Redis 高频面试题

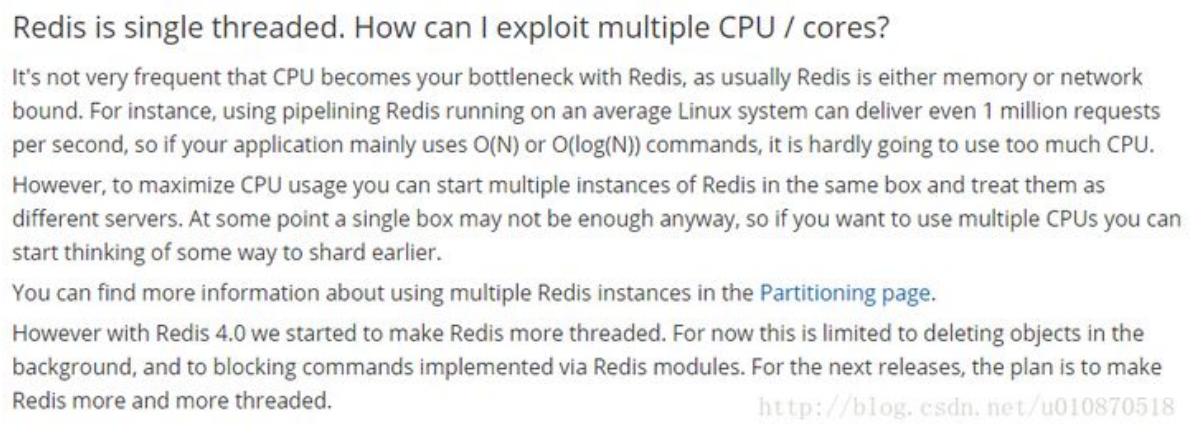
Redis是什么?

> Redis是一种运行速度很快，并发很强的跑在内存上的NoSql数据库，支持键到五种数据类型的映射。

来来来，讲一讲为什么Redis这么快?

> **首先，采用了多路复用io阻塞机制**
> **然后，数据结构简单，操作节省时间**
> **最后，运行在内存中，自然速度快**

Redis为什么是单线程的?



Redis is single threaded. How can I exploit multiple CPU / cores?

It's not very frequent that CPU becomes your bottleneck with Redis, as usually Redis is either memory or network bound. For instance, using pipelining Redis running on an average Linux system can deliver even 1 million requests per second, so if your application mainly uses O(N) or O(log(N)) commands, it is hardly going to use too much CPU.

However, to maximize CPU usage you can start multiple instances of Redis in the same box and treat them as different servers. At some point a single box may not be enough anyway, so if you want to use multiple CPUs you can start thinking of some way to shard earlier.

You can find more information about using multiple Redis instances in the Partitioning page.

However with Redis 4.0 we started to make Redis more threaded. For now this is limited to deleting objects in the background, and to blocking commands implemented via Redis modules. For the next releases, the plan is to make Redis more and more threaded.

> Redis官方很敷衍就随便给了一点解释。不过基本要点也都说了，因为Redis的瓶颈不是cpu的运行速度，而往往是网络带宽和机器的内存大小。再说了，单线程切换开销小，容易实现既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了。

如果万一CPU成为你的Redis瓶颈了，或者，你就是不想让服务器其他核闲置，那怎么办?

> 那也很简单，你**多起几个Redis进程就好了。**Redis是keyvalue数据库，又不是关系数据库，数据之间没有约束。只要客户端分清哪些key放在哪个Redis进程上就可以了。**redis-cluster**可以帮你做的更好。

单线程可以处理高并发请求吗?

> 当然可以了，Redis都实现了。有一点概念需要澄清，并发并不是并行。
> （相关概念：并发性I/O流，意味着能够让一个计算单元来处理来自多个客户端的流请求。并行性，意味着服务器能够同时执行几个事情，具有多个计算单元）

我们使用单线程的方式是无法发挥多核CPU 性能，有什么办法发挥多核CPU的性能嘛?

