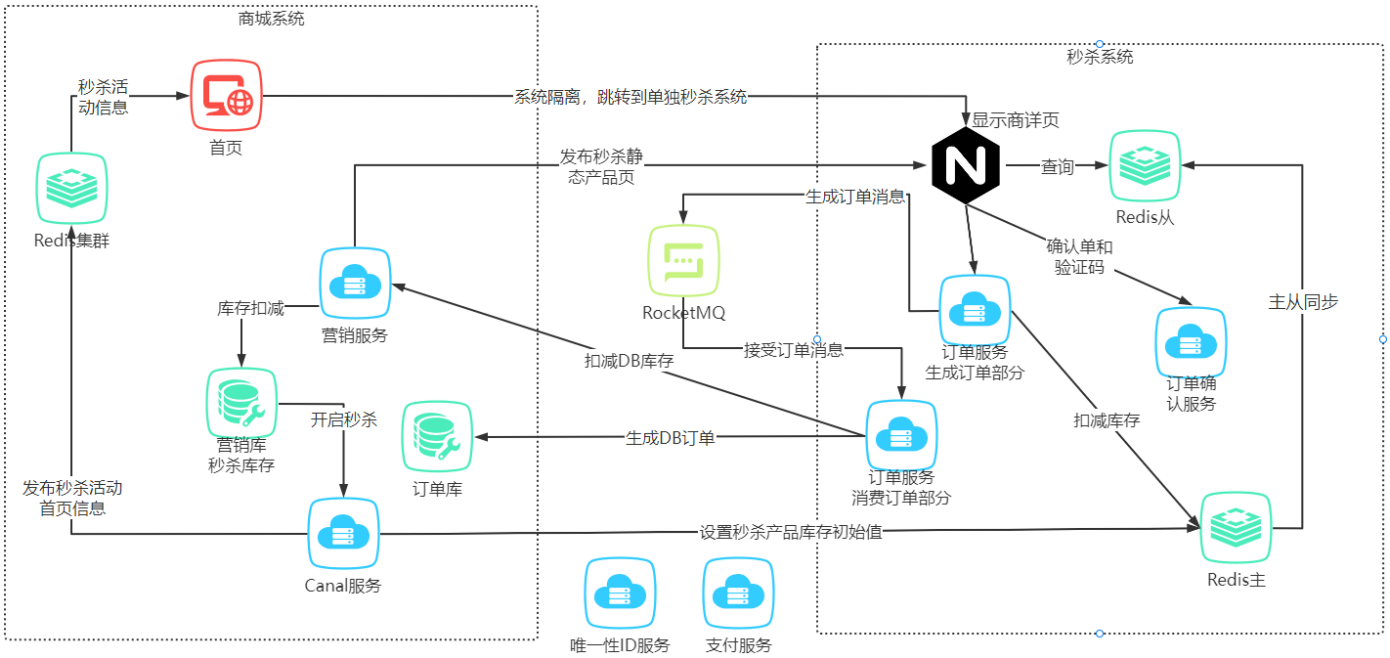
互联网秒杀系统架构升级方案

这一部分主要是结合其他互联网公司的经验，分享一下针对电商项目当前的秒杀系统，还可以做哪些架构层面的优化设计。

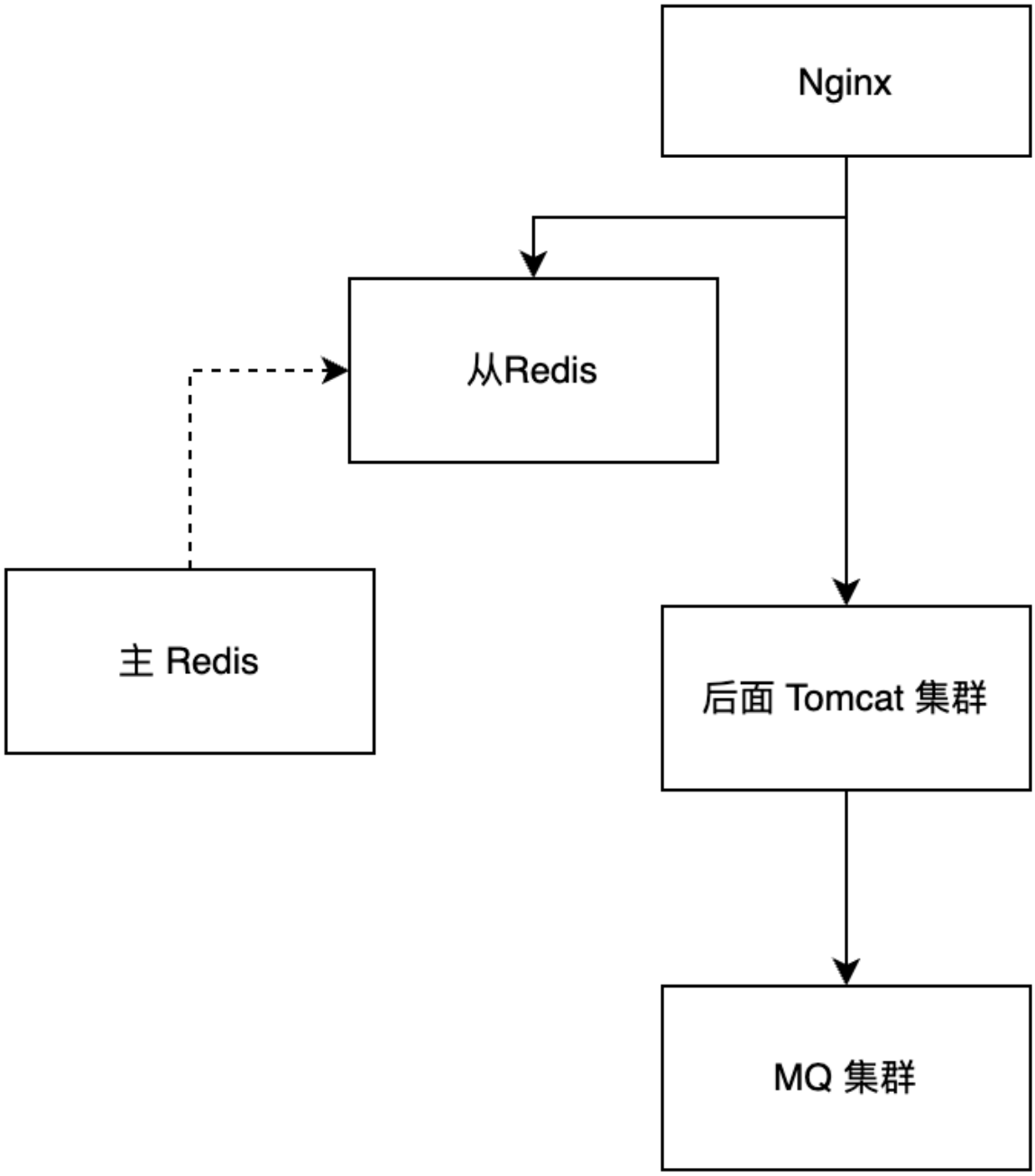
一、秒杀系统当前的架构梳理



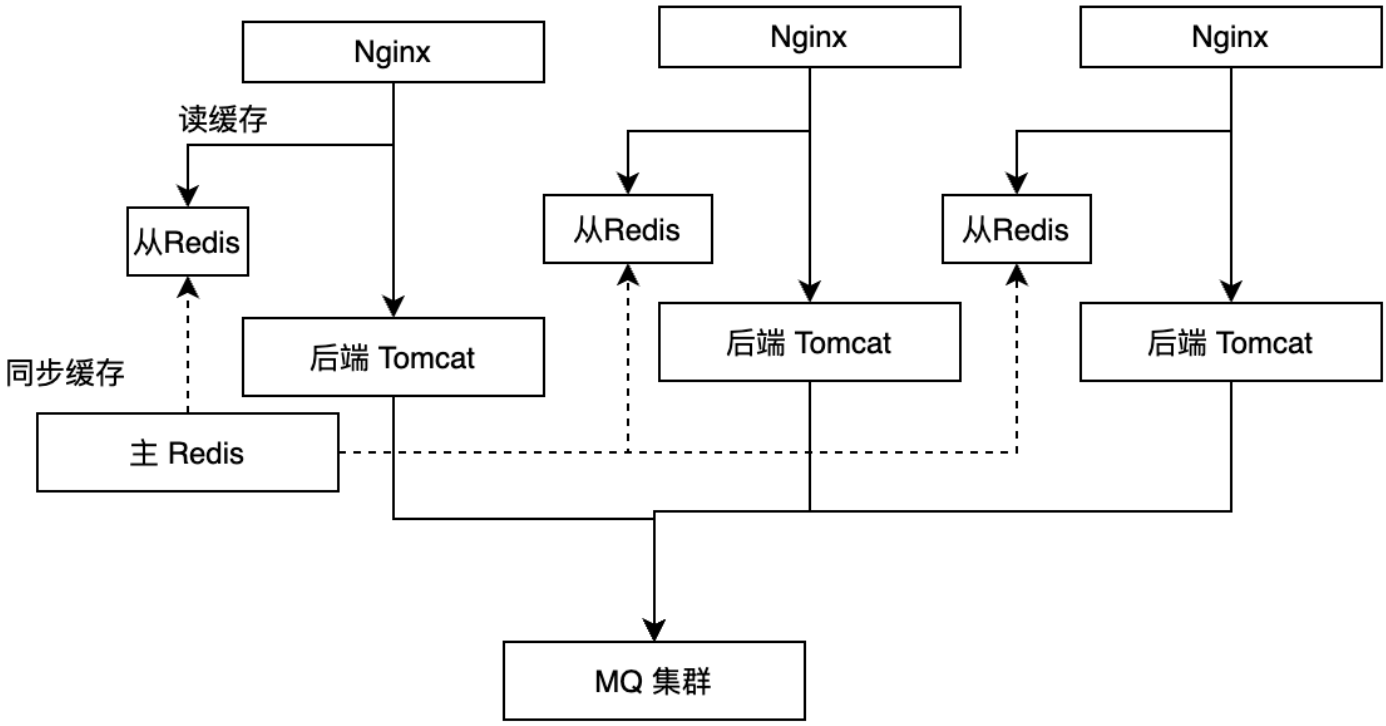
二、当前架构的痛点分析

