08 | 事件驱动：C10M是如何实现的?

2020-05-15 陶辉

讲述：陶辉
时长 14:02 大小 12.86M

你好，我是陶辉。

上一讲介绍了广播与组播这种一对多通讯方式，从这一讲开始，我们回到主流的一对一通讯方式。

早些年我们谈到高并发，总是会提到 C10K，这是指服务器同时处理 1 万个 TCP 连接。随着服务器性能的提升，近来我们更希望单台服务器的并发能力可以达到 C10M，也就是同时可以处理 1 千万个 TCP 连接。从 C10K 到 C10M，实现技术并没有本质变化，都是事件驱动和异步开发实现的。[第 5 讲] 介绍过的协程，也是依赖这二者实现高并发的。

做过异步开发的同学都知道，处理基于 TCP 的应用层协议时，一个请求的处理代码必须被拆分到多个回调函数中，由异步框架在相应的事件生成时调用它们。这就是事件驱动方式，它通过减少上下文切换次数，实现了 C10M 级别的高并发。

不过，做应用开发的同学往往不清楚什么叫做“事件”，不了解处理 HTTP 请求的回调函数与事件间的关系。这样，在高并发下，当多个 HTTP 请求争抢执行时，涉及资源分配、释放等重要工作的回调函数，就可能在错误的时间被调用，进而引发一系列问题。比如，不同的回调函数对应不同的事件，如果某个函数执行时间过长，就会影响其他请求，可能导致大量请求出现超时而处理失败。

这一讲我们就来介绍一下，事件是怎样产生的？它是如何驱动请求执行的？多路复用技术是怎样协助实现异步开发的？理解了这些，你也就明白了这种事件驱动的解决方案，知道了怎么样实现 C10M。

事件是怎么产生的?

要了解“事件驱动”的运作机制，首先就要搞清楚到底什么是事件。这就需要你对网络原理有深入的理解了。

简单来说，从网络中接收到一个报文，就可能产生一个事件。如上一讲介绍过的 UDP 请求就是最简单的例子，一个 UDP 请求通常仅由一个网络报文组成，所以，当收到一个 UDP 报文，就意味着收到一个请求，它会生成一个事件，进而触发回调函数执行。

