第15章　微软Visual Basic Winsock控件

本章讲解微软Visual Basic Winsock控件的问题。这是一种非常新的控件，用于将 Winsock接口简化成易于使用的Visual Basic内部接口。在这种控件问世之前，要想通过 Visual Basic进行网络编程，唯一的办法便是将所有 Winsock函数都从DLL（动态链接库）中导入，然后重新定义必要的结构。而你可以想象，那些结构的数量有多少！这一过程会耗费程序员大量的时间，且极易出错。常见错误包括类型声明的错误配合等等。然而，假如你真的需要直接从Winsock导入Visual Basic，从而获取更大的灵活性，亦可参考一下本书第二部分一直都在讲述的各个Visual Basic示例。在每个Visual Basic示例中，都包含了一个文件：Winsock.bas，它负责着必要常数及函数的导入。本章的重点只放在 Visual Basic Winsock控件身上。在本章的开头，将概述控件的属性及方法，然后提供几个例子，引导大家具体使用这种控件。

第一个Winsock控件是随Visual Basic 5.0引入的。后来，随同Visual StudioService Pack 2（SP2），提供了一个经过修订的控件。后来又发布了 Service Pack 3（SP3），但该控件并未对SP2的那个版本进行什么改进。而在 Visual Basic 6.0中，我们拿到了最新的 Winsock控件。在本章要结束的时候，会对其各个版本的差异进行详细讲解。

要注意的是，Winsock控件只为Winsock API函数提供了一个基本接口。和Winsock不一样，Winsock并非一个“与协议无关”的接口，它只能使用 IP传送协议！除此以外，该控件建立在Winsock 1.1版本规范的基础上。控件同时支持 TCP以及UDP，但支持程度偏低，许多特性都无法使用。控件本身不能访问任何套接字选项。换言之，像多播及广播之类的特性便无法使用。总之，只有在你仅仅需要基本的网络数据通信的前提下，才可考虑使用 Winsock控件。它的性能无论如何都不是最好的，因为在数据传给系统之前，需要先将数据缓冲到控件内部，这样便增加了额外的开销，同时产生了一些不确定性的因素。

15.1　属性

现在，大家对控件具有的一些功能已有了一个基本概念。接下来，且让我来看看该控件提供了哪些属性。在表 15-1中，我们对可用的属性进行了一番总结。这些属性可对控件的行为产生影响，同时可用来获取与控件状态有关的信息。

通过第7章的阅读，大家应该早已熟悉了这些基本属性。它们明显类似于第　7章在客户机／服务器示例中讨论的各个基本 Winsock函数。但也有少数几个属性并非与 Winsock API存在严格的对应关系，应好好地设置它们，以便正确地使用该控件。首先要注意的是 Protocol属性，我们应设置它，告诉控件自己需要的是哪种类型的套接字 —— 要么是SOCK_STREAM（数据流），要么是SOCK_DGRAM（数据报）。在实际工作中，控件会自动执行实际的套接字创建工作，而该属性是我们唯一能对其施加控制的渠道。在连接成功建立之后，或在完成了服务器的绑定，令其等候一次连接之后，可读取 SocketHandle（套接字句柄）属性的内容。之所以要这样做，是由于在某些情况下，我们希望将句柄传递给自一个　DLL导入的其他Winsock API函数。利用State（状态）属性，我们可获取与控件目前正在做的工作有关的信息。

这种信息是至关重要的，因为控件本身是以“异步”方式工作的，事件可能在什么时刻“触发”。利用这个属性，便可确定套接字正处在一个有效的状态，可放心大胆地进行后续的操作。在表15-2中，我们对套接字各种可能的状态进行了总结，同时讲解了它们的含义。

表15-1 Winsock控件属性

属性名称	返回值	是否为只读	说 明
BytesReceived	Long	是	返回接收缓冲区内“待决”（等候处理）的字节数量。可用GetData方法来取回数据
LocalHostName	String	是	返回本地机器的名字
LocalIP	String	是	返回本地机器采用“点分十进制”格式的IP地址
LocalPort	Long	否	返回要使用的本地端口集合。若将端口设为0，表明系统需要随机性地挑选一个可用的端口。通常，只能一个客户机使用端口0
Protocol	Long	否	为控件返回或设置协议，注意控件本身支持TCP或UDP。需要设置的常数值包括sckTCPProtocol（TCP协议）以及sckUDPProtocol（UDP协议），分别对应于值0和1
RemoeHost	String	否	返回或设置远程机器的名字。既可使用字串形式的主机名，亦可使用采用“点分十进制”格式的字串表达形式
RemoteHostIP	String	是	返回远程机器的IP地址。对于TCP连接，该字段会在成功建立连接之后设置；而对于UDP连接，该字段会在产生了DataArrival（数据抵达）事件后设置，即设置之后，会包含负责发送数据的那台机器的IP地址
RemotePort	Long	否	返回或设置要连接的远程端口
SocketHandle	Long	是	返回与套接字句柄对应的一个值
State	Integer	是	返回控件状态，注意这是一个列举类型。请参考表15-2，了解各个不同的套接字常数

