五种IO模型：

阻塞IO、非阻塞IO、信号驱动IO、异步IO、多路转接IO

阻塞/非阻塞：

同步/异步：

多路转接IO：

非阻塞IO的设置：fcntl F_GETFL F_SETFL O_NONBLOCK

多路转接模型select：

select(max_fd,read,write,except,time)

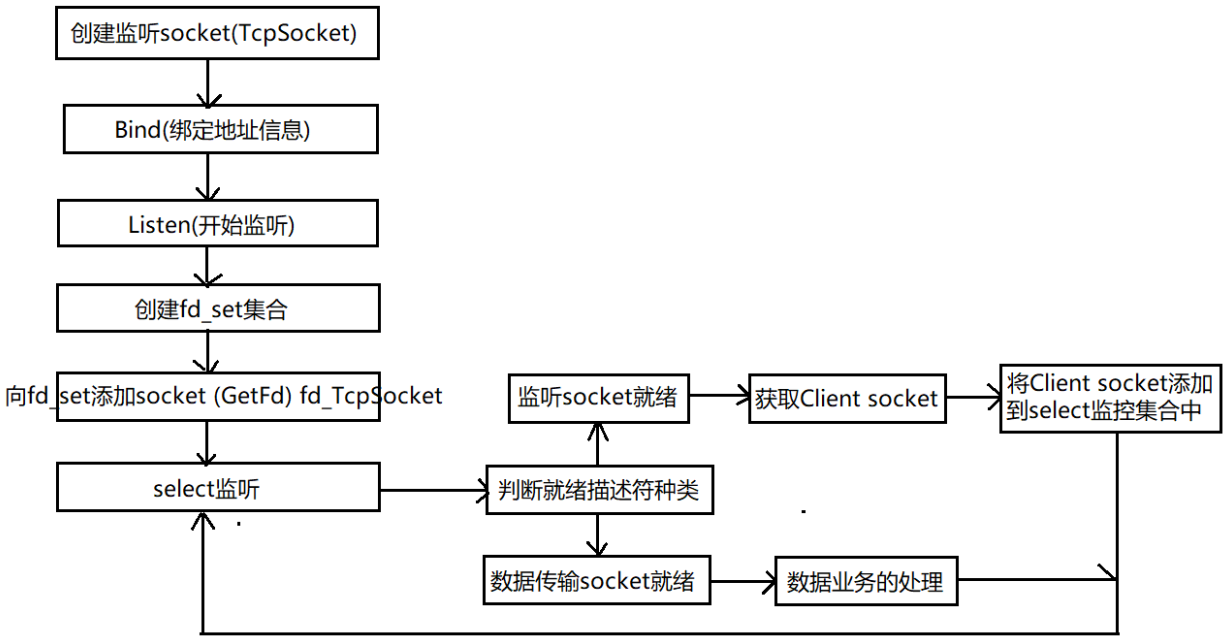
select内部监控的过程就是，每过一段时间就把这些描述符遍历一次，看现在哪一个描述符就绪，也有可能同时有多个描述符就绪，当遍历到就绪的时候他就返回，但是这个集合的最大大小为1024，也就是最大可以添加1024个描述符，select的内部实现就是每隔一段时间把描述符遍历一遍，也就需要从0-1024进行遍历，这样的话遍历就太慢了，为了提高效率，select的第一个参数就是最大的描述符max_fd，代表从0开始遍历，遍历到最大的这个描述符就可以了，最后一个参数time:超时时间，指的是select本身是一个阻塞操作，当监控当中的描述符没有发生状态改变的时候，他会在这里一直阻塞监控但是有些场景下一直监控阻塞并不是我们期待的场景，我们希望它超时之后给我们返回一下，告诉我们这些描述符超时了，我们把超时的描述符干掉，所以，在这里设置一个select的阻塞超时时间，这个超时时间是一个struct timeval