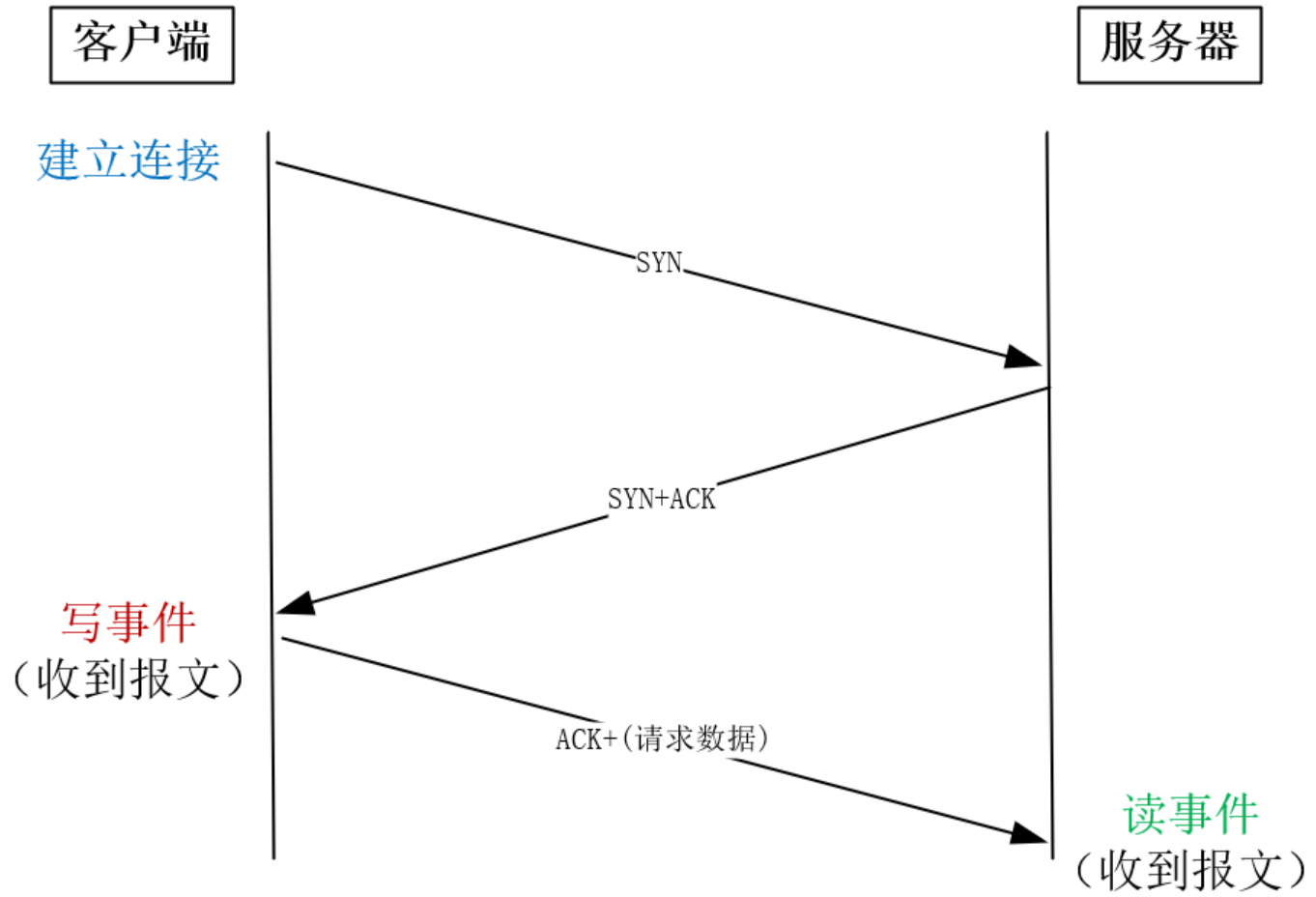
不过，常见的 HTTP 等协议都是基于 TCP 实现的。由于 TCP 是一种面向字节流的协议，HTTP 请求的大小并不受限制，当一个 HTTP 请求的大小超过 TCP 报文的最大长度时，请求会被拆分到多个报文中运输，在接收端的缓冲区中重组、排序。因此，并不是每个到达的 TCP 报文都能生成事件的。

如果不理解事件和 TCP 报文的关系，就没法准确地掌握处理 HTTP 请求的函数何时被调用。当然，作为应用开发工程师，我们无须在意实现细节，只要了解 TCP 连接建立、关闭，以及消息的发送和接收这四个场景中，报文与事件间的关系就可以了。

事件并没有你想象中那么复杂，它只有两种类型：读事件与写事件，其中，读事件表示有到达的消息需要处理，而写事件表示可以发送消息（TCP 连接的写缓冲区中有可用空间）。我们先从三次握手建立连接说起，这一过程会产生一读、一写两个事件。

