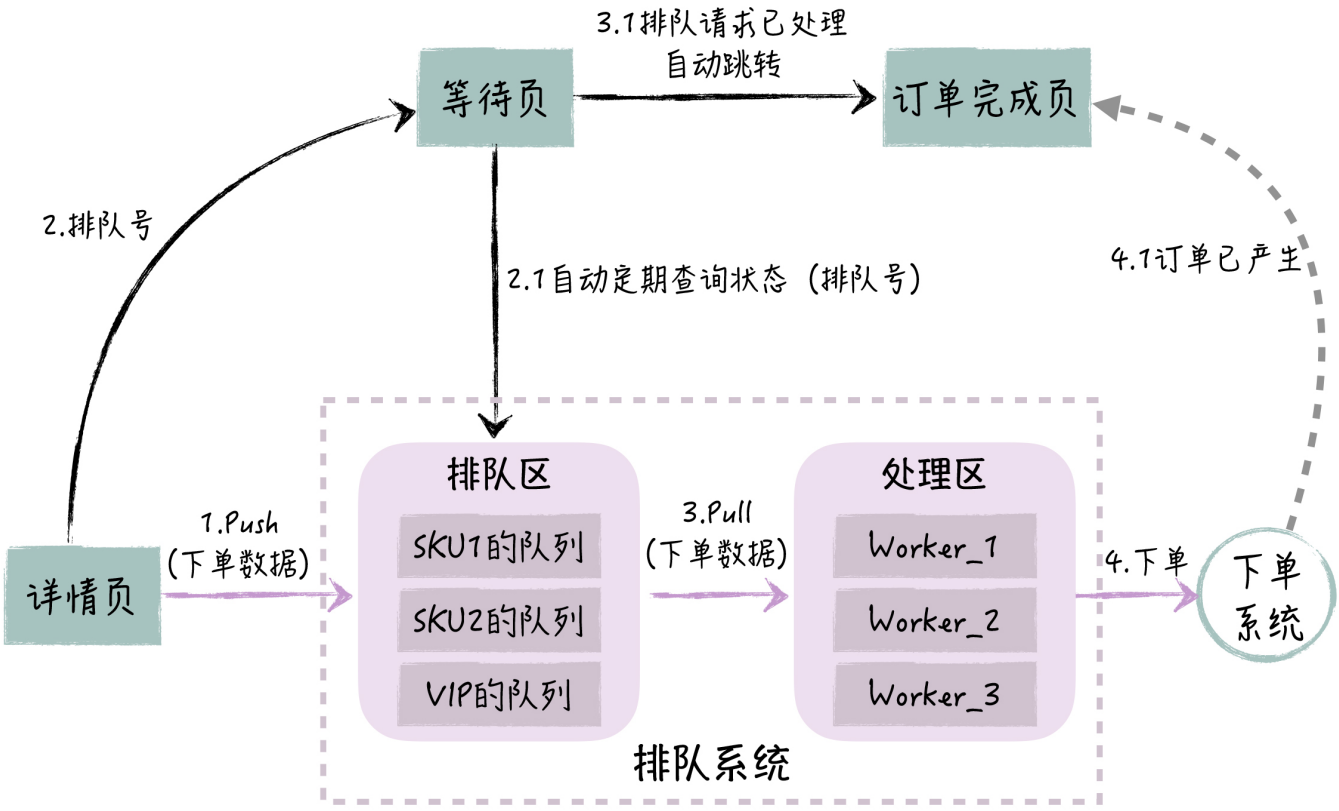
我们先在前端接收用户所有的下单请求，但不在后端实时生成订单，而是放在队列里；然后系统根据后端订单中心的实际处理能力，从队列里获取订单请求，再交给订单中心生成实际的订单。同时，系统告诉用户当前的处理进度，有多少订单排在 TA 的前面，TA 还要等多久。

这样对于用户来说，在前台下单一次就可以了，然后等系统慢慢处理，这也符合先到先得的原则，非常公平合理。对系统来说，只要根据大促的商品总量，一定程度上增强系统处理能力，保证下单请求从进来到最后处理完成，这个时间相对合理就可以了。

比如说，有 20 万件的商品，每人限购一件，预计用户会在 2 分钟内完成下单，但用户能够接受系统在 20 分钟内完成订单处理。这样，系统只要保证每分钟能处理 1 万订单就行；而

如果不采取排队的方式，系统就需要每分钟处理 10 万订单，它的压力就会提升一个数量级。

基于排队的思路，系统总体架构设计如下图所示：



在这个架构中，我们**在前台和后台下单系统之间，新增了排队系统，它包括排队区和处理区两个部分。**系统整体的处理过程是这样的：

1. 用户在商品详情页提交订单后，这个订单会作为预订单进入排队区，同时排队系统会返回用户一个排队编号，这个编号用于跟踪后续的订单处理进度；
2. 用户被引导到一个等待页，这个页面会根据排队号，定时地查询排队系统，排队系统会返回预订单在队列中的位置信息，包括它前面还有多少未处理的预订单，以及后台系统大概还要多久会处理这个预订单，这样用户就不会焦虑；
3. 在排队系统的处理区，有很多消费者，它们依次从排队区的队列里获取预订单，然后调用后台下单系统生成实际的订单；
4. 随着预订单变成正式的订单，队列里的预订单会逐渐变少，如果当前的预订单已经从队列里被移除了，用户的等待页就会检测到这个情况，页面自动跳转到订单完成页，这就和常规的购物流程一样了，用户进行最后的支付，最终完成整个前台下单过程。

