
回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芋艿这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2019-11-22

精尽 Redis 面试题「最新更新时间：2023-07」

> 这个面试题是建立在胖友看过 《精尽【缓存 】面试题》 。

以下面试题，基于网络整理，和自己编辑。具体参考的文章，会在文末给出所有的链接。

如果胖友有自己的疑问，欢迎在星球提问，我们一起整理吊吊的 Redis 面试题的大保健。

而题目的难度，芋艿尽量按照从容易到困难的顺序，逐步下去。

什么是 Redis ？

Redis ，全称 Remote Dictionary Server ，是一个基于内存的高性能 Key-Value 数据库。

Redis 已经成为互联网公司在缓存组件选择的唯一。例如说，在各种公有云上，缓存服务都是提供的 Redis。再例如说，招聘简历要求上，都会要求掌握 Redis 。

Redis 有什么优点？

1. 速度快

因为数据存在内存中，类似于 HashMap ，HashMap 的优势就是查找和操作的时间复杂度都是O（1）。

> Redis 本质上是一个 Key-Value 类型的内存数据库，很像 Memcached ，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。
>
> 因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 Key-Value 数据库。
>
> 如果我们查看在阿里云销售的 Redis 规格，最低的也是 8W QPS 。

2. 支持丰富数据类型

支持 String ，List，Set，Sorted Set，Hash 五种基础的数据结构。

> Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，此外单个 Value 的最大限制是 1GB，不像 Memcached只能保存 1MB 的数据，因此 Redis 可以用来实现很多有用的功能。比方说：

用他的 List 来做 FIFO 双向链表，实现一个轻量级的高性能消息队列服务。
用他的 Set 可以做高性能的 tag 系统等等。

同时，在基础的数据结构之上，还提供 Bitmap、HyperLogLog、GEO 等高级的数据结构。

如果面试想要加分，胖友一定要去看看这些高级的数据结构，面试与日常开发，必备神器。

3. 丰富的特性

订阅发布 Pub / Sub 功能
Key 过期策略
事务
支持多个 DB
计数
...

并且在 Redis 5.0 增加了 Stream 功能，一个新的强大的支持多播的可持久化的消息队列，提供类似 Kafka 的功能。

4. 持久化存储

Redis 提供 RDB 和 AOF 两种数据的持久化存储方案，解决内存数据库最担心的万一 Redis 挂掉，数据会消失掉。

5、高可用

内置 Redis Sentinel ，提供高可用方案，实现主从故障自动转移。

内置 Redis Cluster ，提供集群方案，实现基于槽的分片方案，从而支持更大的 Redis 规模。

Redis 有什么缺点？

1、由于 Redis 是内存数据库，所以，单台机器，存储的数据量，跟机器本身的内存大小。虽然 Redis 本身有 Key 过期策略，但是还是需要提前预估和节约内存。如果内存增长过快，需要定期删除数据。

另外，可使用 Redis Cluster、Codis 等方案，对 Redis 进行分区，从单机 Redis 变成集群 Redis 。

2、如果进行完整重同步，由于需要生成 RDB 文件，并进行传输，会占用主机的 CPU ，并会消耗现网的带宽。不过 Redis2.8 版本，已经有部分重同步的功能，但是还是有可能有完整重同步的。比如，新上线的备机。

3、修改配置文件，进行重启，将硬盘中的数据加载进内存，时间比较久。在这个过程中，Redis 不能提供服务。

Redis 和 Memcached 的区别有哪些？

芮芮：随着 Memcached 日渐没落，这个问题问的越来越少了。

1. Redis 支持复杂的数据结构

Memcached 仅提供简单的字符串。
Redis 提供复杂的数据结构，丰富的数据操作。

也因为 Redis 支持复杂的数据结构，Redis 即使晚于 Memcached 推出，却获得更多开发者的青睐。

Redis 相比 Memcached 来说，拥有更多的数据结构，能支持更丰富的数据操作。如果需要缓存能够支持更复杂的结构和操作，Redis 会是不错的选择。

2. Redis 原生支持集群模式

   在 Redis3.x 版本中，官方便能支持 Cluster 模式。
   Memcached 没有原生的集群模式，需要依靠客户端来实现往集群中分片写入数据。

3. 性能对比

   Redis 只使用单核，而 Memcached 可以使用多核，所以平均每一个核上 Redis在存储小数据时比 Memcached 性能更高。
   在 100k 以上的数据中，Memcached 性能要高于 Redis 。虽然 Redis 最近也在存储大数据的性能上进行优化，但是比起 Memcached，还是稍有逊色。

更多关于性能的对比，可以看看 《Memcached 与 Redis 的关键性能指标比较》 。

4. 内存管理机制不同

相比来说，Redis 的内存管理机制，会更加简单。

   Redis 采用的是包装的 malloc/free ，使用时现场申请的方式。
   Memcached 采用的是 Slab Allocation 机制管理内存，预分配的内存池的方式。

如果对比两者的内存使用效率：

   简单的 Key-Value 存储的话，Memcached 的内存利用率更高，可以使用类似内存池。
   如果 Redis 采用 hash 结构来做 key-value 存储，由于其组合式的压缩， 其内存利用率会高于 Memcached 。

5. 网络 IO 模型

   Memcached 是多线程，非阻塞 IO 复用的网络模型，原型上接近 Nignx 。
   Redis 使用单线程的 IO 复用模型，自己封装了一个简单的 AeEvent 事件处理框架，主要实现了 epoll ， kqueue 和 select ，更接近 Apache 早期的模式。

