

18-波动的响应延迟：如何应对变慢的Redis？（上）

你好，我是蒋德钧。

在Redis的实际部署应用中，有一个非常严重的问题，那就是Redis突然变慢了。一旦出现这个问题，不仅会直接影响用户的使用体验，还可能会影响到“旁人”，也就是和Redis在同一个业务系统中的其他系统，比如说数据库。

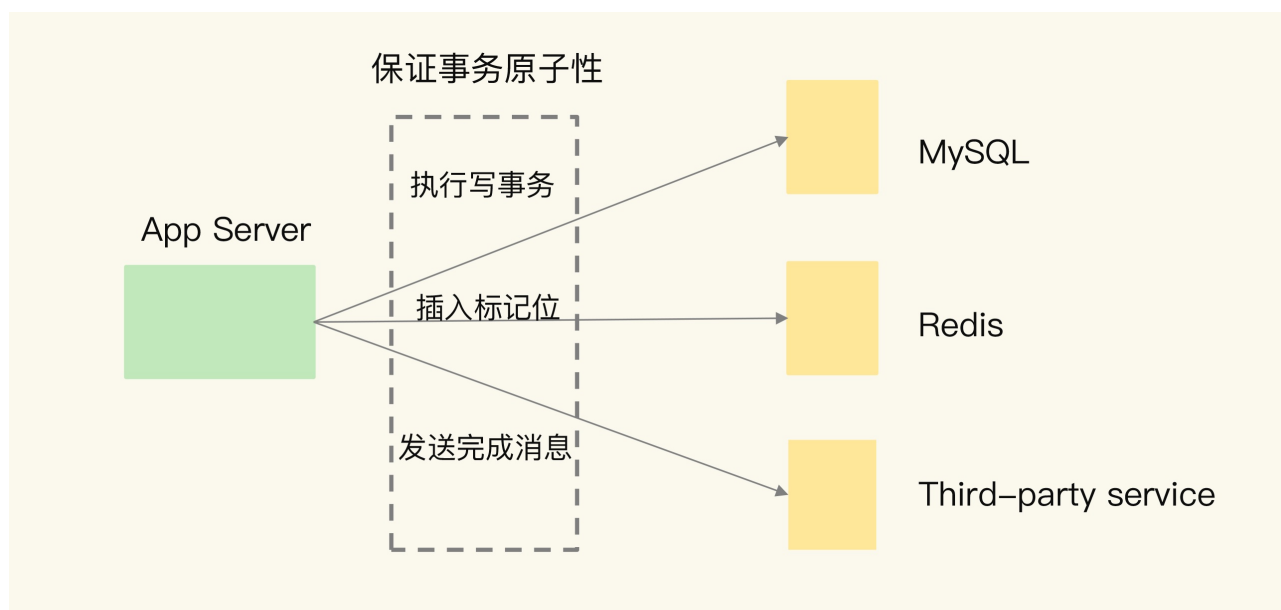
举个小例子，在秒杀场景下，一旦Redis变慢了，大量的用户下单请求就会被拖慢，也就是说，用户提交了下单申请，却没有收到任何响应，这会给用户带来非常糟糕的使用体验，甚至可能会导致用户流失。

而且，在实际生产环境中，Redis往往是业务系统中的一个环节（例如作为缓存或是作为数据库）。一旦Redis上的请求延迟增加，就可能引起业务系统中的一串儿“连锁反应”。

我借助一个包含了Redis的业务逻辑的小例子，简单地给你解释一下。

应用服务器（App Server）要完成一个事务性操作，包括在MySQL上执行一个写事务，在Redis上插入一个标记位，并通过一个第三方服务给用户发送一条完成消息。

这三个操作都需要保证事务原子性，所以，如果此时Redis的延迟增加，就会拖累App Server端整个事务的执行。这个事务一直完成不了，又会导致MySQL上写事务占用的资源无法释放，进而导致访问MySQL的其他请求被阻塞。很明显，Redis变慢会带来严重的连锁反应。



我相信，不少人遇到过这个问题，那具体该怎么解决呢？

这个时候，切忌“病急乱投医”。如果没有一套行之有效的应对方案，大多数时候我们只能各种尝试，做无用功。在前面的[第16讲](#)、[第17讲](#)中，我们学习了会导致Redis变慢的潜在阻塞点以及相应的解决方案，即异步线程机制和CPU绑核。除此之外，还有一些因素会导致Redis变慢。

