redis

leolinuxer

August 13, 2020

Contents

1 为什么说 Redis 是单线程的以及 Redis 为什么这么快！[1]

1.1 Redis 简介

Redis 是一个开源的内存中的数据结构存储系统，它可以用作：**数据库、缓存和消息中间件**。

它支持多种类型的数据结构，如字符串（String），散列（Hash），列表（List），集合（Set），有序集合（Sorted Set 或者是 ZSet）与范围查询，Bitmaps，Hyperloglogs 和地理空间（Geospatial）索引半径查询。**其中常见的数据结构类型有：String、List、Set、Hash、ZSet 这 5 种**。

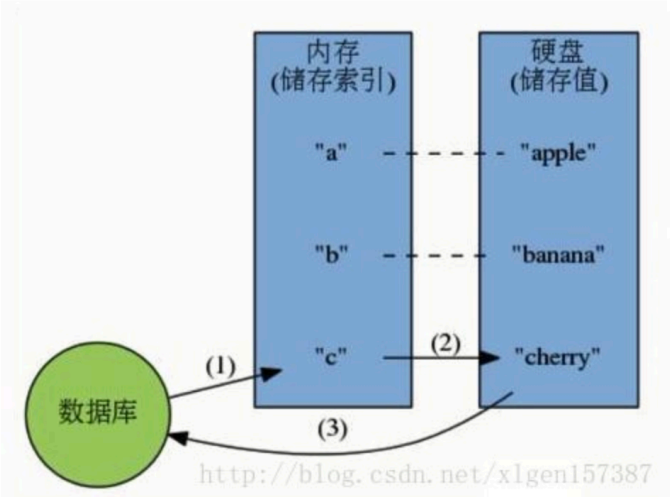
Redis 内置了复制（Replication），LUA 脚本（Lua scripting），LRU 驱动事件（LRU eviction），事务（Transactions）和不同级别的磁盘持久化（Persistence），并通过 Redis 哨兵（Sentinel）和自动分区（Cluster）提供高可用性（High Availability）。

Redis 也提供了持久化的选项，这些选项可以让用户将自己的数据保存到磁盘上面进行存储。根据实际情况，可以每隔一定时间将数据集导出到磁盘（快照），或者追加到命令日志中（AOF 只追加文件），他会在执行写命令时，将被执行的写命令复制到硬盘里面。您也可以关闭持久化功能，将 Redis 作为一个高效的网络的缓存数据功能使用。

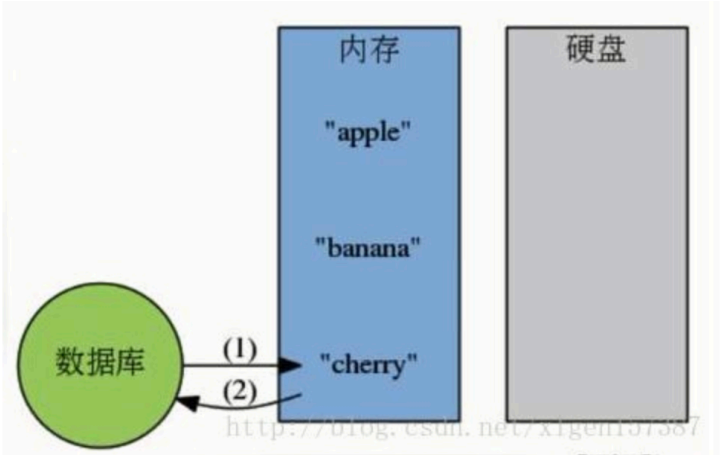
Redis 不使用表，他的数据库不会预定义或者强制去要求用户对 Redis 存储的不同数据进行关联。

