Windows 2000 应用系列

Windows 2000 编程

李多多 等 编著

人 民 邮 电 出 版 社

需要全本请访问 www.ertongbook.com

第 16 章　Windows Socket 程序设计

当今计算机和通信网络的发展和扩张是时代发展的主流。网络化是计算机技术在 20 世纪 90 年代的重要发展趋势之一。TCP/IP 协议在网络互联中占据了重要的地位，以它为基础组建的 Internet 是目前国际上规模最大的计算机网间网。

与计算机网络的普及相呼应的是视窗操作系统的广泛应用。在 Windows 下的各种网络编程接口中，Windows Sockets 脱颖而出，成为当前程序设计的主要手段之一。

Windows Socket 是 Windows 2000 提供的一个最重要的通信程序设计机制。它来源于 BSD（UNIX 上的 Berkeley Software Distribution）版本套接字，并为 Windows 进行了专门的扩展，能够支持 TCP/IP 等常见的许多网络协议。

Windows Sockets 规范是一套开放的、支持多种协议的 Windows 下的网络编程接口，已成为 Windows 网络编程事实上的标准。

16.1 Windows Socket 的概述

不管是想要用使用 Windows 套接字 API 还是使用套接字类来编写应用程序，都要首先理解套接字（Winsock）。

Windows Sockets 规范是建立在 Bekeley 套接口模型上的。这个模型现在已是 TCP/IP 网络的标准。它提供了习惯于 UNIX 套接口编程的程序员极为熟悉的环境，并且简化了移植现有的基于套接口的应用程序源代码的工作。Windows Sockets API 也是和 4.3BSD 的要求一致的。

Windows Socket（WinSock）是微软的窗口系统结构（WOSA）的一部分。它的特定接口可以用来开发客户机应用程序。

Windows Sockets 以 U.C. Berkeley 大学 BSD UNIX 中流行的 Socket 接口为范例定义了一套 Micosoft Windows 下网络编程接口。它不仅包含了人们所熟悉的 Berkeley Socket 风格的库函数，也包含了一组针对 Windows 的扩展库函数，以使程序员能充分地利用 Windows 消息驱动机制进行编程。

套接字的应用环境是所谓的因特网协议集（IPS）。IPS 因特网协议集是由一系列协议组成的，这些协议包括：

(1) 地址转换协议（ARP）：将网络地址转换为硬件地址。

(2) 网间控制报文协议（ICMP）：在 TCP/IP 网络中传输控制信息和差错信息。

(3) 反向地址转换协议（RARP）：将硬件地址转换为网络地址。

(4) 传输控制协议（TCP）：提供网间的可靠的数据流传输，TCP 是面向连接的协议，在交换数据前必须先建立连接。TCP 确保数据能到达目的地，如果传输过程出错则重发。

(5) 用户数据报协议（UDP）：UDP 是无连接的协议，在数据报发送时它不验证连接的合法性，故它是不可靠的，对方有可能接受不到数据包。

(6) 因特网协议（IP）：为 TCP、UDP 和 ICMP 提供数据包服务。

在以上的几个协议中，有两个是我们马上就要接触到的：传输控制协议（TCP）和用户数据报协议（UDP）。其中传输控制协议（TCP）应用在将要讲述到的流式套接字中，而用户数据报协议（UDP）应用在数据报套接字中。

Windows Sockets 规范本意在于提供给应用程序开发者一套简单的 API，并让各家网络软件供应商共同遵守。此外，在一个特定版本 Windows 的基础上，Windows Sockets 也定义了一个二进制接口（ABI），以此来保证应用 Windows Sockets API 的应用程序能够在任何网络软件供应商的符合 Windows Sockets 协议的实现上工作。因此这份规范定义了应用程序开发者能够使用，并且网络软件供应商能够实现的一套库函数调用和相关语义。

遵守这套 Windows Sockets 规范的网络软件，即被称做 Windows Sockets 所兼容的软件。一个网络软件供应商必须百分之百地实现 Windows Sockets 规范才能做到 Windows Sockets 兼容。

