实 验 二 UDP 通信程序

2.1 实验目的

UDP 是一种无连接的数据报网络通信，它没有连接状态是种不可靠的网络通信，其程序流程简单容易实现，本实验实现简单数据传输，远程主机唤醒及主机在线探测功能。

2.2 UDP 协议介绍

UDP（User Datagram Protocol）协议是“用户数据报协议”，它是一种无连接的协议，当计算机利用 UDP 协议进行数据传输的时候，发送方只需设置目标 IP 地址和端口号就可以发送数据，连接操作在 UDP 通信中是无意义的。编写基于 UDP 协议的应用程序比较简单，发送方只需发出数据，接收方从网络上接收数据就可以了，无需考虑时序及数据确认，它不管数据包的顺序、错误或重发。UDP 数据包封装格式简单在信道中极易被伪造安全性较低，UDP 通信无需连接流程具有多播能力这是 TCP 协议做不到的，可通过广播方便的发现网络主机。

网络层 IP 协议的数据包在局域网上以广播方式出现，网卡接收到信道中所有的数据包，经过目标地址匹配后向应用程序转交属于自己的数据包，网络中规定了两个特殊的 IP 地址，IP 地址的网络字段和主机字段二进制位全为 1 称为有限广播地址即为 255.255.255.255，路由器产品不转发目标地址为有限广播的数据包，它只能出现在局域网内；具有明确网络字段值而主机字段二进制位全为 1 的地址称为直接广播，例如 192.168.0.255 就是一个直接广播地址，路由器转发目标地址为直接广播地址的数据包到目标子网。TCP/IP 协议栈中传输层只有 UDP 具有广播功能，网段内所有网络适配器提取广播包数据提交给主机。

2.3 实验内容

实验包含四个网络任务，聊天与传输数据体现了通信功能，远程开机和服务主机探测体现广播功能，四个项目演示 UDP 的典型应用。

2.3.1 使用 UDPClient 的网络聊天程序

编写 UDP 程序既可以使用 Socket 类，也可以使用有一定封装的 UdpClient 类，它隐藏了数据收发的细节问题。UdpClient 类建立远程主机的方式有两种，一是使用远程主机名和端口号创建实例；还有一种是创建 UdpClient 实例，然后调用 Connect 方法指定远程主机。下面是使用同步方式编写的使用 Udpclient 类聊天程序，界面如图2-1。

图 2-1 UDP 网络聊天

程序中窗体线程负责用户输入和响应，发送数据和接收数据分别使用了两个工作线程，接收数据线程采用 UdpClient 类的 Receive 方法以同步方式接收数据，循环结构能够保证多次接收；而发送数据线程采用 UdpClient 类的 Send 方法同步方式发送数据，线程只执行一次发送操作，多次发送则执行多次线程；工作线程通过消息的方式通知窗体更新显示。首先是变量定义部分：

```
private UdpClient receiveUdpClient;
private UdpClient sendUdpClient;
private const int port = 18001;
IPAddress ip;
IPAddress remoteIp;
//定义消息常数
public const int RECV_DATA = 0x500;
public const int SEND_DATA = 0x501;
public static IntPtr main_wnd_handle;//主窗体句柄
//动态链接库引入
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
int wParam, // first message parameter
int lParam // second message parameter
);
private static string sendMessage;
private static string receiveMessage;
private static IPEndPoint remote;
private static string RemoteIp;
```

窗体 Load 函数初始化变量和控件，并启动接收数据线程：

```
private void FormChat_Load(object sender, EventArgs e)
```

```
{
    //获取本机可用 IP 地址
    IPAddress[] ips = Dns.GetHostAddresses(Dns.GetHostName());
    ip = ips[ips.Length - 1];
    remoteIp = ip;
    textBoxRemoteIP.Text = remoteIp.ToString();
    textBoxSend.Text = " 你好！ ";
    //创建一个线程接收远程主机发来的信息
    Thread myThread = new Thread(ReceiveData);
    //将线程设为后台运行
    myThread.IsBackground = true;
    myThread.Start();
    textBoxSend.Focus();
    main_wnd_handle = this.Handle;
}
```

