第13章 TCP/IP网络编程实践

- 网络应用程序的标识问题
- 客户/服务器模型
- TCP/IP应用程序工作模型
- TCP网络应用编程
- UDP网络应用编程
- UDP广播编程

13.1 网络应用程序的标识问题

- 同一系统中不同进程间进行通信时，通过系统分配的进程号(Process ID)就可以唯一标识一个进程
  - 要通信的进程只要知道对方的进程号就可以进行通信
- 网络情况下不能只简单地用进程号来标识不同的进程，因为各主机都独立地分配其进程号
- 为了唯一地标识网络中通信的一个进程(即通信的某一方)，就要使用一个如下的三元组：
  - (本地协议，本地IP地址，本地端口号)

网络应用程序的标识问题

- 一个三元组只指定了通信时半条连接，即通信的一方，因而称为半相关(Half-association)。
- 要完整地表示网络中进行通信的两个进程，那么就要使用一个如下结构的六元组：
  - (本地协议，本地地址，本地端口号，远地协议，远地地址，远地端口号)
- 通信的两个进程在端到端的传输层只能使用相同的协议。
- 一个完整的网间通信就可以简化为用一个五元组来标识通信的两个进程：
  - (协议，本地IP地址，本地端口号，远程IP地址，远程端口号)

13.2 客户/服务器 模型

- 客户/服务器 模型(C/S)是一种通信模式
- 一个应用程序被动地等待，而另一个应用程序通过请求启动通信
- 客户和服务器分别指两个应用程序：
  - 客户端（client）：发出请求方（主动）
  - 服务器（server）：对客户端的请求做出响应（被动）

客户/服务器模型的特点

对于服务器方来说：

- 服务进程一般在启动后就一直运行，以等待客户请求的到来
- 服务器方进程使用的是众所周知的端口
- 一般，服务器拥有较多的资源，具有运算能力强，数据存储容量大，通信速度快，系统的可靠性高等优点
- 服务器方进程一般并行处理一定数量的客户请求
- 服务器方在通信时属于被动的一方

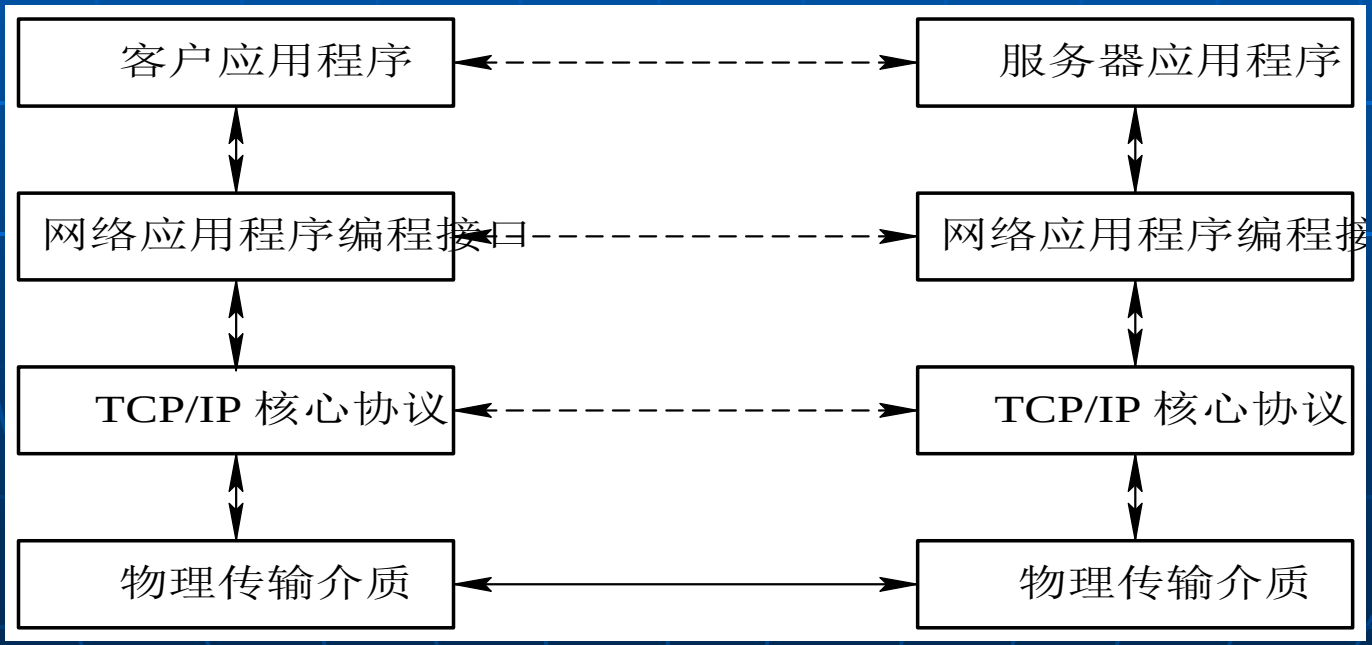
客户/服务器模型的特点

对于客户方来说：

- 在需要时向服务器提出请求，请求的服务得到满足并完成处理任务后，就终止客户程序的执行
- 向系统申请临时端口与服务器方进程进行通信
- 一般客户拥有较少的资源，在各方面的性能一般要比服务器差
- 客户方在通信时属于主动的一方