任何能够与 Windows Sockets 兼容实现协同工作的应用程序都被认为具有 Windows Sockets 接口。这种应用程序被称为 Windows Sockets 应用程序。

Windows Sockets 规范定义了如何使用 API 与 Internet 协议族（IPS，通常指的是 TCP/IP）连接，尤其要指出的是所有的 Windows Sockets 实现都支持流套接口和数据报套接口。

应用程序调用 Windows Sockets 的 API 以实现相互之间的通信。Windows Sockets 又利用下层的网络通信协议功能和操作系统调用实现实际的通信工作。

虽然我们并不反对使用这一套 API 来实现另一通信协议栈（而且我们期望在将来规范的修改中能够讨论这个问题），但这种用法已经超出了我们这一份规范所规定的范围，我们在此将不作讨论。

实际上，套接字主要提供了一种发送和接收数据的机制。

所谓套接字，实际上是通信端点的另一种提法（很新鲜吧），一个套接字就是一个通信端点。典型的通信发生在一个客户机和一个服务器之间，此时存在两个端点，一个在客户机，一个在服务器，这就对应着两个套接字，这两个套接字将建立起客户机和服务器之间数据双向传送的连接。

每个套接字都有一个套接字地址和一个套接字句柄。套接字地址通常为运行程序的计算机的 IP 地址（除非运行应用程序的机器用多个网络卡连到不同网络上）。套接字句柄是 Windows 定义的一种特殊的数据类型，称为 SOCKET。SOCKET 句柄在概念上与窗口的句柄 HWND 相同。

套接字将使用一个端口（PORT）与其他套接字应用程序通信。不同的窗口中可以同时有多个通信功能在执行。这就要求有一种机制能够确保双方的数据不会混杂在一起，而这种确保分离的机制就是端口。FTP 是公用的通信功能，它将使用一个保留的端口（保留的

端口号参见 WinSock2.h），而套接字应用程序需要指定一个未被使用过的端口，或传递 0 作为端口号参数，以自动分配端口号。

每个套接字将以下面两个方式之一进行操作：

- 阻塞方式；
- 无阻塞方式。

套接字基本上分为如下两类：

- 流套接字；
- 数据报套接字。

下面将分别对“流套接字”和“数据报套接字”予以说明。

1．流套接字

流式套接字提供了一种可靠的面向连接的数据传输方法，适用于大量数据的双向传输。它可以传送单个的数据报和数据包，流套接字都提供了一种流式的数据传输。流套接字使用传输控制协议（TCP）。

当需要发送大批量数据时或需要让发出的数据顺序无重复地到达目地地时，流套接字是最好的选择。

注意，当使用流套接字时，若数据传输连接被断开，应用程序会得到通知。

2．数据报套接字

数据报套接字使用用户数据报协议（UDP）。它提供了一种不可靠的、非连接的数据包(PACKET)通信方式。在这种方式下，发送一个数据包不能保证该数据包按照发送的顺序到达。

数据报套接字常可用在轻负载（lightly loaded）的局域网上。此时当互相连接的两台机器上的进程之间进行通信时，数据包不会发生不被发送或没按顺序到达或重复发送的情形。但当在复杂的网络如 Internet 下编写通信应用程序时，数据包套接字的不可靠性就应该要加以考虑。

在 Windows 2000 下，套接字版本号是 WinSock2.0，它具有如下一些特性：

- 重叠的 I/O 和分散/聚合能力；
- 多点传送；
- 独立的域名解析；
- QOS。

这一套 Windows Sockets API 能够在所有 3.0 以上版本的 Windows 和所有 Windows Scokets 实现上使用，所以它不仅为 Windwos Sockets 实现和 Windows Sockets 应用程序提供了 16 位操作环境，而且也提供了 32 位操作环境。

