27-缓存被污染了，该怎么办？

你好，我是蒋德钧。

我们应用Redis缓存时，如果能缓存会被反复访问的数据，那就能加速业务应用的访问。但是，如果发生了缓存污染，那么，缓存对业务应用的加速作用就减少了。

那什么是缓存污染呢？在一些场景下，有些数据被访问的次数非常少，甚至只会被访问一次。当这些数据服务完访问请求后，如果还继续留存在缓存中的话，就只会白白占用缓存空间。这种情况，就是缓存污染。

当缓存污染不严重时，只有少量数据占据缓存空间，此时，对缓存系统的影响不大。但是，缓存污染一旦变得严重后，就会有大量不再访问的数据滞留在缓存中。如果这时数据占满了缓存空间，我们再往缓存中写入新数据时，就需要先把这些数据逐步淘汰出缓存，这就会引入额外的操作时间开销，进而会影响应用的性能。

今天，我们就来看看如何解决缓存污染问题。

如何解决缓存污染问题？

要解决缓存污染，我们也能很容易想到解决方案，那就是得把不会再被访问的数据筛选出来并淘汰掉。这样就不用等到缓存被写满以后，再逐一淘汰旧数据之后，才能写入新数据了。而哪些数据能留存在缓存中，是由缓存的淘汰策略决定的。

到这里，你还记得咱们在第24讲一起学习的8种数据淘汰策略吗？它们分别是noeviction、volatile-random、volatile-ttl、volatile-lru、volatile-lfu、allkeys-lru、allkeys-random和allkeys-lfu策略。

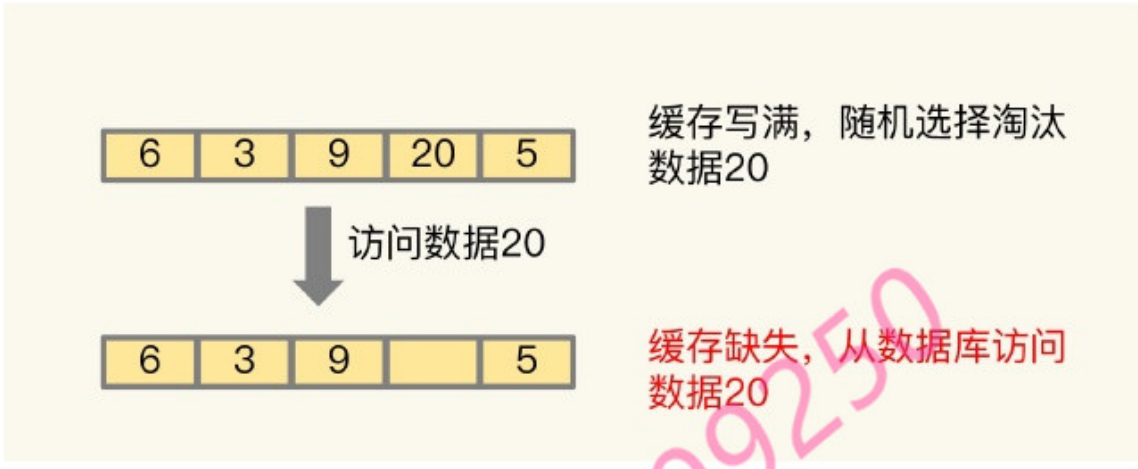
在这8种策略中，noeviction策略是不会进行数据淘汰的。所以，它肯定不能用来解决缓存污染问题。其他的7种策略，都会按照一定的规则来淘汰数据。这里有个关键词是“一定的规则”，那么问题来了，不同的规则对于解决缓存污染问题，是否都有效呢？接下来，我们就一一分析下。

因为LRU算法是我们在缓存数据淘汰策略中广泛应用的算法，所以我们先分析其他策略，然后单独分析淘汰策略使用LRU算法的情况，最后再学习下LFU算法用于淘汰策略时，对缓存污染的应对措施。使用LRU算法和LFU算法的策略各有两种（volatile-lru和allkeys-lru，以及volatile-lfu和allkeys-lfu），为了便于理解，接下来我会统一把它们叫作LRU策略和LFU策略。

