网络编程

一、为什么要学习网络编程

进程间通信方式：

- 无名管道
- 有名管道
- 信号
- 消息队列
- 共享内存
- 信号灯集
- socket 套接字通信

前六种只能实现同主机的进程间通信，而socket 套接字既可以实现同主机的进程间通信也能实现不同主机的进程间通信。

二、课程体系

八天时间

1.网络发展历史、网络编程相关的概念

2.TCP网络编程

3.UDP网络编程

4.IO模型

5.服务器模型

6.网络超时检测

7.广播、组播

8.本地通信

9.数据库

10.原始套接字--选讲

11.结合C基础 C高级 数据结构 IO进程 网络编程 数据库 一个项目。

三、网络发展历史

四个阶段

- APRANET阶段
- TCP/IP两个协议阶段
- OSI开放系统互联模型
- TCP/IP协议族

3.1 APRANET阶段

1968年6月DARPA提出“资源共享计算机网络”

(Resource Sharing Computer Networks)，目的在于让DARPA的所有电脑互连起来，这个网络就叫做ARPAnet，即“阿帕网”，是Interne的最早雏形

早期的ARPAnet使用网络控制协议(Network Control Protocol，**NCP**)，不能互联不同类型的计算机和不同类型的操作系统，没有纠错功能

3.2 TCP/IP两个协议阶段

1974年12月两人正式发表第一份TCP协议详细说明，但此协议在有数据包丢失时不能有效的纠正

TCP/IP协议分成了两个不同的协议：

用来检测网络传输中差错的传输控制协议TCP

专门负责对不同网络进行互联的互联网协议IP

3.3 网络的体系结构

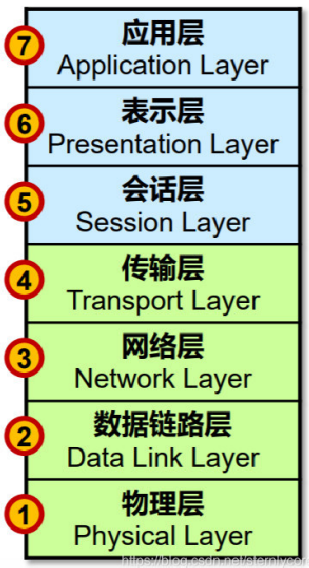
网络采用分而治之的方法设计，将网络的功能划分为不同的模块，以分层的形式有机组合在一起。