Windows Sockets 也支持多线程的 Windows 进程。一个进程包含了一个或多个同时执行的线程。在 Windows 3.1 非多线程版本中，一个任务对应了一个仅具有单个线程的进程。而我们在本书中所提到的线程均是指在多线程 Windows 环境中的真正意义的线程。在非多线程环境中（例如 Windows 3.0）这个术语是指 Windows Sockets 进程。

Windows Sockets 规范中的针对 Windows 的扩展部分为应用程序开发者提供了开发具有

Windows 应用软件的功能。它有利于使程序员写出更加稳定并且更加高效的程序，也有助于在非占先 Windows 版本中使多个应用程序在多任务情况下更好地运作。除了 WSAStartup() 和 WSACleanup()两个函数除外，其他的 Windows 扩展函数的使用不是强制性的。

总而言之，Windows Sockets 是一份独立的规范。它的产生和存在是为了造益于应用程序开发者、网络软件供应商和广大计算机用户。这份规范的每一份正式出版的版本（非草稿）实际上代表了为网络软件供应商实现所需和应用程序开发者所用的一整套 API。

16.2 套接字编程基础

本节将要介绍使用套接字编程所必须具有的一些基本知识。

16.2.1 通信和客户端/服务器模型

通信的基石是套接口，一个套接口是通信的一端。在这一端上可以找到与其对应的一个名字。一个正在被使用的套接口都有它的类型和与其相关的进程。套接口存在于通信域中。通信域是为了处理一般的线程通过套接口通信而引进的一种抽象概念。套接口通常和同一个域中的套接口交换数据（数据交换也可能穿越域的界限，但这时一定要执行某种解释程序）。Windows Sockets 规范支持单一的通信域，即 Internet 域。各种进程使用这个域和 Internet 协议族来进行互相之间的通信。

套接口可以根据通信性质分类；这种性质对于用户是可见的。应用程序一般仅在同一类的套接口间通信。不过只要底层的通信协议允许，不同类型的套接口间也照样可以通信。

如前所述，目前可以使用两种套接口，即流套接口和数据报套接口。其中流套接口提供了双向、有序、无重复且无记录边界的数据流服务。数据报套接口支持双向的数据流，但并不保证是可靠、有序和无重复的。也就是说，一个从数据报套接口接收信息的进程有可能发现信息重复了，或者和发出时的顺序不同。数据报套接口的一个重要特点是它保留了记录边界。对于这一特点，数据报套接口采用了与现在许多包交换网络非常类似的模型。

在建立分布式应用时最常用的便是客户机/服务器模型。在这种方案中客户应用程序向服务器程序请求服务。这种方式隐含了在建立客户机/服务器间通信时的非对称性。客户机/服务器模型工作时要求有一套为客户机和服务器所共识的惯例来保证服务能够被提供（或被接受）。这一套惯例包含了一套协议。它必须在通信的两头都被实现。

根据不同的实际情况，协议可能是对称的或是非对称的。在对称的协议中，每一方都有可能扮演主从角色；在非对称协议中，一方被不可改变地认为是主机，而另一方则是从机。无论具体的协议是对称的或是非对称的，当服务被提供时必然存在“客户进程”和“服务进程”。

服务程序通常在一个众所周知的地址监听对服务的请求，也就是说，服务进程一直处于休眠状态，直到一个客户对这个服务的地址提出了连接请求。在这个时刻，服务程序被“惊醒”并且为客户提供服务——对客户的请求作出适当的反应。虽然基于连接的服务是设计客户机/服务器应用程序时的标准，但有些服务也是可以通过数据报套接口提供的。

需要全本请访问 www.ertongbook.com

16.2.2 公用套接字结构

在具体使用 WinSock 函数编程时，将要用到一些预定义的套接字结构，即公用套接字结构。这些公用套接字结构包括 sockaddr_in、hostent、netent、protoent 和 servernt。其中 sockaddr 结构是更常用的 Internet 特定格式。

