Visual C++
网络通信编程实用案例精选

第二版

四维科技　曹衍龙　刘海英　编著

- Visual C++网络通信编程基础
- 本地计算机网络编程
- 局域网网络通信编程实例
- IE编程实例
- 基本网络编程实例
- 网络通信协议编程实例
- Modem／串口通信编程实例
- 代理服务器编程实例
- 高级实例解析

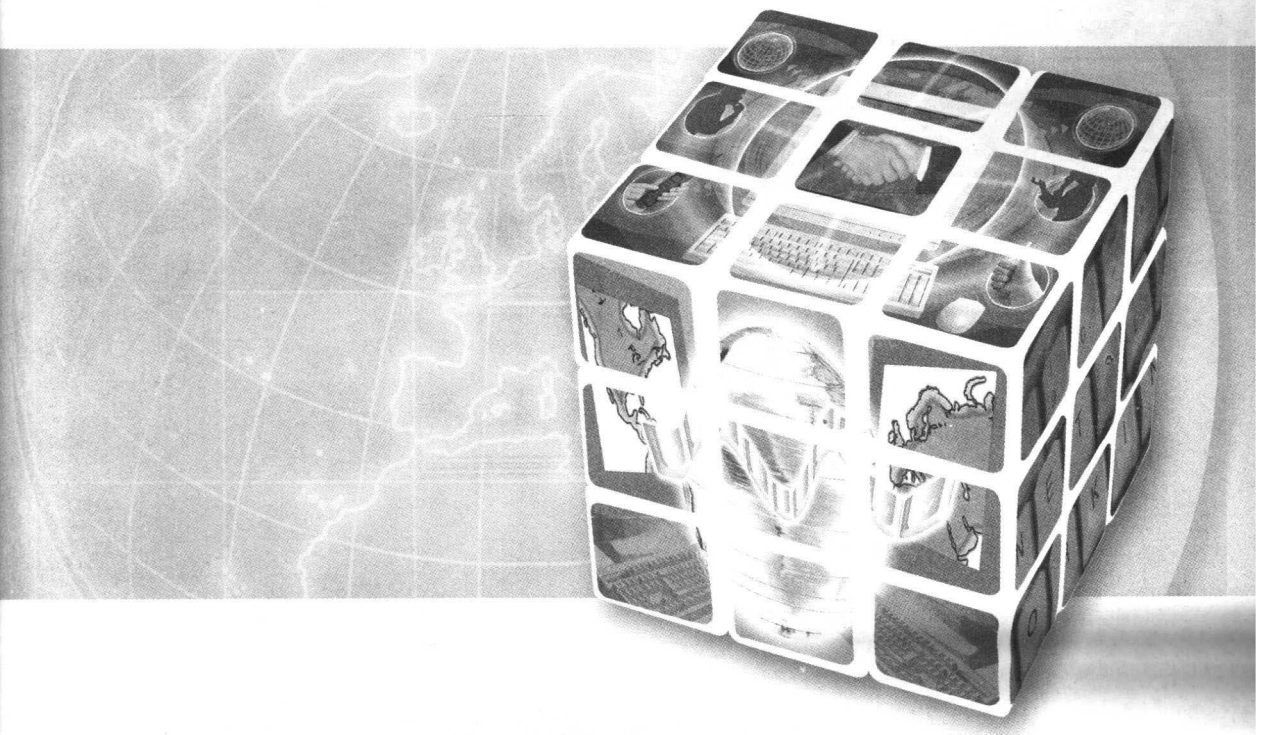
源代码光盘
CD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

Visual C++ 网络通信编程实用案例精选

第二版

四维科技　曹衍龙　刘海英　编著



人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目（CIP）数据

Visual C++网络通信编程实用案例精选：第2版 / 曹衍龙，刘海英编著.
—北京：人民邮电出版社，2006.5
ISBN 7-115-14704-3

Ⅰ.V… Ⅱ.①曹… ②刘… Ⅲ.C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2006）第032706号

