

**socket 类型**

常见的socket有3种类型如下。

（1）流式socket（SOCK\_STREAM ）

流式套接字提供可靠的、面向连接的通信流；它使用TCP 协议，从而保证了数据传输的正确性和顺序性。

（2）数据报socket（SOCK\_DGRAM ）

数据报套接字定义了一种无连接的服 ，数据通过相互独立的报文进行传输，是无序的，并且不保证是可靠、无差错的。它使用数据报协议UDP。

（3）原始socket（SOCK\_RAW）

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

**socket创建和连接**

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

family指定协议族；type参数指定socket的类型：SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW；protocol通常赋值"0"。

socket()调用返回一个整型socket描述符，你可以在后面的调用使用它。

一旦通过socket调用返回一个socket描述符，你应该将该socket与你本机上的一个端口相关联（往往当你在设计服务器端程序时需要调用该函数。随后你就可以在该端口监听服务请求;而客户端一般无须调用该函数）。

int bind(int sockfd, struct sockaddr \*my\_addr, int addrlen);

sockfd是一个socket描述符，my\_addr是一个指向包含有本机IP地址及端口号等信息的sockaddr类型的针; addrlen常被设置为sizeof(struct sockaddr)。 最后，对于bind 函数要说明的一点是，你可以用下面的赋值实现自动获得本机IP地址和随机获取一个没有被占用的端口号：

my\_addr.sin\_port = 0; /\* 系统随机选择一个未被使用的端口号 \*/

my\_addr.sin\_addr.s\_addr = INADDR\_ANY; /\* 填入本机IP地址 \*/

通过将my\_addr.sin\_port置为0，函数会自动为你选择一个未占用的端口来使用。同样，通过将my\_addr.sin\_addr.s\_addr置为INADDR\_ANY，系统会自动填入本机IP地址。bind()函数在成功被调用时返回0；遇到错误时返回"-1"并将errno置为相应的错误号。另外要注意的是，当调用函数时，一般不要将端口号置为小于1024的值，因为1~1024是保留端口号，你可以使用大于1024中任何一个没有被占用的端口号。

当对TCP/IP协议族的套接字进行绑定时，我们通常使用另一个地址结构：

struct sockaddr\_in

{

short sin\_family;

u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8];

};

其中sin\_family置AF\_INET；sin\_port指明端口号；sin\_addr结构体中只有一个唯一的字段s\_addr，表示IP地址，该字段是一个整数，一般用函数inet\_addr（）把字符串形式的IP地址转换成unsigned long型的整数值后再置给s\_addr。有的服务器是多宿主机，至少有两个网卡，那么运行在这样的服务器上的服务程序在为其socket绑定IP地址时可以把htonl(INADDR\_ANY)置给s\_addr，这样做的好处是不论哪个网段上的客户程序都能与该服务程序通信；如果只给运行在多宿主机上的服务程序的socket绑定一个固定的IP地址，那么就只有与该IP地址处于同一个网段上的客户程序才能与该服务程序通信。我们用0来填充sin\_zero数组，目的是让sockaddr\_in结构的大小与sockaddr结构的大小一致。下面是一个bind函数调用的例子：

struct sockaddr\_in saddr;

memset((void \*)&saddr,0,sizeof(saddr));

saddr.sin\_family = AF\_INET;

saddr.sin\_port = htons(8888);

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

//saddr.sin\_addr.s\_addr = inet\_addr("192.168.22.5"); 绑定固定IP

bind(ListenSocket,(struct sockaddr \*)&saddr,sizeof(saddr));

int listen(int sockfd， int backlog);

sockfd是socket系统调用返回的服务器端socket描述符；backlog指定在请求队列中允许的最大请求数，进入的连接请求将在队列中等待accept()它们（参考下文）。backlog对队列中等待服务的请求的数目进行了限制，大多数系统缺省值为20。当listen遇到错误时返回-1，errno被置为相应的错误码。

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

sockfd是被监听的服务器socket描述符，addr通常是一个指向sockaddr\_in变量的指针，该变量用来存放提出连接请求的客户端地址；addrten通常为一个指向值为sizeof(struct sockaddr\_in)的整型指针变量。错误发生时返回一个-1并且设置相应的errno值。accept()函数将返回一个新的socket描述符，来供这个新连接来使用，在新的socket描述符上进行数据send()和recv()操作。

故服务器端程序通常按下列顺序进行函数调用：

socket(); bind(); listen(); /\* accept() goes here \*/

connect()函数用来与远端服务器建立一个TCP连接其函数原型为：

int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen);

sockfd是目的服务器的sockt描述符；serv\_addr是服务器端的IP地址和端口号的地址。遇到错误时返回-1，并且errno中包含相应的错误码。进行客户端程序设计无须调用bind()，因为这种情况下只需知道目的机器的IP地址，而客户通过哪个端口与服务器建立连接并不需要关心，内核会自动选择一个未被占用的端口供客户端来使用。

UDP的“connect”：

1、程序可以使用connect实现UDP连接套接字，作用是在UDP套接字中记住目的地址和目的端口。

2、UDP套接字使用connect后，如果数据报不是connect中指定的地址和端口，将被丢弃。没有调用connect的UDP套接字，将接收所有到达这个端口的UDP数据报，而不区分源端口和地址。

