第 5 章　基于 UDP 协议的程序设计

学习内容和目标

学习内容：

- 使用 UdpClient 类进行单播通信编程。
- 使用套接字选项设置进行广播通信编程。
- 使用套接字选项设置进行多播通信编程。

学习目标：

(1) 学会基于 UDP 协议的点对点通信程序设计和实现能力。

(2) 掌握广播通信程序设计方法及其实现过程。

UDP(User Datagram Protocol,用户数据报协议),主要用来支持那些需要在计算机之间快速传输数据的网络应用,包括网络视频会议系统在内的众多客户/服务器模式的网络应用都需要使用 UDP 协议。UDP 协议的独特应用是广播和多播编程,它还是 RSTP(实时控制传输协议)的基础,并为应用层的 DNS、RIP、SNMP 等协议提供服务。

5.1　UDP 协议介绍

UDP 和 TCP 协议的主要区别是两者在如何实现信息的可靠传递方面不同。TCP 协议中包含了专门的传递保证机制,当数据接收方收到发送方传来的信息时,会自动向发送方发出确认消息;发送方只有在接收到该确认消息之后才继续传送其他信息,否则将一直等待直到收到确认信息为止。与 TCP 不同,UDP 协议并不提供数据传送的保证机制。因此,也称为不可靠的传输协议。

虽然 UDP 是一个不可靠的协议,但它是分发信息的一个理想协议。例如,在屏幕上报告股票市场、在屏幕上显示航空信息等。UDP 也用在路由信息协议(Routing Information Protocol, RIP)中修改路由表。UDP 广泛用在多媒体应用中,例如,RealAudio 软件使用的 RealAudio audio-on-demand protocol 协议就是运行在 UDP 之上的协议;大多数 Internet 电话软件产品也都运行在 UDP 之上。

5.1.1　UDP 数据包格式

UDP 报头由 4 个域组成，即源端口号、目标端口号、数据报长度和校验和。其中，每个域各占用 2B。

UDP 协议使用端口号为不同的应用保留其各自的数据传输通道。UDP 和 TCP 协议正是采用这一机制实现对同一时刻内多项应用同时发送和接收数据的支持。

数据报的长度是指包括报头和数据部分在内的总的字节数。因为报头的长度是固定的，所以该域主要被用来计算可变长度的数据部分（又称为数据负载）。数据报的最大长度根据操作环境的不同而各异。从理论上说，包含报头在内的数据报的最大长度为 65 535B。不过，一些实际应用往往会限制数据报的大小，有时会降低到 8192B。

5.1.2　UDP 协议的主要特性

UDP 协议的重要特性表现在如下几个方面：

（1）UDP 是一个无连接协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去获取来自应用程序的数据，并尽可能快地把它发到网络上。在发送端，UDP 传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP 把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

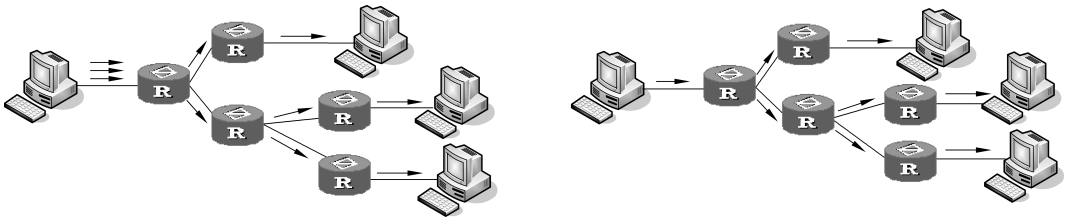
（2）由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

（3）UDP 数据包的报头很短，只有 8B，相对于 TCP 的 20B 报头的额外开销很小，适于快速传输的应用场合。

（4）吞吐量不受拥挤控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。

（5）为广播和多播提供专门服务。基于 UDP 协议向 Internet 发送广播消息的形式可以有三种：单播、广播和多播。单播方式是端对端的通信，能够穿透子网。如果需要传输到多个目的地，就需要发送多份相同的消息。广播和多播方式都是同时向多个设备发送消息，广播方式是向子网中的所有客户发送广播消息；而采用多播可以向 Internet 的不同子网发送广播消息，比如集团公司向下属分布的公司发布信息。单播和多播方式都能够向 Internet 的任意主机发送消息，但其特点不同，其传输形式如图 5-1 所示。如果需要发送广播消息到三个目的地，采用单播方式，就需要同时发出三份相同的数据包；而采用多播方式只需要发出一份数据包。显然，对于大量相同数据包的传输，采用多播方式的优势是明显的。