工作线程与窗体线程间通过共享的全局变量来传递信息，重载窗体消息处理函数处理自定义消息：

```
protected override void DefWndProc(ref Message m)
{//窗体消息处理重载
    switch (m.Msg)
    {
        case RECV_DATA:
            listBoxReceive.Items.Add(string.Format(" 来自 {0}：{1}", remote, receiveMessage));
            listBoxReceive.SelectedIndex = listBoxReceive.Items.Count - 1;
            listBoxReceive.ClearSelected();
            break;
            case SEND_DATA:
            listBoxStatus.Items.Add(string.Format(" 向 {0} 发送：{1}", RemoteIp, sendMessage));
            textBoxSend.Text = "";
            textBoxSend.Focus();
            break;
        default:
            base.DefWndProc(ref m);
            break;
    }
}
```

接收 UDP 数据线程代码，这里使用了循环结构，因此这个线程将一直运行而不结束：

```
private void ReceiveData()
{
    IPEndPoint local = new IPEndPoint(ip, port);
```

```
    receiveUdpClient = new UdpClient(local);
    remote = new IPEndPoint(IPAddress.Any, 0);
    while (true)
    {
      try
      {
        //关闭 udpClient 时此句会产生异常
        byte[] receiveBytes = receiveUdpClient.Receive(ref remote);
        receiveMessage = Encoding.Unicode.GetString(
          receiveBytes, 0, receiveBytes.Length);
        SendMessage(main_wnd_handle, RECV_DATA, 100, 100);
      }
      catch
      {
        break;
      }
    }
}
```

发送 UDP 数据线程，负责发一次数据：

```
private void SendMessage()
{
    sendUdpClient = new UdpClient(0);
    byte[] bytes = System.Text.Encoding.Unicode.GetBytes(sendMessage);
    iep = new IPEndPoint(remoteIp, port);
    try
    {
      sendUdpClient.Send(bytes, bytes.Length, RemoteIp,port);
      SendMessage(main_wnd_handle, SEND_DATA, 100, 100);
    }
    catch (Exception ex)
    {
    }
}
```

启动发送数据线程代码：

```
private void buttonSend_Click(object sender, EventArgs e)
{
    Thread t = new Thread(SendMessage);
    t.IsBackground = true;
    sendMessage = textBoxSend.Text;
    RemoteIp = textBoxRemoteIP.Text;
```

```
    t.Start();
}
```

2.3.2 远程唤醒

目标计算机还没有启动操作系统时目标机还没有有效 IP 地址，是无法进行正常的网络通信功能，AMD 公司发明特殊方法能够实现远程开机，它称为远程唤醒 (Wake on Lan(WOL)，现在多数网卡都支持的功能。远程开机远程唤醒的实现方法是向目标主机发送特殊格式的数据包，它是 AMD 公司制作的 Magic Packet 软件的一种数据包，又称魔术包 (Magic Packet)。这种技术并非世界公认标准，但受到多家网卡制造商的支持，许多具有网络唤醒功能的网卡都能与之兼空。能够被远程唤醒的计算机对硬件有一定的要求，网卡、主板和电源必须支持，用户可查看主板的 BIOS 设置菜单，具有 Wake On Lan 选项的机器支持远程唤醒功能，它一般在 Power Management Setup 主菜单下面。

支持 WoL 的计算机已接通电源但处于关机的情况下网卡能够检测到魔术包，它的格式是连续 6 个字节的"FF" 和连续重复 16 次的 MAC 地址。魔术包可以被封装在任何网络协议的数据包中，目标机关机时没有进程绑定 IP 地址，也不能实现连接操作，无连接的 UDP 广播可发挥作用，广播的 IP 地址是 255.255.255.255 或者 192.168.0.255 均可，数据包的端口值被忽略，通过构造"FFFFFFFFFFFF"+ 连续重复 16 次的 MAC 地址形成数据包，网卡检测到魔术包便会唤醒计算机。获取机器的 MAC 地址应在开机时执行命令 getmac 或者 ipconfig /all 得到本机的 MAC 地址，也可用程序方法得到机器地址。

