网络编程

# 套接字描述符

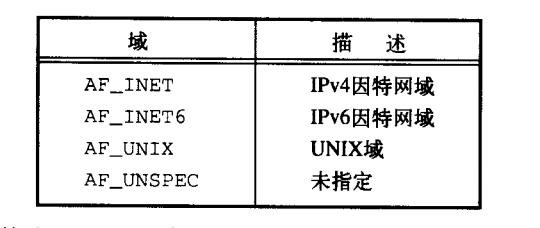
套接字是通信端点的抽象。访问套接字需要使用套接字描述符。套接字描述符在Unix系统是用文件描述符实现的。

## 创建套接字

调用socket函数，创建套接字。

|  |  |
| --- | --- |
| 函数原型 | #include <sys/socket.h>  int socket(int domain, int type, int protocol);  返回值：若成功则返回文件(套接字)描述符，若出错则返回-1 |
| domain | 确定通信的特性，包含地址格式 |
| type | 确定套接字的类型，进一步确定通信特征 |
| protocol | 通常是0，表示按给定的域和套接字类型选择默认协议。当对同一域和套接字类型支持多个协议时，可以使用protocol参数选择一个特定协议 |

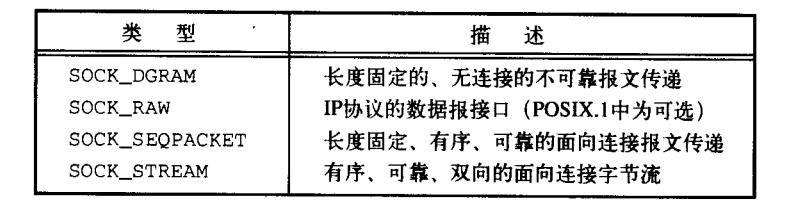
### domain(域)



由POSIX.1指定的各个域。各个域有自己的格式表示地址，而表示各个域的常数都以AF\_开头，意指地址族(address family)。

多数系统还会定义AF\_LOCAL域，这是AF\_UNIX的别名。AF\_UNSPEC域可代表任何域。

### type(套接字类型)



由POSIX.1定义的套接字类型。

#### SOCK\_DGRAM

数据包接口提供了一个无连接的服务，与对方通信时是不需要逻辑连接的。只需要送出一个报文，其地址是一个对方进程所使用的的套接字。

数据报是一种自包含报文。发送数据报近似于给人寄邮件。可以邮寄很多信，但不能保证投递的次序，并且可能有些信件丢失在路上。每封邮件包含接收者的地址，使这封邮件独立于所有其他邮件。每封邮件可能送到不同的接收者。

#### SOCK\_STREAM

字节流(SOCK\_STREAM)要求在交换数据之前，在本地套接字和与之通信的远程套接字之间建立一个逻辑连接。

使用面向连接的协议通信就像与对方打电话。首先，需要通过电话建立一个连接，连接建立好之后，彼此能双向的通信。每个连接都是端到端的通信信道。会话中不包含地址信息，就像呼叫的两端存在一个点对点虚拟连接，并且连接本身暗含特定的源和目的地。

对于SOCK\_STREAM套接字，因为套件字提供的是字节流服务，应用程序意识不到报文界限。这意味着当从套接字读出数据时，它也许不会返回所有由发送进程所写的字节数。最终可以获得发送过来的所有数据，但也许要通过若干次函数调用得到。

#### SOCK\_SEQPACKET

SOCK\_SEQPACKET套接字类似SOCK\_STREAM，但从该套接字得到的是基于报文的服务而不是字节流服务。这意味着从SOCK\_SEQPACKET套接字接收的数据量与对方所发送的一致。流控制传输协议(Stream Control Transmission Protocol, SCTP)提供了因特网域上的顺序数据包服务。

#### SOCK\_RAW

SOCK\_RAW套接字提供一个数据报接口用于直接访问下面的网络层(在因特网域中为IP)。使用这个接口时，应用程序负责构造自己的协议首部，这是因为传输协议(TCP和UDP等)被绕过了。当创建一个原始套接字时需要有超级用户特权，用以防止恶意程序绕过内建安全机制来创建报文。

