**什么是IO多路复用？**

指一旦进程指定的一个或多个IO条件准备就绪(完成”磁盘”→”内核缓冲区”)，内核就通知进程去读取。

同步阻塞IO：在等待就绪，花去太多时间。

同步非阻塞IO：不阻塞进程，但轮询来判断就绪也耗费CPU资源。

多路IO复用：对大量FD进行就绪检查的高性能方案。

select缺点

1. 单个进程能够监视的文件描述符的数量默认最大限制1024
2. 需复制大量的句柄数据结构，产生巨大开销
3. 返回的是整个句柄的列表，应用需要遍历才能发现哪些句柄发生了事件
4. 水平(非边缘)触发方式，应用若未完成对一个已就绪的FD进行IO操作，之后每次select调用还是会将这些FD通知给进程。

poll：

1. 类似select，仅没有FD的最大限制。

优化select式的方案：

1. kernel维护一个事件关注列表，仅修改该列表，无需将句柄数据结构复制到内核
2. 直接返回事件列表，而不是所有句柄列表

epoll：

1. epoll\_create创建kernel中的事件关注列表
2. epoll\_ctl 修改表
3. epoll\_wait 等待IO事件发生

epoll支持水平和边缘触发，理论上说边缘触发性能更高，但使用更复杂，因任意意外的事件丢失都会造成请求处理错误。Nginx使用了边缘触发。

只要句柄**满足某种状态**，水平触发就会发出通知；当句柄**状态改变时**，边缘触发才会发出通知。如一个socket经过长时间等待后接收到100k的数据，两种触发方式都会向程序发出就绪通知。假设程序从这个socket中读取了50k数据，并再次调用监听函数，水平触发依然会发出就绪通知，而边缘触发会因为socket“有数据可读”这个状态没有发生变化而不发出通知且陷入长时间的等待。

在使用边缘触发的api时，要注意每次都要读到 socket返回 EWOULDBLOCK为止。

<http://www.cnblogs.com/Anker/archive/2013/08/17/3263780.html>

<http://www.cnblogs.com/Anker/archive/2013/08/14/3258674.html>

<http://blog.csdn.net/russell_tao/article/details/7204260> 【重要】