表15-2 套接字状态

常 数	值	含 义
sckClosed	0	默认值，表示关闭
sckOpen	1	打开
sckListening	2	正在监听连接
sckConnectionPending	3	连接请求已抵达，但尚未完成
sckResolvingHost	4	主机名正在解析
sckHostResolved	5	主机名解析已完成
sckConnecting	6	连接请求已经启动，但尚未完成
sckConnected	7	连接建立（完成）
sckClosing	8	对方正在初始化一次“关闭”
sckError	9	产生某个错误

15.2 方法

Winsock控件仅提供了少数几个方法。大多数方法名字都与它们的Winsock相似，其中仅存在两处例外，分别是GetData和SendData方法。其中，GetData用来读入待决数据。通常，一旦触发了DataArrival事件（通知我们数据已经抵达），便应调用GetData方法。另外，SendData显然用于数据的发送。除此以外，一个名为PeekData（偷看数据）的方法类似于调用Winsock函数recv，同时设定MSG_PEEK选项。但同样要注意的是，对消息的“偷看”是一种不好的

编程习惯，应尽量避免。在表15-3中，我们列出了所有可用的方法，以及它们的参数。至于这些方法本身的细节问题，将在本章稍后展示客户机和服务器示例的小节中，进行深入探讨。

表15-3 Winsock控件方法

方 法	参 数	返回值	说 明
Accept	RequestID	Void	只能用于TCP连接。处理一个ConnectionRequest事件时，用这个方法接受进入的连接请求
Bind	LocalPort，LocalIP	Void	将套接字同指定的本地端口和IP绑定到一起。假如安装有多个网络适配器（网卡），请使用Bind。Bind必须在Listen之前调用
Close	无	Void	选择连接或者监听套接字
Connect	RemoteHost，RemotePort	Void	在指定的RemotePort（远程端口）编号上，建立与指定RemoteHost（远程主机）的一个TCP连接
GetData	Data，Type，MaxLen	Void	取回当前待决的数据。Type和MaxLen参数均是可选的。Type参数定义要读入的数据的类型。MaxLen参数指定最多要取回多少字节或字符。对于除字节数组及字串以外的其他类型，GetData会忽略MaxLen参数的设定
Listen	无	Void	创建一个套接字，并将其置入监听模式。Listen只用于TCP连接
PeekData	Data，Type，MaxLen	Void	行为与GetData几乎完全一致，只是数据不会从系统缓冲区中删去
SendData	Data	Void	将数据发给远程计算机。假如传递了一个UNICODE字串，那么它会先转换成一个ANSI字串。对于二进制数据，无论如何都要使用一个字节数组

15.3 事件

“事件”是在发生了一个特定的动作后，以异步方式调用的例程。在我们编制的Visual Basic应用程序中，必须对Winsock控件可能生成的各种事件加以控制，以便顺利地使用该控件。通常，这些事件是由通信对方发起的某个行动而触发的。举个例子来说，如果TCP连接的某一方关闭了套接字，便会产生TCP“半关闭”（Half-close）事件。发起关闭的那一方会生成一个FIN信号，而另一方会用一个ACK信号作出响应，确认此次关闭请求。接收FIN的那一方会触发一个Close事件。这样一来，我们的Winsock应用程序便知道对方不再发送数据。随后，应用程序需要读入剩余的所有数据，并调用自己这一方的Close方法，将连接完全关闭掉。在表15-4中，我们总结了可能触发的各种Winsock事件，对每个事件进行了说明。

表15-4 Winsock控件事件

事 件	参 数	说 明
Close	无	远程计算机关闭连接时发生
Connect	无	Connect方法成功完成后发生
ConnectionRequest	RequestID	远程机器请求建立一个连接时发生
DataArrival	bytesTotal	新数据抵达时发生
Error	Number、Description、Scode、Source、HelpFile、HelpContext、CancelDisplay	产生一个Winsock错误时发生
SendComplete	无	发送操作完成后发生
SendProgress	bytesSent、bytesRemaining	数据正在发送时发生