13.3 TCP/IP应用程序工作模型

- TCP/IP协议的核心内容在层次结构的低三层，即网络接口层、IP层和传输层。这三层的功能一般是由操作系统的内核来实现
- 两台主机的进程间通过网络编程接口进行通信的原理图：



TCP/IP应用程序工作模型

使用TCP或UDP端口号作为一个服务器程序的标识

- server启动时在本地主机注册TCP或UDP端口号
- client使用临时端口向server的端口发送信息
- 如果使用的是TCP协议，则client与server先建立连接，然后发送信息；而UDP则不用
- 运行server的主机在注册的端口上收到信息后，将信息转交给注册该端口的server程序处理

13.4 TCP网络应用编程

服务器程序流程（多线程）：

- 初始化并创建SOCK_STREAM类型的socket
- 填写本机地址信息
- 绑定并监听一个固定的端口
- 收到client的连接后建立一个新的socket连接
- 创建线程，并利用该socket与client进行通信
- 子通信结束后中断与client的连接
- 主进程回到第4步，周而复始，直至退出程序

客户端程序流程：

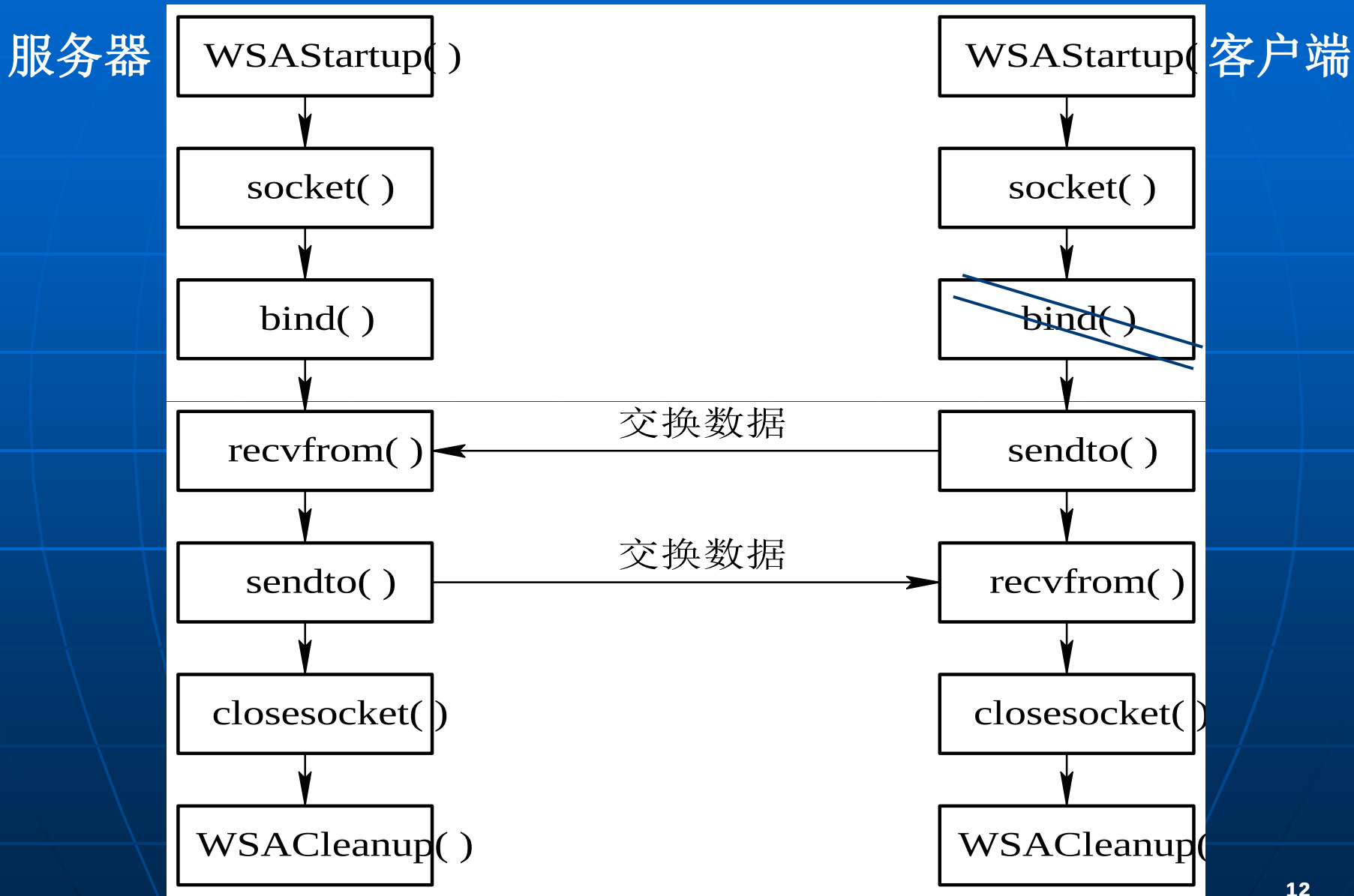
- 初始化并创建SOCK_STREAM类型的socket
- 填写服务器地址信息
- 连接服务器
- 与服务器进行通信
- 通信结束后断开连接