fd_set

select在tcp服务端中的应用：可以将tcp服务端程序提升为一个高并发的服务端程序

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：

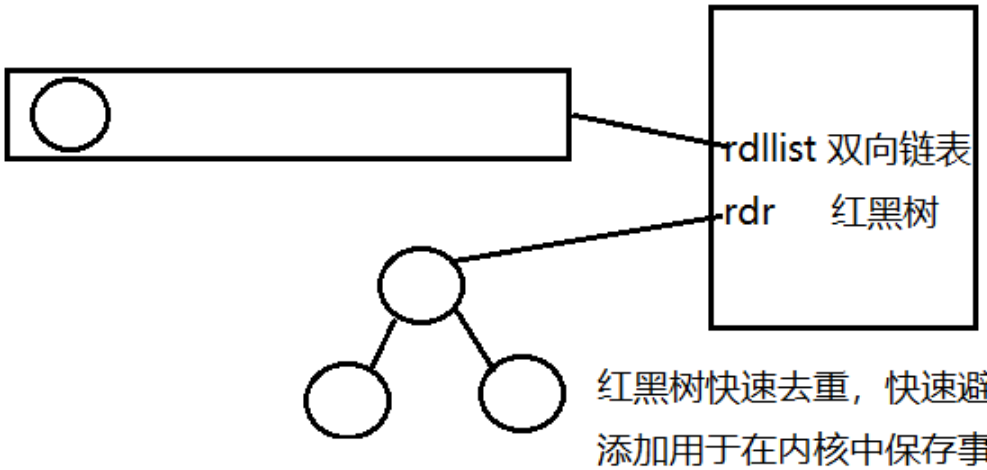
除了描述符无上限，其他雷同select

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



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

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：

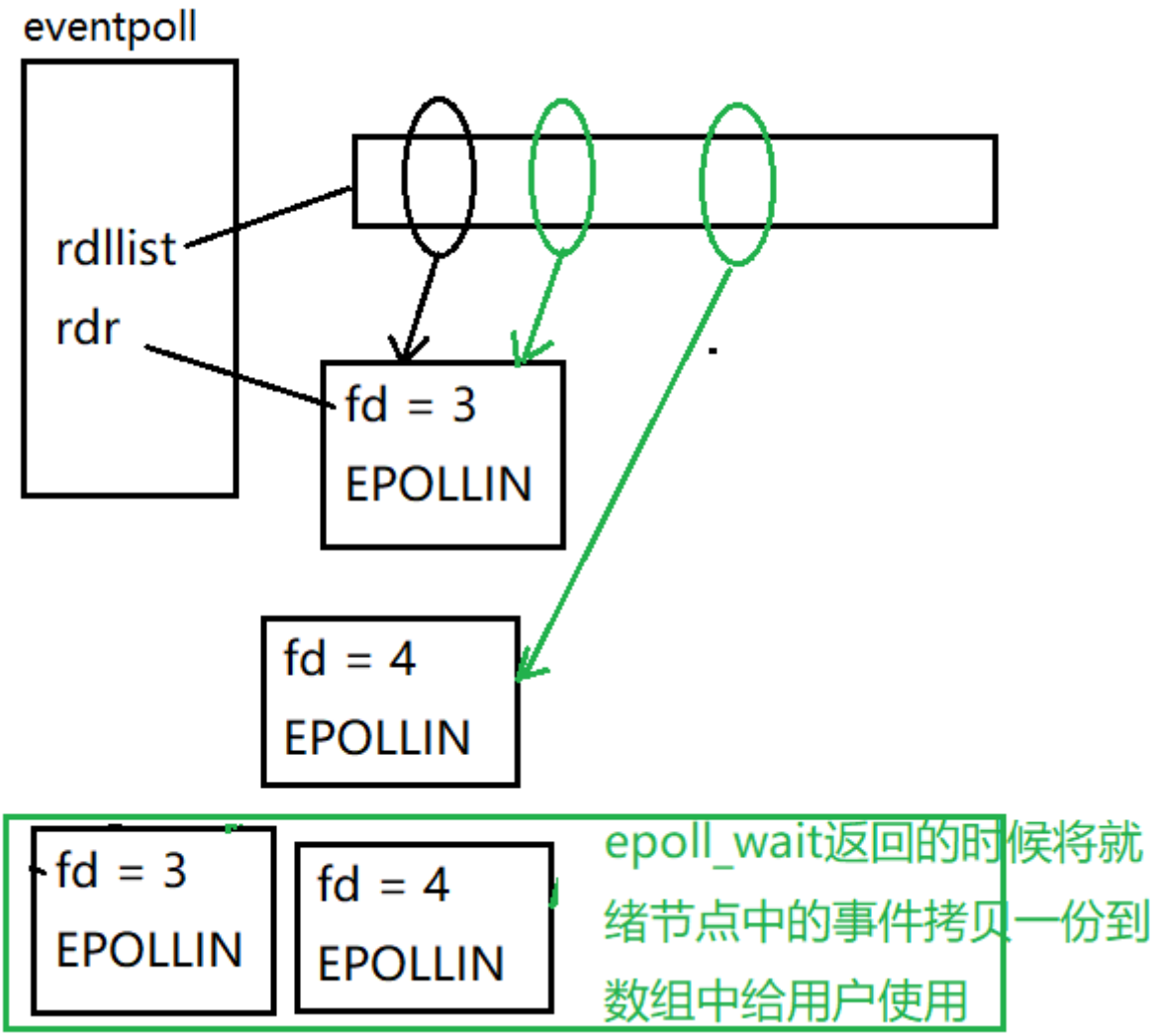
1. epoll每隔一段时间，只需要来看一下双向链表是否为空就可以快

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);
    };
```

select:

是什么：多路转接模型之一

为什么：对大量描述符进行状态改变监控；可以检测到在大量描述符中有哪些描述符都可读/可写/异常；这样我们就可以实现当描述符状态改变时，再去操作描述符，避免了无谓的阻塞；实现了服务端程序的并发操作。

优缺点：

缺点：1.所能监控的描述符数量有上限，为1024

