# 网络编程第一天

## 学习目标

* 了解OSI七层、TCP/IP四层模型结构
* 了解常见网络协议格式
* 掌握网络字节序和主机字节序之间的转换（大端法和小端法）
* 说出TCP服务器端通信流程
* 说出TCP客户端通信流程
* 独立写出TCP服务器端代码
* 独立写出TCP客户端代码

## 1 网络基础概念

### 1.1 协议

概念: 协议事先约定好, 大家共同遵守的一组规则, 如交通信号灯.

从应用程序的角度看, 协议可理解为数据传输和数据解释的规则;

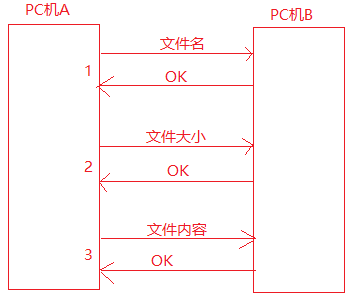
可以简单的理解为各个主机之间进行通信所使用的共同语言.

假设，A、B双方欲传输文件。规定：

第一次: 传输文件名，接收方接收到文件名，应答OK给传输方；

第二次: 发送文件的尺寸，接收方接收到该数据再次应答一个OK；

第三次: 传输文件内容。同样，接收方接收数据完成后应答OK表示文件内 容接收成功。



这种在A和B之间被遵守的协议称之为原始协议, 后来经过不断增加完善改进, 最终形成了一个稳定的完整的传输协议, 被广泛应用于各种文件传输, 该协议逐渐就成了一个标准协议.

几种常见的协议

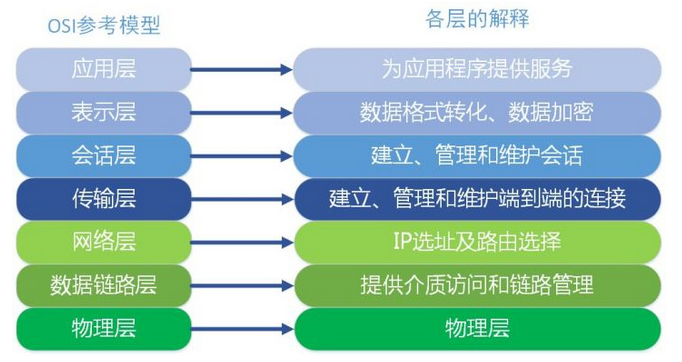
见讲义部分

### 1.2分层模型

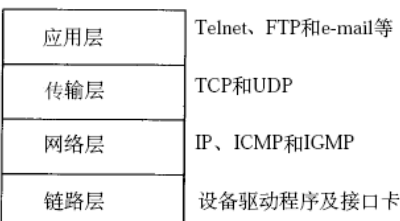
OSI是Open System Interconnection的缩写, 意为开放式系统互联. [国际标准化组织](https://baike.so.com/doc/5340499-5575942.html" \t "https://baike.so.com/doc/_blank)(ISO)制定了OSI模型, 该模型定义了不同计算机互联的标准, 是设计和描述计算机网络通信的基本框架.

网络分层 OSI 7层模型: 物数网传会表应

* 物理层---双绞线，光纤（传输介质），将模拟信号转换为数字信号
* 数据链路层---数据校验，定义了网络传输的基本单位-帧
* 网络层---定义网络，两台机器之间传输的路径选择点到点的传输
* 传输层---传输数据 TCP，UDP，端到端的传输
* 会话层---通过传输层建立数据传输的通道.
* 表示层---编解码，翻译工作.
* 应用层---为客户提供各种应用服务，email服务，ftp服务，ssh服务