客户端与服务器TCP通信建立过程

13.5 UDP网络应用编程

- 无连接的数据报(SOCK_DGRAM)传输服务在传输层使用UDP协议。
- 通信前不需要在客户和服务器之间先建立连接。
- 通信时先发送数据的一方(client)需要知道另一方(server)的端口，而另一方通过捕获数据包得知发送方的地址和端口
- 在数据传输完成后，只要关闭套接口，释放网络资源，通信过程就结束了。

无连接的客户/服务器程序工作流程



无连接的客户/服务器程序工作流程

(1)使用WSAStartup函数检查系统协议栈的安装情况。

(2)使用socket函数创建套接口，以确定相关五元组的协议。

(3) 使用bind函数将创建的套接口与本地地址绑定(服务器方)

(4)使用sendto函数发送数据，使用recvfrom函数接收数据。

(5)使用closesocket函数关闭套接口。

(6)调用WSACleanup函数，结束socket的使用。

至此，一次无连接的数据报传输过程结束。

无连接的客户/服务器程序工作流程

无连接的客户/服务器编程注意事项：

（1）客户方可以不用bind函数绑定IP地址和端口

（2）若没有绑定地址，直接用sendto函数发数据时，系统会自动分配地址和端口

（3）在recvfrom函数接收数据前，套接字必须有指定的本地地址和端口

（4）服务器的一方必须先启动，否则客户请求传不到服务进程

（5）一般由不绑定IP地址和端口的一方首先向绑定地址的一方发送数据

（6）发送数据时，发送方除指定本地套接口的地址外，还需指定接收方套接口的地址，从而在数据收发过程中动态地建立全相关

UDP网络通信程序实例

一个简单的UDP客户/服务器程序实例

程序说明：

- 程序使用的编程环境为Visual C++6.0，使用的是控制台程序“Win32 Console Application”
- 通信协议使用无连接的UDP协议(SOCK_DGRAM)
- 服务器端IP地址使用系统指定的IP地址，端口号在程序中指定为5050，用符号常量定义
- 服务器端从客户端接收并显示信息，客户端向服务器发送的信息为“Hello! I am a client.”
- 服务器也向客户端发送“Hello! I am a server.”信息，客户端接收从服务器发送的信息并进行显示

UDP服务器源代码

```
/********************* 服务器端程序 ********************************************
调试环境：VC++6.0
程序名称：UDPserver.cpp
服务器IP地址：由系统指定
服务器接收端口：5050
服务器发送端口：由系统指定
功能：
1.从端口5050接收客户端发送来的数据，接收成功后显示收到的数据、客户端的IP地址和端口号；
2.向客户端发送"Hello! I am a server."字符串
*****************************************************************************/

#include<Winsock2.h>
#include<stdio.h>
#include<stdlib.h>
//指定链接库
#pragma comment (lib, "ws2_32.lib")
```

```
//服务器端口号为5050
#define DEFAULT_PORT  5050
//接收数据缓冲区长度
#define BUFFER_LENGTH  1024
void main( )
{
     int        iPort=DEFAULT_PORT;
     WSADATA    wsaData;
     SOCKET     sSocket;
     //客户地址长度
     int        iLen;
     //发送的数据长度
     int        iSend;
     //接收的数据长度
     int        iRecv;
     //要发送给客户的信息
     char       send_buf[ ]="Hello! I am a server.\0";
     //接收数据的缓冲区
     char       recv_buf[BUFFER_LENGTH];
     //客户端地址
     struct sockaddr_in  ser , cli;
     //初始化Socket
     if(WSAStartup(MAKEWORD(2,2),&wsaData)!=0)
     {
        printf("Failed to load Winsock.\n ");
        return;
     }
```

```
//产生服务器端套接口
sSocket = socket(AF_INET,SOCK_DGRAM,0);
if(sSocket == INVALID_SOCKET)
{
   printf("socket( ) Failed: %d\n", WSAGetLastError());
   return;
}
//建立服务器端地址
ser.sin_family=AF_INET;
ser.sin_port=htons(DEFAULT_PORT);
ser.sin_addr.s_addr=inet_addr("192.168.132.128");
//绑定地址
if( bind(sSocket, (const sockaddr*)&ser, sizeof(ser)) == SOCKET_ERROR )
{
    printf("bind() Failed: %d \n", WSAGetLastError());
    return;
}
//服务器开始运行
printf("  Server start ...  \n\n");
iLen=sizeof(cli);
```

```
//进入一个无限循环，等待接收客户端数据
while(1)
{
    //初始化接收缓冲区
    memset(recv_buf,'\0',sizeof(recv_buf));
    //从客户端接收数据
    iRecv=recvfrom(sSocket,recv_buf,BUFFER_LENGTH,0,(SOCKADDR*)&cli,&iLen);
    if(iRecv==SOCKET_ERROR)
    {
        printf("recvfrom( ) Failed:%d \n", WSAGetLastError( ));
        continue;
    }
    else    if(iRecv==0)
        continue;
    else
    {
        //输出接收到的数据
        printf("\n---------------------------------------\n");
        printf("  recvfrom( ): %s\n",recv_buf);
        //输出客户IP地址和端口号
        printf("  Accepted client IP:[%s],port:[%d]\n",
              inet_ntoa(cli.sin_addr), ntohs(cli.sin_port));
    }
```

```
    //给客户发送信息
    iSend=sendto(sSocket,send_buf,strlen(send_buf)+1,0,
        (SOCKADDR*)&cli,sizeof(cli));
    if(iSend==SOCKET_ERROR)
    {
        printf("  sendto( ) Failed.: %d\n",     WSAGetLastError( ));
    }
    else    if(iSend==0)
    {    }
    else
    {
        printf("  sendto( ) succeeded!\n");
    }
}

closesocket(sSocket);
WSACleanup( );
}
```