从系统扩容的角度来分析当前架构，Redis 最容易成为性能瓶颈。因为秒杀应用可以基于Nginx反向代理进行水平扩展，RocketMQ 也可以通过集群配置进行水平扩展，而 Redis 则相对扩展比较困难。而对于 Redis，为了保证缓存数据的读写性能，当前架构中优先保证的是从 Redis 与秒杀应用部署在同一个服务器上，这样可以最大限度提升缓存的读写性能。但是，这也让 Redis 成为后续系统扩容的瓶颈。所以，秒杀系统后续如果要进一步提升性能，最需要调整的是其中 Redis 部分。

为此，我们先将当前秒杀系统中Redis部分的核心架构整理出来，以便后续进行重点分析。



你可能第一眼就会看到，这样单点的 Redis 服务，显然是不符合高并发应用的集群化要求的。但是，你要注意，其实，Nginx节点是可以和应用一起扩展的。后续可以部署多个Nginx节点，然后同样给每个 Nginx 节点配置一个 Redis 从节点，这样就可以进行整体扩容。像这样：



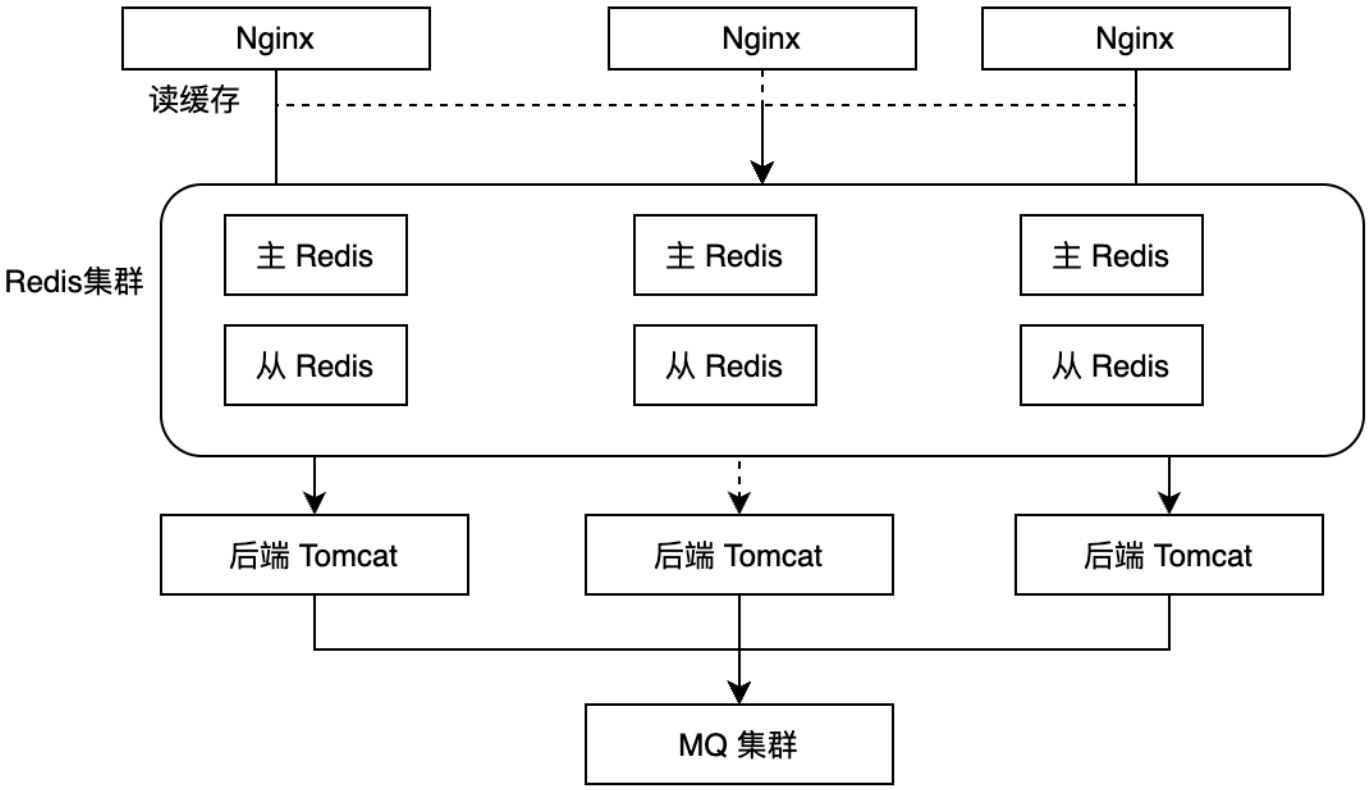
所以，当前架构的核心痛点，其实并不在于单点 Redis 的性能问题。而是，单点 Redis 的数据压力会比较大。