由于 TCP 允许双向传输，所以**建立连接时，会依次在连接的两个方向上建立通道。**主动发起连接的一方叫做客户端，被动监听端口等待连接的一方叫做服务器。

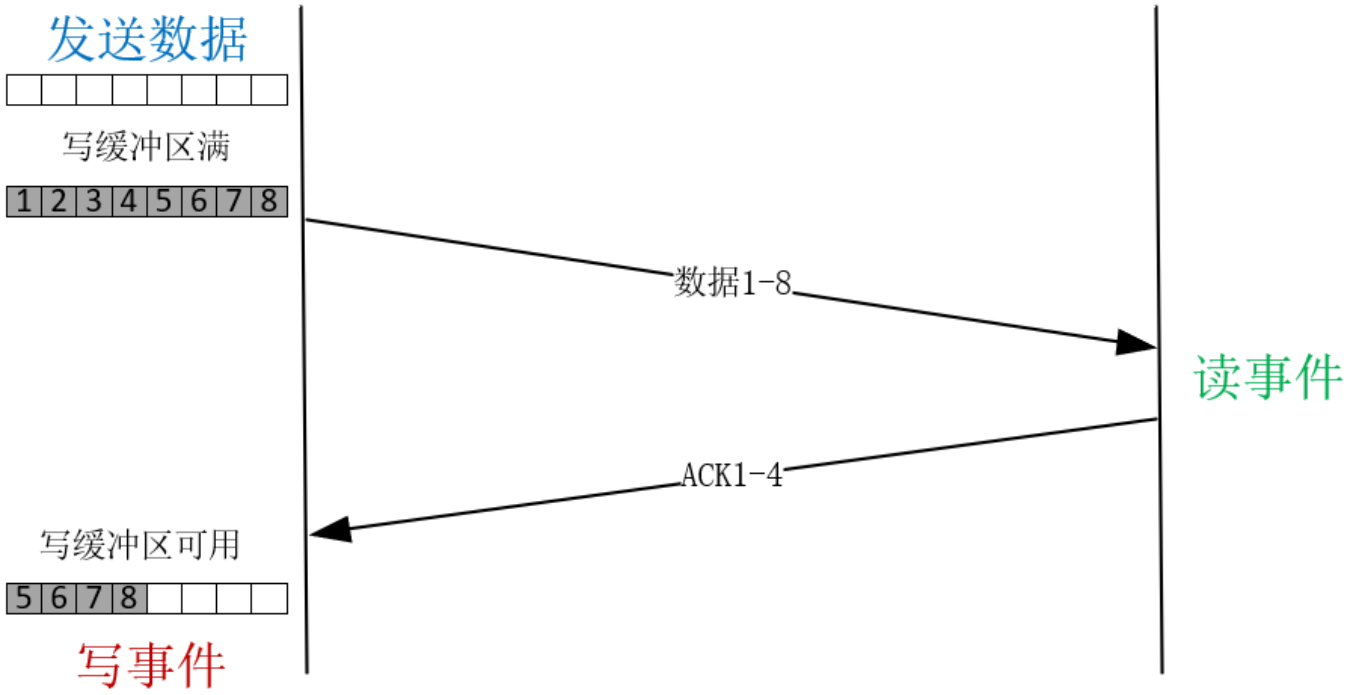
客户端首先发送 SYN 报文给服务器，而服务器收到后回复 ACK 和 SYN（这里我们只需要知道产生事件的过程即可，下一讲会详细介绍这两个报文的含义），**当它们到达客户端时，双向连接中由客户端到服务器的通道就建立好了，此时客户端就已经可以发送请求了，因此客户端会产生写事件。**接着，**客户端发送 ACK 报文，到达服务器后，服务器上会产生读事件**，因为进程原本在监听 80 等端口，此时有新连接建立成功，应当调用 accept 函数读取这个连接，所以这是一个读事件。