UDP客户端源代码

```
/********************* 客户端程序 ************************************
调试环境：VC++6.0
程序名称：UDPclient.cpp
客户端IP地址和端口：由系统指定
程序功能：
1.客户端程序向服务器发送数据“Hello! I am a client.”；
2.客户端程序从服务器接收数据并进行显示。
*********************************************************************/

#include<Winsock2.h>
#include<stdio.h>
//指定链接库
#pragma comment (lib, "ws2_32.lib")
//服务器端口号为5050
#define Ser_PORT  5050
//缓冲区长度
#define DATA_BUFFER   1024
```

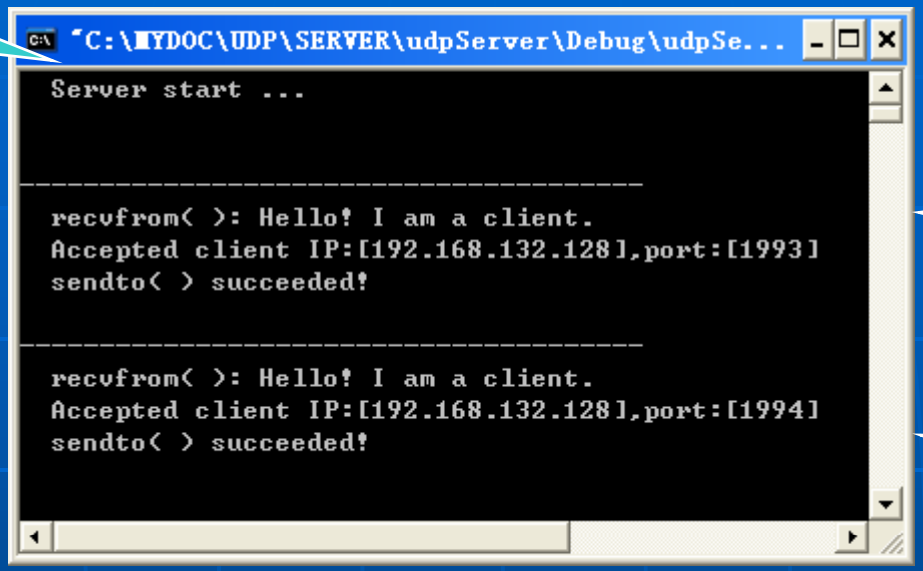
```
void main( )
{
	WSADATA wsaData;
	SOCKET sClient;
	//服务器地址长度
	int		iLen;
	//接收数据的缓冲
	int		iSend;
	int		iRecv;
	//要发送给服务器的信息
	char send_buf[ ]="Hello! I am a client.";
	//接收数据的缓冲区
	char recv_buf[DATA_BUFFER];
	//服务器端地址
	struct sockaddr_in ser;
	//初始化Socket
	if(WSAStartup(MAKEWORD(2,2),&wsaData)!=0)
	{
		printf("Failed to load Winsock.\n");
		return;
	}
```

```
//建立客户端数据报套接口
sClient=socket(AF_INET,SOCK_DGRAM,0);
if(sClient==INVALID_SOCKET)
{
   printf("socket( )Failed:%d\n",WSAGetLastError( ));
   return;
}
//建立服务器端地址
ser.sin_family=AF_INET;
ser.sin_port=htons(Ser_PORT);
ser.sin_addr.s_addr=inet_addr("192.168.132.128");
iLen=sizeof(ser);
//向服务器发送数据
iSend=sendto(sClient,send_buf,sizeof(send_buf),0,
           (struct sockaddr*)&ser,iLen);
if(iSend==SOCKET_ERROR)
{
   printf("sendto( )Failed:%d\n",WSAGetLastError( ));
   return;
}
else
   if(iSend==0)
       return;
else
   printf("sendto( ) succeeded.\n");
```

```
//接收数据的缓冲区初始化
memset(recv_buf,'\0',sizeof(recv_buf));
//从服务器接收数据
iRecv=recvfrom(sClient,recv_buf,sizeof(recv_buf),0, (struct sockaddr*)&ser,&iLen);
if(iRecv==SOCKET_ERROR)
{
   printf("recvfrom( )Failed.:%d\n", WSAGetLastError( ));
   return;
}
else   if (iRecv==0)
       return;
else
   {
       //显示从服务器收到的信息
       printf("recvfrom( ):%s\n",recv_buf);
       printf("------------------------------\n");
   }

closesocket(sClient);
WSACleanup( );
}
```