内 容 提 要

本书是一本介绍利用Visual C++进行网络通信程序开发的书籍。书中精选了大量网络实例，涵盖了本地计算机网络编程、局域网网络通信编程、IE 编程、网络通信协议编程、串口通信编程、代理服务器编程和高级网络通信编程。每个工程实例都提供了完整的源代码，读者可以很容易地根据需要进行二次开发。

本书适合进行网络通信开发的人员阅读，同时也可以作为科研单位、高校相关专业人员的参考书籍。

Visual C++网络通信编程实用案例精选（第二版）

◆ 编　著　四维科技　曹衍龙　刘海英
责任编辑　汤　倩

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京密云春雷印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：32.25
字数：813千字　2006年5月第2版
印数：11 001 – 16 000册　2006年5月北京第1次印刷

ISBN 7-115-14704-3/TP・5369

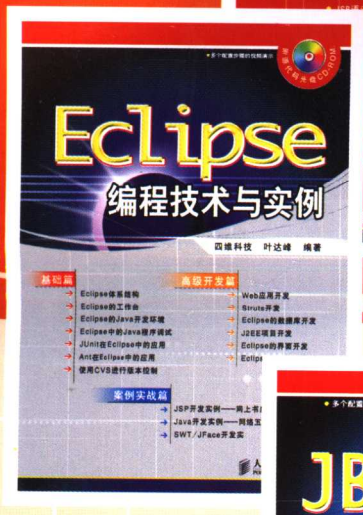
定价：56.00元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223

基础+实例

伴你步入编程高手的行列

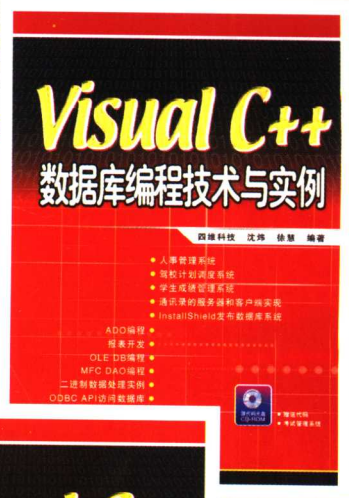
书号：13746
定价：49.00元



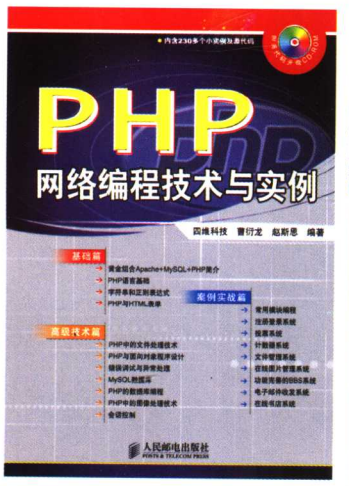
书号：14153
定价：45.00元

书号：14257
定价：39.00元

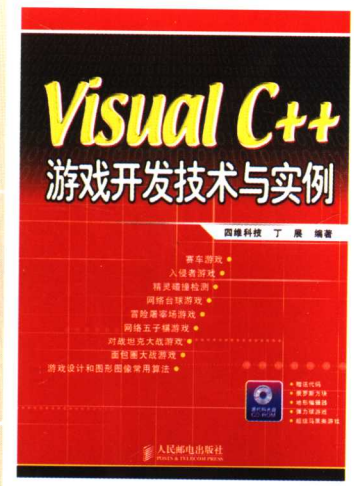
书号：14294
定价：49.00元



书号：13024
定价：49.00元



书号：14529
定价：56.00元



书号：13023
定价：58.00元

系统开发实例精粹丛书

丛书特色

★ **案例精**——书中汇集了作者多年来从事项目开发的经验之精华，很多实例都提炼自作者从事过的实际工程项目。选取的案例具有典型性，非常适合开发中、小型的系统模块。