关于“bind”：

1、client端的socket不需要bind，内核会自动选择一个未被占用的port供client来使用，如果有多个可用的连接（多个IP），内核会根据优先级选择一个IP作为源IP使用。

2、如果socket使用bind绑定到特定的IP和port，则无论是TCP还是UDP，都会从指定的IP和port发送数据。

**socket发送与接收数据**

send()和recv()——数据传输，用于面向连接的socket（SOCK\_STREAM）上进行数据传输

int send(int sockfd, const void \*msg, int len, int flags);

sockfd是你想用来传输数据的socket描述符，msg是一个指向要发送数据的指针。

len是以字节为单位的数据的长度。flags一般情况下置为0（关于该参数的用法可参照man手册）。

send()函数返回实际上发送出的字节数，可能会少于你希望发送的数据。所以需要对send()的返回值进行测量。当send()返回值与len不匹配时，应该对这种情况进行处理。

int recv(int sockfd,void \*buf,int len,unsigned int flags);

sockfd是接受数据的socket描述符；buf 是存放接收数据的缓冲区；len是缓冲的长度。flags也被置为0。recv()返回实际上接收的字节数，或当出现错误时，返回-1并置相应的errno值。

带外数据：在数据流信道之外的信道上传输的数据，常用于对远端进程的同步和控制。在TCP中一次只能发送1字节的带外数据。

发送：send(sock\_fd,'f',1,MSG\_OOB);

接收：recv(sock\_fd,&out\_data,1,MSG\_OOB); 带外数据存储在out\_data中。

sendto() 和 recgfrom() 数据传输，用于面向非连接socket(SOCK\_DGRAM/SOCK\_RAW)上进行数据传输

在无连接的数据报socket方式下，由于本地socket并没有与远端机器建立连接，所以在发送数据时应指明目的地址，sendto()函数原型为：

int sendto(int sockfd, const void \*msg,int len,unsigned int flags,const struct sockaddr \*to,int tolen);

该函数比send()函数多了两个参数，to表示目地机的IP地址和端口号信息，而tolen常常被赋值为sizeof (struct sockaddr)。sendto 函数也返回实际发送的数据字节长度或在出现发送错误时返回-1。

int recvfrom(int sockfd,void \*buf,int len,unsigned int flags,struct sockaddr \*from,int \*fromlen);

from是一个struct sockaddr类型的变量，该变量保存源机的IP地址及端口号。fromlen常置为sizeof(struct sockaddr)。当recvfrom()返回时，fromlen包含实际存入from中的数据字节数。Recvfrom()函数返回接收到的字节数或当出现错误时返回-1，并置相应的errno。

应注意的一点是，当你对于数据报socket调用了connect()函数时，你也可以利用 send()和recv()进行数据传输，但该socket仍然是数据报socket，并且利用传输层的UDP服务。但在发送或接收数据报时，内核会自动为之加上目地和源地址信息。

**关闭socket**

close()和shutdown()——结束数据传输

当所有的数据操作结束以后，你可以调用close()函数来释放该socket，从而停止在该socket上的任何数据操作：

close(sockfd); close()是对套接字的操作，关闭后进程不能在访问这个套接字。

你也可以调用shutdown()函数来关闭该socket。该函数允许你只停止在某个方向上的数据传输，而一个方向上的数据传输继续进行。如你可以关闭某socket的写操作而允许继续在该socket上接受数据，直至读入所有数据。shutdown是对TCP连接的操作。

int shutdown(int sockfd,int how);

sockfd的含义是显而易见的，而参数 how可以设为下列值：

　　·0-------不允许继续接收数据

　　·1-------不允许继续发送数据

　　·2-------不允许继续发送和接收数据，均为允许则调用close()

shutdown在操作成功时返回0，在出现错误时返回-1（并置相应errno）。

IP DNS 等相关函数

in\_addr\_t inet\_addr(const char \* strptr);

将字符串IP地址转换为IPv4地址结构in\_addr值

char \* inet\_ntoa(struct in\_addr \* addrptr);

将IPv4地址结构in\_addr值转换为字符串IP

域名和IP地址的转换：

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

函数返回一种名为hostent的结构类型，它的定义如下：

struct hostent

{

char \*h\_name; /\* 主机的官方域名 \*/

char \*\*h\_aliases; /\* 一个以NULL结尾的主机别名数组 \*/

int h\_addrtype; /\* 返回的地址类型，在Internet环境下为AF-INET \*/

int h\_length; /\*地址的字节长度 \*/

char \*\*h\_addr\_list; /\* 一个以0结尾的数组，包含该主机的所有地址\*/

　　};

#define h\_addr h\_addr\_list[0] /\*在h-addr-list中的第一个地址\*/

注意：以上三个函数都是不可重入的，如果你写下如下代码：

if(strcmp( inet\_ntoa(ip1), inet\_ntoa(ip2) ) == 0 ) //判断2个IP地址是否相同

{

.... ....

}

上面if条件判断永远为真！在使用上面三个函数时，函数返回后，要马上取出结果保存返回值，否则会被下次调用覆盖。因为struct in\_addr 和 struct hostent 在保存时使用了static类型。

<http://www.programminglogic.com/sockets-programming-in-c-using-udp-datagrams/>