15.4 UDP示例

现在，让我们来探讨一个简单的UDP应用程序。首先请看看本书配套光盘Chapter15目录下的SockUDP.vbp这个Visual Basic项目文件。该项目文件成功编译并运行之后，会看到和图15-1差不多的一个对话框。这个示范程序本身既是UDP消息的一个发送者，也是一个接收者，所以只需使用一个程序实例，便能同时实现消息的发送与接收。此外，在程序清单 15-1中，列出了窗体、按钮以及Winsock控件背后的所有代码。

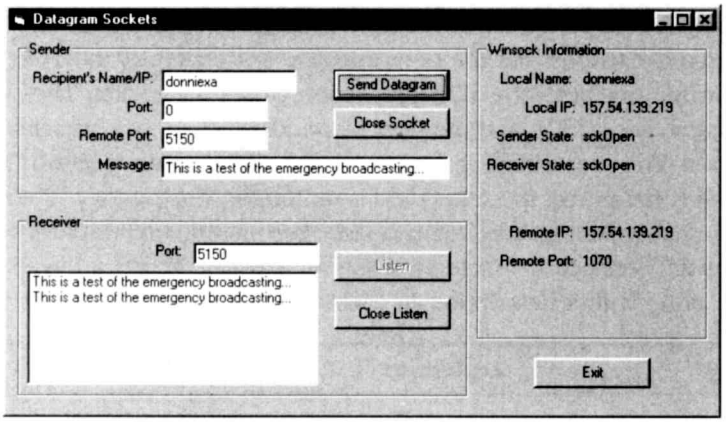

图15-1 示范UDP应用

请观察窗体，可注意到其中总共含有两个Winsock控件。一个控件用于发送数据报，另一个用于接收数据报。另外还有三个分组框：一个由发送者专用，一个由接收者专用，另一个（最右边的那个）则用于显示常规的Winsock信息。对发送者来说，需要一些地方来放置接收者的主机名或IP地址。设置RemoteHost属性时，既可使用机器文本形式的名字，亦可使用数字形式的、采用“点分十进制”格式表达的IP地址字串。如有必要，控件会自动解析出名字。另外，还需要一个远程端口，以便向其发送UDP数据包。除此以外，请注意用于本地端口的文字框（txtSendLocalPort）。对发送者来说，从哪个本地端口发出数据其实关系不大，只需注意将数据发到那个端口就可以了（目标端口）。如将本地端口设为0，系统会自动分配一个尚未使用的端口。最后一个文本框（txtSendData）用于输入要发送出去的字串。另外，请注意共有两个命令按钮：一个用于发送数据，另一个用于关闭套接字。要想发出数据报，首先必须将Winsock控件同一个远程地址、一个远程端口以及一个本地端口绑定起来，然后才能进行实际的数据发送操作。这意味着假如想更改这三个参数中的任何一个，首先便需要关闭套接字，再与新参数重新绑定到一起。这正是窗体中提供了一个Close Socket（关闭套接字）的原因。