实验室上机时相邻的两台机器作为一组进行合作，首先获得 A 机 MAC 地址，关闭 A 机器，在 B 机器上编写程序，将 A 机的 MAC 地址填入程序的数据包中，构造包含目标 MAC 地址的 UDP 数据，发送广播即可实现远程唤醒功能，下面是参考实现代码：

```
Socket socket_send;
IPEndPoint iep;
EndPoint ep;
//设置缓冲数据流
byte[] send_data_buf;
int send_data_len;
IPEndPoint RemoteIpEndPoint;
send_data_buf = new byte[1024];
//初始化一个 Scoket 协议
socket_send = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
//设置该 scoket 实例的发送形式
socket_send.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.Broadcast, 1);
//发送端
//初始化一个发送广播和指定端口的网络端口实例
RemoteIpEndPoint = new IPEndPoint(IPAddress.Broadcast, 9095);
try
{
```

```
    //获取的网卡的 MAC 地址为 44-37-E6-03-16-01
    byte[] b_txt1 = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
    byte[] b_txt2 = { 0x44, 0x37, 0xe6, 0x03, 0x16, 0x01 };
    Buffer.BlockCopy(b_txt1, 0, send_data_buf, 0, 6);
    for (int i = 1; i <= 16; i++)
    {
      Buffer.BlockCopy(b_txt2, 0, send_data_buf, 6 * i, 6);
    }
    send_data_len = 6 * 17;
    socket_send.SendTo(send_data_buf, send_data_len, SocketFlags.None, RemoteIpEndPoint);
  }
  catch (Exception e2)
  {}
```

这段程序根据魔术包格式要求填充数据包，它使用广播地址作为通信的目标地址，所设置的通信端口在魔术包协议中被忽略，数据包发出后，目标网卡检测到自己的 MAC 地址时就会启动机器。这个项目只需编写一个发送端程序即可实现目的。

2.3.3 UDP 网络控制播放器

基于 UDP 协议实现网络通信时，协议流程比较简单，要开发窗体应用程序则要添加异步响应的机制。窗体程序是基于消息机制的，UDP 协议在 windows 平台中支持异步的发送和接收信息功能，UDP 的异步与窗体之间的联系是通过回调函数的方式。这里为区别发送和接收程序逻辑，设计两个窗体程序，发送端只负责发信息，接收端负责接收信息，根据信息的内容控制播放器的动作，接收端界面有一个 Windows Media Player 控件，它可以播放 mp3 格式文件，窗体界面如图 2-2所示。

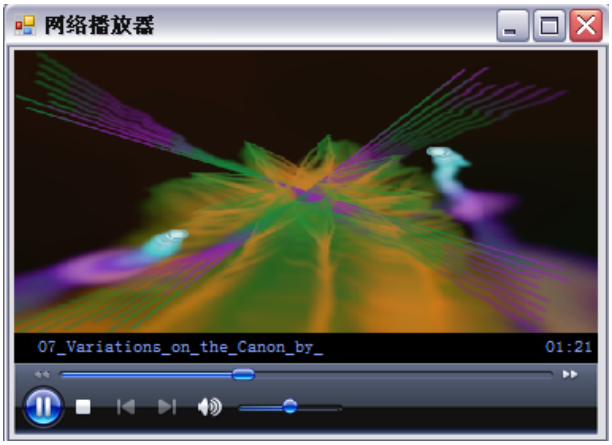

图 2-2 网络控制播放器

在接收端的窗体代码中添加下面的变量声明：

```
public static IntPtr main_wnd_handle;
//定义消息常数
```

```
public const int TRAN_UDP_IN = 0x500;
//动态链接库引入
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
int wParam, // first message parameter
int lParam // second message parameter
);
//定义 UDP 的接收端
public static Socket socket_recv;
public static EndPoint remote_ep;
public static byte[] recv_data_buf;
public static int recv_data_len;
public class Obj_one
{
}
public static Obj_one ob_1;
```

在窗体的启动事件中对变量进行初始化，设定了播放器播放的文件，新建了一个用于 UDP 通信的 Socket 对象，在调用 BeginReceiveFrom 之前，必须使用 Bind 方法显式地将 Socket 绑定到本地终结点，否则 BeginReceiveFrom 将会引发 SocketException。然后对象使用 Socket 类的 BeginReceiveFrom 方法准备进行异步的数据接收，这个异步方法需要指定一个事先定义的回调函数名，回调函数会在网络事件发生时自动调用。

```
private void Form1_Load(object sender, EventArgs e)
{
recv_data_buf = new byte[1024];
main_wnd_handle = this.Handle;
//初始化一个 Scoket 协议
socket_recv = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
//接收端
//初始化一个侦听局域网内部所有 IP 和指定端口
remote_ep = new IPEndPoint(IPAddress.Any, 9095);
//初始化一个发送广播和指定端口的网络端口实例
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9095);
socket_recv.Bind(iep);//绑定这个实例
axWindowsMediaPlayer1.URL = "Canon.mp3";
axWindowsMediaPlayer1.Ctlcontrols.stop();
ob_1 = new Obj_one();
socket_recv.BeginReceiveFrom(recv_data_buf, 0, 1024, SocketFlags.None, ref remote_ep,
ReceiveCallback, ob_1);
```

```
}
```