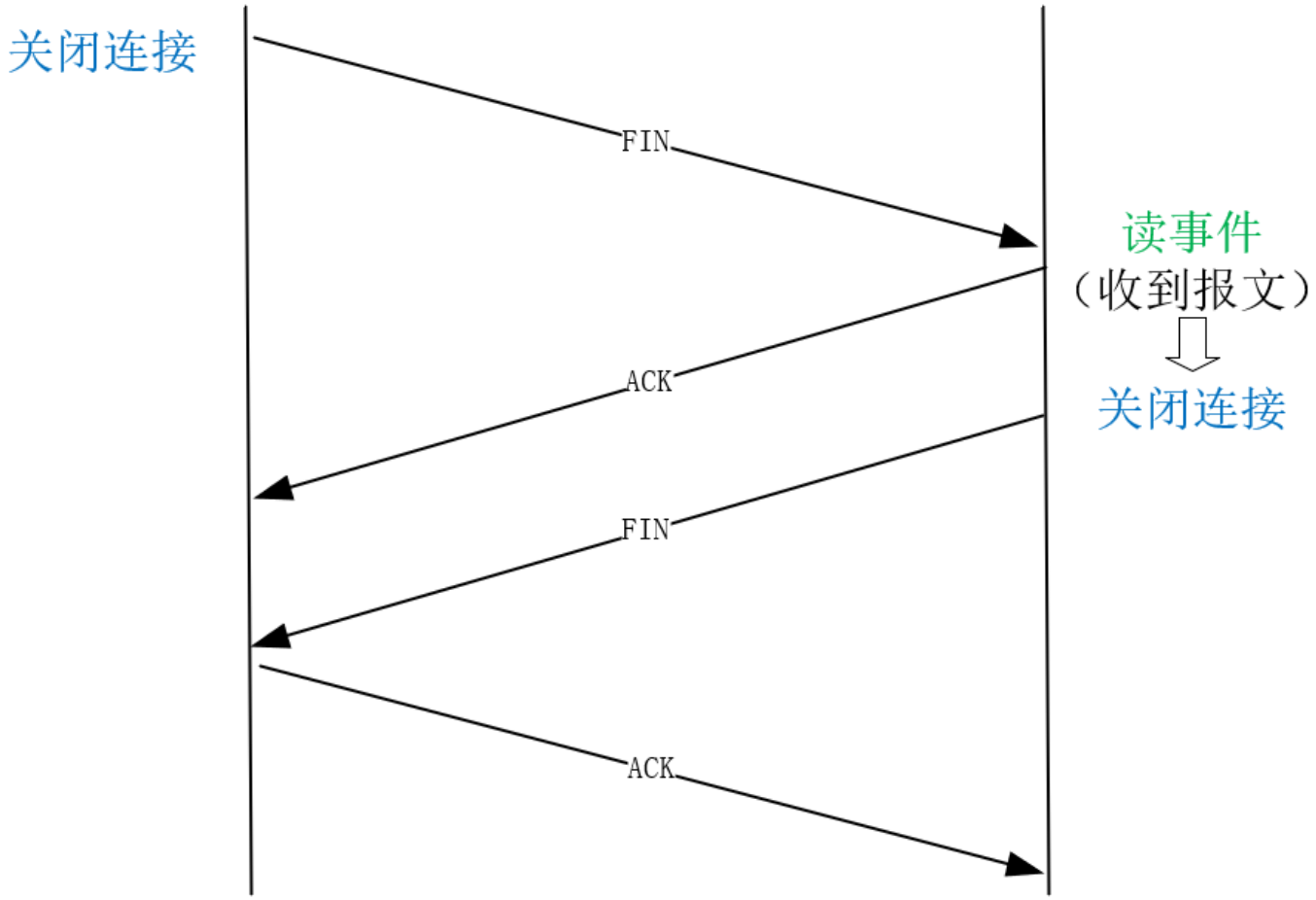
在建立好的 TCP 连接上收发消息时，读事件对应着接收到对方的消息，这很好理解。写事件则稍微复杂些，我们举个例子加以说明。假设要发送一个 2MB 的请求，**当调用 write 函数发送时，会先把内存中的数据拷贝到写缓冲区中后，再发送到网卡上。**

为何要多此一举呢？这是因为在对方没有明确表示收到前，TCP 会通过定时器重发写缓冲区中的数据，保证消息能够到达对方。写缓冲区是有大小限制的，我在[第 10 讲]中会详细介绍。这里假设写缓冲区只有 1MB，所以调用 write 发送 2MB 数据时，write 函数的返回