★ **性价比高**——全书版式紧凑，虽然有些实例的源代码多达上千行，书中只精选了其中的重点和难点代码段进行分析和讲解，其余的代码均收录在随书的光盘中。

★ **涵盖领域广**——每本书都涉及了“系统编程”、“多媒体/图形图像处理”、“网络通信编程”、“数据库/MIS开发”和“游戏编程”等各个开发领域。

★ **光盘内容超值**——光盘中不但涵盖了书中正文实例的全部源代码，而且赠送了大量的典型实例源代码，以帮助读者拓宽视野，更好地完成相关领域的开发工作。

前　言

随着计算机网络的迅猛发展，网络通信编程在程序设计领域变得越来越重要。目前大部分的高等院校计算机以及相关专业都开设了计算机网络通信编程方面的课程。同时很多科研单位、企业也在从事相关网络软件的开发。

计算机网络通信编程的一个主要工具是 Visual C++。虽然其他开发工具如 Delphi、Visual Basic 等也可以进行网络编程，但是在程序运行效率以及灵活性方面，却远远比不上 Visual C++。

为什么写本书

虽然已出版的关于网络通信编程的书籍比较多，但大都介绍的是基本的网络原理和技术以及网络通信编程的部分内容，对于更高级的网络应用，还需要很多复杂的步骤和设计。本书精选了大量网络编程实用案例，基本上涵盖了当前网络通信编程领域的热点问题。

本书特色

✓ 提供了大量的网络通信编程实例，如本地计算机网络编程、局域网网络通信编程等涉及网络编程的各个领域；

✓ 每个实例都提供了完整的源代码和已编译通过的应用程序，读者可以在此基础上很方便地进行二次开发；

✓ 所有的实例都按照设计思路、相关概念、原理，具体的实现方式进行介绍，读者能够很容易地掌握整个应用程序开发的全过程。

✓ 光盘中提供了每个实例的完整源代码，方便读者举一反三，开发出适合自己的工程项目。

新版说明

本书是《Visual C++网络通信编程实用案例精选》的第二版。第一版图书在问世后近两年时间里，受到了广大读者的欢迎，我们感到十分欣慰。许多读者对本书提出了宝贵的意见和建议，与我们探讨了 Visual C++编程方面的经验，在此向他们表示衷心的感谢。

随着网络技术的发展，Visual C++编程技术也在不断发展和完善。我们结合最新的网络应用技术，深入分析了读者的反馈，并进行了归纳和管理，对原书进行了改版。改版主要进行了以下几项工作：

✓ 修订、改正了原书的错漏之处，删除了过时的内容，增加了许多新的、实用的内容；

✓ 在保持原书风格和体例的基础上，更加注重系统性、实用性和全面性。

本书由四维科技曹衍龙主编，参加写作的人员有刘海英、汪杰、续瑞瑞、陆元军、林瑞仲、卜利君、厉蒋等。在编写过程中，我们力求精益求精，但难免存在一些不足之处，如果读者使用本书时遇到问题，可以发 E-mail 到 forway@zj.com 和 tangpian@ptpress.com.cn，我们会及时给您回复。

编　者

目　录

第 1 章　Visual C++网络通信编程基础

网络有自己的“语言”，Internet 上的“语言”就是网络通信协议。Internet 是在 UNIX 系统上发展起来的，在 UNIX 上有许多成熟的编程接口，其中最通用的是一种叫做 Socket 的接口。在 20 世纪 80 年代初，美国政府的高级研究工程机构为加州大学伯克利分校提供资金，委托其在 UNIX 操作系统下实现通信协议 TCP/IP 的开发接口。他们的工作成果就是 Socket，一般将其称为“套接字”。