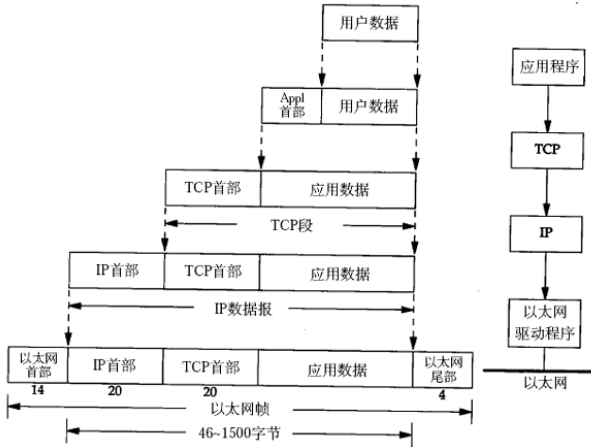


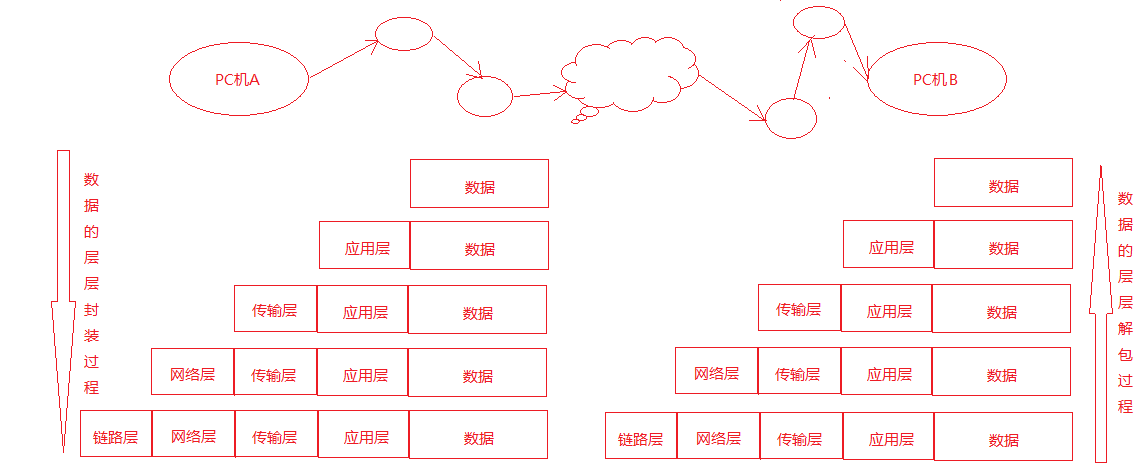


### 1.3 数据通信过程

通信过程: 其实就是发送端层层打包, 接收方层层解包.

注意: 这些操作不是用户自己做的, 而是底层帮我们做好的.





### 1.4 网络应用程序的设计模式

* CS设计模式优缺点：

优点:

客户端在本机上可以保证性能, 可以将数据缓存到本地, 提高数据的传输效率, 提高用户体验效果.

客户端和服务端程序都是由同一个开发团队开发, 协议选择比较灵活.

缺点:

服务器和客户端都需要开发，工作量相对较大, 调试困难, 开发周期长;

从用户的角度看, 需要将客户端安装到用户的主机上, 对用户主机的安 全行构成威胁.

* BS设计模式优缺点：

优点:

无需安装客户端, 可以使用标准的浏览器作为客户端;

只需要开发服务器，工作量相对较小;

由于采用标准的客户端, 所以移植性好, 不受平台限制.

