# 一、网络编程的专业术语

1、网络编程也是属于进程之间的通信，他和系统编程通信方式的区别？

系统编程：作用于同一部主机的两个不同进程之间的通信

网络编程：作用于同一部或跨主机之间的进程间通信

1. 网络编程的结构模型

OSI模型(7层)

应用层 老板说话

表示层 秘书将老板的话写成信

会话层 前台将信塞到信封，并附上对方的地址

传输层 邮差拿到信，决定以什么方式出发：TCP协议/UDP协议

网络层 邮差分析地址在哪：IP协议

数据链路层 邮差选择一条正确的路

物理层 邮差上车出发：网线/网卡

OSI模型在发送的过程中，数据在应用层出发，每经过一层，就要在数据上封装上一层的头，除了物理层

OSI模型在接收的过程中，数据由物理层传入，每经过一层，就要去掉一层的头，最后只剩数据到达应用层

后来又将这7层模型合并成了4层模型：TCP/IP协议模型

应用层 老板将自己要说的话写成信，封装好后再附上对方的地址 abc

传输层 邮差拿到信，决定以什么方式送信：TCP协议/UDP协议 tcp+abc

网络层 邮差分析地址在哪：IP协议 ip +port +tcp+abc

网络接口与物理层 邮差选择了一条正确的路就出发了 mac+ ip +port +tcp+abc

网络接口与物理层 邮差选择了一条正确的路就出发了 mac+ ip +port +tcp+abc

网络层 ip +port +tcp+abc

传输层 tcp+abc

应用层 abc

1. 什么是socket？(套接字)

socket也是一个函数：创建套接字的函数，通过返回值将套接字返回

套接字 = socket()

根据创建时选择的协议不同，socket返回的套接字也不同

TCP套接字 = socket(TCP协议)

UDP套接字 = socket(UDP协议)

