第3章 WinSock编程初步

本章主要介绍使用 WinSock 编写网络通信程序应掌握的一些预备知识和方法。这些知识和方法包括 Windows API 函数的概念、如何将 WinSock 的开发组件和运行组件加载到程序中、WinSock 中的网络地址如何表示、如何查询计算机和网络的配置信息等。

3.1 WinSock API 函数

WinSock 是 Windows 操作系统的网络编程接口规范，其全称为 Windows Sockets，它是微软公司以 BSD UNIX 的 Berkeley Sockets 规范为基础开发定义的，从 1991 年问世到现在出现过两个主版本——WinSock1 和 WinSock2。WinSock1 只支持 TCP/IP 协议栈，最流行的版本是 WinSock1.1；WinSock2 完全兼容 WinSock1，但支持多种协议，如 NETBIOS、IPX 等，最流行的版本是 WinSock2.2。

WinSock 规范给出了一套库函数的函数原型和函数功能说明，这套库函数就是所谓的 Windows 套接字应用程序编程接口，常被简称为 WinSock API 函数或 WinSock 函数；这套函数由底层网络软件供应商实现，并提供给高层网络应用程序开发者使用。

WinSock API 函数可分为两大类：一类是与 Berkeley Sockets 相兼容的基本函数，例如，用于创建套接字的 socket()函数，用于发送数据的 send()函数等，这一类函数不仅提供了与 BSD Socket 完全相同的功能，而且函数格式也兼容，因此使用这类函数编写的网络应用程序可以很容易地移植到 Linux、UNIX 等采用 BSD Socket 的系统中，保证了程序的可移植性；另一类是 Windows 扩展函数，这类函数都以 WSA 为前缀，主要是为便于程序员充分利用 Windows 的消息驱动机制编程而提供的。

应用程序是调用 WinSock API 函数实现相互之间通信的，而 WinSock API 函数又利用了下层 Windows 操作系统中的网络通信协议和相关的系统调用来实现自身的通信功能。WinSock 与网络应用程序和操作系统中的网络通信协议软件之间的关系如图 3.1 所示。

熟练掌握常用 WinSock API 函数的功能和使用方法是使用 WinSock 进行网络编程的基础。对一个具体的 WinSock API 函数来说，学习掌握其使用方法的步骤与学习一般 C/C++库函数的方法是类似的，首先是要记住该函数的名称与函数的主要功能，其次就是要清楚各参数的类型及作用以及函数返回值的类型及意义，最后还应掌握函数成功执行的必要条件并了解造成函数不能成功执行的常见原因。

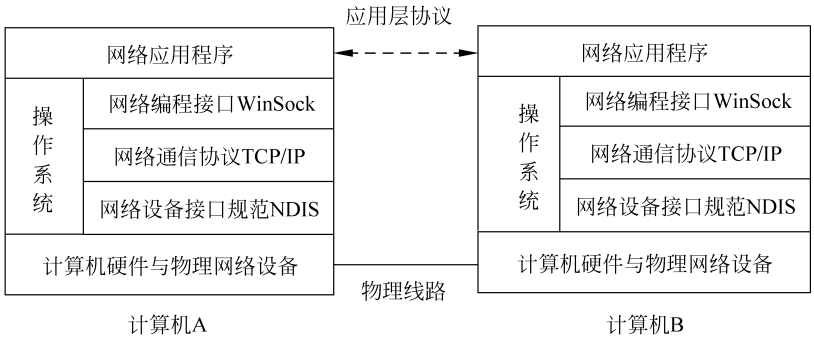

图 3.1 应用程序与 WinSock 以及网络协议软件的关系

3.2 WinSock 开发组件和运行组件

WinSock 是以 Windows 动态链接库(Dynamic Link Library,DLL)的形式实现的。主要由两部分组成——开发组件和运行组件。

动态链接库是一个包含可供多个程序同时使用的函数和全局变量的库,DLL 本身是不可执行的,应用程序在运行过程中可动态装入和连接 DLL 并直接使用其中的函数和全局变量。多个应用程序可同时使用内存中的单个 DLL 副本,DLL 是一种代码共享的方式。

从编程角度,动态链接库可以提高程序模块的灵活性,它本身是可以单独设计、编译和调试的。DLL 的编制与具体的编程语言及编译器无关,只要遵循约定的 DLL 接口规范和调用方式,不管是用何种语言编写的 DLL,各种语言的程序都可使用。

Windows 系统提供了大量的 DLL,这些 DLL 包含允许 Windows 系统本身和应用程序使用的许多函数和资源,这些 DLL 一般被存放在 C:\Windows\System32 目录下。Windows 的 DLL 多数情况下是带有 DLL 扩展名的文件,但也可能是 exe 或其他扩展名。使用 Visual Studio 的程序员可以创建自己的 DLL。

要使用 DLL 中的函数必须要将 DLL 链接到应用程序。链接方式有两种:一种称为显式方式,在这种方式中,首先要在程序中调用 LoadLibrary 调入 DLL 文件,再调用 GetProcAddress 获得 DLL 库中的函数指针,最后再通过指针调用 DLL 中的函数;另一种称为隐式方式,这种方式要用到一种称为 DLL 导入库的文件(.lib 文件),DLL 导入库文件中包含 DLL 中定义的变量和函数等的地址符号表,使用这种方式,程序中只需要按照头文件中的函数声明来调用函数,在建立可执行程序时与 DLL 导入库文件链接就可以了。显然,使用隐式链接方式要方便得多。

WinSock 开发组件是提供给程序员开发程序使用的,最主要的部分是包含程序开发所需的常量定义、宏定义、有关的数据结构定义以及函数调用接口的原型声明等的 C/C++头文件,除此之外,还包括 WinSock 的实现文档以及 WinSock 动态链接库的导入库。运行组件则是指包含 WinSock API 函数的 WinSock 动态链接库。不同版本的 WinSock 所对应的开发组件的相关的文件名字是不同的。WinSock1 对应的头文件是 WinSock.h,导入库文件是 Wsock32.lib,而 WinSock2 的头文件是 WinSock2.h,导入库文件则是 Ws2_32.lib。

由于 WinSock2 完全兼容 WinSock1,因此,当系统中安装的是 WinSock2 时,程序中既可以使用 WinSock1 的头文件和导入库,也可以使用 WinSock2 的头文件和导入库,但如果要使用只有 WinSock2 才有的功能,就只能使用 WinSock2 的头文件和导入库了。

在 Visual C++中使用 WinSock,下面的步骤通常是必不可少的。

1. 包含 WinSock 头文件

包含 WinSock 头文件需要在程序文件首部使用编译预处理命令"#include",比如程序要使用 WinSock2,则程序前面需要使用如下预处理命令将 WinSock2.h 包含进来。

```
#include <WinSock2.h>
```

需要说明一下,如果程序需要包含头文件 Windows.h,但是程序已包含 WinSock 头文件,则程序就不用再包含 Windows.h 了。原因是 WinSock 的实现使用了 Windows.h 中的函数声明以及常量、变量等的定义,在 WinSock 的头文件中已将它包含进去了。

2. 链接 WinSock 导入库

链接 WinSock 导入库的方式有两种:一种是通过在项目属性页中的"配置属性"→"链接器"→"输入"的"附加依赖项"中直接添加导入库名字;另一种则是在程序中使用预处理命令"#pragma comment"。例如,程序要使用 WinSock2 时,可使用如下预处理命令。

```
#pragma comment (lib, "Ws2_32.lib")
```

该命令会添加一个注释到编译生成的 OBJ 文件,告诉链接器链接时要链接库文件 Ws2_32.lib。

3. 加载 WinSock 动态链接库

应用程序运行时必须装入 WinSock 动态链接库才能调用 WinSock 函数实现网络通信功能。加载 WinSock 动态链接库要使用 WSAStartup()函数,该函数原型如下。