6. 持久化存储

   Memcached 不支持持久化存储，重启时，数据被清空。
   Redis 支持持久化存储，重启时，可以恢复已持久化的数据。

也推荐阅读下 《脚踏两只船的困惑 - Memcached 与 Redis》 。

请说说 Redis 的线程模型？

   芳芳：这个是我从网络上找的资料，讲的灰常不错。一般来说，回答道 Redis 是非阻塞 IO ，多路复用。

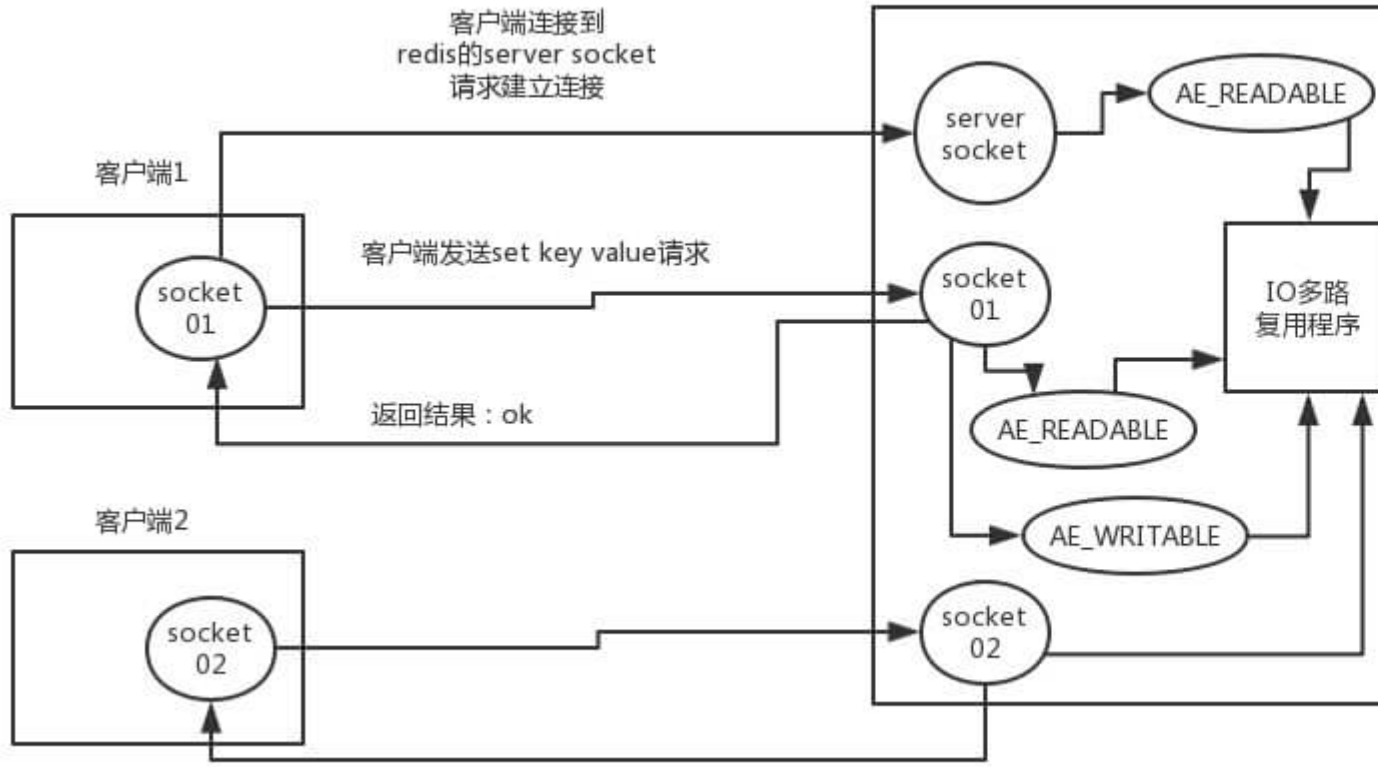
Redis 内部使用文件事件处理器 `file event handler`，这个文件事件处理器是单线程的，所以 Redis 才叫做单线程的模型。它采用 IO 多路复用机制同时监听多个 Socket，根据 Socket 上的事件来选择对应的事件处理器进行处理。

文件事件处理器的结构包含 4 个部分：

多个 Socket 。
IO 多路复用程序。
文件事件分派器。
事件处理器（连接应答处理器、命令请求处理器、命令回复处理器）。

多个 Socket 可能会并发产生不同的操作，每个操作对应不同的文件事件，但是 IO 多路复用程序会监听多个 socket，会将 socket 产生的事件放入队列中排队，事件分派器每次从队列中取出一个事件，把该事件交给对应的事件处理器进行处理。

来看客户端与 redis 的一次通信过程：



客户端 Socket01 向 Redis 的 Server Socket 请求建立连接，此时 Server Socket 会产生一个 AE_READABLE 事件，IO 多路复用程序监听到 server socket 产生的事件后，将该事件压入队列中。文件事件分派器从队列中获取该事件，交给连接应答处理器。连接应答处理器会创建一个能与客户端通信的 Socket01，并将该 Socket01 的 AE_READABLE 事件与命令请求处理器关联。

假设此时客户端发送了一个 set key value 请求，此时 Redis 中的 Socket01 会产生 AE_READABLE 事件，IO 多路复用程序将事件压入队列，此时事件分派器从队列中获取到该事件，由于前面 Socket01 的 AE_READABLE 事件已经与命令请求处理器关联，因此事件分派器将事件交给命令请求处理器来处理。命令请求处理器读取 Scket01 的 set key value 并在自己内存中完成 set key value 的设置。操作完成后，它会将 Scket01 的 AE_WRITABLE 事件与令回复处理器关联。

如果此时客户端准备好接收返回结果了，那么 Redis 中的 Socket01 会产生一个 AE_WRITABLE 事件，同样压入队列中，事件分派器找到相关联的命令回复处理器，由命令回复处理器对 socket01 输入本次操作的一个结果，比如 ok，之后解除 Socket01 的 AE_WRITABLE 事件与命令

回复处理器的关联。

这样便完成了一次通信。　　耐心理解一下，灰常重要。如果还是不能理解，可以在网络上搜一些资料，在理解理解。

为什么 Redis 单线程模型也能效率这么高？

1、C 语言实现。

我们都知道，C 语言的执行速度非常快。

2、纯内存操作。

Redis 为了达到最快的读写速度，将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 Redis 具有快速和数据持久化的特征。

如果不将数据放在内存中，磁盘 I/O 速度为严重影响 Redis 的性能。

3、基于非阻塞的 IO 多路复用机制。

4、单线程，避免了多线程的频繁上下文切换问题。

Redis 利用队列技术，将并发访问变为串行访问，消除了传统数据库串行控制的开销。

实际上，Redis 4.0 开始，也开始有了一些异步线程，用于处理一些耗时操作。例如说，异步线程，实现惰性删除（解决大 KEY 删除，阻塞主线程）和异步 AOF（解决磁盘 IO 紧张时，fsync 执行一次很慢）等等。

5、丰富的数据结构。

Redis 全程使用 hash 结构，读取速度快，还有一些特殊的数据结构，对数据存储进行了优化。例如，压缩表，对短数据进行压缩存储；再再如，跳表，使用有序的数据结构加快读取的速度。

也因为 Redis 是单线程的，所以可以实现丰富的数据结构，无需考虑并发的问题。

Redis 是单线程的，如何提高多核 CPU 的利用率？

可以在同一个服务器部署多个 Redis 的实例，并把他们当作不同的服务器来使用，在某些时候，无论如何一个服务器是不够的， 所以，如果你想使用多个 CPU ，你可以考虑一下分区。

Redis 有几种持久化方式？

芮芮：这个问题有一丢丢长，耐心看完。

面试的时候，如果不能完整回答出来，也不会有大问题。重点，在于有条理，对 RDB 和 AOF 有理解。

持久化方式

Redis 提供了两种方式，实现数据的持久化到硬盘。

1、【全量】RDB 持久化，是指在指定的时间间隔内将内存中的数据集快照写入磁盘。实际操作过程是，fork 一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。
2、【增量】AOF持久化，以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

RDB 优缺点

① 优点

灵活设置备份频率和周期。你可能打算每个小时归档一次最近 24 小时的数据，同时还要每天归档一次最近 30 天的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。
非常适合冷备份，对于灾难恢复而言，RDB 是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上。推荐，可以将这种完整的数据文件发送到一些远程的安全存储上去，比如说 Amazon 的 S3 云服务上去，在国内可以是阿里云的 OSS 分布式存储上。
性能最大化。对于 Redis 的服务进程而言，在开始持久化时，它唯一需要做的只是 fork 出子进程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行 IO 操作了。也就是说，RDB 对 Redis 对外提供的读写服务，影响非常小，可以让 Redis 保持高性能。
恢复更快。相比于 AOF 机制，RDB 的恢复速度更更快，更适合恢复数据，特别是在数据集非常大的情况。