1. 什么是IP地址？

例如”192.168.2.5”，点分制

在一个局域网中，每一台主机都必须有一个IP地址，且IP地址必须是相同网段：IP地址前3个数字一样

每一个IP地址都是32位，如果在网络编程中使用IP地址，就要将这32位转换成网络字节序

也就要使用一些字节序转换函数

1. 什么是端口号？

在局域网中，两台主机要进行通信，除了IP地址要在同一网段下，还要知道对方的端口号

端口号是16位的，取值范围：0 ~ 65535

系统占用的端口号：0 ~ 1023

用户可以使用的端口号：1024 ~ 65535

端口号结束使用后，最好过一段时间后再使用

1. ubuntu网络命令

查看网络信号：ifconfig

gec@ubuntu:/mnt/hgfs/WH2208/08网络编程/day1$ ifconfig

ens33(网卡) Link encap:Ethernet 以太网

HWaddr 00:0c:29:d2:3a:9f 硬件地址

inet addr:192.168.11.176 当前ubuntu的IP地址

Bcast:192.168.11.255 当前局域网的广播地址

Mask:255.255.255.0 子网掩码地址

修改IP地址：ifconfig + 网卡名称 + IP地址

记得使用管理员身份去做

例如：sudo ifconfig ens33 192.168.2.6

验证是否成功，和主机IP进行一次ping

例如我的主机IP地址为192.168.2.5

在ubuntu终端输入：ping 192.168.2.5 就是用ubuntu的IP地址去ping主机IP

如果出现”64 bytes from 192.168.2.5: icmp\_seq=1 ttl=128 time=0.895 ms”这种话就说明交互成功

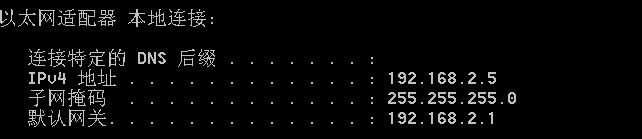
表示ubuntu可以通过网络来访问主机

如果ping不同，先检查是否在同一网段，再检测一下防火墙是否关闭

windows查看IP地址的方法

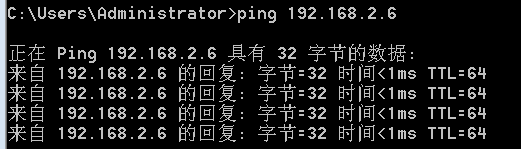
按下win+R键，调出运行框，输入cmd回车进入windows终端

在windows终端输入：ipconfig，就可以得到windowsIP地址等信息



也可以在windows上ping一下ubuntu，看能否ping通

例如ubuntuIP地址为：192.168.2.6

在windows终端输入：ping 192.168.2.6  
 

可以得到回复就说明可以接收到来自ubuntu的数据

# 通信协议

在不同的计算机里，双方都需要遵循同一个规则的通信协议模型

TCP/IP协议 传输控制协议/因特网互联协议

TCP协议 用于检测传输过程中出现的错误

IP协议 负责不同网络之间的数据交互

传输过程，一旦发生错误，要求重新传输数据，直到数据安全为止

1、TCP协议(打电话)

特点：字节传输，面向连接，效率相对较低

提供可靠的传输，确保数据无误，数据无丢失

应用场景：

需要可靠传输的数据，例如一些比较重要的数据

例如：微信账号的管理

2、UDP协议(写信)

