互联网大厂的缓存策略：抵抗超高并发的秘密武器，已开源！ - 冰河团队 - 博客园

互联网大厂的缓存策略：抵抗超高并发的秘密武器，已开源！

大家好，我是冰河~~

最近，有小伙伴私信我：冰哥，我最近出去面试，面试官问我如何设计缓存能让系统在百万级别流量下仍能平稳运行，我当时没回答上来。接着，面试官问我之前的项目是怎么使用缓存的，我说只是缓存了一些数据。当时确实想不到缓存还有哪些用处，估计这次面试是挂了。冰哥，你可以给我讲讲互联网大厂项目是怎么设计和使用缓存的吗？

> 另外，本文缓存方案已经开源，开源地址如下，如果开源方案对你有点帮助或者启发，欢迎在代码仓库给个Star，让更过的小伙伴看到它，互相学习，一起进步。
>
> - GitHub：https://github.com/binghe001/spring-redis
> - Gitee：https://gitee.com/binghe001/spring-redis
> - GitCode：https://gitcode.net/binghe001/spring-redis

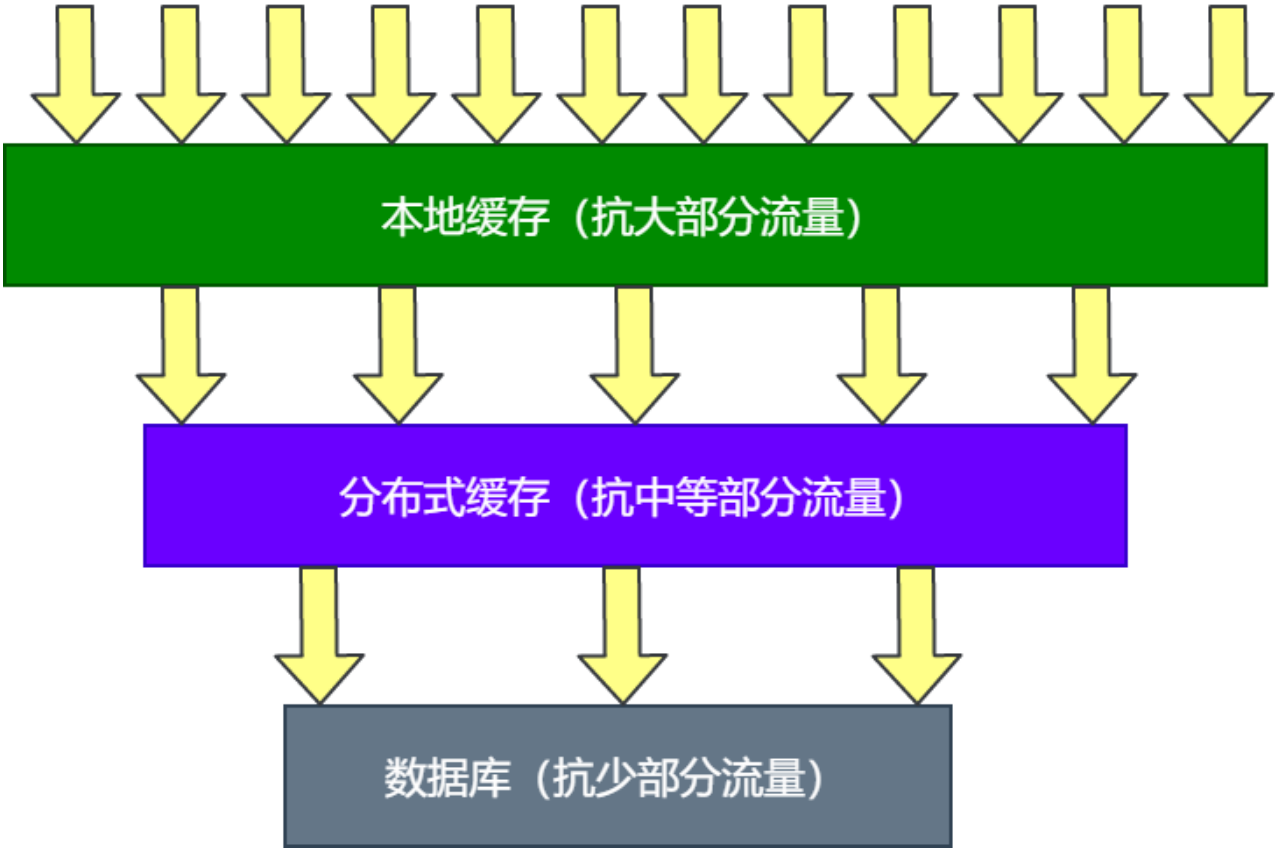
一、前言

通过这位小伙伴的自述，我们明显感受到这位小伙伴对缓存的认识还是停留在简单的存储数据上，没有对使用缓存背后的场景和实现逻辑进行深层次的思考。在互联网大厂项目中，缓存也是一种必不可少的组件，那使用缓存仅仅是为了缓存热点数据，提升读性能吗？如果你对缓存的认识只是停留在这里，那就未免太浅显了。

今天，我们就以高并发、大流量业务场景中最具代表性的 **秒杀系统** 为例，采用市面上大家都比较熟悉的技术，一起探究下 **秒杀系统** 背后是如何设计和使用缓存的。