在 1991 年前后，许多网络软件商都在加紧研制 Windows 下的 TCP/IP 通信组件。为了能使这些组件具有一定的标准，降低开发难度，他们决定为 Windows 系统开发一套标准的、通用的 TCP/IP 编程接口，并使之类似于 UNIX 下的 Socket。这一接口迅速被所有的软件商接受，甚至包括微软和 IBM。到 1994 年，它被正式制定成一项标准，称为 Windows Socket 或称 Winsock，并通过 C 语言的动态链接库方式提供给用户及软件开发者。现在，Windows 下的 Internet 软件都是在 Winsock 的基础上开发的。伴随着 Windows 95 的推出，Winsock 被正式集成到了 Windows 系统中，同时包括了 16 位与 32 位的编程接口。而 Winsock 的开发工具也可以在 Borland C++、Visual C++这些 C 编译器中找到，主要由 Winsock.h 头文件和动态链接库 winsock.dll 组成。

Winsock 主要经历了两个版本：Winsock 1.1 和 Winsock 2.0。Winsock 2.0 实际上是 Winsock 1.1 的扩展，它向下兼容。本章中将会详细介绍这两个版本。在 Visual C++环境中，网络程序经常使用 CAsyncSocket 和 CSocket 类。

CAsyncSocket 和 CSocket 类是 MFC 类库中的成员，CSocket 是 CAsyncSocket 的子类，而 CAsyncSocket 用面向对象的方法封装了 Winsock。本章将会介绍这两个 MFC 类。除此之外，还将介绍 WinInet 编程技术，这种技术主要应用于 Internet 客户端应用程序。

1.1　Winsock1.1 编程技术

先来看一下 Winsock 编程的一些基本内容。

1.1.1　Winsock 的基本概念

Socket 在英文中是“插座”的意思，它的设计者实际上是暗指电话插座。因为在 Socket 环境下编程很像是模拟打电话，Internet 的 IP 就是电话号码，要打电话，需要电话插座，在程序中就是向系统申请一个 Socket，以后两台机器上的程序“交谈”都是通过这个 Socket 来进行的。对程序员来说，也可以把 Socket 看成一个文件指针，只要向指针所指的文件读写数据，就可以实现双向通信。利用 Socket 进行通信，有两种主要的方式。第一种是面向连接的

流方式。顾名思义，在这种方式下，两个通信的应用程序之间先要建立一种连接链路，其过程好像在打电话。一台计算机（电话）要想和另一台计算机（电话）进行数据传输（通话），必须首先获得一条链路，只有确定了这条通路之后，数据（通话）才能被正确接收和发送。这种方式对应的是 TCP（Transport Control Protocol）。第二种叫做无连接的数据报文方式，这时两台计算机像是把数据放在一个信封里，通过网络寄给对方，信在传送的过程中有可能会残缺不全，而且后发出的信也有可能会先收到，它对应的是 UDP（User Datagram Protocol）。流方式的特点是通信可靠，对数据有校验和重发的机制，通常用来做数据文件的传输如 FTP、Telnet 等；数据报文方式由于取消了重发校验机制，能够达到较高的通信速率，可用于对数据可靠性要求不高的通信，如实时的语音、图像转送和广播消息等。

在 ISO 的 OSI 网络七层协议中，Winsock 主要负责控制数据的输入和输出，也就是传输层和网络层。Winsock 屏蔽了数据链路层和物理层，它的出现给 Windows 下的网络编程带来了巨大的变化。

1.1.2　Winsock 的编程特点

在网络通信中，由于网络拥挤或一次发送的数据量过大等原因，经常会发生交换的数据在短时间内不能传送完，收发数据的函数因此不能返回的现象，这种现象叫做阻塞。Winsock 对有可能阻塞的函数提供了两种处理方式，阻塞和非阻塞方式。在阻塞方式下，收发数据的函数在被调用后一直要到传送完毕或者出错才能返回。在阻塞期间，除了等待网络操作的完成不能进行任何操作。对于非阻塞方式，函数被调用后立即返回，当网络操作传送完成后，由 Winsock 给应用程序发送一个消息，通知操作完成，此时可以根据发送的消息传出的参数判断操作是否正常。

在编程时，应尽量使用非阻塞方式，因为在阻塞方式下，用户可能会因为长时间的等待而失去耐心。关闭应用程序的主窗口，这样当网络操作的函数从 Winsock 的动态链接库中返回时，主程序已经从内存中删除，可能会造成内存的异常。虽然现在的操作系统已经在从系统级正确处理这种内存问题，但还是建议读者关注这种情况的发生。