相对安全，不用安装软件

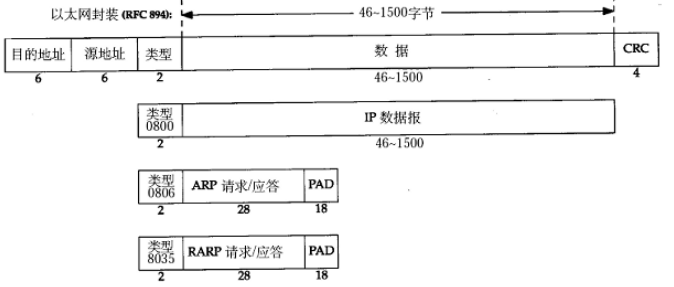
缺点:

由于没有客户端, 数据缓冲不尽人意, 数据传输有限制, 用户体验较差;

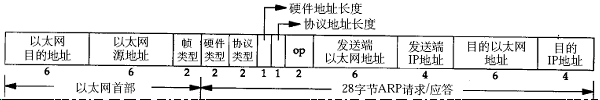
通信协议选择只能使用HTTP协议，协议选择不够灵活;

### 1.5 以太网帧格式

以太网帧格式就是包装在网络接口层(数据链路层)的协议



以ARP为例介绍以太网帧格式



目的端mac地址是通过发送端发送ARP广播, 接收到该ARP数据的主机先判断是否是自己的IP, 若是则应答一个ARP应答报文, 并将mac地址填入应答报文中; 若目的IP不是自己的主机IP, 则直接丢弃该ARP请求报文.

详细讲解ARP协议

IP段格式



协议版本: ipv4, ipv6

16位总长度: 最大65536

8位生存时间ttl(网络连接下一跳的次数): 为了防止网络阻塞

32位源ip地址, 共个4字节！我们熟悉的ip都是点分十进制的，4字节, 每字节对应一个点分位，最大为255 ，实际上就是整形数！

32位目的ip地址

8位协议: 用来区分上层协议是TCP, UDP, ICMP还是IGMP协议.

16位首部校验和: 只校验IP首部, 数据的校验由更高层协议负责.

UDP数据报格式

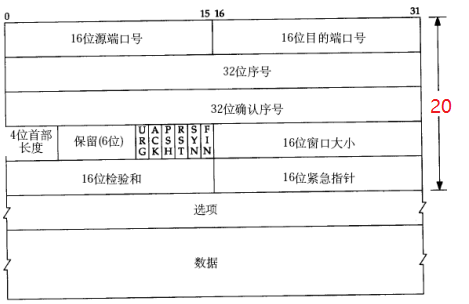


通过IP地址来确定网络环境中的唯一的一台主机;

主机上使用端口号来区分不同的应用程序.

IP+端口唯一确定唯一一台主机上的一个应用程序.

TCP数据流格式:



稳定的, 安全的, 可靠的

序号: TCP是安全可靠的, 每个数据包都带有序号, 当数据包丢失的时候, 需要重传, 要使用序号进行重传. 控制数据有序, 丢包重传.

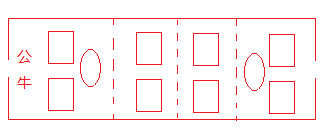
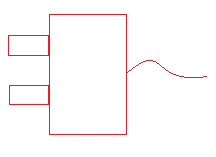
确认序号: 使用确认序号可以知道对方是否已经收到了, 通过确认序号可以知道哪个序号的数据需要重传.

16位窗口大小--滑动窗口(主要进行流量控制)

### 1.6网络名词术语解析(自行阅读扫盲)

## 2 SOCKET编程

传统的进程间通信借助内核提供的IPC机制进行, 但是只能限于本机通信, 若要跨机通信, 就必须使用网络通信.( 本质上借助内核-内核提供了socket伪文件的机制实现通信----实际上是使用文件描述符), 这就需要用到内核提供给用户的socket API函数库.

既然提到socket伪文件, 所以可以使用文件描述符相关的函数read write

可以对比pipe管道讲述socket文件描述符的区别.

使用socket会建立一个socket pair.

如下图, 一个文件描述符操作两个缓冲区, 这点跟管道是不同的, 管道是两个文件描述符操作一个内核缓冲区.



### 2.1 socket编程预备知识

网络字节序:

大端和小端的概念

大端: 低位地址存放高位数据, 高位地址存放低位数据

小端: 低位地址存放低位数据, 高位地址存放高位数据

大端和小端的使用使用场合???