二、秒杀系统缓存核心诉求

秒杀系统在承接瞬时高并发流量时，如果将流量直接打到数据库，那数据库很有可能因为扛不住瞬间的高并发流量而导致崩溃和宕机。所以，需要对秒杀系统进行极致的缓存设计，让大部分流量走缓存。同时，在设计缓存架构方案时，为了进一步提升性能，将采用 **本地缓存+分布式缓存的混合型缓存** 设计方案，让本地缓存抗大部分流量，分布式缓存次之，数据库再次之，如图1所示



并且针对秒杀系统这种瞬时并发量高的场景，在设计缓存时，需要注意的技巧：优先读取本地缓存数据，如果本地缓存失效，则读取分布式缓存数据，并且在同一时刻，只能有一个线程更新本地缓存，防止缓存击穿。没有获取到本地缓存更新机会的其他线程，需要立即返回而不是原地等待。如果分布式缓存失效时，在同一时刻，也只能有一个线程更新分布式缓存，防止缓存击穿。没有获取到分布式缓存更新机会的线程，也需要立即返回而不是原地等待。

另外，需要注意的是：我们提出了采用 本地缓存+分布式缓存的混合型缓存设计方案，后文会着重对这种设计进行说明。

三、秒杀系统缓存使用场景

秒杀系统属于典型的读多写少的高并发系统，应对这种场景的一个有效措施就是使用缓存，不管是单机JVM缓存还是以Redis为例的分布式缓存，其读写性能都会比数据库高得多。所以，在秒杀系统中，为了应对高并发、大流量的业务场景，缓存自然也就成为建设秒杀系统过程中必不可少的环节。

3.1 秒杀系统接口分析

在秒杀系统中，主要是对一些读数据的接口设计缓存策略，而在这些读数据的接口中，获取秒杀活动列表、获取秒杀活动详情、获取秒杀商品列表和获取秒杀商品详情的接口流量比其他接口高。尤其是获取秒杀商品列表和获取秒杀商品详情的接口QPS一般会高于获取秒杀活动列表和秒杀活动详情的接口，毕竟大部分用户在秒杀开始前就已经进入到秒杀详情页，当然这也不是绝对的，还是要看秒杀系统对于这些接口的设计。

3.2 秒杀系统缓存场景

尽管获取秒杀商品列表和获取秒杀商品详情的接口QPS一般会高于获取秒杀活动列表和秒杀活动详情的接口，但是我们在设计缓存时，需要对这些接口一视同仁，都要以严格的高标准来设计这些接口，不然稍有不慎，一个接口出现问题，就可能导致整场秒杀活动以失败告终。秒杀系统缓存的使用场景如图2所示。

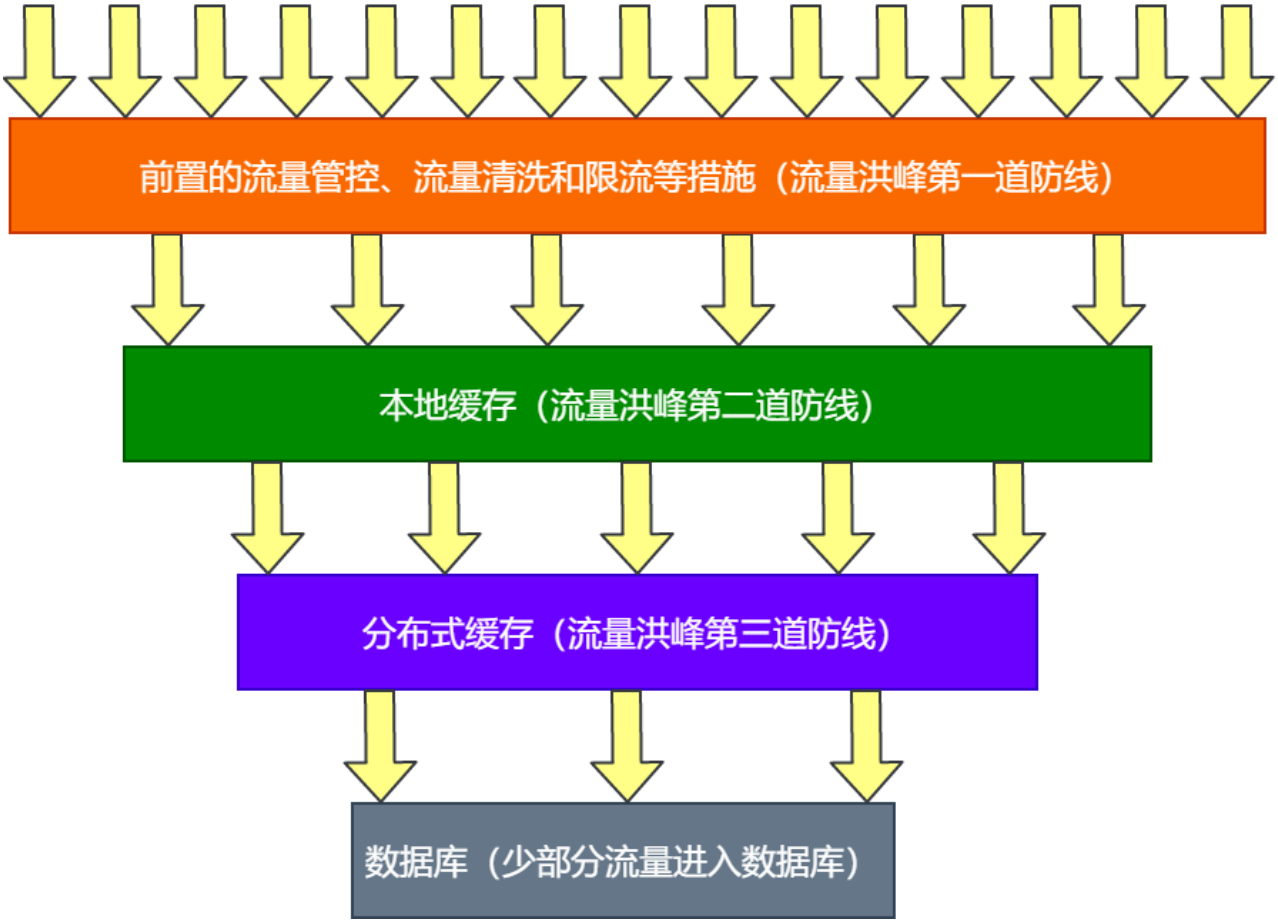
所以，在秒杀系统中，会对获取秒杀活动列表、获取秒杀活动详情、获取秒杀商品列表和获取秒杀商品详情的接口设计缓存策略。