1.1.3　Winsock 基本的 API

本节介绍一些常用的 Winsock API。

（1）WSAStartup

```
int PASCAL FAR WSAStartup ( WORD wVersionRequested, LPWSADATA lpWSAData) ;
```

Windows Socket 的功能由 DLL 形式提供，为了完成一系列初始化操作，每一个使用 Windows Socket 的应用程序都必须进行 WSAStartup()函数调用，并只有在成功地完成调用之后才能使用 Socket。

参数 wVersionRequested 表示欲使用的 Windows Sockets API 版本；这是一个 WORD 类型的整数，它的高位字节定义的是次版本号，它的低位字节定义的是主版本号。程序中可以使用宏 MAKEWORD(X,Y)（其中，X 是高位字节，Y 是低位字节）设置这个参数。目前 Winsock 版本是 2.2。

参数 lpWSAData 是指向 WSADATA 结构的指针，WSAStartup 加载的动态库的有关信息都填充在这个结构中。

WSAStartup 调用成功返回 0。返回失败则有以下可能值。

- WSASYSNOTREADY：网络设备没有准备好。
- WSAVERNOTSUPPORTED：Winsock 的版本信息号不支持。
- WSAEINPROGRESS：一个阻塞式的 Winsock1.1 存在于进程中。
- WSAEPROCLIM：已经达到 Winsock 使用量的上限。
- WSAEFAULT：lpWSAData 不是一个有效的指针。

说明：此函数是应用程序调用 Windows Socket API 中的第一个，也惟有此函数成功后，才可以再调用其他 Windows Socket 的函数。

（2）socket

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

所有的通信在建立之前都要创建一个套接字，该函数的功能与文件操作中 fopen 类似。参数 af 是指协议的地址族（address family）。如果想建立一个 UDP 或 TCP 的套接字，则该值设置成 AF_INET，表示在网络层采用网际协议（IP）。

参数 type 是协议的套接字类型，可以是以下 5 个值：SOCK_STREAM、SOCK_DGRAM、SOCK_SEQPACKET、SOCK_RAW、SOCK_RDM。当采用流连接方式时用 SOCK_STREAM，采用数据报文方式时用 SOCK_DGRAM。

参数 proctocol 是协议字段，当指定了地址家族和套接字类型后，该值的取值范围就被确定了，默认情况下可以将该值直接设为 0。

socket 函数的返回值是由 Winsock 定义的一种数据类型 SOCKET，它实际是一个整型数据。当套接字创建成功时，Winsock 分配给程序的一个套接字编号，后面调用传输函数时，就可以把它像文件指针一样引用。

如果套接字建立失败，函数的返回值为 INVALID_SOCKET。通常，函数失败的原因是程序中忘记调用 WSAStartup，没有启动 Winsock 动态库。

（3）bind

```
int bind(SOCKET s, strut sockaddr_in*  name, int namelen);
```

成功创建套接字后，下面就应该选定本机通信的接口对象。首先，一台计算机上可能装有多张网卡，每个网卡都有一个独立的 IP 地址。其次，不同的通信程序可以使用不同的端口号，这样一台计算机上可以有几个程序同时使用一个 IP 地址通信，而不互相干扰，IP 地址与端口号的关系就好像电话总机号码与分机号码的关系。在 Internet 上，1024 以下的端口号已经被一些常用的网络服务，如 FTP（21）、Telnet（23）等占用。所以，编写自己的通信程序时，应指定大于 1024 的端口。

在 bind 函数中，参数 s 是上一步创建好的套接字。name 是指向描述通信对象地址信息的结构体的指针，namelen 是该结构体的长度。sockaddr_in 结构定义如下：

```
struct sockaddr_in{
    short               sin_family;
    unsigned short      sin_port;
```

```
    struct   in_addr        sin_addr;
    char                    sin_zero[8];
};
```