每层实现不同的功能，其内部实现方法对外部其他层次来说是透明的。每层向上层提供服务，同时使 用下层提供的服务

网络体系结构即指网络的层次结构和每层所使用协议的集合

OSI开放系统互联模型

是由 ISO 国际标准化组织提出的。

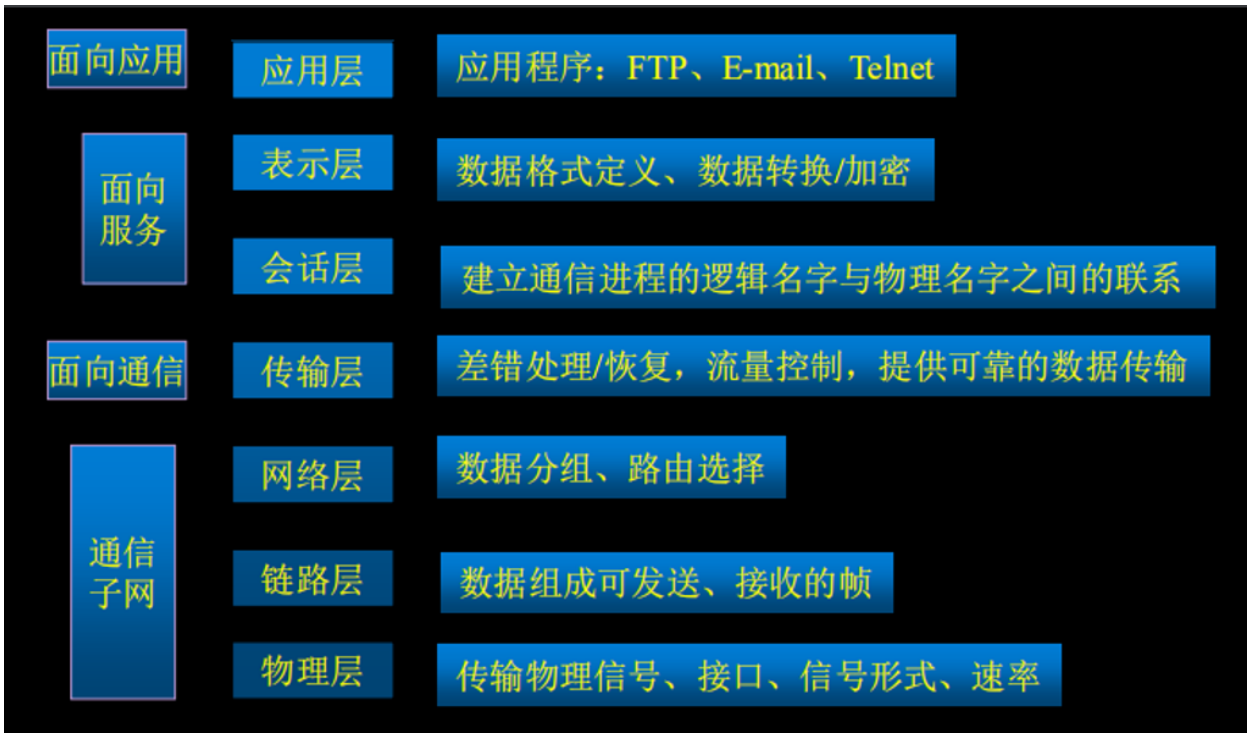


从下往上说也行

从上往下说也行

注意：顺序不能颠倒。

物数网传会表应

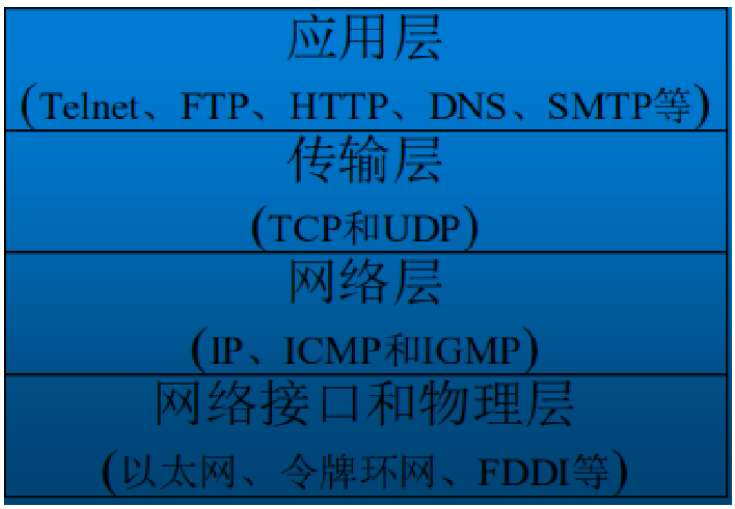


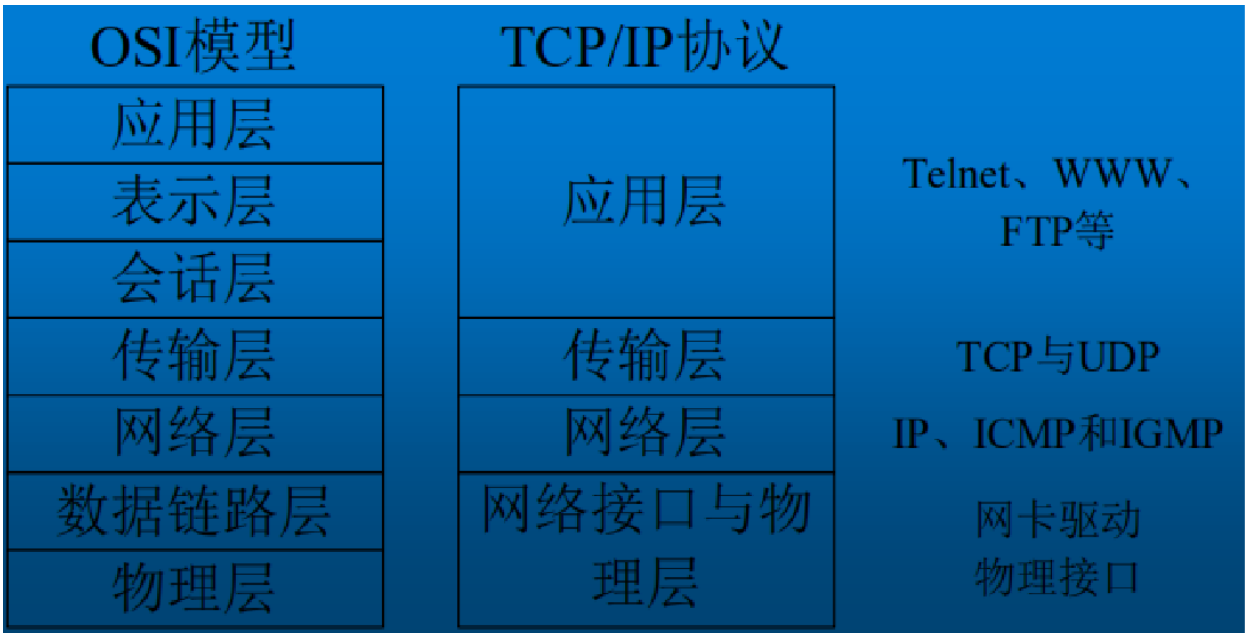
3.4 TCP/IP协议族(簇)的体系结构

实际生产过程中，由于OSI开放系统互联模型太过繁琐，所以没人使用，

但是他的思想思路是正确的，后面提出的其他体系结构，都是基于OSI开放系统互联模型而来的。

TCP/IP协议族是Internet事实上的工业标准。





TCP/IP协议族体系结构每层常见的协议

应用层：

HTTP(Hypertext Transfer Protocol) 超文本传输协议

万维网的数据通信的基础

FTP(File Transfer Protocol) 文件传输协议

是用于在网络上进行文件传输的一套标准协议，使用TCP传输

TFTP(Trivial File Transfer Protocol) 简单文件传输协议

