24-替换策略：缓存满了怎么办？

你好，我是蒋德钧。

Redis缓存使用内存来保存数据，避免业务应用从后端数据库中读取数据，可以提升应用的响应速度。那么，如果我们把所有要访问的数据都放入缓存，是不是一个很好的设计选择呢？其实，这样做的性价比反而不高。

举个例子吧。MySQL中有1TB的数据，如果我们使用Redis把这1TB的数据都缓存起来，虽然应用都能在内存中访问数据了，但是，这样配置并不合理，因为性价比很低。一方面，1TB内存的价格大约是3.5万元，而1TB磁盘的价格大约是1000元。另一方面，数据访问都是有局部性的，也就是我们通常所说的“八二原理”，80%的请求实际只访问了20%的数据。所以，用1TB的内存做缓存，并没有必要。

为了保证较高的性价比，缓存的空间容量必然要小于后端数据库的数据总量。不过，内存大小毕竟有限，随着要缓存的数据量越来越大，有限的缓存空间不可避免地会被写满。此时，该怎么办呢？

解决这个问题就涉及到缓存系统的一个重要机制，即**缓存数据的淘汰机制**。简单来说，数据淘汰机制包括两步：第一，根据一定的策略，筛选出对应用访问来说“不重要”的数据；第二，将这些数据从缓存中删除，为新来的数据腾出空间，

这节课上，我就来和你聊聊缓存满了之后的数据淘汰机制。通常，我们也把它叫作缓存替换机制，同时还会讲到一系列选择淘汰数据的具体策略。了解了数据淘汰机制和相应策略，我们才可以选择合理的Redis配置，提高缓存命中率，提升应用的访问性能。

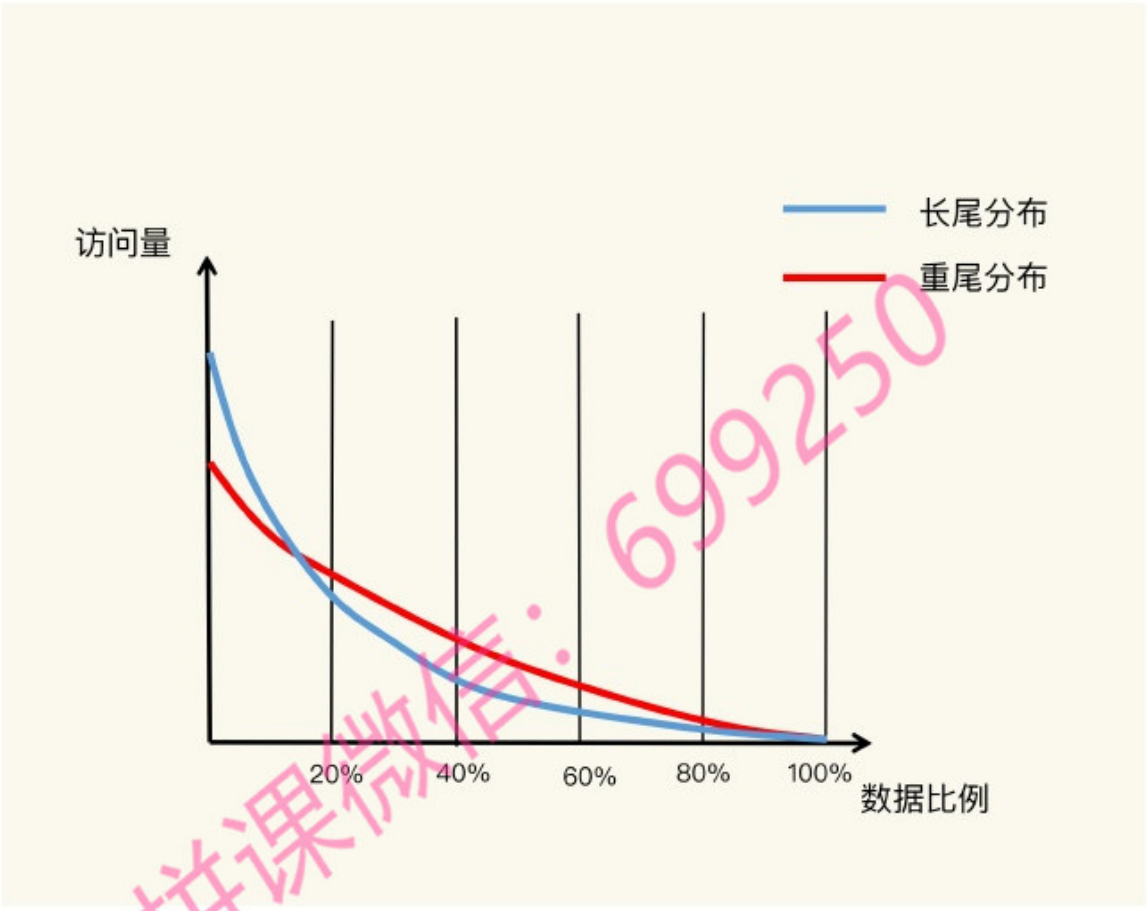
不过，在学习淘汰策略之前，我们首先要知道设置缓存容量的依据和方法。毕竟，在实际使用缓存时，我们需要决定用多大的空间来缓存数据。