下面将详细说明这些结构。

1．sockaddr_in

sockaddr_in 用于存储一个 IP 端口和作为套接字端点的计算机地址，其原形为：

```
struct sockaddr_in
{
        short      sin_family;
       u_short     sin_port;
       struct    in_addr  sin_addr;
        char       sin_zero[8];
};
```

其各个参数含义说明为：

sin_family　只可取 AF_INET。

sin_port　端口号。

sin_addr　IP 地址。

sin_zero　没有实际意义的参数。

2．hostent

hostent 主要用于指定主机名，其函数原形为：

```
struct   hostent
{
        char       FAR  *  h_name;
       char       FAR  *  FAR  *  h_aliases;
        short      h_addrtype;
        short      h_length;
        char       FAR * FAR * h_addr_list;
}
```

下面将给出参数说明 ：

h_name　主机名。

h_aliases　别名数组。

H_addrtype　主机地址的类型。

H_len　为以字节为单位的地址长度。

H_addrlst　用于主机的地址列表。

3．netent

netent **用于网络信息，其函数原形为：**

```
struct   netent
{
        char      FAR  *  n_name;
        char      FAR  *  FAR  *  n_aliases;
        short     n_addrtype;
        u_long    n_net;
};
```

以下是其参数说明：

n_name　**网络名。**

n_aliases　**别名数组。**

n_addrtype　**网络地址的类型。**

n_net　**网络编号。**

4．protoent

protoent **用于协议信息，其原形为：**

```
struct   protoent
{
        char      FAR  *  p_name;
        char      FAR  *  FAR  *  p_aliases;
        short     p_proto;
};
```

参数说明如下：

p_name　**为协议名。**

p_aliases　**别名数组。**

p_proto　**协议编号或** ID**。**

5．servent

servent **用于服务信息，其原形为：**

```
struct   servent
{
        char      FAR  *  s_name;          /*  official  service  name  */
        char      FAR  *  FAR  *  s_aliases;     /* alias list */
        short     s_port;                    /* port # */
        char      FAR  *  s_proto;           /* protocol to use */
};
```

下面将介绍其参数：

s_name 服务名。

s_aliases 别名数组。

s_port 为与服务相联系的端口号。

S_proto 为服务使用的协议名。

16.2.3 字节顺序转换

在不同的计算机系统中，字节存储于一个字中的方式有两种：

1．大端位法

字节的高位处于字的左端，低位处于字的右端，这种存储方法称为“大端位法”。

2．小端位法

字节的高位处于字的右端，低位处于字的左端，这种存储方法称为“小端位法”。

在不同的计算机系统中，使用不同的字节存储顺序。如 Intel 体系结构(如 x86 系列微机)的计算机系统中使用的是“小端位法”，而 RISC 体系结构（如多数 UNIX 系统）、Motorola 体系结构（如 Macintosh 计算机）和 TCP/IP 网络中公用字节顺序都使用“大端位法”。

这样，当向不同字节存储方式的计算机发送数据时，就会出现问题：发送的数据和接收到的数据的每个字节中的值都是反的！

WinSock 提供了一组函数可以完成“大端位法”与“小端位法”之间的转换，这些函数包括了 32 位的版本和 16 位的版本。下面将列出这些函数：

WSAHtonl

小端位法转换成大端位法（32 位）。

WSAHtons

小端位法转换成大端位法（16 位）。

WSANtohl

大端位法转换成小端位法（32 位）。

WSANtohs

大端位法转换成小端位法（16 位）。

如果使用 WinSock 的 MFC 类编写程序，则这种转换将会自动完成。

16.3 API程序设计

Windows 2000 实际上提供了 60 个套接字 API 函数。其中有 23 个 API 函数是 Berkeley 版的，另外的 37 个 API 函数是 Windows 扩展的套接字 API。

16.3.1 套接字的API函数

虽然 Windows 2000 提供了 60 个套接字 API 函数，但是它被分成了两部分，一部分是 Berkeley 版的套接字 API，另一部分是 Windows 扩展的套接字 API，而实际上对于每一个

