> 支持多线程的 Redis 6.0 版本于 2020-05-02 终于发布了，为什么 Redis 忽然要支持多线程？如何开启多线程？开启后性能提升效果如何？线程数量该如何设置？开启多线程后会不会有线程安全问题？多线程的实现原理是怎样的？带着这些疑问，我们来开启 Redis 新特性-多线程连环 13 问。

Redis 6.0 来了

在全国一片祥和 IT 民工欢度五一节假日的时候，Redis 6.0 不声不响地于 5 月 2 日正式发布了，吓得我赶紧从床上爬起来，学无止境！学无止境！

对于 6.0 版本，Redis 之父 Antirez 在 RC1 版本发布时（2019-12-19）在他的博客上连续用了几个“EST”词语来评价：

> *the most “enterprise” Redis version to date // 最“企业级”的*
>
> *the largest release of Redis ever as far as I can tell // 最大的*
>
> *the one where the biggest amount of people participated // 参与人数最多的*

这个版本提供了诸多令人心动的新特性及功能改进，比如新网络协议 RESP3，新的集群代理，ACL 等，其中关注度最高的应该是“多线程”了。

笔者也第一时间体验了一下，带着众多疑问，我们来一起开始“Redis 6.0 新特性-多线程连环 13 问”。

Redis 6.0 多线程连环13问

①Redis 6.0 之前的版本真的是单线程吗?

Redis 在处理客户端的请求时，包括获取（Socket 读）、解析、执行、内容返回（Socket 写）等都由一个顺序串行的主线程处理，这就是所谓的“单线程”。

但如果严格来讲从 Redis 4.0 之后并不是单线程，除了主线程外，它也有后台线程在处理一些较为缓慢的操作，例如清理脏数据、无用连接的释放、大 Key 的删除等等。

②Redis 6.0 之前为什么一直不使用多线程?

官方曾做过类似问题的回复：使用 Redis 时，几乎不存在 CPU 成为瓶颈的情况， Redis 主要受限于内存和网络。

例如在一个普通的 Linux 系统上，Redis 通过使用 Pipelining 每秒可以处理 100 万个请求，所以如果应用程序主要使用 O(N) 或 O(log(N)) 的命令，它几乎不会占用太多 CPU。

使用了单线程后，可维护性高。多线程模型虽然在某些方面表现优异，但是它却引入了程序执行顺序的不确定性，带来了并发读写的一系列问题，增加了系统复杂度、同时可能存在线程切换、甚至加锁解锁、死锁造成的性能损耗。

Redis 通过 AE 事件模型以及 IO 多路复用等技术，处理性能非常高，因此没有必要使用多线程。

单线程机制使得 Redis 内部实现的复杂度大大降低，Hash 的惰性 Rehash、Lpush 等等 “线程不安全” 的命令都可以无锁进行。

③Redis 6.0 为什么要引入多线程呢?

Redis 将所有数据放在内存中，内存的响应时长大约为 100 纳秒，对于小数据包，Redis 服务器可以处理 80,000 到 100,000 QPS，这也是 Redis 处理的极限了，对于 80% 的公司来说，单线程的 Redis 已经足够使用了。

但随着越来越复杂的业务场景，有些公司动不动就上亿的交易量，因此需要更大的 QPS。

常见的解决方案是在分布式架构中对数据进行分区并采用多个服务器，但该方案有非常大的缺点，例如要管理的 Redis 服务器太多，维护代价大。

某些适用于单个 Redis 服务器的命令不适用于数据分区；数据分区无法解决热点读/写问题；数据偏斜，重新分配和放大/缩小变得更加复杂等等。

从 Redis 自身角度来说，因为读写网络的 Read/Write 系统调用占用了 Redis 执行期间大部分 CPU 时间，瓶颈主要在于网络的 IO 消耗。

优化主要有两个方向:

- 提高网络 IO 性能，典型的实现比如使用 DPDK 来替代内核网络栈的方式。
- 使用多线程充分利用多核，典型的实现比如 Memcached。

协议栈优化的这种方式跟 Redis 关系不大，支持多线程是一种最有效最便捷的操作方式。

所以总结起来，Redis 支持多线程主要就是两个原因:

- 可以充分利用服务器 CPU 资源，目前主线程只能利用一个核。
- 多线程任务可以分摊 Redis 同步 IO 读写负荷。

④Redis 6.0 默认是否开启了多线程?

