第 5 章　UDP 通信程序

尽管 TCP 能实现网络实体间的通信，但通信双方是不对等的，必须有一方作为服务器，而另一方（或多方）作为客户端。而 UDP 通信则提供了两种套接字编程模型，即 C/S 模型和 P2P 模型，其中 P2P 模型各方都是对等的。UDP 套接字编程还可方便地实现广播程序或多播程序，这些特点使 UDP 在很多方面弥补了 TCP 的不足。

5.1　UDP 通信程序的原理

UDP 是基于不可靠传输的数据报传输协议，UDP 传输的是无连接数据报，UDP 数据报几乎是原封不动地封装了 IP 数据包，只是在 IP 数据包的包头增加了 UDP 端口号。UDP 服务的特点是无连接、不可靠。无连接的特点决定了数据报的传输非常灵活，具有资源消耗小、处理速度快的优点；而不可靠意味着在网络质量不佳的情况下，会发生数据包丢失的情况，因此上层应用程序在设计时需要考虑网络应用程序的运行环境，数据在传输过程中的丢失、乱序或重复对应用程序带来的负面影响。总的来看，UDP 适用于以下场合：

（1）视频、音频数据的实时传输。

UDP 适合用于视频、音频这类对实时性要求比较高的数据传输。传输的内容通常被切分成独立的数据报，其类型多为编码后的媒体信息。相对于 TCP，UDP 减少了确认、同步等操作，减少了网络开销。

（2）广播与多播的传输应用。

TCP 只能用于一对一的数据通信；而 UDP 不但支持一对一的通信，还支持一对多的通信，可以使用广播方式向某一 IP 子网内的所有用户发送广播数据，或使用多播方式向一组用户传输数据，这类应用包括局域网聊天室或者以广播形式实现的局域网扫描器等。

（3）简单、高效需求高于可靠需求的传输应用。

尽管 UDP 传输不可靠，但其高效的传输特点使其在一些传输应用中受到欢迎。例如，聊天软件通常使用 UDP 传输消息和文件，日志服务器通常设计为基于 UDP 接收日志。这些应用不希望在每次传输短小数据时也要付出昂贵的 TCP 连接的建立与维护代价，而且即使丢失一两个数据包，也不会对接收结果产生太大影响。也可以通过应用层程序检测是否丢包，并对丢失的数据包进行重传。

（4）需要穿透防火墙或代理服务器的通信。

由于很多用户通过企业内部网上网，不同内部网中主机之间的通信需要穿过防火墙、代理服务器、NAT 路由器等设备，此时，不同主机之间能彼此建立 TCP 连接的概率很小。而 UDP 数据包则能穿透大部分的 NAT 路由器等设备（俗称 UDP 打洞），因此 QQ 选择 UDP 作为客户端之间主要的通信协议。

综上所述，UDP 具有多对多通信、不可靠服务、缺乏流控制等传输特点。同时，UDP 采用报文模式，在发送数据时，发送方需要准确指明要发送的字节数。

5.1.1 UDP 的通信模式

使用 UDP 套接字通信有两种通信模式可供选择：一种是客户端/服务器(C/S)模式，另一种是对等(P2P)模式。

1. 客户端/服务器模式

在 UDP 客户端/服务器模式中，只有服务器端绑定了地址，客户端没绑定地址，服务器端在通信开始前并不知道客户端的任何地址信息，因而无法发起通信，双方的通信必须由客户端先向服务器端发送消息来发起，此后服务器端和客户端才能相互发送消息。

