网络io：

建立tcp连接不需要应用层参与。

具体如何解析和发送需要应用层协议设计

六步

socket

bind

listen

accept

recv

send

bug：

为什么出现time-wait ： 服务器主动断开（最好让客户端断开）

为什么出现close-wait：

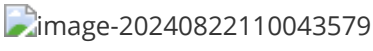
io多路复用

如果一请求一线程：

缺点：耗内存。性能也不高

单线程利用select管理多个io：

select（）：相当于一个秘书，实时监控客户端是否有请求发来，有数据则对应连接的io变为可读，select检测哪些连接可读



// 五个参数，分别是最大fd，读集合，写集合，错误集合，超时时间
// maxfd用于内部循环探寻是否可读

//timeout是多长时间轮询一次

//无论是否有可读连接都会有返回

问题：

fd_set（一个位图）能处理的最大请求是多少个：

1024/（8*sizeof（long）） = 32个字节，最多支持256个描述符

可以修改超过1024，需要去内核修改宏定义（大于1024就别用select了）

缺点：参数较多，不易于维护；

每次调用select需要把rset放进内核再带出，每次都要copy，中间产生很多无效拷贝（比如有1百万个连接，每次只有100个可读，但是每次都要要拷贝**一百万**个），对性能影响

select对io数量有限制；

性能缺陷：

1.copy

2.每次需要遍历io集合，而返回就绪集合

3.每次都需要循环遍历所有连接

poll：

改进select：只带一个参数，易于维护

epoll：

在大量的网络io上首选epoll

三个api：

不限连接数（底层链表）

epoll_ctl（）底层是**红黑树**，增加的时候需要**k**和**v**删除只需要**v**

两种触发：

水平触发LT：有数据就一直触发

场景：解决tcp粘包问题（两拨数据发送间隔短，epoll直接连起来接受了）

边缘触发ET

场景：

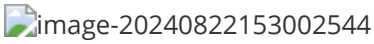
问题：

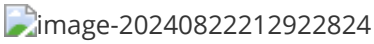
epoll有没有nmap?

epoll可以使用多线程吗？是否线程安全?

Reactor：

事件驱动

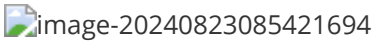




第一步：把各种accept， recv， send封装，并且将触发条件调成**事件**触发，简洁不少

第二步：封装buffer，分成wbuffer，rbuffer，将wbuufer使用好封装成httpresponce可以变成简单的webserver

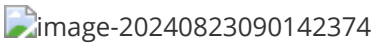
（**wenserver其他部分都是不变的，就是一直在request和response之间不断加内容**）



封装epoll，connfd，使得只需要初始化就可以，不需要知道底层如何实现，用什么都行（select，poll， epoll）；

如何测试reactor性能：

wrk（github）download下来make编译，然后同时启动自己的服务器和nginx对比性能



Reactor对于应用层web开发的意义：

使得我们只需要关注wbuffer，wlen， rbuffer，rlen，别的都不需要改变！