概念

```
1 打印错误信息：https://www.cnblogs.com/Jimmy1988/p/7485133.html
```

- C/S模式：客户端/服务器模式。

  工作原理：Client向Server提交一个请求；Server则使用一些方法处理这个请求，并将效果返回给Client

  优点：①协议选用灵活

  ②缓存数据

  缺点：①对用户安全构成威胁

  ②开发工作量大，调式困难

- B/S结构：浏览器/服务器结构，对C/S结构的一种变化或者改进的结构

  在这种结构下，用户界面完全通过WWW浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。

  优点：优点和缺点和C/S模式是相反的

一、模型分层

- OSI参考模型：物、数、网、传、会、表、应

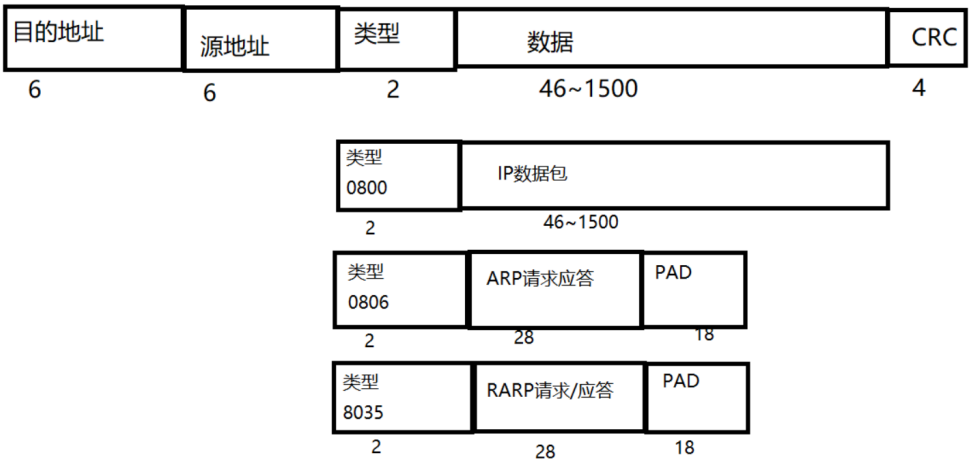
1. 物理层：主要定义物理设备标准，如网线的接口类型等，它的主要作用是传输比特流（1和0）
2. 数据链路层：数据校验，定义了如何让格式化数据以帧为单位进行传输。
3. 网络层：网络传输过程中提供的网络访问的路径。
4. 传输层：定义了一些传输数据的协议和端口号（端口号代表的是进程）
5. 会话层和表示层：是对传输层的数据进行封装和解封装
6. 应用层：主要是针对应用程序所进行的一些封装（fth、http）

数据包的逐层封装与解包都是操作系统来完成的。数据要进行传输必须进行封装

- TCP/IP模型：网络接口层（链路层）、网络层（ IP协议）、传输层（TCP/UDP）、应用层(FTP协议)·：网网传应

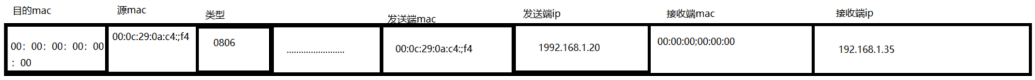
二、数据传输

1. 路由器：路由器根据自身的路由表在网络中选择路径进行寻路（寻找下一个路由节点 ），从而到达目标的IP。通过路由表来选择下一跳的路由节点。
2. 数据传输不单单依靠ip，还会依靠以太网帧格式（数据链路层）。



- 目的地址和源地址指的是电脑网卡的硬件地址（也叫MAC地址），长度是48位
- ARP协议：用来获取下一个路由节点的协议（根据IP地址获取mac地址）
- 以太网帧协议：根据mac地址完成数据传输