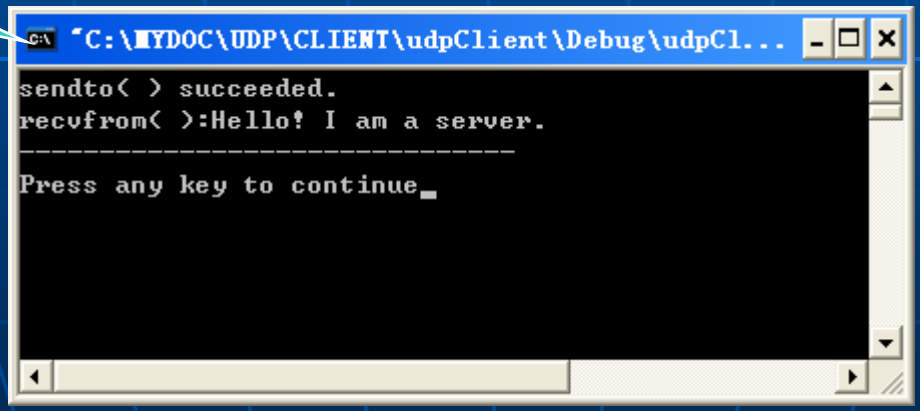
服务器运行结果



第1次客户端连接

第2次客户端连接

客户端运行结果



13.6 UDP广播编程

- 广播报文的发送
- 广播报文的接收

套接口选项

- 使用setsockopt 和getsockopt 函数可以设置或者读取套接口的选项值，这两个函数的定义如下：
- **int setsockopt (SOCKET s, int level, int optname, const char FAR *optval, int optlen);**
- **int getsockopt (SOCKET s, int level , int optname , char FAR * optval , int FAR * optlen ）；**
- 参数说明：
  - 参数s 是套接口描述字
  - **level [IN/lN]是套接口选项的级别**
  - **optname [IN/lN]是用户指定的选项名称**
  - **optval [IN/OUT]是一个指向变量的指针，它的大小由optlen [IN/OUT]指定，通过optval 可以设置或者读取指定选项的值**
  - 如果没有错误发生，函数返回0 ；否则返回SOCKET_ERROR

套接口选项

Winsock 的套接口选项主要有两种类型：

- **布尔型，用于启用或禁止套接口的某个属性;**
- **选项参数，整型或者结构体，设置特定的值。**
  - **为了启用某个布尔型选项，optval 应指向某个非零整数，若要禁用，则指向一个零值整数，同时optlen 参数值应该等于布尔型大小，即sizeof (BOOL) ;**
  - **对于其他选项类型，optval 应指向整型或者结构体，optlen 值为整型或者结构体的大小。**
- **Winsock 支持的全部选项级别有**

  SOL_SOCKET、 IPPROTO_lP 、**IPPROTO_IPV6 、IPPROTO_RM 、IPPROTO_TCP 、IPPROTO_UDP 、NSPROTO_lPX 、SOL_APPLETALK 、SOL_IRLMP 。**

SOL_SOCKET级别的部分选项

(1) SO_BROADCAST , BOTH ， 布尔型（BOOL ）。用于允许 / 禁止发送广播报文。BOTH：该选项既可以设置也可以读取。

(2) SO _ RCVTIMEO , BOTH , struct timeval 结构型，用于设置数据接收超时值。

- 该选项用于在一个阻塞套接口为接收函数设定一个超时值。
- 当调用接收函数时，如果在SO _ RCVTIMEO指定的时间内没有数据到来，那么函数调用也会结束并且返回错误10060 (WSAETIMEDOUT ）。

(3) SO_ SNDTIMEO , BOTH , struct timeval 结构型，用于设置数据发送超时值。

- 该选项用于在一个阻塞套接口为发送函数设定一个超时值。
- 当调用发送函数时，如果在SO_ SNDTIMEO指定的时间内数据还未发送成功，那么函数调用也会结束并且返回错误10060 (WSAETIMEDOUT ）。