四、混合型缓存设计

总体来说，在设计秒杀系统的缓存过程中，会采用 **本地缓存+分布式缓存的混合型缓存** 设计方案。其中，本地缓存指的就是单机缓存，比如JVM内存缓存，单机Cache缓存。分布式缓存指的是以分布式的方式集中管理的缓存，比如Memcached、Redis等，如图3所示。

4.1 抗流量洪峰

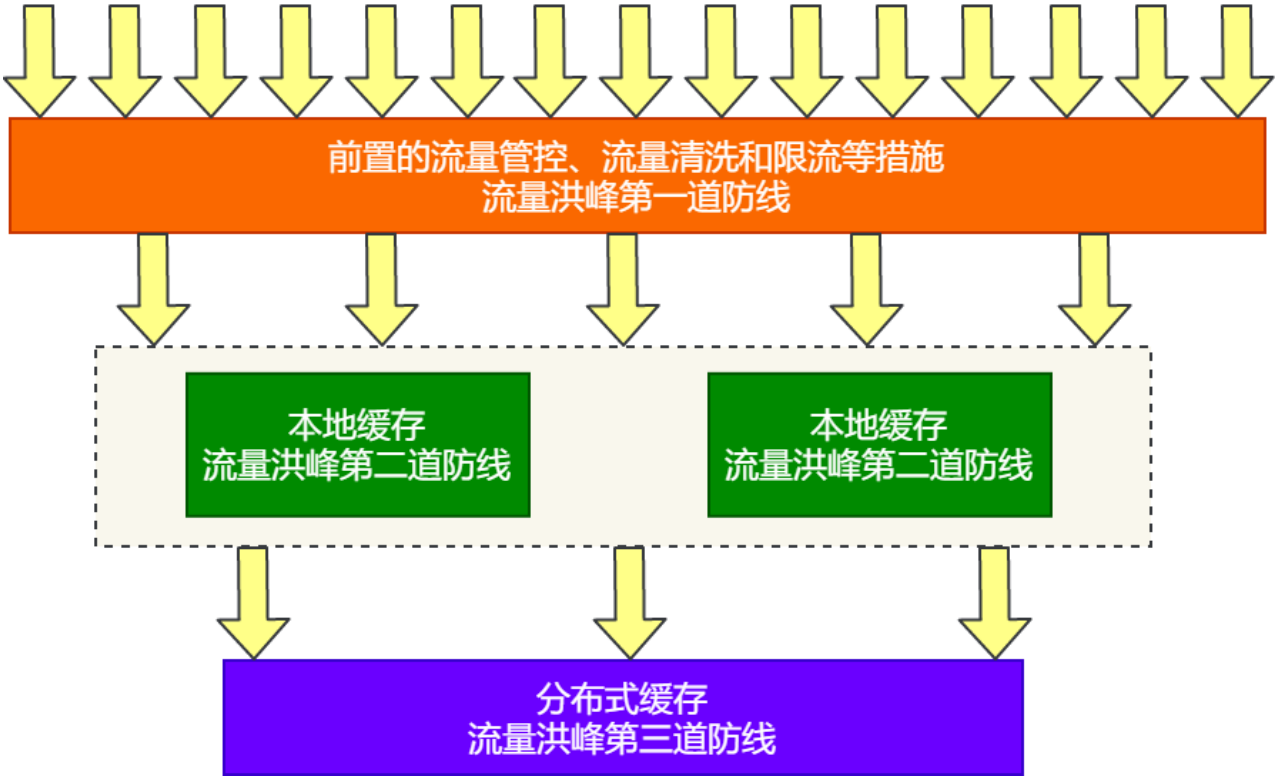
良好的缓存设计不仅仅能够提升系统的总体性能，还能作为抗瞬时流量洪峰的有效防线。可以这么说，如果整个秒杀系统前置的流量管控、流量清洗和限流等是秒杀系统流量洪峰的第一道防线，则本地缓存就是抗流量洪峰的第二道防线，而分布式缓存就是第三道防线，如图4所示。



使用缓存能够抗一定的流量洪峰，经过前置的流量管控、流量清洗和限流等措施的第一道防线、本地缓存的第二道防线、分布式缓存的第三道防线，真正进入数据库的流量就会比较小了。

