1. BIO、AIO、NIO区别

****在高性能的IO体系设计中，有几个名词概念常常会使我们感到迷惑不解。具体如下：****

|  |  |
| --- | --- |
| 序号 | 问题 |
| 1 | 什么是同步？ |
| 2 | 什么是异步？ |
| 3 | 什么是阻塞？ |
| 4 | 什么是非阻塞？ |
| 5 | 什么是同步阻塞？ |
| 6 | 什么是同步非阻塞？ |
| 7 | 什么是异步阻塞？ |
| 8 | 什么是异步非阻塞？ |

****散仙不才，在查了一部分资料后，愿试着以通俗易懂的方式解释下这几个名词。如有不足之处，还望告知。****  
****在弄清楚上面的几个问题之前，我们首先得明白什么是同步，异步，阻塞，非阻塞，只有这几个单个概念理解清楚了，然后在组合理解起来，就相对比较容易了。   
1,同步和异步是针对应用程序和内核的交互而言的。   
2,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。    
由上描述基本可以总结一句简短的话，同步和异步是目的，阻塞和非阻塞是实现方式。****

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 名词 | 解释 | 举例 |
| 1 | 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 | 自己上街买衣服，自己亲自干这件事，别的事干不了。 |
| 2 | 异步 | 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（异步的特点就是通知） | 告诉朋友自己合适衣服的尺寸，大小，颜色，让朋友委托去卖，然后自己可以去干别的事。（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） |
| 3 | 阻塞 | 所谓阻塞方式的意思是指, 当试图对该文件描述符进行读写时, 如果当时没有东西可读,或者暂时不可写, 程序就进入等待 状态, 直到有东西可读或者可写为止 | 去公交站充值，发现这个时候，充值员不在（可能上厕所去了），然后我们就在这里等待，一直等到充值员回来为止。（当然现实社会，可不是这样，但是在计算机里确实如此。） |
| 4 | 非阻塞 | 非阻塞状态下, 如果没有东西可读, 或者不可写, 读写函数马上返回, 而不会等待， | 银行里取款办业务时，领取一张小票，领取完后我们自己可以玩玩手机，或者与别人聊聊天，当轮我们时，银行的喇叭会通知，这时候我们就可以去了。 |

****下面我们再来理解组合方式的IO类型，就好理解多了。   
同步阻塞IO（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。   
同步非阻塞IO(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。用户进程也需要时不时的询问IO操作是否就绪，这就要求用户进程不停的去询问。   
异步阻塞IO（Java NIO）：    
   此种方式下是指应用发起一个IO操作以后，不等待内核IO操作的完成，等内核完成IO操作以后会通知应用程序，这其实就是同步和异步最关键的区别，同步必须等待或者主动的去询问IO是否完成，那么为什么说是阻塞的呢？因为此时是通过select系统调用来完成的，而select函数本身的实现方式是阻塞的，而采用select函数有个好处就是它可以同时监听多个文件句柄（如果从UNP的角度看，select属于同步操作。因为select之后，进程还需要读写数据），从而提高系统的并发性！    
（Java AIO(NIO.2)）异步非阻塞IO:    
   在此种模式下，用户进程只需要发起一个IO操作然后立即返回，等IO操作真正的完成以后，应用程序会得到IO操作完成的通知，此时用户进程只需要对数据进行处理就好了，不需要进行实际的IO读写操作，因为真正的IO读取或者写入操作已经由内核完成了。      
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。****  
****搞清楚了以上概念以后，我们再回过头来看看，Reactor模式和Proactor模式。    
（其实阻塞与非阻塞都可以理解为同步范畴下才有的概念，对于异步，就不会再去分阻塞非阻塞。对于用户进程，接到异步通知后，就直接操作进程用户态空间里的数据好了。）    
  首先来看看Reactor模式，Reactor模式应用于同步I/O的场景。我们分别以读操作和写操作为例来看看Reactor中的具体步骤：    
读取操作：    
1. 应用程序注册读就绪事件和相关联的事件处理器    
2. 事件分离器等待事件的发生      
3. 当发生读就绪事件的时候，事件分离器调用第一步注册的事件处理器    
4. 事件处理器首先执行实际的读取操作，然后根据读取到的内容进行进一步的处理     
写入操作类似于读取操作，只不过第一步注册的是写就绪事件。    
   
