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，以分层的形式有机地结合在一起。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。下面我们将介绍两类非常重要的体系结构：OSI与TCP/IP。

OSI开放系统互联模型：

国际标准化组织(ISO)制定了OSI模型。这个模型把网络通信的工作分为7层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

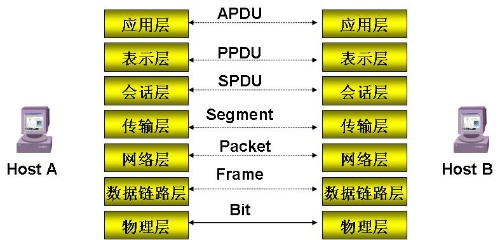
TCP/IP模型：

TCP/IP协议模型将OSI协议模型简化为4层，从而更有利于实现和使用。1983年，TCP/IP协议成为Internet的“世界语”。

OSI与TCP/IP模型的对应关系如下图：

|  |
| --- |
| 应用层 |
| 表示层 |
| 会话层 |
| 传输层 |
| 网络层 |
| 数据链路层 |
| 物理层 |

1. 网络接口层：负责将二进制流(比特流)转换为数据帧，并进行数据帧的发送和接收；
2. 网际层：负责在主机之间的通信中选择数据包的传输路径，即路由；
3. 传输层：负责提供应用程序之间的通信服务，即端到端的通信。



TCP/IP协议族：

TCP------Transfer Control Protocol传输控制协议

IP------Internetworking Protocol网际协议

UDP------User Datagram Protocol用户数据报协议

ICMP------Internet Control Message Protocol互联网控制报文协议

SMTP------Simple Mail Tranfer Protocol简单邮件传输协议

SNMP------ Simple Network Management Protocol简单网络管理协议

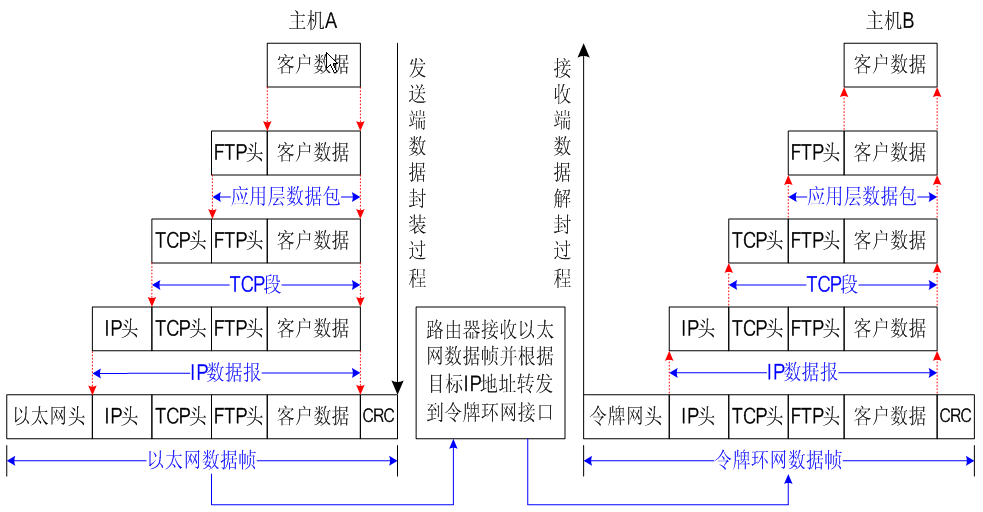
HTTP------Hypertext Tranfer Protocol超文本传输协议

FTP------File Tranfer Protocol文件传输协议

ARP------Address Resolution Protocol地址解析协议

……

数据的封装与传递过程：



**TCP与UDP**

TCP和UDP都是传输层的协议，但不同点在于：

1. TCP：面向连接，可靠；
2. UDP：无连接，不可靠。

TCP：

TCP(即传输控制协议)，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：

1. 适合于对传输质量要求较高，以及传输大量数据的通信；
2. TCP会有较大的时延，不适合对实时性要求较高的场合；
3. MSN/QQ等即时通讯软件的用户登录账户的功能通常采用TCP协议。

3次握手链接：

TCP是面向连接的，所谓面向连接，就是当计算机双方通信时必须先建立连接，然后进行数据通信，最后拆除连接。TCP在建立连接时分三步走：

设：客户端的初始序列号为J，服务器的初始序列号为K

1. 客户通过调用connect进行主动连接。客户端发送一个包含SYN即同步标志的TCP报文，SYN同步报文会指明客户端的初始序列号 J，并进入SYN\_SEND状态，等待服务器的确认。
2. 服务器在接收到客户端的SYN报文后，将返回一个SYN+ACK的报文，表示客户端的请求被接受，同时TCP客户端序列号被加1(J+1)，此时服务器进入SYN\_RECV状态 。
3. 客户收到服务器的SYN+ACK，也返回一个ACK报文给服务器，同样TCP服务器序列号被加1(K+1)，到此，一个TCP连接完成。