## 关闭套接字

调用close来关闭对套接字的访问，并且释放该描述符以便重新使用。

## 禁止套接字输入输出

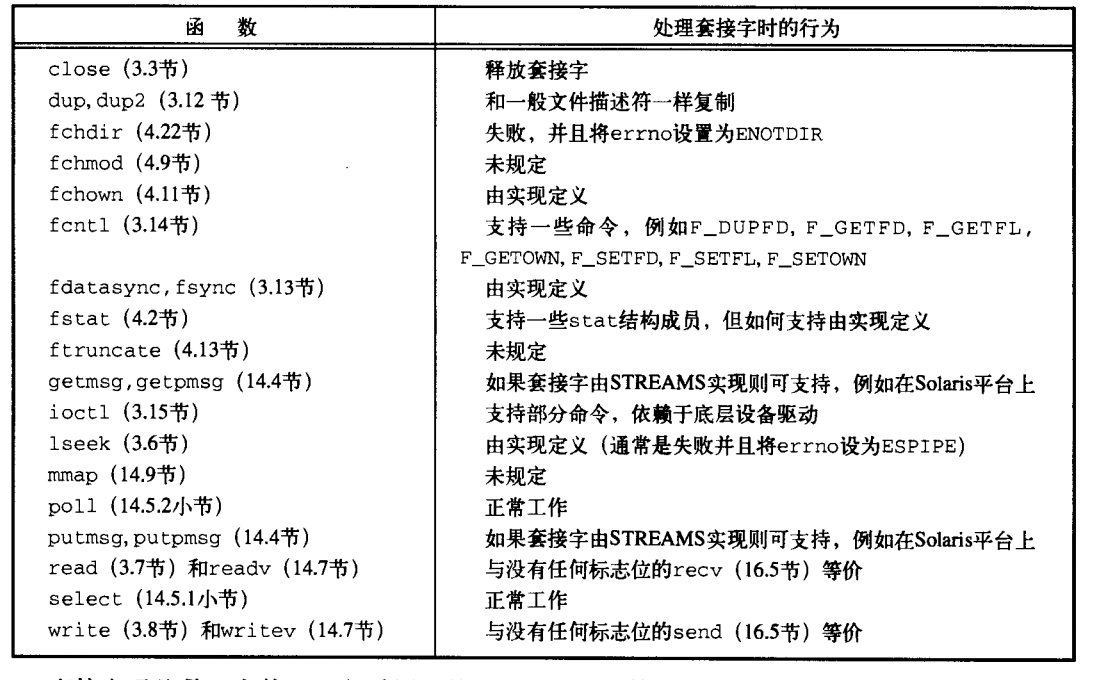
套接字通信是双向的。可以采用函数shutdown来禁止套接字上的输入输出。

|  |  |
| --- | --- |
| 函数原型 | #include <sys/socket.h>  int shutdown(int sockfd, int how);  返回值：成功返回0，出错返回-1 |
| sockfd | 套接字描述符 |
| how | SHUT\_RD：关闭读端，无法从套接字读取数据；  SHUT\_WR：关闭写端，无法使用套接字发送数据；  SHUT\_RDWR：无法读取和发送数据 |

能够close套接字，还使用shutdown的理由：首先，close只有在最后一个活动引用被关闭时才释放网络端点。这意味着如果复制一个套接字（例如采用dup），套接字直到关闭了最后一个引用它的文件描述符之后才会被释放。而shutdown允许使一个套接字处于不活动状态，无论引用它的文件描述符数目多少。其次，有时只关闭套接字双向传输中的一个方向会很方便。

## 其他处理函数

虽然套接字描述符本质上是一个文件描述符，但不是所有参数为文件描述符的函数都可以接受套接字描述符。参加下图：



# 寻址

## 字节序

字节序是一个处理器架构特性，用于指示像整数这样的大数据类型的内部字节顺序。

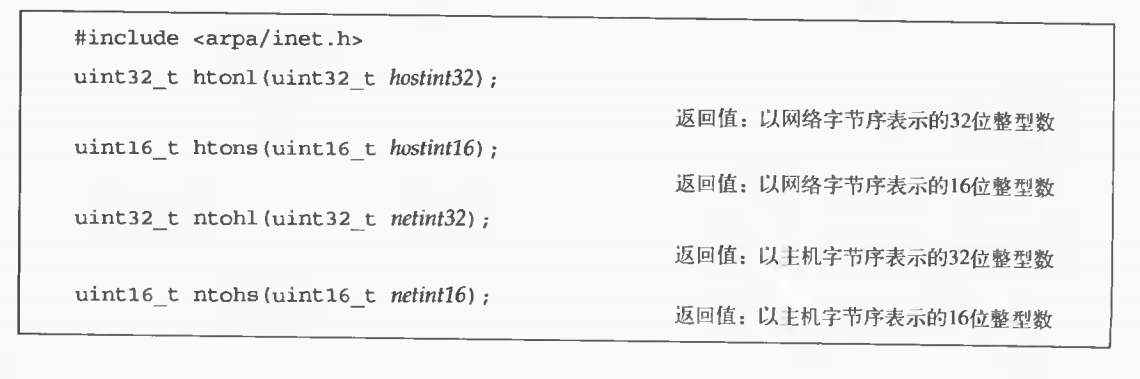
如果处理器架构支持大端(big-endian)字节序，那么最大字节地址对应于数字最低有效字节(LSB)上；小端(little-endian)字节序则相反：数字最低字节对应于最小字节地址。

不管字节如何排序，数字最高位总是在左边，最低位总是在右边。

例如有一个32位整数值0x04030201，数字最高位包含4，数字最低位包含1。不同处理器的排序如下：

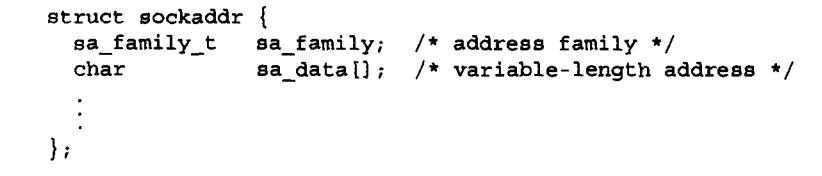
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | cp[0] | cp[1] | cp[2] | cp[3] |
| 大端字节序 | 04 | 03 | 02 | 01 |
| 小端字节序 | 01 | 02 | 03 | 04 |