设置多大的缓存容量合适？

缓存容量设置得是否合理，会直接影响到使用缓存的性价比。我们通常希望以最小的代价去获得最大的收益，所以，把昂贵的内存资源用在关键地方就非常重要了。

就像我刚才说的，实际应用中的数据访问是具有局部性的。下面有一张图，图里有红、蓝两条线，显示了不同比例数据贡献的访问量情况。蓝线代表了“八二原理”表示的数据局部性，而红线则表示在当前应用负载下，数据局部性的变化。

我们先看看蓝线。它表示的就是“八二原理”，有20%的数据贡献了80%的访问了，而剩余的数据虽然体量很大，但只贡献了20%的访问量。这80%的数据在访问量上就形成了一条长长的尾巴，我们也称为“长尾效应”。



所以，如果按照“八二原理”来设置缓存空间容量，也就是把缓存空间容量设置为总数据量的20%的话，就有可能拦截到80%的访问。

为什么说是“有可能”呢？这是因为，“八二原理”是对大量实际应用的数据访问情况做了统计后，得出的一个统计学意义上的数据量和访问量的比例。具体到某一个应用来说，数据访问的规律会和具体的业务场景有关。对于最常被访问的20%的数据来说，它们贡献的访问量，既有可能超过80%，也有可能不到80%。

我们再通过一个电商商品的场景，来说明下“有可能”这件事儿。一方面，在商品促销时，热门商品的信息可能只占到总商品数据信息量的5%，而这些商品信息承载的可能是超过90%的访问请求。这时，我们只要缓存这5%的数据，就能获得很好的性能收益。另一方面，如果业务应用要对所有商品信息进行查询统计，这时候，即使按照“八二原理”缓存了20%的商品数据，也不能获得很好的访问性能，因为80%的数据仍然需要从后端数据库中获取。

接下来，我们再看看数据访问局部性示意图中的红线。近年来，有些研究人员专门对互联网应用（例如视频播放网站）中，用户请求访问内容的分布情况做过分析，得到了这张图中的红线。

在这条红线上，80%的数据贡献的访问量，超过了传统的长尾效应中80%数据能贡献的访问量。原因在于，用户的个性化需求越来越多，在一个业务应用中，不同用户访问的内容可能差别很大，所以，用户请求的数据和它们贡献的访问量比例，不再具备长尾效应中的“八二原理”分布特征了。也就是说，20%的数据可能贡献不了80%的访问，而剩余的80%数据反而贡献了更多的访问量，我们称之为重尾效应。

正是因为20%的数据不一定能贡献80%的访问量，我们不能简单地按照“总数据量的20%”来设置缓存最大空间容量。在实践过程中，我看到过的缓存容量占总数据量的比例，从5%到40%的都有。这个容量规划不

这其实也是我一直在和你分享的经验，系统的设计选择是一个权衡的过程：大容量缓存是能带来性能加速的收益，但是成本也会更高，而小容量缓存不一定就起不到加速访问的效果。一般来说，**我会建议把缓存容量设置为总数据量的15%到30%，兼顾访问性能和内存空间开销**。

对于Redis来说，一旦确定了缓存最大容量，比如4GB，你就可以使用下面这个命令来设定缓存的大小了：

```
CONFIG SET maxmemory 4gb
```

不过，**缓存被写满是不可避免的**。即使你精挑细选，确定了缓存容量，还是要面对缓存写满时的替换操作。缓存替换需要解决两个问题：决定淘汰哪些数据，如何处理那些被淘汰的数据。

接下来，我们就来学习下，Redis中的数据淘汰策略。

Redis缓存有哪些淘汰策略?