下面，首先阐述用于单播的 UdpClient 类的使用实例，然后，分别叙述广播和多播程序设计过程。其中，需要使用套接字选项和 D 类 IP 地址。



(a) 单播方式　　(b) 多播方式

图 5-1　单播和多播的传输形式

5.2　使用 UdpClient 类进行编程

前面几章已经介绍了使用 Socket 类进行网络编程的方法。与 TcpClient 类一样，UdpClient 类是基于 Socket 类的较高级别抽象，使用更加简单，多用于阻塞同步模式下发送和接收无连接 UDP 数据报。

5.2.1　UdpClient 类的使用方法

UdpClient 类的使用与标准的套接字流程基本一致，包括创建实例、收发数据和关闭 3 个阶段。

一个基本的 UdpClient 调用方式如下：

```
//第一阶段：创建 UdpClient 实例
UdpClient udpClient=new UdpClient();
IPAddress remoteAddress=IPAddress.parse("127.0.0.1");
IPEndPoint iep=new IPEndPoint(remoteAddress,8000);
//第二阶段：数据发送
byte[] sendBytes=System.Text.Encoding.Unicode.GetBytes("注意休息!");
udpClient.Send(sendBytes,sendBytes.Length,remoteAddress);
//第二阶段：数据接收
IPEndPoint iep2=new IPEndPoint(IPAddress.Any,0);
Byte[] receiveBytes=udpClient.Receive(ref iep2);
string getData=System.Text.Encoding.Unicode.GetString(receiveBytes);
…
//第三阶段：连接关闭
udpClient.Close();
```

由于 UdpClient()构造函数不同，所以在第一阶段还有下列几种调用方法：

```
IPAddress remoteAddress=IPAddress.parse("127.0.0.1");
UdpClient udpClient=new UdpClient("remoteAddress",8000);
```

或者为：

```
UdpClient udpClient=new UdpClient();
udpClient.Connect("www.software.org",8000);
```

最简单的调用为：

```
UdpClient udpClient=new UdpClient("www.software.org",8000);
```

在这3种方式下，由于明确了远程主机上的进程，所以Send()方法的调用更为简单：

```
udpClient.Send(tmpBytes,tmpBytes.Length);
```

常见的TcpClient类的常见属性和方法如表5-1和表5-2所示。

表5-1　UdpClient类的常用属性列表

属　性	描　述
Available	获取已经从网络接收且可供读取的数据量
Client	获取或设置基础System.Net.Sockets.Socket
DontFragment	获取或设置指定UdpClient实例是否允许对IP协议数据报进行分段的标志
EnableBroadcast	获取或设置指定UdpClient实例是否可以发送或接收广播数据包的标志
ExclusiveAddressUse	获取或设置System.Boolean值，该值指定System.Net.Sockets.UdpClient是否只允许一个客户端使用端口
MulticastLoopback	获取或设置是否将输出多播数据包传递给发送应用程序的标志
TTL	获取或设置指定由UdpClient发送的IP协议数据包的生存时间TTL的值

表5-2　UdpClient类的常用方法列表

方　法	功　能
BeginReceive()	从远程主机异步接收数据报
BeginSend()	将数据报异步发送到远程主机
Close()	关闭UDP连接
Connect()	使用指定的网络终结点建立默认远程主机
DropMulticastGroup()	退出多播组
EndReceive()	结束挂起的异步接收
EndSend()	结束挂起的异步发送
JoinMulticastGroup()	将指定的生存时间(TTL)与System.Net.Sockets.UdpClient一起添加到多播组
Receive()	返回已由远程主机发送的UDP数据报
Send()	将UDP数据报发送到远程主机

5.2.2 UdpClient 类的应用实例

以下给出一个简单的控制台应用程序，在本机上测试，设置服务器的端口号为 8000，客户机的端口号为 8001。如果是在两台计算机上测试，则端口号可以相同。

1. 服务器程序

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace UdpClientServer
{
    class Class1
    {
        static void Main()
        {
            StartListener();
            Console.ReadLine();
        }
        //
        private static void StartListener()
        {
            UdpClient udpServer=new UdpClient(8000);   //服务器方的端口号为 8000
            IPEndPoint myHost=null;
            try
            {
                while(true)
                {
                    Console.WriteLine("等待接收...");
                    byte[] getBytes=udpServer.Receive(ref myHost);
                    string getString=Encoding.Unicode.GetString(getBytes, 0,
                        getBytes.Length);
                        Console.WriteLine("接收信息: {0}",getString);
                        //收到消息"quit"时,跳出循环
                        if(getString=="quit") break;
                        //向客户端回送消息
                        string sendString="你好,多加保重!";
                        Console.WriteLine("发送信息: {0}",sendString);
                        byte[] sendBytes=Encoding.Unicode.GetBytes(sendString);
                        udpServer.Send(sendBytes,sendBytes.Length,"127.0.0.1",
                        8001);
                }
                udpServer.Close();
```