网络协议指定了字节序。TCP/IP协议栈采用大端字节序。处理器字节序与网络字节序之间的转换函数为：

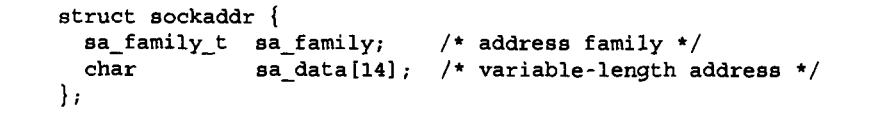


## 地址格式

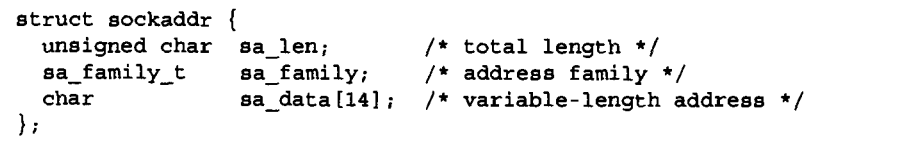
地址标识了特定通信域中的套接字端点，地址格式与特定的通信域相关。为使不同格式地址能够被传入到套接字函数，地址被强制转换成通用的地址结构sockaddr表示：



套接字实现可以自由地添加额外的成员并且定义sa\_data成员的大小。例如在Linux中，该结构定义如下：

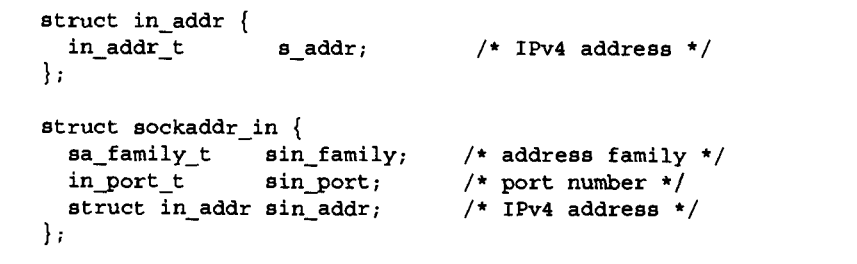


而在FreeBSD中，该结构定义如下：



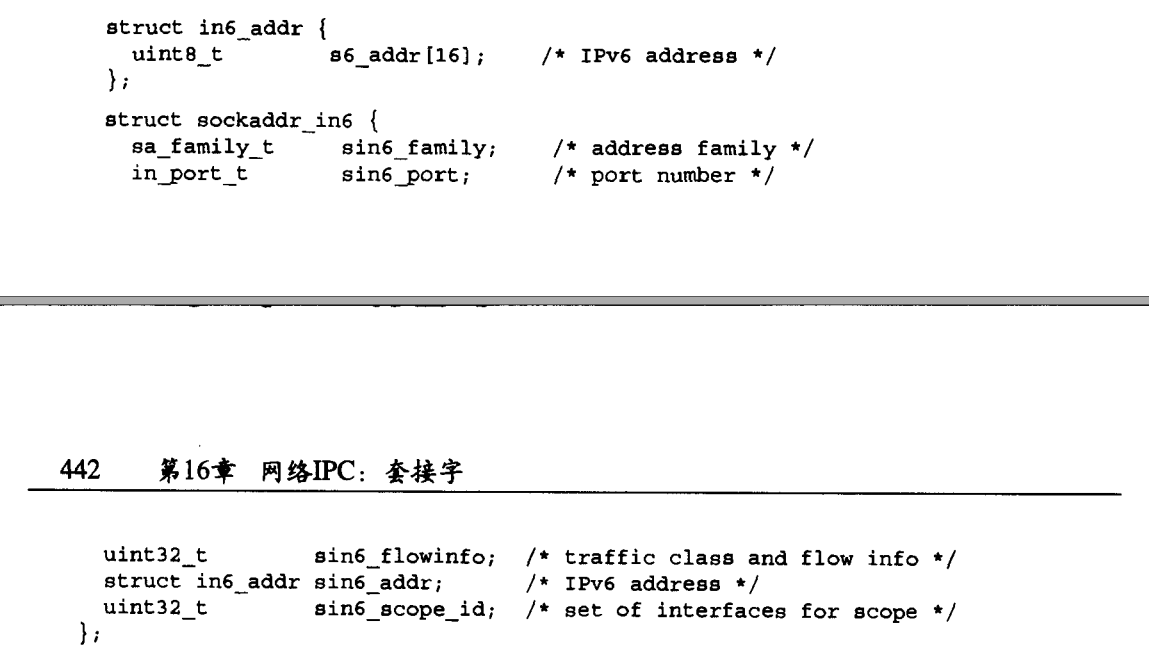
### 因特网地址

因特网地址定义在<netinet/in.h>中。在IPv4因特网域(AF\_INET)中，套接字地址用如下结构sockaddr\_in表示：

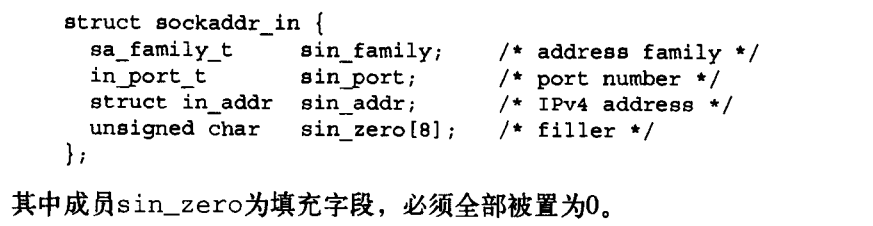


in\_addr\_t定义成uint32\_t，in\_port\_t定义成uint16\_t。

在IPv6因特网域(AF\_INET6)中，套接字地址用如下结构sockaddr\_in6表示：