异步函数名为 ReceiveCallback，在回调函数中应调用 EndReceiveFrom 方法用于完成数据接收，向主窗体发送自定义消息，ReceiveCallback 函数参考代码如下：

```
//异步接收 UDP 数据的回调函数
public void ReceiveCallback(IAsyncResult ar)
{
  try
  {
    recv_data_len = socket_recv.EndReceiveFrom(ar, ref remote_ep);
    SendMessage(main_wnd_handle, TRAN_UDP_IN, 100, 100);
    if (recv_data_len > 0)
    {
      socket_recv.BeginReceiveFrom(recv_data_buf, 0, 1024, SocketFlags.None, ref remote_ep,
ReceiveCallback, ob_1);
    }
    else
    {
      socket_recv.Close();
    }
  }
  // Store the exception message.
  catch (SocketException e)
  {
    MessageBox.Show(e.Message);
  }
}
```

窗体的自定义消息处理函数会根据接收到的文本信息控制播放器动作，参考代码如下：

```
protected override void DefWndProc(ref Message m)
{//窗体消息处理重载
  switch (m.Msg)
  {
    case TRAN_UDP_IN:
      {
        string recv_str = Encoding.Default.GetString(recv_data_buf, 0, recv_data_len);
        switch (recv_str)
        {
          case "go":
            {
              axWindowsMediaPlayer1.Ctlcontrols.play();
              break;
```

```
                }
            case "pause":
                {
                    axWindowsMediaPlayer1.Ctlcontrols.pause();
                    break;
                }
            }
            break;
        }
    default:
        base.DefWndProc(ref m);
        break;
    }
}
```

发送端代码相比接收端则要简短的多，界面使用一个 combobox 控件和一个按钮，首先是变量声明部分：

```
public static Socket socket_send;
public static IPEndPoint iep;
public static EndPoint ep;
public static byte[] send_data_buf;//设置缓冲数据流
public static int send_data_len;
public static IPEndPoint RemoteIpEndPoint;
```

窗体的 Load 事件对变量进行初始化以下是参考代码：

```
private void Form1_Load(object sender, EventArgs e)
{
    send_data_buf = new byte[1024];
    //初始化一个 Scoket 协议
    socket_send = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
    //设置该 scoket 实例的发送形式
    socket_send.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.Broadcast, 1);
    //初始化一个发送广播和指定端口的网络端口实例
    RemoteIpEndPoint = new IPEndPoint(IPAddress.Broadcast, 9095);    }
```

按钮用于发送 combobox 控件的文本以下是参考代码：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        byte[] b_txt;// = Encoding.Default.GetBytes(DateTime.Now.ToString() +" "+ textBox2.Text);
        b_txt = Encoding.Default.GetBytes(comboBox1.Text);
        send_data_len = b_txt.Length;
```

```
        Array.Copy(b_txt, send_data_buf, b_txt.Length);
        socket_send.SendTo(send_data_buf, send_data_len, SocketFlags.None, RemoteIpEnd-
Point);
      }
      catch (Exception e2)
      {
        MessageBox.Show( e2.Message);
      }
    }
```

运行两个项目接收端收到发送端发送的字符串，经过字串比较可控制媒体文件的播放。

2.3.4 网络服务主机探测

通信软件使用 Socket 类须设置通信双方主机 IP 地址，要求用户明确输入 IP 地址参数即不现实又带来不少困惑，广域网内使用域名标识服务主机，局域网内可对网络主机进行探测。客户端发送广播包服务端主机提供回声反馈以显示其存在，客户端根据是否有反馈数据包确定服务主机存在。

演示实现网络服务主机探测包含两个项目，名称为 searchEcho 的窗体项目创建工作线程，它使用 Socket 对象向网络发送 UDP 广播数据包，采用 BeginReceiveFrom 方法绑定回调函数方法实现异步接收 UDP 数据；名称为 Echo 的窗体项目绑定接收数据包的回调函数，数据到达后自动发送回复包。如果 searchEcho 项目中线程发送十次数据包后未收到回复包则认为服务端不存在。

首先是 searchEcho 项目的演示代码，项目中声明的变量如下：

```
//动态链接库引入
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
int wParam, // first message parameter
int lParam // second message parameter
);
public static bool bServerExists;
public static bool udpSockClosed;
public const int NO_SERVER = 0x520;
public const int FOUND_SERVER = 0x521;
public static int iUdprecvDataLen;
public static int recv_data_len;
public static string strLocalHAddr;
public static string strInfo;
public static byte[] udpRecvDataBuf;
```