下面我们来看看Proactor模式中读取操作和写入操作的过程：    
读取操作：    
1. 应用程序初始化一个异步读取操作，然后注册相应的事件处理器，此时事件处理器不关注读取就绪事件，而是关注读取完成事件，这是区别于Reactor的关键。    
2. 事件分离器等待读取操作完成事件    
    
3. 在事件分离器等待读取操作完成的时候，操作系统调用内核线程完成读取操作（异步IO都是操作系统负责将数据读写到应用传递进来的缓冲区供应用程序操作，操作系统扮演了重要角色），并将读取的内容放入用户传递过来的缓存区中。这也是区别于Reactor的一点，Proactor中，应用程序需要传递缓存区。    
4. 事件分离器捕获到读取完成事件后，激活应用程序注册的事件处理器，事件处理器直接从缓存区读取数据，而不需要进行实际的读取操作。    
Proactor中写入操作和读取操作，只不过感兴趣的事件是写入完成事件。    
从上面可以看出，Reactor和Proactor模式的主要区别就是真正的读取和写入操作是有谁来完成的，Reactor中需要应用程序自己读取或者写入数据，而Proactor模式中，应用程序不需要进行实际的读写过程，它只需要从缓存区读取或者写入即可，操作系统会读取缓存区或者写入缓存区到真正的IO设备.    
          综上所述，同步和异步是相对于应用和内核的交互方式而言的，同步 需要主动去询问，而异步的时候内核在IO事件发生的时候通知应用程序，而阻塞和非阻塞仅仅是系统在调用系统调用的时候函数的实现方式而已。****  
****如果你想吃一份宫保鸡丁盖饭：  
同步阻塞：你到饭馆点餐，然后在那等着，还要一边喊：好了没啊！  
同步非阻塞：在饭馆点完餐，就去遛狗了。不过溜一会儿，就回饭馆喊一声：好了没啊！   
异步阻塞：遛狗的时候，接到饭馆电话，说饭做好了，让您亲自去拿。   
异步非阻塞：饭馆打电话说，我们知道您的位置，一会给你送过来，安心遛狗就可以了。****   
  
****“一个IO操作其实分成了两个步骤：发起IO请求和实际的IO操作。   
同步IO和异步IO的区别就在于第二个步骤是否阻塞，如果实际的IO读写阻塞请求进程，那么就是同步IO。   
阻塞IO和非阻塞IO的区别在于第一步，发起IO请求是否会被阻塞，如果阻塞直到完成那么就是传统的阻塞IO，如果不阻塞，那么就是非阻塞IO。   
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。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。   
所以,IO操作可以分为3类：同步阻塞（即早期的IO操作）、同步非阻塞（NIO）、异步（AIO）。   
同步阻塞：   
在此种方式下，用户进程在发起一个IO操作以后，必须等待IO操作的完成，只有当真正完成了IO操作以后，用户进程才能运行。JAVA传统的IO模型属于此种方式。   
同步非阻塞：   
在此种方式下，用户进程发起一个IO操作以后边可返回做其它事情，但是用户进程需要时不时的询问IO操作是否就绪，这就要求用户进程不停的去询问，从而引入不必要的CPU资源浪费。其中目前JAVA的NIO就属于同步非阻塞IO。   
异步：   
此种方式下是指应用发起一个IO操作以后，不等待内核IO操作的完成，等内核完成IO操作以后会通知应用程序。”   
这段话比较清楚****

1. 实操一遍NIO的IO操作过程。

代码如下：

<lesson-one-Java IO演进之路>