接下来的两节课，我再向你介绍一下如何系统性地应对Redis变慢这个问题。我会从问题认定、系统性排查和应对方案这3个方面给你具体讲解。学完这两节课以后，你一定能够有章法地解决Redis变慢的问题。

Redis真的变慢了吗？

在实际解决问题之前，我们首先要弄清楚，如何判断Redis是不是真的变慢了。

一个最直接的方法，就是**查看Redis的响应延迟**。

大部分时候，Redis延迟很低，但是在某些时刻，有些Redis实例会出现很高的响应延迟，甚至能达到几秒到十几秒，不过持续时间不长，这也叫延迟“毛刺”。当你发现Redis命令的执行时间突然就增长到了几秒，基本就可以认定Redis变慢了。

这种方法是看Redis延迟的绝对值，但是，在不同的软硬件环境下，Redis本身的绝对性能并不相同。比如，在我的环境中，当延迟为1ms时，我判定Redis变慢了，但是你的硬件配置高，那么，在你的运行环境下，可能延迟是0.2ms的时候，你就可以认定Redis变慢了。

所以，这里我就要说第二个方法了，也就是基于**当前环境下的Redis基线性能**做判断。所谓的基线性能呢，也就是一个系统在低压力、无干扰下的基本性能，这个性能只由当前的软硬件配置决定。

你可能会问，具体怎么确定基线性能呢？有什么好方法吗？

实际上，从2.8.7版本开始，redis-cli命令提供了--intrinsic-latency选项，可以用来监测和统计测试期间的最大延迟，这个延迟可以作为Redis的基线性能。其中，测试时长可以用--intrinsic-latency选项的参数来指定。

举个例子，比如说，我们运行下面的命令，该命令会打印120秒内监测到的最大延迟。可以看到，这里的最大延迟是119微秒，也就是基线性能为119微秒。一般情况下，运行120秒就足够监测到最大延迟了，所以，我们可以把参数设置为120。

```
./redis-cli --intrinsic-latency 120
Max latency so far: 17 microseconds.
Max latency so far: 44 microseconds.
Max latency so far: 94 microseconds.
Max latency so far: 110 microseconds.
Max latency so far: 119 microseconds.

36481658 total runs (avg latency: 3.2893 microseconds / 3289.32 nanoseconds per run).
Worst run took 36x longer than the average latency.
```

需要注意的是，基线性能和当前的操作系统、硬件配置相关。因此，我们可以把它和Redis运行时的延迟结合起来，再进一步判断Redis性能是否变慢了。

一般来说，你要把运行时延迟和基线性能进行对比，如果你观察到的Redis运行时延迟是其基线性能的2倍及以上，就可以认定Redis变慢了。

判断基线性能这一点，对于在虚拟化环境下运行的Redis来说，非常重要。这是因为，在虚拟化环境（例如虚拟机或容器）中，由于增加了虚拟化软件层，与物理机相比，虚拟机或容器本身就会引入一定的性能开销，所以基线性能会高一些。下面的测试结果，显示的就是某一个虚拟机上运行Redis时测的基线性能。

```
$ ./redis-cli --intrinsic-latency 120
Max latency so far: 692 microseconds.
Max latency so far: 915 microseconds.
Max latency so far: 2193 microseconds.
Max latency so far: 9343 microseconds.
Max latency so far: 9871 microseconds.
```

可以看到，由于虚拟化软件本身的开销，此时的基线性能已经达到了9.871ms。如果该Redis实例的运行延迟为10ms，这并不能算作性能变慢，因为此时，运行时延迟只比基线性能增加了1.3%。如果你不了解基线性能，一看到较高的运行时延迟，就很有可能误判Redis变慢了。

不过，我们通常是通过客户端和网络访问Redis服务，为了避免网络对基线性能的影响，刚刚说的这个命令需要在服务器端直接运行，这也就是说，**我们只考虑服务器端软硬件环境的影响**。

如果你想了解网络对Redis性能的影响，一个简单的方法是用iPerf这样的工具，测量从Redis客户端到服务器端的网络延迟。如果这个延迟有几十毫秒甚至是几百毫秒，就说明，Redis运行的网络环境中很可能有大流量的其他应用程序在运行，导致网络拥塞了。这个时候，你就需要协调网络运维，调整网络的流量分配了。

