系统I/O 可分为**阻塞型， 非阻塞同步型、阻塞异步型以及非阻塞异步型。**

阻塞型I/O意味着控制权只到调用操作结束了才会回到调用者手里。

同步非阻塞是**会立即返回控制权给调用者的**。调用者不需要等等，它从调用的函数获取两种结果：

**要么此次调用成功进行了;**

**要么系统返回错误标识告诉调用者当前资源不可用，你再等等或者再试度看吧。**

比如read()操作， 如果当前socket无数据可读，则立即返回EWOULBLOCK/EAGAIN，告诉调用read()者”数据还没准备好，你稍后再试”。

异步非阻塞调用中，稍有不同。调用函数在立即返回时，还告诉调用者，这次请求已经开始了。**系统会使用另外的资源或者线程来完成这次调用操作，并在完成的时候知会调用者（比如通过回调函数）。**POSIX的aio\_read()来说，调用它之后，函数立即返回，操作系统在后台同时开始读操作。**即是将工作交给了内核去完成这个操作**。

在以上三种IO形式中，**非阻塞异步**是性能最高、伸缩性最好的。

两种IO多路复用方案:**Reactor and Proactor（基于同步和基于异步）**

一般情况下，I/O 复用机制需要**事件分享器**(event demultiplexor)。 事件分享器的作用，即**将那些读写事件源分发给各读写事件的处理者，**就像送快递的在楼下喊: 谁的什么东西送了， 快来拿吧。开发人员在开始的时候需要在分享器那里注册感兴趣的事件，并提供相应的处理者(event handlers)，或者是回调函数; 事件分享器在适当的时候会将请求的事件分发给这些handler或者回调函数。

涉及到事件分享器的两种模式称为：Reactor and Proactor。 **Reactor模式是基于同步I/O的，而Proactor模式基于异步I/O相关的。**

在Reactor模式中，**事件分离者等待某个事件或者可应用或个操作的状态发生**（比如文件描述符可读写，或者是socket可读写），**等待后，**分离者就把这个事件传给事先注册的事件处理函数或者回调函数，由后者来做实际的读写操作。

而在Proactor模式中，事件处理者(或者代由事件分离者发起)**直接发起一个异步读写操作(相当于请求)，而实际的工作是由操作系统来完成的。**发起时，需要提供的参数包括用于存放读到数据的缓存区，读的数据大小，或者用于存放外发数据的缓存区，以及这个请求完后的回调函数等信息。事件分离者得知了这个请求，它默默等待这个请求的完成，然后转发完成事件给相应的事件处理者或者回调。

**这种异步模式的典型实现是基于操作系统底层异步API的，所以我们可称之为“系统级别”的或者“真正意义上”的异步，因为具体的读写是由操作系统代劳的。**

举另外个例子来更好地理解Reactor与Proactor两种模式的区别。这里我们只关注read操作，因为write操作也是差不多的。下面是Reactor的做法：

1.某个事件处理者宣称它对某个socket上的读事件很感兴趣;

2.事件分离者等着这个事件的发生;

3.当事件发生了，事件分离器被唤醒，这负责通知先前那个事件处理者;

4.事件处理者收到消息，于是去那个socket上读数据了。

5.如果需要，它再次宣称对这个socket上的读事件感兴趣，一直重复上面的步骤;

下面再来看看真正意义的异步模式Proactor是如何做的：

1.事件处理者直接投递发一个写操作(当然，操作系统必须支持这个异步操作)。

2.这个时候，事件处理者根本不关心读事件，它只管发这么个请求，它魂牵梦萦的是这个写操作的完成事件。

3.这个处理者很拽，发个命令就不管具体的事情了，只等着别人（系统）帮他搞定的时候给他回个话。

4.事件分离者等着这个读事件的完成(比较下与Reactor的不同);

5.当事件分离者默默等待完成事情到来的同时，操作系统已经在一边开始干活了，它从目标读取数据，放入用户提供的缓存区中，最后通知事件分离者，这个事情我搞完了;

6.事件分离者通知之前的事件处理者: 你吩咐的事情搞定了;

7.事件处理者这时会发现想要读的数据已经乖乖地放在他提供的缓存区中，想怎么处理都行了。

8.如果有需要，事件处理者还像之前一样发起另外一个写操作，和上面的几个步骤一样。

在没有底层异步I/O API支持的操作系统，这种方法可以帮我们隐藏掉socket接口的差异(无论是性能还是其它)， 提供一个完全可用的统一“异步接口”。这样我们就可以开发真正平台独立的通用接口了。

那么，综上所述，这两者的区别是什么呢？

简单直观的理解：

1、Reactor模式是等待关心的动作的发生后，将如何处理这个动作的后续交给了用户态的应用本身来处理，Reactor的事件分离器**只关心事件的发生**，其它的就完全交给应用程序来处理了；而Proactor模式则是只关心由操作系统(内核)完成异步非阻塞的操作后**返回的结果**；

2、Proactor场景中只能够使用异步非阻塞的syscall(系统调用)，而Reactor的场景中更多地是使用同步非阻塞的syscall(系统调用)；

事件分享器的两种模式称为：Reactor and Proactor . Reactor模式是基于同步I/O的，而Proactor模式是和异步I/O相关的。

Reactor ：

1. **应用启动，将关注的事件handle注册到Reactor中；**
2. **调用Reactor，进入无限事件循环，等待注册的事件到来；**
3. **事件到来，select返回，Reactor将事件分发到之前注册的回调函数中处理；**

Proactor ：

1. **应用程序启动，调用异步操作处理器提供的异步操作接口函数，调用之后应用程序和异步操作处理就独立运行；应用程序可以调用新的异步操作，而其它操作可以并发进行；**
2. **应用程序启动Proactor主动器，进行无限的事件循环，等待完成事件到来；**
3. **异步操作处理器执行异步操作，完成后将结果放入到完成事件队列；**
4. **主动器从完成事件队列中取出结果，分发到相应的完成事件回调函数处理逻辑中；**

**1.proactor是处理器发起异步处理事件，直接返回。**

**2.多路复用器等待事件完成后通知处理器，然后处理器处理缓冲区中的数据。**

**redis和nio都用的是多路复用器，多路复用器的思想就是reactor(同步非阻塞)**