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

函数参数

- wVersionRequested:该参数是一个双字节类型数值,用于指明程序中要使用的 WinSock 库的版本号,其中,高位字节指定副版本,低位字节指定主版本。早期 Windows 平台一般使用版本号是 1.1 的 WinSock1,现在常用的则是 WinSock2 的 2.2 版。指定该参数值时可使用十六进制方式给出,比如要加载 WinSock2.2 版,该值可设定为 0x0202,但更常用的方法则是使用宏 MAKEWORD(X,Y),参数 X 为副版本号,Y 为主版本号。
- lpWSAData:该参数用于返回关于使用的 WinSock 版本的详细信息,它是一个指向 WSADATA 结构体变量的指针。WSADATA 结构体的定义如下。

```
#define WSADESCRIPTION_LEN 256
#define WSASYSSTATUS_LEN 128
Typedef struct WSAData {
```

```
    WORD wVersion; //期望程序使用的 WinSock 版本号
    WORD wHighVersion; //加载的 WinSock 库所支持的最高 WinSock 版本号
    char szDescription[WSADESCRIPTION_LEN + 1]; //加载的 WinSock 库说明
    char szSystemStatus[WSASYSSTATUS_LEN + 1]; //状态和配置信息
    unsigned short iMaxSockets;
        //能同时打开的 socket 的最大数(该字段被 WinSock2 及其后版本忽略)
    unsigned short iMaxUdpDg;
      //可发送 UDP 数据报的最大字节数(该字段被 WinSock2 及其后版本忽略)
    char FAR * lpVendorInfo;    //厂商信息(该字段被 WinSock2 及其后版本忽略)
} WSADATA;
```

函数返回值

WSAStartup()函数的返回值是一个整数,函数调用成功返回 0,如果不成功则返回如下所列的值之一,这些常量在 WinSock2.h 中定义。

- **WSASYSNOTREADY**:对应数值为 10091,表示此时 WinSock 不可用,用户应该检查是否存在适合的 Windows Sockets DLL 文件或底层网络子系统能否使用,另外,如果有多于一个的 WinSock DLL 在系统中,必须确保搜索路径中第一个 WinSock DLL 文件是当前加载的网络子系统所需要的,否则也会引发该错误。
- **WSAVERNOTSUPPORTED**:对应数值为 10092,系统当前安装的 WinSock 实现不支持应用程序指定的 WinSock 版本,用户应检查是否有旧的 Windows Socket DLL 文件正在被访问。
- **WSAEINVAL**:对应数值为 10022,表示提供了非法函数参数。
- **WSAEINPROGRESS**:对应数值为 10036,一个阻塞操作正在执行。Windows Sockets 只允许一个任务(或线程)在同一时间可以有一个未完成的阻塞操作,如果此时调用了任何函数(不管此函数是否引用了该套接字或任何其他套接字),此函数将以错误码 WSAEINPROGRESS 返回。
- **WSAEPROCLIM**:对应数值为 10067,Windows Sockets 的实现可能会限制同时使用它的应用程序的数量,如果系统中使用 WinSock 的应用程序数量已达到此限制,再调用 WSAStartup()函数加载 WinSock 库则会失败并返回该错误码。
- **WSAEFAULT**:对应数值为 10014,表示 lpWSAData 指向的地址非法。

假如一个程序要使用 2.2 版本的 WinSock,那么程序中可采用如下代码加载 WinSock 动态链接库。

```
WSADATA wsaData;
WORD wVersionRequested = MAKEWORD( 2, 2 );
if(WSAStartup( wVersionRequested, &wsaData )!= 0)
{
    //WinSock 初始化错误处理代码
    …
}
```

如果程序在没有成功加载 WinSock 动态链接库的情况下调用 WinSock API 函数,则被调函数不能成功执行,并返回错误代码 **WSANOTINITIALISED**,其对应值为 10093。

4. 注销 WinSock 动态链接库

应用程序在完成对 WinSock 动态链接库的使用后,需要解除与 WinSock 库的绑定(注销),并且释放 WinSock 库所占用的系统资源。注销 WinSock 动态链接库需要使用 WSACleanup()函数,该函数原型如下。

```
int WSACleanup (void);
```

该函数无参数,执行后将返回一个整数值,如果操作成功返回 0,否则返回 SOCKET_ERROR。常量 SOCKET_ERROR 在 Winsok2.h 中定义,其对应值为-1。

一个程序中的每一次 WSAStartup()调用,都应该有一个 WSACleanup()调用与之对应。

3.3 网络字节顺序

前面已介绍过,计算机在存储多字节数据时存在大端字节顺序(Big Endian)和小端字节顺序(Little Endian)两种方式,对于字符编码,人们给出的编码标准中明确规定了采用的字节顺序,但对于整型数据则并不存在类似的规定,这是什么原因呢? 整型数据是最基本的数据类型,也是计算机 CPU 指令能直接处理的数据类型,之所以存在大端和小端顺序两种字节顺序,就源于 CPU 内部表示整型数据的字节顺序不同,例如,常见的 PC 上基于 X86 架构的 CPU 是小端字节顺序的,而 PowerPC 系列的 CPU 大多采用的是大端字节顺序。为了提高处理速度,整数各字节无论是在外部存储器还是在内存中,其存放顺序都必须与 CPU 一致。

无论采用的是大端顺序还是小端顺序,在网络通信中,对一台计算机所采用的字节顺序都统称为主机字节顺序。当不同字节顺序的计算机在通过网络交换数据时,如果不做任何处理,将会出现严重问题。例如,一台使用 PowerPC 系列的 CPU、运行 UNIX 的服务器发送一个 16 位数据 0x1234 到一台采用 Intel 酷睿 i5 系列 CPU 运行 Windows 7 的 PC 时,这个 16 位数据将被 Intel 的 CPU 解释为 0x3412,也就是将整数 4660 当成了 13 330。

为了解决这一问题,在编写网络程序时,规定发送端要发送的多字节数据必须先转换成与具体 CPU 无关的网络字节顺序再发送,接收端接收到数据后再将数据转换为主机字节数据。网络字节顺序采用的是大端存储方式。

在指定套接字的网络地址以及端口号时必须使用网络字节顺序,而由套接字函数返回的网络字节顺序的 IP 地址和端口号在本机处理时,则需要转换为主机字节顺序。在套接字编程接口中有专门的函数来完成网络字节顺序和主机字节顺序的转换。

1. htons()函数

该函数将一个 16 位的无符号短整型数据由主机字节顺序转换为网络字节顺序。

函数原型

```
u_short  htons (u_short  hostshort);
```

函数参数

hostshort:一个待转换的主机字节顺序的无符号短整型数据。

返回值

函数调用成功返回一个网络字节顺序的无符号短整型数。如果函数调用失败,则返回SOCKET_ERROR,进一步的出错信息可调用 WSAGetLastError()获取。

2. ntohs()函数

该函数将一个 16 位的无符号短整型数据由网络字节顺序转换为主机字节数顺序返回。

函数原型

```
u_short  ntohs (u_short  netshort);
```

函数参数

netshort:一个待转换的网络字节数顺序的无符号短整型数据。

返回值

函数调用成功返回一个主机字节顺序的无符号短整型数。如果函数调用失败,则返回SOCKET_ERROR,进一步的出错信息可调用 WSAGetLastError()获取。

3. htonl()函数

该函数将一个 32 位的无符号长整型数据由主机字节顺序转换为网络字节顺序返回。

函数原型

```
u_long htonl (u_long hostlong);
```

函数参数

hostlong:一个待转换的主机字节顺序的无符号长整型数据。

返回值

函数调用成功返回一个网络字节顺序的无符号长整型数。如果函数调用失败,则返回SOCKET_ERROR,进一步的出错信息可调用 WSAGetLastError()获取。

4. ntohl()函数

该函数将一个 32 位的无符号长整型数据由网络字节顺序转换为主机字节数顺序返回。

函数原型

```
u_long ntohl (u_long netlong);
```

函数参数

netlong:一个待转换的网络字节数顺序的无符号长整型数据。

返回值

函数调用成功返回一个主机字节顺序的无符号长整型数。如果函数调用失败,则返回SOCKET_ERROR,进一步的出错信息可调用 WSAGetLastError()获取。

3.4 WinSock 的网络地址表示

在 IP 网络环境下,对于一个通信进程而言,必须明确三方面信息:一是进程所在的主机 IP 地址,二是通信所采用的协议,三是所使用的协议端口号。IP 地址可以区分网络中的

不同主机,协议则指明通信所使用的传输层协议是 TCP 还是 UDP,而端口号则可以用于区分同一主机中正在运行的采用同一传输层协议进行通信的不同进程。通过这三方面的信息可以唯一确定在网络中参与通信的一个进程,因此进程的网络地址可以使用三元组(协议,IP 地址,端口号)来标识。

3.4.1 地址结构

程序中可以通过套接字的不同类型来指明通信所使用的传输协议:流式套接字采用 TCP,数据报套接字采用 UDP。IP 地址和协议端口号又是如何表示的呢?针对不同的应用环境,WinSock 定义了三种专门的地址结构来存储 IP 地址和端口号,这三种结构均继承于 BSD Socket 规范。