4.2 缓存集群方案

从缓存集群模式的角度去分析，每台服务器甚至JVM实例都会拥有自己独立的本地缓存，在承载大并发流量时，，以本地缓存为主，分布式缓存次之，如图5所示。



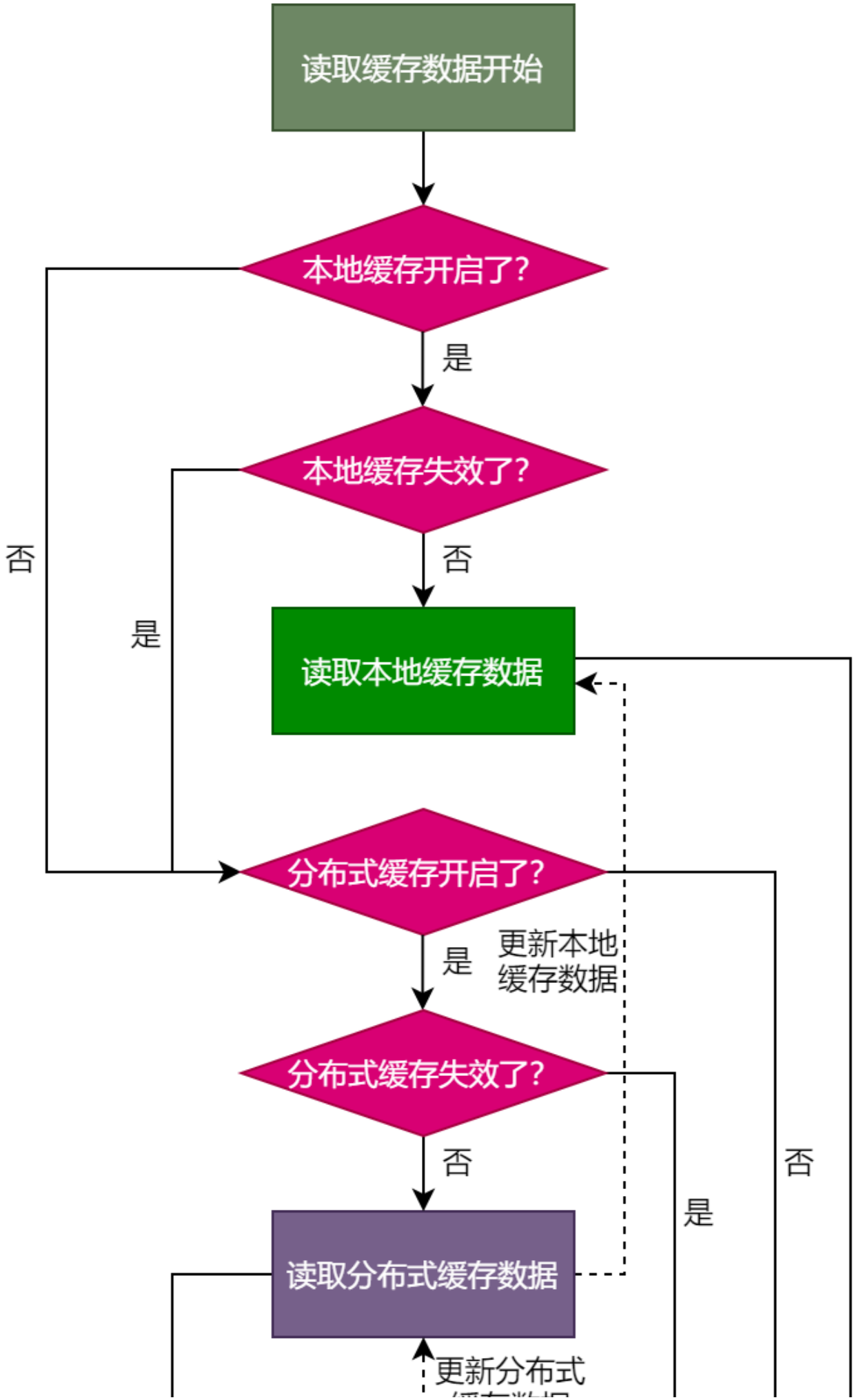
可以看到，从缓存的集群模式角度来看，每台服务器都会自己独立本地缓存，除了前置的流程管控、流量清洗和限流等措施构筑的流量洪峰第一道防线外。本地缓存会承接剩余的大部分流量，构筑成流量洪峰的第二道防线，而分布式缓存则是流量洪峰的第三道防线。并且在缓存的设计上，分布式缓存的作用主要是协调和同步最新数据到本地缓存。