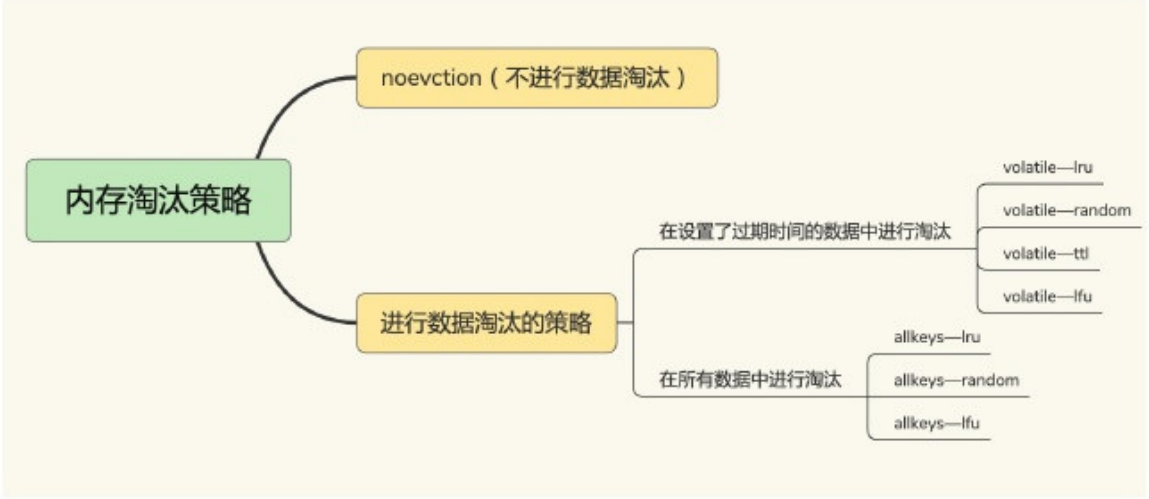
Redis 4.0之前一共实现了6种内存淘汰策略，在4.0之后，又增加了2种策略。我们可以按照是否会进行数据淘汰把它们分成两类：

- 不进行数据淘汰的策略，只有noeviction这一种。
- 会进行淘汰的7种其他策略。

会进行淘汰的7种策略，我们可以再进一步根据淘汰候选数据集的范围把它们分成两类：

- 在设置了过期时间的数据中进行淘汰，包括volatile-random、volatile-ttl、volatile-lru、volatile-lfu（Redis 4.0后新增）四种。
- 在所有数据范围内进行淘汰，包括allkeys-lru、allkeys-random、allkeys-lfu（Redis 4.0后新增）三种。

我把这8种策略的分类，画到了一张图里：



下面我就来具体解释下各个策略。

默认情况下，Redis在使用的内存空间超过maxmemory值时，并不会淘汰数据，也就是设定的**noeviction策略**。对应到Redis缓存，也就是指，一旦缓存被写满了，再有写请求来时，Redis不再提供服务，而是直接返回错误。Redis用作缓存时，实际的数据集通常都是大于缓存容量的，总会有新的数据要写入缓存，这个策略本身不淘汰数据，也就不会腾出新的缓存空间，我们不把它用在Redis缓存中。

我们再分析下volatile-random、volatile-ttl、volatile-lru和volatile-lfu这四种淘汰策略。它们筛选的候选数据范围，被限制在已经设置了过期时间的键值对上。也正因为此，即使缓存没有写满，这些数据如果过期了，也会被删除。

例如，我们使用EXPIRE命令对一批键值对设置了过期时间后，无论是这些键值对的过期时间是快到了，还是Redis的内存使用量达到了maxmemory阈值，Redis都会进一步按照volatile-ttl、volatile-random、volatile-lru、volatile-lfu这四种策略的具体筛选规则进行淘汰。