1. in_addr 结构

用于存储一个 IPv4 地址,结构定义如下。

```
struct in_addr
{
    union
        {
            struct{u_char s_b1,s_b2,s_b3,s_b4}S_un_b;
            struct {u_short s_w1,s_w2}S_un_w;
            U_long S_addr;
        } S_un ;
}
```

该结构只有一个共用体(union)类型的字段 S_un,专门用来存储 32 位的 IP 地址。仔细观察共用体的定义可以发现,共用体的三个成员分别如下。

(1) 由 4 个单字节字段组成的结构体变量 S_un_b,使用该成员便于分别处理 IP 地址的 4 个字节;

(2) 由 2 个双字字段组成的结构体变量 S_un_w,使用该成员可将 IP 地址的高 16 位和低 16 位分别作为无符号短整数处理;

(3) 无符号长整数变量 S_addr,使用该成员可将 IP 地址作为一个无符号长整数处理。

由于人们习惯使用点分十进制表示的 IP 地址,因此当直接为该结构类型的变量赋值时通常使用 S_un_b 成员,具体方法可参见如下代码,这段代码将 IP 地址“192.168.1.1”赋值到 in_addr 型变量 add 中。

```
struct in_addr add;
add.S_un.S_un_b.s_b1 = 192;
add.S_un.S_un_b.s_b2 = 168;
add.S_un.S_un_b.s_b3 = 1;
add.S_un.S_un_b.s_b4 = 1;
```

2. sockaddr_in 结构

用于存储 IP 地址、传输协议端口号等信息。该结构对 IP 地址和协议端口号进行了封

装。结构定义如下。

```
struct sockaddr_in
{
    short sin_family;           //地址族,IP 协议地址对应的值为 AF_INET
    u_short sin_port;           //16 位端口号,需使用网络字节顺序
    struct in_addr sin_addr;    //32 位 IP 地址,需使用网络字节顺序
    char sin_zero[8];           //保留不用
}
```

该结构是专门针对 TCP/IP 协议地址结构的,字段 sin_family 用于存放代表不同协议族地址的代码,TCP/IP 协议族的代码为 AF_INET,该常量在 Winsok2.h 中定义;字段 sin_port 用于存放程序通信使用的 TCP 或 UDP 端口号;sin_addr 用于存放 IP 地址,它是一个 in_addr 结构的变量;最后一个字段 sin_zero 是一个 8 字节大小的字符数组,它在 TCP/IP 协议族地址结构中没有意义,仅仅是为了与通用地址结构兼容而保留的。

3. sockaddr 结构

sockaddr 结构为通用套接字地址结构。当使用 TCP/IP 时,该结构内容与 sockaddr_in 完全相同。结构定义如下。

```
struct sockaddr
{
u_short sa_family;              //协议地址族
char sa_data[14];               //协议地址
}
```

sockaddr 结构是为了兼容多个不同协议族的地址而设计的。事实上,不管是 BSD Sockets 规范还是 WinSock 规范,并不仅仅针对 TCP/IP 协议族,它们在设计时就考虑了要兼容现存的多个网络通信协议族,例如 IPX、NETBIOS 等,不同协议族的地址格式是完全不同的。当程序使用 TCP/IP 时,sockaddr 结构的第一个域 sa_family 应填写 AF_INET,表示 IP 协议族地址;第二个域 sa_data[14]的前两个字节是端口号,随后 4 个字节是 IP 地址,余下的 8 个字节不用。

不难看出,直接填写第二个字节还是比较复杂的,能不能找到一种简单方法为该地址结构类型的变量赋值?答案是肯定的。再回头观察一下 sockaddr_in 结构,可以发现这两个结构存储的内容完全一致,因此,在为 sockaddr 结构的变量赋值时,可先利用 C/C++语言的强制类型转换将该结构变量的地址赋给一个 sockaddr_in 类型的指针,然后通过该指针将各项内容填入结构体变量中。具体方法参见如下代码。首先定义一个 sockaddr_in 结构类型的指针变量 p,通过强制类型转换让 p 指向 sockaddr 结构类型的变量,然后通过指针 p 可按 sockaddr_in 类型的各字段完成赋值。

```
struct sockaddr a;
struct sockaddr_in *p;
p=(sockaddr_in *) &a;
p->sin_family=AF_INET;
p->sin_port=54321;
```

```
p->sin_addr = inet_addr("192.168.1.1");
```

其中,inet_addr()函数的功能是将参数指定的点分十进制表示的IP地址转换为无符号长整型(usigned long)。

当sockaddr结构指针作为函数形参时,可以直接将sockaddr_in结构变量的地址强制转换为struct sockaddr*类型作为实参。由于很多WinSock函数都是以sockaddr结构类型的指针作为参数的,这种方法在以后的例题中会经常使用。

3.4.2 地址转换函数

点分十进制表示的IP地址是一种便于人们书写和记忆的格式,但它并不是计算机内部的IP地址表示方式,计算机内部的IP地址是以无符号长整数方式存储的。因此,在程序中,输入输出IP地址时通常是使用点分十进制表示的IP地址,而程序内部则使用无符号长整型的IP地址,这就导致程序中经常要进行IP地址不同表示形式间的转换。为了编程方便,WinSock提供了一组地址转换函数。

1. inet_addr()函数

函数原型

```
unsigned long inet_addr (const char * cp);
```

函数功能

将参数cp指向的点分十进制字符串表示的IP地址转换为32位的无符号长整型数。

函数参数

cp:指向存放有一个点分十进制表示的IP地址的字符串,当字符串的形式为"a.b.c.d"时,a、b、c、d分别代表IP地址的4个字节;当字符串形式为"a.b.c"时,a、b分别对应IP地址的高位两个字节,c对应于后两个字节;当字符串形式为"a.b"时,a被解释成IP地址的最高一个字节,而b则解释为后面的三个字节24位;当形式为"a"时,a将直接被作为网络字节顺序的二进制表示的IP地址返回。

返回值

函数调用成功后将返回一个无符号长整型数(**unsigned long**),它是以网络字节顺序表示的32位二进制IP地址,如果传入的字符串是一个非法的IP地址,返回值将是INADDR_NONE。

2. inet_aton()函数

函数原型

```
int inet_aton(const char *cp, struct in_addr *inp);
```

函数功能

将参数cp指向的点分十进制字符串表示的IP地址转换为32位的无符号长整型数,并存放于参数inp指向的in_addr结构变量中。

函数参数

- cp:与函数inet_addr的参数cp相同。

• inp：指向 in_addr 结构变量的指针，该结构变量用来保存转换后的 IP 地址。

返回值

如果这个函数成功，函数的返回值非零。如果输入地址不正确则会返回零。

3. inet_ntoa()函数

函数原型

```
char * inet_ntoa (struct in_addr in);
```

函数功能

将一个包含在 in_addr 结构变量中的长整型 IP 地址转换为点分十进制形式。

函数参数

in：是一个保存有 32 位二进制 IP 地址的 in_addr 结构变量。

返回值

函数调用成功返回一个字符指针，该指针指向一个 char 型缓冲区，该缓冲区保存有由参数 in 的值转换而来的点分十进制表示的 IP 地址字符串。如果函数调用失败，则返回一个空指针 NULL。

这三个函数来自于早期的 BSD 套接字规范，并不支持目前已逐渐普及的 IPv6 的地址转换，为了兼容 IPv6 的地址转换，新规范引入了 inet_pton()与 inet_ntop()两个函数来取代上面这三个函数。需要说明的是，这三个函数目前仍被广泛使用，因此读者也应掌握这三个函数的用法。

在 Visual C++ 2017 中使用这三个地址转换函数时，编译器将发出错误警告并停止编译，如果要关闭错误警告继续编译，需要在程序首部添加如下宏定义。

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
```

或者使用以下预处理命令：

```
#pragma warning(disable : 4996)
```

或者在项目属性中将“配置属性/C/C++/常规”中的“SDL 检查”设置为“否(sdl-)”。

下面介绍已逐渐流行的新的地址转换函数 inet_pton()和 inet_ntop()，需要注意，使用这两个函数时需要包含头文件 WS2tcpip.h。

4. inet_pton()函数

函数原型