UDP：

UDP(即用户数据报协议)，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，所以可进行高效率的数据传送。

适用情况：

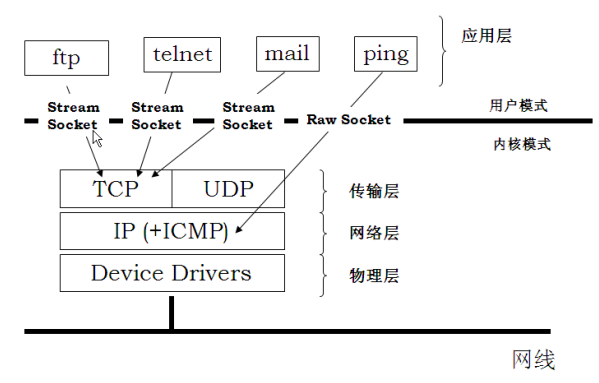
1. 发送小尺寸数据(如对DNS服务器进行IP地址查询时)；
2. 适合于广播/组播式通信中；
3. MSN/QQ等即时通讯软件的点对点文本通讯以及语音/视频通讯常采用UDP协议；
4. 流媒体、VOD、IPTV等网络多媒体服务中通常采用UDP协议进行实时数据传输。

套接字(socket)简介

1. 套接字定义

在Linux中的网络编程是通过socket接口来进行的。套接字socket是一种特殊的I/O接口，它也是一种文件描述符。socket是一种常用的进程间通信机制，通过它不仅能实现本地机器上的进程之间的通信，而且通过网络能在不同机器上的进程之间进行通信。

1. 套接字的位置



在OSI参考模型中，流式套接字和数据报套接字位于会话层和传输层之间；原始套接字位于会话层和网络层之间。

1. 套接字的类型
2. 流式套接字SOCK\_STREAM

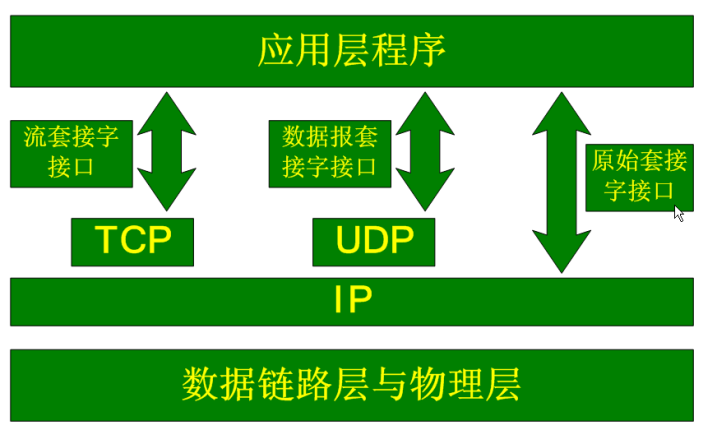
流式套接字提供可靠的、面向连接的数据流，它使用TCP协议，从而保证了数据传输的可靠性和顺序性。

1. 数据报套接字SOCK\_DGRAM

数据报套接字定义了一种不可靠、无连接的服务，数据通过相互独立的报文进行传输，它使用的是UDP协议。

1. 原始套接字SOCK\_RAW

原始套接字允许对底层协议如IP或ICMP进行直接访问，它功能强大但使用较为不便，主要用于一些协议的开发。



IP地址

1. Internet中的主机要与别的机器通信必须具有一个IP地址；
2. IP地址为32位(IPv4)或128位(IPv6)；
3. 每个数据包都必须携带目的IP地址和源IP地址，路由器依靠此信息为数据包选择路由；
4. 表示形式：点分十进制表示。

ip地址的转换：

头文件：#include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h>

函数原型：

1. int inet\_aton(const char \*cp, struct in\_addr \*inp);

将cp所指向的点分十进制表示形式的ip地址转换为网络字节序的二进制值，并存储于inp所指向的结构体中。

1. in\_addr\_t inet\_addr(const char \*cp);

功能同上，返回转换后的ip地址。

typedef unsigned int uint32\_t;

typedef uint32\_t in\_addr\_t;

1. char \*inet\_ntoa(struct in\_addr in);

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。

注意：与地址有关的数据类型和结构体

1. 通用地址结构

struct sockaddr {

u\_short sa\_family;//地址族

char sa\_data[14];//14字节协议地址，包含该socket的ip地址和端口号

};

typedef unsigned short u\_short;