数据库的工作模式按存储方式可分为：硬盘数据库和内存数据库。Redis 将数据储存在内存里面，读写数据的时候都不会受到硬盘 I/O 速度的限制，所以速度极快。

（1）硬盘数据库的工作模式：



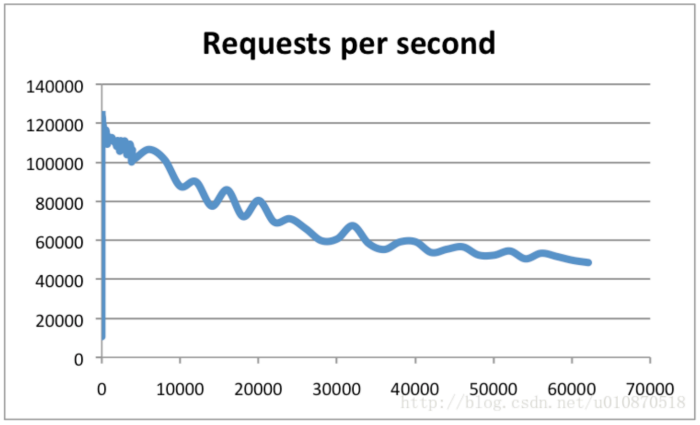
（2）内存数据库的工作模式：



看完上述的描述，对于一些常见的 Redis 相关的面试题，是否有所认识了，例如：什么是 Redis、Redis 常见的数据结构类型有哪些、Redis 是如何进行持久化的等。

1.2 Redis 到底有多快

Redis 采用的是基于内存的采用的是单进程单线程模型的 KV 数据库，由 C 语言编写，官方提供的数据是可以达到 100000+ 的 QPS（每秒内查询次数）。这个数据不比采用单进程多线程的同样基于内存的 KV 数据库 Memcached 差！有兴趣的可以参考官方的基准程序测试《How fast is Redis?》（https://redis.io/topics/benchmarks）



横轴是连接数，纵轴是 QPS。此时，这张图反映了一个数量级，希望大家在面试的时候可以正确的描述出来，不要问你的时候，你回答的数量级相差甚远！

1.3 Redis 为什么这么快

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。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)；

2、数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的；

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4、使用多路 I/O 复用模型，非阻塞 IO；

5、使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis 直接自己构建了 VM 机制，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求；

以上几点都比较好理解，下边我们针对多路 I/O 复用模型进行简单的探讨：

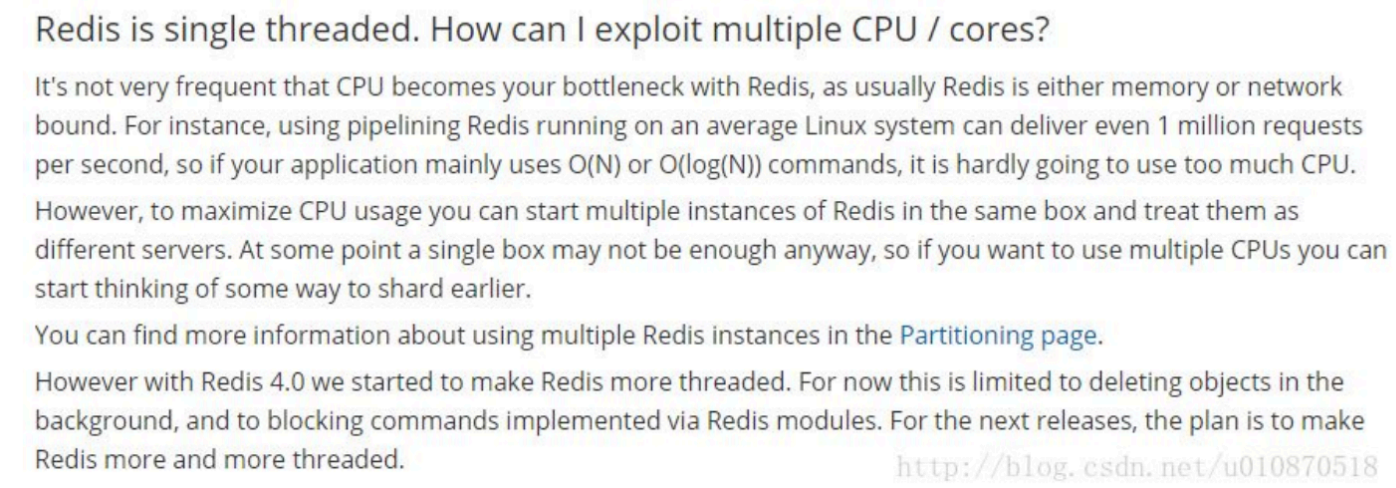
（1）多路 I/O 复用模型

多路 I/O 复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

这里“多路”指的是多个网络连接，“复用”指的是复用同一个线程。采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗），且 Redis 在内存中操作数据的速度非常快，也就是说内存内的操作不会成为影响 Redis 性能的瓶颈，主要由以上几点造就了 Redis 具有很高的吞吐量。