程序清单15-1 UDP示范代码

```
Option Explicit

Private Sub cmdExit_Click()
    Unload Me
End Sub

Private Sub cmdSendDgram_Click()
```

```
    ' If the socket state is closed, we need to bind to a
    ' local port and also to the remote host's IP address and port
    If (sockSend.State = sckClosed) Then
        sockSend.RemoteHost = txtRecipientIP.Text
        sockSend.RemotePort = CInt(txtSendRemotePort.Text)
        sockSend.Bind CInt(txtSendLocalPort.Text)

        cmdCloseSend.Enabled = True
    End If
    '
    ' Now we can send the data
    '
    sockSend.SendData txtSendData.Text
End Sub

Private Sub cmdListen_Click()
    ' Bind to the local port
    '
    sockRecv.Bind CInt(txtRecvLocalPort.Text)
    '
    ' Disable this button since it would be an error to bind
    ' twice (a close needs to be done before rebinding occurs)
    '
    cmdListen.Enabled = False
    cmdCloseListen.Enabled = True
End Sub

Private Sub cmdCloseSend_Click()
    ' Close the sending socket, and disable the Close button
    '
    sockSend.Close
    cmdCloseSend.Enabled = False
End Sub

Private Sub cmdCloseListen_Click()
    ' Close the listening socket
    '
    sockRecv.Close
    ' Enable the right buttons
    '
    cmdListen.Enabled = True
    cmdCloseListen.Enabled = False
    lstRecvData.Clear
End Sub
Private Sub Form_Load()
    ' Initialize the socket protocols, and set up some default
    ' labels and values
    '
    sockSend.Protocol = sckUDPProtocol
    sockRecv.Protocol = sckUDPProtocol

    lblHostName.Caption = sockSend.LocalHostName
    lblLocalIP.Caption = sockSend.LocalIP

    cmdCloseListen.Enabled = False
    cmdCloseSend.Enabled = False
```

```
    Timer1.Interval = 500
    Timer1.Enabled = True
End Sub

Private Sub sockSend_Error(ByVal Number As Integer, _
        Description As String, ByVal Scode As Long, _
        ByVal Source As String, ByVal HelpFile As String, _
        ByVal HelpContext As Long, CancelDisplay As Boolean)
    MsgBox Description
End Sub

Private Sub sockRecv_DataArrival(ByVal bytesTotal As Long)
    Dim data As String

    ' Allocate a string of sufficient size, and get the data;
    ' then add it to the list box
    data = String(bytesTotal + 2, Chr$(0))
    sockRecv.GetData data, , bytesTotal
    lstRecvData.AddItem data
    ' Update the remote IP and port labels
    '
    lblRemoteIP.Caption = sockRecv.RemoteHostIP
    lblRemotePort.Caption = sockRecv.RemotePort
End Sub

Private Sub sockRecv_Error(ByVal Number As Integer, _
        Description As String, ByVal Scode As Long, _
        ByVal Source As String, ByVal HelpFile As String, _
        ByVal HelpContext As Long, CancelDisplay As Boolean)
    MsgBox Description
End Sub

Private Sub Timer1_Timer()
    ' When the timer goes off, update the socket status labels
    '
    Select Case sockSend.State
        Case sckClosed
            lblSenderState.Caption = "sckClosed"
        Case sckOpen
            lblSenderState.Caption = "sckOpen"
        Case sckListening
            lblSenderState.Caption = "sckListening"
        Case sckConnectionPending
            lblSenderState.Caption = "sckConnectionPending"
        Case sckResolvingHost
            lblSenderState.Caption = "sckResolvingHost"
        Case sckHostResolved
            lblSenderState.Caption = "sckHostResolved"
        Case sckConnecting
            lblSenderState.Caption = "sckConnecting"
        Case sckClosing
            lblSenderState.Caption = "sckClosing"
        Case sckError
            lblSenderState.Caption = "sckError"
        Case Else
            lblSenderState.Caption = "unknown"
    End Select
```

```
    Select Case sockRecv.State
        Case sckClosed
            lblReceiverState.Caption = "sckClosed"
        Case sckOpen
            lblReceiverState.Caption = "sckOpen"
        Case sckListening
            lblReceiverState.Caption = "sckListening"
        Case sckConnectionPending
            lblReceiverState.Caption = "sckConnectionPending"
        Case sckResolvingHost
            lblReceiverState.Caption = "sckResolvingHost"
        Case sckHostResolved
            lblReceiverState.Caption = "sckHostResolved"
        Case sckConnecting
            lblReceiverState.Caption = "sckConnecting"
        Case sckClosing
            lblReceiverState.Caption = "sckClosing"
        Case sckError
            lblReceiverState.Caption = "sckError"
        Case Else
            lblReceiverState.Caption = "unknown"
    End Select
End Sub
```

15.4.1 UDP消息的发送

现在，大家已知道了作为一个发送者，它具有哪些方面的常规能力。接下来，且让我们来看看幕后的代码。首先是 Form_Load_例程。第一步是把 socksendWinsock控件的Protocol（协议）属性设为UDP，这是利用sckUDPProtocol列举类型来完成的。除了禁用 cmdCloseSend命令按钮外，该例程中的其他命令都没有用于发送数据。我们这样做是为了完整起见，因为在一个已经关闭的控件上调用 Close（关闭）命令毫无意义。注意， Winsock控件的默认状态就是关闭的。

接下来，我们看看 cmdSendDgram_Click，单击“ 发送数据 ”按钮就会触发这一命令。这是发送UDP消息的关键。示范代码的第一步是检查套接字的状态。如果套接字处于关闭状态，代码就会把一个 UDP套接字和一个远程地址、远程端口以及本地端口绑定在一起。一旦把 UDP Winsock控件与这些参数绑定在一起，这个控件的状态就会从 sckClosed变成sckOpen。如果不执行代码检查，就试图在每一次的数据发送上绑定套接字，则会产生运行时错误 40020“ 当前状态下的操作无效 ”（Invalid operation at current state）。一旦绑定了套接字，这个套接字就会一直存在，直到关闭。这就是代码在一旦绑定 Winsock控件后，为发送套接字启用“ 关闭套接字 ”按钮的原因。最后一步是随用户打算发送的数据一起，调用 SendData方法。SendData方法返回后，代码便完成数据的发送了。