```
int inet_pton(int af, const char * src, void * dst);
```

函数功能

参数 af 取值 AF_INET 时，该函数将参数 src 指向的点分十进制表示的 IPv4 地址转换为 32 位的网络字节顺序的无符号长整型数，并存放于参数 dst 指向的 in_addr 结构变量中；参数 af 取值 AF_INET6 时，该函数将参数 src 指向的冒号十六进制表示的 IPv6 地址转换为二进制 IPv6 地址，并存放于参数 dst 指向的 in6_addr 结构变量中。由于篇幅所限，本书暂不考虑 IPv6 网络编程。

函数参数

- af：使用的地址族，目前有两个取值：AF_INET 表示 IPv4，AF_INET6 表示 IPv6。
- src：指向要转换的字符串形式表示的地址。
- dst：指向用于保存转换结果的地址结构变量，IPv4 时该结构变量的类型为 in_addr，IPv6 则是 in6_addr。

返回值

如果函数执行成功将返回 1，出错将返回－1 并将错误码设置为 EAFNOSUPPORT，如果参数 af 指定的地址族或 src 格式不对，函数将返回 0。

下面的代码是该函数的使用示例，功能是将 IP 地址“192.168.1.1”转换为无符号长整型数存放在 in_addr 型变量 a 中，然后按十六进制输出该地址。

```
in_addr a ;
if (inet_pton(AF_INET, "192.168.1.1", &a) == 1)
{
    cout << hex << ntohl(a.S_un.S_addr) << endl;    //a 中的值是按网络字节顺序存储的
}
```

5. inet_ntop()函数

函数原型

```
const char * inet_ntop(int af, const char * src, char * dst, socklen_t cnt);
```

函数功能

参数 af 取值 AF_INET 时，将参数 src 指向的 32 位的无符号长整型数表示的 IPv4 地址转换为点分十进制，并复制到参数 dst 指向的存储区中；参数 af 取值 AF_INET6 时，该函数将参数 src 指向的 128 位二进制表示的 IPv6 地址转换为冒号十六进制表示的字符串形式的 IPv6 地址，并复制到参数 dst 指向的存储区中。

函数参数

- af：使用的地址族，目前有两个取值：AF_INET 表示 IPv4，AF_INET6 表示 IPv6。
- src：指向要转换的整型数表示的地址。
- dst：指向用于保存转换结果的存储区。
- cnt：dst 所指向的缓存区的大小。

返回值

如果函数执行成功将返回存储转换结果的缓冲区的首地址，否则返回 NULL。

下面的代码演示了 inet_ntop()函数的使用方法。

```
in_addr a ;
char b[20];
a.S_un.S_addr = htonl(0xc0a80101);
 if (inet_ntop(AF_INET, &a, b, sizeof(b))!= NULL)
 {
     cout << b << endl;
 }
```

3.5 WinSock 的错误处理

WinSock 函数在执行结束时都会返回一个值，如果函数执行成功，对那些需要返回函数执行结果的函数通常返回值就是执行结果（该结果一般是用户程序所需要的），而对无执行结果的函数，例如 WSACleanup ()函数，则返回 0；如果函数执行不成功，则绝大多数函数都会返回 SOCKET_ERROR，虽然通过该返回值可以知道函数调用不成功，但无法判断函数不能成功执行的原因，此时调用 WinSock 提供的 WSAGetLastError()函数可解决该问题。

使用 WSAGetLastError()函数可获取上一次 WinSock 函数调用时的错误代码。该函数的函数原型如下。

```
int WSAGetLastError(void);
```

函数的返回值是一个整数，它是上一次调用 WinSock 函数出错时所出错误对应的错误代码。由于引起 WinSock 函数调用出错的原因很多，为了便于编写出界面友好的应用程序和方便编程者调试程序，WinSock 规范列举出了所有的出错原因，并给每一种原因定义了一个整数类型(int)的编号，该编号被称作**错误码**。在头文件 Winsok2.h 和 WinSock.h 中对每一个错误码都定义了一个对应的符号常量。例如，当客户程序使用流式套接字试图与远程服务器建立连接，而远程服务器并没有响应（可能是服务器软件没有启动），这时客户程序中的 connect ( ) 函数调用将会出错，其错误码为 10061，对应的符号常量为 WSAECONNREFUSED。一些常见的错误码及其含义请参见本书的附录。

WinSock 还允许用户使用 WSASetLastError()自己设置 WSAGetLastError()返回的错误码，该函数原型如下：

```
void WSASetLastError(int iError);
```

该函数无返回值，参数 iError 是用户自己设置的错误码，该错误码将被程序后面调用的 WSAGetLastError()函数返回（如果有的话）。

3.6 网络配置信息查询

在编写网络程序时，有时需要获取计算机的名字、IP 地址以及网络服务或是网络协议的相关信息等，为了实现这一目的，WinSock 提供了一系列函数来帮助应用程序完成相应的信息查询操作。

3.6.1 主机名字与 IP 地址查询

1. gethostname()

该函数用来查询本地计算机主机名字。

函数原型

```
int gethostname(char * name, int namelen)
```

函数参数

- name：指向用于存放主机名字的缓冲区，就是一个字符数组。
- namelen：是缓冲区的大小，也就是字符数组 name 的大小。

返回值

函数调用成功则返回 0；函数调用失败，则返回 SOCKET_ERROR，进一步的出错信息可调用 WSAGetLastError()获取。

2. gethostbyname()

该函数可根据主机名字(或 DNS 域名)查询主机的 IP 地址等信息，该查询操作就是通常所谓的“域名解析”。

函数原型

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

函数参数

name：指向主机名字的字符指针(字符串)，该名字除了可以是本机的名字外，还可以是网络上任意一台主机的域名(DNS)。

返回值

如果函数调用成功，将返回一个指向 hostent 结构的指针；如果函数调用失败，将返回一个空指针 NULL，调用 WSAGetLastError()可获取引起函数失败的错误代码。

返回指针指向的 hostent 结构由系统维护，该结构随线程的终结而消亡。应用程序不应该试图修改这个结构或者释放它的任何部分。hostent 结构包含对应给定主机名的 IP 地址等信息，其定义如下。

```
struct hostent
{
    char   * h_name;            //主机的规范名
    char   ** h_aliases;        //指向主机的别名列表
    int  h_addrtype;            //主机 IP 地址的类型(如 AF_INET 表示 IPv4)
    int  h_length;              //主机 IP 地址的长度(4)
    char ** h_addr_list;        //主机 IP 地址列表(网络字节顺序)
};
```

其中，字符指针 h_name 指向主机的正规名字；指向字符数组的指针 h_aliases 指向的是一个以空指针结尾的可选主机名队列；h_addrtype 保存主机地址的类型，对于 Windows Sockets 来说，其值总是 AF_INET；h_length 是每个地址的长度(字节数)，对应于类型 AF_INET 来说，这个域的值为 4；指针 h_addr_list 指向的是一个以空指针结尾的主机地址的列表，列表中的地址是以网络字节顺序排列的。

由于 gethostbyname()并不支持 IPv6 地址，因此新版规范中引入了该函数的替代版本 getaddrinfo()。同 inet_pton()与 inet_ntop()一样，在 Visual C++ 2017 中使用 gethostbyname()也需要关闭错误警告才能进行编译，关闭方法与使用 inet_aton()函数时完全一样。

3. getaddrinfo()

该函数用于从主机名字(或 DNS 域名)查询主机的 IP 地址，也可以根据服务名查询端

口号，它是协议无关的，既可以用于 IPv4，也可用于 IPv6。与 inet_pton()和 inet_ntop()函数一样，程序中使用 getaddrinfo()函数时也需要包含头文件 WS2tcpip.h。

函数原型

```
unsigned int getaddrinfo( const char * pNodeName, const char * pServiceName,
                          const struct addrinfo * pHints, struct addrinfo * * ppresult );
```

函数参数

- pNodeName：指向主机名字的字符指针(字符串)，该名字除了可以是本机的名字外，还可以是网络上任意一台主机的域名(DNS)，也可以是 IPv4 的点分十进制地址串或者 IPv6 的十六进制地址串。
- pServiceName：指定要查询端口号的服务名称或字符串形式的端口号，可以是 ASCII 形式的十进制数值表示的端口号，也可以是已定义的服务名称，如 ftp、http 等。
- pHints：指向某个 addrinfo 结构体的指针，该结构用于填写关于期望返回的信息类型的线索。例如，查询服务的端口号时，若指定的服务既使用 TCP 也使用 UDP，把 hints 结构中的 ai_socktype 成员设置成 SOCK_DGRAM 则函数将仅返回数据报套接字的信息。若不需要提供任何线索时，该参数可以为 NULL。
- ppresult：该参数是一个指向 addrinfo 类型的指针变量的指针，它所指向的指针变量用于保存作为函数查询结果的一个 addrinfo 结构体链表的头指针。

返回值

函数调用成功返回 0，函数调用失败将返回一个标识失败原因的 WinSock 错误代码，部分错误码的含义可参见附录。

函数的查询结果保存在一个元素类型为 struct addrinfo 的链表中，该链表由系统维护，函数只返回该链表的头指针保存于参数 ppresult 所指向的指针变量中。struct addrinfo 结构体类型的定义如下。