首先，我们看下**volatile-random和allkeys-random**这两种策略。它们都是采用随机挑选数据的方式，来筛选即将被淘汰的数据。

既然是随机挑选，那么Redis就不会根据数据的访问情况来筛选数据。如果被淘汰的数据又被访问了，就会发生缓存缺失。也就是说，应用需要到后端数据库中访问这些数据，降低了应用的请求响应速度。所以，volatile-random和allkeys-random策略，在避免缓存污染这个问题上的效果非常有限。

我给你举个例子吧。如下图所示，假设我们配置Redis缓存使用allkeys-random淘汰策略，当缓存写满时，allkeys-random策略随机选择了数据20进行淘汰。不巧的是，数据20紧接着又被访问了，此时，Redis就会发生了缓存缺失。



我们继续看**volatile-ttl**策略是否能有效应对缓存污染。volatile-ttl针对的是设置了过期时间的数据，把这些数据中剩余存活时间最短的筛选出来并淘汰掉。

虽然volatile-ttl策略不再是随机选择淘汰数据了，但是剩余存活时间并不能直接反映数据再次访问的情况。所以，按照volatile-ttl策略淘汰数据，和按随机方式淘汰数据类似，也可能出现数据被淘汰后，被再次访问导致的缓存缺失问题。

这时，你可能会想到一种例外的情况：业务应用在给数据设置过期时间的时候，就明确知道数据被再次访问的情况，并根据访问情况设置过期时间。此时，Redis按照数据的剩余最短存活时间进行筛选，是可以把不会再被访问的数据筛选出来的，进而避免缓存污染。例如，业务部门知道数据被访问的时长就是一个小时，并把数据的过期时间设置为一个小时后。这样一来，被淘汰的数据的确是不会再被访问了。

讲到这里，我们先小结下。除了在明确知道数据被再次访问的情况下，volatile-ttl可以有效避免缓存污染。在其他情况下，volatile-random、allkeys-random、volatile-ttl这三种策略并不能应对缓存污染问题。

接下来，我们再分别分析下LRU策略，以及Redis 4.0后实现的LFU策略。LRU策略会按照数据访问的时效性，来筛选即将被淘汰的数据，应用非常广泛。在第24讲，我们已经学习了Redis是如何实现LRU策略的，所以接下来我们就重点看下它在解决缓存污染问题上的效果。