特点：数据包传输，面向无连接，效率相对较高

可能会出现数据包的丢失

应用场景：

适合广播和组播

例如：网络媒体和视频、直播

3、TCP协议和UDP协议的区别

UDP：数据包套接字

一种对网络要求不高的传输方式，它的数据传输不会受到接收方的影响

UDP不可靠传输，只管发，不负责对象的接收数据

TCP：数据流套接字

一种对网络要求较高的传输方式，每次传输过去，都必须要接收到对方的响应

在连接时，会有一个”三次握手”的操作

在断开时，会有一个”四次挥手”的操作

TCP的可靠传输，由”三次握手”和”四次挥手”来保证，需要连接成功才会进行通信

1. 三次握手和四次挥手：

客户端主动打开，服务器被动打开

术语：

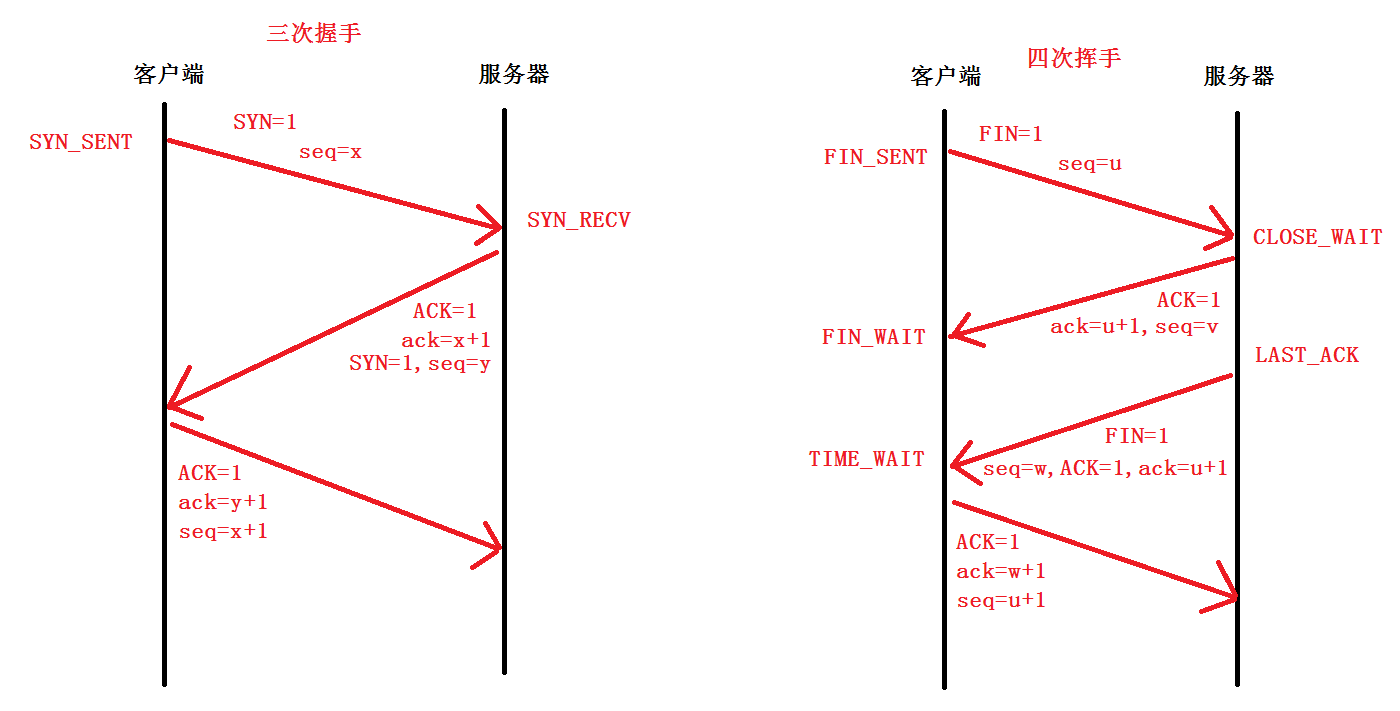
SYN：同步位，表示连接请求，一般为1

seq：序号，一般是随机数

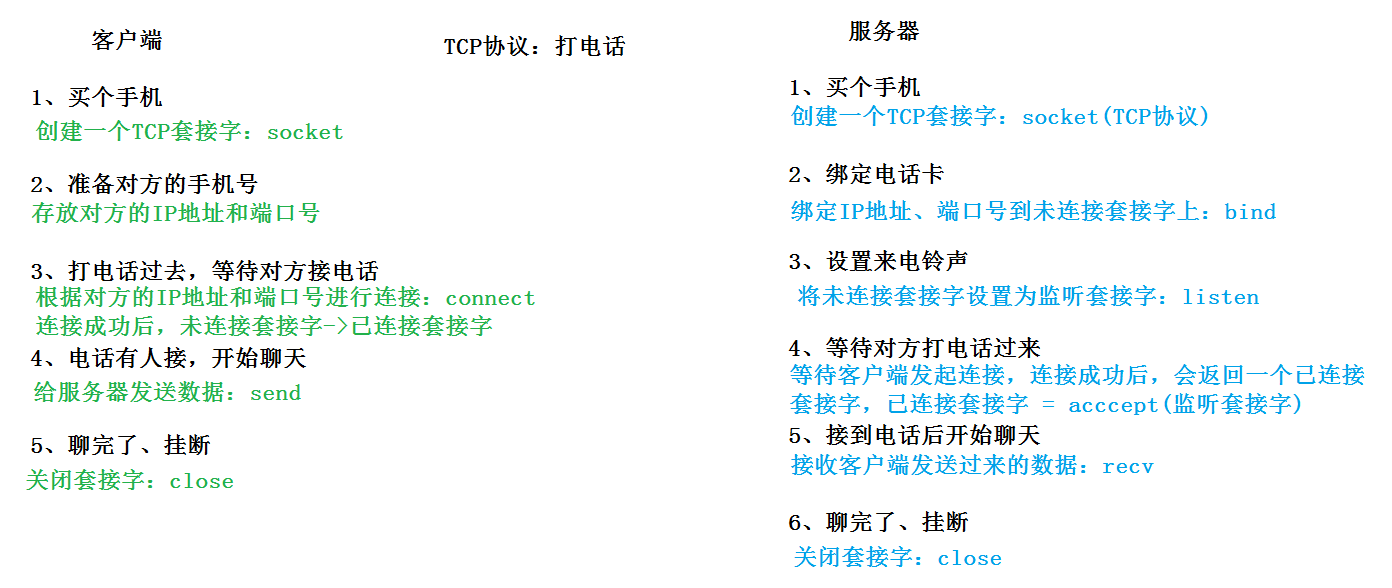
ACK：确认位，ACK=1表示确认有效，无值表示无效

ack：确认号，在对方发送的序号的基础上+1

FIN：取消位，表示断开请求，一般为1



# TCP通信



1. TCP通信的函数接口

1）创建套接字的函数

|  |
| --- |
| 头文件：  #include <sys/types.h>  #include <sys/socket.h>  函数原型：  int socket(int domain, int type, int protocol);  参数：  int domain：域  AF\_INET 网际协议  AF\_UNIX 本地协议  int type：  SOCK\_STREAM 流式套接字(TCP协议)  SOCK\_DGRAM 数据包式套接字(UDP协议)  int protocol： 协议，选择了网际协议就填0  因为选择了网际协议后，在选择流式/数据包式套接字就是选择了对应的协议：TCP协议/UDP协议  返回值：  成功：套接字  失败：-1 |

