原理 1：鞭辟入里 —— 线程 IO 模型

Redis 是个单线程程序！这点必须铭记。

也许你会怀疑高并发的 Redis 中间件怎么可能是单线程。很抱歉，它就是单线程，你的怀疑暴露了你基础知识的不足。莫要瞧不起单线程，除了 Redis 之外，Node.js 也是单线程，Nginx 也是单线程，但是它们都是服务器高性能的典范。

Redis 单线程为什么还能这么快?

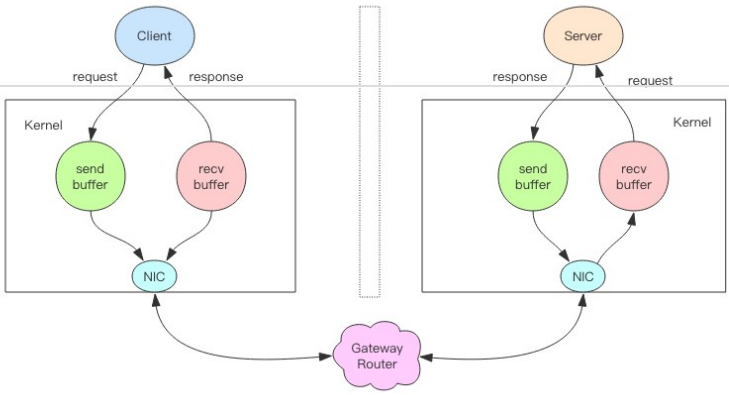
因为它所有的数据都在内存中，所有的运算都是内存级别的运算。正因为 Redis 是单线程，所以要小心使用 Redis 指令，对于那些时间复杂度为 O(n) 级别的指令，一定要谨慎使用，一不小心就可能会导致 Redis 卡顿。

Redis 单线程如何处理那么多的并发客户端连接?

这个问题，有很多中高级程序员都无法回答，因为他们没听过**多路复用**这个词汇，不知道 select 系列的事件轮询 API，没用过非阻塞 IO。

非阻塞 IO

当我们调用套接字的读写方法，默认它们是阻塞的，比如 `read` 方法要传递进去一个参数 `n` ，表示读取这么多字节后再返回，如果没有读够线程就会卡在那里，直到新的数据到来或者连接关闭了， `read` 方法才可以返回，线程才能继续处理。而 `write` 方法一般来说不会阻塞，除非内核为套接字分配的写缓冲区已经满了， `write` 方法就会阻塞，直到缓存区中有空闲空间挪出来了。

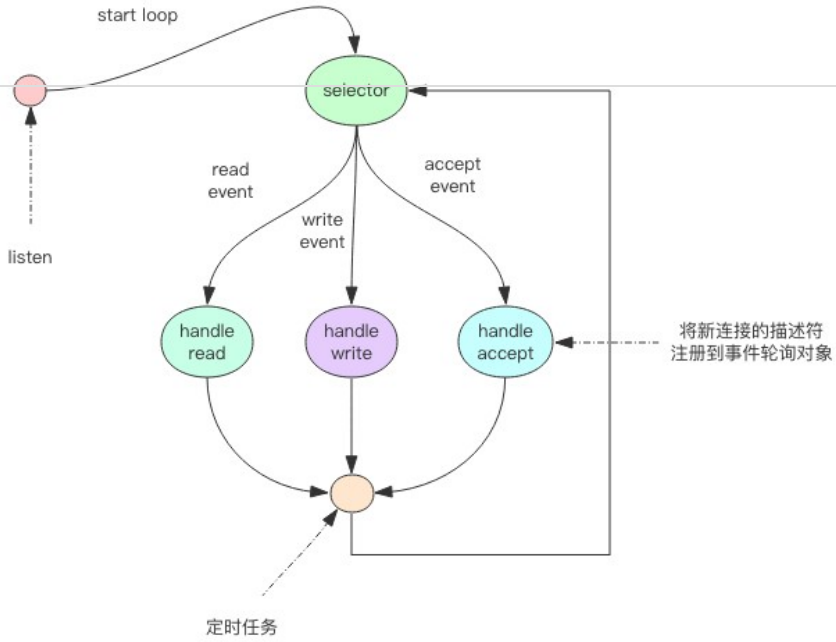


非阻塞 IO 在套接字对象上提供了一个选项 `Non_Blocking` ，当这个选项打开时，读写方法不会阻塞，而是能读多少读多少，能写多少写多少。能读多少取决于内核为套接字分配的读缓冲区内部的数据字节数，能写多少取决于内核为套接字分配的写缓冲区的空闲空间字节数。读方法和写方法都会通过返回值来告知程序实际读写了多少字节。

