# 一、做为 TCP 服务器需要具备的条件呢？

对于 TCP 服务器编程流程：

1.socket()，在进行网络通信时先建立套接字

2.bind()，由于我们需要固定的套接字地址，所以需要绑定

3.listen() ，监听连接套接字请求

4.accept()，接受连接套接字请求

5.recv()，接收数据

6.send()，发生数据

7.close()，关闭套接字

int bind( int sockfd, const struct sockaddr \*myaddr，socklen\_t addrlen );  
功能：

将本地协议地址与 sockfd 绑定，这样 ip、port 就固定了

参数：

sockfd：socket 套接字

myaddr： 指向特定协议的地址结构指针

addrlen：该地址结构的长度

返回值：

成功：返回 0

失败：-1

注意：bind只能绑定自身的地址及端口

int listen(int sockfd, int backlog);  
功能：

将套接字由主动修改为被动，使操作系统为该套接字设置一个连接队列，用来记录所有连接到该套接字的连接。

参数：

sockfd： socket监听套接字

backlog：连接队列的长度

返回值：

成功：返回0

失败：其他

int accept(  int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen );  
功能：

从已连接队列中取出一个已经建立的连接，如果没有任何连接可用，则进入睡眠等待（阻塞）。

参数：

sockfd： socket监听套接字

cliaddr： 用于存放客户端套接字地址结构

addrlen：套接字地址结构体长度的地址

返回值：

成功：已连接套接字。注意：返回的是一个已连接套接字，这个套接字代表当前这个连接

失败：< 0

# 关闭连接：close()

使用 close() 函数即可关闭套接字，关闭一个代表已连接套接字将导致另一端接收到一个 0 长度的数据包

**做服务器时**

关闭监听套接字（ socket()和listen()之后的套接字 ）将导致服务器无法接收新的连接，但不会影响已经建立的连接；

关闭 accept()返回的已连接套接字将导致它所代表的连接被关闭，但不会影响服务器的监听（ socket()和listen()之后的套接字 ）。

**做客户端时**

关闭连接就是关闭连接，不意味着其他。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int main()

{

unsigned short port = 8000; // 本地端口

    //1.创建通信端点：套接字

    int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0)

    {

        perror("socket");

        exit(-1);

    }

    //设置本地地址结构体

    struct sockaddr\_in my\_addr;

    bzero(&my\_addr, sizeof(my\_addr));           // 清空,保证最后8字节为0

    my\_addr.sin\_family = AF\_INET;               // ipv4

    my\_addr.sin\_port   = htons(port);           // 端口

    my\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);// ip，INADDR\_ANY为通配地址其值为0

    //2.绑定：将本地ip、端口与套接字socket相关联起来

    int err\_log = bind(sockfd, (struct sockaddr\*)&my\_addr, sizeof(my\_addr));

    if( err\_log != 0)

    {

        perror("binding");

        close(sockfd);

        exit(-1);

    }

    //3.监听，监听套接字改为被动，创建连接队列

    err\_log = listen(sockfd, 10);

    if(err\_log != 0)

    {

        perror("listen");

        close(sockfd);

        exit(-1);

    }

printf("listen client @port=%d...\n",port);

    while(1)

    {

        //4.从完成连接队列中提取客户端连接

        struct sockaddr\_in client\_addr;             // 用于保存客户的地址

        socklen\_t cliaddr\_len = sizeof(client\_addr);

        int connfd = accept(sockfd, (struct sockaddr\*)&client\_addr, &cliaddr\_len);

        if(connfd < 0)

        {

            perror("accept");

            continue;

        }

        // 打印客户端地址

        char cli\_ip[INET\_ADDRSTRLEN] = "";

        inet\_ntop(AF\_INET, &client\_addr.sin\_addr, cli\_ip, INET\_ADDRSTRLEN);

        printf("----------------------------------------------\n");

        printf("client ip=%s,port=%d\n", cli\_ip,ntohs(client\_addr.sin\_port));

        // 接收数据

        char recv\_buf[512] = "";

        while(recv(connfd, recv\_buf, sizeof(recv\_buf), 0) > 0 )     // 当客户端断开连接时，recv返回值等于0

        {

            printf("\nrecv data:\n");

            printf("%s\n",recv\_buf);

            // 发送数据

            char send\_buf[512] = "i recv you data\0";

            send(connfd, send\_buf, strlen(send\_buf), 0);   // 向客户端发送信息

        }

        close(connfd);     //关闭已连接的客户端套接字

        printf("client closed!\n");

    }

    close(sockfd);         //关闭监听套接字

    return 0;

}