大端和小端只是对数据类型长度是两个及以上的, 如int short, 对于单字节 没限制, 在网络中经常需要考虑大端和小端的是IP和端口.

思考题: 0x12345678如何存放???

如何验证本机上大端还是小端??-----使用共用体.

编写代码endian.c进行测试, 测试本机上是大端模式还是小端模式?

网络传输用的是大端法, 如果机器用的是小端法, 则需要进行大小端的转换.

下面4个函数就是进行大小端转换的函数:

#include <arpa/inet.h>

uint32\_t htonl(uint32\_t hostlong);

uint16\_t htons(uint16\_t hostshort);

uint32\_t ntohl(uint32\_t netlong);

uint16\_t ntohs(uint16\_t netshort);

函数名的h表示主机host, n表示网络network, s表示short, l表示long

上述的几个函数, 如果本来不需要转换函数内部就不会做转换.

IP地址转换函数:

p->表示点分十进制的字符串形式

to->到

n->表示network网络

int inet\_pton(int af, const char \*src, void \*dst);

函数说明: 将字符串形式的点分十进制IP转换为大端模式的网络IP(整形4字节数)

参数说明:

af: AF\_INET

src: 字符串形式的点分十进制的IP地址

dst: 存放转换后的变量的地址

例如: inet\_pton(AF\_INET, "127.0.0.1", &serv.sin\_addr.s\_addr);

手工也可以计算: 如192.168.232.145, 先将4个正数分别转换为16进制数,

192--->0xC0 168--->0xA8 232--->0xE8 145--->0x91

最后按照大端字节序存放: 0x91E8A8C0, 这个就是4字节的整形值.

const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t size);

函数说明: 网络IP转换为字符串形式的点分十进制的IP

参数说明:

af: AF\_INET

src: 网络的整形的IP地址

dst: 转换后的IP地址,一般为字符串数组

size: dst的长度

返回值:

成功--返回执行dst的指针

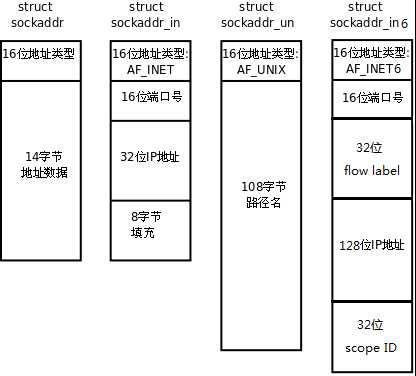
失败--返回NULL, 并设置errno

例如: IP地址为010aa8c0, 转换为点分十进制的格式:

01---->1 0a---->10 a8---->168 c0---->192

由于从网络中的IP地址是高端模式, 所以转换为点分十进制后应该为: 192.168.10.1

socket编程用到的重要的结构体:struct sockaddr



struct sockaddr结构说明:

struct sockaddr {

sa\_family\_t sa\_family;

char sa\_data[14];

}

struct sockaddr\_in结构:

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* address family: AF\_INET \*/

in\_port\_t sin\_port; /\* port in network byte order \*/

struct in\_addr sin\_addr; /\* internet address \*/

};

/\* Internet address. \*/

struct in\_addr {

uint32\_t s\_addr; /\* address in network byte order \*/

}; //网络字节序IP--大端模式

通过man 7 ip可以查看相关说明

### 2.2 socket编程主要的API函数介绍

int socket(int domain, int type, int protocol);

函数描述: 创建socket

参数说明:

domain: 协议版本

AF\_INET IPV4

AF\_INET6 IPV6

AF\_UNIX AF\_LOCAL本地套接字使用

type:协议类型

SOCK\_STREAM 流式, 默认使用的协议是TCP协议

SOCK\_DGRAM 报式, 默认使用的是UDP协议

protocal:

一般填0, 表示使用对应类型的默认协议.