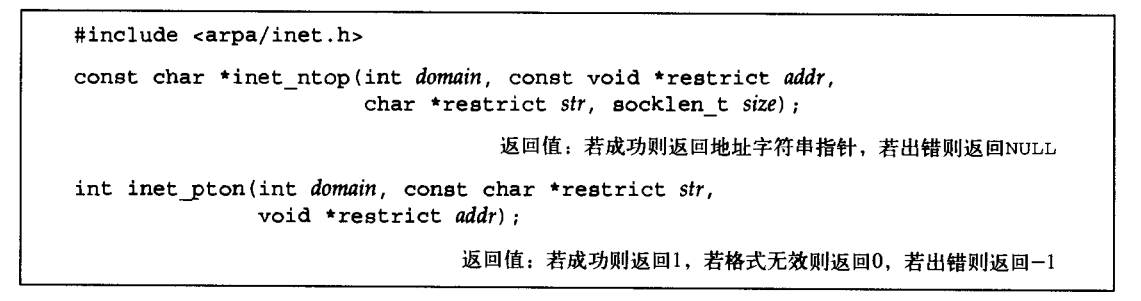
这些是Single UNIX Specification必须的定义，每个实现可以自由地添加额外的字段。例如，在Linux中，sockaddr\_in定义如下：



### 地址格式转换函数

BSD网络软件中包含了函数inet\_addr和inet\_ntoa，用于在二进制地址格式与点分十进制字符串表示(a.b.c.d)之间相互转换。这些函数仅用于IPv4地址。

功能相似的两个新函数inet\_ntop和inet\_pton支持IPv4和IPv6地址。



参数domain仅支持两个值：AF\_INET、AF\_INET6。

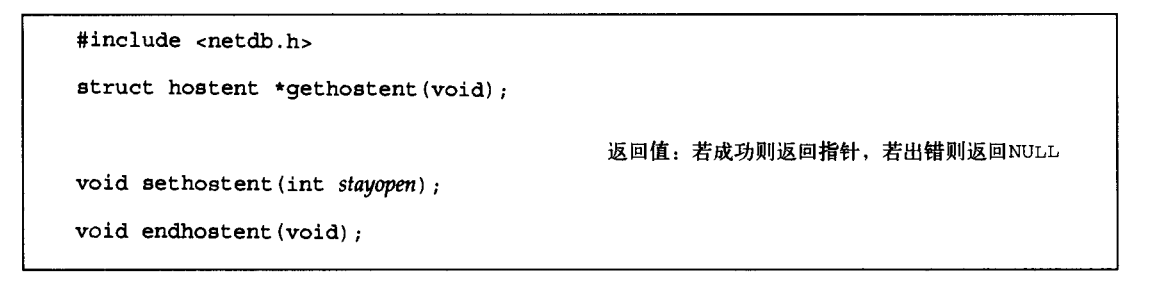
对于inet\_ntop，参数size指定了用以保存文本字符串的缓冲区str的大小。两个常数用于简化工作：INET\_ADDRSTRLEN定义了足够大的空间来存放表示IPv4地址的文本字符串，INET6\_ADDRSTRLEN定义了足够大的空间来存放表示IPv6地址的文本字符串。

对于inet\_pton，如果domain是AF\_INET，缓冲区addr需要有足够大的空间来存放32位地址；如果domain是AF\_INET6，缓冲区addr需要有足够大的空间来存放128位地址。