Berkeley 版的套接字 API 函数来说，都有一个与之相对的 Windows 扩展的套接字 API 函数，不管是实现方法还是功能都是大体一致的，只是相对的 Windows 扩展的套接字 API 函数进行了一些拓展。事实上，Windows 扩展的套接字 API 与 Berkeley 版的套接字 API 之间的最大差别应该是：Windows 扩展的套接字 API 提供了另外一些函数，以在 Windows 环境下进行异步处理。

基于上面的原因，我们在这一节将重点介绍 Berkeley 版的套接字 API，对于 Windows 扩展的套接字 API 中一些重要的函数也会给予详细的说明。

下面将分别介绍 23 个 Berkeley 版的套接字 API 函数和 37 个 Windows 扩展的套接字 API 函数。

一、Berkeley 版的套接字 API 函数

Berkeley 版的套接字 API 函数分为第一类和第二类。

（一）第一类 Berkeley 版的套接字 API 函数

第一类 Berkeley 版的套接字 API 函数(关于 API 函数的分类方法见本部分的开篇)，一共有以下 10 个：

1．Socket

```
SOCKET socket
(
   int  af,
   int  type,
   int  protocol
);
```

socket()函数可用来创建一个绑定于指定的服务提供者的套接字。

参数说明如下：

af　指定地址系列，通常为 AF_INET。

type　套接字的种类，可以为 SOCK_STREAM 或 SOCK_DGRAM，其中 SOCK_STREAM 将创建一个流式套接字，而 SOCK_DGRAM 将创建一个数据报套接字。

protocol　一个针对于指定地址系列的特定协议。

创建一个套接字，是进行进一步操作的先决条件。不管你最终想要干什么，首先必须创建一个套接字。

下面的代码将创建一个使用 UDP 协议的数据报套接字：

```
SOCKET dgramSock;
dgramSock=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
```

而这里的代码将创建一个使用 TCP 协议的流式套接字：

```
SOCKET streamSock;
streamSock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

若一个 socket()函数调用出错后，使用 WSAGetLastError()可以得到出错信息值。

关于这个函数出错信息值见表 16-1。

表 16-1 socket()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEAFNOSUPPORT	指定的地址系列不被支持
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEMFILE	已经没有有效的 SOCKET 句柄
WSAENOBUFS	没有足够的缓冲区以创建套接字
WSAEPROTONOSUPPORT	指定的协议不被支持
WSAEPROTOTYPE	指定的协议错误
WSAESOCKTNOSUPPORT	指定的套接字类型不被该地址系列所支持

2．Bind

```
int bind
(
   SOCKET  s,
  const  struct  sockaddr  FAR*   name,
   int  namelen
);
```

bind()将把一个指定的套接字绑定于一个 IP 地址。bind 函数使用在一个未绑定的套接字调用 connect（）函数或 listen（）函数之前，它同时适用于面向连接的套接字（即流式套接字）和无连接的套接字（数据报套接字）。

参数说明如下：

s 一个未绑定的套接字句柄；

name 一个指向 sockaddr 结构的指针;

namelen 上一个参数的长度。

下面的代码将绑定一个流式套接字：

```
SOCKET streamSock;
streamSock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
sockaddr_in*    pAddr;
pAddr->sin_family=AF_INET;
pAddr->sin_port=0;
pAddr->sin_addr.s_addr=htonl(INADDR_ANY);
if(SOCKET_ERROR==bind(streamSock,（LPSOCKADDR）pAddr,sizeof(sockaddr))
{
        int  nError=WSAGetLastError();
```