是用于在网络上进行文件传输的一套标准协议，使用TCP传输

SMTP(Simple Mail Transfer Protocol) 简单邮件传输协议

一种提供可靠且有效的电子邮件传输的协议

传输层：

TCP(Transport Control Protocol) 传输控制协议

是一种**面向连接的、可靠的**、基于字节流的传输层通信协议

UDP(User Datagram Protocol) 用户数据报协议

是一种**无连接、不可靠**、快速传输的传输层通信协议

网络层：

IP(Internetworking Protocol) 网际互连协议

是指能够在多个不同网络间实现信息传输的协议

ICMP(Internet Control Message Protocol) 互联网控制信息协议

用于在IP主机、路由器之间传递控制消息 ----ping 命令使用的协议

IGMP(Internet Group Management Protocol) 互联网组管理

是一个组播协议，用于主机和组播路由器之间通信

链路层：

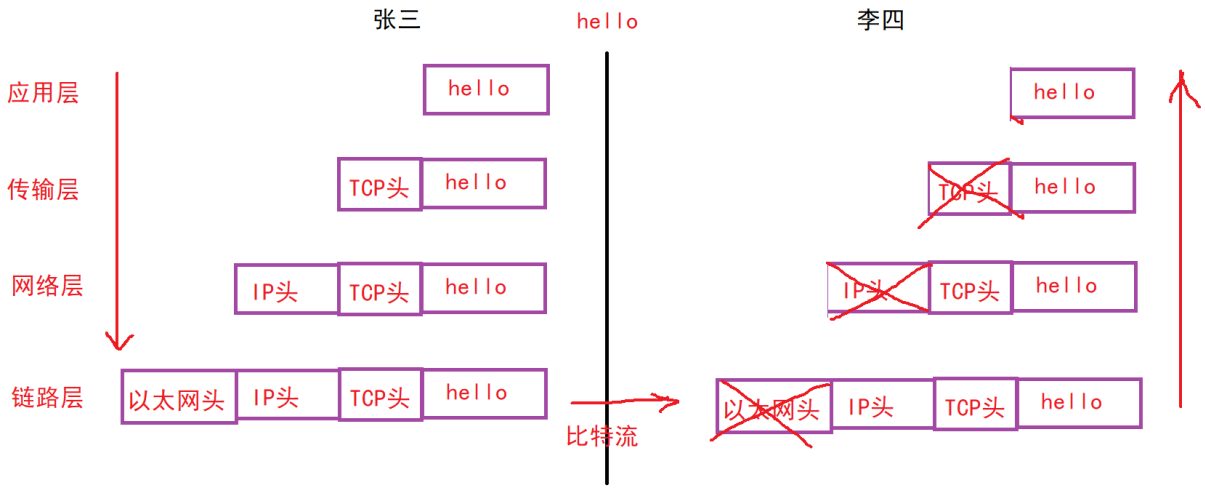
ARP(Address Resolution Protocol) 地址解析协议

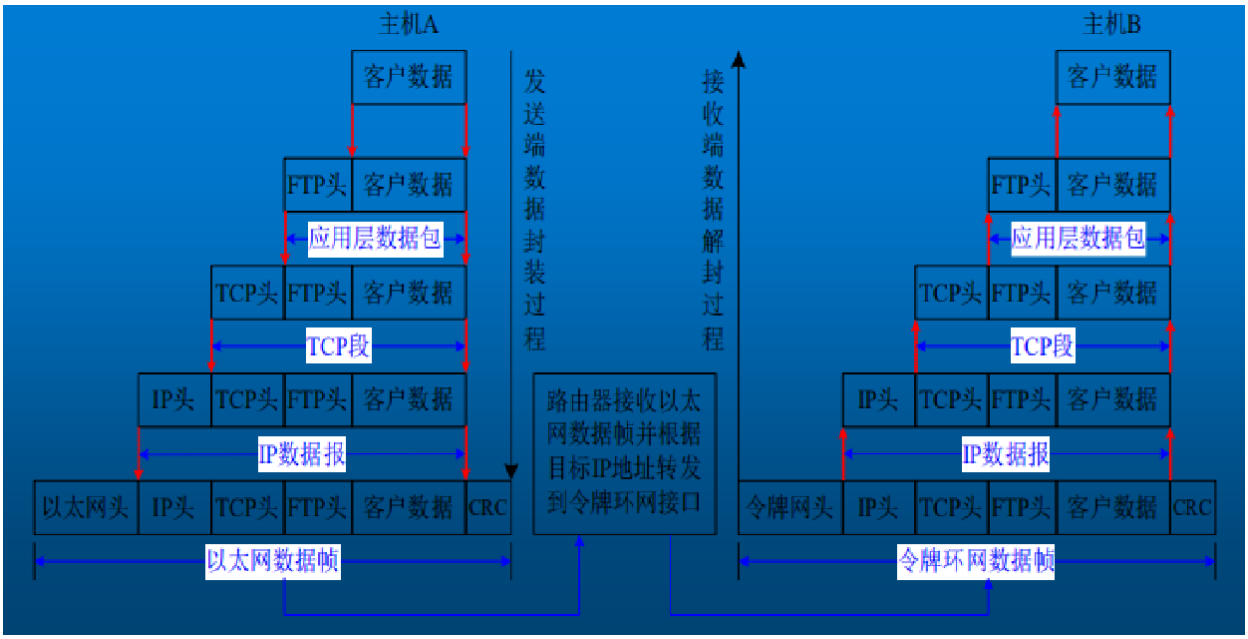
通过IP地址获取对方mac地址