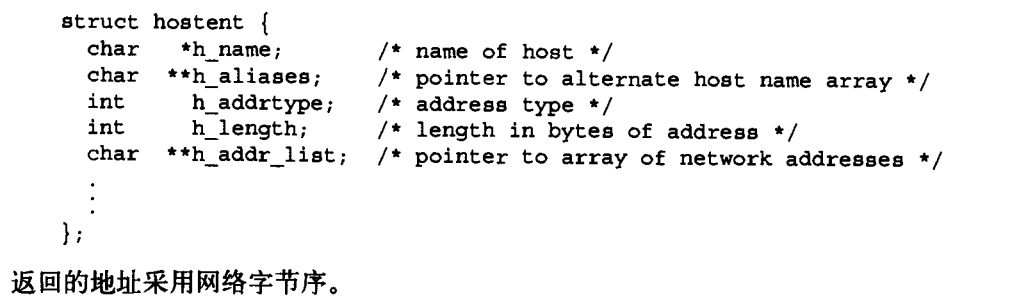
### 地址查询

理想情况下，应用程序不需要了解套接字地址的内部结构。

#### gethostent: 获取给定计算机的主机信息



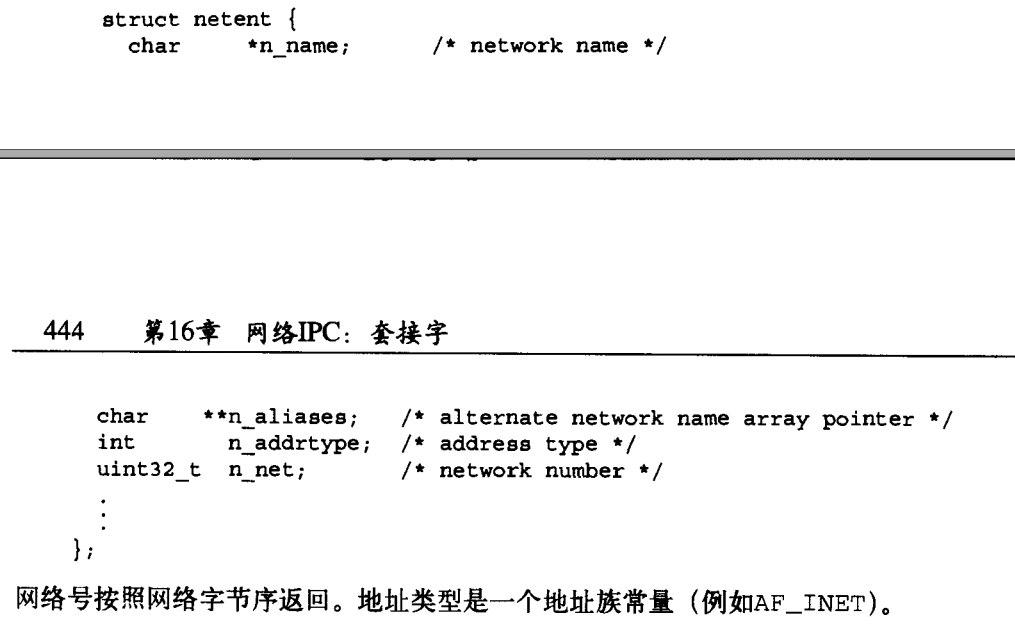
hostent结构可能包含一个静态的数据缓冲区。每次调用gethostent将会覆盖这个缓冲区。数据结构至少包含如下成员：



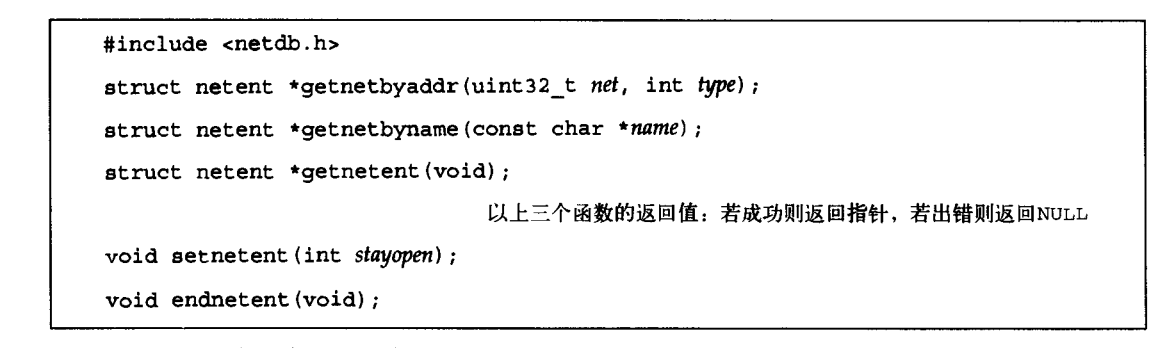
如果主机数据文件没有打开，gethostent会打开它。函数gethostent返回文件的下一个条目。函数sethostent会打开文件，如果文件已经打开，那么将其回绕。函数endhostent将关闭文件。