② 缺点

如果你想保证数据的高可用性，即最大限度的避免数据丢失，那么 RDB 将不是一个很好的选择。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。

所以，RDB 实际场景下，需要和 AOF 一起使用。

由于 RDB 是通过 fork 子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是 1 秒钟。

所以，RDB 建议在业务低估，例如在半夜执行。

AOF 优缺点

① 优点

该机制可以带来更高的数据安全性，即数据持久性。Redis 中提供了 3 种同步策略，即每秒同步、每修改(执行一个命令)同步和不同步。

- 事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。
- 而每修改同步，我们可以将其视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。
- 至于不同步，无需多言，我想大家都能正确的理解它。

由于该机制对日志文件的写入操作采用的是 append 模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。

- 因为以 append-only 模式写入，所以没有任何磁盘寻址的开销，写入性能非常高。
- 另外，如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在

Redis 下一次启动之前，我们可以通过 redis-check-aof 工具来帮助我们解决数据一致性的问题。

如果 AOF 日志过大，Redis 可以自动启用 rewrite 机制。即使出现后台重写操作，也不会影响客户端的读写。因为在 rewrite log 的时候，会对其中的指令进行压缩，创建出一份需要恢复数据的最小日志出来。再创建新日志文件的时候，老的日志文件还是照常写入。当新的 merge 后的日志文件 ready 的时候，再交换新老日志文件即可。

注意，AOF rewrite 机制，和 RDB 一样，也需要 fork 出一次子进程，如果 Redis 内存比较大，可能会因为 fork 阻塞下主进程。

AOF 包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的重建。

② 缺点

对于相同数量的数据集而言，AOF 文件通常要大于 RDB 文件。RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快。
根据同步策略的不同，AOF 在运行效率上往往会慢于 RDB 。总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和 RDB 一样高效。
以前 AOF 发生过 bug ，就是通过 AOF 记录的日志，进行数据恢复的时候，没有恢复一模一样的数据出来。所以说，类似 AOF 这种较为复杂的基于命令日志/merge/回放的方式，比基于 RDB 每次持久化一份完整的数据快照文件的方式，更加脆弱一些，容易有 bug 。不过 AOF 就是为了避免 rewrite 过程导致的 bug ，因此每次 rewrite 并不是基于旧的指令日志进行 merge 的，而是基于当时内存中的数据进行指令的重新构建，这样健壮性会好很多。

如何选择

不要仅仅使用 RDB，因为那样会导致你丢失很多数据。

也不要仅仅使用 AOF，因为那样有两个问题，第一，你通过 AOF 做冷备，没有 RDB 做冷备，来的恢复速度更快; 第二，RDB 每次简单粗暴生成数据快照，更加健壮，可以避免 AOF 这种复杂的备份和恢复机制的 bug 。

Redis 支持同时开启开启两种持久化方式，我们可以综合使用 AOF 和 RDB 两种持久化机制，用 AOF 来保证数据不丢失，作为数据恢复的第一选择; 用 RDB 来做不同程度的冷备，在 AOF 文件都丢失或损坏不可用的时候，还可以使用 RDB 来进行快速的数据恢复。

- 如果同时使用 RDB 和 AOF 两种持久化机制，那么在 Redis 重启的时候，会使用 AOF 来重新构建数据，因为 AOF 中的数据更加完整。

  一般来说， 如果想达到足以媲美 PostgreSQL 的数据安全性， 你应该同时使用两种持久化功能。如果你非常关心你的数据， 但仍然可以承受数分钟以内的数据丢失，那么你可以只使用 RDB 持久化。

  有很多用户都只使用 AOF 持久化，但并不推荐这种方式：因为定时生成 RDB 快照（snapshot）非常便于进行数据库备份， 并且 RDB 恢复数据集的速度也要比 AOF恢复的速度要快，除此之外，使用 RDB 还可以避免之前提到的 AOF 程序的 bug。

在 Redis4.0 版本开始，允许你使用 RDB-AOF 混合持久化方式，详细可见 《Redis4.0 之 RDB-AOF 混合持久化》 。也因此，RDB 和 AOF 同时使用，是希望达到安全的持久化的推荐方式。

另外，RDB 和 AOF 涉及的知识点蛮多的，可以看看：

《Redis 设计与实现 —— RDB》
《Redis 设计与实现 —— AOF》

如下是老钱对这块的总结，可能更加适合面试的场景：

> bgsave 做镜像全量持久化，AOF 做增量持久化。因为 bgsave 会耗费较长时间，不够实时，在停机的时候会导致大量丢失数据，所以需要 AOF 来配合使用。在 Redis 实例重启时，会使用 bgsave 持久化文件重新构建内存，再使用 AOF 重放近期的操作指令来实现完整恢复重启之前的状态。
>
> > 和老钱沟通了下，最后一句重启恢复，使用的是 RDB-AOF 的混合方案。
>
> 对方追问那如果突然机器掉电会怎样？取决于 AOF 日志 sync 属性的配置，如果不要求性能，在每条写指令时都 sync 一下磁盘，就不会丢失数据。但是在高性能的要求下每次都 sync 是不现实的，一般都使用定时 sync ，比如 1 秒 1 次，这个时候最多就会丢失 1 秒的数据。
>
> > 实际上，极端情况下，是最多丢失 2 秒的数据。因为 AOF 线程，负责每秒执行一次 fsync 操作，操作完成后，记录最后同步时间。主线程，负责对比上次同步时间，如果超过 2 秒，阻塞等待成功。
>
> 对方追问 bgsave 的原理是什么？你给出两个词汇就可以了，fork 和 cow 。fork 是指 Redis 通过创建子进程来进行 bgsave 操作。cow 指的是 copy on write ，子进程创建后，父子进程共享数据段，父进程继续提供读写服务，写脏的页面数据会逐渐和子进程分离开来。
>
> > 芮芮：这里 bgsave 操作后，会产生 RDB 快照文件。