> 我们可以通过在单机开多个Redis 实例来完善!
> 警告：这里**我们一直在强调的单线程，只是在处理我们的网络请求的时候只有一个线程来处理**，一个正式的Redis Server运行的时候肯定是不止一个线程的，这里需要大家明确的注意一下!
> 例如Redis进行持久化的时候会以子进程或者子线程的方式执行（具体是子线程还是子进程待读者深入研究）

简述一下Redis值的五种类型

> String 整数，浮点数或者字符串
> Set 集合
> Zset 有序集合
> Hash 散列表
> List 列表

Redis数据类型应用场景

类型	简介	特性	场景
string（字符串）	二进制安全	可以包含任何数据，比如jpg图片或者序列化的对象，一个键最大能存储512M	验证码、token等场景
Hash（字典）	键值对集合，即编程语言中的Map类型	适合存储对象，并且可以像数据库中update一个属性一样只修改某一项的属性值	存储购物车、详情页等场景
List（列表）	链表（双向链表）	增删快，提供了操作某一段元素的API	最新消息排行(评论列表)、消息队列等
Set（集合）	哈希表实现，元素不重复	添加，删除，查找的复杂度都是O(1)，为集合提供了交集，并集，差集等操作	抽奖随机事件、共同好友、统计网站的所有ip、数据处理等场景
Sorted Set（有序集合）	将Set中的元素增加一个权重参数score，元素按score有序排列	数据插入数据集合时，已经进行天然排序了	排行榜等场景
Bitmaps	二进制安全，存储bit位信息	二进制格式数据，典型的二值状态存储	访问统计、签到统计等场景
HypefLogLog	有点类似集合类型，实际是一种字符串类型的数据结构	最大的好处就是减少空间、但是也存在一定的误差率	不需要精确计算的统计等场景
GEO类型	地理信息存储	地理位置信息存储	一些距离计算、附近推荐等场景
Stream类型	Redis 5.0 版本新增加的数据结构	消息的持久化和主备复制功能	消息队列场景

有序集合的实现方式是哪种数据结构?

跳跃表。

请列举几个用得到Redis的常用使用场景?

缓存，毫无疑问这是Redis当今最为人熟知的使用场景。再提升服务器性能方面非常有效；

排行榜，在使用传统的关系型数据库（mysql oracle 等）来做这个事儿，非常的麻烦，而利用Redis的SortSet(有序集合)数据结构能够简单的搞定；

计算器/限速器，利用Redis中原子性的自增操作，我们可以统计类似用户点赞数、用户访问数等，这类操作如果用MySQL，频繁的读写会带来相当大的压力；限速器比较典型的使用场景是限制某个用户访问某个API的频率，常用的有抢购时，防止用户疯狂点击带来不必要的压力；

好友关系，利用集合的一些命令，比如求交集、并集、差集等。可以方便搞定一些共同好友、共同爱好之类的功能；

简单消息队列，除了Redis自身的发布/订阅模式，我们也可以利用List来实现一个队列机制，比如：到货通知、邮件发送之类的需求，不需要高可靠，但是会带来非常大的DB压力，完全可以用List来完成异步解耦；

Session共享，以PHP为例，默认Session是保存在服务器的文件中，如果是集群服务，同一个用户过来可能落在不同机器上，这就会导致用户频繁登陆；采用Redis保存Session后，无论用户落在那台机器上都能够获取到对应的Session信息。