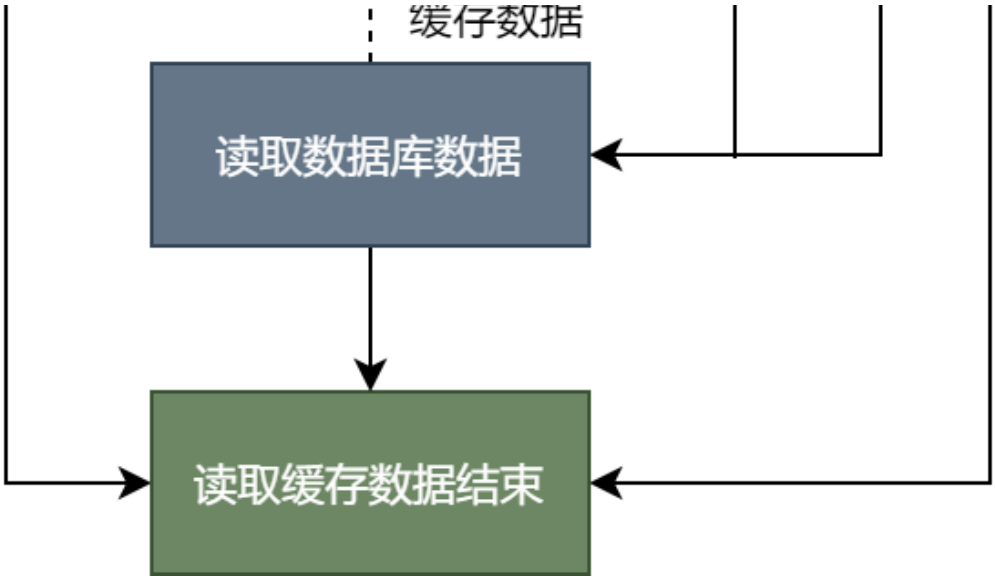
也就是说，只有本地缓存失效时，才会访问分布式缓存，将分布式缓存中的数据更新到本地缓存中，并且同一时刻只能有一个线程对本地缓存进行更新操作，以避免多个线程并发更新本地缓存。同样的，如果分布式缓存失效，则同一时刻只能有一个线程访问数据库来获取对应的数据，并将其更新到分布式缓存。

在集群模式下，我们应该尽最大努力将流量拦截在本地缓存，避免过多的请求访问分布式缓存，提高秒杀系统的性能，并且降低秒杀系统由于大量的远程IO导致的各种风险。

4.3 缓存交互流程

采用本地缓存+分布式缓存的混合型缓存架构设计方案时，在读取缓存数据时，会优先读取本地缓存的数据，如果本地缓存未开启，或者已经失效，此时就会使用分布式缓存，也就是说，优先读取本地缓存中的数据，如果本地缓存未开启或者缓存数据失效，则读取分布式缓存中的数据，如图6所示。


读取缓存数据开始
本地缓存开启了?
是
本地缓存失效了?
否
否
是
读取本地缓存数据
分布式缓存开启了?
是
更新本地
缓存数据
分布式缓存失效了?
否
否
是
读取分布式缓存数据
更新分布式



可以看到，只有在本地缓存未开启或者缓存失效的情况下，才会去访问分布式缓存，读取分布式缓存中的数据，并且在同一个时刻只能有一个线程更新本地缓存中的数据，这种方式可以最大限度减少远程IO为秒杀系统带来的风险。具体的流程如下所示。

（1）判断本地缓存是否开启，如果开启则进行第2步，否则进行第4步。

（2）判断本地缓存是否失效，如果未失效，则进行第3步，否则进行第4步。

（3）读取本地缓存数据，读取缓存流程结束。

（4）判断分布式缓存是否开启，如果开启则进行第5步，否则进行第7步。

（5）判断分布式缓存是否失效，如果未失效，则进行第6步，否则进行第7步。

（6）读取分布式缓存数据，同一时刻只有一个线程更新本地缓存数据，读取缓存流程结束。

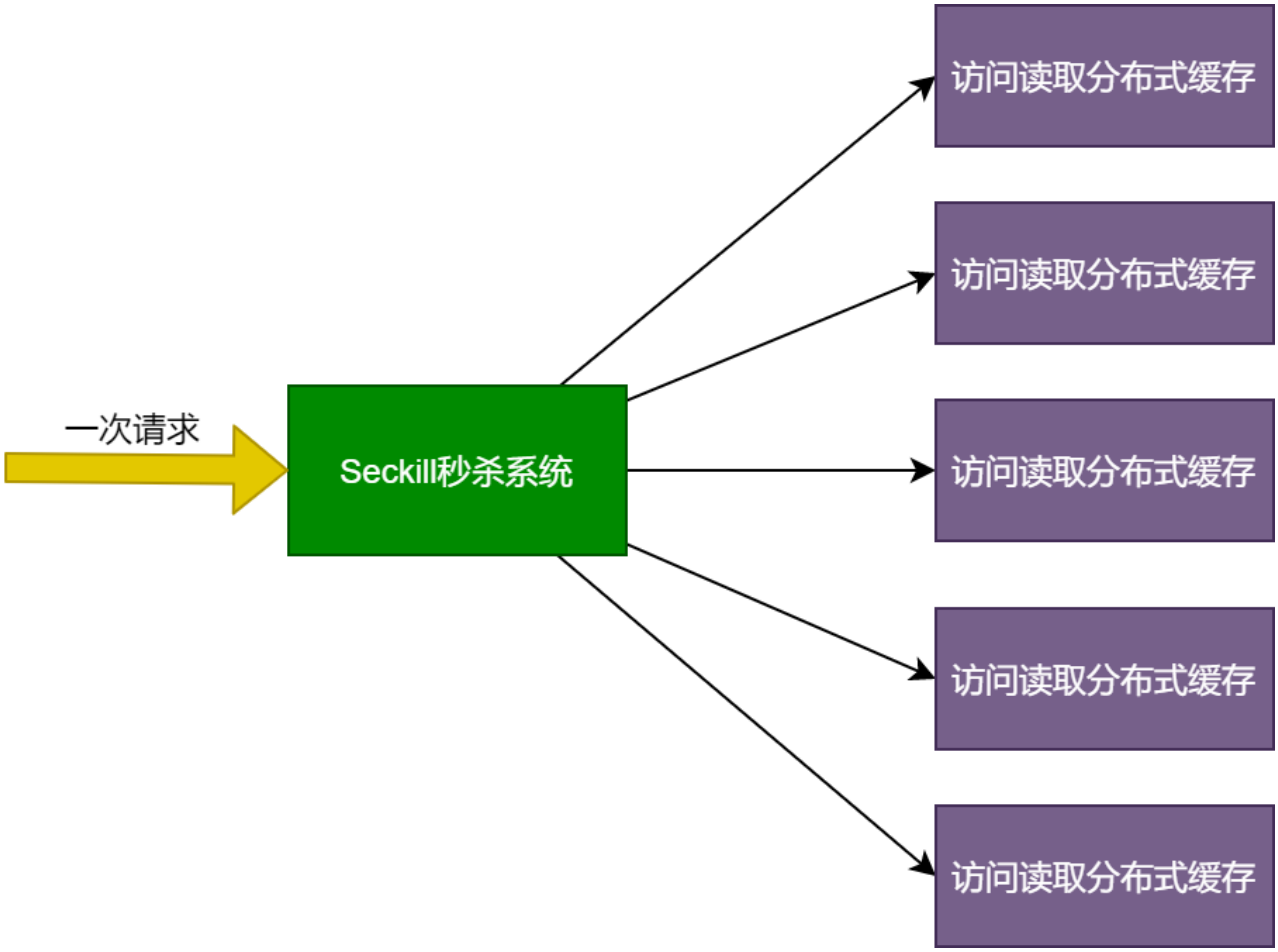
（7）读取数据库数据，同一时刻只有一个线程更新分布式缓存数据，读取缓存流程结束。