当前我们电商项目中，秒杀的商品还不太多，数据比较少。但是如果后续秒杀商品多起来，单点 Redis 的数据压力就会比较大。这会极大的影响 Redis 的性能。

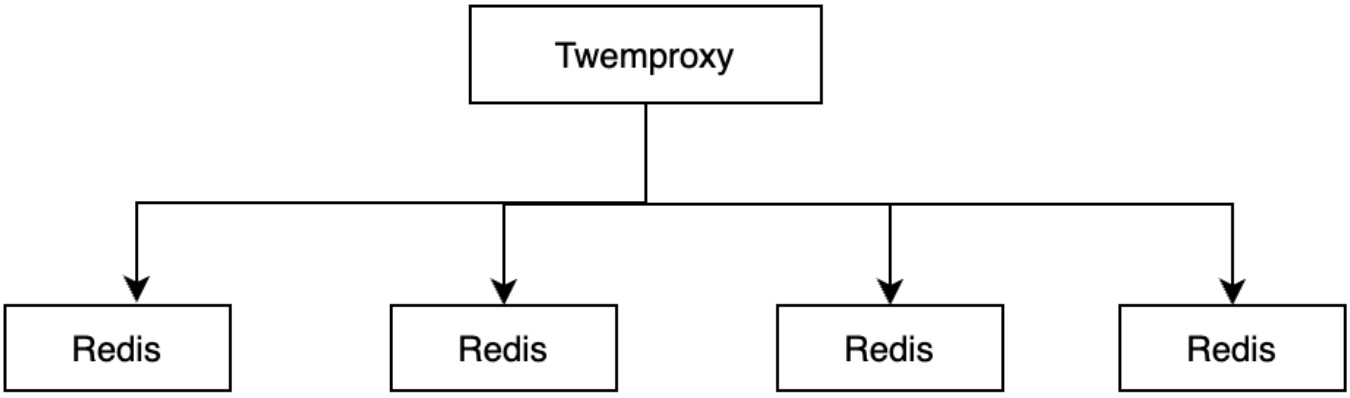
三、常见改进方案

1、Redis 垂直拆分方案

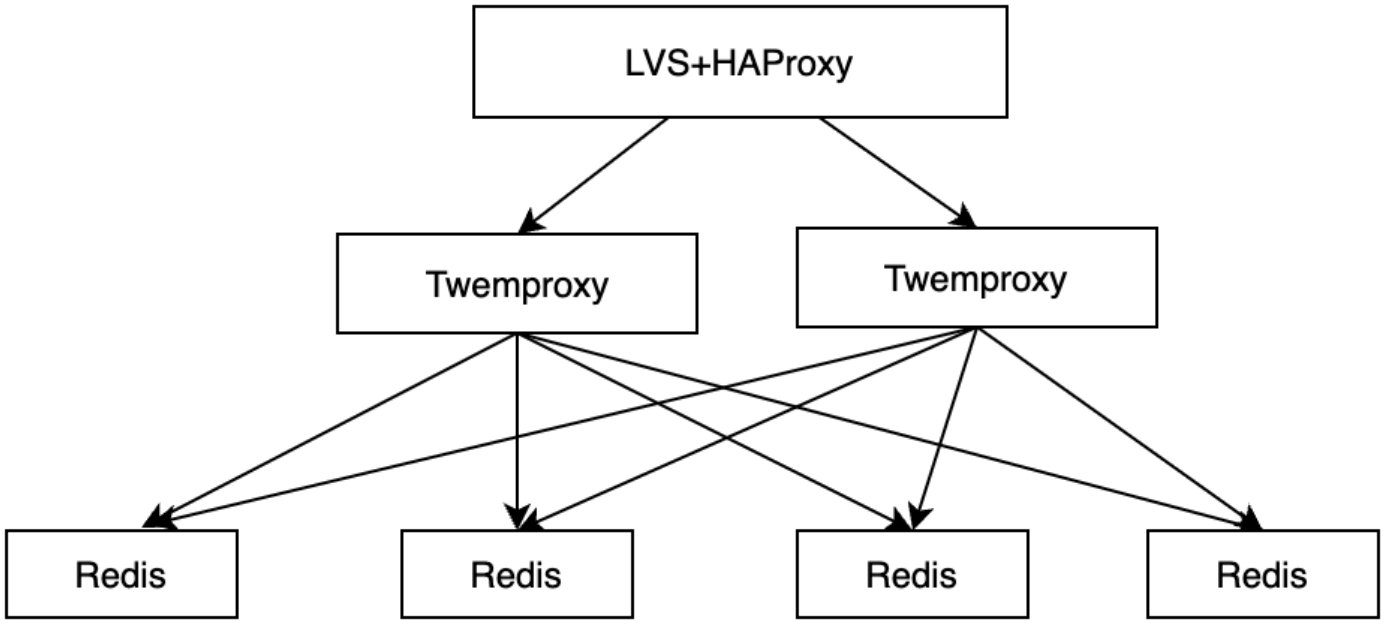
既然单点的 Redis 数据量不够，那么最简单的想法就是将单机 Redis 升级成为集群。通过集群机制扩充 Redis 的数据量。



升级成为 Redis 集群后，Redis 的数据量问题得到了解决。但是，要注意一下，在秒杀场景下，我们也不是简单升级完 Redis 集群就可以了的。