当使用select或epoll IO复用模型时，对于监听套接字，只需要关注读端即可，不需要关注写端。对于已连接套接字，两个方向可能都需要关注。有一点必须清楚，在IO复用的情况下，对于单个套接字描述符，仍然可以设置阻塞和非阻塞两种模式，默认是阻塞模式，下面分别针对read和write来分析这两种模式：

* **Read**：对于套接字的读操作（包括监听套接字和已连接套接字），设置为阻塞或非阻塞模式的区别在哪呢？

1. **阻塞模式**：那么如果有数据到达，select或epoll会返回IO可读，此时业务调用read可以成功返回。但是这里一定不能设置为循环的读，因为**循环的读**可能会导致进程阻塞在read调用。从而没法调用select或epoll\_wait获取其他文件描述符的状态更新。因为只能调用一次read，因此如果size设置的过小可能会导致数据没有读取完全。
2. **非阻塞模式**：数据到达时，返回IO可读，此时设置循环读，直到返回-1，并且errno为EAGAIN或EWOULDBLOCK时，退出循环，再次调用IO复用监听。因此，可以确保每次都把接收缓存中的数据全部读取完。

* **Write**：对于套接字的写操作，阻塞模式和非阻塞模式的差异较大。

1. **阻塞模式**：如果发送小数据量（即一次write操作即可发送完），那无所谓。但是如果我们的应用需要**发送大数据量**，这意味着无法通过一次write操作发送完所有数据，也就是说**一次write操作后，再次调用write操作可能会阻塞**。这种情况下，必须将write设置为非阻塞模式。
2. **非阻塞模式**：当发送大数据量时，必须设置为非阻塞模式，并且业务循环的去write，直到返回-1，并且errno为EAGAIN或EWOULDBLOCK‘，说明发送缓存满了，此时退出循环，重新调用IO复用并且监听文件描述符的写。一旦所有数据发送完成，再从监听套接字中去掉该写文件描述符。这是因为如果我们监听了写描述符但是又没有数据可写时，那么select或epoll\_wait每次都会返回IO可写，这将是极大的浪费。

**epoll的LT和ET模式**

epoll默认使用LT，上面的描述即是针对select和epoll的LT模式，下面介绍**epoll ET模式**下的read和write：

* **Read**：每次IO可读时，必须通过while循环把缓存中的所有数据都读取完，因为epoll\_wait只在数据到达时返回一次。
* **Write**：由于只有在状态变更时epoll才会递交一次事件，因此不用担心epoll\_wait会频繁的返回。所以这里可以静态的监听写套接字，而不用频繁的打开和关闭监听。

**epoll的缺点**：从性能角度讲，epoll存在一个设计上的缺陷；它不能在单次系统调用中**更新多个文件描述符。**当你的兴趣集中有100个文件描述符需要更新状态时，你不得不调用100次epoll\_ctl()函数。