只有两个子例程和发送UDP消息相关。第一个是cmdCloseSend，名符其实的关照套接字，允许用户在再次发送数据之前，改变远程主机、远程端口或本地端口这几个参数。另一个是 sockSend_Error，它是一个Winsock事件，只要产生 Winsock错误，就会触发这一事件。由于UDP是不可靠的，所以几乎不会产生错误。即便出现了错误，代码也只是简单地显示错误说明。这个应用程序中，用户看到的仅仅只是“ 目标不可抵达 ”这个消息。

15.4.2 UDP消息的接收

众所周知，利用Winsock控件发送UDP包是非常简单而直接的。UDP包的接收更简单。我们回头看看Form_Load例程，以便了解接收UDP消息需要怎么做。从利用Winsock控件发送数据可以看出，代码把Protocol属性设为UDP。另外，还禁用了“ Close Listen ”（关闭监听）按钮。再次提醒大家注意，关闭一个已关闭的套接字毫无意义，但为了完整起见，示范代码中仍然执行了关闭。同时，在编程过程中，最好经常性地设想一下“如果这时调用 X方法，又会如何呢？”。许多程序员碰到的控件问题均由于在控件状态无效时调用方法。一个例子就是：在一个已经连接的Winsock控件上调用connect（连接）方法。

要监听接入的UDP包，先来看看cmdListen_Click例程。这是“ Listen ”（监听）按钮句柄。只需一步，便可监听接入 UDP包：在接收Winsock控件上调用Bind方法，投递用户打算在上面监听接入UDP数据报的本地端口。在监听接入 UDP数据包时，代码只需要本地端口，和发送数据的远程端口无关。在示范代码绑定接收 Winsock控件之后，就禁用了 cmdListen按钮，这样可防止用户再次单击“监听”按钮。试图绑定一个已绑定的控件会失败，出现运行时错误。

此时，sockRecv控件已注册为接收UDP数据。控件在与它绑定的端口上接收 UDP数据时，会触发DataArrival事件。这个事件是在sockRecv_DataArrival例程中执行的。投入该事件的参数，bytesTotal是可以读取的字节数。代码分配了一个字串，该字串的长度略大于正在阅读的数据长。然后，代码调用 GetGetData方法，把分配得到的字串当作第一个参数送出去。第二个参数默认为Visual Basic类型vbString，第三个参数指定需要读取的字节数，在这个例子中，是bytesTotal这个值。如果代码请求读取的字节数小于 bytesTotal参数指定的值，就会产生运行时错误。一旦把数据读入了字符缓冲区，代码就会把它增加到读取的消息列表框中。这个子例程的最后几个步骤是：为远程主机的 IP地址和端口号设置标签说明。每次收到 UDP包后，针对才收到的包，把RemoteHostIP和RemotePort属性设为远程主机的IP地址和端口号。因此，如果数据报收到多个主机发来的UDP数据包，这些属性值就会频繁发生变化。

与接收UDP消息有关的最后两个子例程是 cmdCloseListen_Click和sockRecv_Error。单击“关闭监听”按钮，就可以实行 cmdCloseListen_Click句柄了。这个例程只是在Winsock控件上简单地调用Close方法。关闭UDP控件会释放基层套接字描述符。只要出现 Winsock错误，就会触发sockRecv_Error事件。正如前一小节中提到的那样，由于 UDP本身的不可靠性，因此，有时会发生UDP错误。

15.4.3 获取Winsock信息

我们的UDP示范代码中，最后一部分是“ Winsock Information group box ”（Winsock信息组框）。本地名和本地IP标签都是在表单加载时设置的。表单加载和 Winsock控件一旦作为例示，LocalHostName和LocalIP属性就会被设为主机名和主机的 IP地址，而且，任何时候都可以读取。下面两个标签“发送端状态”和“接收端状态”，显示了应用程序所用的两个Winsock控件的当前状态。状态信息的刷新频率是每半秒一次。“计时器”控件也会进入信息中。每隔500毫秒，计时器控件便会触发一次“计时器句柄”，查询套接字状态和更新标签。我们只显示了套接字状态。最后两个标签，“远程IP”和“远程端口”，是在每次收到UDP消息时设置的，这一点，我们在前一段曾讲过。