采用集群部署 Redis 后，因为大部分的 Redis 客户端都是通过连接池实现的，此时 Redis 的连接数就会逐渐成为瓶颈。一般的办法是通过中间件来减少连接数。例如，使用TwemProxy于 Redis 之间建立单链接交互，并通过Twemproxy实现分片逻辑。这样我们就可以水平扩展出更多的Twemproxy来增加连接数。



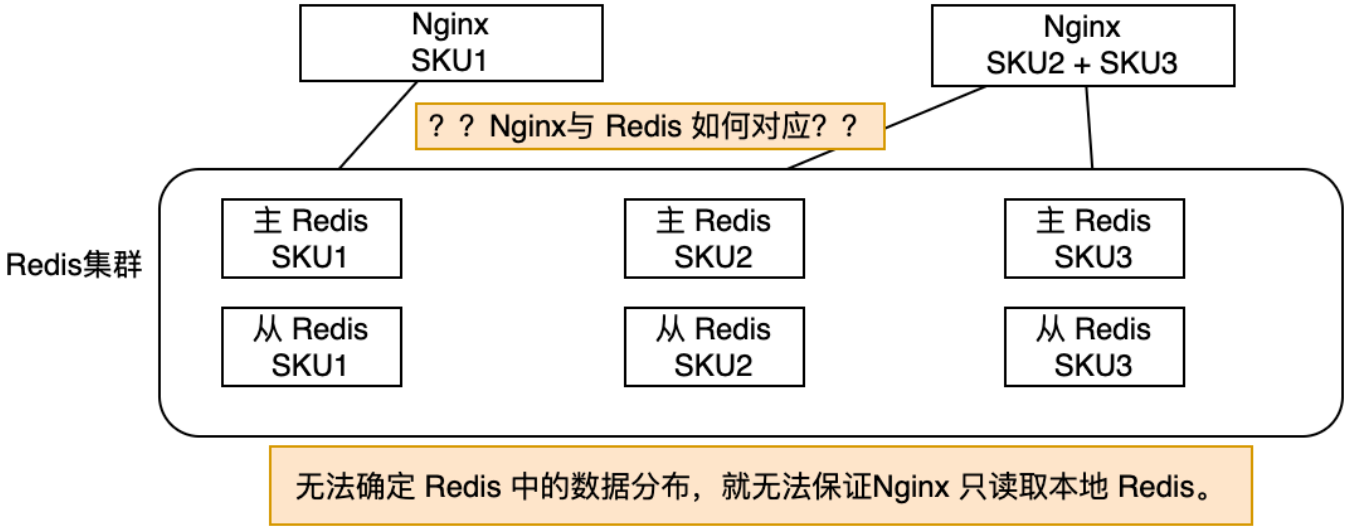
此时遇到的问题就是Twemproxy实例众多，应用维护、配置困难，需要在这之上做负载均衡。比如，通过 LVA/HaProxy 实现VIP(虚拟 IP)，就可以做到节点切换对应用透明、故障自动转移。还可以通过实现内网 DNS 来做其负载均衡。