LRU缓存策略

我们先复习下LRU策略的核心思想：如果一个数据刚刚被访问，那么这个数据肯定是热数据，还会被再次访问。

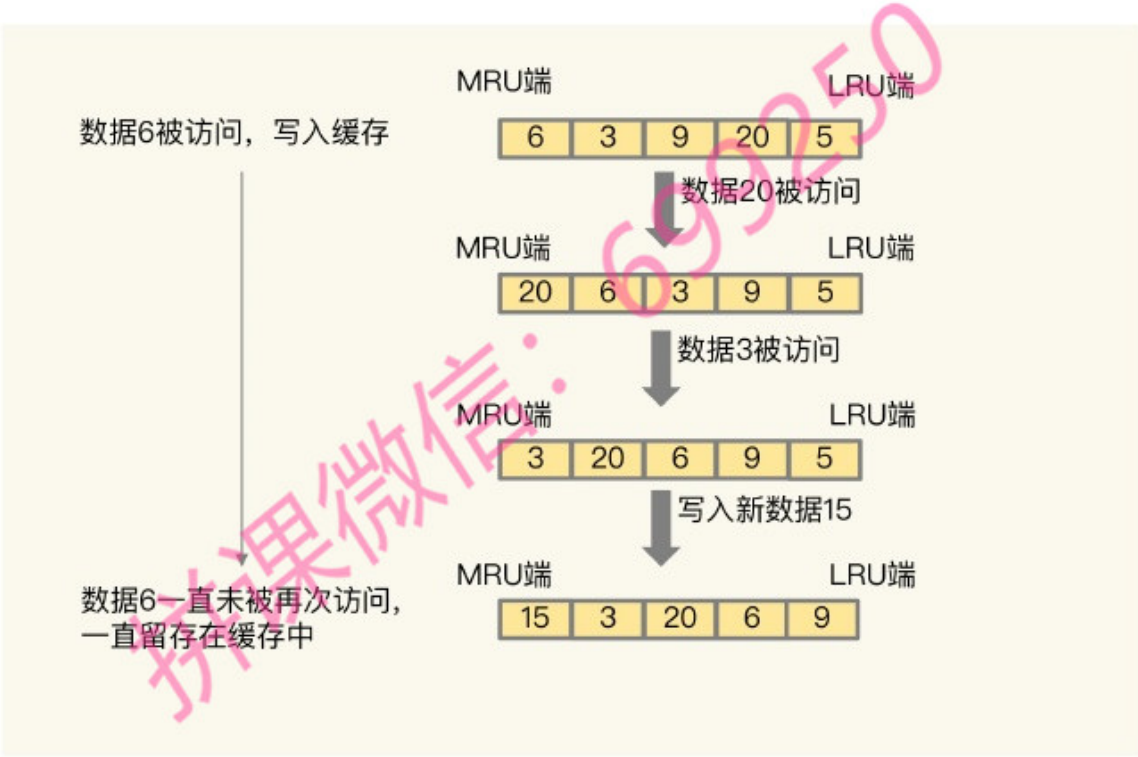
按照这个核心思想，Redis中的LRU策略，会在每个数据对应的RedisObject结构体中设置一个lru字段，用来记录数据的访问时间戳。在进行数据淘汰时，LRU策略会在候选数据集中淘汰掉lru字段值最小的数据（也就是访问时间最久的数据）。

所以，在数据被频繁访问的业务场景中，LRU策略的确能有效留存访问时间最近的数据。而且，因为留存的这些数据还会被再次访问，所以又可以提升业务应用的访问速度。

但是，也正是**因为只看数据的访问时间，使用LRU策略在处理扫描式单次查询操作时，无法解决缓存污染**。所谓的扫描式单次查询操作，就是指应用对大量的数据进行一次全体读取，每个数据都会被读取，而且只会被读取一次。此时，因为这些被查询的数据刚刚被访问过，所以lru字段值都很大。

在使用LRU策略淘汰数据时，这些数据会留存在缓存中很长一段时间，造成缓存污染。如果查询的数据量很大，这些数据占满了缓存空间，却又不会服务新的缓存请求，此时，再有新数据要写入缓存的话，还是需要先把这些旧数据替换出缓存才行，这会影响缓存的性能。

为了方便你理解，我给你举个例子。如下图所示，数据6被访问后，被写入Redis缓存。但是，在此之后，数据6一直没有被再次访问，这就导致数据6滞留在缓存中，造成了污染。



所以，对于采用了LRU策略的Redis缓存来说，扫描式单次查询会造成缓存污染。为了应对这类缓存污染问题，Redis从4.0版本开始增加了LFU淘汰策略。

与LRU策略相比，LFU策略中会从两个维度来筛选并淘汰数据：一是，数据访问的时效性（访问时间离当前时间的远近）；二是，数据的被访问次数。

那Redis的LFU策略是怎么实现的，又是如何解决缓存污染问题的呢？我们来看一下。

LFU缓存策略的优化

LFU缓存策略是在LRU策略基础上，为每个数据增加了一个计数器，来统计这个数据的访问次数。当使用LFU策略筛选淘汰数据时，首先会根据数据的访问次数进行筛选，把访问次数最低的数据淘汰出缓存。如果两个数据的访问次数相同，LFU策略再比较这两个数据的访问时效性，把距离上一次访问时间更久的数据淘汰出缓存。

和那些被频繁访问的数据相比，扫描式单次查询的数据因为不会被再次访问，所以它们的访问次数不会再增加。因此，LFU策略会优先把这些访问次数低的数据淘汰出缓存。这样一来，LFU策略就可以避免这些数据对缓存造成污染了。