Redis 6.0 的多线程默认是禁用的，只使用主线程。如需开启需要修改 redis.conf 配置文件：io-threads-do-reads yes。

⑤Redis 6.0 多线程开启时，线程数如何设置?

开启多线程后，还需要设置线程数，否则是不生效的。同样修改 redis.conf 配置文件:

关于线程数的设置，官方有一个建议：4 核的机器建议设置为 2 或 3 个线程，8 核的建议设置为 6 个线程，线程数一定要小于机器核数。

还需要注意的是，线程数并不是越大越好，官方认为超过了 8 个基本就没什么意义了。

⑥Redis 6.0 采用多线程后，性能的提升效果如何?

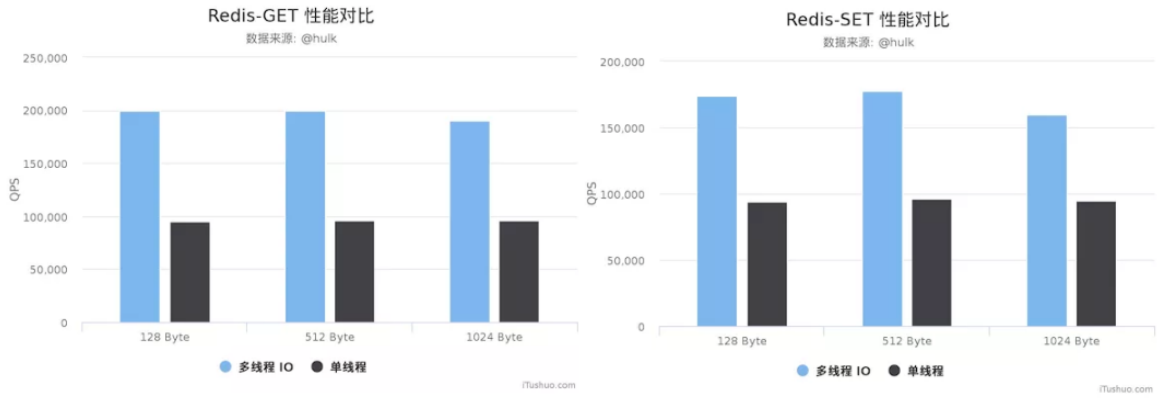
Redis 作者 antirez 在 RedisConf 2019 分享时曾提到：Redis 6 引入的多线程 IO 特性对性能提升至少是一倍以上。

国内也有大牛曾使用 unstable 版本在阿里云 esc 进行过测试，GET/SET 命令在 4 线程 IO 时性能相比单线程是几乎是翻倍了。

测试环境：

> *Redis Server：阿里云 Ubuntu 18.04，8 CPU 2.5 GHZ, 8G 内存，主机型号 ecs.ic5.2xlarge*
>
> *Redis Benchmark Client：阿里云 Ubuntu 18.04，8 2.5 GHZ CPU, 8G 内存，主机型号 ecs.ic5.2xlarge*