请求：（0806）



回应：将目的mac和接收端mac进行填充。发送端和目的端与请求的时候相反

- PAD表示填充（凑够46字节）
- 一个路由节点称为下一跳
- arp数据报：获取下一跳mac地址
- TTL指的是数据包最长生存周期（单位是跳）

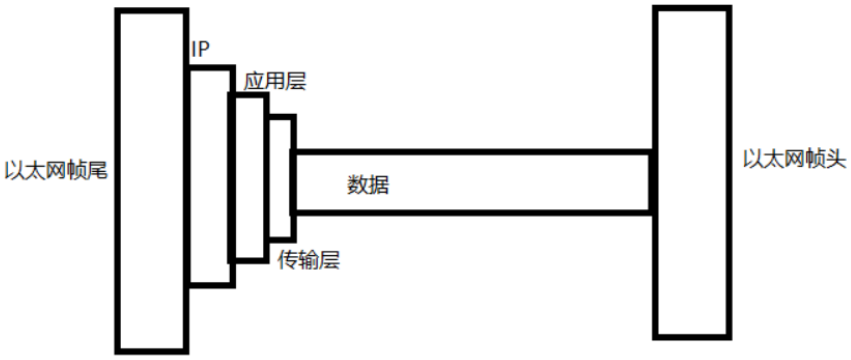
3.IP段格式：

TTL:time to live

IP段主要是要知道其中的32位源ip和32位目的IP地址，它和以太网帧格式中源IP和目的IP不一样。

- IP段中的源ip和目的ip指的是起始位置跟终止位置的ip地址
- 以太网帧格式中的源ip和目的ip指的是一个路由节点到下一个路由节点的地址
- ip段长度最长为65535字节

数据封装图



- 数据-应用层-网络层-链路层（以太网）
- IP段对应到以太网帧格式里面对用的是46~1500字节部分

4.传输层：UDP和TCP

- UDP数据格式

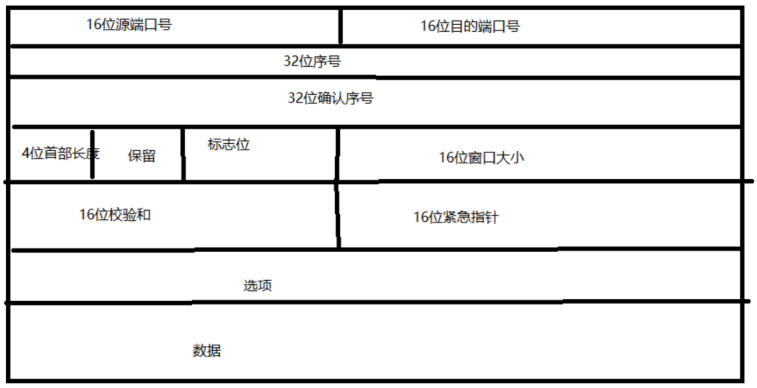
IP地址：可以在网络环境中唯一标识一台主机

端口号：可以在网络的一台主机上，唯一标识一个进程

IP地址+端口号 可以在网络环境中唯一标识一个进程

16位源端口号，16位目的端口号

- TCP数据格式



5.NAT映射表和打洞机制

将一个ip映射为另外一个地址，接下来的路由节点会接受这个映射到的ip。一个公网ip可以下发N多私有IP，私有ip在nat映射表下映射出的IP地址相同，但是端口号不同。

详见https://blog.csdn.net/it_is_me_a/article/details/100532139

三、套接字

IP地址：在网络中唯一标识一台主机

端口号：在主机中唯一标识一个进程

IP+Port：在网络中唯一标识一个进程（socket），所以socket需要绑定IP和端口号

socket----套接字是Linux的一种文件类型（伪文件）

一个文件描述符指向一个套接字，该套接字内部由内核借助两个两个缓存区实现

LINUX的文件类型

1. 普通文件类型
2. 目录文件
3. 块设备文件
4. 字符设备
5. 套接字文件
6. 管道文件：FIFO
7. 链接文件

- socket套接字在内核中一个文件描述符对应两个缓冲区（全双工）一个读一个写，成对出现。

网络字节序

- 网络字节流采用的是大端模式（高地址-低位）。而计算机大多采用的是小端模式
- 大端法：高位存高位，低位存低位
- 小端法：高位存低位，低位存高位
- 网络字节序和主机字节序的转换

```
#include<arpa/inet.h>
uint32_t htonl(uint32_t hostlong)//本地字节序转换成网络字节序32位;
uint16_t htons(uint16_t hostshort)//本地字节序转换成网络字节序16位
uint32_t ntohl(uint32_t netlong )//网络字节序转换成本地字节序32位
```

```
uint16_t ntohs(uint16_t netshort)//网络字节序转换成本地字节序16位
H表述host，N表示network，L表示32位长整数，S表示16位短整数
```