RARP(Reverse Address Resolution Protocol) 逆向地址解析协议

通过mac地址获取ip地址

数据封包和拆包的过程：





插一句：

LINUX内核五大功能：

进程管理：时间片轮转、上下文切换

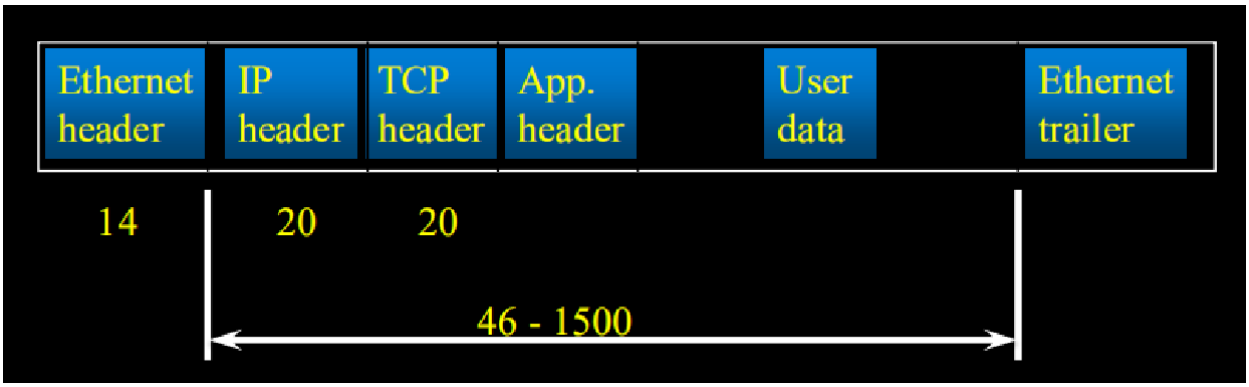
内存管理：内存的分配和回收

文件管理：将一堆 0 1 转换成人类能识别的字符

设备管理：一切皆文件

网络管理：网络协议栈(网络协议栈可以理解成内核提供的函数)

一帧数据说明：



有些书上写的 64-1518 (包括了以太网头和以太网尾)

MTU 最大传输单元 linux 系统默认的就是1500，超过这个值，会被拆分成多个帧发送。

四、UDP和TCP的异同

相同点:

都是传输层协议

不同点:

TCP（即传输控制协议）概念:

是一种面向连接的传输层协议，它能提供高可

靠性通信(即数据无误、数据无丢失、数据无失序、数

据无重复到达的通信)

适用情况:

适合于对传输质量要求较高，以及传输大量数据的通信。

在需要可靠数据传输的场合，通常使用TCP协议

MSN/QQ等即时通讯软件的用户登录账户管理相关的功能，通常采用TCP协议

UDP（User Datagram Protocol）用户数据报协议

是不可靠的无连接的协议。在数据发送前，

因为不需要进行连接，所以可以进行高效率的数据传输。

适用情况:

发送小尺寸数据（如对DNS服务器进行IP地址查询时）

在接收到数据，给出应答较困难的网络中使用UDP。（如：无线网络）

适合于广播/组播式通信中。

MSN/QQ/Skype等即时通讯软件的点对点文本通讯以及音视频通讯通常采用UDP协议

流媒体、VOD、VoIP、IPTV等网络多媒体服务中通常采用UDP方式进行实时数据传输

五、网络编程基础知识

5.1 字节序

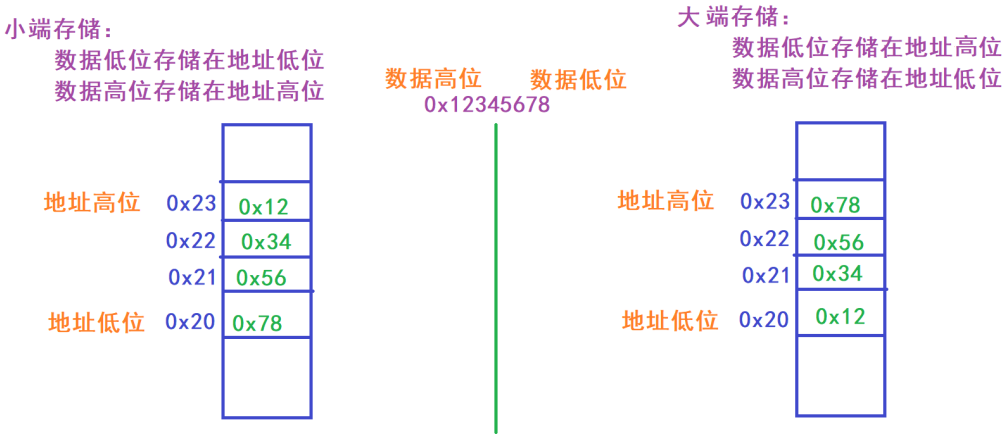
不同类型CPU的主机中，内存存储多字节整数序列有两种方法，称为主机字节序(HBO)：

小端序（little-endian） - 低序字节存储在低地址

将低字节存储在起始地址，称为“Little-Endian”字节序，Intel、AMD等采用的是这种方式；

大端序（big-endian）- 高序字节存储在低地址

将高字节存储在起始地址，称为“Big-Endian”字节序，由ARM、Motorola等所采用



判断自己的主机字节序?

```c
#include <stdio.h>

int main(){
    int m = 0x12345678;
    char *p = (char *)&m;
    if(*p == 0x78){
        printf("小端\n");
    }else if(*p == 0x12){
        printf("大端\n");
    }

    return 0;
}
```

由于不同类型的CPU主机字节序不一定相同，所以发送端发送的数据和接收端接收到的数据就有可能出现不一致的情况，所以就发明了 网络字节序 的概念，规定**数据在网络中传输，必须是大端序**，也就是说：发送端需要将发送的数据转换成大端序发送，接收到接收到的就是大端序的数据，在根据自己的主机的字节序做转换。

什么情况下需要考虑字节序转换的问题?

1.如果数据是1个字节的，无需考虑字节序的问题，

如果超过了1个字节的数据作为一个整体时，就需要考虑了

2.如果明确知道通信的双方主机字节序一样，也可以不考虑

思考：如何将小端序的无符号四字节整型转换成大端序?

```c
#include <stdio.h>

int main(){
    unsigned int m = 0x12345678;
    unsigned int n = 0;
    char *p = (char *)&m;
    char *q = p+3;
    unsigned char temp = 0;
    temp = *p;
    *p = *q;
    *q = temp;
    p++;
    q--;
    temp = *p;
    *p = *q;
    *q = temp;
    printf("%#x  -->  %#x\n", m, n);
    return 0;
}
```

字节序转换函数

h host 主机 n network 网络 l 长 s 短

uint32_t htonl(uint32_t hostlong);//主机转网络 4字节
uint16_t htons(uint16_t hostshort);//主机转网络 2字节
uint32_t ntohl(uint32_t netlong);//网络转主机 4字节
uint16_t ntohs(uint16_t netshort);//网络转主机 2字节

```c
//以 htonl 为例
#include <arpa/inet.h>
#include <stdio.h>

int main(){
    unsigned int m = 0x12345678;
    unsigned int n = htonl(m);
    printf("%#x --> %#x\n", m, n);
```

```
    return 0;
}
```

执行结果：

```
yangfs@hqyj:~/Teach_Node/22051/network/day1$ ./a.out
0x12345678 --> 0x78563412
yangfs@hqyj:~/Teach_Node/22051/network/day1$
```

5.2 socket

socket本来也是用于本地进程间通信的，后来有了TCP/IP协议族的加入，才能实现跨主机通信。

socket是一个函数，我们可以指定参数告诉内核封装什么样的协议

socket是一种特殊的文件描述符 (everything in Unix is a file)

并不仅限于TCP/IP协议，其他体系结构也会用到 socket

套接字分为：

流式套接字(SOCK_STREAM)TCP

提供了一个面向连接、可靠的数据传输服务，数据无差错、无重复的发送且按发送顺序接收。内设置流量控制，避免数据流淹没慢的接收方。数据被看作是字节流，无长度限制。

数据报套接字(SOCK_DGRAM)UDP

提供无连接服务。数据包以独立数据包的形式被发送，不提供无差错保证，数据可能丢失或重复，顺序发送，可能乱序接收。

原始套接字(SOCK_RAW)

可以对较低层次协议如IP、ICMP直接访问。

5.3 IP地址

ip地址是主机在网络中的编号，这个编号就是IP地址。

IP地址和MAC地址区别：

每张网卡在出厂时都会有一个唯一的标识，叫做MAC地址。

在局域网内部通信，都是使用MAC地址通信的。交换机是工作在链路层的设备

如果数据想要走出局域网，就要使用IP地址了。路由器是工作在网络层的设备

IP地址分为 IPV4 和 IPV6

IPV4 4字节（32bit）

IPV6 16字节 (128bit)

为什么会有IPV4和IPV6之分?