```
                Console.WriteLine("对方已经退出,请按 Enter 键退出。");
            }
            catch(Exception err)
            {
                Console.WriteLine(err.ToString());
            }
        }
    }
}
```

2. 客户机程序

```
using System;
using System.Net;
using System.Net.Sockets;
namespace testUdpClient
{
    class Class1
    {
        static void Main(string[] args)
        {
            string sendString="你好,继续努力!";
            Send(sendString);
            Send("quit");
            Console.ReadLine();
        }

        private static void Send(string message)
        {
            UdpClient udpClient=new UdpClient(8001);      //客户端的端口号为 8001
            try
            {
                Console.WriteLine("向服务器发送数据: {0}",message);
                byte[] sendBytes=System.Text.Encoding.Unicode. GetBytes(message);
                udpClient.Send(sendBytes,sendBytes.Length,"127.0.0.1",8000);
                if(message=="quit")
                {
                    Console.WriteLine("已经向对方发送 quit 信息,请按 Enter 键退出
                    程序。");
                    return;
                }
                IPEndPoint myHost=null;
                byte[] getBytes=udpClient.Receive(ref myHost);
                string getString=System.Text.Encoding.Unicode.GetString(getBytes);
                Console.WriteLine("接收信息: {0}",getString);
```

```
                udpClient.Close();
            }
            catch(Exception err)
            {
                Console.WriteLine(err.ToString());
            }
        }
    }
}
```

程序的运行界面如图 5-2 和图 5-3 所示。

图 5-2　客户机运行界面

图 5-3　服务器运行界面

课堂练习：

（1）手工输入客户机程序，以便进一步熟悉 C＃编程环境；

（2）开展 UDP 协议的单播测试，并通过修改网络地址后进行网络测试；

（3）修改以上服务器和客户机为窗体应用程序。

5.3　网络广播程序设计

按照 IPv4 标准规定的 IP 地址分类，广播地址有全球广播和本地广播两类地址：

全球广播地址是 255.255.255.255，表明数据包的目的地是网络上的所有设备。但是，由于路由器会自动过滤掉该全球广播，所以只有本地子网上的主机能够接收到分组。

本地广播是向本地子网中的所有设备发送广播消息，而其他网络不受影响。IP 地址由网络号和主机号两部分组成，A 类、B 类和 C 类 IP 地址中主机号为全 1 的地址都是直接广播地址，用来使路由器将一个分组以广播方式发送给特定网络上的所有主机。比如 C 类地址为 201.1.16.255，这是一个广播地址，只能作为目的地址。发送的分组将通过路由器向网络 201.1.16.0 进行广播，其原理如图 5-4 所示。此时，主机 201.1.16.2 和 201.1.16.56 都能收到该分组。

如果有多个进程都发送广播数据，则该子网将会阻塞，影响到网络性能，这是广播方式的缺点。

为了实现广播通信，需要在套接字函数中设置广播选项，然后使用 recvfrom()和 sendto()等函数收发广播数据。

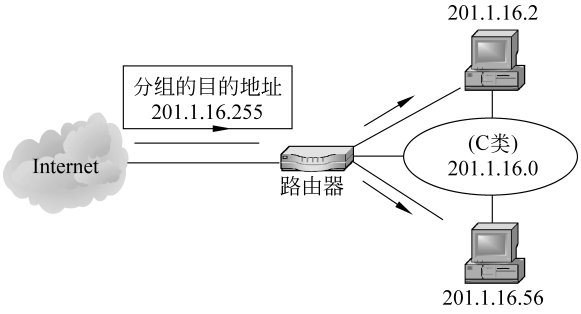


图 5-4　直接广播地址的应用原理示意图

5.3.1　广播程序设计示例

以下给出广播数据包的发送和接收程序。

1. 发送广播消息程序

```
Socket socket = new Socket (AddressFamily. InterNetwork, SocketType. Dgram,
ProtocolType.Udp);
IPEndPoint iep=new IPEndPoint(IPAddress.Broadcast,8000);
//设置 Broadcast 值为表示允许套接字发送广播信息,该值默认为(不允许)
socket. SetSocketOption ( SocketOptionLevel. Socket, SocketOptionName.
Broadcast,1);
//将发送内容转换为字节数组
byte[ ] sendBytes = System. Text. Encoding. Unicode. GetBytes (this. textBox1.
Text);
//向子网发送信息
socket.SendTo(sendBytes,iep);
socket.Close();
```

在程序中,关键的语句是 IP 地址设置和套接字选项设置。

在本机上测试时,对象实例 iep 应该修改为:

```
IPEndPoint iep=new IPEndPoint(IPAddress.Parse("127.0.0.1"),8000);
```

2. 接收广播消息程序

```
Socket socket=new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram,ProtocolType.Udp);
IPEndPoint iep=new IPEndPoint(IPAddress.Any,8000);
socket.Bind(iep);
EndPoint ep=(EndPoint)iep;
Byte[] getBytes=new byte[1024];
while(true)
{
    socket.ReceiveFrom(getBytes,ref ep);
```