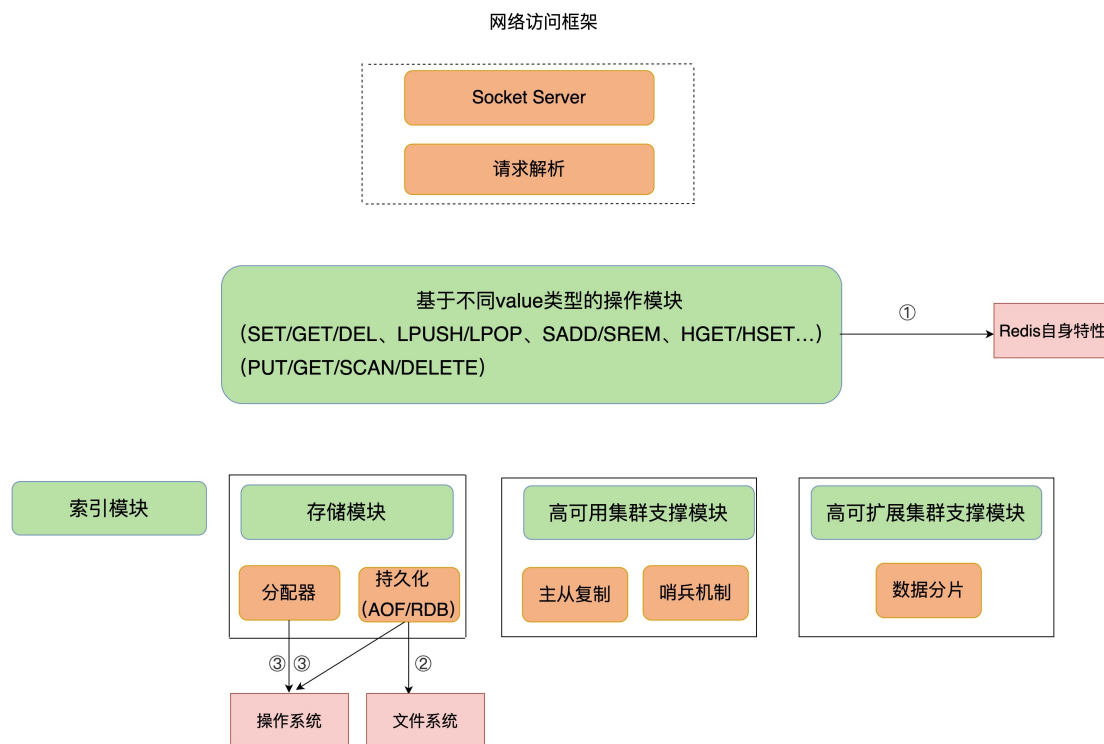
如何应对Redis变慢?

经过了上一步之后，你已经能够确定Redis是否变慢了。一旦发现变慢了，接下来，就要开始查找原因并解决这个问题了，这其实是一个很有意思的诊断过程。

此时的你就像一名医生，而Redis则是一位病人。在给病人看病时，你要知道人体的机制，还要知道可能对身体造成影响的外部因素，比如不健康的食物、不好的情绪等，然后要拍CT、心电图等找出病因，最后再确定治疗方案。

在诊断“Redis变慢”这个病症时，同样也是这样。你要基于自己对Redis本身的工作原理的理解，并且结合和它交互的操作系统、存储以及网络等外部系统关键机制，再借助一些辅助工具来定位原因，并制定行之有效的解决方案。

医生诊断一般都是有章可循的。同样，Redis的性能诊断也有章可依，这就是影响Redis的关键因素。下面这张图你应该有印象，这是我们在[第一节](#)课画的Redis架构图。你可以重点关注下我在图上新增的红色模块，也就是Redis自身的操作特性、文件系统和操作系统，它们是影响Redis性能的三大要素。



接下来，我将从这三大要素入手，结合实际的应用场景，依次给你介绍从不同要素出发排查和解决问题的实践经验。这节课我先给你介绍Redis的自身操作特性的影响，下节课我们再重点研究操作系统和文件系统的影响。

Redis自身操作特性的影响

首先，我们来学习下Redis提供的键值对命令操作对延迟性能的影响。我重点介绍两类关键操作：慢查询命令和过期key操作。

1.慢查询命令

慢查询命令，就是指在Redis中执行速度慢的命令，这会导致Redis延迟增加。Redis提供的命令操作很多，并不是所有命令都慢，这和命令操作的复杂度有关。所以，我们必须要知道Redis的不同命令的复杂度。