测试结果：



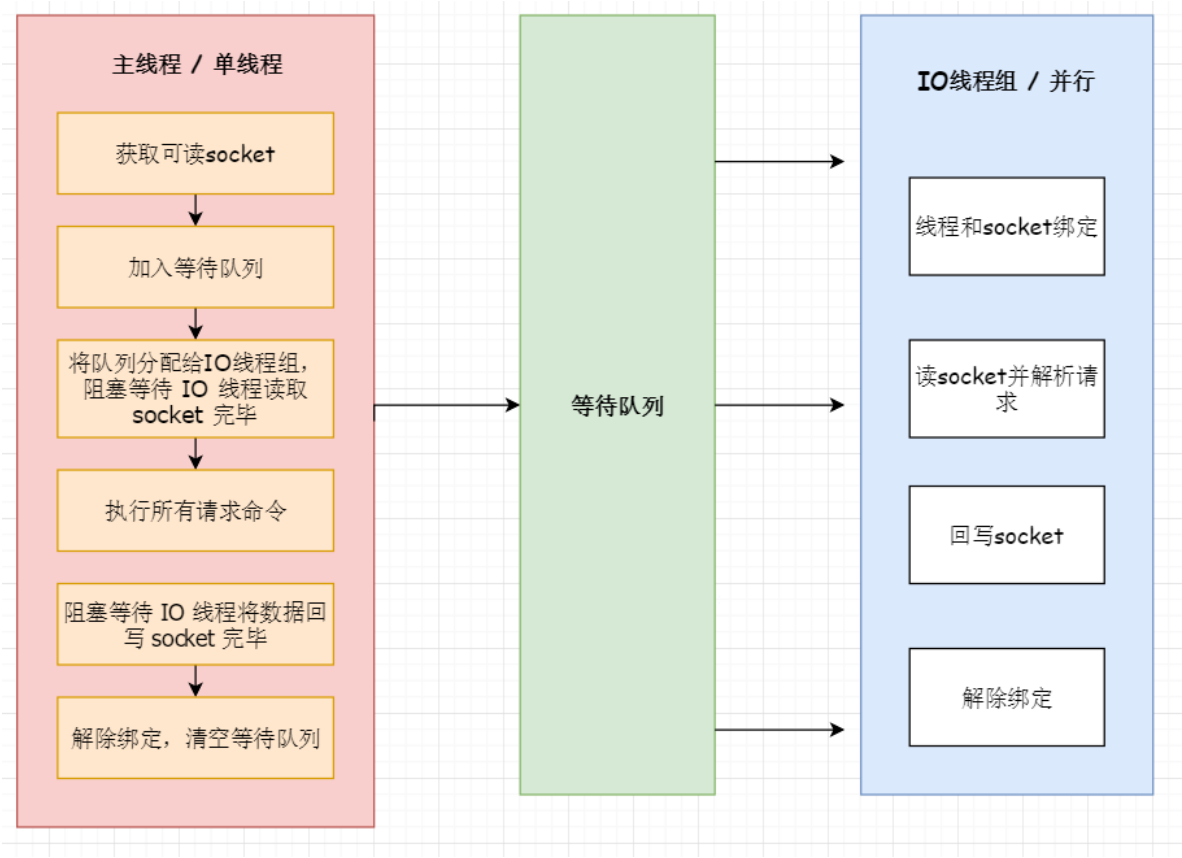
详见：

```
https://zhuanlan.zhihu.com/p/76788470
```

说明 1：这些性能验证的测试并没有针对严谨的延时控制和不同并发的场景进行压测。数据仅供验证参考而不能作为线上指标。