```
public static byte[] udpDataSendBuf;
public static IntPtr mWndhandle;
public static IPAddress LocalHostIPAddress;
public static IPEndPoint ipeServerRemoteIp;
public static IPEndPoint RemoteServerIpEP;
public static EndPoint remoteEp;
public static EndPoint remoteUdp_ep;
public static ManualResetEvent mrEventGotServer;
public static Socket sockUdpSearchSend;
public static Socket sockUdpSearchRecv;
public static bool threadNoExist;
```

窗体的 Load 事件初始化变量，检测可用网卡信息并生成有效广播地址。

```
private void FrmMain_Load(object sender, EventArgs e)
{
  mWndhandle = this.Handle;
  //初始时服务器标志为 false
  bServerExists = false;
  IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
  //检测可用网卡的方式是网关不为 0.0.0.0，掩码不为空
  NetworkInterface[] NetworkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
  foreach (NetworkInterface NetworkIntf in NetworkInterfaces)
  {
    IPInterfaceProperties IPInterfaceProperties = NetworkIntf.GetIPProperties();
    UnicastIPAddressInformationCollection UnicastIPAddressInformationCollection
      = IPInterfaceProperties.UnicastAddresses;
    foreach (UnicastIPAddressInformation UnicastIPAddressInformation
      in UnicastIPAddressInformationCollection)
    {
      if (UnicastIPAddressInformation.Address.AddressFamily
        == AddressFamily.InterNetwork)
      {
        if ((IPInterfaceProperties.GatewayAddresses != null) &&
          (UnicastIPAddressInformation.IPv4Mask != null))
        {
          if (IPInterfaceProperties.GatewayAddresses[0].Address.ToString().CompareTo("0.0.0.0")
!= 0)
          {
            strLocalHAddr = UnicastIPAddressInformation.Address.ToString();
            break;
          }
```

```
            }
          }
        }
    }
    if (strLocalHAddr == null)
    {
      MessageBox.Show(" 没有网络连接");
    }
    else
    {
      strLocalHAddr = strLocalHAddr.Substring(0,
        strLocalHAddr.LastIndexOf('.') + 1) + "255";
    }
    threadNoExist = true;
    udpRecvDataBuf =new byte[1024];
}
```

回调函数手于接收 UDP 协议数据回复包。

```
public static void ReceiveServerCallback(IAsyncResult ar)
{
    try
    {
      EndPoint tempRemoteEP = (EndPoint)ipeServerRemoteIp;
      //关闭 Socket 对象也将引发此回调事件，须进行区分
      if (!udpSockClosed)
      {
        iUdprecvDataLen = sockUdpSearchRecv.EndReceiveFrom(ar, ref tempRemoteEP);
        ipeServerRemoteIp = (IPEndPoint)tempRemoteEP;
        strInfo = Encoding.UTF8.GetString(udpRecvDataBuf, 0, recv_data_len);
        mrEventGotServer.Set();
      }
    }
    catch (SocketException se)
    {
      MessageBox.Show(se.Message);
    }
}
```

发送探测包的线程参考代码如下：

```
static void thrSearchServer()
{
    threadNoExist = false;
```

```
//使用事件对象准备
mrEventGotServer = new ManualResetEvent(false);
udpSockClosed = false;
//创建使用的 Socket 对象初始化一个 Scoket 协议
sockUdpSearchSend = new Socket(AddressFamily.InterNetwork,
  SocketType.Dgram, ProtocolType.Udp);
//设置该 scoket 实例的发送形式
sockUdpSearchSend.SetSocketOption(SocketOptionLevel.Socket,
  SocketOptionName.Broadcast, 1);
RemoteServerIpEP = new IPEndPoint(IPAddress.Parse(strLocalHAddr), 9095);
ipeServerRemoteIp = new IPEndPoint(IPAddress.Any, 0);
udpRecvDataBuf = new byte[1024];
remoteEp = new IPEndPoint(IPAddress.Any, 0);
sockUdpSearchRecv = new Socket(AddressFamily.InterNetwork,
  SocketType.Dgram, ProtocolType.Udp);
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9096);//接收服务器 UDP 的 9096 端口
sockUdpSearchRecv.Bind(iep);
sockUdpSearchRecv.BeginReceiveFrom(udpRecvDataBuf,
  0, 1024, SocketFlags.None, ref remoteEp, ReceiveServerCallback, new object());
byte[] sendDataBuf;
sendDataBuf = new byte[1024];
int sendDataLen;
try
{
  bool noUdpReply = true;
  byte[] b_txt;
  b_txt = Encoding.UTF8.GetBytes("are you online?");
  sendDataLen = b_txt.Length;
  Buffer.BlockCopy(b_txt, 0, udpDataSendBuf, 0, sendDataLen);
  for (int i = 0; (i < 10) && noUdpReply; i++)
  { //发送 100 个 UDP 数据包，相当于线程最多运行 20 秒钟
    sockUdpSearchSend.SendTo(udpDataSendBuf, sendDataLen,
      SocketFlags.None, RemoteServerIpEP);
    //等待时间间隔为 200 毫秒
    noUdpReply = !mrEventGotServer.WaitOne(200);
  }
  if (noUdpReply)
  { //通知窗体显示没有收到网络回复
    SendMessage(mWndhandle, NO_SERVER, 100, 100);
  }
```