1. 绑定IP地址和端口号到套接字的函数

|  |
| --- |
| 函数原型：  int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);  参数：  int sockfd： 套接字  const struct sockaddr \*addr：存放着IP地址和端口号等信息的结构体的地址  socklen\_t addrlen：地址结构体的大小  返回值：  成功：0  失败：-1 |
| 关于存放IP地址和端口号的结构体类型：  struct sockaddr {  sa\_family\_t sa\_family; //协议/地址族 2字节  char sa\_data[14]; //IP地址、端口号  };  缺点：将IP地址和端口号存放在一个数组中，不方便无名进行写入  为了解决这个问题，诞生了另一种结构体，和这个结构体大小一致，方便我们进行类型转化  可以在第7本man手册中查看：man 7 ip  struct sockaddr\_in {  sa\_family\_t sin\_family; /\* 地址族: AF\_INET \*/  in\_port\_t sin\_port; /\* 端口号 \*/  struct in\_addr sin\_addr; /\* IP地址 \*/  };  /\* 存放IP地址的结构体类型 \*/  struct in\_addr {  uint32\_t s\_addr; /\* 32位无符号网络地址 \*/  };  要将字符串类型的端口号和IP地址转换成对应的网络字节序，就要使用下面的字节序转换函数 |

1. 字节序转换函数

|  |
| --- |
| 头文件：  #include <arpa/inet.h>  函数原型：  //h(host):主机 to n(net):网络 主机字节序转换成网络字节序  uint32\_t htonl(uint32\_t hostlong);  uint16\_t htons(uint16\_t hostshort);  //l(long):长4字节 s(short):短2字节  uint32\_t ntohl(uint32\_t netlong);  uint16\_t ntohs(uint16\_t netshort);  参数：  传入主机/网络字节序  返回值：  网络/主机字节序 |
| 转换端口号要使用：htons，但这个函数要求传入的是无符号16位整型数据，而传入的端口号都是字符串  所以还要使用一个字符串转整型的函数：atoi  头文件：  #include <stdlib.h>  函数原型：  int atoi(const char \*nptr);  传入一个都是数字组成的字符串，通过返回值将转换后的整型数返回 |
| 将字符串类型的IP地址转换成网络字节序的函数：  函数原型：  int inet\_pton(int af, const char \*src, void \*dst);  参数：  int af：转换的地址类型：AF\_INET->ipv4  const char \*src：要转换的字符串  void \*dst：拆分转换后的数据的内存地址  如果要将网络字节序的IP地址转换成字符串来打印，可以使用下面这个函数：  char \*inet\_ntoa(struct in\_addr in);  传入IP地址，会将字符串返回出来 |

1. 将未连接套接字设置为监听套接字的函数

|  |
| --- |
| 函数原型：  int listen(int sockfd, int backlog);  参数：  int sockfd：套接字  int backlog：最大同时连接请求个数  返回值：  成功：0  失败：-1  未连接套接字->listen->监听套接字 |

1. 等待客户端连接的函数

|  |
| --- |
| 函数原型：  int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);  参数：  int sockfd：监听套接字  struct sockaddr \*addr：存放连接对象的IP地址和端口号等信息的结构体  socklen\_t \*addrlen：存放结构体大小的变量的地址  返回值：  成功：已连接套接字  失败：-1 |

1. 连接对方监听套接字的函数

|  |
| --- |
| 函数原型：  int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);  参数：  int sockfd：套接字  const struct sockaddr \*addr：存放对方IP地址和端口号的结构体的地址  socklen\_t addrlen：结构体的大小  返回值：  成功：0  失败：-1  未连接套接字->connect->已连接套接字 |

1. 向套接字发送数据的函数

|  |
| --- |
| 函数原型：  ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);  参数：  int sockfd： 套接字  const void \*buf： 存放要发送的数据的内存  size\_t len： 要发送的数据长度  int flags： 发送标志，默认填0  返回值：  成功：发送的字节数  失败：-1 |