#### 获取网络名字和网络号

结构netent至少包含如下成员：

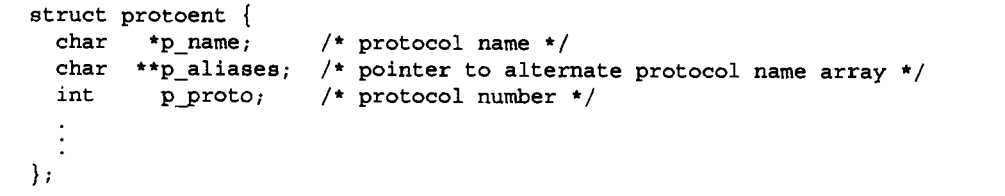


获取接口：

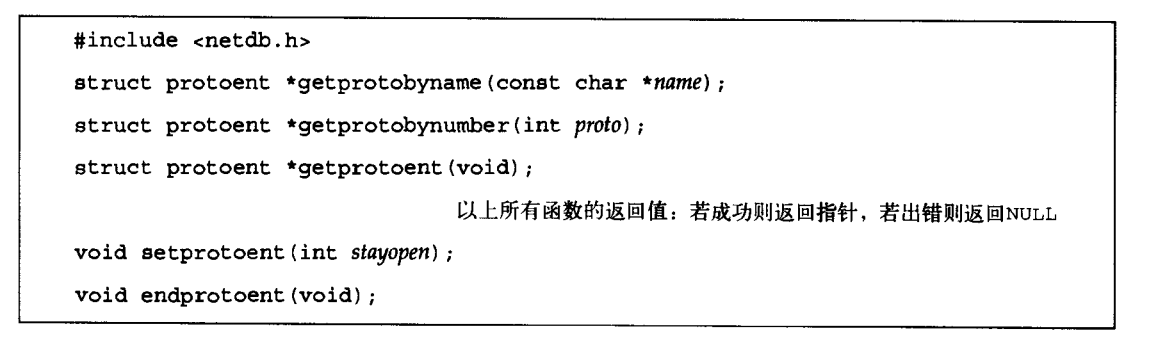


#### 协议名字和协议号映射

POSIX.1定义的结构protoent至少包含如下成员：

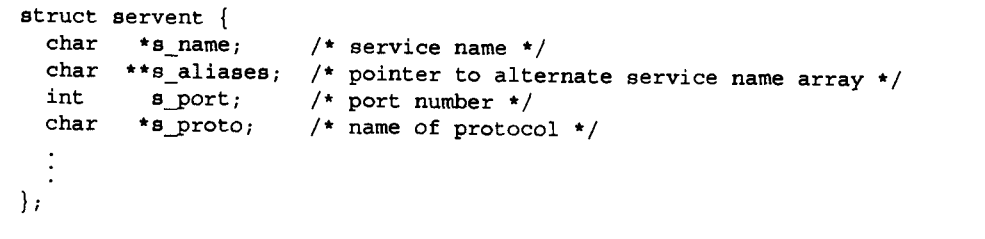


映射接口：

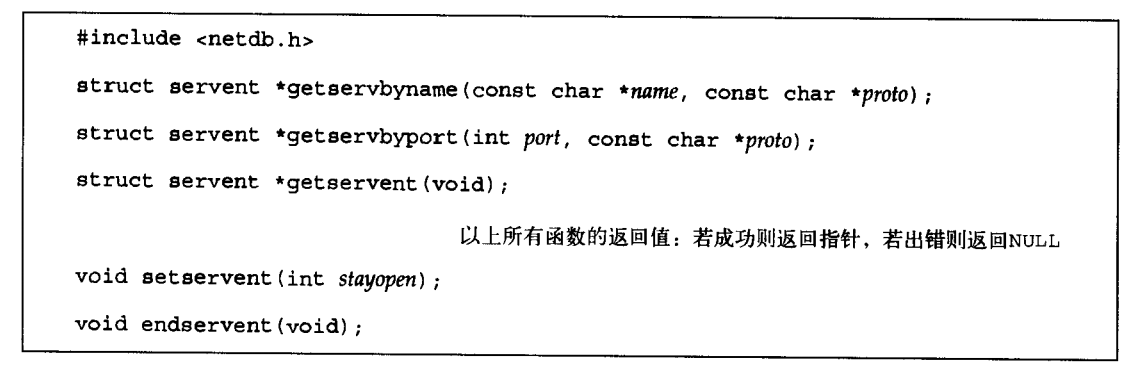


#### 服务名字和端口号映射

结构servent至少包含如下成员：



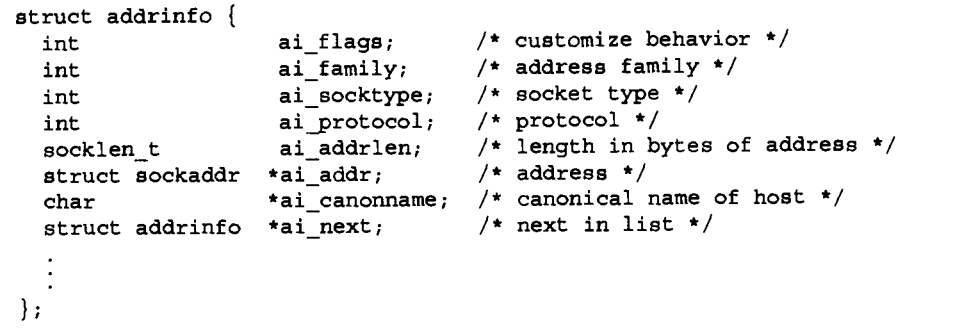
映射接口：



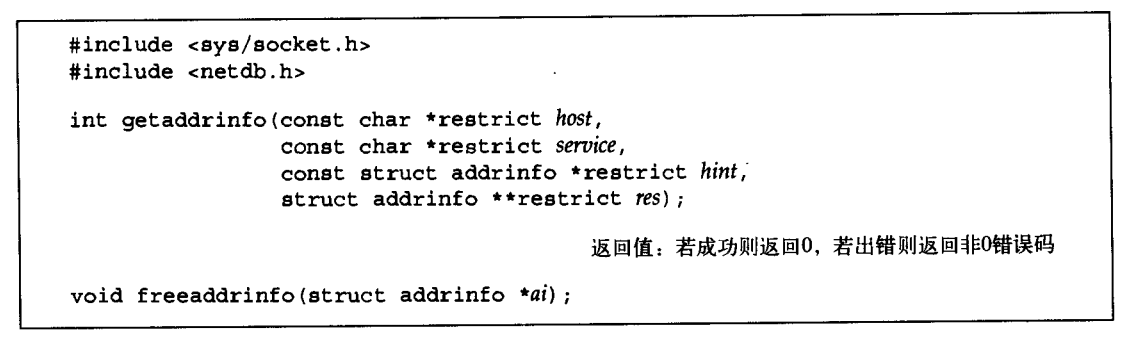
服务是由地址的端口号部分表示的。每个服务由一个唯一的、熟知的端口号来提供。采用函数getservbyname可以将一个服务名字映射到一个端口号。函数getservbyport将一个端口号映射到一个服务名。函数getservent顺序扫描服务数据库。

#### 服务名和主机名映射到地址

结构addrinfo的定义至少包含如下成员：

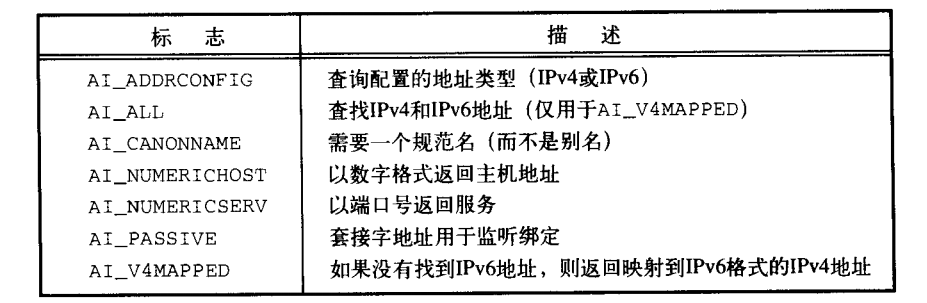