那么，LFU策略具体又是如何实现的呢？既然LFU策略是在LRU策略上做的优化，那它们的实现必定有些关系。所以，我们就再复习下第24讲学习过的LRU策略的实现。

为了避免操作链表的开销，Redis在实现LRU策略时使用了两个近似方法：

- Redis是用RedisObject结构来保存数据的，RedisObject结构中设置了一个lru字段，用来记录数据的访问时间戳；
- Redis并没有为所有的数据维护一个全局的链表，而是通过随机采样方式，选取一定数量（例如10个）的数据放入候选集合，后续在候选集合中根据lru字段值的大小进行筛选。

在此基础上，**Redis在实现LFU策略的时候，只是把原来24bit大小的lru字段，又进一步拆分成了两部分**。

1. ldt值：lru字段的前16bit，表示数据的访问时间戳；
2. counter值：lru字段的后8bit，表示数据的访问次数。

总结一下：当LFU策略筛选数据时，Redis会在候选集合中，根据数据lru字段的后8bit选择访问次数最少的数据进行淘汰。当访问次数相同时，再根据lru字段的前16bit值大小，选择访问时间最久远的数据进行淘汰。

到这里，还没结束，**Redis只使用了8bit记录数据的访问次数，而8bit记录的最大值是255**，这样可以吗？

在实际应用中，一个数据可能会被访问成千上万次。如果每被访问一次，counter值就加1的话，那么，只要访问次数超过了255，数据的counter值就一样了。在进行数据淘汰时，LFU策略就无法很好地区分并筛选这些数据，反而还可能会把不怎么访问的数据留存在了缓存中。

我们一起来看个例子。

假设第一个数据A的累计访问次数是256，访问时间戳是202010010909，所以它的counter值为255，而第二个数据B的累计访问次数是1024，访问时间戳是202010010810。如果counter值只能记录到255，那么数据B的counter值也是255。

此时，缓存写满了，Redis使用LFU策略进行淘汰。数据A和B的counter值都是255，LFU策略再比较A和B的访问时间戳，发现数据B的上一次访问时间早于A，就会把B淘汰掉。但其实数据B的访问次数远大于数据A，很可能会被再次访问。这样一来，使用LFU策略来淘汰数据就不合适了。

的确，Redis也注意到了这个问题。因此，**在实现LFU策略时，Redis并没有采用数据每被访问一次，就给对应的counter值加1的计数规则，而是采用了一个更优化的计数规则**。

简单来说，LFU策略实现的计数规则是：每当数据被访问一次时，首先，用计数器当前的值乘以配置项lfu_log_factor再加1，再取其倒数，得到一个p值；然后，把这个p值和一个取值范围在（0，1）间的随机数r值比大小，只有p值大于r值时，计数器才加1。

下面这段Redis的部分源码，显示了LFU策略增加计数器值的计算逻辑。其中，baseval是计数器当前的值。计数器的初始值默认是5（由代码中的LFU_INIT_VAL常量设置），而不是0，这样可以避免数据刚被写入缓存，就因为访问次数少而被立即淘汰。

```
double r = (double)rand()/RAND_MAX;
...
```

```
if (r < p) counter++;
```

使用了这种计算规则后，我们可以通过设置不同的lfu_log_factor配置项，来控制计数器值增加的速度，避免counter值很快就到255了。

为了更进一步说明LFU策略计数器递增的效果，你可以看下下面这张表。这是Redis官网上提供的一张表，它记录了当lfu_log_factor取不同值时，在不同的实际访问次数情况下，计数器的值是如何变化的。

lfu_log_factor	100 hits	1K hits	100K hits	1M hits	10M hits
0	104	255	255	255	255
1	18	49	255	255	255
10	10	18	142	255	255
100	8	11	49	143	255

可以看到，当lfu_log_factor取值为1时，实际访问次数为100K后，counter值就达到255了，无法再区分实际访问次数更多的数据了。而当lfu_log_factor取值为100时，当实际访问次数为10M时，counter值才达到255，此时，实际访问次数小于10M的不同数据都可以通过counter值区分出来。

正是因为使用了非线性递增的计数器方法，即使缓存数据的访问次数成千上万，LFU策略也可以有效地区分不同的访问次数，从而进行合理的数据筛选。从刚才的表中，我们可以看到，当lfu_log_factor取值为10时，百、千、十万级别的访问次数对应的counter值已经有明显的区分了，所以，我们在应用LFU策略时，一般可以将lfu_log_factor取值为10。