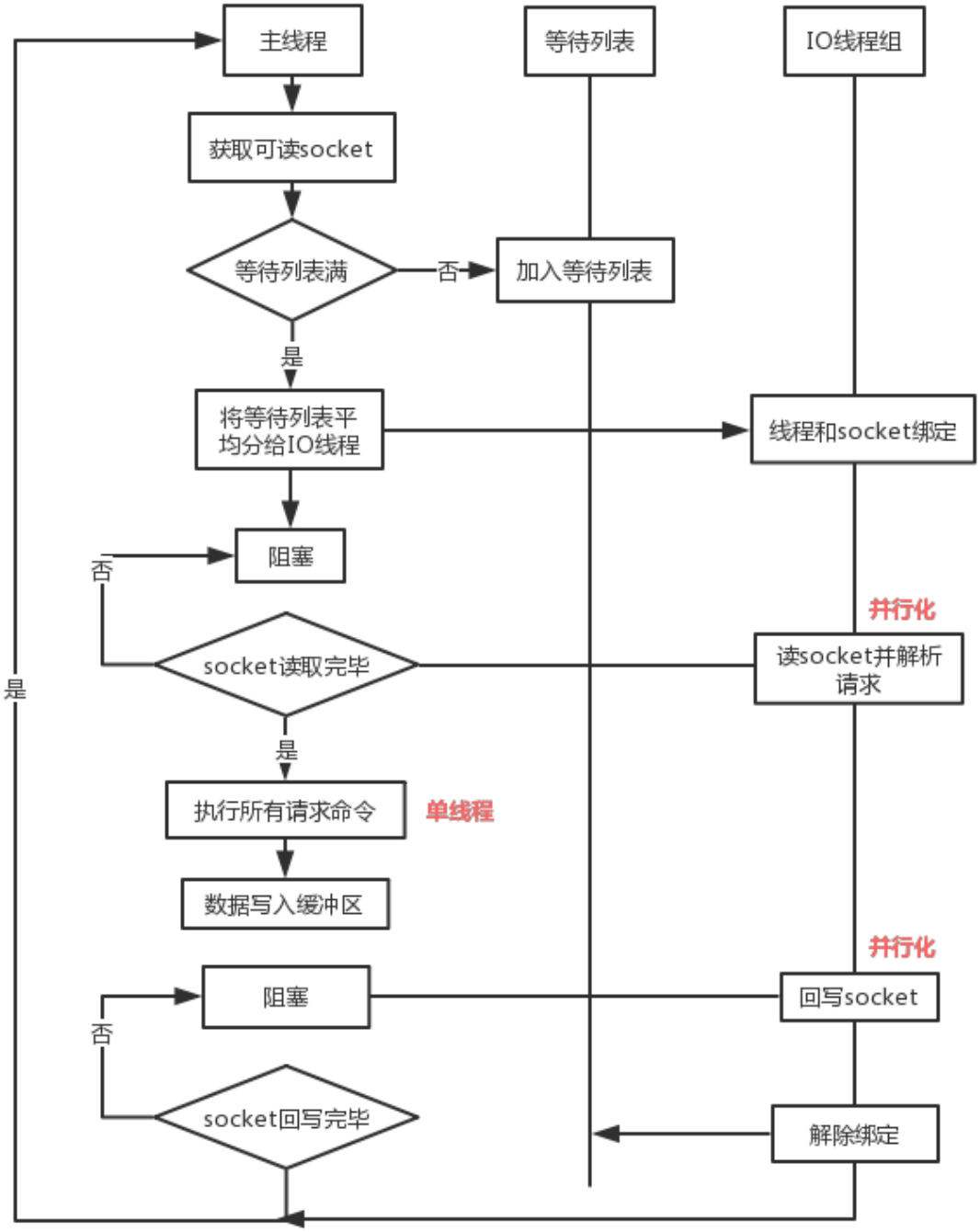
说明 2：如果开启多线程，至少要 4 核的机器，且 Redis 实例已经占用相当大的 CPU 耗时的时候才建议采用，否则使用多线程没有意义。所以估计 80% 的公司开发人员看看就好。