使用客户端/服务器模式的 UDP 通信程序流程如图 5-1 所示。服务器端必须先启动，而通信由客户端先发送数据来发起。

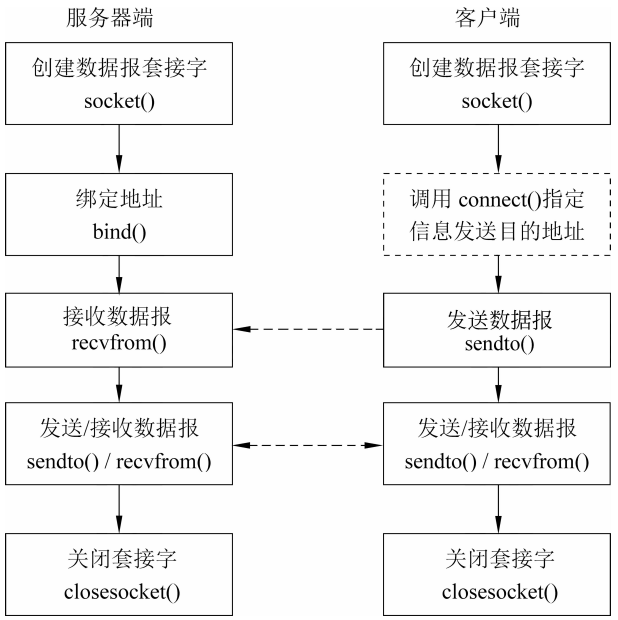

图 5-1 客户端/服务器模式的 UDP 通信程序流程

UDP 服务器端的通信流程如下：

(1) 加载套接字库 (WSAStartup())。

(2) 创建套接字(socket())，将第 2 个参数设置为 SOCK_DGRAM。

(3) 绑定套接字到本机地址和通信端口(bind())。

(4) 等待来自客户端的数据。

(5) 进行数据传输(sendto()和 recvfrom())。

(6) 关闭套接字 (closesocket())，并清除加载的套接字库，释放资源(WSACleanup())。

UDP客户端的通信流程如下：

(1) 加载套接字库（WSAStartup()）。

(2) 创建套接字(socket())，将第2个参数设置为SOCK_DGRAM。

(3) 使用connect()函数为套接字指定通信对端的IP地址和端口号。这一步是可选的，使用connect()函数后，就既可以使用send()和recv()函数传输数据，也可以使用sendto()和recvfrom()函数传输数据；而如果省略这一步，随后的数据传输就只能使用sendto()和recvfrom()函数了，因为需要使用这两个函数的后两个参数去指定对端的地址。

(4) 向服务器端发送数据(sendto())。

(5) 进行数据传输(sendto()和recvfrom())。

(6) 关闭套接字（closesocket()），并清除加载的套接字库，释放资源(WSACleanup())。

注意：在UDP程序中，不绑定IP地址和端口号的一方必须首先向绑定IP地址和端口号的一方发送数据。

2. 对等模式

在对等模式中，通信双方并无客户端和服务器端之分，通信双方地位完全对等，每一方既作为服务器端又作为客户端，谁都可以首先向对方发送数据，只要事先知道对方的IP地址即可。在这种模式下，还可以实现一对多或多对多通信。

由于通信各方都要能主动向其他方发送信息，因此，每个通信方都必须先用bind()函数绑定本机地址，在发送数据之前需要知道数据接收方的IP地址及所用端口号。

基于对等模式的UDP通信程序流程如图5-2所示，可见，通信双方的程序流程完全一样，程序代码也完全一样，因此只要编写一个程序，然后再运行该程序的多个实例，即可相互通信。