一些频繁被访问的数据，经常被访问的数据如果放在关系型数据库，每次查询的开销都会很大，而放在redis中，因为redis 是放在内存中的可以很高效的访问

简述Redis的数据淘汰机制

> 当maxmemory限制达到的时候Redis会使用的行为由 Redis的maxmemory-policy配置指令来进行配置。
> 以下的策略是可用的:
> 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）
>
> 当内存使用超过配置的时候会返回错误，不会驱逐任何键
>
> allkeys-lru: 尝试回收最近最久未使用的键（LRU），针对所有key，使得新添加的数据有空间存放。
> volatile-lru: 尝试回收最近最久未使用的键（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-lfu: 尝试回收最近最少使用的键（LFU），针对所有key，使得新添加的数据有空间存放。
> volatile-lfu: 尝试回收最近最少使用的键（LFU），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: 针对所有key，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。
>
> 如果没有键满足回收的前提条件的话，策略volatile-lru，volatile-lfu, volatile-random以及volatile-ttl就和noeviction 差不多了。
> 选择正确的回收策略是非常重要的，这取决于你的应用的访问模式，不过你可以在运行时进行相关的策略调整，并且监控缓存命中率和没命中的次数，通过RedisINFO命令输出以便调优。
> 一般的经验规则:
> 使用allkeys-lru策略：当你希望你的请求符合一个幂定律分布，也就是说，你希望部分的子集元素将比其它其它元素被访问的更多。如果你不确定选择什么，这是个很好的选择。.
>
> 使用allkeys-random：如果你是循环访问，所有的键被连续的扫描，或者你希望请求分布正常（所有元素被访问的概率都差不多）。
>
> 使用volatile-ttl：如果你想要通过创建缓存对象时设置TTL值，来决定哪些对象应该被过期。

> 作为缓存来说noeviction作为缓存来说肯定是不用的，random过于随机，因此一般在LRU和LFU中进行选择。

Redis怎样防止异常数据不丢失?

> **RDB 持久化**
> 将某个时间点的所有数据都存放到硬盘上。
> 可以将快照复制到其它服务器从而创建具有相同数据的服务器副本。
> 如果系统发生故障，将会丢失最后一次创建快照之后的数据。
> 如果数据量很大，保存快照的时间会很长。
> **AOF 持久化**
> 将写命令添加到 AOF 文件（Append Only File）的末尾。
> 使用 AOF 持久化需要设置同步选项，从而确保写命令同步到磁盘文件上的时机。这是因为对文件进行写入并不会马上将内容同步到磁盘上，而是先存储到缓冲区，然后由操作系统决定什么时候同步到磁盘。有以下同步选项：
> 选项同步频率always每个写命令都同步everysec每秒同步一次no让操作系统来决定何时同步
> **always** 选项会严重减低服务器的性能；
> **everysec** 选项比较合适，可以保证系统崩溃时只会丢失一秒左右的数据，并且 Redis 每秒执行一次同步对服务器性能几乎没有任何影响；
> **no** 选项并不能给服务器性能带来多大的提升，而且也会增加系统崩溃时数据丢失的数量
> 随着服务器写请求的增多，AOF 文件会越来越大。Redis 提供了一种将 AOF 重写的特性，能够去除 AOF 文件中的冗余写命令。

讲一讲缓存穿透，缓存雪崩以及缓存击穿吧

> **缓存穿透：就是客户持续向服务器发起对不存在服务器中数据的请求。客户先在Redis中查询，查询不到后去数据库中查询。**
> **缓存击穿：就是一个很热门的数据，突然失效，大量请求到服务器数据库中**
> **缓存雪崩：就是大量数据同一时间失效。**
> **打个比方，你是个很有钱的人，开满了百度云，腾讯视频各种杂七杂八的会员，但是你就是没有netflix的会员，然后你把这些账号和密码发布到一个你自己做的网站上，然后你有一个朋友每过十秒钟就查询你的网站，发现你的网站没有Netflix的会员后打电话向你要。你就相当于是个数据库，网站就是Redis。这就是缓存穿透。**
> **大家都喜欢看腾讯视频上的《水果传》，但是你的会员突然到期了，大家在你的网站上看不到腾讯视频的账号，纷纷打电话向你询问，这就是缓存击穿**
> **你的各种会员突然同一时间都失效了，那这就是缓存雪崩了。**
>
> **放心，肯定有办法解决的。**
> **缓存穿透：**
> **1.接口层增加校验，对传参进行个校验，比如说我们的id是从1开始的，那么id<=0的直接拦截；**
> **2.缓存中取不到的数据，在数据库中也没有取到，这时可以将key-value对写为key-null，这样可以防止攻击用户反复用同一个id暴力攻击**
> **缓存击穿：**
> **最好的办法就是设置热点数据永不过期，拿到刚才的比方里，那就是你买腾讯一个永久会员**
> **缓存雪崩：**
> **1.缓存数据的过期时间设置随机，防止同一时间大量数据过期现象发生。**
> **2.如果缓存数据库是分布式部署，将热点数据均匀分布在不同搞得缓存数据库中。**