为什么不建议在主 Redis 节点开启 RDB 功能呢？因为会带来一定时间的阻塞，特别是数据量大的时候。

> 如下来自球友【jian】的回答，感恩~
>
> > 【重点】子进程 fork 相关的阻塞：在 bgsave 的时候，Redis 主进程会 fork 一个子进程，利用操作系统的写时复制技术，这个子进程在拷贝父进程的时候理论上是很快的，因为并不需要全拷贝，比如主进程虽然占了 10G 内存，但子进程拷贝他可能只要 200 毫秒，我认为也就阻塞了 200 毫秒(此耗时基本跟主进程占用的内存是成正比的)，这个具体的时间可以通过统计项 info stats 里的 last_fork_usec 查看。
> > CPU 单线程相关的阻塞：Redis 主进程是单线程跑在单核 CPU 上的，如果显示绑定了CPU ，则子进程会与主进程共享一个 CPU ，而子进程进行持久化的时候是非常占CPU（强势 90%），因此这种情况也可能导致提供服务的主进程发生阻塞（因此如果需要持久化功能，不建议绑定CPU）。
> > 内存相关的阻塞：虽然利用写时复制技术可以大大降低进程拷贝的内存消耗，但这也导致了父进程在处理写请求时需要维护修改的内存页，因此这部分内存过大的话（修改页数多或每页占空间大）也会导致父进程的写操作阻塞。（而不巧的是，Linux中TransparentHugePage 会将复制内存页面单位有 4K 变成 2M ，这对于 Redis 来说是比较不友好的，也是建议优化的，具体可百度之）
> > 磁盘相关的阻塞：极端情况下，假设整个机器的内存已经所剩无几，触发了内存交换（SWAP），则整个 Redis的效率将会非常低下（显然这不仅仅针对 save/bgsave），因此，关注系统的 io 情况，也是定位阻塞问题的一种方法。
>
> 芮芮后来又看了下这个答案，是 《Redis 开发与运维》 的「5.3 持久化 —— 问题定位于优化」小节。

Redis 有几种数据“过期”策略？

Redis 的过期策略，就是指当 Redis 中缓存的 key 过期了，Redis 如何处理。

Redis 提供了 3 种数据过期策略：

> 被动删除：当读/写一个已经过期的 key 时，会触发惰性删除策略，直接删除掉这个过期 key 。
> 主动删除：由于惰性删除策略无法保证冷数据被及时删掉，所以 Redis 会定期主动淘汰一批已过期的 key 。
> 主动删除：当前已用内存超过 maxmemory 限定时，触发主动清理策略，即 「数据“淘汰”策略」 。

在 Redis 中，同时使用了上述 3 种策略，即它们非互斥的。

想要进一步了解，可以看看 《关于 Redis 数据过期策略》 文章。

Redis 有哪几种数据“淘汰”策略？

Redis 内存数据集大小上升到一定大小的时候，就会进行数据淘汰策略。

Redis 提供了 6 种数据淘汰策略：

1. volatile-lru
2. volatile-ttl
3. volatile-random
4. allkeys-lru
5. allkeys-random
6. 【默认策略】no-enviction

具体的每种数据淘汰策略的定义，和如何选择讨论策略，可见 《Redis实战（二） 内存淘汰机制》 。

在 Redis 4.0 后，基于 LFU（Least Frequently Used）最近最少使用算法，增加了 2 种淘汰策略：

1. volatile-lfu
2. allkeys-lfu

Redis LRU 算法

另外，Redis 的 LRU 算法，并不是一个严格的 LRU 实现。这意味着 Redis 不能选择最佳候选键来回收，也就是最久未被访问的那些键。相反，Redis 会尝试执行一个近似的 LRU 算法，通过采样一小部分键，然后在采样键中回收最适合(拥有最久未被访问时间)的那个。

Redis 没有使用真正实现严格的 LRU 算是的原因是，因为消耗更多的内存。然而对于使用 Redis 的应用来说，使用近似的 LRU 算法，事实上是等价的。

具体的可以看看如下文章：

> 《想不到！面试官问我：Redis 内存满了怎么办？》
> 《使用 Redis 作为一个 LRU 缓存》

MySQL 里有 2000w 数据，Redis 中只存 20w 的数据，如何保证 Redis 中的数据都是热点数据

？

> 芳芳：这个是从网络上找到的一个神奇的问题，并且看了答案之后，觉得有点莫名的对不上。
>
> 所以，感觉这个问题的目的是，如何保证热点数据不要被淘汰。

在 「Redis 有哪几种数据“淘汰”策略？」 问题中，我们已经看到，“Redis 内存数据集大小上升到一定 maxmemory 的时候，就会进行数据淘汰策略。” 。

那么，如果我们此时要保证热点数据不被淘汰，那么需要选择 volatile-lru 或 allkeys-lru 这两个基于 LRU 算法的淘汰策略。

相比较来说，最终会选择 allkeys-lru 淘汰策略。原因是，如果我们的应用对缓存的访问符合幂律分布，也就是存在相对热点数据，或者我们不太清楚我们应用的缓存访问分布状况，我们可以选择 allkeys-lru 策略。如果在 Redis 4.0 版本，可以考虑使用 volatile-lfu ，更加符合“热”的概念，频率越高，代表越热。

Redis 回收进程如何工作的？

理解回收进程如何工作是非常重要的：

> 一个客户端运行了新的写命令，添加了新的数据。
> Redis 检查内存使用情况，如果大于 maxmemory 的限制，则根据设定好的策略进行回收。
> Redis 执行新命令。

所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下（跌宕起伏）。

如果有大量的 key 需要设置同一时间过期，一般需要注意什么？

如果大量的 key 过期时间设置的过于集中，到过期的那个时间点，Redis可能会出现短暂的卡顿现象。

一般需要在时间上加一个随机值，使得过期时间分散一些。

上次基友也碰到这个问题，请教了下，他的方案是调大 hz 参数，每次过期的 key 更多，从而最终达到避免一次过期过多。

> 这个定期的频率，由配置文件中的 hz 参数决定，代表了一秒钟内，后台任务期望被调用的次数。Redis 3.0.0 中的默认值是 10 ，代表每秒钟调用 10 次后台任务。
>
> hz 调大将会提高 Redis 主动淘汰的频率，如果你的 Redis 存储中包含很多冷数据占用内存过大的话，可以考虑将这个值调大，但 Redis 作者建议这个值不要超过 100 。我们实际线上将这个值调大到 100 ，观察到 CPU 会增加 2% 左右，但对冷数据的内存释放速度确实有明显的提高（通过观察 keyspace 个数和 used_memory 大小）。

Redis 有哪些数据结构？

如果你是 Redis 普通玩家，可能你的回答是如下五种数据结构：

> 字符串 String
> 字典Hash

列表List
集合Set
有序集合 SortedSet

如果你是 Redis 中级玩家，还需要加上下面几种数据结构：

HyperLogLog
Geo
Bitmap

如果你是 Redis 高端玩家，你可能玩过 Redis Module ，可以再加上下面几种数据结构：

BloomFilter
RedisSearch
Redis-ML
JSON

另外，在 Redis 5.0 增加了 Stream 功能，一个新的强大的支持多播的可持久化的消息队列，提供类似 Kafka 的功能。 默默跟面试官在装一波。

聊聊 Redis 使用场景

Redis 可用的场景非常之多：

数据缓存
会话缓存
时效性数据
访问频率
计数器
社交列表
记录用户判定信息
交集、并集和差集
热门列表与排行榜
最新动态
消息队列
分布式锁

详细的介绍，可以看看如下文章：

《聊聊 Redis 使用场景》
《Redis 应用场景及实例》
《Redis 常见的应用场景解析》
《Redis 和 Memcached 各有什么优缺点，主要的应用场景是什么样的？》

Redis 支持的 Java 客户端都有哪些？

使用比较广泛的有三个 Java 客户端：

Redisson

Redisson ，是一个高级的分布式协调 Redis 客服端，能帮助用户在分布式环境中轻松实现一些 Java 的对象 (Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap,

Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog)。

Jedis

Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持。

Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，Jedis 功能较为简单。

Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

Lettuce

Lettuce 是一个可伸缩线程安全的 Redis 客户端。多个线程可以共享同一个 RedisConnection 。它利用优秀 Netty NIO 框架来高效地管理多个连接。

Redis 官方推荐使用 Redisson 或 Jedis 。

Spring Boot 2.x 内置支持 Jedis 和 Lettuce 。一般情况下，建议：

使用 Spring Data Redis ，提供了透明使用 Jedis 和 Lettuce 的封装。也就是说，大多数时候，我们可以通过配置使用 Jedis 或 Lettuce 进行 Redis 的操作，而上层使用 Spring Data Redis 提供的统一 API 。
从目前来说，Jedis 会比 Lettuce 更加流行，并且更加稳定。虽然说 Jedis 有一段时间，不再进行更新，但是突然又开始更新，可能是诈尸了。
如果想要更加丰富的特性，例如说分布式锁，布隆过滤器，可以考虑研究下 Redisson 。

如何使用 Redis 实现分布式锁？

Redis 实现分布式锁，需要考虑如下几个方面：

1、正确的获得锁

set 指令附带 nx 参数，保证有且只有一个进程获得到。

2、正确的释放锁

使用 Lua 脚本，比对锁持有的是不是自己。如果是，则进行删除来释放。

3、超时的自动释放锁

set 指令附带 expire 参数，通过过期机制来实现超时释放。

4、未获得到锁的等待机制

sleep 或者基于 Redis 的订阅 Pub/Sub 机制。

一些业务场景，可能需要支持获得不到锁，直接返回 false ，不等待。

5、【可选】锁的重入性

通过 ThreadLocal<Integer> 记录是第几次获得相同的锁。

1）有且第一次计数为 1 && 获得锁时，才向 Redis 发起获得锁的操作。 2）有且计数为 0 && 释放锁时，才向 Redis 发起释放锁的操作。

6、锁超时的处理

一般情况下，可以考虑告警 + 后台线程自动续锁的超时时间。通过这样的机制，保证有且仅有一个线程，正在持有锁。

7、Redis 分布式锁丢失问题

具体看「方案二：Redlock」。

下面，我们来详细说下每个方案。

方案一：set 指令

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。

这时候对方会告诉你说你回答得不错，然后接着问如果在 setnx 之后执行 expire 之前进程意外 crash 或者要重启维护了，那会怎么样？
这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和 expire 合成一条指令来用的！对方这时会显露笑容，心里开始默念：摁，这小子还不错。

所以，我们可以使用 set 指令，实现分布式锁。指令如下：

```
SET key value [EX seconds] [PX milliseconds] [NX|XX]
```

可以使用 `SET key value EX seconds NX` 命令，尝试获得锁。
具体的实现，可以参考如下文章：

- 《精尽 Redisson 源码分析 —— 可重入分布式锁 ReentrantLock》
- 《Redis 分布式锁进化史解读 + 缺陷分析》
- 《Redis 分布式锁的正确实现方式（Java 版）》

方案二：Redlock

set 指令的方案，适合用于在单机 Redis 节点的场景下，在多 Redis 节点的场景下，会存在分布式锁丢失的问题。所以，Redis 作者 Antirez 基于分布式环境下提出了一种更高级的分布式锁的实现方式：Redlock 。

具体的源码解析，可以看看 《精尽 Redisson 源码分析 —— 可靠分布式锁 RedLock》 文章。

具体的方案，胖友可以看看老友飞哥的两篇博客：

《Redlock：Redis分布式锁最牛逼的实现》
《Redisson 实现 Redis 分布式锁的 N 种姿势》

最近芳芳画了一个 Redisson 实现分布式锁的流程图，胖友可以点击传送门阅读。

对比 Zookeeper 分布式锁

从可靠性上来说，Zookeeper 分布式锁好于 Redis 分布式锁。
从性能上来说，Redis 分布式锁好于 Zookeeper 分布式锁。

所以，没有绝对的好坏，可以根据自己的业务来具体选择。如果想要更简单，甚至可以考虑基于 MySQL 行锁来实现分布式锁。

如何使用 Redis 实现分布式限流？

在 Spring Cloud Gateway 中，提供了 Redis 分布式限流器的实现，具体直接看艿艿写的 《Spring-Cloud-Gateway 源码解析 —— 过滤器 (4.10) 之 RequestRateLimiterGatewayFilterFactory 请求限流》 的 「5.3 Redis Lua 脚本」 部分。

另外，Redisson 库中，也提供了 Redis 分布式限流的实现，不过需要使用 Pro 版本。

请用 Redis 和任意语言实现一段恶意登录保护的代码，限制 1 小时内每用户 Id 最多只能登录 5 次。

这个问题，关键点，就是每个用户，每 3600 秒，只能登陆 5 次。这么一想，其实就是一个如何使用 Redis 实现限流的问题。Redis 实现限流，一共有两种方案：

使用 zset 实现滑动窗口限流。代码如下：

```
public boolean isActionAllowed(String userId, String actionKey, int period,
    int maxCount) {
    String key = String.format("hist:%s:%s", userId, actionKey); // 使用用户编号 + 行为作为 KEY 。这样，我们就
    long nowTs = System.currentTimeMillis(); // 获取当前时间。
    Pipeline pipe = jedis.pipelined(); // pipeline 批量操作，提升效率。
    pipe.multi(); // 此处启动了事务，可以保证指令的原子性。
    pipe.zadd(key, nowTs, "" + nowTs); // zset 添加，key value score 要看下。
    pipe.zremrangeByScore(key, 0, nowTs - (period * 1000)); // zremrangeByScore ，移除超过周期的 value 。

    Response<Long> count = pipe.zcard(key); // zcard ，计算 zset 的数量
    pipe.expire(key, period + 1); // 设置过期。这里多 + 1 秒，为了防止网络延迟。
    pipe.exec(); // pipeline 执行
    pipe.close();

    return count.get() <= maxCount; // 是否超过最大次数。
}
```

- 该实现会存在一个问题，可能一个无效的操作，也被记录到次数中。完美的话，可能需要基于 Lua 脚本实现。
- 另外，上述代码是每秒操作的时间，实际需要改成每 N 秒。比较简单，直接上手怼即可。

使用 Lua 脚本，实现令牌桶限流算法。具体可以看看艿艿对 《Spring-Cloud-Gateway 源码解析 —— 过滤器 (4.10) 之 RequestRateLimiterGatewayFilterFactory 请求限流》 的源码解析。

使用 Lua 脚本，实现简单的滑动窗口。具体可以看看艿艿对 《精尽 Redisson 源码分析 —— 限流器 RateLimiter》 的源码解析。

如何使用 Redis 实现消息队列？

一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。

如果对方追问可不可以不用 sleep 呢？list 还有个指令叫 blpop ，在没有消息的时候，它

会阻塞住直到消息到来。

如果对方追问能不能生产一次消费多次呢？使用 pub / sub 主题订阅者模式，可以实现 1:N 的消息队列。

如果对方追问 pub / sub 有什么缺点？在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 rabbitmq 等。