这里，你可以看到，**前台**的预订单有瞬时的大流量，但我们只是把它们放到队列里，这个处理起来很快，排队系统可以轻松应对；而**后台**生成实际的订单是匀速的，并且最大化地发挥了下单系统的处理能力。另一方面，对于用户体验来说，用户可以选择在等待页等候，实时获取订单处理进度的反馈，也可以选择离开，然后在用户中心的“待支付订单”里完成支付。通过这样的设计，排队系统既保证了系统处理订单的能力，也保证了用户良好的体验。

下面是一张用户等待页的效果图，你可以直观地了解秒杀系统的用户体验。

现在，你已经了解了秒杀系统的总体设计。接下来，我深入介绍下这个排队系统的内部设计细节，帮助你更好地理解它。

内部设计

首先，**针对队列的技术选型**，排队系统使用的是 **Redis**，而不是 MQ。因为相对于 MQ 来说，Redis 更轻量级，性能更好，它内置了队列数据结构，除了和 MQ 一样支持消息的先进先出以外，我们还可以获取队列的长度，以及通过排队号获取消息在队列中的位置，这样我们就可以给前端反馈预订单的处理进度。

对于秒杀场景来说，一个订单只能包含一个商品，这里我们**为每个秒杀商品提供一个单独的队列**，这样就可以分散数据在 Redis 中的存取，多个队列可以提供更好的性能。

关于队列的调度问题，也就是消费者优先从哪个队列里拿预订单，排队系统会结合下单时间和队列的长度来确定，以保证用户合理的时间体验。比如说，某个秒杀商品的队列很长，消

费者会优先从这个队列拿预订单，从而避免用户等待太长的时间。

关于队列长度，为了保证用户能够买到商品，我们并不是把所有前台的下单请求都会放到队列里，而是根据参与活动的秒杀商品数量，按照 1:1 的比例，设置队列初始长度，这样就保证了进入队列的请求最终都能生成订单。

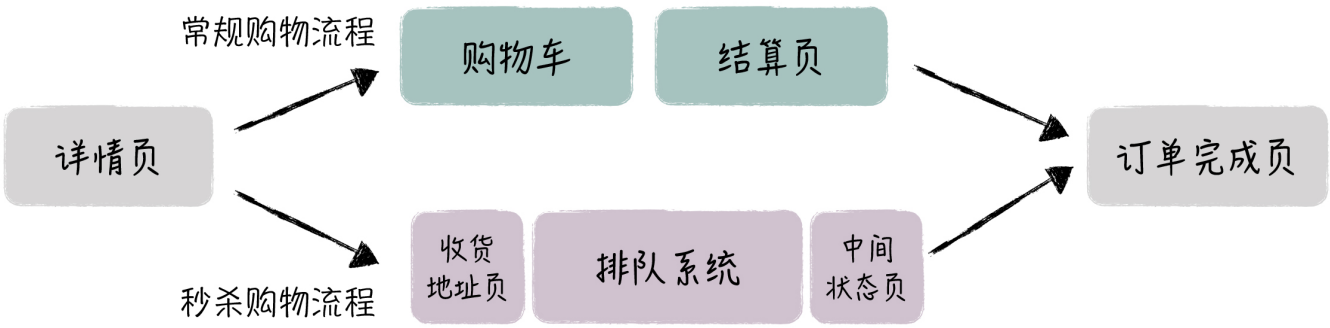
这个可用队列长度会随着预订单进入队列，不断地减少，当数值变为 0 时，下单前台会拒绝接受新请求进入队列，直接反馈用户下单失败。当然，如果后台订单生成异常或用户取消订单后，可用队列长度会增加，前台会重新开放预订单进入队列。

更多优化：建立活动库存

除了秒杀流程，系统还有**常规的购物流程**，这两个购物方式都是从详情页开始，到订单完成页结束。不同的地方是，常规购物流程走的是购物车和结算页，系统是同步处理的，这样可以有更好的用户体验。

在这里，我们在系统设计上，可以很好地同时支持秒杀流程和常规购物流程。

如果运营人员在后台上架商品的时候，设置这是一个秒杀商品，那么从详情页开始，系统就会引导用户走秒杀流程，否则就走常规购物流程。特别是在早期秒杀系统刚落地的时候，如果发现秒杀流程有问题，我们还可以快速切回到常规的购物流程，实现了一定程度上的系统互备。



此外，对于秒杀活动来说，参与活动的商品种类是有限的，但这些商品库存的扣减非常频繁，因此我们建立了**活动库存**的概念，把少量参与促销的商品种类单独放在一个库里，避免和大量常规的商品放在一起，这样也大幅度地提高了库存数据库的读写性能。