- volatile-ttl在筛选时，会针对设置了过期时间的键值对，根据过期时间的先后进行删除，越早过期的越先被删除。
- volatile-random就像它的名称一样，在设置了过期时间的键值对中，进行随机删除。
- volatile-lru会使用LRU算法筛选设置了过期时间的键值对。
- volatile-lfu会使用LFU算法选择设置了过期时间的键值对。

可以看到，volatile-ttl和volatile-random筛选规则比较简单，而volatile-lru因为涉及了LRU算法，所以我会在分析allkeys-lru策略时再详细解释。volatile-lfu使用了LFU算法，我会在第26讲中具体解释，现在你只需要知道，它是在LRU算法的基础上，同时考虑了数据的访问时效性和数据的访问次数，可以看作是对淘汰策略的优化。

相对于volatile-ttl、volatile-random、volatile-lru、volatile-lfu这四种策略淘汰的是设置了过期时间的数据，allkeys-lru、allkeys-random、allkeys-lfu这三种淘汰策略的备选淘汰数据范围，就扩大到了所有键值对，无论这些键值对是否设置了过期时间。它们筛选数据进行淘汰的规则是：

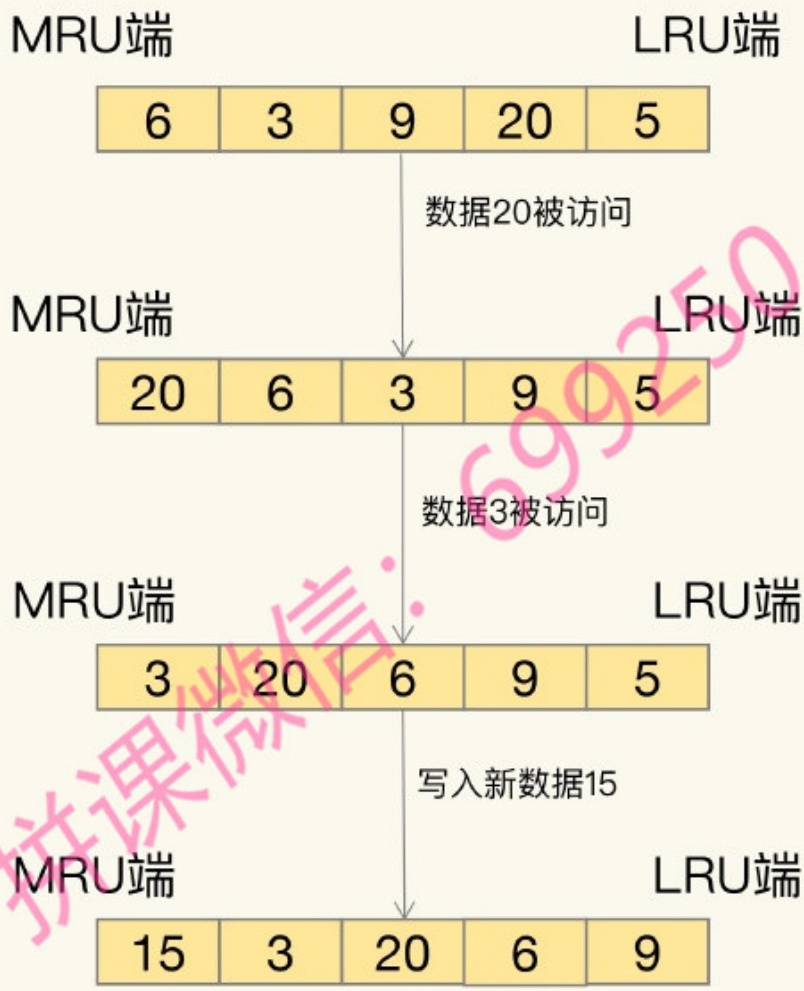
- allkeys-random策略，从所有键值对中随机选择并删除数据；
- allkeys-lru策略，使用LRU算法在所有数据中进行筛选。
- allkeys-lfu策略，使用LFU算法在所有数据中进行筛选。