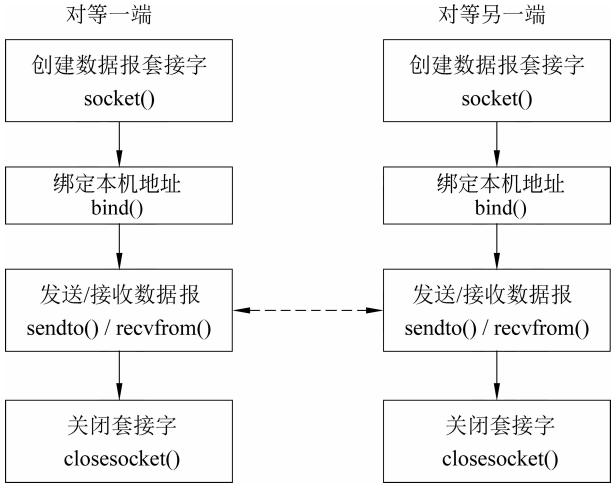

图5-2 对等模式的UDP通信程序流程

需要说明的是，在对等模式中，发送数据前同样可以先调用connect()函数指定对端

的地址，然后就可使用 send()函数和 recv()函数传输数据，这一方法在图 5-2 中没有给出。

5.1.2 UDP 的数据收发函数

UDP 的发送、接收数据函数分别是 sendto()和 recvfrom()，它们都有 6 个参数；而 TCP 的发送、接收数据函数分别是 send()和 recv()，它们都只有 4 个参数。sendto()/send()和 recv()/recvfrom()这两对函数的前 4 个参数的含义是相同的。

因为 UDP 是无连接通信，必须知道对方的地址才能收发数据，所以 sendto()和 recvfrom()多了两个参数，分别用来设置对方的套接字地址和套接字地址变量的长度。另外，recvfrom()的第 6 个参数是一个地址，带有 & 符，而 sendto()的第 6 个参数是一个值。

1. sendto()函数

sendto()函数通常用于数据报套接字发送数据，但也可用于流式套接字。其函数原型如下：

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

该函数有 6 个参数，各个参数的含义如下：

- s：一个套接字。
- buf：保存待发送数据的缓冲区。
- len：缓冲区中待发送数据的长度。
- flags：调用操作方式，通常取值为 0。
- to：(可选)指针，指向目的套接字的地址。
- tolen：to 所指地址的长度。

若无错误发生，该函数返回发送数据的字节数；否则返回 SOCKET_ERROR 错误值。

2. recvfrom()函数

recvfrom()函数通常用于数据报套接字接收数据，但也可用于流式套接字。其函数原型如下：

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

该函数有 6 个参数，各个参数的含义如下：

- s：指定接收数据的套接字。
- buf：接收数据缓冲区。
- len：缓冲区长度。

- flags：调用操作方式。
- from：(可选)指针，指向装有源地址的缓冲区。
- fromlen：(可选)指针，指向 from 缓冲区长度值。

若无错误发生，该函数返回收到的字节数；如果连接已中止，返回 0；若发生错误，返回 SOCKET_ERROR。

提示：recvfrom()函数不仅能从套接字接收数据，还能捕获数据发送源地址。因为该函数的 from 参数保存了发送源地址(网络字节顺序形式)。

5.2 控制台版本的 UDP 通信程序实例

本节制作一个控制台版本的 UDP 通信程序，该程序分为服务器端与客户端，如图 5-3 所示。其中，左图为服务器端，右图为客户端。其功能与 2.2 节中的 TCP 通信程序基本相同，但区别是本程序的客户端必须先发送消息给服务器端。

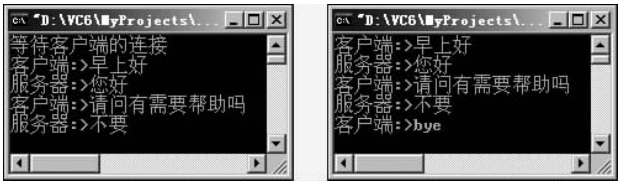

图 5-3 控制台版本的 UDP 通信程序

5.2.1 服务器端程序的编制

UDP 通信程序

服务器端程序的编制步骤如下：

(1) 新建工程，选择 Win32 Console Application，输入工程名(如 UDPServer)，单击“下一步”按钮，在 Win32 Application 对话框中选择“一个简单的 Win32 程序”单选按钮。

