# 【第02章-NIO入门】【02-伪异步IO编程】

## 伪异步IO编程

同步阻塞IO一个链路需要一个线程处理，有人对它的线程模型进行了优化——后端通过一个线程池来处理多个客户端的请求接入，形成客户端个数M： 线程池最大线程数N 的比例关系，其中M 可以远远大于N。通过线程地可以灵活地调配线程资源，设置线程的最大值，防止由于海量并发接入导致线程耗尽。

## 伪异步IO模型图

采用线程池和任务队列可以实现一种叫做伪异步的IO 通信框架，它的模型图如下图所示。当有新的客户端接入时， 将客户端的Socket 封装成一个Task（该任务实现java.lang.Runnable 接口）投递到后端的线程池中进行处理， JDK 的线程地维护一个消息队列和N个活跃线程， 对消息队列中的任务进行处理。由于线程池可以设置消息队列的大小和最大线程数， 因此，它的资源占用是可控的， 无论多少个客户端并发访问， 都不会导致资源的耗尽和宕机。



图1 伪异步IO服务端通信模型（M:N）

## 伪异步IO模型弊端

对于字节输入流java.io.InputStream公开的read方法，其说明都有这么一句：

This method blocks until input data is available, end of file is detected, or an exception is thrown.

当对Socket 的输入流进行读取操作的时候，它会一直阻塞下去， 直到发生如下三种事件。

* 有数据可读
* 可用数据已经读取完毕
* 发生空指针或者IO异常

这意味着当对方发送请求或者应答消息比较缓慢，或者网络传输较慢时，读取输入流一方的通信线程将被长时间阻塞，如果对方要60s 才能够将数据发送完成，读取一方的110线程也将会被同步阻塞60s，在此期间，其他接入消息只能在消息队列中排队。

对于java.io.OutputStream类，API文档是虽然没有明确说明write是否阻塞。调用OutputStrearn 的write方法写输出流的时候，它将会被阻塞，直到所有需发送的字节全部写入完毕，或者发生异常。学习过TCP/IP 相关知识的人都知道，当消息的接收方处理缓慢的时候，将不能及时地从TCP 缓冲区读取数据，这将会导致发送方的TCP window size（滑动窗口）不断减小，直到为0，双方处于Keep-Alive状态，消息发送方将不能再向TCP缓冲区写入消息，这时如果采用的是同步阻塞IO write操作将会被无限期阻塞， 直到TCP window size 大于0 或者发生IO异常。

通过对输入和输出流的AP I 文档进行分析，我们了解到读和写操作都是同步阻塞的，阻塞的时间取决于对方IO线程的处理速度和网络IO的传输速度。本质上来讲， 我们无法保证生产环境的网络状况和对端的应用程序能足够快，如果我们的应用程序依赖对方的处理速度，它的可靠性就非常差。也许在实验室进行的性能测试结果令人满意，但是一旦上线运行，面对恶劣的网络环境和l良莠不齐的第三方系统， 问题就会如火山一样喷发。

伪异步IO实际上仅仅是对之前IO线程模型的一个简单优化，它无法从根本上解决同步IO导致的通信线程阻塞问题。下面我们就简单分析下通信对方返回应答时间过长会引起的级联故障。

1. 服务端处理缓慢，返回应答消息耗费60s ， 平时只需要10ms.
2. 采用伪异步IO的钱程正在读取故障服务节点的响应，由于读取输入流是阻塞的，它将会被同步阻塞60s。
3. 假如所有的可用线程都被故障服务器阻塞，那后续所有的IO 消息都将在队列中排队。
4. 由于线程池采用阻塞队列实现，当队列积满之后，后续入队列的操作将被阻塞。
5. 由于前端只有一个Accptor 线程接收客户端接入，它被阻塞在线程池的同步阻塞队列之后， 新的客户端请求消息将被拒绝， 客户端会发生大量的连接超时。
6. 由于几乎所有的连接都超时，调用者会认为系统已经崩溃，无法接收新的请求消息。