```
        //错误处理
    }
```

在上面的代码中，首先创建了一个流式套接字，刚创建好的套接字在绑定以前是不能进行其他如 connect()或 listen()等操作的，所以接下来的代码使用了 bind 函数来绑定该流式套接字和本地机器的 IP 地址。

Bind 函数要求传进一个指向 sockaddr 结构的指针，上面的代码中对 pAddr 指向的结构进行了填充，对于 IP 地址，sin_family 的值始终是 AF_INET，sin_port 要求被赋予一个端口号，端口号可以使用 1-65535 之间的任何一个值，但是要注意，有很多的端口号是保留的，如 Ftp、Http、Telnet、Finger 等服务都有保留的端口号，所以事实上端口号被分为以下几段：

端口 1~255，保留给特定的服务；

端口 256~1023，保留给别的一般服务；

端口 1024~4999，提供给任意客户机（Client）端口使用;

端口 5000~65535，提供给任意服务器 (Sever) 端口使用；

上面的代码中给 sin_port 赋的值为 0,这样 bind 函数会自动给你的应用程序分配一个非保留的端口号。需要填充的另一个值是 sin_addr 的 s_addr，这里我们给出的是 INADDR_ANY，这样 bind()函数将自动得到本地机器的 IP 地址。

若一个 bind()函数调用出错后，使用 WSAGetLastError()可以得到出错信息值。

关于这个函数出错信息值见表 16-2。

表 16-2　　bind()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEADDRINUSE	重复绑定错误
WSAEADDRNOTAVAIL	指定的地址无效
WSAEFAULT	名字或名字长无效
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEINVAL	指定套接字已被绑定
WSAENOBUFS	连接太多，没有足够的缓冲区
WSAENOTSOCK	给出的句柄非套接字句柄

3．Sendto

```
int sendto
(
   SOCKET  s,
   const  char  FAR  *  buf,
   int  len,
   int  flags,
   const  struct  sockaddr  FAR  *  to,
```


需要全本请访问 www.ertongbook.com

int tolen
);

sendto()函数用于向数据报套接字发送数据。

参数说明如下：

s 一个套接字句柄；

buf 一个指向存储发送数据的缓冲区的指针；

len 缓冲区中的数据的长度；

flags 指定发送方式的标志；

to 一个指向目标套接字的指针；

tolen 上一个参数的长度。

表 16-3 sendto()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEACCES	未设置合适的标志 flag
WSAEINVAL	指定了一个未知的标志 flag
WSAEINTR	一个阻塞套接字被中断
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEFAULT	tolen 参数太小或 to、buf 参数无效
WSAENETRESET	连接被打断
WSAENOBUFS	没有足够的缓冲区
WSAENOTCONN	套接字未连接
WSAENOTSOCK	给出的句柄非套接字句柄
WSAEOPNOTSUPP	带外数据不被该套接字支持
WSAESHUTDOWN	套接字被终止发送功能
WSAEWOULDBLOCK	套接字非阻塞，但执行的操作可能会发生阻塞
WSAEMSGSIZE	发送的消息过大
WSAEHOSTUNREACH	不能连接主机
WSAECONNABORTED	超时中断
WSAECONNRESET	连接复位
WSAEADDRNOTAVAIL	远程地址无效
WSAEAFNOSUPPORT	套接字不支持该地址
WSAEDESTADDRREQ	缺少目标地址
WSAENETUNREACH	网络不可达
WSAETIMEDOUT	因系统或网络的原因，连接中断

如果发送成功，sendto()函数将返回实际发送的字节数，如果它未能发送所有的数据，它的返回值将和传到 len 中的数值不同。如果未能发送，则它将返回 SOCKET_ERROR，这

需要进行相应的出错处理。

若一个 sendto()函数调用出错后，使用 WSAGetLastError()可以得到出错信息值。

关于这个函数出错信息值见表 16-3。

4．Recvfrom

```
int recvfrom
(
   SOCKET  s,
   char  FAR*  buf,
   int  len,
   int  flags,
  struct  sockaddr  FAR*  from,
   int  FAR*  fromlen
    );
```

recvfrom（）函数用于从数据报套接字接收数据。

参数说明如下：

s　一个套接字句柄；

buf　一个指向存储接收数据的缓冲区的指针；

len　缓冲区的长度；

flags　指定接收方式的标志；

from　一个指向发送数据的套接字的指针；

fromlen　上一个参数的长度。

如果数据读入成功的话，revfrom()将返回所接受到的数据的字节数。而如果数据接收错误，函数将返回 SOCKET_ERROR。

若一个 revfrom()函数调用出错后，使用 WSAGetLastError()可以得到出错信息值。

关于这个函数出错信息值见表 16-4。

5．Listen

