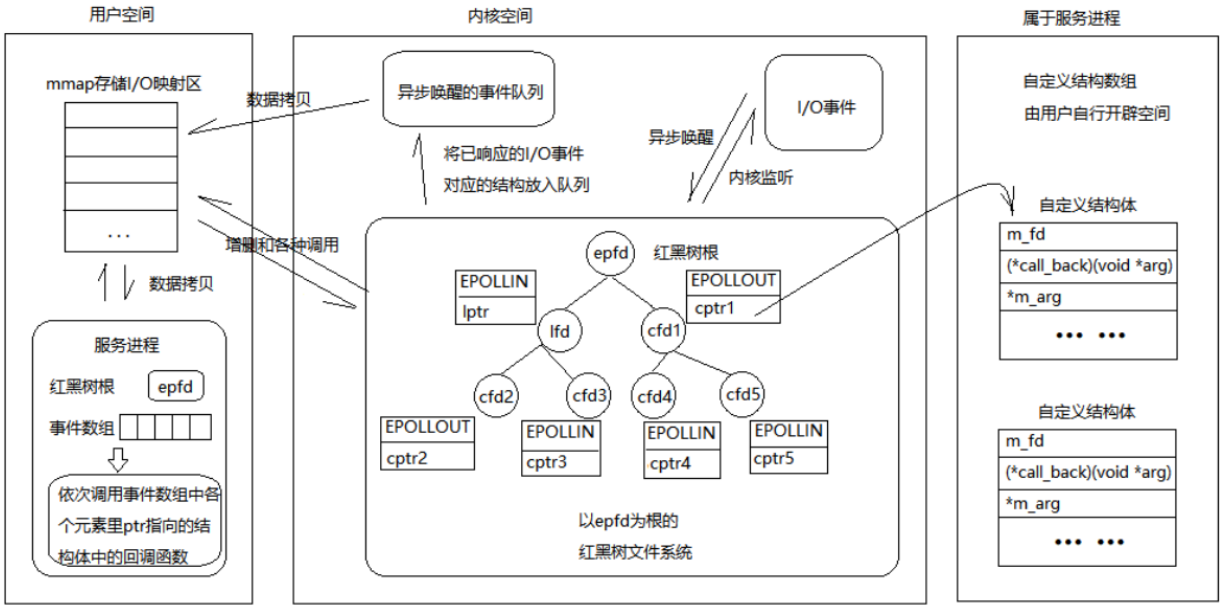
知识点1【epoll反应堆的概述】（了解）

epoll反应堆的核心思想：将文件描述符、事件、回调函数用自定义结构体封装在一起，当某个文件描述符的事件被触发，调用其回调函数即可



知识点2【epoll反应堆的实现tcp_echo_并发服务器】（了解）

```c
#include <stdio.h>
#include <sys/socket.h> //socket
#include <unistd.h>     //_exit
#include <netinet/in.h> //struct sockaddr_in
#include <string.h>     //bzero
#include <sys/time.h>
#include <sys/types.h> //select
#include <arpa/inet.h> //inet_ntop
#include <sys/epoll.h> //epoll
 #include <stdlib.h>
 //定义回调函数类型
 //第一个参数：文件描述符对应的自定义空间地址
 typedef void (*CALLBACK)(void *);

 //自定义结构体类型
 typedef struct
 {
     int epfd;          //树的根
```

```
19      int fd;             //存放文件描述符
20      int event;          //存放事件
21      CALLBACK callback; //回调函数
22  } MY_EVENT;
23  int create_tcp_socket(unsigned short port)
24  {
25      //创建tcp监听套接字
26      int lfd = socket(AF_INET, SOCK_STREAM, 0);
27      if (lfd < 0)
28      {
29          perror("socket");
30          _exit(-1);
31      }
32
33      //设置端口复用
34      int yes = 1;
35      setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));
36
37      //bind给lfd绑定固定的ip port
38      struct sockaddr_in my_addr;
39      bzero(&my_addr, sizeof(my_addr));
40      my_addr.sin_family = AF_INET;
41      my_addr.sin_port = htons(port);
42      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
43      int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
44      if (ret < 0)
45      {
46          perror("bind");
47          _exit(-1);
48      }
49
50      //listen进行监听
51      listen(lfd, 128);
52
53      return lfd;
54  }
55  void event_add(int epfd, int fd, int event, CALLBACK callback, MY_EVENT
y_ev_p)
56  {
57      //对自定义的空间成员 赋值
58      my_ev_p->epfd = epfd;
```

```
    my_ev_p->fd = fd;
    my_ev_p->event = event;
    my_ev_p->callback = callback;

    //定义需要上树的节点
    struct epoll_event ev;
    ev.events = event;
    ev.data.ptr = my_ev_p;

    //上树
    epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &ev);
}
void read_data(void *arg)
{
    MY_EVENT *my_ev_p = (MY_EVENT *)arg;

    unsigned char buf[1024] = "";
    int len = recv(my_ev_p->fd, buf, sizeof(buf), 0);
    if (len <= 0) //客户端退出
    {
        //下树
        struct epoll_event ev;
        ev.events = my_ev_p->event;
        ev.data.ptr = NULL;
        epoll_ctl(my_ev_p->epfd, EPOLL_CTL_DEL, my_ev_p->fd, &ev);

        printf("%d已退出\n", my_ev_p->fd);
        //关闭文件描述符
        close(my_ev_p->fd);

        //释放自定义类型空间
        if (my_ev_p != NULL)
        {
            free(my_ev_p);
            my_ev_p = NULL;
        }
        }
    else
    {
        send(my_ev_p->fd, buf, len, 0);
```

```
99          printf("buf=%s\n", buf);
100     }
101 }
102 void initAccept(void *arg)
103 {
104     MY_EVENT *my_ev_p = (MY_EVENT *)arg;
105     //提取与客户端通信的已连接套接字cfd
106     struct sockaddr_in cli_addr;
107     socklen_t cli_len = sizeof(cli_addr);
108     int cfd = accept(my_ev_p-
>fd, (struct sockaddr *)&cli_addr, &cli_len);
109 
110     //打印一下客户端的信息
111     char ip[16] = "";
112     printf("%s:%hu的连接到来\n",
113            inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip, 16),
114            ntohs(cli_addr.sin_port));
115 
116     MY_EVENT *cfd_p = (MY_EVENT *)calloc(1, sizeof(MY_EVENT));
117     //将cfd进行上树
118     event_add(my_ev_p->epfd, cfd, EPOLLIN, read_data, cfd_p);
119 }
120 int main(int argc, char const *argv[])
121 {
122     //创建监听套接字
123     int lfd = create_tcp_socket(8000);
124 
125     //创建一个epoll的树 并得到epoll句柄
126     int epfd = epoll_create(1);
127 
128     //上树（添加监听节点）
129     MY_EVENT *lfd_p = (MY_EVENT *)calloc(1, sizeof(MY_EVENT));
130     event_add(epfd, lfd, EPOLLIN, initAccept, lfd_p);
131 
132     //监听
133     struct epoll_event evs[1024];
134 
135     while (1)
136     {
137         int n = epoll_wait(epfd, evs, 1024, -1);
138         if (n <= 0)
```

```
        {
            perror("epoll_wait");
            _exit(-1);
        }
        else
        {
            int i = 0;
            for (i = 0; i < n; i++)
            {
                //从数组中取出 自定义空间的地址
                MY_EVENT *p = evs[i].data.ptr;
                if (evs[i].events == p->event)
                {
                    //执行回调函数
                    p->callback(p);
                }
            }
        }
    }

    close(lfd);
    return 0;
}

```