IPPROTO_IP级别的选项

(1) IP_HDRINCL , BOTH ， 布尔型，仅适用于原始套接口（SOCK_RAW ）

- 如果应用程序希望能接收IP 层及IP 层以上的所有数据或者自行组装包含IP 层在内的报文，那么可以设置该选项为TRUE

(2) IP_ADD_MEMBERSHIP, SET , struct ip_mreq结构型，用于加入多播组

- 该选项用于将指定网络接口上的套接口加入IP 多播组，此套接口必须是AF_INET 地址族并且类型为SOCK_DGRAM

IPPROTO_IP级别的选项

(3) IP_DROP_MEMBERSHIP, SET, struct ip_mreq 结构型，用于退出多播组。该选项的使用与IP_ADD_MEMBERSHIP 类似。

(4）IP_MULTICAST_IF, BOTH , DWORD 类型，设置/读取多播的本地接口。

- 该 选项用于设置或读取本地接口。
- 在设置了本地的多播外出接口后，本地机器以后发出的任何多播数据都会经由它传送出去，该选项适用于多宿主主机。

IPPROTO_IP级别的选项

(5) IP_MULTICAST_LOOP, BOTH，布尔型，用于启用或者禁止多播报文环回。

- 在默认的情况下，当发送IP 多播数据时，如果发送套接口本身也属于该多播组，那么数据会原封不动地返回一份至套接口——环回（Ioopback）。
- 若将该选项设为FALSE，发出的任何数据都不会投递至套接字的进入数据队列中。

(6) IP_MULTICAST_TTL, BOTH, DWORD 类型，设置／读取套接口上IP 多播的TTL 值。

- 在默认情况下，多播数据报采用的TTL 值为1。
- 该 选项可用于读取或者设定该值。
- 多播TTL 值的大小影响到多播数据的传播范围，只有在有效范围内的组成员才会收到数据。

广播报文的发送

在默认情况下，UDP 套接口是无法发送广播报文的。

- 通过下面的代码段，可以确认广播选项是默认关闭的

```
bool  bBroadcast ;
int optlen = sizeof(bBroadcast) ;
if ( getsockopt (sock, SOL_SOCKET, SO_BROADCAST,
(char *) & bBroadcast , &optlen) == SOCKET_ERROR )  {
    closesocket (sock);
     WSACleanup( ) ;
     return -1;
 }
 if ( bBroadcast )
     printf("Broadcast enabled default !\n");
else
    printf("Broadcast disenabled default !\n");
```

- 程序输出"Broadcast disenabled default !"

广播报文的发送

- 套接口启用 / 禁止广播是通过

  SOL_SOCKET → SO_BROADCAST 选项来完成的

- 要发送广播报文，必须先启用**SO_BROADCAST**选项：

```
bool  bBroadcast = true;
int  optlen = sizeof( bBroadcast ) ;
setsockopt(sock , SOL_SOCKET ,  SO_BROADCAST, (char *)
&bBroadcast ,  optlen )
```

广播报文的发送

成功地启用SO_BROADCAST后，就可以发送广播报文：

```
struct  sockaddr_in addr;
memset( &addr,  0 , seizeof(addr) ) ;
addr.sin_addr. s_addr = INADDR_BROADCAST;  // 广播地址
addr.sin_family = AF_INET;
addr.sin_port = htons(9999);  // 广播端口


const  char *msg = "Hello! Broadcast Test!";

int  len = strlen( msg );

sendto(sock, msg, len, 0, (struct sockaddr *) &addr, sizeof(addr))
```

广播报文的接收

- 接收广播报文不需要任何额外的设置，创建UDP套接口后，绑定到广播端口即可
- 以上面的发送程序为例，接收方程序段如下：

```
SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);
 struct  sockaddr_in  addr;
 memset(&addr, 0, sizeof(addr));
 addr.sin_family = AF_INET;
 addr.sin_port = htons(9999);
 addr.sin_addr.s_addr = INADDR_ANY;
 bind( sock, (struct sockaddr *) &addr, sizeof(addr) );
```

广播报文的接收

```
int  ret;
char  buf [5000];
memset( buf , 0 , sizeof( buf ) ) ;
while( 1 )
{
        ret = recvfrom(sock, buf, sizeof(buf), 0, NULL, NULL);
        if (ret == SOCKET_ERROR) {
            printf("recvfrorn: %d\n", WSAGetLastError( ) ) ;
            break;
        }
    else
            printf("recvd %d bytes  :  %s \n",  ret ,  buf   );
 }
```