```
int listen
(
   SOCKET  s,
   int  backlog
);
```

使用 listen()函数之前，你必须已经创建了一个流式套接字（使用 sock()函数），并已将一个地址与之绑定了起来（使用 bind()函数）。一切都准备好了之后，你需要建立起和客户机的连接，这就要使用到 listen()函数。

Listen()函数将把一个套接字置于侦听连接请求的状态中。

表 16-4 revfrom()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEFAULT	buf 或 from 参数无效
WSAEINTR	一个阻塞套接字被中断
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEINVAL	套接字未绑定或指定了一个未知的标志 flag
WSAEISCONN	该函数不支持连接的套接字
WSAENETRESET	连接被打断
WSAENOTSOCK	给出的句柄非套接字句柄
WSAEOPNOTSUPP	带外数据不被该套接字支持
WSAESHUTDOWN	套接字被终止接收功能
WSAEWOULDBLOCK	套接字非阻塞，但执行的操作可能会发生阻塞
WSAEMSGSIZE	发送的消息过大
WSAETIMEDOUT	因系统或网络的原因，连接中断
WSAECONNRESET	连接复位

参数说明如下：

s **一个套接字句柄；**

backlog **指定侦听请求连接的客户机的个数。**

下面的代码将创建并绑定一个流式套接字，并使它开始侦听连接请求。

```
SOCKET streamServerSock;
streamServerSock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
sockaddr_in*    pAddr;
pAddr->sin_family=AF_INET;
pAddr->sin_port=htons(6000);
pAddr->sin_addr.s_addr=htonl(INADDR_ANY);
if(SOCKET_ERROR==bind(streamServerSock,（LPSOCKADDR）pAddr,
  sizeof(sockaddr)))
{
        int  nError=WSAGetLastError();
        //错误处理
}
if(SOCKET_ERROR==listen(streamServerSock,4))
{
        int  nError=WSAGetLastError();
        //错误处理
```

}

若一个 listen()函数调用出错后，使用 WSAGetLastError()可以得到出错信息值。关于这个函数出错信息值见表 16-5。

表 16-5　　listen()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEADDRINUSE	套接字本地地址已被使用
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEINVAL	套接字未绑定
WSAEISCONN	套接字已经连接
WSAEMFILE	已经没有有效的 SOCKET 句柄
WSAENOBUFS	没有足够的缓冲区
WSAENOTSOCK	给出的句柄非套接字句柄
WSAEOPNOTSUPP	给出的套接字不被 listen()函数支持

6．Connect

```
int connect
(
   SOCKET  s,
  const  struct  sockaddr  FAR*    name,
   int  namelen
);
```

当应用程序需要使用流式套接字发送并接收数据时，首先要创建并绑定一个流式套接字，然后使用 listen()函数将之置于侦听连接请求的状态，这时就需要客户机发起连接请求，请求得到确认并且建立起连接插口后，客户机和服务器就可以通信了。

在上述步骤中，客户机发起连接请求，就是由 connect()函数实现的。

参数说明如下：

s　一个套接字句柄；

name　想要连接的套接字的地址和端口；

namelen　上一个参数的长度。

Connect()函数的第二个参数是一个指向 sockaddr 结构的指针，该结构中包含了想要连接的套接字的地址和端口号，大家应该还记得，bind()函数的第二个参数也是这样一个指针，实际上这两者的关系是，绑定服务器的套接字使用 bind()函数时，传给它的第二个参数，应该和传给想要和该服务器建立连接的客户机所使用的 connect()函数的第二个参数是相同的。

下面的代码将给出使用 connect（）的例子。//创建 streamClientSock

SOCKET　streamClientSock;