其中 sin_family 是指地址族，当采用 TCP 和 UDP 类型的套接字时该值为 AF_INET，表示网络层采用 IP；sin_port 是指定的端口号；sin_addr 是指定的 IP 地址。在使用 sin_port 和 sin_addr 的时候，程序中需要一次类型转换操作，这是因为网络和主机采用不同的端方法。例如绑定 IP 地址为 10.214.50.124，端口 80 的地址，方法如下：

```
localAddr.sin_addr.s_addr=inet_addr("10.124.50.124");
localAddr.sin_port=htons(80);
```

其中，inet_addr 函数首先将标准点格式的 IP 地址转换成网络上的 unsigned long 型数据；htons 函数负责将主机 unsigned short 类型的数据转换成网络上的 unsigned short 型数据。

sockaddr_in 的最后一个成员 sin_zero 是一个冗余字段，目的是使 sockaddr_in 结构和 SOCKADDR 结构大小相同（SOCKADDR 结构是由一个无符号 short 型和一个长度为 14 的 char 型数组构成，这个结构占用 16 字节）。在 sockaddr_in 中添加这个长度为 8 的数组，从而使 sockaddr_in 的长度也为 16（即 2+2+4+8）字节。

地址结构 in_addr 定义如下。

```
struct   in_addr {
     union{
          struct{
               unsigned  char   s_b1, s_b2, s_b3, s_b4;
          }S_un_b;
          struct{
               unsigned  short  s_w1, s_w2;
          } S_un_w;
          unsigned long  S_addr;
     } S_un;
};
```

如果用户机器只有一个 IP 地址，则可以使用如下代码绑定套接字到默认的 IP 地址。

```
addr.sin_addr.S_un.S_addr = INADDR_ANY;
```

对于采用 TCP 类型的套接字而言，通常只有服务器程序需要调用 bind 函数绑定具体的端口号。对于客户端程序，通常不需要指定端口号，因为 Winsock 会自动为该套接字指定一个空闲的端口号。如果想知道具体被分配了哪个端口，可以调用 getsockname 函数获得具体套接字的地址信息。另外，服务器端在调用 bind 函数的时候还可以将监听端口设置为 0。例如：

```
localAddr.sin_port=htons(0);
```

此时 Winsock 也将为 Socket 分配一个没有冲突的监听端口号。这种用法在 FTP 协议开发中广泛使用。

提示：TCP 和 UDP 的端口号是相互独立的，可以使用相同的端口号而不会相互干扰。

如果 bind 失败，返回值是 SOCKET_ERROR。最常见的错误是 WSAEADDRINUSE，表示需要绑定的 IP 地址和端口号，或者在早些时候被其他进程中的套接字所占用，或者该地址

还处于 TIME_WAIT 状态。如果绑定的是无效地址，则 GetLastError 返回的错误码是 WSAEFAULT。

（4）listen

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

对于 TCP 类型的服务器套接字，在完成 bind 操作后，下一步就应该等待客户机的连接请求。listen 就是把套接字设置成这种状态。

参数 backlog 是并发连接等待队列的长度。如果当某个客户程序要求连接时，服务器此时正在处理和其他客户程序的连接，则后来的连接请求会被放到等待队列中，等待服务器空闲的时候再与之连接。如果服务器调用 listen 后迟迟不使用 accept 接受连接请求，则当队列达到指定长度时，再来的连接请求将被直接拒绝，错误返回码是 WSAECONNREFUSED。

listen 函数调用失败返回 SOCKET_ERROR。最常见的错误是 WSAEINVAL，它表示程序没有调用 bind 函数。

（5）accept

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

当没有连接请求时，对于阻塞式套接字，如果程序调用了 accept 函数，那么线程就将进入等待状态，直至有一个请求到达为止。accept 在接收到连接请求后，会为这个连接建立一个新的套接字，该套接字将负责和客户端通信。需要指出的是，新套接字与监听套接字的端口号是相同的。当 accept 函数返回时，监听套接字将会继续等待其他连接请求，而新套接字才是与客户端进行通信的实际套接字。所以一般将参数中的 SOCKET 称做监听套接字，它只负责接受连接，而不负责通话；而对于 accept 函数返回的 SOCKET，把它称做会话套接字，它只负责与客户端通话。参数中的指针 addr 和 addrlen 用来返回客户机的地址。如果服务器并不关心对方的地址，则可将 accept 函数中后两个参数都设为空，如下所示：

```
socket= accept(g_ListenSocket,NULL,NULL);
```