比如说，Value类型为String时，GET/SET操作主要就是操作Redis的哈希表索引。这个操作复杂度基本是固定的，即 $O(1)$ 。但是，当Value类型为Set时，SORT、SUNION/SMEMBERS操作复杂度分别为 $O(N+M*\log(M))$ 和 $O(N)$ 。其中，N为Set中的元素个数，M为SORT操作返回的元素个数。这个复杂度就增加了很多。[Redis官方文档](#)中对每个命令的复杂度都有介绍，当你需要了解某个命令的复杂度时，可以直接查询。

那该怎么应对这个问题呢？在这儿，我就要给你排查建议和解决方法了，这也是今天的第一个方法。

当你发现Redis性能变慢时，可以通过Redis日志，或者是latency monitor工具，查询变慢的请求，根据请求对应的具体命令以及官方文档，确认下是否采用了复杂度高的慢查询命令。

如果的确有大量的慢查询命令，有两种处理方式：

1. **用其他高效命令代替。**比如说，如果你需要返回一个SET中的所有成员时，不要使用SMEMBERS命令，而是要使用SSCAN多次迭代返回，避免一次返回大量数据，造成线程阻塞。
2. **当你需要执行排序、交集、并集操作时，可以在客户端完成，而不要用SORT、SUNION、SINTER这些命令，以免拖慢Redis实例。**

当然，如果业务逻辑就是要求使用慢查询命令，那你得考虑采用性能更好的CPU，更快地完成查询命令，避免慢查询的影响。

还有一个比较容易忽略的慢查询命令，就是KEYS。它用于返回和输入模式匹配的所有key，例如，以下命令返回所有包含“name”字符串的keys。

```
redis> KEYS *name*
1) "lastname"
2) "firstname"
```

因为KEYS命令需要遍历存储的键值对，所以操作延时高。如果你不了解它的实现而使用了它，就会导致Redis性能变慢。所以，**KEYS命令一般不被建议用于生产环境中。**

2.过期key操作

接下来，我们来看过期key的自动删除机制。它是Redis用来回收内存空间的常用机制，应用广泛，本身就会引起Redis操作阻塞，导致性能变慢，所以，你必须要知道该机制对性能的影响。

Redis键值对的key可以设置过期时间。默认情况下，Redis每100毫秒会删除一些过期key，具体的算法如下：

1. 采样ACTIVE_EXPIRE_CYCLE_LOOKUPS_PER_LOOP个数的key，并将其中过期的key全部删除；
2. 如果超过25%的key过期了，则重复删除的过程，直到过期key的比例降至25%以下。

ACTIVE_EXPIRE_CYCLE_LOOKUPS_PER_LOOP是Redis的一个参数，默认是20，那么，一秒内基本有200个过期key会被删除。这一策略对清除过期key、释放内存空间很有帮助。如果每秒钟删除200个过期key，并不会对Redis造成太大影响。

但是，如果触发了上面这个算法的第二条，Redis就会一直删除以释放内存空间。注意，**删除操作是阻塞的**（Redis 4.0后可以用异步线程机制来减少阻塞影响）。所以，一旦该条件触发，Redis的线程就会一直执行删除，这样一来，就没办法正常服务其他的键值操作了，就会进一步引起其他键值操作的延迟增加，Redis就会变慢。

那么，算法的第二条是怎么被触发的呢？其中一个重要来源，就是**频繁使用带有相同时间参数的EXPIREAT命令设置过期key**，这就会导致，在同一秒内有大量的key同时过期。

现在，我就要给出第二条排查建议和解决方法了。

你要检查业务代码在使用EXPIREAT命令设置key过期时间时，是否使用了相同的UNIX时间戳，有没有使用

EXPIRE命令给批量的key设置相同的过期秒数。因为，这都会造成大量key在同一时间过期，导致性能变慢。

遇到这种情况时，千万不要嫌麻烦，你首先要根据实际业务的使用需求，决定EXPIREAT和EXPIRE的过期时间参数。其次，如果一批key的确是同时过期，你还可以在EXPIREAT和EXPIRE的过期时间参数上，加上一个一定大小范围内的随机数，这样，既保证了key在一个邻近时间范围内被删除，又避免了同时过期造成的压力。

小结

这节课，我首先给你介绍了Redis性能变慢带来的重要影响，希望你能充分重视这个问题。我重点介绍了判断Redis变慢的方法，一个是看响应延迟，一个是看基线性能。同时，我还给了你两种排查和解决Redis变慢这个问题的方法：