```
sockaddr_in*    pAddr;
sockaddr_in*    pSAddr;
streamClientSock= socket(AF_INET,SOCK_STREAM,0);
pAddr->sin_family=AF_INET;
pAddr->sin_port=0;
pAddr->sin_addr.s_addr=htonl(INADDR_ANY);
if(SOCKET_ERROR==bind(streamClientSock,（LPSOCKADDR）pAddr,
  sizeof(sockaddr)))
{
      int  nError=WSAGetLastError();
       //错误处理
}
//发出请求
pSAddr->sin_family=AF_INET;
pSAddr->sin_port=htons(6000);
pSAddr->sin_addr.s_addr=htonl(INADDR_ANY);
if(SOCKET_ERROR==connect(streamClientSock,
                                    (LPSOCKADDR)pSAddr,sizeof(sockaddr)))
{
      int  nError=WSAGetLastError();
       //错误处理
}
```

上面的例子中，先创建了一个客户机流式套接字 streamClientSock**，然后向一个服务器发出连接请求，**streamClientSock **想要连接的是上一个** listen()**的例程中创建并处于侦听状态的** streamServerSock**。**

连接请求发出后，就可等待服务器确认并建立连接了。

若一个 connect()**函数调用出错后，使用** WSAGetLastError()**可以得到出错信息值。**

关于这个函数出错信息值见表 16-6**。**

7．Accept

```
SOCKET accept
(
   SOCKET  s,
  struct  sockaddr  FAR*  addr,
   int  FAR*  addrlen
);
```

如客户机使用 connect()**函数向服务器发出了连接的请求，而此时服务器也已经使用** listen()**函数置于了侦听状态，并在侦听的过程中接收到了客户机发出的连接请求，则可以通**

过 accept（）函数来建立起连接。

表 16-6　　connect()出错信息值

标志	说明
WSANOTINITIALISED	未初始化
WSAENETDOWN	与服务提供者联系的网络子系统无效
WSAEADDRINUSE	套接字本地地址已被使用
WSAEINPROGRESS	服务提供者正在处理一个回调函数
WSAEINTR	一个阻塞套接字被中断
WSAEALREADY	一个非阻塞的连接呼叫
WSAEADDRNOTAVAIL	指定的地址无效
WSAEAFNOSUPPORT	指定的地址系列不被支持
WSAECONNREFUSED	放弃连接的尝试
WSAEFAULT	name 或 namelen 参数无效
WSAEINVAL	参数位侦听套接字
WSAEISCONN	套接字已连接
WSAENETUNREACH	网络不可达
WSAENOBUFS	没有足够的缓冲区
WSAENOTSOCK	给出的句柄非套接字句柄
WSAETIMEDOUT	因超时而连接失败
WSAEWOULDBLOCK	套接字非阻塞，但执行的操作可能会发生阻塞
WSAEACCES	广播数据报失败

参数说明如下：

s　一个处于侦听状态的套接字句柄;

addr　客户机套接字的地址；

addrlen　上一个参数的长度。

accept()函数的第二个参数是一个指向 sockaddr 结构的指针，这个结构将用来返回 accept()所接受到的提出连接请求的客户机套接字的地址。

如果连接被成功的接收了，accept()函数将返回一个新的套接字句柄，这个新的套接字句柄将用来处理该连接。这一处理完成了以后，服务器套接字将会继续置于侦听状态，等待处理下一个连接请求。如果又有客户机发起连接请求，accept()函数又将处理它，并返回又一个新的套接字句柄。也就是说，每一个已被接收的连接请求都将创建自己的套接字，这将需要一个套接字的句柄的数组来实现。

下面给出了使用 accept()函数的例子：

```
SOCKET streamServerSock;
SOCKET    connectSock[10]；
sockaddr    pconnectAddr[10];
int      nLen=sizeof(sockaddr);
```


需要全本请访问 www.ertongbook.com