通过 Redis 的集群化部署，多个 SKU 的数据，最终会分散到各个 Redis 节点当中，我们就可以解决 Redis 数据量太大的问题。并且，基于 Redis 集群的方式，Redis 服务的可用性也得到了提高。我们可以通过再加入哨兵机制等方式，保证少量 Redis 节点挂了后，整个 Redis 集群不会出现问题。

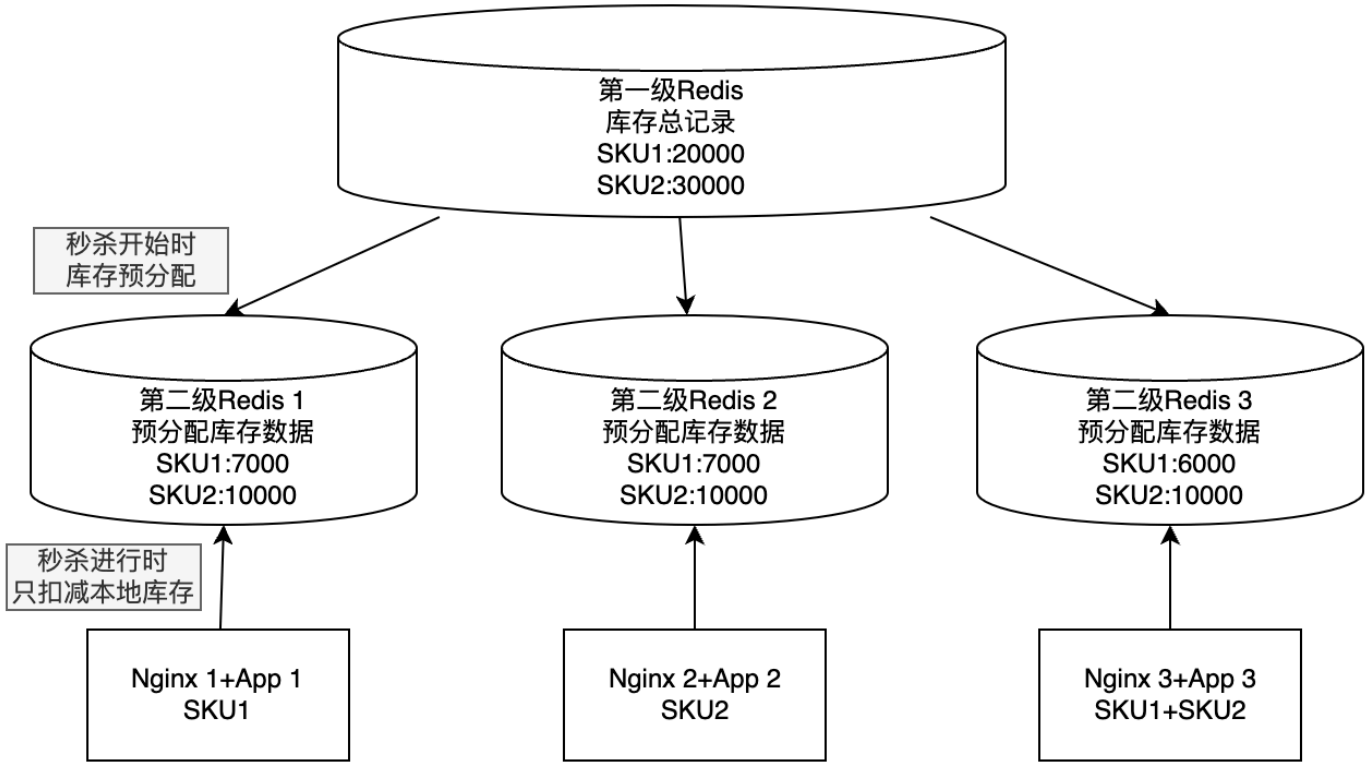
2、库存预分配方案

通过 Redis 的集群化部署，我们就可以解决 Redis 数据量太大的问题。但是，与之前架构不同的是，我们只能使用一致性算法实现 Redis 集群，而这样就无法保证每个Nginx只读取本地的 Redis，这样其实会对 Redis 的读写性能产生一定的影响。



所以，在很多互联网企业中，不会直接使用 Redis 的集群架构，而是搭建多个 Redis 节点。并通过库存预分配的方式，自行控制 Redis 中的数据分布。

例如，自行搭建多级 Redis 缓存，第一级 Redis 管理总的秒杀库存。在秒杀活动开始时，增加一个库存预分配的程序，将总库存分配到多个二级 Redis 中。然后，在每个二级 Redis 上，就可以搭建一个对应的 Nginx 加 APP 的秒杀应用。每个应用就只访问对应的二级 Redis。而二级的 Redis 就可以还原成我们当前秒杀系统中的单点架构，从而保证每个Nginx 加 APP 的秒杀应用可以只访问本地的 Redis。