2.每次都要将描述符拷贝到内核

3.每次使用监控都需要重新添加描述符到集合(因为就绪会剔除未就绪描述符)

4.内核实现监控是轮询遍历，性能随着描述符增多而下降

5.因为不会告诉用户具体就绪的描述符，因此需要用户遍历就绪集合，性能随着描述符增多而下降

优点：跨平台；阻塞超时时间控制可以精细到微妙

poll模型：

相对于select优点：

1.描述符无上限

2.将事件集合融合为一个事件数组

相对于select缺点：雷同

缺点：不能跨平台

epoll模型：

epoll模型是Linux下多路转接模型中性能最高的，并且不会随着描述符增多而性能降低。

工作流程：

1.创建epoll；在内核中创建epollerent结构(rdllist--就绪事件双

向链表；rdr--红黑树事件集合)

2.向事件集合中添加事件结点，(每个事件只需要向内核添加一次即

可)

3.epoll在内核使用信号驱动事件回调方式实现监控(当内核检测到

集合中的描述符就绪，则直接调用回调函数，将就绪事件结点指

针，添加到就绪事件链表中；epoll每隔一段时间就看一下双向链表

是否为控股，来判断是否有事件就绪，如果有的话，则将就绪事件

拷贝到用户态供用户操作)

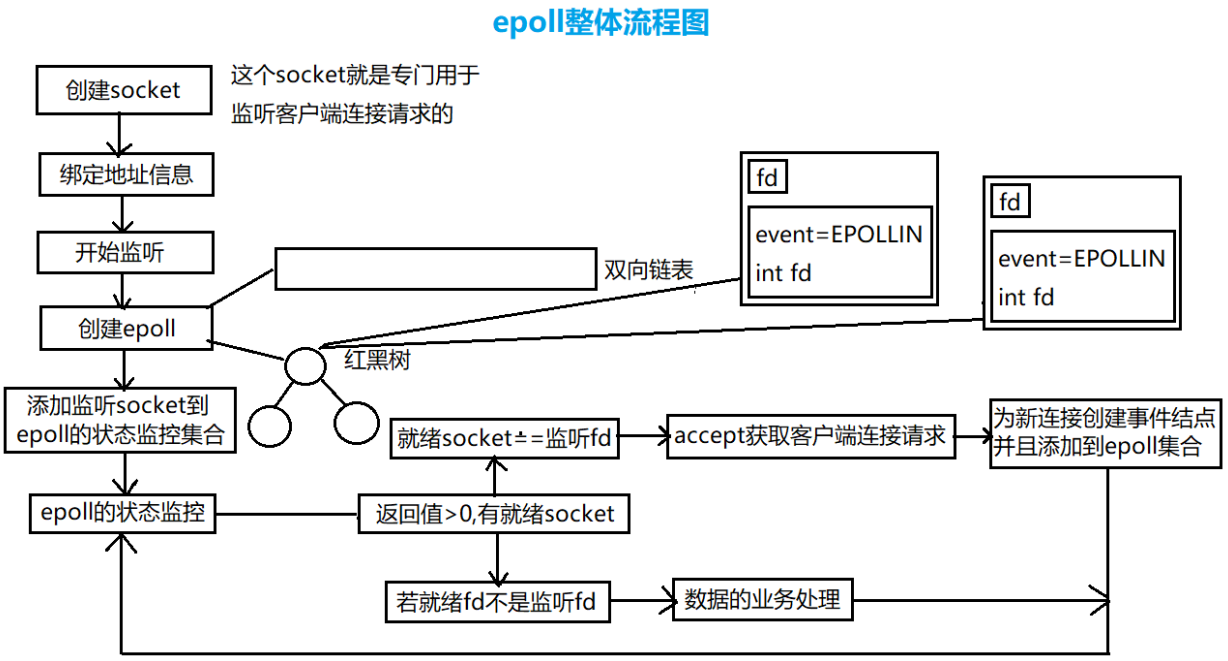
4.开始监控后，用户直接拿到的就是就绪的事件，直接进行相应的

操作；