如果以后想知道对方的地址，则可以调用 getpeername 函数。

accept 函数调用失败将返回 INVALID_SOCKET。最常见的错误是 WSAEWOULDBLOCK，通常意味着当前套接字处于非阻塞状态，同时没有连接请求到来。

注意：listen()和 accept()函数只用于 TCP 类型的服务程序套接字，属于被动等待的函数。

（6）connect

对于 TCP 类型的客户端套接字，在创建完套接字后，程序就可以直接调用 connect 函数向服务器提出连接请求。

```
int connect (SOCKET s, struct sockaddr_in * name, int namlen);
```

其中，参数 s 是上面建立的套接字，name 与 namelen 的含义、使用方法与 bind 相同，用来指定通信对象。

connect 函数失败返回 SOCKET_ERROR。最常见的错误是 WSAECONNREFUSED，表示所连接的指定服务器端口上并不存在监听套接字。

（7）send/recv

```
int send (SOCKET s, char* buf, int len, int flags);
```

```
int recv (SOCKET s, char* buf, int len, int flags);
```

参数 s 是建立连接的套接字。buf 和 len 是发送或接收的数据包及其长度。

send 函数的最后一个参数 flags 一般取 0，还可以取值为 MSG_DONTROUTE 或 MSG_OOB。MSG_DONTROUTE 要求套接字传输的数据不要路由，如果传输协议不支持该选项，则这个请求会被忽略。MSG_OOB 表示套接字此时传输的数据是带外数据，需要紧急处理。

recv 函数的最后一个参数 flags 一般取 0，还可以取值为 MSG_PEEK 或 MSG_OOB。MSG_PEEK 要求套接字从 Winsock 缓冲区中接收数据，但不从缓冲区中将数据删除。MSG_OOB 表示套接字此时传输的数据是带外数据。

recv 函数实际上是读取 send 函数发过来的一个数据包。当读到的数据字节少于规定接收的数目时，就把数据全部接收，并返回实际接收到的字节数；当读到的数据多于规定值时，在流方式下，剩余的数据由下一个 recv 读出，在数据报文方式下，多余的数据将被丢弃。

这两个函数出错时都返回 SOCKET_ERROR。最常见的错误是 WSAECONNABORTED，表示对方关闭了连接或操作超时。

如果函数调用成功，则返回的是实际发送或接收的字节数。对于 send 函数而言，如果想成功发送指定数量的数据，调用方法如下：

```
i=0;
while(length>0)
{
      ret = send( sock, &(buffer[i]), length, 0);
      if(ret==0)
               break;
      else if(ret==SOCKET_ERROR)
              break;
      i+=ret;
      length-=ret;
}
```

每次调用 send 的时候都需要检测实际发送的数据量，直到剩余数据量为 0 时才停止发送。

（8）sendto/recvfrom

```
int recvfrom (SOCKET s, char* buf, int len, int flags,struct sockaddr_infrom ,
int*fromlen);
int sendto (SOCKET s, char* buf, int len, int flags,struct sockaddr_into,int*
tolen);
```

UDP 类型的套接字，由于事先不需要建立连接，所以在 bind 函数完成后就可以直接调用函数 recvfrom 和 sendto 收发数据。

参数 from、fromlen、to、tolen 的含义和用法与 connect 中的相同，分别表示接收和发送数据的对方套接字地址。

（9）closesocket

```
closesocket (SOCKET s);
```

通信结束，关闭指定套接字。