值只有 1MB，表示写缓冲区已用尽。当收到对方发来的 ACK 报文后，缓冲区中的数据才能释放，就会产生写事件通知进程发送剩余的那 1MB 数据。



如同建立连接需要双向建立一样，关闭连接也需要双方各自关闭每个方向的通道。主动关闭的一方发送 FIN 报文，到达被动方后，内核自动回复 ACK 报文，这表示从主动方到被动方的通道已经关闭。**但被动方到主动方的通道也需要关闭，所以此时被动方会产生读事件，提醒被动方调用 close 函数关闭连接。**



这样，我们就清楚了 TCP 报文如何产生事件，也明白回调函数何时执行了。然而，同步代码拆分成多个异步函数成本并不低，咱们手里拿着事件驱动这个锤子，可不能看到什么都像是钉子。

什么样的代码值得基于事件来做拆分呢？还得回到高性能这个最终目标上来。我们知道，做性能优化一定要找出性能瓶颈，针对瓶颈做优化性价比才最高。对于服务器来说，对最慢的操作做异步化改造，才能值回开发效率的损失。而服务里对资源的操作速度由快到慢，依次是 CPU、内存、磁盘和网络。CPU 和内存的执行速度都是纳秒级的，无须考虑事件驱动，而磁盘和网络都可以采用事件驱动的异步方式处理。

相对而言，网络不只速度慢，而且波动很大，既受制于连接对端的性能，也受制于网络传输路径。把操作网络的同步 API 改为事件驱动的异步 API 收益最大。而磁盘（特别是机械硬盘）访问速度虽然不快，但它最慢时也不过几十毫秒，是可控的。而且目前磁盘异步 IO 技术（参见[第 4 讲]）还不成熟，它绕过了 PageCache 性能损失很大。所以当下的事件驱动，主要就是指网络事件。

该怎样处理网络事件？

有了网络事件的概念后，我们再来看用户态代码如何处理事件。

网络事件是由内核产生的，进程该怎样获取到它们呢？如 epoll 这样的多路复用技术可以帮我们做到。多路复用是通讯领域的词汇，有些抽象但原理确很简单。

比如，一条高速的光纤上，允许多个用户用较低的网速同时通讯，这就是多路复用。同样道理，一个进程虽然任一时刻只能处理一个请求，但处理每个请求产生的事件时，若耗时控制在 1 毫秒以内，这样 1 秒钟就可以处理数千个请求，从更长的时间维度上看，多个请求复用了一个进程，也叫做多路复用（或者叫做时分多路复用）。我们熟知的 epoll，就是内核提供给用户态的多路复用接口，进程可以通过它从内核中获取事件。

epoll 是如何获取网络事件的呢？最简单的方法，就是在获取事件时，把所有并发连接传给内核，再由内核返回产生了事件的连接，再处理这些连接对应的请求即可。epoll 前的 select 等多路复用函数就是这么干的。

然而，C10M 意味着有一千万个连接，若每个 socket 是 4 字节，那么 1 千万连接就是 40M 字节。这样，每收集一次事件，就需要从用户态复制 40M 字节到内核态。而且，高性能 Server 必须及时地处理网络事件，所以每隔几十毫秒就要收集一次事件，性能消耗巨大。

epoll 为了降低性能消耗，把获取事件拆分成两步。

> 第一步把需要监控的 socket 传给内核（epoll_ctl 函数），它仅在连接建立等有限的时机调用；
>
> 第二步收集事件（epoll_wait 函数）便不用传递 socket 了，这样就把 socket 的重复传递改为了一次传递，降低了性能损耗。

由于网卡的处理能力有限，千兆网卡下，每秒只能接收 100MB 左右的数据，如果每个请求约 10KB，那么每秒大概有 1 万个请求到达、10 万个事件需要处理。这样，即使每隔 100 毫秒收集一次事件（调用 epoll_wait），每次也不过只有 1 万个事件（100000 Event/s * 0.1s = 10000 Event/s）需要处理，只要保证处理一个事件的平均时间小于 10 微秒（多核处理器可以做到），100 毫秒内就可以处理完这些事件（100ms = 10us * 10000）。 因此，哪怕有 1 千万并发连接，也能保证 1 万 RPS 的处理能力，这就是 epoll 能在 C10M 下实现高吞吐量的原因。