这里，有一个设计技巧需要大家注意：如果本地缓存失效，并且某个线程没有获取到更新本地缓存的机会，这个线程需要立即返回而不是在原地阻塞等待，这种方式可以最大限度的节省服务器资源和线程切换的成本，尤其是像在秒杀系统这种承接瞬时高并发流量的系统中，这种设计能够节省不少服务器资源。这种线程未获取到更新数据的机会而快速返回的机制，需要客户端配合在适配处理，也就是说，客户端对这种情况需要进行静默处理，不要提示错误信息，也不做其他处理，稍后重新调用接口进行重试即可。

4.4 混合型缓存设计的优点

采用本地缓存+分布式缓存的混合型缓存架构设计方案存在诸多的优点。其中，本地缓存一个很大的优势就在于不会发生远程IO操作，性能更高，有利于服务的横向伸缩，大部分请求会命中本地单机缓存。这里，我们可以从整体的请求链路上进行分析。

例如，当前请求链路上需要读取5次分布式缓存中的数据，这样，如果秒杀系统承接了100万的请求，则会产生500万读取分布式缓存的IO操作。这成倍的IO风险对于秒杀系统来说，是绝对不能忽视的风险因素，如图7所示。



可以看到，一次请求会访问5次分布式缓存，这在无形当中就增加了分布式缓存的IO成本，这对秒杀系统来说，是不容忽视的风险项，稍有不慎，则系统可能会由于IO瓶颈引发各种事故，最终造成系统崩溃或者宕机。所以，在设计秒杀系统时，一定要注意这种放大效应带来的风险。所以，在高并发大流量的场景下，很有必要精心的设计本地缓存。

五、缓存刷新机制

数据存放到缓存中，并不是一成不变的，也不会永久存放到缓存中。也就是说，存放到缓存中的数据终归是要失效或者过期的，也就是存放到缓存中的数据会有相应的生命周期，为此需要以一定的策略对缓存中的数据进行刷新操作，以防止缓存中的数据长时间过期而导致大部分流量直接打入数据库。本节，就从本地缓存和分布式缓存两个角度简单聊聊缓存的生命周期。

5.1 本地缓存刷新机制

假设本地缓存基于Guava Cache实现，在设计本地缓存时，本地缓存的容量不宜过大，有效时长不宜过大，并且在设计本地缓存时，可以基于版本号机制来实现缓存的失效策略。

对于本地缓存会实现两种刷新机制：

（1）主动刷新

请求接口传入的版本号如果大于本地缓存中的版本号，说明本地缓存已经失效，此时，就需要从分布式缓存中重新获取数据进行刷新。

（2）被动刷新

本地缓存自动过期，被动从缓存中移除，此时，需要从分布式缓存中重新获取数据进行刷新。

5.2 分布式缓存刷新机制

假设分布式缓存基于Redis实现，对于分布式缓存来说，也需要设置缓存的过期时间，不能让缓存数据永久性驻留到Redis中。相比于本地缓存来说，分布式缓存的过期时间要稍微长一些，并且分布式缓存在刷新机制上与本地缓存略有不同。

（1）主动刷新

业务数据变更驱动刷新分布式缓存数据。当业务数据发生变更时，会主动刷新分布式缓存中的数据。

（2）被动刷新

可以基于Redis提供的缓存过期策略，比如基于LRU、TTL等策略淘汰缓存中的数据。后续在访问分布式缓存中的数据时，如果检测到分布式缓存中的数据已经过期，则会使用一个线程来刷新分布式缓存中的数据。