(2) 在 UDPServer.cpp 文件中输入如下代码：

```
#include "iostream.h"
#include "winsock2.h"
#pragma comment(lib, "ws2_32.lib ")
int main(){
    WSADATA wsaData;
    if(WSAStartup(MAKEWORD(2,2), &wsaData)) {
        cout<<"Winsock can not be init!"<<endl;
        WSACleanup();
        return 0;
    }
    SOCKET sockSer;                          //创建服务器套接字
    sockSer=socket(AF_INET, SOCK_DGRAM, 0);
```

```
    sockaddr_in addrSer,addrCli;          //创建服务器地址和客户端地址结构
    addrSer.sin_family=AF_INET;
    addrSer.sin_port=htons(5566);
    //addrSer.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
    addrSer.sin_addr.s_addr=INADDR_ANY; //设置服务器地址为任意地址
    bind(sockSer,(SOCKADDR*)&addrSer,sizeof(addrSer));
    cout<<"等待客户端的连接"<<endl;
    char sendbuf[256]="\0";               //该字符数组无论赋0值还是不赋值都可以
    char recvbuf[256]="\0";
    int len=sizeof(SOCKADDR);
    while(1){
        recvfrom(sockSer,recvbuf,256,0,(SOCKADDR *)&addrCli,&len);
        cout<<"客户端:>" <<recvbuf<<endl;
        cout<<"服务器:>";
        cin>>sendbuf;
        if(strcmp(sendbuf,"bye")==0)
            break;
        sendto(sockSer,sendbuf,strlen(sendbuf)+1,0,(SOCKADDR *)&addrCli,len);
    }
    closesocket(sockSer);
    WSACleanup();
    return 0;
}
```

最后,编译并运行以上代码。

UDP 程序和 TCP 程序有以下两个不同之处:

(1) socket()函数中第 2 个参数取值为 SOCK_DGRAM。

(2) UDP 的服务器端和客户端都只要创建一个套接字,而 TCP 程序服务器端必须创建两个套接字,因为 accept()函数的参数是一个套接字,返回值是另一个套接字。

UDP 程序和 TCP 程序的相同点是:服务器端都需要创建两个套接字地址(例如 SOCKADDR_IN addrSer,addrCli),而客户端都只需要一个套接字地址(sockaddr_in addrSer)。

5.2.2 客户端程序的编制

客户端程序的编制步骤如下:

(1) 新建工程,选择 Win32 Console Application,输入工程名(如 UDPCli),单击“下一步”按钮,在 Win32 Application 对话框中选中“一个简单的 Win32 程序”单选按钮。

(2) 在 UDPCli.cpp 文件中输入如下代码:

```
#include "iostream.h"
#include "winsock2.h"
#pragma comment(lib, "ws2_32.lib ")
int main(){
```

```
    WSADATA wsaData;
    if(WSAStartup(MAKEWORD(2,2), &wsaData)) {
        cout<<"Winsock can not be init!"<<endl;
        WSACleanup();
        return 0;
    }
    SOCKET sockCli;
    sockCli=socket(AF_INET,SOCK_DGRAM,0);          //创建数据报套接字
    SOCKADDR_IN addrSer;                           //存放服务器端地址
    addrSer.sin_family=AF_INET;
    addrSer.sin_port=htons(5566);
    addrSer.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
    char sendbuf[256],recvbuf[256];
    int len=sizeof(SOCKADDR);
    while(1){
        cout<<"客户端:>";
        cin>>sendbuf;
        if(strcmp(sendbuf,"bye")==0)
            break;
        sendto(sockCli,sendbuf,strlen(sendbuf)+1,0,(SOCKADDR*)&addrSer,len);
        recvfrom(sockCli,recvbuf,256,0,(SOCKADDR*)&addrSer,&len);
        cout<<"服务器:>"<<recvbuf<<endl;
    }
    closesocket(sockCli);
    return 0;
}
```

最后,编译并运行以上代码。

思考:将客户端程序改为使用 connect()函数指定通信对端地址,并使用 send()和 recv()函数传输数据。

5.3 异步对等 UDP 通信程序实例

本节将采用异步通信模式和 UDP 制作一个对等通信程序。该通信程序中的各方都是完全相同的,因此只需编写一个程序,然后运行该程序若干次,并为各个程序副本绑定不同的端口号或 IP 地址,各个程序副本之间就能相互发送和接收消息。该程序的运行效果如图 5-4 所示,其中启动了 3 个程序副本。

该程序分为以下 4 个功能模块:

(1) 在窗口初始化消息 WM_INITDIALOG 中,初始化 WinSock 协议栈,并设置文本框默认显示的内容。

(2) 在“启动”按钮的消息中,创建数据报套接字,并将套接字设置为异步模式,然后将套接字绑定到文本框中设置的本机地址。

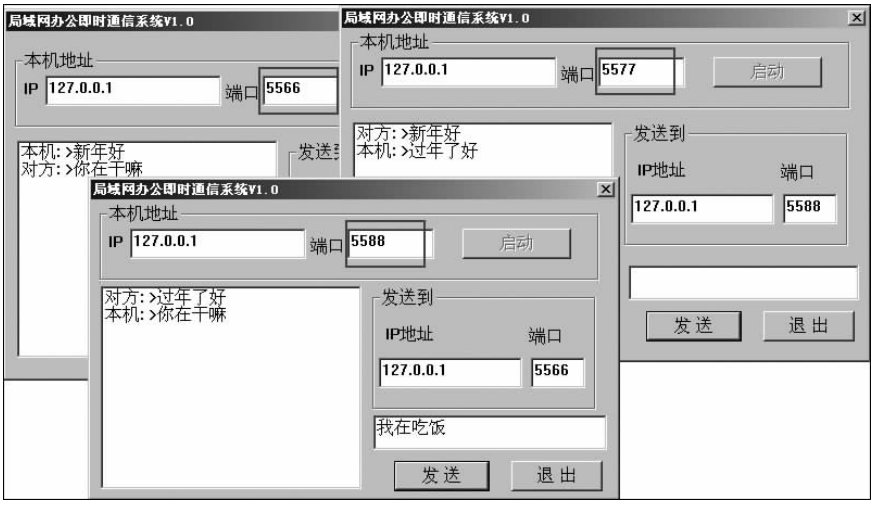

图 5-4　异步对等 UDP 通信程序

(3) 在“发送”按钮的消息中,获取“发送到”选项组中“IP 地址”和“端口”两个文本框中的对端地址,用 sendto()函数将数据内容发送出去,并将发送的数据内容显示在左侧的列表框中。

(4) 在自定义套接字消息 WM_SOCKET 的 FD_READ 事件中,用 recvfrom()函数接收数据,并将接收的数据内容显示在左侧的列表框中。

该程序的具体编制步骤如下:

(1) 新建工程,选择 Win32 Application,输入工程名(如 UDPCom),然后选择“一个典型的 Hello World!程序”。

(2) 在工作空间窗口左侧选择 FileView 选项卡,找到对应的源文件(如 UDPCom.cpp),将 WinMain()函数中 DialogBox()函数行(148 行)和 return 0;行保留,将其他代码全部删除,再将 DialogBox()函数中第 3 个参数 HWnd 改为 NULL。

(3) 切换到 ResourceView 选项卡,找到 Dialog 下的 IDD_ABOUTBOX,将对话框的界面改为如图 5-5 所示,并设置各个控件的 ID 值。