1. internet协议地址结构

struct sockaddr\_in {

u\_short sin\_family;//地址族

u\_short sin\_port;//端口号占2 bytes(1~65535)

struct in\_addr sin\_addr;//IPv4地址

char sin\_zero[8];//8 bytes填充为0以保持与struct sockaddr同样大小

};

本地地址结构为struct sockaddr\_un。

1. IPv4地址结构

struct in\_addr {

in\_addr\_t s\_addr;//u32 network address

};

端口号：

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处理，使用端口号来区别。端口号由IANA管理：

1. 1~255为众所周知的端口号；
2. 256~1023端口通常由UNIX系统占用；
3. 1024~49151为已登记端口；
4. 49152~65535为用户可用端口。(216=65536,2bytes)

字节序：

1. 主机字节序HBO：大端序big-endian和小端序little-endian；
2. 网络字节序NBO：大端字节序。

注意：

big-endian：低字节存放于高地址，高字节存放于低地址；

little-endian：低字节存放于低地址，高字节存放于高地址。

当应用进程将整数送入socket之前，需要转化为网络字节序；当应用进程从socket取出整数后，需要将其转化为主机字节序。

字节序转换函数：

1. 主机字节序到网络字节序：

u\_short htons(u\_short short);

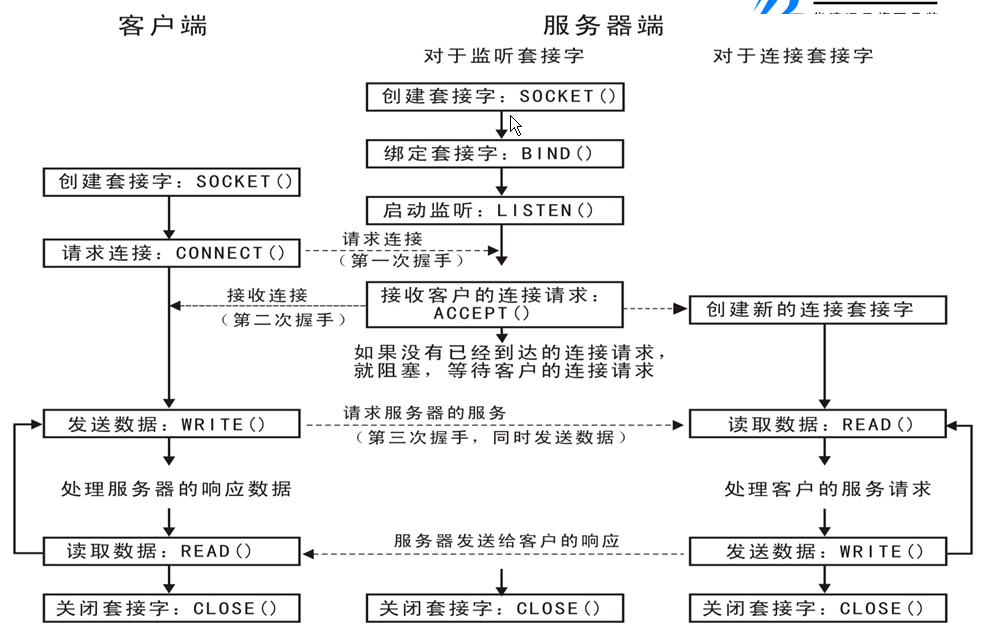
u\_long htonl(u\_long long);

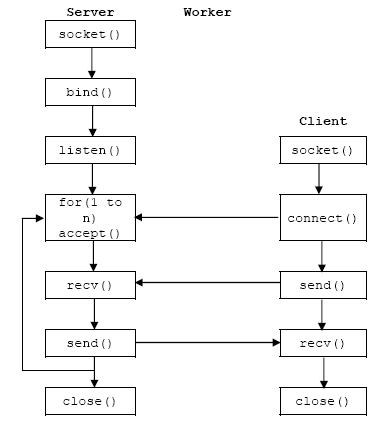
1. 网络字节序到主机字节序：

u\_short ntohs(u\_short short);

u\_long ntohl(u\_long long);

TCP客户端&服务器流程：





网络编程相关API

socket()：------创建套接字

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | int socket(int domain, int type, int protocol); |
| 参数 | domain：AF\_INET、AF\_UNIX等 |
| type：SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW |
| protocol：通常为0，当type为SOCK\_RAW时，该参数有其他取值 |
| 返回值 | 成功：套接字描述符sockfd |
| 失败：-1(errno) |

bind()：------绑定本机ip地址和端口

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | int bind(int sockfd, struct sockaddr \*addr, socklen\_t addrlen); |
| 参数 | sockfd：socket的返回值 |
| addr：要绑定的地址结构指针 |
| addrlen：地址结构长度 |
| 返回值 | 成功：0 |
| 失败：-1(errno) |

listen()：------设置监听套接字(TCP服务器端)

