二

22 第11～21讲课后思考题答案及常见问题答疑

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。

上一页 下一页