这也就是说，如果一个键值对被删除策略选中了，即使它的过期时间还没到，也需要被删除。当然，如果它的过期时间到了但未被策略选中，同样也会被删除。

接下来，我们就看看volatile-lru和allkeys-lru策略都用到的LRU算法吧。LRU算法工作机制并不复杂，我们一起学习下。

LRU算法的全称是Least Recently Used，从名字上就可以看出，这是按照最近最少使用的原则来筛选数据，最不常用的数据会被筛选出来，而最近频繁使用的数据会留在缓存中。

那具体是怎么筛选的呢？LRU会把所有的数据组织成一个链表，链表的头和尾分别表示MRU端和LRU端，分别代表最近最常使用的数据和最近最不常用的数据。我们看一个例子。



我们现在有数据6、3、9、20、5。如果数据20和3被先后访问，它们都会从现有的链表位置移到MRU端，而链表中在它们之前的数据则相应地往后移一位。因为，LRU算法选择删除数据时，都是从LRU端开始，所以把刚刚被访问的数据移到MRU端，就可以让它们尽可能地留在缓存中。

如果有一个新数据15要被写入缓存，但此时已经没有缓存空间了，也就是链表没有空余位置了，那么，LRU算法做两件事：

1. 数据15是刚被访问的，所以它会被放到MRU端；
2. 算法把LRU端的数据5从缓存中删除，相应的链表中就没有数据5的记录了。

其实，LRU算法背后的想法非常朴素：它认为刚刚被访问的数据，肯定还会被再次访问，所以就把它放在MRU端；长久不访问的数据，肯定就不会再被访问了，所以就让它逐渐后移到LRU端，在缓存满时，就优先删除它。

不过，LRU算法在实际实现时，需要用链表管理所有的缓存数据，这会**带来额外的空间开销**。而且，当有数据被访问时，需要在链表上把该数据移动到MRU端，如果有大量数据被访问，就会带来很多链表移动操作，会很耗时，进而会降低Redis缓存性能。

所以，在Redis中，LRU算法被做了简化，以减轻数据淘汰对缓存性能的影响。具体来说，Redis默认会记录每个数据的最近一次访问的时间戳（由键值对数据结构RedisObject中的lru字段记录）。然后，Redis在决定淘汰的数据时，第一次会随机选出N个数据，把它们作为一个候选集合。接下来，Redis会比较这N个数据的lru字段，把lru字段值最小的数据从缓存中淘汰出去。

Redis提供了一个配置参数maxmemory-samples，这个参数就是Redis选出的数据个数N。例如，我们执行如下命令，可以让Redis选出100个数据作为候选数据集：

```
CONFIG SET maxmemory-samples 100
```

当需要再次淘汰数据时，Redis需要挑选数据进入第一次淘汰时创建的候选集合。这儿的挑选标准是：**能进入候选集合的数据的lru字段值必须小于候选集合中最小的lru值**。当有新数据进入候选数据集后，如果候选数据集中的数据个数达到了maxmemory-samples，Redis就把候选数据集中lru字段值最小的数据淘汰出去。