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h> #include <sys/socket.h> |
| 原型 | int listen(int sockfd, int backlog); |
| 参数 | sockfd：listen调用成功后，sockfd即为监听套接字 |
| backlog：指定了正在等待连接的最大队列长度，它的作用在于处理可能同时出现的多个连接请求 |
| 返回值 | 成功：0 |
| 失败：-1(errno) |

DoS(拒绝服务)攻击即利用了这个原理，非法的连接占用了全部的连接数，造成正常的连接请求被拒绝。

accept()：------接受连接(TCP服务器端)

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); |
| 参数 | sockfd：监听套接字描述符sockfd |
| addr：对方地址结构指针 |
| addrlen：对方地址结构长度指针，用户必须对其进行初始化 |
| 返回值 | 成功：连接套接字描述符 |
| 失败：-1(errno) |

connect()：------请求(建立)连接(TCP客户端)

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen); |
| 参数 | sockfd：客户端套接字描述符sockfd |
| addr：对方地址结构指针 |
| addrlen：对方地址结构长度 |
| 返回值 | 成功：0 |
| 失败：-1(errno) |

send()：------发送数据(同write)

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);  ssize\_t write(int fd, const void \*buf, size\_t count); |
| 参数 | sockfd：套接字描述符sockfd |
| buf：发送缓冲区首地址 |
| len：要发送的字节数 |
| flags：发送方式，通常为0 |
| 返回值 | 成功：实际发送的字节数 |
| 失败：-1(errno) |

recv()：------接收数据(同read)

|  |  |
| --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> |
| 原型 | ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);  ssize\_t read(int fd, void \*buf, size\_t count); |
| 参数 | sockfd：套接字描述符sockfd |
| buf：接收缓冲区首地址 |
| len：接收字节数 |
| flags：接收方式，通常为0 |
| 返回值 | 成功：实际接收的字节数 |
| 失败：-1(errno) |

还有sendto()函数和recvfrom()函数，同样可以用于发送和接收数据，这两个函数一般用于利用UDP协议通信的场合。

网络属性的获取和设置：

头文件： #include <sys/types.h> #include <sys/socket.h>

原型：

1. int getsockopt(int sockfd, int level, int optname, void \*optval, socklen\_t \*optlen);
2. int setsockopt(int sockfd, int level, int optname, void \*optval, socklen\_t optlen);

参数：

1. sockfd：套接字文件描述符；
2. level：指定套接字的层次：SOL\_SOCKET, IPPROTO\_IP, IPPROTO\_TCP；
3. optname：选项名称；
4. optval存放选项值的缓冲区地址；
5. optlen：选项值长度(指针)。

注意：

1. 我们可以利用setsockopt()允许sockfd发送广播或加入多播组；
2. 可以利用getsockopt()获取tcp接收缓冲区的大小，会发现其大小是有一个上限和下限的，且一般情况下，其大小为我们设置值的2倍。

gethostbyname()：

作用：进行域名解析，将域名解析为对应的ip地址。

头文件：#include <netdb.h>

原型：struct hostent \*gethostbyname(const char \*name);

参数：name：指向域名字符串的指针；

返回值：结构体指针。

The hostent structure is defined in <netdb.h> as follows:

struct hostent {

char \*h\_name; /\* official name of host \*/官方名

char \*\*h\_aliases; /\* alias list \*/

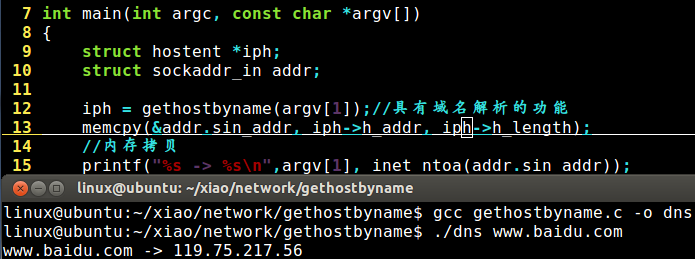
int h\_addrtype; /\* host address type \*/ip类型

int h\_length; /\* length of address \*/ip地址长度

char \*\*h\_addr\_list; /\* list of addresses \*/

}

#define h\_addr h\_addr\_list[0] /\* for backward compatibility \*/



套接字的关闭：

int close(int sockfd);

关闭双向通讯。

TCP连接是双向的(可读写)，当我们使用close时，会把读写通道都关闭，有时我们希望只关闭一个方向，可以使用shutdown函数。

注意：

1. 基于TCP的通讯，既有发送缓冲区，又有接收缓冲区，send和recv均会发生阻塞；当写端关闭时，若无数据，读端recv会返回0，这一点与管道很像；
2. 基于UDP的通讯，有接收缓冲区，无发送缓冲区，所以sento不会阻塞，但recvfrom会发生阻塞；当写端关闭时，且无数据时，recvfrom同样会返回0。