进程获取到产生事件的 socket 后，又该如何处理它呢？这里的核心约束是，处理任何一个事件的耗时都应该是微秒级或者毫秒级，否则就会延误其他事件的处理，不只降低了用户的体验，而且会形成恶性循环。

我们知道，为了应对网络的不确定性，每个参与网络通讯的进程都会为请求设置超时时间。一旦某个 socket 上的事件迟迟不被处理，当客户端的超时定时器触发时，客户端往往会关闭连接并重发请求，这会让服务器雪上加霜。

怎样保证处理一个事件的时间不会太长呢？ 我们把处理事件的代码分为三类来看。

第一类是计算任务，虽然内存、CPU 的速度很快，然而循环执行也可能耗时达到秒级。所以，如果一定要引入需要密集计算才能完成的请求，为了不阻碍其他事件的处理，要么把这样的请求放在独立的线程中完成，要么把请求的处理过程拆分成多段，确保每段能够快速执行完，同时每段执行完都要均等地处理其他事件，这样通过放慢该请求的处理时间，就保障了其他请求的及时处理。

第二类会读写磁盘，由于磁盘的写入操作使用了 PageCache 的延迟写特性，当 write 函数返回时只是复制到了内存中，所以写入操作很快。磁盘的读取操作就比较慢了，这时，通常要把大文件的读取，拆分成许多份，每份仅有几十 KB，降低单次操作的耗时。

第三类是通过网络访问上游服务。与处理客户端请求相似，我们必须使用非阻塞 socket，用事件驱动方式处理请求。需要注意的是，许多网络服务提供的 SDK，都是基于阻塞 socket 实现的，使用前必须先做完非阻塞改造。比如 Memcached 的官方 SDK 是用阻塞 socket 实现的，Nginx 如果直接使用该 SDK 访问它，性能就会一落千丈。正确的访问方式，是使用第三方提供的 ngx_http_memcached_module 模块，它用非阻塞 socket 重新封装了 SDK。

总之，网络报文到达后，内核就产生了读、写事件，而 epoll 函数使得进程可以高效地收集到这些事件。接下来，要确保在进程中处理每个事件的时间足够短，才能及时地处理所有请求，这个过程中既要避免阻塞 socket 的使用，也要把耗时过长的操作拆成多份执行。最终，通过快速、及时、均等地执行所有事件，异步 Server 实现了高并发。

小结

最后我们对这一讲做个小结。异步服务改为从事件层面处理请求，在 epoll 这样的多路复用机制协助下，最终实现了 C10M 级别的高并发服务。

事件有很多种，网络消息的传输既慢又不可控，所以用网络事件驱动请求的性价比最高。这样，就需要你了解 TCP 报文是如何产生事件的。

TCP 连接建立时，会在客户端产生写事件，在服务器端产生读事件。连接关闭时，则会在被动关闭端产生读事件。在连接上收发消息时，也会产生事件，其中发送消息前的写事件与内核分配的缓冲区有关。

清楚了事件与 TCP 报文的关系后，可以用多路复用技术获取事件，其中 epoll 是佼佼者，它取消了收集事件时重复传递的大量 socket 参数，给 C10M 的实现提供了基础。

你需要注意的是，处理 epoll 收集到的事件时，必须保证处理一个事件的平均时间在毫秒级以内。传统的阻塞 socket 是做不到的，所以必须用非阻塞 socket 替换阻塞 socket。如果事件的回调函数耗时过长，也得拆分为多个耗时短的函数，用多次事件（比如定时器事件）的触发来替代。