15.4.4 运行UDP示例

现在，大家应该已经理解了如何收发 UDP消息。接下来，让我们来看看它的一个实际运行示例。要想对其进行测试，最好的办法是在三台不同的机器上分别运行该应用程序的一个实例。在其中一个程序中，点击“ Listen ”（监听）按钮。而在另外的两个程序中，请在“ Recipient’s Name/IP ”（接收者的名字／IP）文本框内输入第一个应用程序正在运行的那台机器的名字。你既可输入一个主机名，也可输入对应的 IP地址。接下来，点击几次“ Send Datagram ”（发送数据报）按钮，消息应该在接收者的消息窗口内显示出来。收到每一条消息之后，在“ Winsock Information ”（Winsock信息）区域，显示内容应发生更新，填上消息发出时所在的那个发送者的IP地址以及端口号。甚至可用与接收者相同的应用程序上的“ Sender ”（发送者）命令，在同一台机器上发送消息。

另一个有趣的测试是使用由子网定向的广播或广播数据报。假定我们的三台测试机器都位于同一个子网上，我们可将一个数据报发送到一个指定的子网，所有正在监听的应用程序都应该接收到消息。举个例子来说，在我们的测试机器中，有两个是单址机（单一宿主），IP地址分别是157.54.185.186和157.54.185.224。最后一个机器则是多址机（多宿主），它的IP地址是169.254.26.113以及157.54.185.206。可以看出，这三台机器都共享同一个子网：157.54.185.255。现在，让我们先转移一下重点，讨论一个重要的细节。如果你打算接收 UDP消息，那么在调用 Bind方法时，必须隐式地同网络绑定中保存的第一个 IP地址绑定到一起。如果你的机器只安装了一张网卡，这样做便足够了。但某些情况下，机器中可能安装了多张网卡（有多个网络接口），造成在一台机器内存在多个 IP地址。这些情况下， Bind方法的第二个参数必须设为准备绑定的具体 IP地址。不幸的是， Winsock控件属性LocalIP只能返回一个IP地址。此外，控件本身并没有提供其他方法来取得同本地机器关联在一起的其他 IP地址。

现在，且让我们来试试广播通信。首先关闭在每个应用程序运行实例上的发送或监听套接字。在两台单址机上，请点击“ Listen ”（监听）按钮，使每台机器都能接收到数据报消息。在此，我们不打算使用那台多址机，这是由于在代码中，我们尚未同任何具体的 IP地址绑定到一起。在第三台机器上，请输入接收者的地址：157.54.185.255，然后点击几次“ Send Data ”（发送数据）按钮。随后，应发现两个监听应用程序都会收到消息。假如负责发送的机器也是一个“ 多址机 ”，那么大家可能会感到不解，它如何决定数据报该从哪个网络接口发送出去呢？事实上，路由表在这里会发挥至关重要的作用，它可以决定用于发送消息的最佳接口，只要事先知道了消息的目标地址以及本地机器上每个接口的地址。我们要进行的最后一次测试是关闭第三台机器上的发送者套接字。请将接收者的地址设为 255.255.255.255，然后点击几次“ Send Datagram ”（发送数据报）按钮。其结果应该是相同的：另外两个监听程序会接收到消息。然而，对多址机来说，此时存在的一个区别是， UDP消息已经广播到同机器连接的每一个网络上了。

15.4.5 UDP状态

现在，对于到底采用哪种方法调用顺序，以便成功地收发数据报，大家可能有些迷惑不解。如前面所述，用 Winsock控件编程时，最容易犯的错误是：调用一个方法，但那个方法的操作对控件当前的状态来说却是无效的。为帮助大家避免此类错误，请看图 15-2。该图演示了使用 UDP消息时，套接字状态的变化情况。请注意，默认的起始状态无论如何都是

