22-第11～21讲课后思考题答案及常见问题答疑

你好，我是蒋德钧。

咱们的课程已经更新到第21讲了，今天，我们来进行一场答疑。

前半部分，我会给你讲解第11～21讲的课后思考题。在学习这部分内容时，可以和你的答案进行对照，看看还有哪里没有考虑到。当然，有些问题不一定有标准答案，我们还可以继续讨论。

后半部分，我会围绕着许多同学都很关注的如何排查慢查询命令和bigkey的问题，重点解释一下，希望可以解答你的困惑。

好了，我们现在开始。

课后思考题答案

第11讲

问题：除了String类型和Hash类型，还有什么类型适合保存第11讲中所说的图片吗？

答案：除了String和Hash，我们还可以使用Sorted Set类型进行保存。Sorted Set的元素有member值和score值，可以像Hash那样，使用二级编码进行保存。具体做法是，把图片ID的前7位作为Sorted Set的key，把图片ID的后3位作为member值，图片存储对象ID作为score值。

Sorted Set中元素较少时，Redis会使用压缩列表进行存储，可以节省内存空间。不过，和Hash不一样，Sorted Set插入数据时，需要按score值的大小排序。当底层结构是压缩列表时，Sorted Set的插入性能就比不上Hash。所以，在我们这节课描述的场景中，Sorted Set类型虽然可以用来保存，但并不是最优选项。

第12讲

问题：我在第12讲中介绍了4种典型的统计模式，分别是聚合统计、排序统计、二值状态统计和基数统计，以及它们各自适合的集合类型。你还遇到过其他的统计场景吗？用的是什么集合类型呢？

答案：@海拉鲁同学在留言中提供了一种场景：他们曾使用List+Lua统计最近200个客户的触达率。具体做法是，每个List元素表示一个客户，元素值为0，代表触达；元素值为1，就代表未触达。在进行统计时，应用程序会把代表客户的元素写入队列中。当需要统计触达率时，就使用LRANGE key 0 -1 取出全部元素，计算0的比例，这个比例就是触达率。

这个例子需要获取全部元素，不过数据量只有200个，不算大，所以，使用List，在实际应用中也是可以接受的。但是，如果数据量很大，又有其他查询需求的话（例如查询单个元素的触达情况），List的操作复杂度较高，就不合适了，可以考虑使用Hash类型。

第13讲

问题：你在日常的实践过程中，还用过Redis的其他数据类型吗？

答案：除了我们课程上介绍的5大基本数据类型，以及HyperLogLog、Bitmap、GEO，Redis还有一种数据类型，叫作布隆过滤器。它的查询效率很高，经常会用在缓存场景中，可以用来判断数据是否存在缓存中。

我会在后面（第25讲）具体地介绍一下它。

第14讲

问题：在用Sorted Set保存时间序列数据时，如果把时间戳作为score，把实际的数据作为member，这样保存数据有没有潜在的风险？另外，如果你是Redis的开发维护者，你会把聚合计算也设计为Sorted Set的一个内在功能吗？

答案：Sorted Set和Set一样，都会对集合中的元素进行去重，也就是说，如果我们往集合中插入的member值，和之前已经存在的member值一样，那么，原来member的score就会被新写入的member的score覆盖。相同member的值，在Sorted Set中只会保留一个。

对于时间序列数据来说，这种去重的特性是会带来数据丢失风险的。毕竟，某一时间段内的多个时间序列数据的值可能是相同的。如果我们往Sorted Set中写入的数据是在不同时刻产生的，但是写入的时刻不同，Sorted Set中只会保存一份最近时刻的数据。这样一来，其他时刻的数据就都没有保存下来。

举个例子，在记录物联网设备的温度时，一个设备一个上午的温度值可能都是26。在Sorted Set中，我们把温度值作为member，把时间戳作为score。我们用ZADD命令把上午不同时刻的温度值写入Sorted Set。由于member值一样，所以只会把score更新为最新时间戳，最后只有一个最新时间戳（例如上午12点）下的温度值。这肯定是无法满足保存多个时刻数据的需求的。