六、数据一致性

可以这么说，只要系统中使用了缓存，就或多或少会涉及到数据一致性的问题，在秒杀系统中，数据一致性的问题主要包括：本地缓存与分布式缓存数据一致性问题，缓存与数据库数据一致性问题。同时，在数据一致性保证方面，就包括强一致性保证和弱一致性保证。

6.1 强一致性保证

CAP理论为数据的强一致性奠定了理论基础，但是CAP理论下的数据强一致性，很难做到既保证系统高性能的同时，又要保证数据的绝对一致。在秒杀系统的设计中，我们会将数据的强一致性保证交给数据库和业务规则来实现，在业务规则层面结合数据库来实现强一致。

例如，假设用户在抢购秒杀商品中，缓存中存在商品库存，通过了缓存中的校验逻辑。在真正下单时，还要校验数据库中的商品库存，如果此时数据库中已经没有商品剩余库存了，则终止下单逻辑，提示用户商品已售罄。

6.2 弱一致性保证

强一致性保证交由业务规则和数据库共同约束实现，缓存层面的数据就可以实现为弱一致性。也就是说，在很小的一段时间内，允许缓存中的数据存在延迟，允许缓存中的数据与数据库中的数据在短时间内的不一致，只要在可接受的时间范围内最终达到一致即可。充分发挥缓存的实际作用，即：缓存数据，提供系统的读写性能和抗系统流量。

七、缓存落地实现

在秒杀系统中本地缓存和分布式缓存相结合，能够抗住进入秒杀系统内部的大部分流量。并且在技术选型上，假设本地缓存默认基于Guava Cache实现，分布式缓存默认基于Redis实现。并且本地缓存不仅仅只是支持Guava Cache，分布式缓存不仅仅只是支持Redis，在代码层面，都是面向接口编程，而非面向具体实现类编程，不管是本地缓存还是分布式缓存，都可以根据简单的配置切换具体的实现方式。

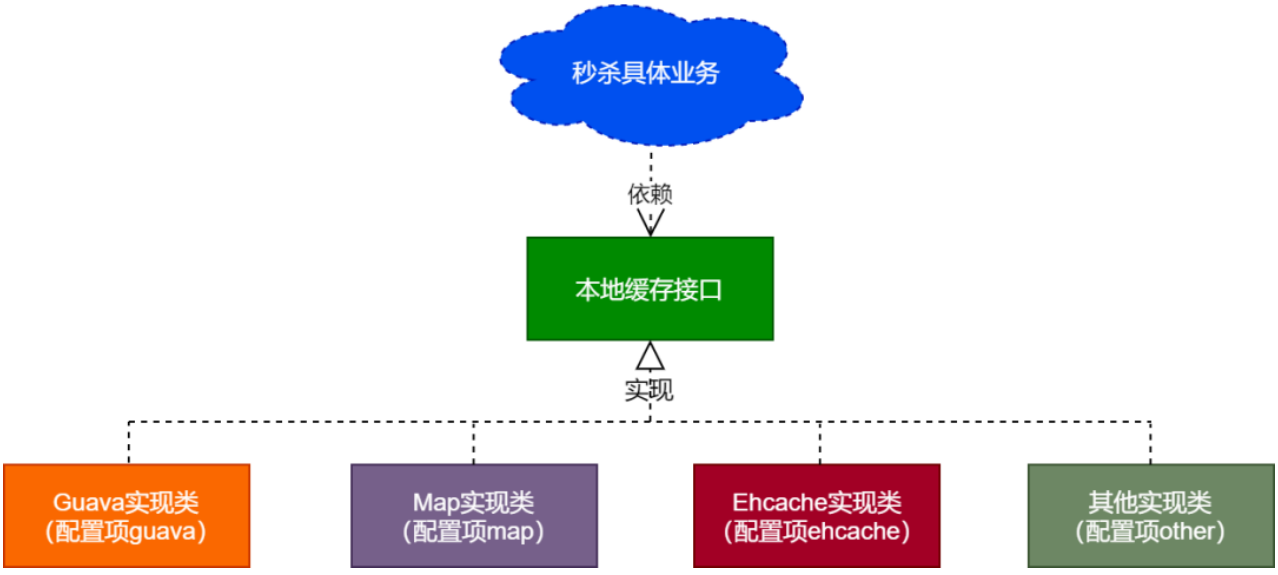
7.1 扩展性描述

代码具备良好的扩展性，后续维护和升级的成本就比较低。相反，如果代码写的杂乱无章，犹如“屎山”，那后期维护起来是相当痛苦的，谁也不想天天面对着一堆“屎山”，哪来有问题改哪里。所以，从一开始写的代码就要有良好的扩展性，方便后期的维护和升级。

假设秒杀系统整体基于SpringBoot+SpringCloud Alibaba技术栈实现，那如何写代码具备良好的扩展性呢？**总体的原则就是面向接口编程，而非面向具体的实现类编程，具体业务逻辑里依赖的是接口，而非实现类，在接口不变的前提下，可以随时切换具体的实现类，也可以随时新增接口的实现类。业务中可以根据配置加载接口的某个具体实现类。**

