



## 1、什么是 Redis?

**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 DB。

**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。另外 **Redis** 也可以对存入的 Key-Value 设置 expire 时间, 因此也可以被当作一个功能加强版的 memcached 来用。

**Redis** 的主要缺点是数据库容量受到物理内存的限制, 不能用作海量数据的高性能读写, 因此 **Redis** 适合的场景主要局限在较小数据量的高性能操作和运算上。

## 2、Redis 相比 memcached 有哪些优势?

- (1) memcached 所有的值均是简单的字符串, **Redis** 作为其替代者, 支持更为丰富的数据类型
- (2) **Redis** 的速度比 memcached 快很多
- (3) **Redis** 可以持久化其数据

## 3、Redis 支持哪几种数据类型?

String、List、Set、Sorted Set、hashes

## 4、Redis 主要消耗什么物理资源?

内存。

## 5、Redis 的全称是什么?

Remote Dictionary Server。

## 6、Redis 有哪几种数据淘汰策略?

noeviction: 返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令 (大部分的写入指令, 但 DEL 和几个例外)

allkeys-lru: 尝试回收最少使用的键 (LRU), 使得新添加的数据有空间存放。

volatile-lru: 尝试回收最少使用的键 (LRU), 但仅限于在过期集合的键, 使得新添加的数据有空间存放。

allkeys-random: 回收随机的键使得新添加的数据有空间存放。

volatile-random: 回收随机的键使得新添加的数据有空间存放, 但仅限于在过期集合的键。

volatile-ttl: 回收在过期集合的键, 并且优先回收存活时间 (TTL) 较短的键, 使得新添加的数据有空间存放。

## 7、Redis 官方为什么不提供 Windows 版本？

因为目前 Linux 版本已经相当稳定，而且用户量很大，无需开发 windows 版本，反而会带来兼容性问题。

## 8、一个字符串类型的值能存储最大容量是多少？

512M

## 9、为什么 Redis 需要把所有数据放到内存中？

**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。在内存越来越便宜的今天，**Redis** 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

## 10、Redis 集群方案应该怎么做？都有哪些方案？

1. twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通 **Redis** 无任何区别，设置好它下属的多个 **Redis** 实例后，使用时在本需要连接 **Redis** 的地方改为连接 twemproxy，它会以一个代理的身份接收请求并使用一致性 hash 算法，将请求转接到具体 **Redis**，将结果再返回 twemproxy。使用方式简便(相对 **Redis** 只需修改连接端口)，对旧项目扩展的首选。问题：twemproxy 自身单端口实例的压力，使用一致性 hash 后，对 **Redis** 节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2. codis，目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在节点数量改变情况下，旧节点数据可恢复到新 hash 节点。

3. **Redis** cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

4. 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。

## 11、Redis 集群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 失败了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

## 12、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？

Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

### 13、Redis 有哪些适合的场景？

#### (1)、会话缓存 (Session Cache)

最常用的一种使用 Redis 的情景是会话缓存 (session cache)。用 Redis 缓存会话比其他存储 (如 Memcached) 的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？

幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

#### (2)、全页缓存 (FPC)

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。

再次以 Magento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。

此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-Redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

#### (3)、队列

Redis 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言 (如 Python) 对 list 的 push/pop 操作。

如果你快速的在 Google 中搜索 “Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

#### (4)、排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合 (Set) 和有序集合 (Sorted Set) 也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的 10 个用户 - 我们称之为 “user\_scores”，我们只需要像下面一样执行即可：

当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你

需要这样执行：

```
ZRANGE user_scores 0 10 WITHSCORES
```

Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 **Redis** 来存储数据的，你可以在这里看到。

#### (5)、发布/订阅

最后（但肯定不是最不重要的）是 **Redis** 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 **Redis** 的发布/订阅功能来建立聊天系统！（不，这是真的，你可以去核实）。

## 14、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？

**Redisson**、**Jedis**、**lettuce** 等等，官方推荐使用 **Redisson**。

## 15、Redis 和 Redisson 有什么关系？

**Redisson** 是一个高级的分布式协调 **Redis** 客户端，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)。

## 16、Jedis 与 Redisson 对比有什么优缺点？

**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，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 **Redis** 特性。**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。

## 17、Redis 如何设置密码及验证密码？

设置密码：config set requirepass 123456

授权密码：auth 123456

## 18、说说 Redis 哈希槽的概念？

**Redis** 集群没有使用一致性 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。

## 19、Redis 集群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 个复制品。

## 20、Redis 集群会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，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。

## 21、Redis 集群之间是如何复制的？

异步复制

## 22、Redis 集群最大节点个数是多少？

16384 个。

## 23、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。

## 24、怎么测试 Redis 的连通性？

ping

## 25、Redis 中的管道有什么用？

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。

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。

## 26、怎么理解 Redis 事务？

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

## 27、Redis 事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

## 28、Redis key 的过期时间和永久有效分别怎么设置？

EXPIRE 和 PERSIST 命令。

## 29、Redis 如何做内存优化？

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key，而是应该把这个用户的所有信息存储到一张散列表里面。

## 30、Redis 回收进程如何工作的？

一个客户端运行了新的命令，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。

一个新的命令被执行，等等。

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。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