这样一来，Redis缓存不用为所有的数据维护一个大链表，也不用在每次数据访问时都移动链表项，提升了缓存的性能。

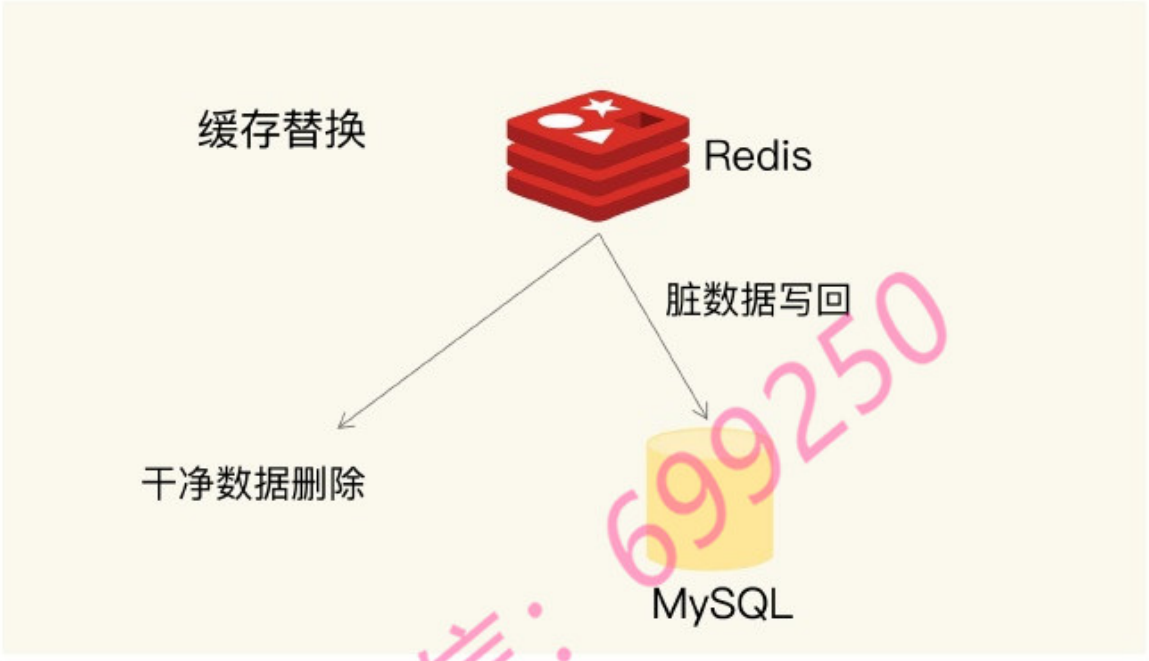
好了，到这里，我们就学完了除了使用LFU算法以外的5种缓存淘汰策略，我再给你三个使用建议。

- **优先使用allkeys-lru策略**。这样，可以充分利用LRU这一经典缓存算法的优势，把最近最常访问的数据留在缓存中，提升应用的访问性能。如果你的业务数据中有明显的冷热数据区分，我建议你使用allkeys-lru策略。
- 如果业务应用中的数据访问频率相差不大，没有明显的冷热数据区分，建议使用allkeys-random策略，随机选择淘汰的数据就行。
- **如果你的业务中有置顶的需求**，比如置顶新闻、置顶视频，那么，可以使用volatile-lru策略，同时不给这些置顶数据设置过期时间。这样一来，这些需要置顶的数据一直不会被删除，而其他数据会在过期时根据LRU规则进行筛选。

一旦被淘汰的数据被选定后，Redis怎么处理这些数据呢？这就要说到缓存替换时的具体操作了。

如何处理被淘汰的数据？

一般来说，一旦被淘汰的数据选定后，如果这个数据是干净数据，那么我们就直接删除；如果这个数据是脏数据，我们需要把它写回数据库，如下图所示：



那怎么判断一个数据到底是干净的还是脏的呢？

干净数据和脏数据的区别就在于，和最初从后端数据库里读取时的值相比，有没有被修改过。干净数据一直没有被修改，所以后端数据库里的数据也是最新值。在替换时，它可以被直接删除。

而脏数据就是曾经被修改过的，已经和后端数据库中保存的数据不一致了。此时，如果不把脏数据写回到数据库中，这个数据的最新值就丢失了，就会影响应用的正常使用。

这么一来，缓存替换既腾出了缓存空间，用来缓存新的数据，同时，将脏数据写回数据库，也保证了最新数据不会丢失。

不过，对于Redis来说，它决定了被淘汰的数据后，会把它们删除。即使淘汰的数据是脏数据，Redis也不会把它们写回数据库。所以，我们在使用Redis缓存时，如果数据被修改了，需要在数据修改时就将它写回数据库。否则，这个脏数据被淘汰时，会被Redis删除，而数据库里也没有最新的数据了。

小结

在这节课上，我围绕着“缓存满了该怎么办”这一问题，向你介绍了缓存替换时的数据淘汰策略，以及被淘汰数据的处理方法。

Redis 4.0版本以后一共提供了8种数据淘汰策略，从淘汰数据的候选集范围来看，我们有两种候选范围：一种是所有数据都是候选集，一种是设置了过期时间的数据是候选集。另外，无论是面向哪种候选数据集进行淘汰数据选择，我们都有三种策略，分别是随机选择，根据LRU算法选择，以及根据LFU算法选择。当然，当面向设置了过期时间的数据集选择淘汰数据时，我们还可以根据数据离过期时间的远近来决定。