1. 从慢查询命令开始排查，并且根据业务需求替换慢查询命令；
2. 排查过期key的时间设置，并根据实际使用需求，设置不同的过期时间。

性能诊断通常是一件困难的事，所以我们一定不能毫无目标地“乱找”。这节课给你介绍的内容，就是排查和解决Redis性能变慢的章法，你一定要按照章法逐一排查，这样才可能尽快地找出原因。

当然，要真正把Redis用好，除了要了解Redis本身的原理，还要了解和Redis交互的各底层系统的关键机制，包括操作系统和文件系统。通常情况下，一些难以排查的问题是Redis的用法或设置和底层系统的工作机制不协调导致的。下节课，我会着重给你介绍文件系统、操作系统对Redis性能的影响，以及相应的排查方法和解决方案。

每课一问

这节课，我提到了KEYS命令，因为它的复杂度很高，容易引起Redis线程操作阻塞，不适用于生产环境。但是，KEYS命令本身提供的功能是上层业务应用经常需要的，即返回与输入模式匹配的keys。

请思考一下，在Redis中，还有哪些其他命令可以代替KEYS命令，实现同样的功能呢？这些命令的复杂度会导致Redis变慢吗？

欢迎在留言区写下你的思考和答案，我们一起讨论，共同学习进步。如果你觉得有所收获，欢迎你把今天的内容分享给你的朋友。

精选留言：

- Kaito 2020-09-18 00:15:10
在Redis中，还有哪些其他命令可以代替KEYS命令，实现同样的功能呢？这些命令的复杂度会导致Redis变慢吗？

如果想要获取整个实例的所有key，建议使用SCAN命令代替。客户端通过执行SCAN \$cursor COUNT \$count可以得到一批key以及下一个游标\$cursor，然后把这个\$cursor当作SCAN的参数，再次执行，以此往复，直到返回的\$cursor为0时，就把整个实例中的所有key遍历出来了。

关于SCAN讨论最多的问题就是，Redis在做Rehash时，会不会漏key或返回重复的key。

在使用SCAN命令时，不会漏key，但可能会得到重复的key，这主要和Redis的Rehash机制有关。Redis

的所有key存在一个全局的哈希表中，如果存入的key慢慢变多，在达到一定阈值后，为了避免哈希冲突导致查询效率降低，这个哈希表会进行扩容。与之对应的，key数量逐渐变少时，这个哈希表会缩容以节省空间。

1、为什么不会漏key？Redis在SCAN遍历全局哈希表时，采用*高位进位法*的方式遍历哈希桶（可网上查询图例，一看就明白），当哈希表扩容后，通过这种算法遍历，旧哈希表中的数据映射到新哈希表，依旧会保留原来的先后顺序，这样就可以保证遍历时不会遗漏也不会重复。

2、为什么SCAN会得到重复的key？这个情况主要发生在哈希表缩容。已经遍历过的哈希桶在缩容时，会映射到新哈希表没有遍历到的位置，所以继续遍历就会对同一个key返回多次。

SCAN是遍历整个实例的所有key，另外Redis针对Hash/Set/Sorted Set也提供了HSCAN/SSCAN/ZSCAN命令，用于遍历一个key中的所有元素，建议在获取一个bigkey的所有数据时使用，避免发生阻塞风险。

但是使用HSCAN/SSCAN/ZSCAN命令，返回的元素数量与执行SCAN逻辑可能不同。执行SCAN \$cursor COUNT \$count时一次最多返回count个数的key，数量不会超过count。

但Hash/Set/Sorted Set元素数量比较少时，底层会采用intset/ziplist方式存储，如果以这种方式存储，在执行HSCAN/SSCAN/ZSCAN命令时，会无视count参数，直接把所有元素一次性返回，也就是说，得到的元素数量是会大于count参数的。当底层转为哈希表或跳表存储时，才会真正使用发count参数，最多返回count个元素。[10赞]

- 青青子衿 2020-09-18 09:11:59

当你发现 Redis 性能变慢时，可以通过 Redis 日志，或者是 latency monitor 工具，查询变慢的请求，根据请求对应的具体命令以及官方文档，确认下是否采用了复杂度高的慢查询命令。
其实这个排除过程才是我们最想学习的，却被作者一带而过了。。。。

- test 2020-09-18 09:06:43

通常线上是不能使用keys的，标准替代方案就是scan。scan不会导致redis变慢，只是如果在scan过程中kv表扩容的话可能会遇到重复key。

PS：sort的时间复杂度是 $O(N+M*\log(M))$ 是因为需要创建一个新的数字，并且用快排去排序。