关于是否把聚合计算作为Sorted Set的内在功能，考虑到Redis的读写功能是由单线程执行，在进行数据读写时，本身就会消耗较多的CPU资源，如果再在Sorted Set中实现聚合计算，就会进一步增加CPU的资源消耗，影响到Redis的正常数据读取。所以，如果我是Redis的开发维护者，除非对Redis的线程模型做修改，比如说在Redis中使用额外的线程池做聚合计算，否则，我不会把聚合计算作为Redis的内在功能实现的。

第15讲

问题：如果一个生产者发送给消息队列的消息，需要被多个消费者进行读取和处理（例如，一个消息是一条从业务系统采集的数据，既要被消费者1读取并进行实时计算，也要被消费者2读取并留存到分布式文件系统HDFS中，以便后续进行历史查询），你会使用Redis的什么数据类型来解决这个问题呢？

答案：有同学提到，可以使用Streams数据类型的消费组，同时消费生产者的数据，这是可以的。但是，有个地方需要注意，如果只是使用一个消费组的话，消费组内的多个消费者在消费消息时是互斥的，换句话说，在一个消费组内，一个消息只能被一个消费者消费。我们希望消息既要被消费者1读取，也要被消费者2读取，是一个多消费者的需求。所以，如果使用消费组模式，需要让消费者1和消费者2属于不同的消费组，这样它们就能同时消费了。

另外，Redis基于字典和链表数据结构，实现了发布和订阅功能，这个功能可以实现一个消息被多个消费者消费使用，可以满足问题中的场景需求。

第16讲

问题：Redis的写操作（例如SET、HSET、SADD等）是在关键路径上吗？

答案：Redis本身是内存数据库，所以，写操作都需要在内存上完成执行后才能返回，这就意味着，如果这些写操作处理的是大数据集，例如1万个数据，那么，主线程需要等这1万个数据都写完，才能继续执行后面的命令。所以说，Redis的写操作也是在关键路径上的。

这个问题是希望你把面向内存和面向磁盘的写操作区分开。当一个写操作需要把数据写到磁盘时，一般来说，写操作只要把数据写到操作系统的内核缓冲区就行。不过，如果我们执行了同步写操作，那就必须要等到数据写回磁盘。所以，面向磁盘的写操作一般不会在关键路径上。

我看到有同学说，根据写操作命令的返回值来决定是否在关键路径上，如果返回值是OK，或者客户端不关心是否写成功，那么，此时的写操作就不算在关键路径上。

这个思路不错，不过，需要注意的是，客户端经常会阻塞等待发送的命令返回结果，在上一个命令还没有返回结果前，客户端会一直等待，直到返回结果后，才会发送下一个命令。此时，即使我们不关心返回结果，客户端也要等到写操作执行完成才行。所以，在不关心写操作返回结果的场景下，可以对Redis客户端做异步改造。具体点说，就是使用异步线程发送这些不关心返回结果的命令，而不是在Redis客户端中等待这些命令的结果。

第17讲

问题：在一台有两个CPU Socket（每个Socket 8个物理核）的服务器上，我们部署了一个有着8个实例的Redis切片集群（8个实例都为主节点，没有主备关系），现在有两个方案：

1. 在同一个CPU Socket上运行8个实例，并和8个CPU核绑定；
2. 在两个CPU Socket上各运行4个实例，并和相应Socket上的核绑定。

如果不考虑网络数据读取的影响，你会选择哪个方案呢？

答案：建议使用第二个方案，主要有两方面的原因。

1. 同一个CPU Socket上的进程，会共享L3缓存。如果把8个实例都部署在同一个Socket上，它们会竞争L3缓存，这就会导致它们的L3缓存命中率降低，影响访问性能。
2. 同一个CPU Socket上的进程，会使用同一个Socket上的内存空间。8个实例共享同一个Socket上的内存空间，肯定会竞争内存资源。如果有实例保存的数据量大，其他实例能用到的内存空间可能就不够了，此时，其他实例就会跨Socket申请内存，进而造成跨Socket访问内存，造成实例的性能降低。