```
typedef struct addrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    size_t ai_addrlen;
    char * ai_canonname;
    struct sockaddr * ai_addr;
    struct addrinfo * ai_next;
} ADDRINFOA, * PADDRINFOA;
```

各成员的含义如下。

- ai_flags：标志。较少使用，在此不做解释，感兴趣的读者可自行查阅相关资料。
- ai_family：地址族，AF_INET 表示 IPv4，AF_INET6 表示 IPv6，AF_UNSPEC 表示协议无关。
- ai_socktype：套接字类型，取值为 SOCK_STREAM、SOCK_DGRAM 等。
- ai_protocol：协议类型，取值为 IPPROTO_IP，IPPROTO_IPV4，IPPROTO_UDP，IPPROTO_TCP 等。

- ai_addrlen：成员 ai_addr 指向的地址缓冲区的长度。
- ai_canonname：主机的规范名。
- ai_addr：指向 sockaddr 结构变量的指针。
- ai_next：addrinfo 结构类型的指针。在一个 addrinfo 结构体变量构成的链表中，ai_next 指向链表中的下一个 addrinfo 结构，链表中最后一个结构变量的 ai_next 应设置为 NULL。

4. gethostbyaddr()

该函数可根据传入的 IP 地址查询与该 IP 地址相对应的信息。

函数原型

```
struct hostent * gethostbyaddr(const char * addr, int len, int type);
```

函数参数

- addr：指向网络字节顺序 IP 地址的指针。
- len：地址的长度，在 AF_INET 类型地址中为 4。
- type：IPv4 地址对应的类型为 AF_INET。

返回值

如果函数调用成功，则函数返回一个指向 hostent 结构的指针；如果函数调用失败，将返回一个空指针 NULL，调用 WSAGetLastError()可获取引起函数失败的错误代码。

例 3.1 使用 gethostname()函数以及 getaddrinfo()函数查询本机的主机名称及 IPv4 地址。

```
#include <iostream>
#include "WinSock2.h"                          //包含 WinSock 库头文件
#include "ws2tcpip.h"
#pragma comment(lib, "ws2_32.lib")             //链接 WinSock 导入库
using namespace std;
int main(int argc, char **argv)
{
  /*** 加载 WinSock DLL ***/
  WSADATA wsaData;
  if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
  {
      cout << "加载 WinSock DLL 失败!\n";
      return 0;
  }
  char hostname[256];                          //用于存放获取的本机名称或输入的远程主机域名
  if (gethostname(hostname, sizeof(hostname)))  //获取本机名字
  {
      cout << "获取主机名字失败!\n" << endl;
      WSACleanup();
      return 0;
  }
  cout << "本机名字为: " << hostname << endl;
  /* 根据主机名字查询主机的 IPv4 地址 */
```

```
    struct addrinfo hints, * p_addrinfo, * p;
    memset(&hints,0, sizeof(hints));
    hints.ai_family = AF_INET;                    //只查询 IPv4 地址
    unsigned int retval = getaddrinfo(hostname, NULL, &hints, &p_addrinfo);
    if (retval != 0) {
        printf("getaddrinfo failed with error: %d\n", retval);
        WSACleanup();
        return 1;
    }
    /*** 输出 IP 地址 ***/
    p = p_addrinfo;
    cout << "本机 IP 地址: " << endl;
    char ipaddr[20];
    in_addr addr;
    while (p != NULL)
    {
        addr = ((sockaddr_in *)(p->ai_addr))->sin_addr;
        cout << inet_ntop(AF_INET, (void *)&addr, ipaddr, 20) << endl;
        p = p->ai_next;
    }
    WSACleanup();
    return 0;
}
```

如果不查询主机地址,而是直接从键盘输入因特网上某网站服务器的域名到字符数组hostname中,比如输入山东农业大学的Web服务器地址www.sdau.edu.cn,则可解析出该主机的IP地址。作为练习,请读者自己将该程序的功能改为输入主机域名输出相应服务器的IP地址。

3.6.2 服务查询

服务器通常都能提供多种不同的服务,比如Web服务、FTP文件服务、Telnet虚拟终端等,每种服务都要使用TCP或是UDP在一个知名端口号上侦听客户发来的服务请求。WinSock提供了一组函数可以查询本机上所提供的服务、协议及端口号,程序中可以使用这些函数来获得服务对应的端口,或者是获得正在使用指定端口的服务。在这里,服务的名称通常是对应服务器程序的名称。

1. getservbyname()

该函数根据给定的服务名和所使用的协议名获取服务的端口号等信息。

函数原型

```
struct servent * getservbyname(const char * name, const char * proto);
```

函数参数

- name:一个指向服务名的指针。
- proto:指向协议名的指针(可选)。如果这个指针为空,getservbyname()返回第一

个 name 与 s_name 或者某一个 s_aliases 匹配的服务条目。否则,getservbyname()对 name 和 proto 都进行匹配。

返回值

函数调用成功,函数将返回一个指向 servent 结构的指针,该结构由 WinSock 创建并管理,应用程序不能修改或者释放它的任何部分;函数调用不成功,将返回一个空指针 NULL,调用 WSAGetLastError()可获得到引起函数失败的错误所对应的错误码。servent 结构的声明如下:

```
struct servent {
    char * s_name;                    //服务名
    char ** s_aliases;                // 一个以空指针结尾的服务别名队列
    Short s_port;                     //连接该服务所需的端口号,以网络字节顺序排列
    char * s_proto;                   //连接该服务所用的传输协议名
};
```

该结构中 s_proto 所指向的传输协议名通常为 TCP 的协议名“TCP”或 UDP 的协议名称“UDP”。

2. getservbyport()

该函数根据给定的协议和端口号查询服务名称等信息。

函数原型

```
struct servent * getservbyport(int port, const char * proto);
```

函数参数

- port:给定的端口号,以网络字节顺序排列。
- proto:指向传输协议名的指针,可以为空指针。传输协议名通常为 TCP 或 UDP。如果为空指针,函数返回第一个端口号与 port 匹配的服务条目。否则返回端口号与 port 以及传输协议名与 proto 都匹配的服务条目。

返回值

如果函数调用成功,将返回一个指向 servent 结构的指针,该结构由 WinSock 创建并管理,应用程序不能修改或者释放它的任何部分;如果函数调用不成功,将返回一个空指针 NULL,调用 WSAGetLastError()可获得引起函数失败的错误所对应的错误码。

例 3.2 显示本机上所有的使用 TCP 的服务的名称及端口号。

```
#include "WinSock2.h"
#include "iostream"
#pragma comment(lib, "ws2_32.lib")
using namespace std;
int main(int argc, char ** argv)
{
    /*** 加载 WinSock DLL ***/
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        cout << "加载 WinSock DLL 失败!\n";
```

```
        return 1;
    }
    /*** 根据 TCP 端口号查询服务 ***/
    struct servent * pServer;
    cout <<"本机运行的使用 TCP 的服务:"<< endl;
    for(int i = 1;i < 65535;i++)
    {
        pServer = getservbyport(htons(i), "TCP"); //获取使用 TCP 的端口号为 i 的服务信息
        if(pServer!= NULL)                    //输出服务名称及端口号
            cout <<"服务:"<< pServer -> s_name <<" 端口:"<< ntohs(pServer -> s_port)<< endl;
    }
    //注销 WinSock DLL
    WSACleanup();
    return 0;
}
```

3.6.3 协议查询

因特网的每一个协议都有一个正式的名字,比如 IP、ICMP、TCP 等,但在网络内部,协议并不是使用它们的名字标识的,而是使用分配给它们的一个唯一编号来表示的,比如 IP 的编号为 0,ICMP 的编号为 1,TCP 的编号为 6 等。因特网协议的编号是由 IANA 统一管理的,它是一个范围在 0～255 的整数。

协议的名字主要为了便于人们阅读,而协议的编号则有利于计算机处理。WinSock 提供了两个函数 getprotobyname()和 getprotobynumber()来帮助程序完成协议名字和编号之间的转换。

1. getprotobyname()

根据协议的名字查询相应的协议编号等信息。

函数原型

