五种IO模型：

1.阻塞IO

2.非阻塞IO

3.信号驱动IO

4.多路转接IO

5.异步IO

阻塞：为了完成功能发起调用，但是如果当前不具备完成条件，则等待

非阻塞：为了完成功能发起调用，但是如果当前不具备完成条件，则报错返回

阻塞与非阻塞最直观的区别：不具备完成条件时发起一个调用后是否会立即返回。

同步：为了完成功能发起调用，但是如果当前不具备完成条件，则等待，直到完成功能

异步：为了完成功能发起调用，但是如果当前不具备完成条件，则立即返回(将完成功能交给操作系统，当操作系统完成功能则通过一些其他方式(信号通知)告诉我们功能完成)；

同步与异步的区别：要完成某个功能，在条件不具备的情况下，是否会阻塞完成

钓鱼例子：

异步IO：Linux下的AIO(大量磁盘读写的时候)

多路转接：

让别人来提我们监控整个等待的过程，看现在有哪一个就绪好了，直接我弄成相应的操作

让多路转接模型替用户完成监控多个描述符的等待过程，如果哪一个描述符就绪(可写/可读),则完成监控过程，并且通知我我们有哪些描述符就绪，接下来用户直接读取数据即可。

非阻塞IO设置：

fcntl F_SETFL F_GETFL

多路转接模型(多路复用模型)：

select

int select(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);

select对描述符进行监控，分了三种监控，分别是可写事件，可读事件，异常事件，如果想要对描述符进行某个事件的监控，则需要将描述符添加到指定的集合中；

这个集合实际是一个位图，位图的大小默认是1024，添加指定描述符到集合中，就是置描述符数据所对应的比特位为1

FD_SETSIZE = 1024

fd_set这个结构体实际是一个位图

long int fd_bits[1024/sizeof(long int)]

select建立监控集合，对描述符监控指定事件(可写，可读，异常)，当监控集合中的某个描述符就绪了，则返回，并且告诉我们有几个描述符就绪了，将监控集合中没有就绪的描述符从几何中移除

就绪：缓冲区有一个低水位标记，可读就是当接收缓冲区中的数据大小达到低水位标记大小，则表述可读；可写就是当发送缓冲区剩余空间大小，大于低水位标记的时候则可写

在tcp服务端程序中，因为我们不知道何时该接收数据，或者何时accept获取连接，因此在没有数据到来/没有连接请求到来的时候调用recv/accept就会阻塞，导致服务端程序要不然只能连续处理一个客户端的数据，要不然就每个客户端只能处理一次

假如我们知道客户端数据/连接请求什么时候来，我们在合适的时候调用recv/accept则不会造成阻塞，并且可以实现谁的数据到来，处理谁的数据，达到高并发服务端程序的编写目的

使用C++封装一个select类，实现方便的使用

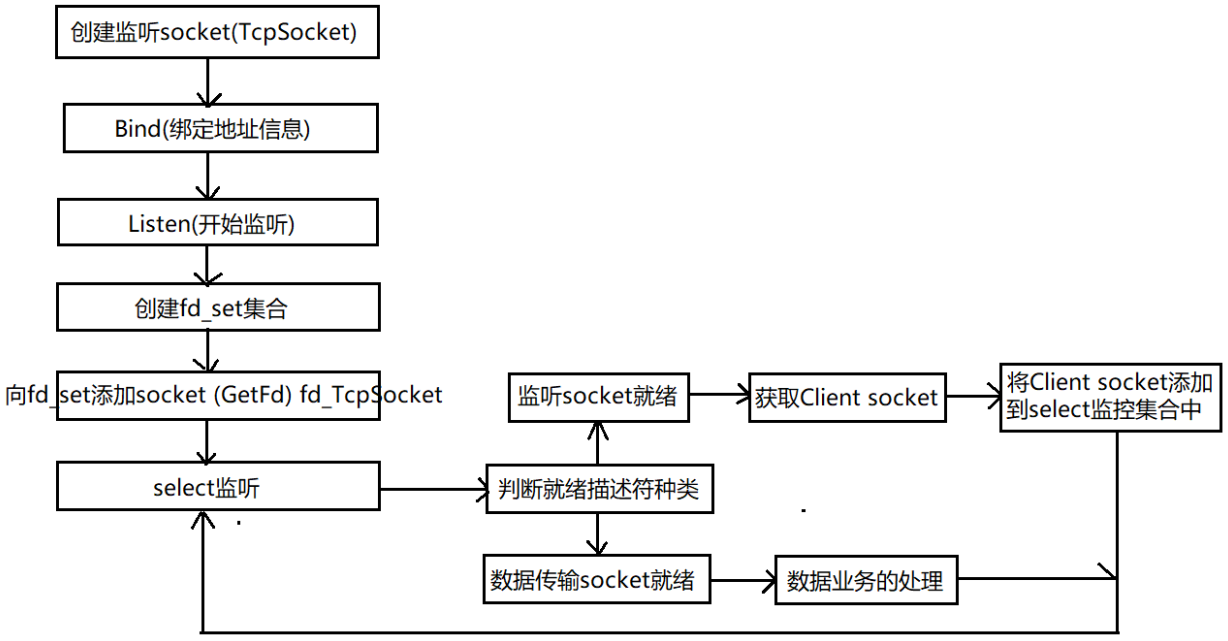
```
class Select{
    private:
        int_max_fd;                 // 最大的描述符
        fd_set_rfds;                // 为了避免每次重新获取描述符
        std::unordered_map<int ,TcpSocket> _map;
        // 记录描述符与TcpSocket的字典
    public:
        bool Addk(TcpSocket &sock);
        bool Del(TcpSocket &sock);
        bool Wait(std::vector<TcpSocket> *list);
```