```
        string getData=System.Text.Encoding.Unicode.GetString(getBytes);
        //注意不能省略 getData.TrimEnd('\u0000'),否则看不到后面的信息
        getData=getData.TrimEnd('\u0000')+"\n\n 希望继续接收此类消息吗?\n";
        string message="来自"+ep.ToString()+"的消息";
        DialogResult result=MessageBox.Show(getData,message,MessageBoxButtons.
        YesNo);
        if(result==DialogResult.No)
        {
            break;
        }
    }
```

程序中指定 IP 地址为 IPAddress.Any，表示提供一个 IP 地址，指示服务器应侦听所有网络接口上的客户机活动。

两个程序的运行界面如图 5-5 至图 5-7 所示。发送广播消息“现在下课，大家休息 5 分钟……”后，接收主机立即收到该消息，并提示是否继续接收此类消息。如果不接收，则显示图 5-7 状态。如果单击“继续接收消息”按钮，则能够继续收到发送方发来的广播消息。

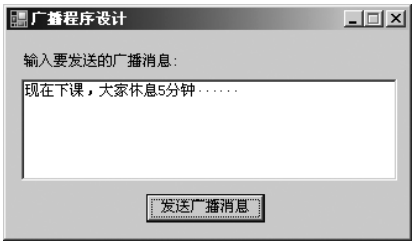


图 5-5　发送端运行界面

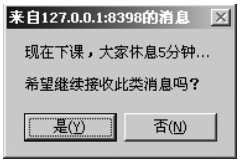


图 5-6　接收端收到消息

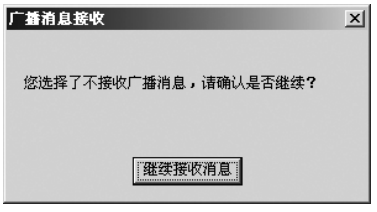


图 5-7　接收端的选择界面

课堂练习：

（1）请设计“垃圾信息过滤功能”，阻断发来的恶意广播信息；

（2）对于接收方，当收到垃圾信息后，如何反击对方，或者给予对方警告。

提示：可以分析对方的 IP 地址，将列为黑名单的 IP 地址发来的信息屏蔽，不显示到界面上。只要定位对方 IP 地址，就能够回复信息。

5.3.2　套接字选项设置方法

套接字选项设置属于套接字的高级应用，主要是调用 SetSocketOption 方法。其完整的调用原型有四种。

- SetSocketOption（SocketOptionLevel optionLevel，SocketOptionName optionName，bool optionValue）。

- SetSocketOption(SocketOptionLevel optionLevel, SocketOptionName optionName, byte[] optionValue)。
- SetSocketOption(SocketOptionLevel optionLevel, SocketOptionName optionName, int optionValue)。
- SetSocketOption(SocketOptionLevel optionLevel, SocketOptionName optionName, object optionValue)。

它们的差别在于第三个参数。

1. 定义套接字选项级别：SocketOptionLevel

在本节程序中，可以指定为 Socket 或 UDP，如表 5-3 所示。

表 5-3　套接字选项级别的选项列表

选项级别	描　述	选项级别	描　述
IP	仅适用于 IP 套接字	TCP	仅适用于 TCP 套接字
IPv6	仅适用于 IPv6 套接字	UDP	仅适用于 UDP 套接字
Socket	适用于所有套接字		

2. 定义选项名称：SocketOptionName

选项名称非常丰富，选用相应名称后直接影响选项值的设置，如表 5-4 所示。

表 5-4　套接字配置选项名称部分列表

配置选项名称	描　述	选项值类型
Broadcast	允许在套接字上发送广播消息	bool
DontRoute	不路由，将数据包直接发送到接口地址	bool
ExclusiveAddressUse	使套接字能够为独占访问进行绑定	bool
ReceiveBuffer	指定为接收保留的每个套接字缓冲区空间的总量。这与最大消息大小或 TCP 窗口的大小无关	int
ReceiveTimeout	接收超时。此选项只适用与同步方法，对异步方法无效	int
ReuseAddress	允许将套接字绑定到已在使用中的地址	bool
SendBuffer	指定为发送保留的每个套接字缓冲区空间的总量。这与最大消息或 TCP 窗口的大小无关	int
SendTimeout	发送超时。此选项只适用与同步方法，对异步方法无效	int

其中，选项 ReceiveTimeout 和 SendTimeout 的应用在 4.2.3 节的“超时控制方法”中已进行了说明。

3. 定义选项的辅助信息：optionValue

这需要与 SocketOptionName 配套使用，比如：

```
socket.SetSocketOption(SocketOptionLevel.Socket,SocketOptionName.Broadcast,1);
```