是因为IPV4地址不够用了。

IPV4地址不够用了，不一定必须使用IPV6，因为现在有很多技术，都可以

弥补IPV4数量不足的缺陷，如 NAT 技术：任意一个IP地址都可以经过路由器

下发局域网IP地址，局域网内部通信，使用局域网的IP地址，如果数据要走出局域网

会通过NAT技术，将数据包中的源IP地址从局域网IP替换成公网IP。

通过百度查询 到的 IP地址，是我们付费在运营商那里租用过来的。

通过我们自己的路由器，可以下发多个局域网IP地址，也就是我们ifconfig查询到的ip地址。

ip地址的表示形式:

192.168.70.10 叫做点分十进制，是方便我们人类看的，是一个字符串类型。

而在计算中，ip地址就是一个无符号的4字节整型。

IPV4的地址的组成:

由 网络号 和 主机号 组成。

IPV4地址分类:

网络号 主机号 规定最高位 范围 使用单位

A 1字节 3字节 0 0.0.0.0 - 127.255.255.255 政府/大公司/学校

B 2字节 2字节 10 128.0.0.0-191.255.255.255 中等规模的公司

C 3字节 1字节 110 192.0.0.0-223.255.255.255 个人

192.168.1.255 广播地址

D 1110 [224-239] 组播

E 11110 [240-255] 保留测试用的

其中每个IP地址又可以通过路由器，下发局域网IP地址，

每类IP地址都有专门划分子网的保留段。

类型	私有IP地址范围
A类	10.0.0.1~10.225.255.254
B类	172.16.0.1~172.31.255.254
C类	192.168.0.1~192.168.255.254

子网掩码：

是由一堆连续的1和连续的0组成的。

用来和IP地址取与运算来获取网络号的。

从而能限制某一网段内能容纳的最大主机数。

如，ip地址是192.168.70.8 子网掩码设置成 255.255.255.0

取与运算可以得到的结果：192.168.70.0 ----这是网络号，网络号相同时，才能进行同信

这是，该网段内共有IP地址 256 个：

其中 192.168.70.0 是网络号，是不能占用的

192.168.70.255 是广播的地址，也不能占用

网关设备也需要占用一个IP地址，一般是同一局域网内可用的编号最小的。

192.168.70.1

所以能容纳的主机数：256-1-1-1(网关设备也可以算作一台IP主机) = 254

192.168.70.x 网段将子网掩码设置成 255.255.255.128，同一子网能容纳的最大主机数?

答案：这种方式可以将同一网段再划分成两个子网

第一个子网IP地址范围：192.168.70.0~192.168.70.127

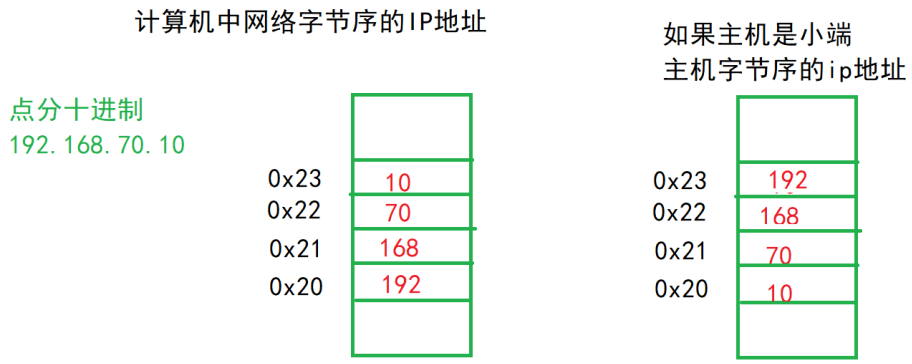
第二个子网IP地址范围：192.168.70.128~192.168.70.255

其中每个子网中有需要减掉 网络号 广播地址 网关地址

所以子网掩码不一定都是255255255.0 网关设备的IP地址也不一定是编号最小的。

既然是无符号4字节整型，就涉及字节序的问题。

在网络上字节序下，ip地址的存储：



IP地址转换的函数：

inet_addr()

将strptr所指的字符串转换成32位的网络字节序二进制值。

in_addr_t inet_addr(const char *strptr);

inet_ntoa()

将32位网络字节序二进制地址转换成点分十进制的字符串。

char *inet_ntoa(stuct in_addr inaddr);