前面我们也提到了，应用负载的情况是很复杂的。在一些场景下，有些数据在短时间内被大量访问后就不会再被访问了。那么再按照访问次数来筛选的话，这些数据会被留存在缓存中，但不会提升缓存命中率。为此，Redis在实现LFU策略时，还设计了一个counter值的衰减机制。

简单来说，LFU策略使用衰减因子配置项lfu_decay_time来控制访问次数的衰减。LFU策略会计算当前时间和数据最近一次访问时间的差值，并把这个差值换算成以分钟为单位。然后，LFU策略再把这个差值除以lfu_decay_time值，所得的结果就是数据counter要衰减的值。

简单举个例子，假设lfu_decay_time取值为1，如果数据在N分钟内没有被访问，那么它的访问次数就要减N。如果lfu_decay_time取值更大，那么相应的衰减值会变小，衰减效果也会减弱。所以，如果业务应用中有短时高频访问的数据的话，建议把lfu_decay_time值设置为1，这样一来，LFU策略在它们不再被访问后，会较快地衰减它们的访问次数，尽早把它们从缓存中淘汰出去，避免缓存污染。

小结

今天这节课，我们学习的是“如何解决缓存污染”这个问题。

缓存污染问题指的是留存在缓存中的数据，实际不会被再次访问了，但是又占据了缓存空间。如果这样的数据体量很大，甚至占满了缓存，每次有新数据写入缓存时，还需要把这些数据逐步淘汰出缓存，就会增加缓存操作的时间开销。

因此，要解决缓存污染问题，最关键的技术点就是能识别出这些只访问一次或是访问次数很少的数据，在淘汰数据时，优先把它们筛选出来并淘汰掉。因为noviction策略不涉及数据淘汰，所以这节课，我们就从能否有效解决缓存污染这个维度，分析了Redis的其他7种数据淘汰策略。

volatile-random和allkeys-random是随机选择数据进行淘汰，无法把不再访问的数据筛选出来，可能会造成缓存污染。如果业务层明确知道数据的访问时长，可以给数据设置合理的过期时间，再设置Redis缓存使用volatile-ttl策略。当缓存写满时，剩余存活时间最短的数据就会被淘汰出缓存，避免滞留在缓存中，造成污染。

当我们使用LRU策略时，由于LRU策略只考虑数据的访问时效，对于只访问一次的数据来说，LRU策略无法很快将其筛选出来。而LFU策略在LRU策略基础上进行了优化，在筛选数据时，首先会筛选并淘汰访问次数少的数据，然后针对访问次数相同的数据，再筛选并淘汰访问时间最久远的数据。

在具体实现上，相对于LRU策略，Redis只是把原来24bit大小的lru字段，又进一步拆分成了16bit的ldt和8bit的counter，分别用来表示数据的访问时间戳和访问次数。为了避开8bit最大只能记录255的限制，LFU策略设计使用非线性增长的计数器来表示数据的访问次数。

在实际业务应用中，LRU和LFU两个策略都有应用。LRU和LFU两个策略关注的数据访问特征各有侧重，LRU策略更加关注数据的时效性，而LFU策略更加关注数据的访问频次。通常情况下，实际应用的负载具有较好的时间局部性，所以LRU策略的应用会更加广泛。但是，在扫描式查询的应用场景中，LFU策略就可以很好地应对缓存污染问题了，建议你优先使用。

此外，如果业务应用中有短时高频访问的数据，除了LFU策略本身会对数据的访问次数进行自动衰减以外，我再给你个小建议：你可以优先使用volatile-lfu策略，并根据这些数据的访问时限设置它们的过期时间，以免它们留存在缓存中造成污染。

每课一问

按照惯例，我给你提个小问题。使用了LFU策略后，你觉得缓存还会被污染吗？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事。我们下节课见。

