**IO的方式通常分为几种，**

**同步阻塞的BIO、**

**同步非阻塞的NIO、**

**异步非阻塞的AIO。**

一、BIO

     在JDK1.4出来之前，我们建立网络连接的时候采用BIO模式，需要先在服务端启动一个ServerSocket，然后在客户端启动Socket来对服务端进行通信，

默认情况下服务端需要对每个请求建立一堆线程等待请求，**而客户端发送请求后，先咨询服务端是否有线程相应，如果没有则会一直等待或者遭到拒绝请求，如果有的话，客户端会线程会等待请求结束后才继续执行。(等待的过程即是阻塞)**

二、NIO

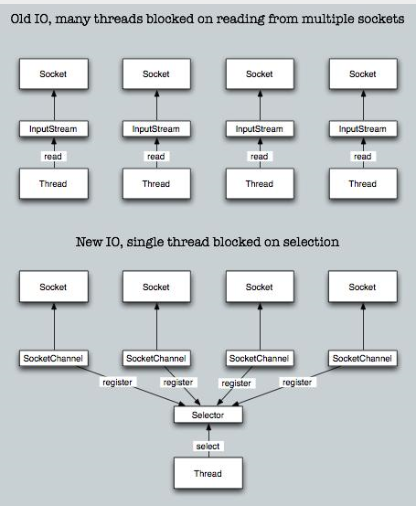
NIO本身是基于事件驱动思想来完成的，其主要想解决的是BIO的大并发问题：

在使用**同步I/O的网络应用中**，如果要同时处理多个客户端请求，或是在客户端要同时和多个服务器进行通讯，就必须使用多线程来处理。也就是说，**将每一个客户端请求分配给一个线程来单独处理**。这样做虽然可以达到我们的要求，但同时又会带来另外一个问题。由于每创建一个线程，就要为这个线程分配一定的内存空间（也叫工作存储器），而且操作系统本身也对线程的总数有一定的限制。如果客户端的请求过多，服务端程序可能会因为不堪重负而拒绝客户端的请求，甚至服务器可能会因此而瘫痪。

**NIO基于Reactor，当socket有流可读或可写入socket时，操作系统会相应的通知应用程序进行处理，应用再将流读取到缓冲区或写入操作系统。**

也就是说，这个时候，已经不是一个连接就要对应一个处理线程了，而**是有效的请求，对应一个线程，当连接没有数据时，是没有工作线程来处理的。**

   BIO与NIO一个比较重要的不同，是我们使用BIO的时候往往会引入多线程，每个连接一个单独的线程；而NIO则**是使用单线程或者只使用少量的多线程，多个socket共用一个线程。**



      NIO的最重要的地方是当一个连接创建后，**不需要对应一个线程，这个连接会被注册到多路复用器上面，**所以**所有的连接只需要一个线程**就可以搞定，当这个线程中的多路复用器进行轮询的时候，**发现连接上有请求的话，才开启一个线程进行处理，也就是一个请求一个线程模式。**

      在NIO的处理方式中，当**一个请求来的话，开启线程进行处理，可能会等待后端应用的资源(JDBC连接等)，其实这个线程就被阻塞了**，当并发上来的话，还是会有BIO一样的问题。

　　HTTP/1.1出现后，有了Http长连接，这样除了超时和指明特定关闭的http header外，这个链接是一直打开的状态的，这样在NIO处理中可以进一步的进化，在**后端资源中可以实现资源池或者队列**，当请求来的话，开启的线程**把请求和请求数据传送**给后端资源池或者**队列里面就返回**，并且在全局的地方保持住这个现场(哪个连接的哪个请求等)，这样**前面的线程还是可以去接受其他的请求，而后端的应用的处理只需要执行队列里面的就可以了，这样请求处理和后端应用是异步的.**当后端处理完，到全局地方得到现场，产生响应，**这个就实现了异步处理。**

三、AIO

     与NIO不同，**当进行读写操作时，只须直接调用API的read或write方法即可**。

这两种方法均为异步的，对于读操作而言，当有流可读取时，操作系统会将可读的流传入read方法的缓冲区，再通知应用程序；

对于写操作而言，当操作系统将write方法传递的流写入完毕时，操作系统主动通知应用程序。  **即可以理解为，read/write方法都是异步的，完成后会主动调用回调函数。（同步的就是有流进来了，通知应用程序自己来读写）**

在JDK1.7中，这部分内容被称作NIO.2，主要在java.nio.channels包下增加了下面四个异步通道：

* AsynchronousSocketChannel
* AsynchronousServerSocketChannel
* AsynchronousFileChannel
* AsynchronousDatagramChannel

其中的read/write方法，会返回一个带回调函数的对象，当执行完读取/写入操作后，直接调用回调函数。

先来个例子理解一下概念，以银行取款为例：

* 同步 ： 自己亲自出马持银行卡到银行取钱（使用同步IO时，Java自己处理IO读写）；
* 异步 ： 委托一小弟拿银行卡到银行取钱，然后给你（使用异步IO时，Java将IO读写委托给OS处理，需要将数据缓冲区地址和大小传给OS(银行卡和密码)，OS需要支持异步IO操作API）；
* 阻塞 ： ATM排队取款，你只能等待（使用阻塞IO时，Java调用会一直阻塞到读写完成才返回）；
* 非阻塞 ： 柜台取款，取个号，然后坐在椅子上做其它事，等号广播会通知你办理，没到号你就不能去，你可以不断问大堂经理排到了没有，大堂经理如果说还没到你就不能去（使用非阻塞IO时，如果不能读写Java调用会马上返回，当IO事件分发器会通知可读写时再继续进行读写，不断循环直到读写完成）

Java对BIO、NIO、AIO的支持：

* Java BIO ： **同步并阻塞**，服务器实现模式为一个连接一个线程，即**客户端有连接请求时服务器端就需要启动一个线程进行处理**，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。
* Java NIO ： **同步非阻塞**，服务器实现模式为**一个请求一个线程**，即客户端**发送的连接请求都会注册到多路复用器上**，多路复用器**轮询到连接有I/O请求时才启动一个线程进行处理**。
* Java AIO(NIO.2) ： 异步非阻塞，服务器实现模式为**一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理，**

BIO、NIO、AIO适用场景分析:

* BIO方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4以前的唯一选择，但程序直观简单易理解。
* NIO方式适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持。
* AIO方式使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持。

另外，I/O属于底层操作，需要操作系统支持，并发也需要操作系统的支持，所以性能方面不同操作系统差异会比较明显。

在高性能的I/O设计中，有两个比较著名的模式**Reactor和Proactor模式**，

其中Reactor模式用于**同步I/O，**而Proactor运用于**异步I/O操作**。

在比较这两个模式之前，我们首先的搞明白几个概念，什么是阻塞和非阻塞，什么是同步和异步,

**同步和异步是针对应用程序和内核的交互而言的，**

**同步指的是用户进程触发IO操作并等待或者轮询的去查看IO操作是否就绪，而异步是指用户进程触发IO操作以后便开始做自己的事情，而当IO操作已经完成的时候会得到IO完成的通知(自己完成为同步，OS完成为异步)。**

而阻塞和非阻塞是针对于进程在访问数据的时候，根据IO操作的就绪状态来采取的不同方式，说白了是一种读取或者写入操作函数的实现方式，

**阻塞方式下读取或者写入函数将一直等待，而非阻塞方式下，读取或者写入函数会立即返回一个状态值。**