1.select==》时间复杂度O(n)
它仅仅知道了，有I/O事件发生了，却并不知道是哪几个流(可能有一个，多个，甚至全部)，我们只能去差别轮询所有流，找出能读出数据，或者写入数据的流，对他们进行操作。所以select具有O(n)的无差别轮询复杂度，同时处理的流越多，无差别轮询时间就越长。
2.poll==>时间复杂度O(n)
poll本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询到每个fd（文件描述符）的设备状态，但是它没有最大连接数的限制，原因是它是基于链表来存储的。
3.epoll==>时间复杂度O(1)
epoll可以理解为enevt poll，不同于盲轮询和无差别轮询，epoll会把哪个流发生了怎样的I/O事件通知我们。所以我们说epoll实际上是事件驱动（每个事件关联上fd）的，此时我们对这些流的操作都是有意义的。复杂度降低到了O(1)

select、poll、epoll都是IO多路复用的机制。I/O多路复用就通过一种机制，可以监视多个描述符，一旦某个描述符就绪(一般都是读就绪或者写就绪)，就能够通知程序进行相应的读写操作。但select、poll、epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责读写，也就是说这个读写过程是阻塞的，而异步I/O则无需自己负责进行读写，异步I/O会负责把数据从内核拷贝到用户空间。

select:
select本质上是通过设置或者检查存放fd标志位的数据结构来进行下一步处理。
1.单个进程可监视的fd数量被限制，即能监听端口的大小有限。
一般来说这个数目和系统的内存关系很大，具体数目可以cat /proc/sys/fs/file-max查看。
2.socket进行扫描时是线性扫描，即采用轮询的方法，效率较低
当套接字较多的时候，每次select()都要通过遍历FD_SETSIZE个Socket来完成调度，不管Socket是不是活跃的，都遍历一遍。这会浪费很多CPU事件。如果能给套接字注册某个回调函数，当他们活跃时，自动完成相关操作，那就避免了轮询，这正是epoll与kqueue做的
3.需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大。

poll：
poll本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态，如果设备就绪则在设备等待队列中加入一项并继续遍历，如果遍历完所有的fd

后没有发现就绪设备，则挂起当前进程，直到设备就绪或者主动超时，被唤醒后它又要再次遍历fd。这个过程经历了多次无谓的遍历。

它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：
1.大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。
2.poll还有一个特点就是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。