说一下Redis中的Master-Slave模式

连接过程

1. 主服务器创建快照文件，发送给从服务器，并在发送期间使用缓冲区记录执行的写命令。快照文件发送完毕之后，开始向从服务器发送存储在缓冲区中的写命令；
2. 从服务器丢弃所有旧数据，载入主服务器发来的快照文件，之后从服务器开始接受主服务器发来的写命令；
3. 主服务器每执行一次写命令，就向从服务器发送相同的写命令。

主从链

随着负载不断上升，主服务器可能无法很快地更新所有从服务器，或者重新连接和重新同步从服务器将导致系统超载。为了解决这个问题，可以创建一个中间层来分担主服务器的复制工作。中间层的服务器是最上层服务器的从服务器，又是最下层服务器的主服务器。

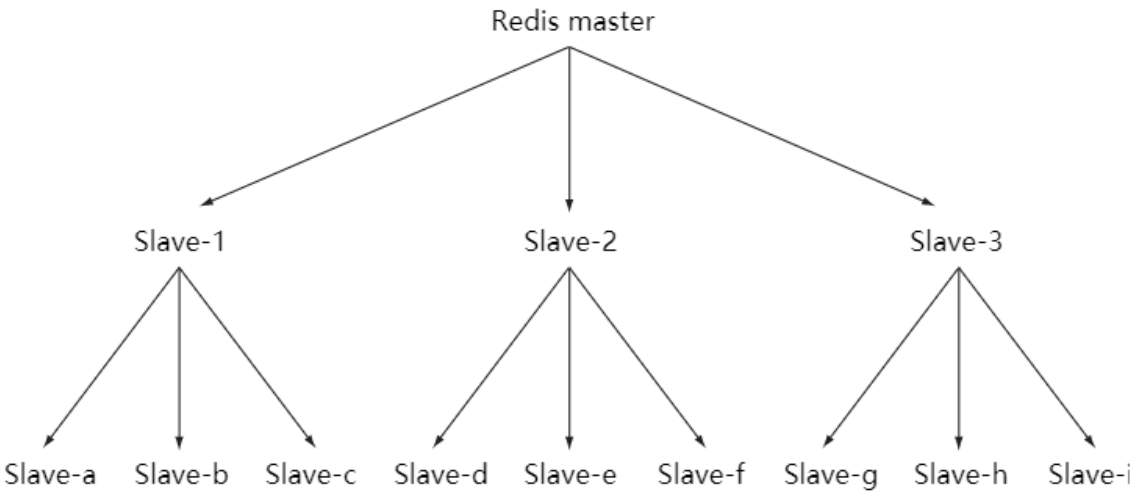


Figure 4.1 An example Redis master/slave replica tree with nine lowest-level slaves and three intermediate replication helper servers

Sentinel（哨兵）可以监听集群中的服务器，并在主服务器进入下线状态时，自动从从服务器中选举出新的主服务器。

分片

> 分片是将数据划分为多个部分的方法，可以将数据存储到多台机器里面，这种方法在解决某些问题时可以获得线性级别的性能提升。
> 假设有 4 个 Redis 实例 R0，R1，R2，R3，还有很多表示用户的键 user:1，user:2，... ，有不同的方式来选择一个指定的键存储在哪个实例中。
>
> 最简单的方式是范围分片，例如用户 id 从 0~1000 的存储到实例 R0 中，用户 id 从 1001~2000 的存储到实例 R1 中，等等。但是这样需要维护一张映射范围表，维护操作代价很高。
> 还有一种方式是哈希分片，使用 CRC32 哈希函数将键转换为一个数字，再对实例数量求模就能知道应该存储的实例。
>
> 根据执行分片的位置，可以分为三种分片方式：
> 客户端分片：客户端使用一致性哈希等算法决定键应当分布到哪个节点。
> 代理分片：将客户端请求发送到代理上，由代理转发请求到正确的节点上。
> 服务器分片：Redis Cluster