1.4 那么为什么 Redis 是单线程的

我们首先要明白，上边的种种分析，都是为了营造一个 Redis 很快的氛围！官方 FAQ 表示，因为 Redis 是基于内存的操作，CPU 不是 Redis 的瓶颈，Redis 的瓶颈最有可能是机器内存的大小或者网络带宽。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（毕竟采用多线程会有很多麻烦！）。



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.

可以参考：https://redis.io/topics/faq

但是，我们使用单线程的方式是无法发挥多核 CPU 性能，不过我们可以通过在单机开多个 Redis 实例来完善！

警告 1：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）；例如我在测试服务器上查看 Redis 进程，然后找到该进程下的线程：

```
xuliugen@xuliugen:~/server/redis-stable/src$ ps -ef | grep redis-server
xuliugen  8545  8356  0 04:20 pts/2    00:00:00 grep --color=auto redis-server
root     12400 12374  0 Jan04 ?        00:23:21 ./src/redis-server 0.0.0.0:6379
xuliugen@xuliugen:~/server/redis-stable/src$ ps -T -p 12400
  PID  SPID TTY          TIME CMD
12400 12400 ?        00:23:21 redis-server
12400 12401 ?        00:00:00 redis-server
12400 12402 ?        00:00:00 redis-server
xuliugen@xuliugen:~/server/redis-stable/src$
```


ps 命令的“-T”参数表示显示线程（Show threads, possibly with SPID column.）“SID”栏表示线程 ID，而“CMD”栏则显示了线程名称。

警告 2：在上图中 FAQ 中的最后一段，表述了从 Redis 4.0 版本开始会支持多线程的方式，但是，只是在某一些操作上进行多线程的操作！所以该篇文章在以后的版本中是否还是单线程的方式需要读者考

证!

1.5 注意点

1、我们知道 Redis 是用”单线程-多路复用 IO 模型”来实现高性能的内存数据服务的，这种机制避免了使用锁，但是同时这种机制在进行 sunion 之类的比较耗时的命令时会使 redis 的并发下降。因为是单一线程，所以同一时刻只有一个操作在进行，所以，耗时的命令会导致并发的下降，不只是读并发，写并发也会下降。而单一线程也只能用到一个 CPU 核心，所以可以在同一个多核的服务器中，可以启动多个实例，组成 master-master 或者 master-slave 的形式，耗时的读命令可以完全在 slave 进行。

需要改的 redis.conf 项：

```
pidfile /var/run/redis/redis_6377.pid  #pidfile要加上端口号
port 6377  #这个是必须改的
logfile /var/log/redis/redis_6377.log #logfile的名称也加上端口号
dbfilename dump_6377.rdb  #rdbfile也加上端口号
```

2、“我们不能任由操作系统负载均衡，因为我们自己更了解自己的程序，所以，我们可以手动地为其分配 CPU 核，而不会过多地占用 CPU，或是让我们关键进程和一堆别的进程挤在一起。”。CPU 是一个重要的影响因素，由于是单线程模型，Redis 更喜欢大缓存快速 CPU，而不是多核

在多核 CPU 服务器上面，Redis 的性能还依赖 NUMA 配置和处理器绑定位置。最明显的影响是 redis-benchmark 会随机使用 CPU 内核。为了获得精准的结果，需要使用固定处理器工具（在 Linux 上可以使用 taskset）。最有效的办法是将客户端和服务端分离到两个不同的 CPU 来高校使用三级缓存。