> 图中，Nginx与 App 的对应关系，是通过Nginx的负载均衡配置灵活指定。因此，他们是可以通过 Nginx 组合成一个个相对独立的秒杀应用的。
>
> 但是，为了尽量保证每个Nginx 和秒杀应用都是操作本地的 Redis，所以，通常建议都配置成 1+1 的组合。而且，保持这个原则，也就不用再另外维护秒杀应用与二级 Redis 的对应关系了。

这样调整后，就给整个秒杀架构带来了非常明显的优点：

1、通过预分配库存的方式，可以让每个应用，都有一套独立的库存数据。各个应用之间，处理库存的时候，互不干扰。从而，大大降低了并发两，以应对高并发。这样一来，应用就可以横向扩展。另外，由于各个应用扣减库存的速度不一致，这也可以一定程度上防止羊毛党。

2、具体扩展多少个应用和预分配多少套库存数据，也要看两个纬度，一个是并发量，一个是活动的库存数据。不同热度的秒杀活动可以配备不同的资源。在当前项目中，只要在前端引导用户进入不同的静态页面即可。

3、可以一定程度上解决热点商品的问题。由于每个二级 Redis 里可以分配多个 SKU 的活动信息，因此每个二级 Redis 对应的秒杀应用，也是可以承载多个商品的秒杀活动的。这样，即使出现某一个特别火热的热点商品，也可以通过部署多个秒杀应用的方式，增加热点商品的承载能力。

但是，这个架构也是有他自己的不足之处的：

1、前端秒杀页合理导航的问题。 在我们的秒杀系统中，是通过往OpenResty中发布的静态页面作为秒杀的入口的。如果想要多个服务平均的承载同一个商品的秒杀服务，就需要在前端进行合理的导航。将不同的用户导入到不同的OpenResyt对应的秒杀页面。而这时，前端的导航很难与预分配的库存进行沟通。可能某一个二级 Redis 服务分配的库存很多，但是前端导航很少，这就造成了商品的浪费。

2、服务出现问题时，库存数据很难回收。为了尽量让秒杀应用只读取本地的 Redis 数据，所以二级 Redis 大概率还是会采用单点的方式进行部署。而此时，由于每个二级 Redis 中的库存数据并不完整，如果二级 Redis 在活动中途出现崩溃，那么基于这个 Redis 的库存数据就无法回滚。这样，就会影响到整体的库存管理。

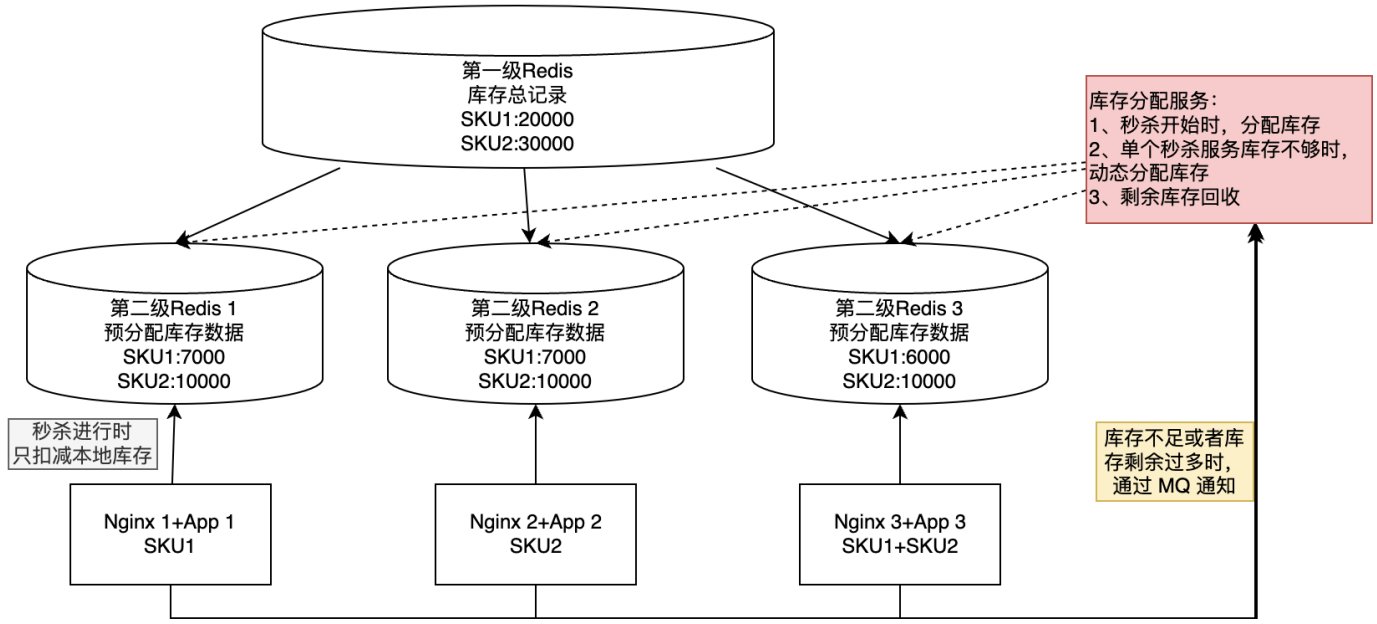
> 虽然我们说过，对于秒杀应用，单节点的 Redis 可用性不是首要的。但是，在纯粹单节点情况下，至少活动数据是可以通过日志管理的，即使出了问题，也可以通过日志快速恢复，后续可以进行返场等活动进行补救。
>
> 但是，采用预分配之后，库存的管理是分散的，中途出现问题，库存的管理就会变得很复杂，就无法恢复到整个集群全部正常时的场景。