```
struct protoent * getprotobyname(const char * name);
```

函数参数

name:一个指向协议名的指针。

返回值

函数调用成功,将返回一个指向 protoent 结构的指针,该 protoent 结构由 WinSock 创建并管理,应用程序不能修改或者释放它的任何部分;函数调用失败则返回一个空指针,应用程序可以通过 WSAGetLastError()来获取错误代码。

protoent 结构的声明如下:

```
struct protoent {
    char p_name;                          //正规的协议名
    char ** p_aliases;                    //一个以空指针结尾的可选协议名队列
    short p_proto;                        //主机字节顺序的协议号
};
```

例如,查询 UDP 的协议号可用如下代码实现。

```
struct protoent * pProto;
pProto = getprotobyname("UDP");
cout << "UDP 的协议编号为: " << pProto -> p_proto << endl;
```

2. getprotobynumber()

根据协议号查询协议的名字等信息。

函数原型

```
struct protoent * getprotobynumber(int number);
```

函数参数

number:一个以主机字节顺序排列的协议号。

返回值

函数调用成功,将返回一个指向 protoent 结构的指针,该 protoent 结构由 WinSock 创建并管理,应用程序不能修改或者释放它的任何部分;函数调用失败则返回一个空指针,应用程序可以通过 WSAGetLastError()来获取错误代码。

例 3.3 显示本机运行的所有因特网协议的名称及相应的协议号。

```
#include "iostream"
#include "WinSock2.h"
#pragma comment(lib, "ws2_32.lib")
using namespace std;
int main(int argc, char ** argv)
{
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        cout << "加载 WinSock DLL 失败!\n";
        return 1;
    }
    struct protoent * pProto;
    for (int i = 0; i <= 255; i++)
    {
        if ((pProto = getprotobynumber(i)) != NULL)
            cout << "协议名:" << pProto -> p_name << " 编号:" << pProto -> p_proto << endl;
    }
    WSACleanup();
    return 0;
}
```

3.6.4 异步信息查询函数及其编程方法

前面所介绍的函数都与标准的 Berkeley 套接字函数兼容,当函数调用后,如果函数的功能没有执行完成,函数是不会返回的,紧跟在这些函数调用之后的语句在函数未返回前是无法执行的。如果一个函数要完成其功能需要花费较长时间,则应用程序也只能在这个函数上等待而不能执行其他操作(这种情况称为阻塞),一直到函数完成功能返回或者函数出

错返回。套接字函数的这种工作模型被称为同步模型。

与同步模型所对应的是异步模型，这里所谓的“异步”，也就是指函数启动了规定的任务后立即返回调用方，并返回一个用来标识所启动任务的异步任务句柄。

为了适应 Windows 的消息驱动机制，同时也可以解决同步模型中那些耗时较多的套接字函数所引起的程序执行效率低的问题，WinSock2 引入 Windows 的消息机制对近二十个套接字函数进行了扩展，就是所谓的异步扩展，扩展后的函数都以“WSA”开头。

在扩展后的“异步”模型中，套接字函数通常只是启动一个任务并向 Windows 系统注册一条消息后就立即返回，尽管所启动的任务还没有完成。函数返回后应用程序可以执行其他操作。当异步函数所启动的任务完成时，Windows 系统就会向应用程序发送那条函数注册的消息，收到消息后，应用程序就可以根据任务完成的结果继续下一步的操作。

下面以 gethostbyname()函数的异步扩展版本 WSAAsyncGetHostByName()函数为例，介绍 WinSock2 的异步函数的格式以及编程方法。

与 gethostbyname()函数一样，WSAAsyncGetHostByName()函数的功能是根据主机域名获取主机的 IP 地址等主机信息。函数被调用时，如果给定的参数均有效，该函数将初始化并启动查询操作后立即返回，返回值是异步任务句柄。在多数情况下，应用程序应该保存返回的句柄，它主要有两个用途：一是区分不同的查询操作，主要用于应用程序发起了多个查询操作时，可以根据该句柄判断到底是哪一个查询操作完成了；二是要取消该异步任务时使用，如果应用程序成功调用了该函数后，在规定的时间内一直没收到完成的消息，那么可通过该句柄调用 WSACancelAsyncRequest()取消该操作。

WSAAsyncGetHostByName()函数的格式如下。

```
HANDLE WSAAsyncGetHostByName(
HWND hWnd,
unsigned int wMsg,
const char FAR * name,
const char FAR * buf,
int buflen
);
```

函数参数

- hWnd：是一个窗口句柄，表示异步请求完成时应该收到本函数发出的消息的窗口。
- wMsg：当异步请求完成时，hWnd 代表窗口将要收到的消息，一般是用户自定义的消息。
- name：指向主机的名字的字符指针(字符串)，主机名字既可以是本机的名字，也可以是网络上任意一台主机的域名(DNS)。
- buf：接收 hostent 数据结构的数据区指针，该指针指向的数据区必须要比 hostent 结构所占用的存储空间大，这是因为该缓冲区不仅要容纳一个 hostent 结构，而且 hostent 结构所有成员所引用的所有数据也要保存在该存储区内。建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区。
- buflen：buf 所指缓冲区的大小。

返回值

函数返回值为函数所启动的异步查询任务句柄。

如果函数启动的异步查询任务执行成功，则将主机名和地址信息复制到 buf 缓冲区中，

同时向句柄为 hWnd 的应用程序窗口发送消息 wMsg。

消息的 wParam 参数包含 WSAAsyncGetHostByName()函数在调用时返回的异步任务句柄。

消息的 lParam 参数的高 16 位包含着错误代码,如果成功该错误代码为 0,若该错误代码为 WSAENOBUFS,则说明 buflen 指出的缓冲区太小了,此时,lParma 的低 16 位为实际所需缓冲区的大小。获取错误代码和 lParma 的低 16 位值可使用如下的宏。

```
WSAGETASYNCERROR(lParma);
WSAGETASYNCBUFLEN(lParma);
```

函数的返回值指出异步操作是否成功启动,但并不能表明异步操作是否成功执行。若操作启动成功,返回该异步请求任务的句柄(一个 HANDLE 类型的非 0 值)。如果启动不成功则返回 0,此时调用 WSAGetLastError()可获取引起启动失败的错误代码。

需要特别说明,WSAAsyncGetHostByName()函数现在已不被提倡使用,在 Visual C++ 2017 中使用该函数时,编译器将发出错误警告并停止编译,如果要关闭错误警告继续编译,可以在头文件 stdafx.h 中的头部预处理命令 #include "targetver.h"之下,添加如下宏定义。

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
```

或者在主对话框类的头文件中,在类定义之前添加:

```
#pragma warning(disable : 4996)
```

通过上面函数的介绍,可以知道,在使用异步套接字函数编程时:

(1) 程序必须有至少一个窗口,用于接收异步任务完成时发送给程序的消息;

(2) 必须为应用程序添加一条自定义消息,用于函数调用时作为参数与函数所启动的异步任务相关联;

(3) 需要编写该自定义消息的消息处理函数,用于处理任务完成后返回的结果。

下面的例题演示了使用 WSAAsyncGetHostByName()函数编程的方法。

例 3.4 编写程序查询本机的主机名称及 IP 地址。要求使用 WSAAsyncGetHostByName()函数。程序的运行界面如图 3.2 所示。

图 3.2 例 3.4 的程序界面

程序编写步骤如下。

(1) 使用 MFC 应用程序向导创建一个基于对话框的应用程序框架,取该项目名称为“Example3.4”;创建项目过程中需要选中“Windows 套接字”复选框,因为该项目中要调用 WinSock 函数。

调整对话框上原有的一个静态文本控件(ID_STATIC1)和两个命令按钮(IDOK、IDCANCLE)的位置,并添加一个静态文本框(ID_STATIC2)、一个编辑框(IDC_EDIT1)和一个列表框 ListBox(IDC_LISTBOX1),调整它们的布局如图 3.2 所示,并按表 3.1 给出的值设置相应控件的 Caption 属性。

表 3.1 例 3.4 各控件的属性设置

控件的 ID	Caption 属性	Read Only 属性
ID_STATIC1	计算机名	无
ID_STATIC2	IP 地址	无
IDC_EDIT1	无	True
IDOK	显示主机名字与 IP 地址	无
IDCANCLE	退出	无

(2) 编辑框(IDC_EDIT1)和列表框 ID_LISTBOX1 添加 Control 类别的控件变量。在主对话框的“对话框资源编辑器”中右击编辑框(IDC_EDIT1),在弹出菜单中单击“添加成员变量”按钮,在弹出的“添加成员变量窗口”中,输入成员变量名称为“m_sName”,类别设为默认的 Control,变量类型保持默认值,单击“确定”按钮,成员变量添加完成。