7.2 本地缓存落地实现

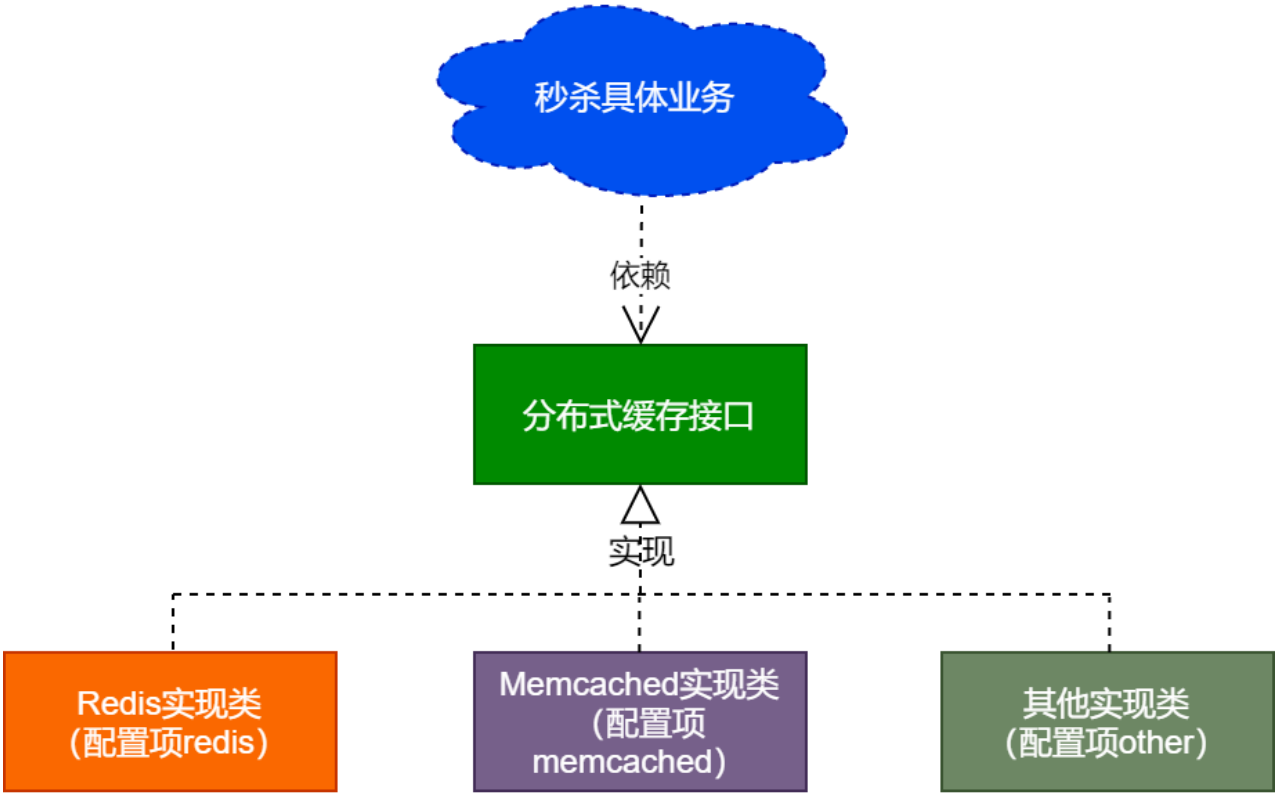
本地缓存的落地实现示意图如图8所示。



可以看到，具体秒杀业务中会依赖本地缓存的接口，而非具体的实现类。本地缓存的接口可以有多个实现类，在秒杀业务中可以根据具体的配置项指定要加载并使用哪个实现类，也可以根据具体的需求和业务场景随时新增本地接口的实现类，大大提高了程序的扩展性。

7.3 分布式缓存落地实现

分布式缓存的落地实现示意图如图9所示。



可以看到，分布式缓存在扩展性方面的设计与本地缓存类似，同样是秒杀系统在具体业务中依赖分布式缓存的接口，而非分布式缓存的具体实现类。分布式缓存的接口可以有多个实现类，在秒杀业务中可以根据具体的配置项加载并实例化具体的实现类，也可以根据具体的需求和业务场景新增分布式缓存接口的实现类，提高了实现分布式缓存程序的扩展性。

八、总结

缓存不仅仅可以用来存储热点数据，提升热点数据的读性能，还是业务系统中抗高并发、大流量的利器。以秒杀系统为例，采用本地缓存+分布式缓存的混合型缓存方案时，如果整个秒杀系统前置的流量管控、流量清洗和限流等是秒杀系统流量洪峰的第一道防线，则本地缓存就是抗流量洪峰的第二道防线，而分布式缓存就是第三道防线，经过层层流量过滤，最终进入数据库的流量就比较可控了。

同时，引入本地缓存+分布式缓存的混合型缓存方案后，要考虑缓存的刷新机制，数据一致性问题，在代码落地的过程中，还要最大程度避免缓存穿透、击穿和雪崩问题，并实现代码的高度可扩展性。

在提供的开源方案中，已经解决了缓存穿透、击穿和雪崩问题，开源地址如下：

- GitHub：https://github.com/binghe001/spring-redis
- Gitee：https://gitee.com/binghe001/spring-redis
- GitCode：https://gitcode.net/binghe001/spring-redis

如果开源方案对你有点帮助或者启发，欢迎在代码仓库给个Star，让更过的小伙伴看到它，互相学习，一起进步。

好了，今天就到这儿吧，我是冰河，我们下期见~~