UDP广播服务器源代码

```
/* ------------ 服务器广播示例代码 ---------------------------
--- 向局域网内的所用主机广播信息，包括“点名”命令
--- 如果广播了“点名”命令，则接受客户主机回传的学号和姓名-----*/

#include "Winsock2.h"
#include "stdio.h"
#include "string.h"
#pragma comment(lib,"wsock32.lib")

#define RECV_PORT 3000    // 接收端口
#define SEND_PORT 3001    // 广播端口

SOCKET sock;              // 用于广播的套接字
SOCKET ServerSock;        // 用于接受的套接字
sockaddr_in ServerAddr;   // 服务器接收地址
sockaddr_in broadAddr;    // 广播地址
```

```
void main()
{
    // ----------- 初始化套接字 --------------
    WSADATA WSAData;
    if(WSAStartup(MAKEWORD(2,2),&WSAData)!=0)
    {
        printf("socket init fail!\n");
        return ;
    }

    // ----------- 创建用于广播的套接字 --------
    sock=socket(AF_INET,SOCK_DGRAM,0);
    if(sock==SOCKET_ERROR)
    {
        printf("socket create fail!\n");
        WSACleanup();
        return ;
    }

    // ----------- 创建用于接收的套接字 --------
    ServerSock=socket(AF_INET,SOCK_DGRAM,0);
    if(ServerSock==SOCKET_ERROR)
    {
        printf("socket create fail!\n");
        WSACleanup();
        return ;
    }
```

```
// ---------- 设置服务器地址 ---------------------------------------
memset(&ServerAddr,0,sizeof(ServerAddr));
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_addr.s_addr = inet_addr("192.168.132.128") ;  // INADDR_ANY;
ServerAddr.sin_port = htons(RECV_PORT);

// ---------- 设置广播地址 ---------------------------------------
memset(&broadAddr,0,sizeof(broadAddr));
broadAddr.sin_family = AF_INET;
broadAddr.sin_addr.s_addr =  inet_addr("255.255.255.255"); //INADDR_BROADCAST ;
printf("Broadcase addr: %s \n", inet_ntoa(broadAddr.sin_addr) );
broadAddr.sin_port = htons(SEND_PORT);

// ---------- 设置允许广播 -----------------------
bool bBroadcast= true;
int n = setsockopt(sock,SOL_SOCKET,SO_BROADCAST,(const char*)&bBroadcast,sizeof(bBroadcast));

// ---------- 检测是否可以广播 -------------------------
int opt = sizeof(bBroadcast);
getsockopt(sock,SOL_SOCKET,SO_BROADCAST,(char*)&bBroadcast,&opt);
```

```
if( bBroadcast == true )
{
    printf("Test Broadcast: ok \n");
}
else
{
    printf("Test Broadcast: can not Broadcast \n");
    return;
}

// ---------- 开始广播 ---------------------------------------
char buf[1024] = "/name:Please send your name\0";
n = sendto(sock,buf,strlen(buf)+1,0,(const sockaddr *)&broadAddr,sizeof(broadAddr));

if(n == SOCKET_ERROR)
{
    printf("  BROADCAST data error!  \n");
    printf("  The error code: %d \n", WSAGetLastError() );
}
else
{
    printf("Send BROADCAST data: \"%s\" ok! \n\n", buf);
}
```

```
// ------------ 下面准备接收客户端的回传信息 ---------------------------------

// ---------- 服务器绑定地址 --------------------------
if( bind(ServerSock,(struct sockaddr *)&ServerAddr, sizeof(ServerAddr)) == SOCKET_ERROR)
{
    printf("  Binding error! \n");
    printf("  The error code: %d \n", WSAGetLastError() );
    closesocket(ServerSock);
    getchar();
    return;
}
else
{
    printf("Binding ok! start recieving data from clients ... \n");
}
```

```
// ---------- 接收客户端发回的数据 ------------------
while(1)
{
    sockaddr_in ClientAddr;
    memset(&buf,'\0',sizeof(buf));
    memset(&ClientAddr,0,sizeof(ClientAddr));
    n = sizeof(ClientAddr);
    n = recvfrom(ServerSock,buf,sizeof(buf),0,(struct sockaddr *)&ClientAddr, &n);

    if( n != SOCKET_ERROR )
    {
        printf("\n  接收远程主机 %s 传回的数据: %s \n",
               inet_ntoa(ClientAddr.sin_addr), buf);
    }
} // end of while

closesocket(sock);
closesocket(ServerSock);
WSACleanup();
}
```