虽然我们有了上述的事件驱动方案，但实现 C10M 还需要更谨慎地使用不过数百 GB 的服务器内存。关于如何降低内存的消耗，可以关注[第 2 讲] 提到的内存池，[第 11 讲] 还会介绍如何减少连接缓冲区的空间占用。

这一讲我们介绍了事件驱动的总体方案，但 C10M 需要高效的用心几乎所有服务器资源，所以，我们还得通过 Linux 更精细地控制 TCP 的行为，接下来的 3 讲我们将深入 Linux，讨论如何优化 TCP 的性能。

思考题

最后，留给你一个思考题，需要 CPU 做密集计算的请求，该如何拆分到事件驱动框架中呢？欢迎你在留言区留言，与大家一起探讨。

感谢阅读，如果你觉得这节课对你有一些启发，也欢迎把它分享给你的朋友。

上一篇 07 | 性能好，效率高的一对多通讯该如何实现?

下一篇 09 | 如何提升TCP三次握手的性能?

精选留言 (15)

写留言

忆水寒

2020-05-17

我觉得myrfy的方案很好，如果计算密集型达到一定标准，则可以使用专门的集群去处理，甚至专用芯片。

我之前计算使用傅立叶变换需要计算大量的基带信号，我就是使用专用的FPGA芯片进行处理的。

如果计算量在本机可以做，则可以由专门的reactor线程收进来，其他reactor线程池去计...

展开

作者回复: 对的，专用的环境更有效率

7

myrfy
2020-05-15

我认为涉及到CPU密集计算，就不应该再考虑C10M的问题了。C10M我们可以放在例如网关等逻辑简单的地方，负责数据转发即可。真正的计算放在本地线程池是一种方法，但我更倾向于转发到额外的计算处理集群去处理。

展开

作者回复: myrfy站在了更高的架构层面去设计，很好的方案！

6

RISE
2020-05-18

CPU密集计算的拆分个人认为可以使用协程或多线程先分而治之缩短执行时间，如果还不能满足要求的话可以考虑分发到集群中，或者干脆将计算任务进行有效的拆分，不要一次占用过多的CPU时间

2

那时刻
2020-05-15

老师，关于读事件和写事件有个疑问的地方，基于定义：读事件表示有到达的消息需要处理，而写事件表示可以发送消息。在三次握手里的SYN+ACK，客户端产生写事件，为什么不是读事件呢？先读到SYN+ACK然后触发写事件恢复ACK到服务端

展开

作者回复: 这个读、写面对的对象，是应用代码。SYN+ACK对于系统层来说，是到达了1个报文，需要内核发送ACK去处理，但对于应用层来说，是需要发送消息了，因此是写事件，这是我的理解

2

一步
2020-05-24

如果每个请求约 10KB，那么每秒大概有 1 万个请求到达、10 万个事件需要处理

这里为什么 1 万个请求会有 10万个事件呢？ 每一个 TCP 链接在服务端不就是会产生一个读事件 吗?

展开

作者回复: 你好一步，我这里想表达的是：一个请求若是10KB，那么MSS在1KB左右的话，意味着一个请求会有10个左右的报文，若是报文不是连续到达的话，是可以产生10个事件的

1

Bitstream
2020-05-17

关于思考题的一些想法：
（1）如果真的将一个任务定性为”计算密集型“，个人认为将任务放到独立的计算线程（或线程池）处理，可以充分利用多核cpu（单核cpu另说，现在常见于低端嵌入式平台）的并行计算能力，并且可以将线程绑定于特定cpu核，同时通过优化算法实现，充分利用cpu缓存以进一步提升性能。至于编程效率方面的开销，现在很多异步框架应该可以将类...
展开

1

分清云淡
2020-05-15

见过的最详细、系统的关于写缓冲的分析：https://plantegg.github.io/2019/09/28/%E5%B0%B1%E6%98%AF%E8%A6%81%E4%BD%A0%E6%87%82TCP--%E6%80%A7%E8%83%BD%E5%92%8C%E5%8F%91%E9%80%81%E6%8E%A5%E6%94%B6Buffer%E7%9A%84%E5%85%B3%E7%B3%BB/
展开