另外，在切片集群中，不同实例间通过网络进行消息通信和数据迁移，并不会使用共享内存空间进行跨实例的数据访问。所以，即使把不同的实例部署到不同的Socket上，它们之间也不会发生跨Socket内存的访问，不会受跨Socket内存访问的负面影响。

第18讲

问题：在Redis中，还有哪些命令可以代替KEYS命令，实现对键值对的key的模糊查询呢？这些命令的复杂度会导致Redis变慢吗？

答案：Redis提供的SCAN命令，以及针对集合类型数据提供的SSCAN、HSCAN等，可以根据执行时设定的数量参数，返回指定数量的数据，这就可以避免像KEYS命令一样同时返回所有匹配的数据，不会导致Redis变慢。以HSCAN为例，我们可以执行下面的命令，从user这个Hash集合中返回key前缀以103开头的100个键值对。

```
HSCAN user 0  match "103*" 100
```

第19讲

问题：你遇到过Redis变慢的情况吗？如果有的话，你是怎么解决的呢？

答案：@Kaito同学在留言区分享了他排查Redis变慢问题的Checklist，而且还提供了解决方案，非常好，我把Kaito同学给出的导致Redis变慢的原因汇总并完善一下，分享给你：

1. 使用复杂度过高的命令或一次查询全量数据；
2. 操作bigkey；
3. 大量key集中过期；
4. 内存达到maxmemory；
5. 客户端使用短连接和Redis相连；
6. 当Redis实例的数据量大时，无论是生成RDB，还是AOF重写，都会导致fork耗时严重；
7. AOF的写回策略为always，导致每个操作都要同步刷回磁盘；
8. Redis实例运行机器的内存不足，导致swap发生，Redis需要到swap分区读取数据；
9. 进程绑定CPU不合理；
10. Redis实例运行机器上开启了透明内存大页机制；
11. 网卡压力过大。

第20讲

问题：我们可以使用mem_fragmentation_ratio来判断Redis当前的内存碎片率是否严重，我给出的经验阈值都是大于1的。我想请你思考一下，如果mem_fragmentation_ratio小于1，Redis的内存使用是什么情况呢？会对Redis的性能和内存空间利用率造成什么影响呢？

答案：如果mem_fragmentation_ratio小于1，就表明，操作系统分配给Redis的内存空间已经小于Redis所申请的空间大小了，此时，运行Redis实例的服务器上的内存已经不够用了，可能已经发生swap了。这样一来，Redis的读写性能也会受到影响，因为Redis实例需要在磁盘上的swap分区中读写数据，速度较慢。

第21讲

问题：在和Redis实例交互时，应用程序中使用的客户端需要使用缓冲区吗？如果使用的话，对Redis的性能和内存使用会有影响吗？

答案：应用程序中使用的Redis客户端，需要把要发送的请求暂存在缓冲区。这有两方面的好处。

一方面，可以在客户端控制发送速率，避免把过多的请求一下子全部发到Redis实例，导致实例因压力过大而性能下降。不过，客户端缓冲区不会太大，所以，对Redis实例的内存使用没有什么影响。

另一方面，在应用Redis主从集群时，主从节点进行故障切换是需要一定时间的，此时，主节点无法服务外来请求。如果客户端有缓冲区暂存请求，那么，客户端仍然可以正常接收业务应用的请求，这就可以避免直接给应用返回无法服务的错误。

代表性问题

在前面的课程中，我重点介绍了避免Redis变慢的方法。慢查询命令的执行时间和bigkey操作的耗时都很

长，会阻塞Redis。很多同学学完之后，知道了要尽量避免Redis阻塞，但是还不太清楚，具体应该如何排查阻塞的命令和bigkey呢。

所以，接下来，我就再重点解释一下，如何排查慢查询命令，以及如何排查bigkey。

问题1：如何使用慢查询日志和latency monitor排查执行慢的操作？

在第18讲中，我提到，可以使用Redis日志（慢查询日志）和latency monitor来排查执行较慢的命令操作，那么，我们该如何使用慢查询日志和latency monitor呢？