sckClosed。而且对于无效的主机名来说，不会产生任何错误。

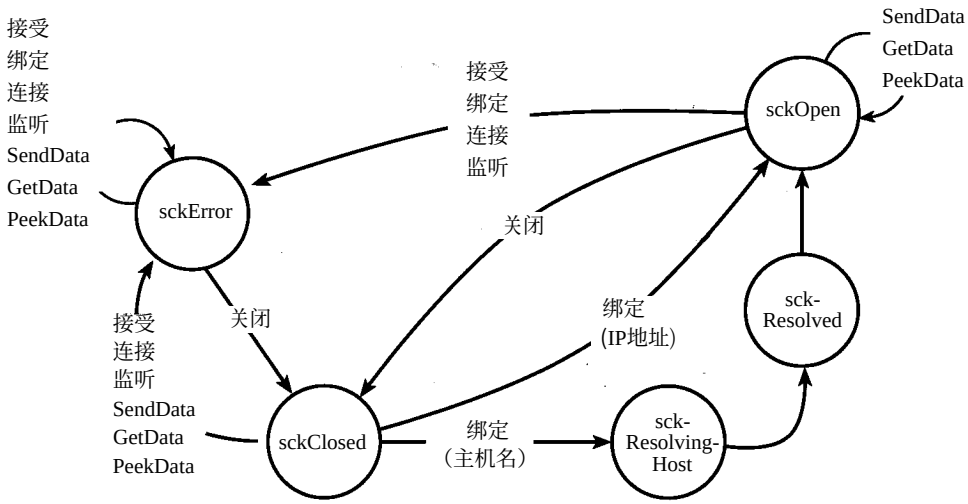

图15-2 UDP状态示意图

15.5 TCP示例

假如将Winsock控件用于TCP协议的处理，那么和处理UDP时比较起来，程序员需要做更多的工作，整个过程显得比较复杂。和我们处理 UDP时一样，首先会向大家展示一个示范性的TCP程序。接下来，讲述该程序的一些规格，以便对成功使用 TCP连接所涉及到的步骤有一个大致了解。在图 15-3中，我们展示了这个程序运行时的样子。程序清单 15-2则列出了该程序的源代码。在本书配套光盘的第 15章目录下，大家也可找到该Visual Basic项目文件的源码，文件名是SockTCP.vbp。

首先来观察一下图 15-3的窗体布局，对该程序的功能有一个大致的印象。同样，其中有三个分组框：TCP Server（TCP服务器）、TCP Client（TCP客户机）以及Winsock Information（Winsock信息）。首先让我们来看看该程序的 TCP服务器部分。服务器提供了一个文本框，名为txtServerPort，用于指定服务器打算与之绑定的本地端口，以便监听进入的客户机连接请求。此外，服务器还提供了两个按钮。一个用于将服务器置于监听模式，另一个用于关闭服务器，并停止接受进入连接。最后，服务器还提供了一个 Winsock控件，名为sockServer。观察一下属性页，便会发现Index属性被设为0。这意味着，该控件实际上是一个数组，可容纳 Winsock控件的多个实例。0表示在窗体载入时，只会创建它的一个实例（亦即数组的元素 0）。在任何时候，我们都可在一个数组元素中动态载入 Winsock控件的另一个实例。

Winsock控件数组实际上是发挥服务器功能至关重要的一个环节。请记住，每个 Winsock控件只有一个套接字句柄同它关联在一起。通过第 7章的学习，大家已经知道，服务器接受一个进入连接时，会新建一个套接字，以便对那个连接进行控制。对我们的应用程序来说，它的设计宗旨是：在建好一个客户机连接后，能够动态装载额外的 Winsock控件，以便连接能够传给新载入的控件，同时不会打断服务器套接字对连接的控制。要想达到这一目的，另一个方法是在程序设计的时候，将数量为 x的Winsock控件实际地置入窗体。然而，这样做的效率显然非常低下，以后也不易于扩展。应用程序启动时，必须耗费大量的时间来装载每个控件

所需的资源。此外，还必须解决具体打算使用多少个控件的问题。放置了 x个控件之后，能够同时连接的客户机数量便限定在 x之内。假如应用程序只允许同时建立固定数量的客户机连接，那么事先在窗体中设置固定数量的 Winsock控件，便显得较为合理，而且可能比使用数组简单得多。然而，对多数应用程序来说， Winsock控件数组是最佳的编程方案。