一般来说，缓存系统对于选定的被淘汰数据，会根据其是干净数据还是脏数据，选择直接删除还是写回数据库。但是，在Redis中，被淘汰数据无论干净与否都会被删除，所以，这是我们在使用Redis缓存时要特别注意的：当数据修改成为脏数据时，需要在数据库中也把数据修改过来。

选择哪种缓存策略是值得我们多加琢磨的，它在筛选数据方面是否能筛选出可能被再次访问的数据，直接决

定了缓存效率的高与低。

很简单的一个对比，如果我们使用随机策略，刚筛选出来的要被删除的数据可能正好又被访问了，此时应用就只能花费几毫秒从数据库中读取数据了。而如果使用LRU策略，被筛选出来的数据往往是经过时间验证了，如果在一段时间内一直没有访问，本身被再次访问的概率也很低了。

所以，我给你的建议是，先根据是否有始终会被频繁访问的数据（例如置顶消息），来选择淘汰数据的候选集，也就是决定是针对所有数据进行淘汰，还是针对设置了过期时间的数据进行淘汰。候选数据集范围选定后，建议优先使用LRU算法，也就是，allkeys-lru或volatile-lru策略。

当然，设置缓存容量的大小也很重要，我的建议是：结合实际应用的数据总量、热数据的体量，以及成本预算，把缓存空间大小设置在总数据量的15%到30%这个区间就可以。

每课一问

按照惯例，我给你提一个小问题。这节课，我向你介绍了Redis缓存在应对脏数据时，需要在数据修改的同时，也把它写回数据库，针对我们上节课介绍的缓存读写模式：只读缓存，以及读写缓存中的两种写回策略，请你思考下，Redis缓存对应哪一种或哪几种模式？

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 2020-10-12 00:03:31
  Redis在用作缓存时，使用只读缓存或读写缓存的哪种模式？

  1、只读缓存模式：每次修改直接写入后端数据库，如果Redis缓存不命中，则什么都不用操作，如果Redis缓存命中，则删除缓存中的数据，待下次读取时从后端数据库中加载最新值到缓存中。

  2、读写缓存模式+同步直写策略：由于Redis在淘汰数据时，直接在内部删除键值对，外部无法介入处理脏数据写回数据库，所以使用Redis作读写缓存时，只能采用同步直写策略，修改缓存的同时也要写入到后端数据库中，从而保证修改操作不被丢失。但这种方案在并发场景下会导致数据库和缓存的不一致，需要在特定业务场景下或者配合分布式锁使用。

  当一个系统引入缓存时，需要面临最大的问题就是，如何保证缓存和后端数据库的一致性问题，最常见的3个解决方案分别是Cache Aside、Read/Write Throught和Write Back缓存更新策略。

  1、Cache Aside策略：就是文章所讲的只读缓存模式。读操作命中缓存直接返回，否则从后端数据库加载到缓存再返回。写操作直接更新数据库，然后删除缓存。这种策略的优点是一切以后端数据库为准，可以保证缓存和数据库的一致性。缺点是写操作会让缓存失效，再次读取时需要从数据库中加载。这种策略是我们在开发软件时最常用的，在使用Memcached或Redis时一般都采用这种方案。

  2、Read/Write Throught策略：应用层读写只需要操作缓存，不需要关心后端数据库。应用层在操作缓存时，缓存层会自动从数据库中加载或写回到数据库中，这种策略的优点是，对于应用层的使用非常友好，只需要操作缓存即可，缺点是需要缓存层支持和后端数据库的联动。

  3、Write Back策略：类似于文章所讲的读写缓存模式+异步写回策略。写操作只写缓存，比较简单。而读操作如果命中缓存则直接返回，否则需要从数据库中加载到缓存中，在加载之前，如果缓存已满，则先把

需要淘汰的缓存数据写回到后端数据库中，再把对应的数据放入到缓存中。这种策略的优点是，写操作飞快（只写缓存），缺点是如果数据还未来得及写入后端数据库，系统发生异常会导致缓存和数据库的不一致。这种策略经常使用在操作系统Page Cache中，或者应对大量写操作的数据库引擎中。