好了，通过这个秒杀系统的架构设计，你可以看到，我们巧妙地通过请求的异步化处理，对流量进行削峰，从而保证了系统的高性能。这里我们不需要增加太多的机器，在系统落地时，我们通过排队系统对前后台解耦，后台下单系统基本上也不需要修改，系统整体改造的工作量不大，整个落地过程也非常顺利。

不过值得注意的是，**这种方式比较适合瞬时有高并发流量的场景**，比如这里说的秒杀场景。如果订单高峰会持续一段较长的时间，而用户对订单处理又有比较高的时间要求，那就不适合采用这种异步削峰的方式。

举个例子，外卖订单的午高峰通常会持续两个小时，而用户普遍期待订单半小时能够送达。对于这种情况，我们就需要正面应对高峰流量，比如通过水平扩展各个节点，提升系统的处理能力。这也要求系统能够做到弹性伸缩，高效地支持资源的缩容或扩容，节省成本。

总结

今天，我针对 1 号店的大促业务挑战，与你分享了一个秒杀系统的具体设计，对照我在上一讲中介绍的高性能应对策略，秒杀系统主要使用了**异步化处理**的方式，这也符合实际的业务场景。

通过今天的分享，相信你对如何保障系统的高性能有了更深入的体会，如果你也有类似的瞬时高并发的场景，你也可以在实践中参考这里的做法。

最后，给你留一道思考题： 你的公司业务上有高并发的场景吗，系统是如何应对的呢？

欢迎给我留言，我会及时给你反馈。如果你觉得有收获，也欢迎你把这篇文章分享给你的朋友。感谢你的阅读，我们下期再见。

上一篇　16 | 高性能和可伸缩架构：业务增长，能不能加台机器就搞定?

下一篇　18 | 可伸缩架构案例：数据太多，如何无限扩展你的数据库?

精选留言 (9)

写留言

小祺
2020-03-31

20W用户下单成功，后面的用户提示下单失败，预下单成功的用户全部放弃支付，导致活动结束了但是商品没卖出去，怎么解决?

展开

> 作者回复: 后台下单时，会有异常订单自动审核，比如黄牛下的单，恶意订单等，这部分库存会回到前台可用队列中，只要是正常订单，不会出现大面积不支付的情况

1

Jeff.Smile
2020-03-30

如果有些前置知识会比较好，比如流量洪峰到来的时候对于秒杀这种类型的活动，首先要

把流量接下来，起码不能在网关入口处就被拒绝了，这个时候能用到的技术有哪些，比如nginx或者其他的方式。接下来，流量怎么分发到后端的应用服务器，负载均衡或者是直接操作redis，这里也有一些技术点。
老师整体上讲的比较言简意赅，不过我还是想关注下一些技术点，哪怕一笔带过略微提...
展开

作者回复: 对于接入系统来说，这点流量照常接，没有变化，所以文章里就没提

1

zeor
2020-03-30

老师您好，当正常下单时采用同步方式，但流量也很大，如果按您说的，库存只有一个库，但库已经达到了瓶颈，有什么方式在同步情况下解决这个瓶颈？
展开

作者回复: 可以用redis放库存数据，利用redis的锁保证互斥访问。这样相当于把redis作为前置数据库。
应用直接改redis数据，通过发消息，再异步改db的数据。

1

蓝天
2020-03-31

排队数据存在redis中是不也需要开启持久化呢
展开

作者回复: 很好的思考，要持久化的，这是关键的业务数据。

正在减肥的胖籽。
2020-03-31

老师您好。我有几个问题先请教你下。
1.如果系统有上万个商品，redis一个商品一个队列，然后生成上万个队列？
2.异步化处理逻辑，来削峰流量。但是如果需要同步，库存事实扣的？这种可以讲一下吗？因为数据库是有状态的，数据库并发能力就比较弱，这方面可以讲一下吗？
3.
展开

作者回复: 1.秒杀不会有上万个商品，一般就十几个
2.异步处理，后台下单根据实际能力来，库存正常该怎么扣就怎么扣，一分钟更新1-2万次不是问题

孙同学
2020-03-30

https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3 整理学习更新 又学到一些 自己之前做过一个思想类似的 利用redis做缓存 异步慢慢消费数据


tt
2020-03-30

这节课讲得方法挺适合我们的，我们之前上过一个抢购的活动，由于没有经验，所以跑起来很卡，今天讲的方法不复杂，还能解决问题，真好。

展开


Jxin
2020-03-30

1.秒杀案例讲得很棒。
2.外卖那个场景，其实单纯讲加机器，易伸缩。感觉没说到点子上。这个场景其实是做性能优化，价值会很高的场景。如何通过技术手段和各种权衡，在有限的资源下满足更多的并发。这个话题会比较在点子上。
3.限于篇幅，已经很棒，感谢栏主分享。

展开


阿杜
2020-03-30

讲解的比较精炼，也很实用，能满足大部分秒杀业务，业务落地简单快速。