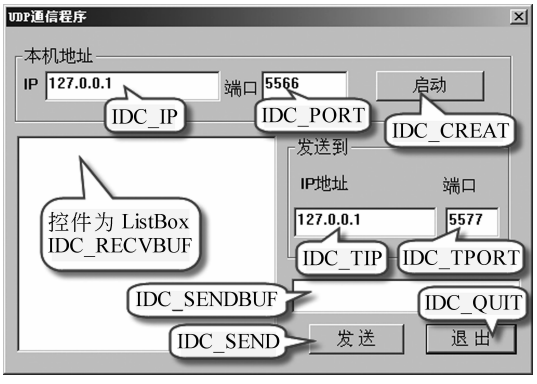

图 5-5　异步对等 UDP 通信程序界面及各控件 ID 值

(4) 打开 UDPCom.cpp 文件,编写如下代码:

```
#include "stdafx.h"
#include "resource.h"
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib ")
#define WM_SOCKET WM_USER+0x10              //自定义套接字消息
HINSTANCE hInst;
LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);
SOCKET Client, ServerSocket;                //创建两个套接字
SOCKADDR_IN addrSer,addrCli;                //创建两个套接字地址结构
int socklen;
int iIndex=0;
int len=sizeof(addrSer);
char sendbuf[256], recvbuf[256];            //发送和接收缓冲区
char clibuf[999]="对方:>", serbuf[999]="本机:>";
int APIENTRY WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow){
    DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, NULL, (DLGPROC)About);
    return 0;
}
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam){
    char port[5], ip[16];                   //本机端口和 IP 地址
    char tport[5], tip[16];
    int iErrorCode;
    switch(message) {
        case WM_INITDIALOG:                 //对话框启动时
            SetDlgItemText(hDlg,IDC_IP,"127.0.0.1");    //设置本机 IP 地址
            SetDlgItemText(hDlg,IDC_PORT,"5566");
            SetDlgItemText(hDlg,IDC_TIP,"127.0.0.1");   //设置目的主机 IP 地址
            SetDlgItemText(hDlg,IDC_TPORT,"5577");
            EnableWindow(GetDlgItem(hDlg,IDC_SEND),FALSE);
            WSADATA wsaData;
            if(WSAStartup(MAKEWORD(2,2), &wsaData)) {    //初始化协议栈
                MessageBox(hDlg,"WinSock 初始化失败!","警告",0);
                WSACleanup();
        }
            return TRUE;
        case WM_COMMAND: {                          //处理单击按钮的消息
            switch(LOWORD(wParam)){
                case IDC_QUIT:                      //单击了"退出"按钮
                    closesocket(ServerSocket);      //关闭套接字
                    shutdown(ServerSocket,2);       //关闭套接字的另一种方法(可选)
                    WSACleanup();
```

```
            EndDialog(hDlg, LOWORD(wParam)); //关闭对话框
            return TRUE;
        case IDC_CREAT:                    //单击了"启动"按钮,创建套接字
            GetDlgItemText(hDlg,IDC_IP,ip,16);
            GetDlgItemText(hDlg,IDC_PORT,port,5);
            EnableWindow(GetDlgItem(hDlg,IDC_CREAT),FALSE);
            EnableWindow(GetDlgItem(hDlg,IDC_SEND),TRUE);
            ServerSocket=socket(AF_INET,SOCK_DGRAM,0);
            //设置套接字为异步模式
            iErrorCode=WSAAsyncSelect(ServerSocket,hDlg,WM_SOCKET,FD_
            READ | FD_CLOSE);
            if(iErrorCode==SOCKET_ERROR) {
                MessageBox(hDlg,"WSAAsyncSelect 设定失败!","失败!",0);
                return 0;
            }
            addrSer.sin_family=AF_INET;
            addrSer.sin_port=htons(atoi(port));
            addrSer.sin_addr.S_un.S_addr=inet_addr(ip);
            //绑定本机地址
                 if (bind (ServerSocket, (SOCKADDR * ) &addrSer, sizeof
                 (addrSer))==SOCKET_ERROR) {
                MessageBox(hDlg,"绑定地址失败!","失败!",0);
                return 0;
            }
            break;
        case IDC_SEND:                                  //发送数据
            GetDlgItemText(hDlg,IDC_TIP,tip,16);        //获取对方 IP 地址
            GetDlgItemText(hDlg,IDC_TPORT,tport,5);
            addrCli.sin_family=AF_INET;
            addrCli.sin_port=htons(atoi(tport));
            addrCli.sin_addr.S_un.S_addr=inet_addr(tip);
            GetDlgItemText(hDlg,IDC_SENDBUF,sendbuf,256);
                                                        //获得发送框信息
            //发送数据
            sendto(ServerSocket,sendbuf,strlen(sendbuf)+1,0,(SOCKADDR * )
            &addrCli,sizeof(addrCli));
            SetDlgItemText(hDlg,IDC_SENDBUF,"");        //清空发送框
            strcat(serbuf,sendbuf);
            //将已发送的消息显示在列表框中
            SendDlgItemMessage(hDlg,IDC_RECVBUF,LB_ADDSTRING,0,
            (LPARAM)serbuf);
            strcpy(serbuf, "本机：>");
            break;
    }
```