按照同样方法为列表框 ID_LISTBOX1 添加 Control 类别的控件变量 m_CtrListIP。

(3) 由于 WSAAsyncGetHostByName()函数需要一个字符类型的缓冲区存放返回的 hostent 结构及 hostent 结构的成员引用的所有数据,因此调用该函数前需要事先定义该缓冲区。定义该缓冲区可直接在头文件 Example3.4Dlg.h 的类定义中添加如下代码。

```
char buf[MAXGETHOSTSTRUCT];
```

其中,MAXGETHOSTSTRUCT 是一个常量,表示存放 hostent 结构及相关数据最多所需要的存储空间大小。

(4) 如果在使用“应用程序向导”创建该程序框架时,没有在“高级功能”窗口中选择“Windows 套接字”复选框,则需要手动添加 WinSock 动态链接库的加载代码。在 Example3.4Dlg.cpp 中的类成员函数 BOOL CExample34Dlg::OnInitDialog()中添加如下代码。

```
WORD wVersionRequested;
WSADATA wsaData;
wVersionRequested = MAKEWORD(2,2);          //生成版本号 2.2
if(WSAStartup(wVersionRequested,&wsaData)!= 0) return false ;
```

如果已选中“Windows 套接字”复选框,则不需要该步骤。

(5) 添加命令按钮的消息处理函数。

打开主对话框资源编辑器,双击对话框资源编辑器中的“确定”按钮,在打开的代码资源编辑器中就会看到“确定”按钮的 BN_CLICKED 消息的处理函数已被添加进来了,按如下所示代码添加该函数所要完成的功能。

```
void CExample34Dlg::OnBnClickedOk()
{
    // TODO: 在此添加控件通知处理程序代码
    char hostname[32];
    if( gethostname(hostname,sizeof(hostname))) //获取主机名
    {//如果出错,在窗口中的静态文本框 ID_STATIC2 上显示出错信息
        m_sName.SetWindowTextW(_T("gethostname calling error\n"));
                    //本例使用的是 Unicode 编码,若使用 ANSI 编码需用 SetWindowTextA()方法
    }
    else
    {
        CString a(hostname);
        m_sName.SetWindowTextW(a);          //在静态文本框 ID_STATIC2 上显示计算机名
        WSAAsyncGetHostByName(this->GetSafeHwnd(), MY_MESSAGE, hostname,
                                    buf, sizeof(buf)); //启动获取主机信息的异步事件
    }
}
```

该函数首先调用 gethostname()函数获取主机名,然后 WSAAsyncGetHostByName()获取主机的 IP 地址信息并发送自定义消息 MY_MESSAGE。

在上述代码中,WSAAsyncGetHostByName()函数的第二个参数 MY_MESSAGE 是用户自定义的消息,获取主机网络信息的异步任务执行完后将发送该消息,该消息将由第一个参数指定的窗口(在这里为本对话框)处理。在本程序中,希望主对话框即 OnBnClickedOk()所属的对话框捕获并处理 WSAAsyncGetHostByName()函数在获取主机网络信息的异步任务完成后所发送的通知消息 MY_MESSAGE,因此 WSAAsyncGetHostByName()的第一个参数应为 OnBnClickedOk()函数所属的对话框的句柄。获取本对话框的窗口句柄可以用 this-> GetSafeHwnd()函数。

(6) 添加自定义消息 MY_MESSAGE 的处理程序。

首先在文件 Example3.4Dlg.h 的类定义前添加如下代码来定义一个消息。

```
#define MY_MESSAGE      WM_USER +100
```

启动 MFC 类向导,选择类名为 CExample34Dlg,选择“消息”选项卡,单击位于“消息”选项卡下部的“添加自定义消息”按钮,弹出“添加自定义消息”对话框,在“消息”文本框中输入“MY_MESSAGE”,消息处理程序名称采用默认的 OnMyMessage。单击“确定”按钮返回。

单击 MFC 类向导中的“编辑代码”按钮,在打开的 Example3.4Dlg.cpp 中的成员函数 afx_msg LRESULT CExample34Dlg::OnMyMessage()中添加相应代码,得到如下函数。

```
afx_msg CExample34Dlg ::OnMyMessage(WPARAM wParam, LPARAM lParam)
{
    struct hostent * hptr;
    CString mtempstr;          //用于临时保存点分十进制表示的 IP 地址的字符串变量
    hptr = (struct hostent * )&buf;
    char ** pptr;
    pptr = hptr->h_addr_list;
    int itemCount = m_CtrListIP.GetCount(); //获取列表框控件显示内容的条数
    for(int i = 0;i < itemCount;i++)
    m_CtrListIP.DeleteString(0);            //清空列表框
```

```
    for(; * pptr!= NULL;pptr++)
    {   //读取 IP 地址并转换为点分十进制格式后添加到列表框中
        mtempstr = inet_ntoa( * (struct in_addr * )( * pptr));
        m_CtrListIP.AddString(mtempstr) ;
    }
    return 0;
}
```

该函数在窗口收到自定义消息 MY_MESSAGE 后执行,其主要功能是在窗口的列表框中显示计算机的所有 IP 地址。MY_MESSAGE 消息是由 OnBnClickedOk()函数调用的 WSAAsyncGetHostByName()函数发出的,该函数在发出消息前,已获取了本机的 IP 地址等相关信息并存入了 buf 指向的缓冲区中。因此该函数所做的工作就是从 buf 中读出有关信息,并根据这些信息将 IP 地址显示在列表框中。

由于 WSAAsyncGetHostByName()函数参数类型的要求,buf 被定义成了字符数组类型,直接读取其中的相关信息并不方便。但是,buf 指向的数据区的最前面包含一个 hostent 结构,通过该 hostent 结构的成员就可以访问网络的 IP 地址等信息。因此,函数中先定义一个 struct hostent 类型的指针 hptr,并使该指针指向 buf 的首地址,通过该指针就可以访问 buf 中的 hostent 结构,进而通过 hostent 结构中的 IP 地址列表指针 h_addr_list 就可以读取网络 IP 地址了。

(7) 为了能让 Visual C++ 2017 忽略对 WSAAsyncGetHostByName()函数的警告而顺利编译,在头文件 stdafx.h 的头部,预处理命令 #include "targetver.h"之下,添加如下宏定义。