poll(选学--被淘汰了)

epoll

多路转接：

功能：对大量描述符进行事件阻塞监控，当描述符状态发生改变，则返回



select：

功能：同时对大量描述符进行事件阻塞监控(可读、可写、异常)，状态改变时返回

1.select创建三个描述符集合，分别监控可读事件、可写事件、异常事件

集合实际是一个位图，添加描述符就是修改位图对应比特位，位图大小取决于FD_SETSIZE 默认1024

2.向这三个集合中添加描述符，对描述符关注什么状态九天假到指定的集合中

3.将集合中的数据拷贝到内核，进行监控(间隔时间，轮询遍历，判断是否有描述符就绪）

就绪(描述符对应缓冲区中数据大小/空闲空间大小是否大于低水位标记)

如果遍历没有描述符就绪，则继续休眠等待/判断是否超时，超时则返回0

如果有描述符就绪，将集合中没有就绪的描述符全部从集合中移除

(集合中保留的都是就绪描述符)

(因为对集合进行了修改，因此需要每次清空集合，向集合中重新添加描述符)

4.遍历从0-max_fd的描述符，判断是否在集合中(目的：为了找到具体是那个描述符就绪)

5.对就绪描述符进行操作

select优缺点：

缺点：

1.能够监控的描述符有最大上限(因为位图最大取决于FD_SETSIZE)

2.因为select判断集合中描述符就绪后会修改集合内容，因此需要每次重新添加描述符(编程麻烦，效率低)

3.因为select不会告诉我们具体那个描述符就绪，因此需要用户进行遍历集合判断(编程麻烦，并且随着描述符增多效率降低)

4.因为select每次都需要将集合数据拷贝到内核，并且在内核是轮询遍历实现监控，因此(性能随着描述符增多而降低)

优点：

1.跨平台

2.超时时间的控制比较精细

poll原理：

相较于select优点：

1.描述符无上限

2.监控集合只有一个，每个节点可以关注不同事件，不用针对不同的事件进行多次遍历

相较于select缺点：

不能跨平台

缺点：

除了描述符无上限，其他雷同select

epoll模型：

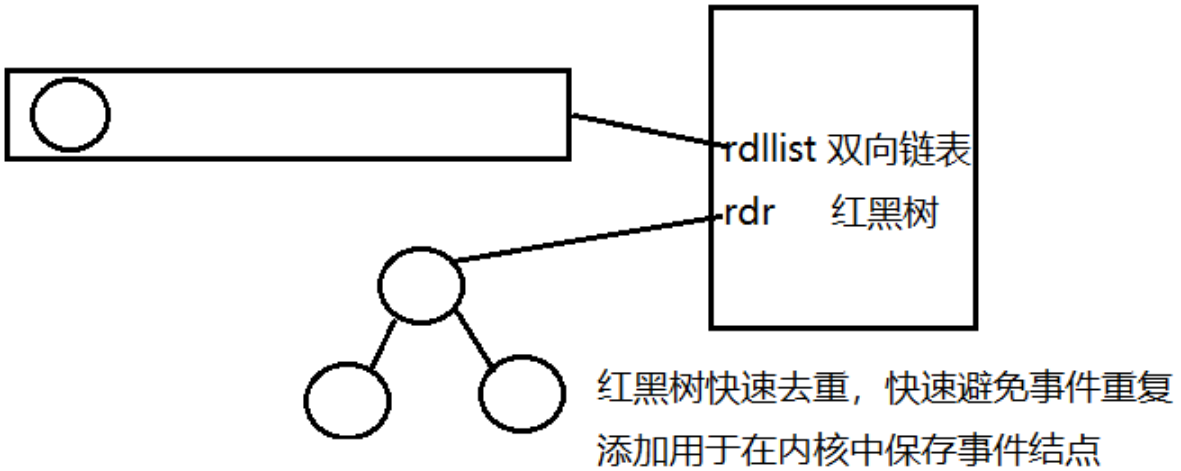
epoll是Linux下使用度最高，性能最高的多路转接模型

```
struct epoll_event{
    union{
        int fd;
        void *ptr;
    }
    ushort events
};
```

epoll_create　　创建epoll

epoll_ctl　　epoll控制(事件的添加/移除/监控)

epoll_wait　　开始监控



epoll采用事件回调机制来实现监控

ep_poll_callback()

指针，向rdlist双向链表中添加一份；

优点:

