链滴

# 缓存详解：穿透型缓存与旁路型缓存，缓存穿透，雪崩与击穿

作者：zhengliwei

原文链接：https://ld246.com/article/1604111988289

来源网站：链滴

hello，大家好，欢迎来到银之庭。我是 Z，一个普通的程序员。在实际开发中，我们通常面临都是读多写少的场景，这种场景下通常会需要缓存一些数据，来减少后端数据库的压力，今天我们就详细讲解下缓存架构的设计。
<h2 id="1--缓存类型">1. 缓存类型</h2>
<h2 id="1-1-穿透型缓存">1.1 穿透型缓存</h2>
我们先来看一种不太常见的缓存架构，叫做穿透型缓存，它的架构大致如下：

读服务总是读缓存系统，不直接与数据库进行交互，缓存系统会判断当前请求的数据是否在本地当然，也可能是某种外部的高速存储），如果在，就直接返回，如果不在，就请求数据库，获取数据保存到本地然后返回给上游服务。写服务也一样，只把数据写入缓存系统，由缓存系统写入数据库。
这种架构的好处是对读写服务友好，不需要关心读写数据库的逻辑，但缺点更多，一是缓存系统比较复杂，实现起来不方便，而且缓存与数据库同步实际上是业务逻辑，这就相当于业务逻辑散落在游服务和缓存系统两个系统上，不方便维护，另外，这种架构必须保证缓存系统读写请求的原子性，则可能产生数据不一致的问题，比如，缓存系统从数据库读出了内容，在写入本地时，接收到一个写求，把最新数据写入了本地缓存，这时候读请求可能用旧的数据覆盖新数据（虽然写请求会把新数据入数据库，可以认为持久化没问题，但以后的一定时间内的读请求都会读到旧数据，造成困扰）。所，这种架构并没有成为主流，实际上，我也不建议你在生产环境使用这种架构。我建议使用下面介绍的旁路型缓存。
<h2 id="1-2-旁路型缓存">1.2 旁路型缓存</h2>
我们生产环境中一般会使用旁路型缓存，它的架构大致如下：

这种架构有几个要点：
<ol>
<li>写服务是删除缓存，而不是更新缓存。假如是更新缓存的话，在个写服务并发的时候，如果不能保证写数据库和更新缓存的原子性，会存在两个请求操作叉，在数据库层面是服务 1 先更新，服务 2 后更新，最终是服务 2 的数据，但在缓存中是服务 2 先新，服务 1 后更新，最终是服务 1 的数据，造成缓存和数据库的数据不一致，而删除操作则没有这个题，不管服务 1 和服务 2 哪个先到缓存，删除操作的效果都是一样的。</li>
<li>写服务要先更新数据库，再删除缓存。假如先删除缓存的话，在写请求并发的情况下，可能出现缓存被写请求删除，但写请求还没来得及更新数据库，这读请求发现缓存中没有数据，去数据库读数据写入缓存，这时候就会把老数据写入缓存（然后写请求更新数据库）。而先更新数据库再删除缓存就可以避免这种情况。</li>
</ol>
<h2 id="2--可能的问题">2. 可能的问题</h2>
<h2 id="2-1-缓存穿透">2.1 缓存穿透</h2>
缓存穿透是指读服务大量请求一些不可能存在在缓存里的数据，比如缓存是商品 ID（写服务保证不为负数）为键，商品信息为值的键值对的话，读服务大量请求商品 ID 为-1 商品信息，这时如果读服务判断缓存里没有，直接回数据库查询的话，可能对数据库造成很大压力，垮数据库，这种情况一般是黑客攻击，毕竟，正常页面操作应该是不会请求负数的商品 ID 的（除非是 bug）。
这种问题的解决方案一般是：
<ol>
<li>读服务对参数做一些校验，过滤掉一些明显不符合逻辑的参数请求.比如负数的商品 ID 等。</li>
<li>读服务可以把查数据库为空的键也缓存起来，值设置为一个特殊的表示空的值，这个操作可以防黑客使用同一个或少数几个不存在的参数进行攻击的情况（这个黑客偷懒了）。不过，如果黑客有大不存在的参数的话，这个手段就不起作用了，因为黑客只需要在一个这些键都没有被缓存的时机（键会过期的，而且这些键正常不会有人访问，所以过期后一般不会重新被缓存），同时使用这些键调用口即可，可以瞬间打垮数据库。所以，这两个手段应该同时使用，才能起到较好的效果。</li>
<li>高阶一点的方案是用布隆过滤器校验参数是否真实存在，布隆过滤器的特点是如果认定某个元素<</li>
</ol>

trong>不在某个集合中时，是一定不在的；而如果认定某个元素在某个集合中时，不一定在。我们只需要在读缓存前把参数过下布隆过滤器，如果不在，就不用请求缓存了，直接返回兜底结果就行，这样以阻挡绝大多数不存在的参数请求，不过，引入了布隆过滤器，实现起来比较复杂，尤其是在参数集很多的时候，我们在设计技术方案时要综合考虑实现成本和收益来决定使用哪种方案。</li>
</ol>
<h2 id="2-2-缓存雪崩">2.2 缓存雪崩</h2>
缓存雪崩是指在某个时间，有大量缓存键过期，导致读请求的缓存命中率大幅下降，对数据库的压力增大导致压垮数据库。这种情况一般是写缓存逻辑不合理造成的，比如写入量在同一时间过期的键。
这种问题的解决方案一般是：
<ol>
<li>写缓存时保证键的过期时间不要集中在一个时间。</li>
</ol>
<h2 id="2-3-缓存击穿">2.3 缓存击穿</h2>
缓存击穿和缓存雪崩类似，都是由于键过期导致对数据库压力增大，区别在于，缓存雪崩是大量同时过期导致的，而缓存击穿是由于一个或少量几个热键过期导致的。从名字可以大概感觉出来：雪崩听起来就是大面积的压力压垮了数据库，而击穿听起来就是一根针或一道闪击穿了缓存到达了数据库，并打垮了数据库。
缓存击穿一般出现在有热键的业务场景里，解决方案一般是：
<ol>
<li>开发时预估当前业务是否会出现热键，如果会的话，需要对这个键的缓存进行特殊处理。</li>
<li>一种偷懒的做法是提前设置热键永不过期。</li>
<li>或者，当缓存过期时，在读数据库的操作上加分布式锁，让热键过期后只有一个线程能去查数据更新缓存，其他读请求等待一定时间，再次尝试读缓存。</li>
<li>再或者，写缓存的动作不由读请求来做，而是由一个后台线程去做，这个后台线程定期刷新缓存周期应该比缓存过期时间短一点，只要这个刷新过程足够短，对读请求的影响就不大，当然，读请求需要对缓存不存在的情况做兜底处理。</li>
</ol>
<h2 id="3--小结">3. 小结</h2>
以上我们学习了缓存系统的两种架构：穿透型和旁路型，穿透型缓存由于实现复杂度和数据不一（主要）的问题，不推荐使用。旁路型缓存的设计需要注意两点：1. 先写数据库再删缓存; 2.删除缓，而不是更新缓存。接着，我们学习了缓存系统常见的三个问题：缓存穿透，缓存雪崩和缓存击穿。存穿透一般是黑客攻击导致的，需要读服务做参数校验和缓存不存在的键来避免。缓存雪崩和缓存击都是代码不合理导致的，我们在实现时需要注意：1. 避免大量键在同一时间过期，2. 对热键做特殊处，避免大量读请求同时访问数据库。