Redis的慢查询日志记录了执行时间超过一定阈值的命令操作。当我们发现Redis响应变慢、请求延迟增加时，就可以在慢查询日志中进行查找，确定究竟是哪些命令执行时间很长。

在使用慢查询日志前，我们需要设置两个参数。

- **slowlog-log-slower-than**：这个参数表示，慢查询日志对执行时间大于多少微秒的命令进行记录。
- **slowlog-max-len**：这个参数表示，慢查询日志最多能记录多少条命令记录。慢查询日志的底层实现是一个具有预定大小的先进先出队列，一旦记录的命令数量超过了队列长度，最先记录的命令操作就会被删除。这个值默认是128。但是，如果慢查询命令较多的话，日志里就存不下了；如果这个值太大了，又会占用一定的内存空间。所以，一般建议设置为1000左右，这样既可以多记录些慢查询命令，方便排查，也可以避免内存开销。

设置好参数后，慢查询日志就会把执行时间超过slowlog-log-slower-than阈值的命令操作记录在日志中。

我们可以使用SLOWLOG GET命令，来查看慢查询日志中记录的命令操作，例如，我们执行如下命令，可以查看最近的一条慢查询的日志信息。

```
SLOWLOG GET 1
1) 1) (integer) 33           //每条日志的唯一ID编号
   2) (integer) 1600990583   //命令执行时的时间戳
   3) (integer) 20906        //命令执行的时长，单位是微秒
   4) 1) "keys"               //具体的执行命令和参数
      2) "abc*"
   5) "127.0.0.1:54793"      //客户端的IP和端口号
   6) ""                     //客户端的名称，此处为空
```

可以看到，KEYS "abc*"这条命令的执行时间是20906微秒，大约20毫秒，的确是一条执行较慢的命令操作。如果我们想查看更多的慢日志，只要把SLOWLOG GET后面的数字参数改为想查看的日志条数，就可以了。

好了，有了慢查询日志后，我们就可以快速确认，究竟是哪些命令的执行时间比较长，然后可以反馈给业务部门，让业务开发人员避免在应用Redis的过程中使用这些命令，或是减少操作的数据量，从而降低命令的执行复杂度。

除了慢查询日志以外，Redis从2.8.13版本开始，还提供了latency monitor监控工具，这个工具可以用来监

控Redis运行过程中的峰值延迟情况。

和慢查询日志的设置相类似，要使用latency monitor，首先要设置命令执行时长的阈值。当一个命令的实际执行时长超过该阈值时，就会被latency monitor监控到。比如，我们可以把latency monitor监控的命令执行时长阈值设为1000微秒，如下所示：

```
config set latency-monitor-threshold 1000
```

设置好了latency monitor的参数后，我们可以使用latency latest命令，查看最新和最大的超过阈值的延迟情况，如下所示：

```
latency latest
1) 1) "command"
   2) (integer) 1600991500    //命令执行的时间戳
   3) (integer) 2500           //最近的超过阈值的延迟
   4) (integer) 10100          //最大的超过阈值的延迟
```

问题2：如何排查Redis的bigkey?

在应用Redis时，我们要尽量避免bigkey的使用，这是因为，Redis主线程在操作bigkey时，会被阻塞。那么，一旦业务应用中使用了bigkey，我们该如何进行排查呢?