映射接口：

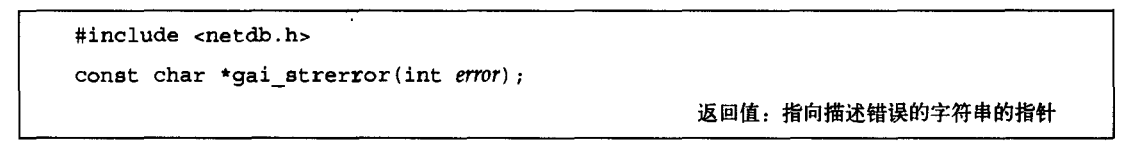


需要提供主机名字、服务名字，或者两者都提供。如果仅仅提供一个名字，另外一个必须是一个空指针。主机名字可以是一个节点名或点分十进制记法表示的主机地址。

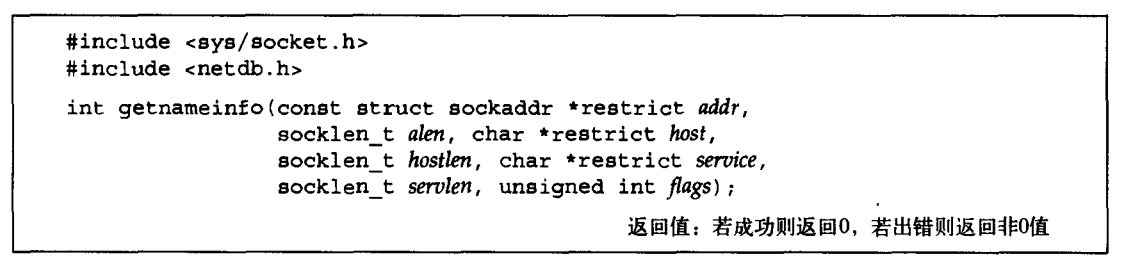
根据某些规则，可以提供一个可选的hint来选择地址。hint是一个用于过滤地址的模板，仅适用sa\_family、ai\_flags、ai\_protocol和ai\_socktype字段。剩余的整数字段必须设为0，并且指针字段为空。下表列出了ai\_flags中所用的标志：



如果getaddrinfo失败，不能使用perror或strerror来生成错误信息。应该使用gai\_strerror将返回的错误码转换成错误信息。

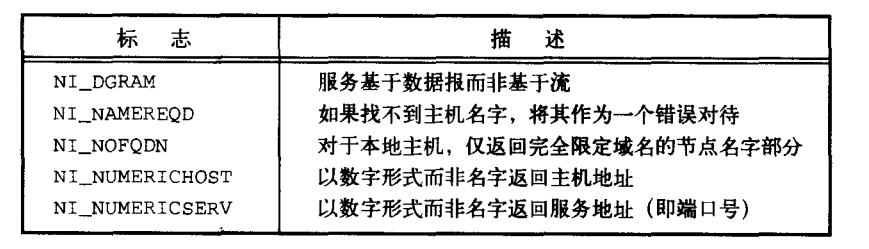


#### 将地址转换为主机名或者服务名



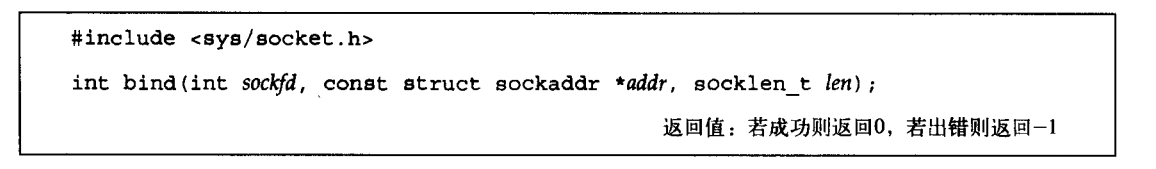
套接字地址addr被转换为主机名或服务名。如果host非空，它指向一个长度为hostlen字节的缓冲区用于存储返回的主机名。如果service非空，它指向一个长度为servlen字节的缓冲区用于存储返回的服务名。