返回值:

成功: 返回一个大于0的文件描述符

失败: 返回-1, 并设置errno

当调用socket函数以后, 返回一个文件描述符, 内核会提供与该文件描述符相对应的读和写缓冲区, 同时还有两个队列, 分别是请求连接队列和已连接队列.



int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

函数描述: 将socket文件描述符和IP,PORT绑定

参数说明:

socket: 调用socket函数返回的文件描述符

addr: 本地服务器的IP地址和PORT,

struct sockaddr\_in serv;

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(8888);

//serv.sin\_addr.s\_addr = htonl(INADDR\_ANY);

//INADDR\_ANY: 表示使用本机任意有效的可用IP

inet\_pton(AF\_INET, "127.0.0.1", &serv.sin\_addr.s\_addr);

addrlen: addr变量的占用的内存大小

返回值:

成功: 返回0

失败: 返回-1, 并设置errno

int listen(int sockfd, int backlog);

函数描述: 将套接字由主动态变为被动态

参数说明:

sockfd: 调用socket函数返回的文件描述符

backlog: 同时请求连接的最大个数(还未建立连接)

返回值:

成功: 返回0

失败: 返回-1, 并设置errno

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

函数说明:获得一个连接, 若当前没有连接则会阻塞等待.

函数参数:

sockfd: 调用socket函数返回的文件描述符

addr: 传出参数, 保存客户端的地址信息

addrlen: 传入传出参数, addr变量所占内存空间大小

返回值:

成功: 返回一个新的文件描述符,用于和客户端通信

失败: 返回-1, 并设置errno值.

accept函数是一个阻塞函数, 若没有新的连接请求, 则一直阻塞.

从已连接队列中获取一个新的连接, 并获得一个新的文件描述符, 该文件描述符用于和客户端通信. (内核会负责将请求队列中的连接拿到已连接队列中)

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

函数说明: 连接服务器

函数参数:

sockfd: 调用socket函数返回的文件描述符

addr: 服务端的地址信息

addrlen: addr变量的内存大小

返回值:

成功: 返回0

失败: 返回-1, 并设置errno值

接下来就可以使用write和read函数进行读写操作了.

除了使用read/write函数以外, 还可以使用recv和send函数

读取数据和发送数据:

ssize\_t read(int fd, void \*buf, size\_t count);

ssize\_t write(int fd, const void \*buf, size\_t count);

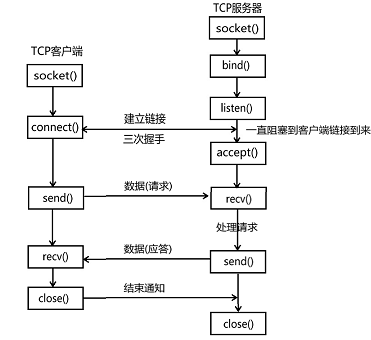
ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

对应recv和send这两个函数flags直接填0就可以了.

注意: 如果写缓冲区已满, write也会阻塞, read读操作的时候, 若读缓冲区没有数据会引起阻塞.

使用socket的API函数编写服务端和客户端程序的步骤图示:



根据服务端和客户端编写代码的流程, 编写代码并进行测试.

测试过程中可以使用netstat命令查看监听状态和连接状态

netstat命令:

a表示显示所有,

n表示显示的时候以数字的方式来显示

p表示显示进程信息(进程名和进程PID)

作业:

自己编写代码熟悉一下服务端和客户端的代码开发流程;

设计服务端和客户端通信协议(属于业务层的协议)

如发送结构体

typedef struct teacher\_{

int tid;

char name[30];

int age;

char sex[30];

int sal;

} teacher;

typedef struct student\_{

int sid;

char name[30];

int age;

char sex[30];

}student;

typedef struct SendMsg\_{

int type;//1 - teacher;2 - student

int len;//

char buf[0];//变长发送数据

}SendMsg;