有了非阻塞 IO 意味着线程在读写 IO 时可以不必再阻塞了，读写可以瞬间完成然后线程可以继续干别的事了。

事件轮询 (多路复用)

非阻塞 IO 有个问题，那就是线程要读数据，结果读了一部分就返回了，线程如何知道何时才应该继续读。也就是当数据到来时，线程如何得到通知。写也是一样，如果缓冲区满了，写不完，剩下的数据何时才应该继续写，线程也应该得到通知。



事件轮询 API 就是用来解决这个问题的，最简单的事件轮询 API 是 `select` 函数，它是操作系统提供给用户程序的 API。输入是读写描述符列表 `read_fds & write_fds` ，输出是与之对应的可读可写事件。同时还提供了一个 `timeout` 参数，如果没有任何事件到来，那么就最多等待 `timeout` 时间，线程处于阻塞状态。一旦期间有任何事件到来，就可以立即返回。时间过了之后还是没有任何事件到来，也会立即返回。拿到事件后，线程就可以继续挨个处理相应的事件。处理完了继续过来轮询。于是线程就进入了一个死循环，我们把这个死循环称为事件循环，一个循环为一个周期。

每个客户端套接字 `socket` 都有对应的读写文件描述符。

py

```
read_events, write_events = select(read_fds, write_fds, timeout)
for event in read_events:
    handle_read(event.fd)
for event in write_events:
    handle_write(event.fd)
handle_others()  # 处理其它事情，如定时任务等
```

因为我们通过 `select` 系统调用同时处理多个通道描述符的读写事件，因此我们将这类系统调用称为多路复用 API。现代操作系统的多路复用 API 已经不再使用 `select` 系统调用，而改用 `epoll(linux)` 和 `kqueue(freebsd & macosx)` ，因为 select 系统调用的性能在描述符特别多时性能会非常差。它们使用起来可能在形

式上略有差异，但是本质上都是差不多的，都可以使用上面的伪代码逻辑进行理解。

服务器套接字 `serversocket` 对象的读操作是指调用 `accept` 接受客户端新连接。何时有新连接到来，也是通过 `select` 系统调用的读事件来得到通知的。

事件轮询 API 就是 Java 语言里面的 NIO 技术

Java 的 NIO 并不是 Java 特有的技术，其它计算机语言都有这个技术，只不过换了一个词汇，不叫 NIO 而已。

指令队列

Redis 会将每个客户端套接字都关联一个指令队列。客户端的指令通过队列来排队进行顺序处理，先到先服务。

响应队列

Redis 同样也会为每个客户端套接字关联一个响应队列。Redis 服务器通过响应队列来将指令的返回结果回复给客户端。 如果队列为空，那么意味着连接暂时处于空闲状态，不需要去获取写事件，也就是可以将当前的客户端描述符从 `write_fds` 里面移出来。等到队列有数据了，再将描述符放进去。避免 `select` 系统调用立即返回写事件，结果发现没什么数据可以写。出这种情况的线程会飙高 CPU。

定时任务

服务器处理要响应 IO 事件外，还要处理其它事情。比如定时任务就是非常重要的一件事。如果线程阻塞在 select 系统调用上，定时任务将无法得到准时调度。那 Redis 是如何解决这个问题的呢?

Redis 的定时任务会记录在一个称为 `最小堆` 的数据结构中。这个堆中，最快要执行的任务排在堆的最上方。在每个循环周期，Redis 都会将最小堆里面已经到点的任务立即进行处理。处理完毕后，将最快要执行的任务还需要的时间记录下来，这个时间就是 `select` 系统调用的 `timeout` 参数。因为 Redis 知道未来

`timeout` 时间内，没有其它定时任务需要处理，所以可以安心睡眠 `timeout` 的时间。

Nginx 和 Node 的事件处理原理和 Redis 也是一样的

扩展阅读

请阅读老钱的另一篇火爆的文章《跟着动画来学习TCP三次握手和四次挥手》