(IP地址转换)

```
inet_pton和inet_ntop:

192.168.1.24----->unsigned int ->htonl()->网络字节序
192.168.1.24------>网络字节序 inet_pton() //字符串（p）->net\\
192.168.1.24 -> string -> atoi -> int ->htol ->网络字节序
```

inet_pton和inet_ntop:点分文本的IP地址转换为二进制**网络字节序**"的IP地址

- sockaddr数据结构

struct sockaddr这种结构体已经弃用了，使用的时候定义struct sockaddr_in，但是在bind等函数中使用的时候还是要强制转换成sockaddr类型

命令man 7 ip有这个结构体的声明

```
 struct sockaddr_in {
               sa_family_t    sin_family; /* address family: AF_INET /AF_INET6*/ IPV4orI
               in_port_t      sin_port;   /* port in network byte order */需要进行网络字节
               struct in_addr sin_addr;   /* internet address */需要进行网络字节序的转换
           };
            struct in_addr {
               uint32_t       s_addr;     /* address in network byte order */
           };

```

- socket函数：创建socket

```
命令man socket
int socket（int domain,int type,int protocol）;
domain:选择IPV4还是ipv6
type：选择 SOCK_STREAM 是字节流的连接，使用TCP； SOCK_DGRAM 数据包进行链接，使用UDP
protocol:传0表示使用默认协议
参数含义详见 man socket
成功返回新创建的socket的文件描述符，失败，返回-1，设置errno
```

- bind函数：绑定ip和端口号

```
命令：man bind
 int bind(int sockfd, const struct sockaddr *addr,
```

```
                socklen_t addrlen);
                成功返回0，失败返回-1
  sockaddr:现在一般初始化sockaddr_in 结构，但是使用的时候还是强制转换成sockaddr *结构
```

- listen函数：用来指定监听上限数（同时允许多少客户端建立连接）
- accept函数：接收连接请求

```
int accept（int sockfd , struct sockaddr * addr , socklen_t * addrlen ）;
返回值为链接到的客户端的fd
```

- read函数

```
ssize_t read(int fd, void *buf, size_t count);
fd是客户端的fd
```

-
-
-

四、服务器和客户端

TCP

重要：socket 客户端服务器端共3个

1、server.c

- socket():建立套接字
- bind():绑定IP和端口号(struct sockaddr_in addr 初始化)
- listen():设置同时能够建立链接的最大数目
- accept():等待接收链接请求，返回的是链接到的客户端的套接字。参数2：是一个传出参数指的是客户端的地址；参数3是传入传出参数，表示客户端地址的长度。因为由传入参数的作用，所以使用accept函数之前需要初始化。
- read()：函数中第一个参数指的是客户端的套接字句柄。
- 将读取的数据进行处理
- write()：回写给客户端
- read()
- close

2、client.c

- socket()
- bind():可以依赖“隐式绑定”
- connect();发起链接

```
int connect(int sockfd, const struct sockaddr *addr,
                    socklen_t addrlen);
```

- write（）：主动发送数据
- read（）：将服务器处理完的数据读回来
- close（）

```c
//server.c
#include <stdio.h>
#include <unistd.h>//对于类Unix系统，unistd.h中所定义的接口通常都是大量针对系统调用的封装（英
#include <sys/socket.h>
#include <stdio.h>

#include <ctype.h>//定义了一批C语言字符分类函数（C character classification functions），用
#include <arpa/inet.h>//提供IP地址转换函数
#define server_port 6666
#define server_ip 192.168.135.153
int socket_fd;
int client_fd;
struct sockaddr_in server_addr,client_addr;
socklen_t client_addr_len;
char buf[BUFSIZ];
int n,i;
int main()
{
    socket_fd = socket(AF_INET,SOCK_STREAM,0);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(server_port);
    //server_addr.sin_addr.s_addr = inet_pton(INADDR_ANY);//监听外部客户端程序发送到服务器端
    server_addr.sin_addr.s_addr = INADDR_ANY;//自动取出系统中有效的任意IP地址：二进程类型
    bind(socket_fd,(struct sockaddr *)&server_addr,sizeof(server_addr));
    listen(socket_fd,128);
    client_addr_len = sizeof(client_addr);//这是一个传入传出参数，所以这里初始化一次
    client_fd=  accept(socket_fd,(struct sockaddr *)&client_addr,&client_addr_len);
    n = read(client_fd,buf,sizeof(buf));
    for(i=0;i<n;n++){
        buf[i] =toupper(buf[i]);
    }
    write(client_fd,buf,n);//写回给客户端
    close(socket_fd);
    close(client_fd);
    return 0;
}
```

2、客户端：