1. 从套接字接收数据的函数

|  |
| --- |
| 函数原型：  ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);  参数：  int sockfd： 套接字  void \*buf： 存放接收到的数据的内存  size\_t len： 要接收的字节数  int flags： 接收标志，默认填0  返回值：  成功：接收到的字节数  失败：-1 |

9）关闭套接字的函数

close(套接字)

1. 实际应用

|  |
| --- |
| 服务器：  void \*recv\_data(void \*arg)  {  int connfd = \*(int \*)arg;  char buf[50];  while(1)  {  bzero(buf,sizeof(buf));  recv(connfd,buf,sizeof(buf),0);  printf("recv is %s\n",buf);  if(!strncmp(buf,"quit",4))  break;  }  close(connfd);  exit(0);  }  //./server 192.168.2.6 50001  int main(int argc,char \*argv[])  {  //创建一个TCP协议的套接字  int sockfd = socket(AF\_INET,SOCK\_STREAM,0);  if(sockfd == -1)  {  perror("make socket error\n");  return -1;  }  //定义一个存放自己的IP地址的结构体  struct sockaddr\_in server\_addr;  bzero(&server\_addr,sizeof(server\_addr));  //将IP地址和端口号存放进结构体中  server\_addr.sin\_family = AF\_INET;//地址族  server\_addr.sin\_port = htons(atoi(argv[2])); //端口号  inet\_pton(AF\_INET,argv[1],&server\_addr.sin\_addr);//IP地址  //服务器绑定自己的IP地址  bind(sockfd,(struct sockaddr \*)&server\_addr,sizeof(server\_addr));  //将绑定好的套接字设置为监听套接字  listen(sockfd,5);  //定义一个存放连接对象的IP地址和端口号等信息的结构体  struct sockaddr\_in client\_addr;  int client\_addr\_size = sizeof(client\_addr);  bzero(&client\_addr,client\_addr\_size);  //等待客户端的连接  int connfd = accept(sockfd,(struct sockaddr \*)&client\_addr,&client\_addr\_size);  if(connfd == -1)  {  perror("connect error\n");  return -1;  }  printf("%s is connect\n",inet\_ntoa(client\_addr.sin\_addr));  pthread\_t tid;  pthread\_create(&tid,NULL,recv\_data,(void \*)&connfd);  char buf[50];  while(1)  {  bzero(buf,sizeof(buf));  scanf("%s",buf);  send(connfd,buf,strlen(buf),0);  if(!strncmp(buf,"quit",4))  break;  }  close(sockfd);  close(connfd);  return 0;  } |
| 客户端：  void \*recv\_data(void \*arg)  {  int sockfd = \*(int \*)arg;  char buf[50];  while(1)  {  bzero(buf,sizeof(buf));  recv(sockfd,buf,sizeof(buf),0);  printf("recv is %s\n",buf);  if(!strncmp(buf,"quit",4))  break;  }  close(sockfd);  exit(0);  }  //./client 192.168.2.6 50001  int main(int argc,char \*argv[])  {  //创建一个TCP协议的套接字  int sockfd = socket(AF\_INET,SOCK\_STREAM,0);  if(sockfd == -1)  {  perror("make socket error\n");  return -1;  }  //准备一个存放对方IP地址和端口号的结构体  struct sockaddr\_in server\_addr;  bzero(&server\_addr,sizeof(server\_addr));  //将对方的IP地址和端口号存放进结构体中  server\_addr.sin\_family = AF\_INET;//地址族  server\_addr.sin\_port = htons(atoi(argv[2])); //端口号  inet\_pton(AF\_INET,argv[1],&server\_addr.sin\_addr);//IP地址  //连接对方的监听套接字  connect(sockfd,(struct sockaddr \*)&server\_addr,sizeof(server\_addr));  pthread\_t tid;  pthread\_create(&tid,NULL,recv\_data,(void \*)&sockfd);  char buf[50];  while(1)  {  bzero(buf,sizeof(buf));  scanf("%s",buf);  send(sockfd,buf,strlen(buf),0);  if(!strncmp(buf,"quit",4))  break;  }  close(sockfd);  return 0;  } |