精选留言：

- Kaito 2020-10-19 11:48:30
  使用了 LFU 策略后，缓存还会被污染吗？

  我觉得还是有被污染的可能性，被污染的概率取决于LFU的配置，也就是lfu-log-factor和lfu-decay-time参数。

  1、根据LRU counter计数规则可以得出，counter递增的概率取决于2个因素：

  a) counter值越大，递增概率越低
  b) lfu-log-factor设置越大，递增概率越低

  所以当访问次数counter越来越大时，或者lfu-log-factor参数配置过大时，counter递增的概率都会越来越低，这种情况下可能会导致一些key虽然访问次数较高，但是counter值却递增困难，进而导致这些访

问频次较高的key却优先被淘汰掉了。

另外由于counter在递增时，有随机数比较的逻辑，这也会存在一定概率导致访问频次低的key的counter反而大于访问频次高的key的counter情况出现。

2、如果lfu-decay-time配置过大，则counter衰减会变慢，也会导致数据淘汰发生推迟的情况。

3、另外，由于LRU的ldt字段只采用了16位存储，其精度是分钟级别的，在counter衰减时可能会产生同一分钟内，后访问的key比先访问的key的counter值优先衰减，进而先被淘汰掉的情况。

可见，Redis实现的LFU策略，也是近似的LFU算法。Redis在实现时，权衡了内存使用、性能开销、LFU的正确性，通过复用并拆分lru字段的方式，配合算法策略来实现近似的结果，虽然会有一定概率的偏差，但在内存数据库这种场景下，已经做得足够好了。 [14赞]

- test 2020-10-19 09:47:37
1.选候选集：volatile前缀的是设置了过期时间的key，all前缀的是全部key；
2.算法：lru是最近最少使用，lfu是最近最不频繁使用，ttl是距离到期时间长短，ramdon是随机；
2.1 lru是自带了访问时间
2.2 lfu是带了访问次数，但是因为只有八位保存所以不是每访问一次就+1，而是每次原来访问次数乘以lfu_log_factor，加一再倒数，看是否大于随机值，大于则+1：double r = (double)rand()/RAND_MAX;double p = 1.0/(baseval*server.lfu_log_factor+1);if (r < p) counter++;
还有衰减机制，由lfu_decay_time控制，已过去n个lfu_decay_time，则将idle time加n。
3.淘汰规则：每次sample n个key，比如10个，放到pool里面，淘汰idle时间最长的key。再次淘汰的时候，继续抽样10个key，只把小于原pool中某个idle时间的key插入进去，即如果抽样出来的key比pool中所有key的idle时间都小，则不插入。然后淘汰pool中idle小的，保持pool在10个；
[2赞]

- 甜宝仙女的专属饲养员 2020-10-19 09:14:18
又刷新了对lru和lfu的认知，这种突然打开知识盲区的天窗的感觉就如同一盆冷水，把我从地铁上这种迷迷糊糊的状态给满血复活 [1赞]

- 写点啥呢 2020-10-19 08:58:23
关于计数衰减想请问老师，它发生的时机是在缓存被访问到还是Redis会定时刷新所有缓存计数进行衰减呢？对这两种衰减时机感觉都有不足，再次访问时候衰减能维持较低的性能损耗但是对于短期热点数据如果不会被访问那么计数就不准确还会导致污染问题。如果是全量定时刷新那么又会带来很多性能损耗。 [1赞]

- Bug? Feature! 2020-10-19 09:33:40
LRU 策略更加关注数据的时效性，而 LFU 策略更加关注数据的访问频次。

- yeek 2020-10-19 08:59:46
偏极端情况下，数据短期内被高频访问，且计数器达到了很大值，且计数器的衰减设置的很大，导致衰减很慢，此时该数据就可能在缓存中长期驻留。

从长期来看，我觉得应该是避免频繁执行数据淘汰，否则会影响redis的效率，较好的做法应该是监控redis服务器的内存情况，以及相应的报警机制，定期统计redis中的key分布情况，交由使用方check数据合理性，以检验程序中对redis数据设置的过期时间，访问失效等是否合理。

- Litt1eQ 2020-10-19 07:49:00

率就会降低 甚至不会被访问到 这样热点数据只能通过计数衰减淘汰 有可能这些数据成为污染缓存的数据