除了以上提到的缓存和数据库的更新策略之外，还有一个问题就是操作缓存或数据库发生异常时如何处理？例如缓存操作成功，数据库操作失败，或者反过来，还是有可能会产生不一致的情况。

比较简单的解决方案是，根据业务设计好更新缓存和数据库的先后顺序来降低影响，或者给缓存设置较短的有效期来降低不一致的时间。如果需要严格保证缓存和数据库的一致性，即保证两者操作的原子性，这就涉及到分布式事务问题了，常见的解决方案就是我们经常听到的两阶段提交（2PC）、三阶段提交（3PC）、TCC、消息队列等方式来保证了，方案也会比较复杂，一般用在对于一致性要求较高的业务场景中。[25赞]

- humor 2020-10-12 22:50:27
能进入候选集合的数据的 lru 字段值必须小于候选集合中最小的 lru 值。
感觉这句话有问题，如果能进入候选集合的数据的lru字段值都小于候选集合中的最小的lru值的话，每次淘汰的肯定是刚进入候选集合的这条数据啊，这样这条被选择进行候选集合的数据就没有必要再进入候选集合了啊，直接删除就可以了吧 [1赞]

作者回复2020-10-13 09:50:09
在实际运行时，每次往候选集中插入的数据可能不止一个，而在淘汰数据时，也是会根据使用内存量超过maxmemory的情况，来决定要淘汰的数据量，所以可能也不止一个数据被淘汰。候选集的作用是先把符合条件（lru值小）的数据准备好。候选集本身是会按照lru值大小排序的，等待要淘汰时，会根据要淘汰的量，从候选集中淘汰数据。所以，并不是刚进入候选集就立马就淘汰了。准备候选集和淘汰数据实际是两个解耦的逻辑操作。

- 小喵喵 2020-10-12 15:56:56
请教下老师：
1、volatile-ttl这个全称是啥？
2、干净数据是缓存里面的数据和数据库里面是一致的，既然一致，为什么要删除呢？
3、脏数据为什么要写回到数据库呢？难道是先更新cache，然后去更新数据库吗？ [1赞]

- 不正经、绅士 2020-10-12 11:14:12
能进入候选集合的数据的 lru 字段值必须小于候选集合中最小的 lru 值。
这里有个疑问，请教老师，这样第二次及后续进入的备选淘汰集合中的数据lru都小于第一次的，淘汰的也是lru最小的，那第一次进入淘汰集合的数据这样不就不会被选中淘汰了呢 [1赞]

作者回复2020-10-13 07:35:19
是有这种可能的，第一次进入候选集合的数据是随机选取的，数据的lru值可能大可能小。第二次及后续再进入候选集的数据的lru值需要小于候选集中的最小lru值。

同时，候选集的实现是一个链表，数据是按照lru值排序的，链表头是lru值最大的，链表尾是lru值最小的。当第二次及后续进入候选集的数据lru更小，但是候选集中已经没有空位置时，候选集链表头的数据会被移出候选集，把位置空出来，给新进入的数据。这样的话，这个被移出的数据就不会作为被淘汰的候选数据了。

- yeek 2020-10-12 08:56:30
记录几个问题：

1. 淘汰对当前请求的延迟问题：

2. 淘汰数据的上限是多少？仅满足当前set所需的内存空间么？
3. 如果随机多次依然不存在比候选队列中最小lru还小的数据，且内存空间还需要继续释放，是否有执行时间上限？ [1赞]

- 叶子。 2020-10-13 11:09:26
老师能帮我看看这个问题吗
数据淘汰是为了防止操作系统swap吗？我理解就像是JVM的-XMX设置了maxmemory之后是不是就不会发生swap了？因为swap之前说是发生在分配内存小于实际使用内存，而设置了maxmemory之后，当出现内存不足就发生了淘汰了，也就不会发生swap了？

- xueyuan 2020-10-13 00:29:44
读写模式的异步回写需要所有写请求都先在缓存中处理。等到这些增改的数据要被从缓存中淘汰出来时，缓存将它们写回后端数据库;这个redis做不到。

redis可以支持读写模式的同步回写和只读模式