```
      else
      {   //通知窗体显示收到主机回复
        SendMessage(mWndhandle, FOUND_SERVER, 100, 100);
      }
      //关闭 Socket 对象
      udpSockClosed = true;
      sockUdpSearchSend.Close();
      sockUdpSearchRecv.Close();
    }
    catch (Exception e2)
    { MessageBox.Show(e2.Message); }
    threadNoExist = true;
}
```

Echo 项目的演示代码，项目中声明的变量如下：

```
public class Obj_one
{
}
public static Obj_one ob_1;
public static IPEndPoint ipeRemoteClient;
public static int iUdprecvDataLen;
public static string strInfo;
public static byte[] udpRecvDataBuf;
public static byte[] udpSendDataBuf;
public static Socket sockUdpRecv;
public static Socket sockUdpSend;
```

Echo 项目中接收 UDP 数据包的回调函数演示代码如下，每次自动向发送者回复数据包：

```
public void ReceiveUdpCallback(IAsyncResult ar)
{
  try
  {
    EndPoint tempRemoteEP = (EndPoint)ipeRemoteClient;
    iUdprecvDataLen = sockUdpRecv.EndReceiveFrom(ar, ref tempRemoteEP);
    strInfo = Encoding.UTF8.GetString(udpRecvDataBuf, 0, iUdprecvDataLen);
    udpSendDataBuf = new byte[1024];
    ipeRemoteClient = (IPEndPoint)tempRemoteEP;
    IPEndPoint iep = new IPEndPoint(ipeRemoteClient.Address, 9096);
    byte[] databyte = Encoding.UTF8.GetBytes("hello from server");
    sockUdpSend.SendTo(databyte, iep);
    //继续接收 UDP 数据包
    sockUdpRecv.BeginReceiveFrom(udpRecvDataBuf, 0, 1024, SocketFlags.None, ref tem-
```

```
pRemoteEP, ReceiveUdpCallback, ob_1);
    }
    catch (SocketException se)
    {
      MessageBox.Show(se.Message);
    }
  }
    private void button1_Click(object sender, EventArgs e)
  {
    button1.Text = " 正在响应";
    button1.Enabled = false;
    sockUdpSend = new Socket(AddressFamily.InterNetwork,
      SocketType.Dgram, ProtocolType.Udp);//初始化一个 Scoket 协议, 用于发送数据
    //开始进行 UDP 数据包接收，接收到广播包就进行回复，表示服务器存在
    sockUdpRecv = new Socket(AddressFamily.InterNetwork,
      SocketType.Dgram, ProtocolType.Udp);//初始化一个 Scoket 协议, 用于接收数据
    //初始化指定端口的网络端口实例
    ipeRemoteClient = new IPEndPoint(IPAddress.Any, 9095);
    //绑定接收端口的网络端口实例
    EndPoint iep = new IPEndPoint(IPAddress.Any, 9095);
    sockUdpRecv.Bind(iep);//绑定这个实例
    ob_1 = new Obj_one();
    sockUdpRecv.BeginReceiveFrom(udpRecvDataBuf, 0, 1024,
      SocketFlags.None, ref iep, ReceiveUdpCallback, ob_1);
  }
```

读者可参考前面内容添加 searchEcho 项目的窗体消息重载函数并显示服务端 IP 信息，局域网中运行 searchEcho 和 Echo 项目可实现服务端 IP 端探测。

2.4 实验作业

1. 完成本实验中的程序项目。
2. 使用 Socket 类编写 UDP 异步接收的聊天软件。