1

龙龙
2020-05-31

因此，哪怕有 1 千万并发连接，也能保证 1 万 RPS 的处理能力，这就是 epoll 能在 C10M 下实现高吞吐量的原因。
————————————————————————————-
老师，这段话我不太理解，1千万的并发连接，只有1万的RPS 这能算高吞吐量吗？相当于每秒只有1000个人中的1个人得到响应。还是我理解错了，您表述的是另一层意思？请老...
展开

哈哈
2020-05-25

您好，看了您的其他回复，我没太明白拆成小任务多次执行的原因，这样做的好处是什么呢?

作者回复: 举个例子，比如一次执行1秒计算的任务，如果有100个并发请求，就会导致大量请求出现超时重试。比如，通常客户端配置的是60秒超时，那么60个并发请求，将持续导致一颗CPU在1分钟内不能处理其他任何请求，这样第61及之后的请求都会因为超时，被客户端关闭。
反之，拆成100个10ms的小任务，即使每个请求处理的慢了一些，但不会导致大量客户端请求超时。

流浪地球
2020-05-15

老师您好，关于这段描述理解不是很清楚：
但被动方到主动方的通道也需要关闭，所以此时被动方会产生读事件，提醒被动方调用 close 函数关闭连接。
此时，被动方此时应该不会再收到主动方的消息了，为什么产生的是读事件，而不是应该是写事件。...
展开

作者回复: 产生读事件，提醒应用程序需要处理对方的FIN报文。此时未必需要调用close函数，当主动方使用shutdown函数发送FIN报文时，被动方可以长时间保持半关闭连接状态，单向传输数据

凌晨
2020-05-15

思考题正是我们目前项目遇到的问题，请老师指点一下：

简单描述下问题：我们在nginx中实现一个上传文件时同时计算文件sha256的功能，客户端会携带sha256值用于校验完整性（这个需求是死的，客户端不可能改）。计算sha256是CPU密集型操作，一定要考虑异步化，否则ngx worker进程会"停顿"，出现大量毛刺。...
展开

作者回复: 你好凌晨，我推荐用第3种方式，这样的开发复杂度最小，也最可控！

唐朝首都
2020-05-15

文中在单线程10us处理事件，能够支持C10M的吞吐量，但是在实际项目的运行中事件没有这么快的处理能力，我了解的处理方式有Reactor Pattern，主线程不断循环轮训事件，将就绪的事件丢到Worker线程池中进行处理。

展开


凉人。
2020-05-15

感觉c10k提升，绝大部分是在压榨性能上的提升。
单位时间处理数。
内存池。
非阻塞。
异步事件驱动。...

展开

作者回复: 是的


那时刻
2020-05-15

我的想法是采用线程池和队列加入到事件驱动框架里来处理CPU密集计算的请求。读事件是请求的读取，把读取的请求放到队列里，然后事件处理线程池从队列里取事件处理，处理完之后触发写事件（写网络或者写文件等）。因为是CPU密集计算，线程数量不能过多，一般为CPU核数的2倍。队列的长度可以稍微长点，来容纳不能及时处理的事件。

展开

作者回复: 你好那时刻，当处理请求的过程无法拆分时（比如第三方无法拆分的SDK），用独立的线程池处理是个好方案！如果能够拆分的话，直接拆成小任务多次执行就不错，无须针对多线程同步付出额外的成本，你可以参考下凌晨同学的答案。


安排
2020-05-15

思考题：可以弄一个计算线程池，多路复用只负责将网络事件收上来，然后交给线程池计算，这样多路复用的回调可以立刻返回并处理下一个事件回调。计算线程池弄一个eventfd，多路复用也监听这个eventfd，线程池算完了之后，通过eventfd通知多路复用。多路复用将计算结果写到socket缓冲区.

展开