> 之前生产中，芮芮就碰到因为网络闪断，导致订阅的 pub/sub 消息丢失，导致 JVM 应用的数据字典和系统参数等缓存未刷新，业务受到影响。所以，最好还是使用专业的消息队列的订阅功能（广播消费）。

如果对方追问 redis 如何实现延时队列？我估计现在你很想把面试官一棒打死如果你手上有一根棒球棍的话，怎么问的这么详细。但是你很克制，然后神态自若的回答道：使用 sortedset ，拿时间戳作为 score ，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。

> 可能很多胖友会觉得抽象，可以看看 《Redis 学习笔记之延时队列》 。面试中，能回答到 Redis zset 实现延迟队列，还是蛮加分的。

到这里，面试官暗地里已经对你竖起了大拇指。但是他不知道的是此刻你却竖起了中指，在椅子背后。

当然，实际上 Redis 真的真的真的不推荐作为消息队列使用，它最多只是消息队列的存储层，上层的逻辑，还需要做大量的封装和支持。

另外，在 Redis 5.0 增加了 Stream 功能，一个新的强大的支持多播的可持久化的消息队列，提供类似 Kafka 的功能。

什么是 Redis Pipelining ？

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应。这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

> 注意，Redis Pipelining 是 Redis Client 实现的功能，而不是 Redis Server 提供的特性。假设我们有 3 个请求进行下举例子。
>
> > 未使用 Pipeline 时，那么整个执行的顺序是，req1->resp1->req2->resp2->req3->resp3 。
> > 在使用 Pipeline 时，那么整个执行的顺序是，[req1,req2,req3] 一起发给 Redis Server ，而 Redis Server 收到请求后，一个一个请求进行执行，然后响应，不会进行什么特殊处理。而 Client 在收到 resp1,resp2,resp3 后，进行响应给业务上层。
>
> 所以，Pipeline 的作用，是避免每发一个请求，就阻塞等待这个请求的结果。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多 POP3 协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

Redis 很早就支持管道（pipelining）技术，因此无论你运行的是什么版本，你都可以使用管道（pipelining）操作 Redis。

Redis 如何做大量数据插入？

Redis 2.6 开始，Redis-cli 支持一种新的被称之为 pipe mode 的新模式用于执行大量数据插入工

作。

具体可见 《Redis 大量数据插入》 文章。

什么是 Redis 事务？

和众多其它数据库一样，Redis 作为 NoSQL 数据库也同样提供了事务机制。在 Redis 中，MULTI / EXEC / DISCARD / WATCH 这四个命令是我们实现事务的基石。相信对有关系型数据库开发经验的开发者而言这一概念并不陌生，即便如此，我们还是会简要的列出 Redis 中事务的实现特征：

1、在事务中的所有命令都将会被串行化的顺序执行，事务执行期间，Redis 不会再为其它客户端的请求提供任何服务，从而保证了事物中的所有命令被原子的执行。

Lua 脚本，也能实现该功能。

2、和关系型数据库中的事务相比，在 Redis 事务中如果有某一条命令执行失败，其后的命令仍然会被继续执行。

这一点，非常重要。回答错了，就回家面壁思过，一天不许喝可乐。

这一点，是 Lua 脚本不具备的。

3、我们可以通过 MULTI 命令开启一个事务，有关系型数据库开发经验的人可以将其理解为 "BEGIN TRANSACTION" 语句。在该语句之后执行的命令，都将被视为事务之内的操作，最后我们可以通过执行 EXEC / DISCARD 命令来提交 / 回滚该事务内的所有操作。这两个 Redis 命令，可被视为等同于关系型数据库中的 COMMIT / ROLLBACK 语句。

开启事务后，所有语句，发送给 Redis Server ，都会暂存在 Server 中。

4、在事务开启之前，如果客户端与服务器之间出现通讯故障并导致网络断开，其后所有待执行的语句都将不会被服务器执行。然而如果网络中断事件是发生在客户端执行 EXEC 命令之后，那么该事务中的所有命令都会被服务器执行。

如何实现 Redis CAS 操作？

在 Redis 的事务中，WATCH 命令可用于提供 CAS(check-and-set) 功能。

假设我们通过 WATCH 命令在事务执行之前监控了多个 keys ，倘若在 WATCH 之后有任何 Key 的值发生了变化，EXEC 命令执行的事务都将被放弃，同时返回 nil 应答以通知调用者事务执行失败。

具体的示例，可以看看 《Redis 事务锁 CAS 实现以及深入误区》 。

Redis 集群都有哪些方案？

Redis 集群方案如下：

1、Redis Sentinel
2、Redis Cluster
3、Twemproxy
4、Codis
5、客户端分片

关于前四种，可以看看 《Redis 实战（四）集群机制》 这篇文章。

关于最后一种，客户端分片，在 Redis Cluster 出现之前使用较多，目前已经使用比较少了。实现方式如下：

> 在业务代码层实现，起几个毫无关联的 Redis 实例，在代码层，对 Key 进行 hash 计算，然后去对应的 Redis 实例操作数据。
>
> 这种方式对 hash 层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

选择

目前一般在选型上来说：

> 体量较小时，选择 Redis Sentinel ，单主 Redis 足以支撑业务。
>
> 体量较大时，选择 Redis Cluster ，通过分片，使用更多内存。
>
> > 关于这个问题，多大体量需要使用 Redis Cluster 呢？朋友的建议是 10G+ 的时候。主要原因是：
> >
> > - 1、一次 RDB 时间随着内存越大，会变大越来越久。同时，一次 fork 的时间也会变久。还有，重启通过 RDB 文件，或者 AOF 日志，恢复时间都会变长。
> > - 2、体量大之后，读写的 QPS 势必比体量小的时候打的多，那么使用 Redis Cluster 相比 Redis Sentinel ，可以分散读写压力到不同的集群中。

Redis 集群如何扩容？

> ~~如果 Redis 被当做缓存使用，使用一致性哈希实现动态扩容缩容。~~
>
> > 删除的原因是，不考虑客户端分片的情况，目前基本已经不在用了。
>
> 如果 Redis 被当做一个持久化存储使用，必须使用固定的 keys-to-nodes 映射关系，节点的数量一旦确定不能变化。否则的话(即 Redis 节点需要动态变化的情况），必须使用可以在运行时进行数据再平衡的一套系统，而当前只有 Redis Cluster、Codis 可以做到这样。

如果是 Redis Cluster 集群的扩容，可以看看 《Redis 开发与运维》 的「10.4 集群 —— 集群伸缩」小节。简单来说，一共三步：

> 1、准备新节点。
> 2、加入集群。
> 3、迁移槽和数据。

什么是 Redis 主从同步？

Redis 主从同步

Redis 的主从同步(replication)机制，允许 Slave 从 Master 那里，通过网络传输拷贝到完整的数据备份，从而达到主从机制。

> 主数据库可以进行读写操作，当发生写操作的时候自动将数据同步到从数据库，而从数据库一般是只读的，并接收主数据库同步过来的数据。
> 一个主数据库可以有多个从数据库，而一个从数据库只能有一个主数据库。
> 第一次同步时，主节点做一次 bgsave 操作，并同时将后续修改操作记录到内存 buffer ，待完成后将 RDB 文件全量同步到复制节点，复制节点接受完成后将 RDB 镜像加载到内存。加载