## 31、Redis 回收使用的是什么算法？

LRU 算法

## 32、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。

## 33、为什么要做 Redis 分区？

分区可以让 Redis 管理更大的内存，Redis 将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使 Redis 的计算能力通过简单地增加计算机得到成倍提升，Redis 的网络带宽也会随着计算机和网卡的增加而成倍增长。

## 34、你知道有哪些 Redis 分区实现方案？

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。

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。**Redis** 和 memcached 的一种代理实现就是 Twemproxy

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 redirected 到正确的 **Redis** 节点。

## 35、Redis 分区有什么缺点?

涉及多个 key 的操作通常不会被支持。例如你不能对两个集合求交集,因为他们可能被存储到不同的 **Redis** 实例(实际上这种情况也有办法,但是不能直接使用交集指令)。

同时操作多个 key,则不能使用 **Redis** 事务。

分区使用的粒度是 key,不能使用一个非常长的排序 key 存储一个数据集(The partitioning granularity is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set)。

当使用分区的时候,数据处理会非常复杂,例如为了备份你必须从不同的 **Redis** 实例和主机同时收集 RDB / AOF 文件。

分区时动态扩容或缩容可能非常复杂。**Redis** 集群在运行时增加或者删除 **Redis** 节点,能做到最大程度对用户透明地数据再平衡,但其他一些客户端分区或者代理分区方法则不支持这种特性。然而,有一种预分片的技术也可以较好的解决这个问题。

## 36、Redis 持久化数据和缓存怎么做扩容?

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。

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** 集群可以做到这样。

## 37、分布式 Redis 是前期做还是后期规模上来了再做好? 为什么?

既然 **Redis** 是如此的轻量(单实例只使用 1M 内存),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。

## 38、Twemproxy 是什么？

Twemproxy 是 Twitter 维护的(缓存)代理系统，代理 Memcached 的 ASCII 协议和 **Redis** 协议。它是单线程程序，使用 c 语言编写，运行起来非常快。它是采用 Apache 2.0 license 的开源软件。

Twemproxy 支持自动分区，如果其代理的其中一个 **Redis** 节点不可用时，会自动将该节点排除（这将改变原来的 keys-instances 的映射关系，所以你应该仅在把 **Redis** 当缓存时使用 Twemproxy）。

Twemproxy 本身不存在单点问题，因为你可以启动多个 Twemproxy 实例，然后让你的客户端去连接任意一个 Twemproxy 实例。

Twemproxy 是 **Redis** 客户端和服务端的一个中间层，由它来处理分区功能应该不算复杂，并且应该算比较可靠的。

## 39、支持一致性哈希的客户端有哪些？

**Redis-rb**、**PRedis** 等。

## 40、Redis 与其他 key-value 存储有什么不同？

**Redis** 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。**Redis** 的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

**Redis** 运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样 **Redis** 可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

## 41、Redis 的内存占用情况怎么样？

给你举个例子：100 万个键值对（键是 0 到 999999 值是字符串 “hello world”）在我的 32 位的 Mac 笔记本上用了 100MB。同样的数据放到一个 key 里只需要 16MB，这是因为键值有一个很大的开销。在 Memcached 上执行也是类似的结果，但是相对 **Redis** 的开销要小一点点，因为 **Redis** 会记录类型信息引用计数等等。

当然，大键值对时两者的比例要好很多。



64 位的系统比 32 位的需要更多的内存开销，尤其是键值对都较小时，这是因为 64 位的系统里指针占用了 8 个字节。但是，当然，64 位系统支持更大的内存，所以为了运行大型的 Redis 服务器或多或少的需要使用 64 位的系统。

## 42、都有哪些办法可以降低 Redis 的内存使用情况呢？

如果你使用的是 32 位的 Redis 实例，可以好好利用 Hash,list,sorted set,set 等集合类型数据，因为通常情况下很多小的 Key-Value 可以用更紧凑的方式存放到一起。

## 43、查看 Redis 使用情况及状态信息用什么命令？

info

## 44、Redis 的内存用完了会发生什么？

如果达到设置的上限，Redis 的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将 Redis 当缓存来使用配置淘汰机制，当 Redis 达到内存上限时会冲刷掉旧的内容。

## 45、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，你可以考虑一下分片（shard）。

## 46、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 232 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。我们正在测试一些较大的值。

任何 list、set、和 sorted set 都可以放 232 个元素。

换句话说，Redis 的存储极限是系统中的可用内存值。

## 47、Redis 常见性能问题和解决方案？

(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

(2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 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。

## 48、Redis 提供了哪几种持久化方式?

RDB 持久化方式能够在指定的时间间隔能对你的数据进行快照存储。

AOF 持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF 命令以 **Redis** 协议追加保存每次写的操作到文件末尾。**Redis** 还能对 AOF 文件进行后台重写,使得 AOF 文件的体积不至于过大。

如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式。

你也可以同时开启两种持久化方式,在这种情况下,当 **Redis** 重启的时候会优先载入 AOF 文件来恢复原始的数据,因为在通常情况下 AOF 文件保存的数据集要比 RDB 文件保存的数据集要完整。

最重要的事情是了解 RDB 和 AOF 持久化方式的不同,让我们以 RDB 持久化方式开始。

## 49、如何选择合适的持久化方式?

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。

## 50、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。