UDP广播客户端源代码

```
/* -------- UDP客户端 ------------------------------
--- 接受UDP服务器的广播并显示广播信息
--- 如果是“点名”命令，则回传自己的学号和姓名 ----*/

#include "Winsock2.h"
#include "stdio.h"
#include "string.h"
#pragma comment(lib,"wsock32.lib")

#define SEND_PORT 3000     // 发送端口
#define RECV_PORT 3001     // 接收端口

SOCKET sock;
sockaddr_in ServerAddr;    // 服务器地址
sockaddr_in ClientAddr;    // 客户端(本地)地址
```

```
void main()
{
    int n;
    // ----------- 初始化套接字 --------------
    WSADATA WSAData;
    if(WSAStartup(MAKEWORD(2,2),&WSAData)!=0)
    {
        printf("socket init fail!\n");
        return ;
    }

    // ----------- 创建用于收发的套接字 --------
    sock=socket(AF_INET,SOCK_DGRAM,0);
    if(sock==SOCKET_ERROR)
    {
        printf("socket create fail!\n");
        WSACleanup();
        return ;
    }
```

```
// ---------- 设置客户端（本地）地址 ------------------
memset(&ClientAddr,0,sizeof(ClientAddr));
ClientAddr.sin_family = AF_INET;
ClientAddr.sin_addr.s_addr = inet_addr("192.168.132.128") ; // INADDR_ANY ;
ClientAddr.sin_port = htons(RECV_PORT);

// ---------- 绑定套接字到本地地址 --------------------
n = bind(sock , (struct sockaddr *)&ClientAddr, sizeof(ClientAddr));
if( n == SOCKET_ERROR)
{
    printf(" Bind error! \n");
    return;
}
else
{
    printf(" Bind Address ok!  \n");
}

// ------------- 在指定端口侦听，并接受服务器广播的信息 --------------
char buf[1024] = "\0";
int len ;
printf(" Bengin receiving datas... \n");
```

```
while(1)  // 无限循环侦听
{
    memset(&buf,'\0',sizeof(buf));    // 清空接收缓冲区
    memset(&ServerAddr,0,sizeof(ServerAddr)); // 清空地址空间
    len = sizeof(ServerAddr);

    // 接收服务器广播信息
    n = recvfrom(sock,buf,sizeof(buf),0,(struct sockaddr *)&ServerAddr,(int *)&len);

    if(n == SOCKET_ERROR)
    {
        printf("\n  Get BROADCAST Data error!  \n");
        printf("\n  The error code: %d \n", WSAGetLastError() );
    }
```

```
else
{
    // ------- 收到广播信息，则显示 ------------------------
    printf("\n Get BROADCAST Data: \"%s\" from %s \n" , buf , inet_ntoa(ServerAddr.sin_addr) );

    if( strcmp( buf , "/name") >= 0 )  // 如果是"点名"命令
    {
    // ------- 设置服务器地址，准备回传 ---------------------
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_addr.s_addr = inet_addr("192.168.132.128");
    ServerAddr.sin_port = htons(SEND_PORT);

    // ------- 回传自己的学号和姓名 ------------------------
    memset(&buf,'\0',sizeof(buf));
    strcpy(buf,"10001 张三");
    n = sendto(sock,buf,strlen(buf)+1,0,(struct sockaddr *)&ServerAddr, sizeof(ServerAddr)) ;
```

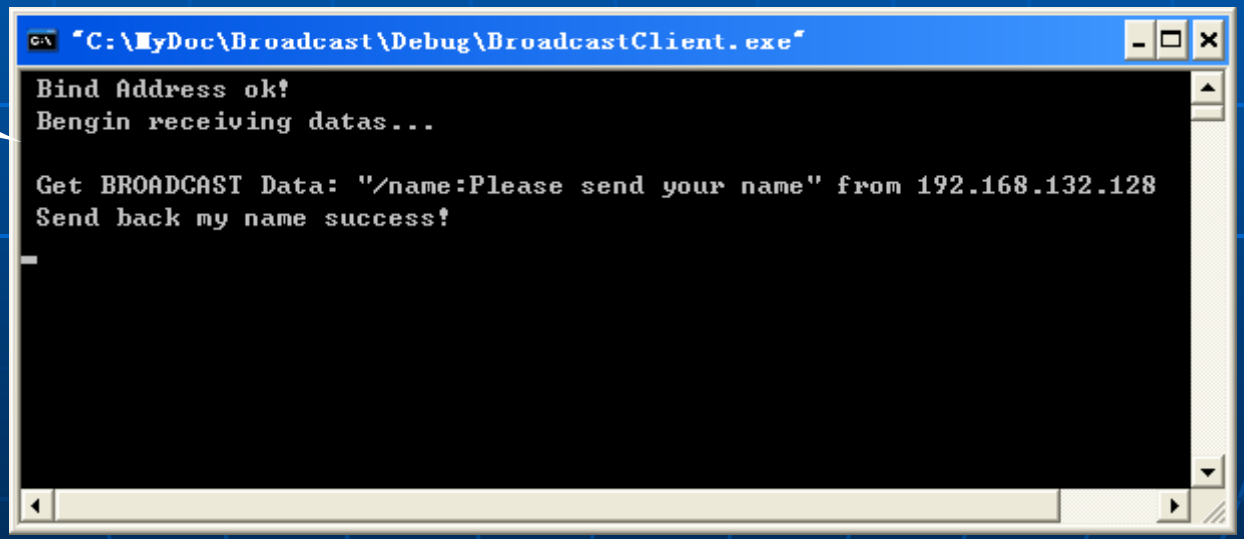
```
        if( n == SOCKET_ERROR)
        {
            printf(" Send back my name false! \n");
        }
        else if( n==0)
        {
            printf(" Send back my name false 0 \n!");
        }
        else if (n != SOCKET_ERROR)
        {
            printf(" Send back my name success! \n");
        }
        }
    }

    Sleep(1000);
}

getchar();
closesocket(sock);
WSACleanup();
}
```

服务器运行结果

客户端运行结果



注：本实例程序需要客户端先执行，等候接收广播信息

谢谢！