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, const char *argv[])
{
    unsigned char ip_str[] = "192.168.70.10";
    unsigned int ip_int = 0;

    ip_int = inet_addr(ip_str);

    unsigned char *p = (unsigned char *)&ip_int;
    printf("%d.%d.%d.%d\n", *p, *(p+1), *(p+2), *(p+3));
      //执行结果  192.168.70.10    ----网络字节序的无符号4字节整型

    return 0;
}
```

5.4 端口号

虽然在linux系统中，进程号是用来标识唯一进程的，但是由于进程号是有操作系统分配的
且进程停止再运行时，不能保证进程号不变，所以使用进程号来标识进程就可能出现问题。
所以就发明了端口号来标识进程，端口号是人为可以自己指定的，
为了区分一台主机接收到的数据包应该转交给哪个进程来进行处理，使用端口号来区别

端口号的范围 0~65535 ,使用 unsigned short 存储
也需要考虑字节序的问题

端口号一般由IANA (Internet Assigned Numbers Authority) 管理
众所周知端口：1~1023（1~255之间为众所周知端口，256~1023端口通常由UNIX系统占用）
已登记端口：1024~49151
动态或私有端口：49152~65535

我们学习阶段 8888 9999 6789 8989 测试使用即可，实际开发过程中，
端口号是由用户自己管理的。

常见的服务使用端口号：

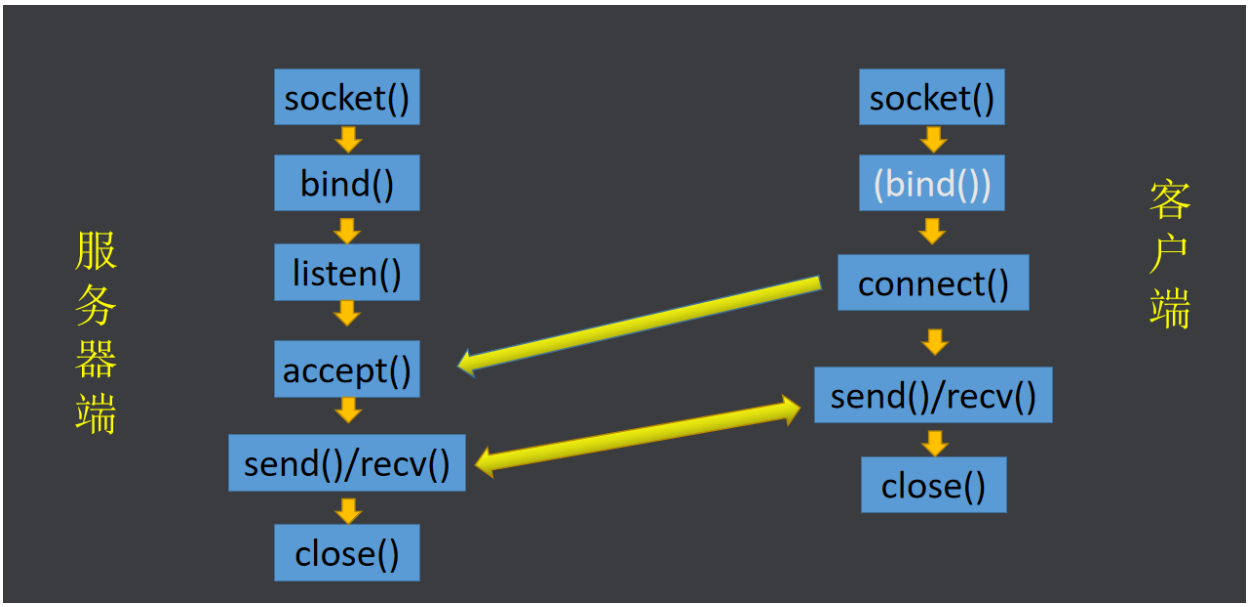
FTP 21
SSH 22
TFTP 69
HTTP 80 8080

六、TCP网络编程

网络编程模型：

C/S 模型：客户端 服务器 模型 ----我们课上研究的都是C/S模型
B/S 模型：浏览器 服务器 模型

6.1 流程



流程必须要掌握！！！

服务器：

创建流式套接字--socket()

填充服务器的网络信息结构体

将套接字和服务器的网络信息结构体绑定--bind()

将套接字设置成被动监听状态--listen()

阻塞等待客户端连接--accept()

数据收发--read write

关闭套接字--close()

客户端：

创建流式套接字--socket()

填充服务器的网络信息结构体

与服务器建立连接--connect()

数据收发--read write

关闭套接字--close()

6.2 函数说明

6.2.1 socket函数

```
功能：
    创建套接字，用于通信
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：
    domain:
        指定通信域：
            AF_INET             IPV4使用
            AF_INET6            IPV6使用
            AF_UNIX 或 AF_LOCAL 本地通信使用
            AF_PACKET           原始套接字使用
    type:
        套接字类型
            SOCK_STREAM     TCP使用
            SOCK_DGRAM      UDP使用
            SOCK_RAW        原始套接字使用
    protocol:
```