完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

好处

通过 Redis 的复制功，能可以很好的实现数据库的读写分离，提高服务器的负载能力。主数据库主要进行写操作，而从数据库负责读操作。

实际上，我们不是非常推荐在 Redis 中，使用读写分离。主要有两个原因：

> Redis Sentinel 只保证主节点的故障的失效转移，而例如说 Jedis 库，也只监听了主节点的变化，但是从节点故障的情况，Jedis 是不进行处理的。这就会导致，Jedis 读会访问到从节点，导致问题。当然，Redisson 库的功能比较强大，已经支持从节点的故障监听。
> 如果到达需要读写分离的体量，一般写操作也不一定会少，可以考虑上 Redis Cluster 方案，更加可靠。

Redis 主从同步，是很多 Redis 集群方案的基础，例如 Redis Sentinel、Redis Cluster 等等。

更多详细，可以看看如下：

> 因为主从复制的内容很多，芋艿这里就不详细哔哔了。实际场景下，对于开发的面试，我们也不会特别问，毕竟更偏运维的内容。
>
> 《Redis 官方文档 —— 复制》
> 《Redis 开发与运维》 的「6. 复制」章节，更加详细完整。

<!--《Redis 主从架构》 。-->

如何使用 Redis Sentinel 实现高可用？

详细，可以看看如下：

> 因为 Redis Sentinel 的内容很多，芋艿这里就不详细哔哔了。实际场景下，对于开发的面试，我们也不会特别问，毕竟更偏运维的内容。
>
> 《Redis 官方文档 —— Sentinel 高可用》
> 《Redis 开发与运维》 的「9. 哨兵」章节，更加详细完整。

<!--《Redis 哨兵集群实现高可用》 。-->

如果使用 Redis Cluster 实现高可用？

详细，可以看看如下：

> 因为 Redis Sentinel 的内容很多，芋艿这里就不详细哔哔了。实际场景下，对于开发的面试，我们也不会特别问，毕竟更偏运维的内容。
>
> 《Redis 官方文档 —— Redis Cluster 集群》
> 《Redis 开发与运维》 的「10. 集群」章节，更加详细完整。

<!--《Redis 集群模式的工作原理能说一下么？》 精简版-->

说说 Redis 哈希槽的概念？

Redis Cluster 没有使用一致性 hash ，而是引入了哈希槽的概念。

Redis 集群有 16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

因为最大是 16384 个哈希槽，所以考虑 Redis 集群中的每个节点都能分配到一个哈希槽，所以最多支持 16384 个 Redis 节点。

为什么是 16384 呢？主要考虑集群内的网络带宽，而 16384 刚好是 2K 字节大小。

Redis Cluster 的主从复制模型是怎样的？

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型，每个节点都会有 N-1 个复制节点。

所以，Redis Cluster 可以说是 Redis Sentinel 带分片的加强版。也可以说：

> Redis Sentinel 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master ，继续提供服务。
> Redis Cluster 着眼于扩展性，在单个 Redis 内存不足时，使用 Cluster 进行分片存储。

Redis Cluster 方案什么情况下会导致整个集群不可用？

有 A，B，C 三个节点的集群，在没有复制模型的情况下，如果节点 B 宕机了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。当然，这种情况也可以配置 `cluster-require-full-coverage=no` ，整个集群无需所有槽位覆盖。

Redis Cluster 会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，而是【异步复制】，这意味这在实际中集群在特定的条件下可能会丢失写操作。

> 艿艿：一定一定一定要注意，无论对于 Redis Sentinel 还是 Redis Cluster 方案，都是通过主从复制，所以在数据的复制方面，都存在相同的情况。

Redis 集群如何选择数据库？

Redis 集群目前无法做数据库选择，默认在 0 数据库。

请说说生产环境中的 Redis 是怎么部署的？

> 重点问题，仔细理解。
>
> Redis Cluster ，10 台机器，5 台机器部署了 Redis 主实例，另外 5 台机器部署了 Redis 的从实例，每个主实例挂了一个从实例，5 个节点对外提供读写服务，每个节点的读写高峰 qps 可能可以达到每秒 5 万，5 台机器最多是 25 万读写请求每秒。
>
> 机器是什么配置？32G 内存 + 8 核 CPU + 1T 磁盘，但是分配给 Redis 进程的是 10G 内存，一般线上生产环境，Redis 的内存尽量不要超过 10G，超过 10G 可能会有问题。那么，5 台机器对外提供读写，一共有 50G 内存。
>
> 因为每个主实例都挂了一个从实例，所以是高可用的，任何一个主实例宕机，都会自动故障迁移，Redis 从实例会自动变成主实例继续提供读写服务。
>
> 你往内存里写的是什么数据？每条数据的大小是多少？商品数据，每条数据是 10kb 。100 条数据是 1mb ，10 万条数据是 1G 。常驻内存的是 200 万条商品数据，占用内存是 20G ，仅

仅不到总内存的 50% 。目前高峰期每秒就是 3500 左右的请求量。

> 一般来说，当公司体量大了之后，建议是一个业务线独占一个或多个 Redis Cluster 集群，实现好业务线与业务线之间的隔离。

其实大型的公司，会有基础架构的 Team 负责缓存集群的运维。

什么是 Redis 分区？

这个问题，和 「Redis 集群都有哪些方案？」 是同类问题。

简单看看即可，重点还是去理解 Redis Cluster 集群方案。

关于如下四个问题，直接看 《Redis 分区》 文章。

Redis 分区是什么？
分区的优势？
分区的不足？
分区类型？

可能有胖友会懵逼，又是 Redis 主从复制，又是 Redis 分区，又是 Redis 集群。傻傻分不清啊！

> Redis 分区是一种模式，将数据分区到不同的 Redis 节点上，而 Redis 集群的 Redis Cluster、Twemproxy、Codis、客户端分片（ 不包括 Redis Sentinel ） 这四种方案，是 Redis 分区的具体实现。
>
> > 注意，Redis Sentinel 实现的是 Redis 的高可用，一定要分清楚。实际上，胖友可以对比 MySQL 和 MongoDB 的高可用、集群的方案，发现思路都是一致的。
>
> Redis 每个分区，如果想要实现高可用，需要使用到 Redis 主从复制。

你知道有哪些 Redis 分区实现方案？

Redis 分区方案，主要分成两种类型：

> 客户端分区，就是在客户端就已经决定数据会被存储到哪个 Redis 节点或者从哪个 Redis 节点读取。大多数客户端已经实现了客户端分区。
>
> - 案例：Redis Cluster 和客户端分区。
>
> 代理分区，意味着客户端将请求发送给代理，然后代理决定去哪个节点写数据或者读数据。代理根据分区规则决定请求哪些 Redis 实例，然后根据 Redis 的响应结果返回给客户端。
>
> - 案例：Twemproxy 和 Codis 。