⑦Redis 6.0 多线程的实现机制?



流程简述如下：

- 主线程负责接收建立连接请求，获取 Socket 放入全局等待读处理队列。
- 主线程处理完读事件之后，通过 RR（Round Robin）将这些连接分配给这些 IO 线程。
- 主线程阻塞等待 IO 线程读取 Socket 完毕。
- 主线程通过单线程的方式执行请求命令，请求数据读取并解析完成，但并不执行。
- 主线程阻塞等待 IO 线程将数据回写 Socket 完毕。
- 解除绑定，清空等待队列。



图片来源：https://ruby-china.org/topics/38957

该设计有如下特点：

- IO 线程要么同时在读 Socket，要么同时在写，不会同时读或写。
- IO 线程只负责读写 Socket 解析命令，不负责命令处理。

⑧开启多线程后，是否会存在线程并发安全问题？

从上面的实现机制可以看出，Redis 的多线程部分只是用来处理网络数据的读写和协议解析，执行命令仍然是单线程顺序执行。

所以我们不需要去考虑控制 Key、Lua、事务，LPUSH/LPOP 等等的并发及线程安全问题。

⑨Linux 环境上如何安装 Redis 6.0.1（6.0 的正式版是 6.0.1）？

这个和安装其他版本的 Redis 没有任何区别，整个流程跑下来也没有任何的坑，所以这里就不做描述了。

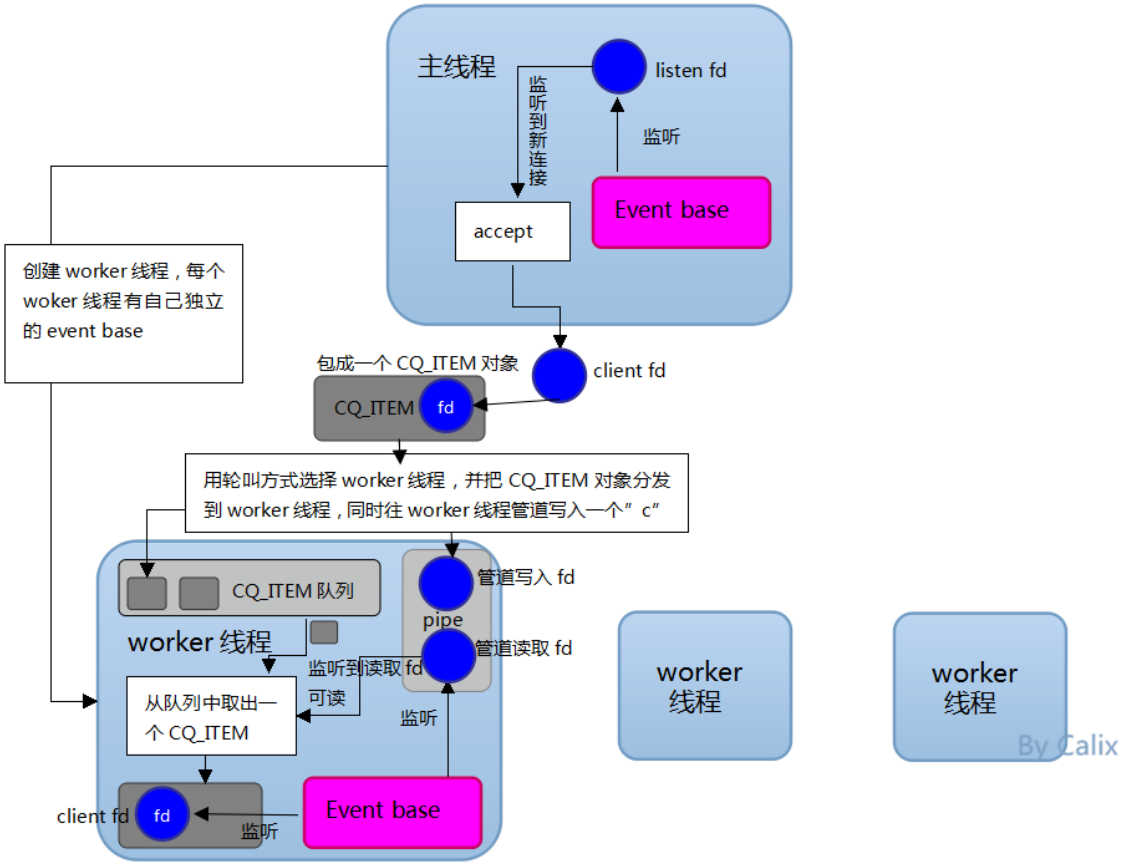
唯一要注意的就是配置多线程数一定要小于 CPU 的核心数，查看核心数量命令：

```
[root@centos7.5 ~]# lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
```

⑩Redis 6.0 的多线程和 Memcached 多线程模型进行对比

前些年 Memcached 是各大互联网公司常用的缓存方案，因此 Redis 和 Memcached 的区别基本成了面试官缓存方面必问的面试题，最近几年 Memcached 用的少了，基本都是 Redis。

不过随着 Redis 6.0 加入了多线程特性，类似的问题可能还会出现，接下来我们只针对多线程模型来简单比较一下。



如上图所示：Memcached 服务器采用 master-woker 模式进行工作，服务端采用 Socket 与客户端通讯。主线程、工作线程 采用 Pipe 管道进行通讯。

主线程采用 Libevent 监听 Listen、Accept 的读事件，事件响应后将连接信息的数据结构封装起来，根据算法选择合适的工作线程，将连接任务携带连接信息分发出去，相应的线程利用连接描述符建立与客户端的 Socket 连接并进行后续的存取数据操作。

Redis 6.0 与 Memcached 多线程模型对比：

- **相同点：**都采用了 Master 线程 -Worker 线程的模型。

- 不同点：Memcached 执行主逻辑也是在 Worker 线程里，模型更加简单，实现了真正的线程隔离，符合我们对线程隔离的常规理解。

  而 Redis 把处理逻辑交还给 Master 线程，虽然一定程度上增加了模型复杂度，但也解决了线程并发安全等问题。

⑪Redis 作者是如何点评 “多线程”这个新特性的?

关于多线程这个特性，在 6.0 RC1 时，Antirez 曾做过说明：

> *Redis 支持多线程有 2 种可行的方式：*
>
> *第一种就是像“Memcached”那样，一个 Redis 实例开启多个线程，从而提升GET/SET等简单命令中每秒可以执行的操作。这涉及到 I/O、命令解析等多线程处理，因此，我们将其称之为“I/O threading”。*
>
> *另一种就是允许在不同的线程中执行较耗时较慢的命令，以确保其它客户端不被阻塞，我们将这种线程模型称为“Slow commands threading”。*

经过深思熟虑，Redis 不会采用“I/O threading”，Redis 在运行时主要受制于网络和内存，所以提升 Redis 性能主要是通过在多个 Redis 实例，特别是 Redis 集群。

接下来我们主要会考虑改进两个方面：

- Redis 集群的多个实例通过编排能够合理地使用本地实例的磁盘，避免同时重写 AOF。
- 提供一个 Redis 集群代理，便于用户在没有较好的集群协议客户端时抽象出一个集群。