3、这种预分配方案中，各个服务之间无法进行沟通，这会造成很多尴尬的业务问题。比如图中，每个二级 Redis 上，都分配全量的 SKU 信息(只有这样才比较好实现)，但是只有二级Redis 1和二级 Redis 3 上承载了 SKU 1 的秒杀活动，当这两个 Redis 上的库存扣减完成后，就无法再继续扣减了。而实际上，二级 Redis 2上，还有很多 SKU 1 的库存，就无法正常售卖。

另外，如果秒杀活动，一次可以抢购多个商品的话，如果出现了某一个服务上的库存不够一个订单时，也无法与别的服务进行沟通，进行合理的凑单。比如每个二级 Redis 中还剩下 1 个库存，但是某一个订单需要抢购 2 个商品。总的库存是足够的，但是却无法完成这个订单。

3、动态库存分配方案

之前库存预分配方案的核心问题在于库存只是在活动开始时进行分配，而不能在活动扣减过程当中进行灵活调整。所以，我们要解决这样的问题，就需要独立出一个库存预分配的服务，在扣减过程中，灵活调整各个服务的库存。



首先，将库存分配服务独立出来，这样便于与其他秒杀应用进行沟通。

接下来，在秒杀活动开始时，进行库存预分配。这时，就不一定要将所有库存全部分配完成，可以在一级 Redis 中预留一部分库存。

然后，在每个秒杀应用独立扣减库存，当发现库存低于某一个阈值下限时(不要等到库存扣没了才通知)，通知库存分配服务，库存分配服务再访问各级 Redis，对库存进行统一重分配。

重分配时，优先从一级 Redis 中分配库存，这样的速度是比较快的。一级 Redis 中库存扣减完成后，再协调二级 Redis 进行库存动态分配。并且，在分配过程中，可以用一级 Redis 作为统一缓存进行协调。比如，发现二级 Redis 2 上的 SKU 1 的库存就没有扣减过，这时，就可以直接将二级Redis 2上的 SKU 1 库存全部回收，分配给其他二级Redis。而当其他二级Redis中的库存超过了某一个阈值上限时(由于其他 Redis 上的库存扣减速度不一致，所以不建议一次全部平均分配完。预留一部分库存作为机动库存，可以加快分配的速度)，将多出来的库存放回到一

级 Redis 中。

最后，在秒杀活动快要结束或者是库存快要完全售完时(比如，所有二级 Redis 中的库存全都低于某一个临界点)，可以将剩余库存全部回收，然后一起分配给某一个二级 Redis。这样就可以在遇到大订单时，努力进行最后的拼单。

很多互联网企业都采用了这样一套架构。如果做得比较细致的话，库存分配服务还可以用来监控各个秒杀应用的状态。如果发现某个秒杀应用挂了，也可以及时回收其对应的二级 Redis 库存。而更进一步，如果做到了对每个秒杀服务状态的准备把控，那么，甚至可以将二级 Redis 进一步改为秒杀应用的本地 RocksDB 缓存，这样，就可以再次提升秒杀应用的并发性能。

总之，独立出这个库存分配服务后，就可以动态调整各个零散的二级 Redis 库存。这样，即保证了每个秒杀应用可以只操作本地的 Redis 缓存，最大限度增加应用的吞吐量，又能保证这些独立的秒杀应用可以整体进行协调，像一个整体进行工作。