1. epoll每隔一段时间，只需要来看一下双向链表是否为空就可以快速判断是否有描述符就绪
2. 一旦判断有描述符就绪，那么直接将双向链表中的结点中的epoll_event结构拷贝一份；意味着用户拿到的结点都是就绪的结点，可以直接处理

红黑树快速去重，快速避免事件重复添加用于在内核中保存事件结点

epoll采用事件回调机制来实现监控

ep_poll_callback

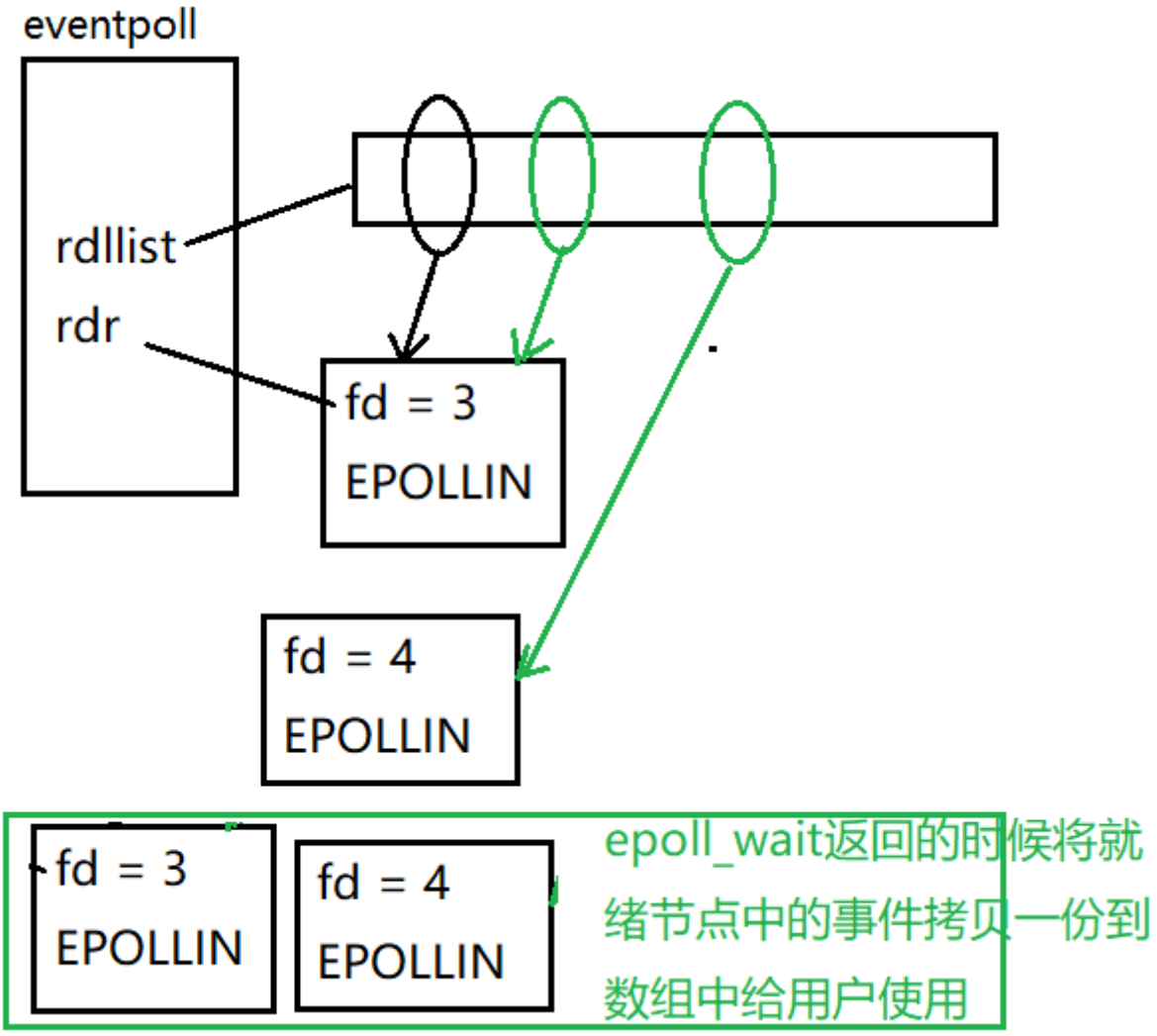
指针，向rdlist双向链表中添加一份；

优点：

1. epoll每隔一段时间，只需要来看一下双向链表是否为空就可以快速判断是否有描述符就绪

2. 一旦判断有描述符就绪，那么直接将双向链表中的结点中的epoll_event结构拷贝一份；意味着用户拿到的结点都是就绪的结

点，可以直接处理




```
class Epoll{
    private:
        int _epfd;
        std::unordered_map<TcpSocket> _map;
    public:
        bool Create();
        bool Add(TcpSocket sock, uint32_t events = EPOLLIN);
        bool Del(TcpSocket sock);
```

```
    bool Wait(std::vector<TcpSocket> *list);
};
```