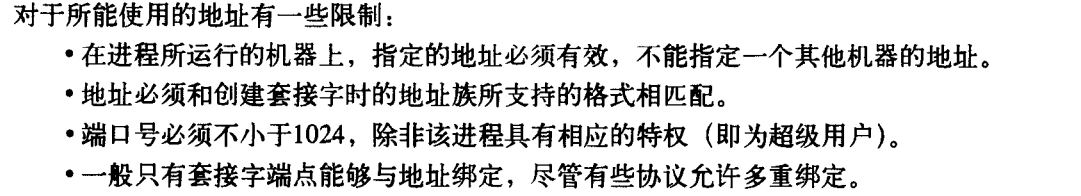
下表列出了flags中所用的标志：



### 将套接字与地址绑定

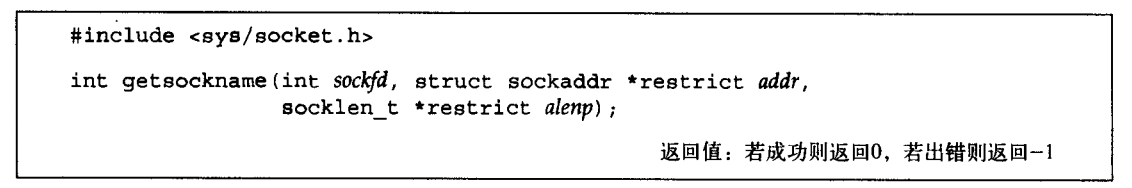
用bind函数将地址绑定到一个套接字。

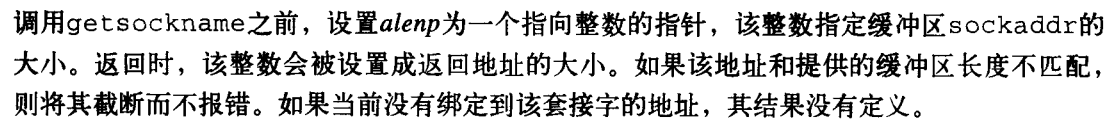




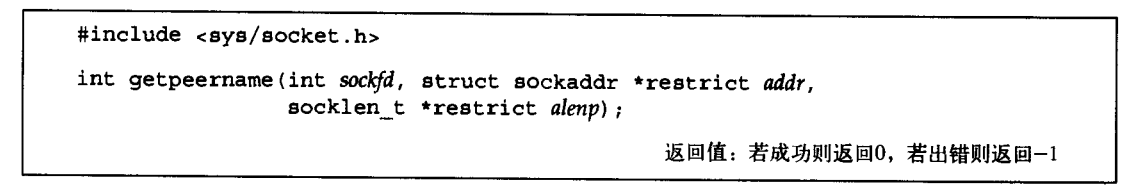
对于因特网域，如果指定IP地址为INADDR\_ANY，套接字端点可以被绑定到所有的系统网络接口。这意味着可以收到这个系统所安装的所有网卡的数据包。如果调用connect或listen，但没有绑定地址到一个套接字，系统会选一个地址并将其绑定到套接字。

#### getsockname:获取绑定到套接字的地址





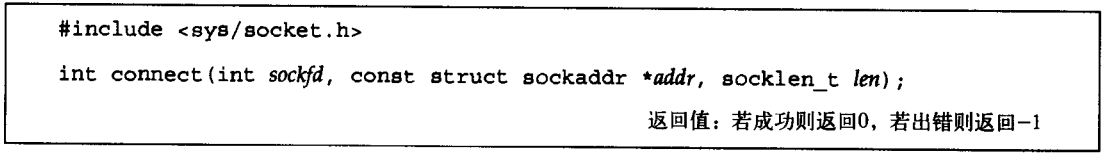
#### getpeername:获取对方的地址



如果套接字已经和对方连接，调用getpeername找到对方的地址。

# 建立连接

处理面向连接的网络服务(SOCK\_STREAM或SOCK\_SEQPACKET)时，开始交换数据之前，需要在客户端套接字与服务端套接字之间建立一个连接，由客户端发起连接调用。



connect中的addr参数是想与之通信的服务器地址。如果sockfd没有绑定到一个地址，connect会给调用者绑定一个默认地址。

当连接一个服务器时，连接可能失败。要连接的机器必须开启并且正在运行，服务器必须绑定到一个想与之连接的地址，并且在服务器的等待连接队列中应有足够的空间。因此，必须处理connect返回的错误。