```
:1s/select/epoll/g        第一行select全部替换为epoll
:1,2s/select/epoll/g      第一行到第二行select全部替换为epoll
:1,3s/select/epoll/gc     第一行到第三行select全部替换为epoll，替换之前询问是否替换
:%s/select/epoll/gc       全文替换

sudo netstat -anptu |grep 9000
```

超时等待



epoll优缺点：

优点：

1. 描述符无上限

2. 每个事件只需要向内核拷贝一次

3. epoll在内核使用事件回调将就绪事件添加到双向链表，每隔一会判

断双向链表是否为空来判断是否有描述符就绪，性能不会随着描述符增

多而降低

4. epoll直接将就绪的事件拷贝给用户，用户直接拿到就绪事件即可操

作，不用哦那个所有事件中找就绪，性能也相对select有所提高

缺点：

1. 相较于select无法跨平台

多路转接的适用场景：

应用于有大量连接，但是同一时间只有少量连接活跃的场景，因为多路转接是并发处理请求；

epoll的惊群问题；课后调研

边缘触发/水平触发

epoll的两种事件触发方式

边缘触发：每条新数据的到来，将就绪返回一次(并不管缓冲区数据是否大于低水位标记)

水平触发：只要缓冲区数据大于低水位标记，就会触发就绪事件

epoll有两种触发模式可选，默认是水平触发；而select只用水平触发模式

一旦epoll的触发模式被设置为边缘触发，将要要求用户每次都将缓冲区中的数据读完进行处理(因为不触发就绪，那么socket按照程序逻辑就不会多次处理)

一旦epoll的触发方式被设置为边缘触发，必须一次将数据读完，因为读不完数据不全，有可能无法处理，要一次读完的话，多读几次，但是多读几次有可能会阻塞。但是在并发模型中不能存在阻塞情况，因此要将描述符设置为非阻塞状态！！

epoll：多路转接模型之一；并且是Linux下性能最高的多路转接模型

epoll流程：

epoll优缺点：

边缘触发/水平触发：

epoll惊群：