Redis可以在执行redis-cli命令时带上–bigkeys选项，进而对整个数据库中的键值对大小情况进行统计分析，比如说，统计每种数据类型的键值对个数以及平均大小。此外，这个命令执行后，会输出每种数据类型中最大的bigkey的信息，对于String类型来说，会输出最大bigkey的字节长度，对于集合类型来说，会输出最大bigkey的元素个数，如下所示：

```
./redis-cli  --bigkeys

-------- summary -------
Sampled 32 keys in the keyspace!
Total key length in bytes is 184 (avg len 5.75)

//统计每种数据类型中元素个数最多的bigkey
Biggest   list found 'product1' has 8 items
Biggest   hash found 'dtemp' has 5 fields
Biggest string found 'page2' has 28 bytes
Biggest stream found 'mqstream' has 4 entries
Biggest    set found 'userid' has 5 members
Biggest   zset found 'device:temperature' has 6 members

//统计每种数据类型的总键值个数，占所有键值个数的比例，以及平均大小
4 lists with 15 items (12.50% of keys, avg size 3.75)
5 hashs with 14 fields (15.62% of keys, avg size 2.80)
10 strings with 68 bytes (31.25% of keys, avg size 6.80)
1 streams with 4 entries (03.12% of keys, avg size 4.00)
7 sets with 19 members (21.88% of keys, avg size 2.71)
```

```
5 zsets with 17 members (15.62% of keys, avg size 3.40)
```

不过，在使用–bigkeys选项时，有一个地方需要注意一下。这个工具是通过扫描数据库来查找bigkey的，所以，在执行的过程中，会对Redis实例的性能产生影响。如果你在使用主从集群，我建议你在从节点上执行该命令。因为主节点上执行时，会阻塞主节点。如果没有从节点，那么，我给你两个小建议：第一个建议是，在Redis实例业务压力的低峰阶段进行扫描查询，以免影响到实例的正常运行；第二个建议是，可以使用-i参数控制扫描间隔，避免长时间扫描降低Redis实例的性能。例如，我们执行如下命令时，redis-cli会每扫描100次暂停100毫秒（0.1秒）。

```
./redis-cli  --bigkeys -i 0.1
```

当然，使用Redis自带的–bigkeys选项排查bigkey，有两个不足的地方：

1. 这个方法只能返回每种类型中最大的那个bigkey，无法得到大小排在前N位的bigkey；
2. 对于集合类型来说，这个方法只统计集合元素个数的多少，而不是实际占用的内存量。但是，一个集合中的元素个数多，并不一定占用的内存就多。因为，有可能每个元素占用的内存很小，这样的话，即使元素个数有很多，总内存开销也不大。

所以，如果我们想统计每个数据类型中占用内存最多的前N个bigkey，可以自己开发一个程序，来进行统计。

我给你提供一个基本的开发思路：使用SCAN命令对数据库扫描，然后用TYPE命令获取返回的每一个key的类型。接下来，对于String类型，可以直接使用STRLEN命令获取字符串的长度，也就是占用的内存空间字节数。

对于集合类型来说，有两种方法可以获得它占用的内存大小。

如果你能够预先从业务层知道集合元素的平均大小，那么，可以使用下面的命令获取集合元素的个数，然后乘以集合元素的平均大小，这样就能获得集合占用的内存大小了。

- List类型：LLEN命令；
- Hash类型：HLEN命令；
- Set类型：SCARD命令；
- Sorted Set类型：ZCARD命令；

如果你不能提前知道写入集合的元素大小，可以使用MEMORY USAGE命令（需要Redis 4.0及以上版本），查询一个键值对占用的内存空间。例如，执行以下命令，可以获得key为user:info这个集合类型占用的内存空间大小。

```
MEMORY USAGE user:info
(integer) 315663239
```

这样一来，你就可以在开发的程序中，把每一种数据类型中的占用内存空间大小排在前 N 位的key统计出来，这也就是每个数据类型中的前N个bigkey。

总结

从第11讲到第21讲，我们重点介绍的知识点比较多，也比较细。其实，我们可以分成两大部分来掌握：一个是多种多样的数据结构，另一个是如何避免Redis性能变慢。

希望这节课的答疑，能帮助你更加深入地理解前面学过的内容。通过这节课，你应该也看到了，课后思考题是一种很好地梳理重点内容、拓展思路的方式，所以，在接下来的课程里，希望你能多留言聊一聊你的想法，这样可以进一步巩固你所学的知识。而且，还能在和其他同学的交流中，收获更多东西。好了，这节课就到这里，我们下节课见。

精选留言：

- 九时四 2020-09-30 09:28:22
  豁然开朗 [1赞]

- jinjunzhu 2020-10-01 10:06:48
  柳暗花明又一村