1.6 扩展

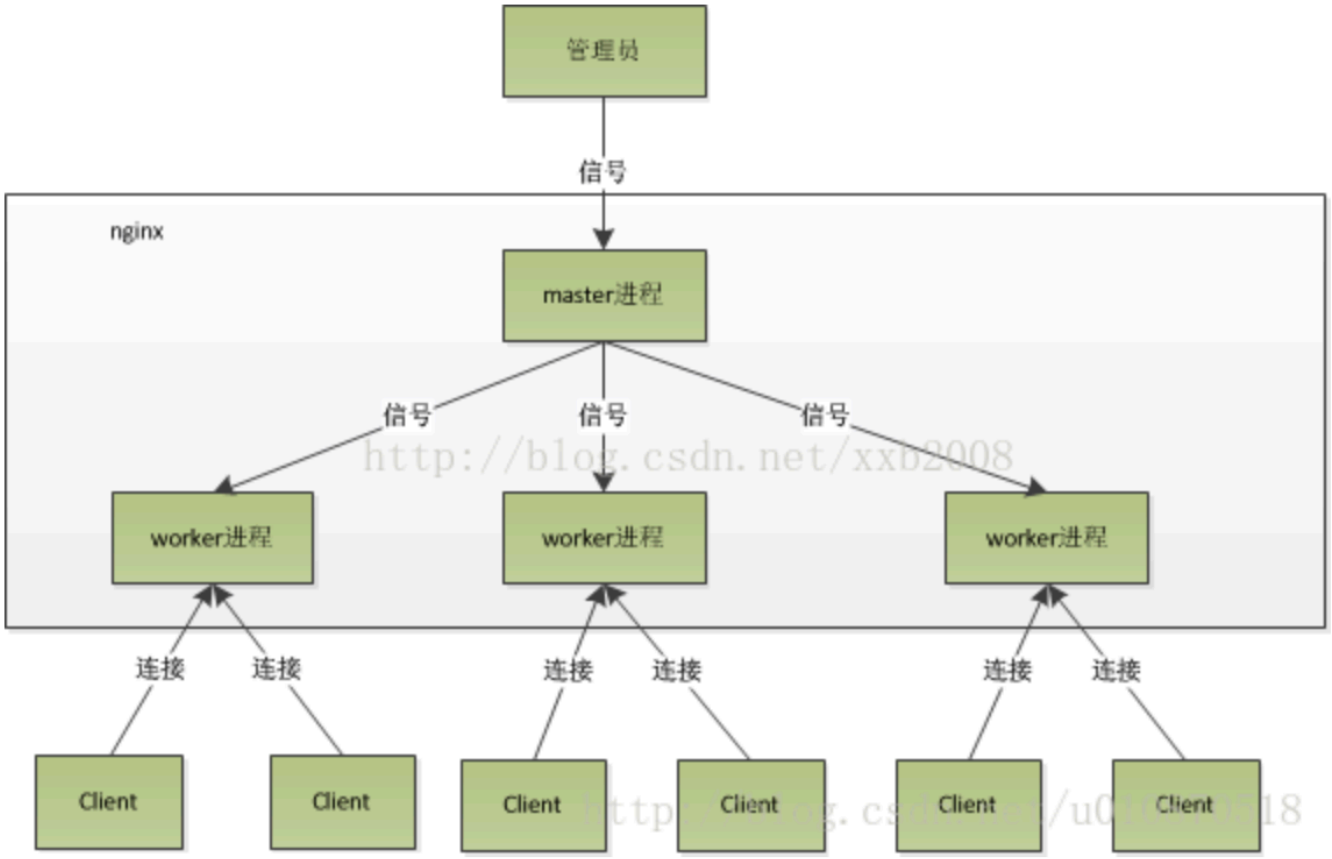
以下也是你应该知道的几种模型，祝你的面试一臂之力!

1、单进程多线程模型：MySQL、Memcached、Oracle（Windows 版本）；

2、多进程模型：Oracle（Linux 版本）；

3、Nginx 有两类进程，一类称为 Master 进程 (相当于管理进程)，另一类称为 Worker 进程（实际工作进程）。启动方式有两种：

- 单进程启动：此时系统中仅有一个进程，该进程既充当 Master 进程的角色，也充当 Worker 进程的角色。
- 多进程启动：此时系统有且仅有一个 Master 进程，至少有一个 Worker 进程工作。
- Master 进程主要进行一些全局性的初始化工作和管理 Worker 的工作；事件处理是在 Worker 中进行的。



References

[1] “为什么说 redis 是单线程的以及 redis 为什么这么快！.” [Online]. Available: https://zhuanlan.zhihu.com/p/34438275