



链滴

如何理解 BIO、NIO、AIO 的区别?

作者: [valarchie](#)

原文链接: <https://ld246.com/article/1570167923186>

来源网站: 链滴

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

很多文章在谈论到BIO、NIO、AIO的时候仅仅是抛出一堆定义，以及一些生动的例子。看似很好理解。但是并没有将最基础的本质原理显现出来，如果没有从IO的原理出发的话是很难理解这三者之间的区别的。所以本篇文章从Java是如何进行IO操作作为开头进行分析。

Java中的IO原理

首先Java中的IO都是依赖操作系统内核进行的，我们程序中的IO读写其实调用的是操作系统内核中的read&write两大系统调用。

那内核是如何进行IO交互的呢？

1. 网卡收到经过网线传来的网络数据，并将网络数据写到内存中。
2. 当网卡把数据写入到内存后，网卡向cpu发出一个中断信号，操作系统便能得知有新数据到来，再过网卡中断程序去处理数据。
3. 将内存中的网络数据写入到对应socket的接收缓冲区中。
4. 当接收缓冲区的数据写好之后，应用程序开始进行数据处理。

对应抽象到java的socket代码简单示例如下：

```
public class SocketServer {
    public static void main(String[] args) throws Exception {
        // 监听指定的端口
        int port = 8080;
        ServerSocket server = new ServerSocket(port);
        // server将一直等待连接的到来
        Socket socket = server.accept();
        // 建立好连接后，从socket中获取输入流，并建立缓冲区进行读取
        InputStream inputStream = socket.getInputStream();
        byte[] bytes = new byte[1024];
        int len;
        while ((len = inputStream.read(bytes)) != -1) {
            //获取数据进行处理
            String message = new String(bytes, 0, len, "UTF-8");
        }
        // socket、server，流关闭操作，省略不表
    }
}
```

可以看到这个过程和底层内核的网络IO很类似，主要体现在accept()等待从网络中的请求到来然后bytes[]数组作为缓冲区等待数据填满后进行处理。而BIO、NIO、AIO之间的区别就在于这些操作是同步还是异步，阻塞还是非阻塞。

所以我们引出同步异步，阻塞与非阻塞的概念。

同步与异步

同步和异步指的是一个执行流程中每个方法是否必须依赖前一个方法完成后才可以继续执行。假设我的执行流程中：依次是方法一和方法二。

同步指的是调用一旦开始，调用者必须等到方法调用返回后，才能继续后续的行为。即方法二一定要

到方法一执行完成后才可以执行。

异步指的是调用立刻返回，调用者不必等待方法内的代码执行结束，就可以继续后续的行为。（具体法内的代码交由另外的线程执行完成后，可能会进行回调）。即执行方法一的时候，直接交给其他线程执行，不由主线程执行，也就不会阻塞主线程，所以方法二不必等到方法一完成即可开始执行。

同步与异步关注的是方法的执行方是主线程还是其他线程，主线程的话需要等待方法执行完成，其他线程的话无需等待立刻返回方法调用，主线程可以直接执行接下来的代码。

同步与异步是从多个线程之间的协调来实现效率差异。

为什么需要异步呢？笔者认为异步的本质就是为了解决主线程的阻塞，所以网上很多讨论把同步异步阻塞非阻塞进行了四种组合，其中一种就有异步阻塞这一情形，如果异步也是阻塞的？那为什么要特地进行异步操作呢？

阻塞与非阻塞

阻塞与非阻塞指的是单个线程内遇到同步等待时，是否在原地不做任何操作。

阻塞指的是遇到同步等待后，一直在原地等待同步方法处理完成。

非阻塞指的是遇到同步等待，不在原地等待，先去做其他的操作，隔断时间再来观察同步方法是否完成。

阻塞与非阻塞关注的是线程是否在原地等待。

笔者认为阻塞和非阻塞仅能与同步进行组合。而异步天然就是非阻塞的，而这个非阻塞是对主线程而言。（可能有人认为异步方法里面放入阻塞操作的话就是异步阻塞，但是思考一下，正是因为是阻塞操作所以才会将它放入异步方法中，不要阻塞主线程）

例子讲解

海底捞很好吃，但是经常要排队。我们就以生活中的这个例子进行讲解。

- A顾客去吃海底捞，就这样干坐着等了一小时，然后才开始吃火锅。(BIO)
- B顾客去吃海底捞，他一看要等挺久，于是去逛商场，每次逛一会就跑回来看有没有排到他。于是最后既购了物，又吃上海底捞了。(NIO)
- C顾客去吃海底捞，由于他是高级会员，所以店长说，你去商场随便玩吧，等下有位置，我立马打电话给你。于是C顾客不用干坐着等，也不用每过一会儿就跑回来看有没有等到，最后也吃上了海底捞(AIO)

哪种方式更有效率呢？是不是一目了然呢？

BIO

BIO全称是Blocking IO，是JDK1.4之前的传统IO模型，本身是同步阻塞模式。

线程发起IO请求后，一直阻塞IO，直到缓冲区数据就绪后，再进入下一步操作。针对网络通信都是一求一应答的方式，虽然简化了上层的应用开发，但在性能和可靠性方面存在着巨大瓶颈，试想一下如每个请求都需要新建一个线程来专门处理，那么在高开发的场景下，机器资源很快就会被耗尽。

NIO

NIO也叫Non-Blocking IO 是同步非阻塞的IO模型。线程发起io请求后，立即返回（非阻塞io）。同指的是必须等待IO缓冲区内的数据就绪，而非阻塞指的是，用户线程不原地等待IO缓冲区，可以先做些其他操作，但是要定时轮询检查IO缓冲区数据是否就绪。Java中的NIO 是new IO的意思。其实是NIO加上IO多路复用技术。普通的NIO是线程轮询查看一个IO缓冲区是否就绪，而Java中的new IO指的线程轮询地去查看一堆IO缓冲区中哪些就绪，这是一种IO多路复用的思想。IO多路复用模型中，将检查IO数据是否就绪的任务，交给系统级别的select或epoll模型，由系统进行监控，减轻用户线程负担。

NIO主要有buffer、channel、selector三种技术的整合，通过零拷贝的buffer取得数据，每一个客户通过channel在selector（多路复用器）上进行注册。服务端不断轮询channel来获取客户端的信息。channel上有connect,accept（阻塞）、read（可读）、write(可写)四种状态标识。根据标识来进行后操作。所以一个服务端可接收无限多的channel。不需要新开一个线程。大大提升了性能。

AIO

AIO是真正意义上的异步非阻塞IO模型。

上述NIO实现中，需要用户线程定时轮询，去检查IO缓冲区数据是否就绪，占用应用程序线程资源，实轮询相当于还是阻塞的，并非真正解放当前线程，因为它还是需要去查询哪些IO就绪。而真正的理的异步非阻塞IO应该让内核系统完成，用户线程只需要告诉内核，当缓冲区就绪后，通知我或者执行交给你的回调函数。

AIO可以做到真正的异步的操作，但实现起来比较复杂，支持纯异步IO的操作系统非常少，目前也就windows是IOCP技术实现了，而在Linux上，底层还是使用的epoll实现的。

以上是笔者的一点拙见，如有理解偏差恳请大家评论指正