```
        附加协议   如果没有附加协议   传 0 即可
返回值：
    成功   返回文件描述符
    失败   -1
```

6.2.2 bind函数

```
功能：
    将套接字和网络信息结构体绑定
头文件：
    #include <sys/types.h>
    #include <sys/socket.h>
函数原型：
    int bind(int sockfd, const struct sockaddr *addr,
                 socklen_t addrlen);
参数：
    sockfd :  socket函数返回的套接字
            struct sockaddr {//这个结构体只是为了做强转防止编译警告的
                sa_family_t sa_family;
                char        sa_data[14];
            }

            //我们使用的结构体是下面这个
            struct sockaddr_in {
                sa_family_t    sin_family; /* AF_INET */
                in_port_t      sin_port;   /* 网络字节序的端口号 */
                struct in_addr sin_addr;   /* IP地址 */
            };

            /* Internet address. */
            struct in_addr {
                nt32_t       s_addr;     /* 网络字节序的无符号4字节整型的IP地址 */
            };
    addrlen：
        addr的大小

返回值：
    成功   0
    失败 -1
```

6.2.3 listen函数

```
功能：
    将套接字设置成被动监听状态
头文件：
    #include <sys/types.h>
    #include <sys/socket.h>
函数原型：
    int listen(int sockfd, int backlog);
参数：
    sockfd：  socket函数返回的套接字
    backlog：允许同时连接服务的客户端的个数
            设置成 5  或者 10  都可以  只要不是0 就行
返回值：
    成功  0
    失败  -1
```

6.2.4 accpet函数

```
功能：
    阻塞等待客户端连接，一旦有客户端连接，就会返回一个
    新的套接字(文件描述符)专门用于和该客户端进行通信
头文件：
    #include <sys/socket.h>
函数原型：
    int accept(int socket, struct sockaddr * address,
                socklen_t * address_len);
参数：
    socket:         socket函数返回的套接字
    address:        用来保存客户端网络信息结构体的首地址，
                    如果不关心客户端的信息，可以传NULL
    address_len:    address长度，如果没有可以传NULL
返回值：
    成功  返回文件描述符 专门用于和该客户端通信
    失败  -1
```

6.2.5 connect 函数

```
功能：
    与服务器建立连接
头文件：
    #include <sys/types.h>
    #include <sys/socket.h>
函数原型：
    int connect(int sockfd, const struct sockaddr *addr,
                    socklen_t addrlen);
参数：
    sockfd：socket函数返回的文件描述符
    addr：要连接的服务器的网络信息结构体
    addrlen：  addr 的大小
返回值：
    成功  0
    失败  -1
```

6.3 代码实现

服务器：

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>

int main(){
    //创建 流式 套接字 ----买了一部手机
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(-1 == sockfd){
        perror("socket error");
        exit(-1);
    }

    //填充服务器网络信息结构体  ----办卡
```

```c
    struct sockaddr_in serveraddr;
    memset(&serveraddr, 0, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    //网络字节序的端口号 8888  9999  6789 等 都可以
    serveraddr.sin_port = htons(9999);
    //网络字节序的IP地址，IP地址不能乱填
    //自己的主机ifconfig 查到的ip地址是多少就填多少
    //如果本机测试使用 也可以填写 127.0.0.1
    serveraddr.sin_addr.s_addr = inet_addr("192.168.60.109");

    socklen_t serveraddr_len = sizeof(serveraddr);

    //将套接字和网络信息结构体绑定 ----插卡
    if(-1 == bind(sockfd, (struct sockaddr *)&serveraddr, serveraddr_len)){
        perror("bind error");
        exit(-1);
    }

    //将套接字设置成被动监听状态  ----这种状态才能接受客户端的连接
    if(-1 == listen(sockfd, 5)){
        perror("listen error");
        exit(-1);
    }

    //阻塞等待客户端连接
    int acceptfd = 0;
    if(-1 == (acceptfd = accept(sockfd, NULL, NULL))){
        perror("acceptfd error");
        exit(-1);
    }
    printf("有新的客户端连接了\n");
    char buff[128] = {0};
    //收发数据
    while(1){
        memset(buff, 0, 128);
        //接受数据
        read(acceptfd, buff, 128);
        //打印数据
        printf("buff:[%s]\n", buff);
        //组装应答
```

```
60          strcat(buff, "--hqyj");
61          //发送应答
62          write(acceptfd, buff, 128);
63      }
64      //关闭套接字
65      close(acceptfd);
66      close(sockfd);
67
68      return 0;
69  }
```

作业：

1.复习课上知识，tcp服务器代码自己必须要能写出来

2.没什么问题的同学自己man一下 connect 函数 尝试写出 TCP客户端的代码