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
```

至此,该程序就完成了。编译并运行便可显示如图 3.2 所示的程序界面,单击“显示主机名字与 IP 地址”命令按钮,便在文本框中显示本机的计算机名与本机的 IP 地址。

除了 WSAAsyncGetHostByName()函数外,WinSock 还包括 gethostbyaddr()、getservbyname()、getservbyport()、getprotobyname()、getprotobynumber()等函数的异步扩展版本,它们的函数原型及使用方法与 WSAAsyncGetHostByName()相似,需要学习的读者可查看 Visual Studio 的帮助文档,这里不再一一举例说明。

习题

1. 选择题

(1) 在 Visual C++程序设计中,通常使用 WinSock 2.2 实现网络通信的功能,则需要引用的头文件为(　　)。

A. Winsock.h　　B. winsock2.h　　C. winsock22.h　　D. winsock2.2.h

(2) WinSock2 的开发组件不包括(　　)。

A. 头文件是 WinSock2.h　　B. 导入库文件 Ws2_32.lib

C. Windows Sockets 实现文档　　D. WinSock 动态链接库

(3) 运行在不同系统平台上的进程间交换数据时,(　　)。

A. 数据中的整数需要转换为网络字节顺序,浮点数和汉字则不需要

B. 数据中整数和汉字需要转换为网络字节顺序,浮点数则不需要

C. 数据中整数和浮点数需要转换为网络字节顺序,汉字则不需要

D. 数据中整数、浮点数和汉字均需要转换为网络字节顺序

(4) 以下叙述错误的是(　　)。

A. in_addr 结构中的 IP 地址是网络字节顺序存储的

B. in_addr 结构可以同时存储 3 种不同方式表示的 IPv4 地址

C. sockaddr_in 结构变量可以保存地址族、端口号和 IP 地址三种地址信息

D. sockaddr_in 结构变量与 sockaddr 结构变量中存储的内容与顺序完全一致

(5) 一个进程要想将数据发送给网络上的另一进程必须要知道对方的网络进程地址,网络进程地址包括(　　)。

A. IP 地址、端口号　　B. IP 地址、传输层协议、端口号

C. IP 地址、进程号　　D. 进程号

(6) 可以将 in_addr 结构中的 IP 地址转换为点分十进制字符串方式的函数是(　　)。

A. inet_addr()　　B. inet_aton()　　C. inet_ntoa()　　D. ntohs()

(7) 下面能实现把端口号从本机字节顺序转换到网络字节顺序的函数是(　　)。

A. htons()　　B. htonl()　　C. ntohl()　　D. ntohs()

(8) 为了解决在不同体系结构的主机之间进行数据传递可能会造成歧义的问题,以下常用来对双字节或者四字节数据进行字节顺序转换的函数是(　　)。

A. htons()/htonl()/ntohs()/ntohl()

B. inet_addr()/inet_aton()/inet_ntoa()

C. gethostbyname()/gethostbyaddr()

D. (struct sockaddr *)&(struct sockaddr_in 类型参数)

(9) 关于 getaddrinfo()函数,下列叙述不正确的是(　　)。

A. 可以通过本地主机名称或 DNS 域名来查询相应主机的 IP 地址等信息

B. 使用该函数必须包含头文件

C. 可以完全取代 gethostbyname()函数的功能

D. 返回值为保存查询结果的 struct addrinfo 结构变量的指针 WS2tcpip.h

(10) 异步查询函数 WSAAsyncGetHostByName()成功执行并返回后,(　　)。

A. 会将本地主机的 IP 地址等信息存放在函数参数指定的存储区中

B. 会发送消息给指定窗口,通知窗口处理查询到的结果

C. 会返回一个指向 hostent 结构变量的指针

D. 只是初始化并启动了一个异步查询任务,查询任务并没有完成

2. 填空题

(1) 套接字编程接口源于美国加州大学伯克利分校开发推广的一个包括 TCP/IP 的操作系统_____,它是该操作系统提供给其用户的网络应用程序编程接口。

(2) WinSock API 函数中的 Windows 扩展函数,函数名都以_____为前缀,主要是为了便于程序员充分利用 Windows 的消息驱动机制编程而提供的。

(3) 在程序中链接 WinSock2 导入库 Ws2_32.lib 使用的编译预处理命令是_____。

(4) 当程序使用完 WinSock.dll 提供的服务后,应用程序应调用_____函数,来解除与 WinSock.dll 库的绑定,释放 WinSock 实现分配给应用程序的系统资源。

(5) inet_aton()函数的功能是将其第一个参数指向的点分十进制字符串表示的IP地址转换成_____,存放在第二个参数指向的in_addr结构中。

(6) 函数inet_addr()的原型为:

```
unsigned long inet_addr(const  char *cp);
```

其参数cp指向的是一个点分十进制字符串表示的IP地址,该函数如果调用成功,则其返回值是_____。

(7) 函数WSAGetLastError()的返回值是一个整数,该整数是上一次WinSock函数调用出错时的_____。

(8) 语句struct hostent *hptr = gethostbyname("www.sdau.edu.cn");的功能是_____。如果使用getaddrinfo()函数实现该功能,则相应的语句为_____。

(9) 执行下列程序代码的输出结果是_____。

```
in_addr a ;
a.S_un.S_addr = htonl(0xc0a8021a);
char ip[20];
cout << inet_ntop(AF_INET, &a, ip, sizeof(ip))<< endl;
```

(10) 语句inet_pton(AF_INET, "192.168.1.1", &a)的功能是_____;其中变量a的类型为_____。

3. 简答题

(1) 如果系统支持WinSock2,编程时能否使用头文件Winsock.h和导入库Wsock32.lib?为什么?

(2) 将WinSock导入库链接到应用程序有哪两种方式?如何操作?

(3) sockaddr_in结构与sockaddr结构有何异同?当使用TCP/IP时,如何为sockaddr结构变量赋值?举例说明。

(4) 由于网络原因,调试程序时发送数据总是不成功,问采用何种方法可找出发送数据不成功的具体原因?

(5) 什么是网络字节顺序?当网络通信程序在发送整数时为什么应该使用网络字节顺序?

4. 编程题

(1) 编写一个控制台应用程序,调用gethostname()和gethostbyname()函数,查询并显示本机的主机名称及IP地址。

(2) 编写一个控制台应用程序,查询并显示TCP和UDP的协议编号。

实验2 查询主机网络配置信息

一、实验目的

(1) 掌握WinSock动态链接库的加载和注销方法;

(2) 熟悉并掌握gethostname()、getaddrinfo()、getservbyname()、getservbyport()、getprotobyname()、getprotobynumber()等函数的功能及用法;

(3) 掌握使用异步信息查询函数的编程方法。

二、实验设备及软件

运行 Windows 系统的计算机,Visual Studio 2017(已选择安装 MFC)。

三、实验内容

(1) 编写一个控制台应用程序,查询并显示本机运行的所有因特网协议的名称及相应的协议号。

(2) 编写一个控制台应用程序,从键盘输入一远程主机的 DNS 域名,使用 getaddrinfo()函数解析该主机的 IP 地址。

(3) 编写一个控制台应用程序,调用 getprotobyname(),查询并显示 TCP 和 UDP 的协议编号。

(4) 编写一个对话框应用程序,程序界面如图 3.3 所示。程序功能:在文本框中输入域名后,单击"解析域名"命令按钮便可在下面的列表框中显示解析后的域名。要求使用 WSAAsyncGetHostByName()函数实现。

图 3.3 实验 2 实验内容(4)的程序界面

四、实验步骤

实验内容 1、2、3 的步骤请参阅 3.6 节各部分内容。下面简要给出实验内容 4 的步骤。

(1) 使用 MFC 应用程序向导创建一个基于对话框的应用程序,注意创建项目过程中需要选中"Windows 套接字"复选框,因为该项目中要调用 WinSock 函数。

(2) 按如图 3.3 所示界面为程序添加控件并设置控件的相关属性。

(3) 为文本编辑框和列表框添加控件变量,变量名自己指定,编辑框对应的变量类别为 Value,列表框控件变量的类别为 Control。这里分别取为 m_Edit 和 M_List。

(4) 为 WSAAsyncGetHostByName()函数定义一个字符类型的缓冲区,该缓冲区用于存放函数返回的 hostent 结构和 hostent 结构的成员所引用的数据。该缓冲区可定义为主对话框类的成员,即可在对话框类的定义中添加类似于下面的代码。

```
char buf[MAXGETHOSTSTRUCT];
```

(5) 使用类向导为程序添加一条自定义消息,消息名称由编程者自己指定,这里是 WM_MESSAGE,消息处理函数名称采用默认名。注意,自定义消息的名称需要在 stdax.h 文件或 resource.h 文件中使用编译预处理命令 #define 定义。

(6) 编写"解析域名"按钮的 BN_CLICKED 消息的消息处理函数,该函数调用 WSAAsyncGetHostByName()启动获取主机信息的异步事件。具体代码如下。

```
char hostname[256];
UpdateData();
if(!m_Edit.IsEmpty())//如果已输入域名
```

```
{
    strcpy(hostname,m_Edit.GetBuffer());
                              //将 CString 对象 m_Edit 中 D 的字符串复制到字符数组 hostname
    WSAAsyncGetHostByName(this->GetSafeHwnd(),WM_MESSAGE,hostname,buf,sizeof(buf));
                                                  //启动获取主机信息的异步事件
}
```

(7) 编写自定义消息的处理函数,该函数将 WSAAsyncGetHostByName()函数返回的 hostent 结构中的 IP 地址信息显示在列表框中。具体代码如下。

```
struct hostent *hptr;
CString mtempstr;       //用于临时保存点分十进制表示的 IP 地址的字符串变量
hptr = (struct hostent *)&buf;
char **pptr;
pptr = hptr->h_addr_list;
int itemCount = m_List.GetCount();                          //获取列表框控件显示内容的条数
for(int i = 0;i < itemCount;i++)m_List.DeleteString(0); //清空列表框
for(;*pptr!= NULL;pptr++){  //读取 IP 地址转换为点分十进制格式后添加到列表框中
    mtempstr = inet_ntoa(*(struct in_addr *)(*pptr));
    m_List.AddString(mtempstr) ;
```

(8) 在头文件 stdafx.h 的头部预处理命令 #include "targetver.h"之下,添加如下宏定义:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
```

(9) 编译运行程序,在文本框中输入“www.sdau.edu.cn”“www.tup.com.cn”等网址进行验证。

五、思考题

实验内容(4)中,如果不使用 WSAAsyncGetHostByName()函数而是使用 getaddrinfo()函数是否可以?如果可以的话程序应如何实现?