补充说明一下，Redis 和 Memcached 一样是一个内存系统，但不同于 Memcached。

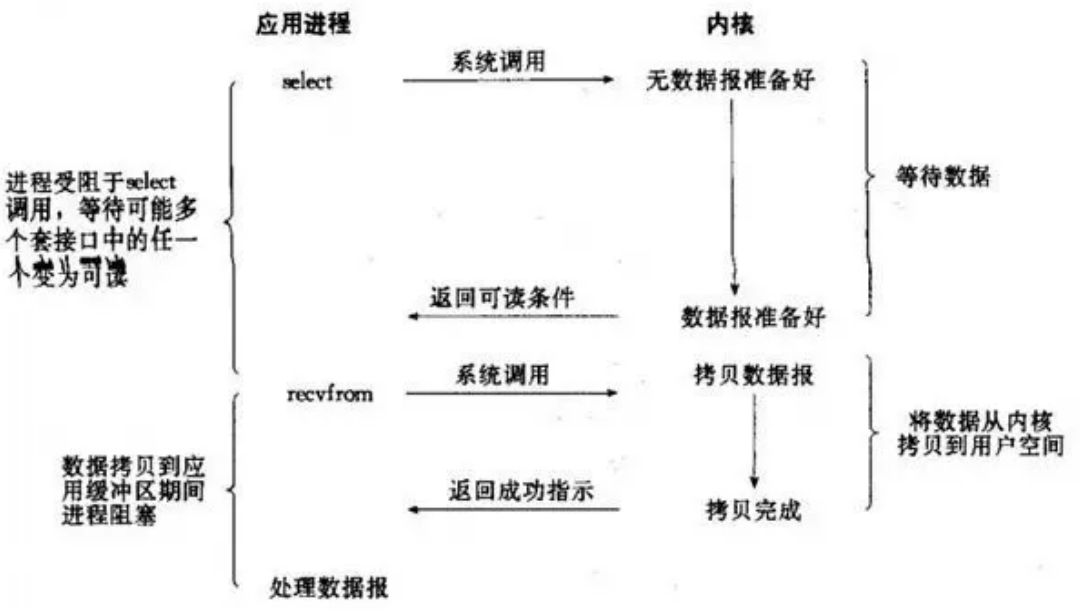
多线程是复杂的，必须考虑使用简单的数据模型，执行 LPUSH 的线程需要服务其他执行 LPOP 的线程。

我真正期望的实际是“slow operations threading”，在 Redis 6 或 Redis 7 中，将提供“key-level locking”，使得线程可以完全获得对键的控制以处理缓慢的操作。详见：

```
http://antirez.com/news/126
```

⑫Redis 线程中经常提到 IO 多路复用，如何理解?

这是 IO 模型的一种，即经典的 Reactor 设计模式，有时也称为异步阻塞 IO。



多路指的是多个 Socket 连接，复用指的是复用一个线程。多路复用主要有三种技术：Select，Poll，Epoll。

Epoll 是最新的也是目前最好的多路复用技术。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（内存内的操作不会成为这里的性能瓶颈），主要以上两点造就了 Redis 具有很高的吞吐量。

⑬你知道 Redis 的彩蛋 LOLWUT 吗?

这个其实从 Redis 5.0 就开始有了，但是原谅我刚刚知道。作者是这么描述这个功能的《LOLWUT: a piece of art inside a database command》，“数据库命令中的一件艺术品”。

你可以把它称之为情怀，也可以称之为彩蛋，具体是什么，我就不透露了。和我一样不清楚是什么的小伙伴可以参见：http://antirez.com/news/123，每次运行都会随机生成的噢。

来源：https://mp.weixin.qq.com/s?__biz=MzU1Nzg4NjgyMw==&mid=2247484957&idx=1&sn=215df72861bc33f9bdf0e0209f92e948&chksm=fc2fba15cb58330313ef38d0716d554b64285f4d86d7d8cf74cb1e46335fe410c0fdbdf58bd6&scene=0&xtrack=1&key=f86ddf70db7ccb584e518d4535f6aa5b8394459f71fcd23dcb5e5d30a7e7f59f80311cfb760387a988e380be788008c6739bdeddcc0579e442c7af8c520af21097bcb62dac7222d3d3d44b98a579c764&ascene=1&uin=MTQ2MjA1MzkzNg%3D%3D&devicetype=Windows+10+x64&version=6209007b&lang=zh_CN&exportkey=AxKljkzqhHZHohj9s0GKrh0%3D&pass_ticket=vZ98BB%2B5438OtLqRSIu5swRq4u6AC0mCDz%2FadgDjRapP37WZSblE5agnTChbd0CN