查询路由(Query routing)的意思，是客户端随机地请求任意一个 Redis 实例，然后由 Redis 将请求转发给正确的 Redis 节点。Redis Cluster 实现了一种混合形式的查询路由，但并不是直接将请求从一个Redis 节点转发到另一个 Redis 节点，而是在客户端的帮助下直接 Redirect 到正确的 Redis 节点。

> Redis Cluster 的重定向，可以认真看看 《Redis 开发与运维》 的「10.5 集群 - 请求路由」章节。

分布式 Redis 是前期做还是后期规模上来了再做好？为什么？？

如下是网络上的一个答案：

> 既然 Redis 是如此的轻量，为防止以后的扩容，最好的办法就是一开始就启动较多实例

。即便你只有一台服务器，你也可以一开始就让 Redis 以分布式的方式运行，使用分区，在同一台服务器上启动多个实例。

一开始就多设置几个 Redis 实例，例如 32 或者 64 个实例，对大多数用户来说这操作起来可能比较麻烦，但是从长久来看做这点牺牲是值得的。

这样的话，当你的数据不断增长，需要更多的 Redis 服务器时，你需要做的就是仅仅将 Redis 实例从一台服务迁移到另外一台服务器而已（而不用考虑重新分区的问题）。一旦你添加了另一台服务器，你需要将你一半的 Redis 实例从第一台机器迁移到第二台机器。

和飞哥沟通了下，这个操作不是很合理。
无论怎么说，建议，需要搭建下 Redis Sentinel 高可用，至于拓展性，根据自己的情况，是否使用 Redis Cluster 集群。同时， Redis Cluster 集群会有运维的复杂性，同时会存在跨分片操作（例如说 mget 等等）、事务等操作是不支持的。

Redis 有哪些重要的健康指标？

推荐阅读 《Redis 几个重要的健康指标》

存活情况
连接数
阻塞客户端数量
使用内存峰值
内存碎片率
缓存命中率
OPS
持久化
失效KEY
慢日志

如何提高 Redis 命中率？

推荐阅读 《如何提高缓存命中率（Redis）》 。

怎么优化 Redis 的内存占用？

推荐阅读 《Redis 的内存优化》

redisObject 对象
缩减键值对象
共享对象池
字符串优化
编码优化
控制 key 的数量

一个 Redis 实例最多能存放多少的 keys？List、Set、Sorted Set 他们最多能存放多少元素？

一个 Redis 实例，最多能存放多少的 keys ，List、Set、Sorted Set 他们最多能存放多少元素。

理论上，Redis 可以处理多达 2^32 的 keys ，并且在实际中进行了测试，每个实例至少存放了 2 亿 5 千万的 keys。

任何 list、set、和 sorted set 都可以放 2^32 个元素。

假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

使用 keys 指令可以扫出指定模式的 key 列表。

> 对方接着追问：如果这个 Redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？
> 这个时候你要回答 Redis 关键的一个特性：Redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

Redis 常见的性能问题都有哪些？如何解决？

1、Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件。

> 经过和朋友讨论，主节点开启 AOF 日志功能，尽量避免 AOF 重写。

- Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以 Master 最好不要写内存快照。
- Master AOF 持久化，如果不重写 AOF 文件，这个持久化方式对性能的影响是最小的，但是 AOF 文件会不断增大，AOF 文件过大会影响 Master 重启的恢复速度。
- 所以，Master 最好不要做任何持久化工作，包括内存快照和 AOF 日志文件，特别是不要启用内存快照做持久化。如果数据比较关键，某个 Slave 开启AOF备份数据，策略为每秒同步一次。

2、Master 调用 BGREWRITEAOF 重写 AOF 文件，AOF 在重写的时候会占大量的 CPU 和内存资源，导致服务 load 过高，出现短暂服务暂停现象。

- 一般来说，出现这个问题，很多时候是因为 Master 的内存过大，一次 AOF 重写需要占用的 CPU 和内存的资源较多，此时可以考虑 Redis Cluster 方案。

3、尽量避免在压力很大的主库上增加过多的从库。

- 可以考虑在从上挂载其它的从。

4、主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3... 。

- 这样的结构，也方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master挂了，可以立刻启用 Slave1 做 Master ，其他不变。

> 从节点在切换主节点作为复制源的时候，会重新发起全量复制。所以此处通过 Slave1 挂在 Slave 下，可以规避这个问题。同时，也减少了 Master 的复制压力。当然，坏处就是 Slave1 的延迟可能会高一些些，所以还是需要取舍。

5、Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave 和 Master 最好在同一个局域网内。

和飞哥沟通过后，他们主节点开启 AOF ，从节点开启 AOF + RDB 。

和晓峰沟通后，他们主节点开启 AOF ，从节点开启 RDB 居多，也有开启 AOF + RDB 的。

修改配置不重启 Redis 会实时生效吗？

针对运行实例，有许多配置选项可以通过 `CONFIG SET` 命令进行修改，而无需执行任何形式的重启。

从 Redis 2.2 开始，可以从 AOF 切换到 RDB 的快照持久性或其他方式而不需要重启 Redis。检索 `CONFIG GET *` 命令获取更多信息。

但偶尔重新启动是必须的，如为升级 Redis 程序到新的版本，或者当你需要修改某些目前 CONFIG 命令还不支持的配置参数的时候。

其他问题

有些比较凶残的面试官，可能会问我们一些 Redis 数据结构的问题，例如：

> Skiplist 插入和查询原理？
>
> 压缩列表的原理？
>
> Redis 底层为什么使用跳跃表而不是红黑树？
>
> > 跳跃表在范围查找的时候性能比较高。

想要了解这块，需要花一定的时间去撸一撸源码，推荐可以看如下两块内容：

> 《Redis 深度历险：核心原理与应用实践》
> 《Redis 设计与实现》

推荐先读第一本，可以深入浅出的了解 Redis 原理和源码。然后在读第二本，硬核了解 Redis 的设计与实现（源码）。

666. 彩蛋

哇哦，爽。虽然过程痛苦，但是中间请教了蛮多人问题，收获颇多哈。

嘿嘿，在回答问题的过程中，胖友会发现一直在推荐 《Redis 开发与运维》 这本书。在芮芮整理完第一版 Redis 面试题后，发现对有些 Redis 的面试题，理解还是有所欠缺（当然现在可能也是，哈哈哈），重新翻看了下这本书，发现很多问题都得到了非常不错的解答。所以，推荐再推荐。

参考与推荐如下文章：

> JeffreyLcm 《Redis 面试题》
> 烙印99 《史上最全 Redis 面试题及答案》
> yanglbme 《Redis 和 Memcached 有什么区别？Redis 的线程模型是什么？为什么单线程的 Redis 比多线程的 Memcached 效率要高得多？》
> 老钱 《天下无难试之 Redis 面试题刁难大全》
> yanglbme 《Redis 的持久化有哪几种方式？不同的持久化机制都有什么优缺点？持久化机制具体底层是如何实现的？》

文章目录

回到首页