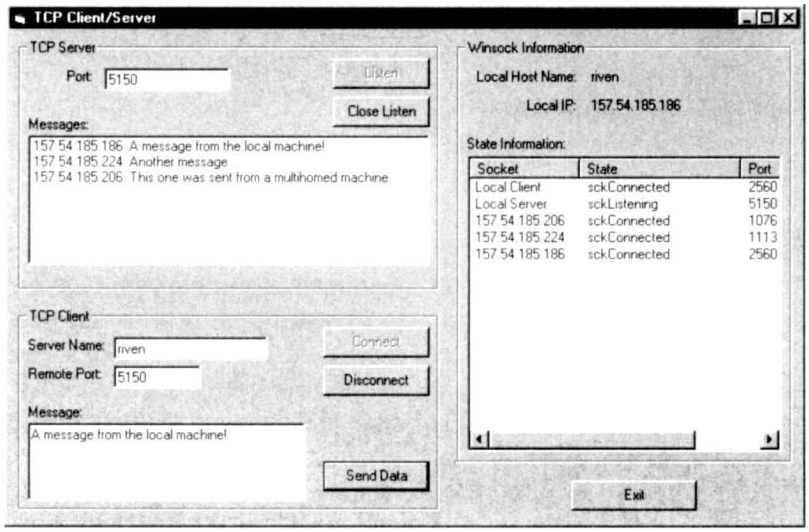

图15-3 示范TCP程序

程序清单15-2 示范TCP程序

```
Option Explicit

' The index value of the last Winsock control dynamically loaded
' in the sockServer array
Private ServerIndex As Long

Private Sub cmdCloseListen_Click()
    Dim itemx As Object
    ' Close the server's listening socket. No more
    ' clients will be allowed to connect.
    '
    sockServer(0).Close
    cmdListen.Enabled = True
    cmdCloseListen.Enabled = False

    Set itemx = lstStates.ListItems.Item(2)
    itemx.SubItems(2) = "-1"
End Sub

Private Sub cmdConnect_Click()
    ' Have the client control attempt to connect to the
    ' specified server on the given port number
    '
    sockClient.LocalPort = 0
    sockClient.RemoteHost = txtServerName.Text
    sockClient.RemotePort = CInt(txtPort.Text)
    sockClient.Connect
```

```
    cmdConnect.Enabled = False
End Sub

Private Sub cmdDisconnect_Click()
    Dim itemx As Object
    ' Close the client's connection and set up the command
    ' buttons for subsequent connections
    '
    sockClient.Close

    cmdConnect.Enabled = True
    cmdSendData.Enabled = False
    cmdDisconnect.Enabled = False
    ' Set the port number to -1 to indicate no connection
    '
    Set itemx = lstStates.ListItems.Item(1)
    itemx.SubItems(2) = "-1"
End Sub

Private Sub cmdExit_Click()
    Unload Me
End Sub
Private Sub cmdListen_Click()
    Dim itemx As Object
    ' Put the server control into listening mode on the given
    ' port number
    '
    sockServer(0).LocalPort = CInt(txtServerPort.Text)
    sockServer(0).Listen

    Set itemx = lstStates.ListItems.Item(2)
    itemx.SubItems(2) = sockServer(0).LocalPort

    cmdCloseListen.Enabled = True
    cmdListen.Enabled = False
End Sub

Private Sub cmdSendData_Click()
    ' If we're connected, send the given data to the server
    '
    If (sockClient.State = sckConnected) Then
        sockClient.SendData txtSendData.Text
    Else
        MsgBox "Unexpected error! Connection closed"
        Call cmdDisconnect_Click
    End If
End Sub

Private Sub Form_Load()
    Dim itemx As Object

    lblLocalHostname.Caption = sockServer(0).LocalHostName
    lblLocalHostIP.Caption = sockServer(0).LocalIP

    ' Initialize the Protocol property to TCP since that's
    ' all we'll be using
    '
    ServerIndex = 0
```

```
    sockServer(0).Protocol = sckTCPProtocol
    sockClient.Protocol = sckTCPProtocol
    ' Set up the buttons
    '
    cmdDisconnect.Enabled = False
    cmdSendData.Enabled = False
    cmdCloseListen.Enabled = False
    ' Initialize the ListView control that contains the
    ' current state of all Winsock controls created (not
    ' necessarily connected or being used)
    '
    Set itemx = lstStates.ListItems.Add(1, , "Local Client")
    itemx.SubItems(1) = "sckClosed"
    itemx.SubItems(2) = "-1"
    Set itemx = lstStates.ListItems.Add(2, , "Local Server")
    itemx.SubItems(1) = "sckClosed"
    itemx.SubItems(2) = "-1"
    ' Initialize the timer, which controls the rate of refresh
    ' on the above socket states
    '
    Timer1.Interval = 500
    Timer1.Enabled = True
End Sub

Private Sub sockClient_Close()
    sockClient.Close
End Sub

Private Sub sockClient_Connect()
    Dim itemx As Object

    ' The connection was successful: enable the transfer data
    ' buttons
    cmdSendData.Enabled = True
    cmdDisconnect.Enabled = True

    Set itemx = lstStates.ListItems.Item(1)
    itemx.SubItems(2) = sockClient.LocalPort
End Sub

Private Sub sockClient_Error(ByVal Number As Integer, _
        Description As String, ByVal Scode As Long, _
        ByVal Source As String, ByVal HelpFile As String, _
        ByVal HelpContext As Long, CancelDisplay As Boolean)
    ' An error occured on the Client control: print a message,
    ' and close the control. An error puts the control in the
    ' sckError state, which is cleared only when the Close
    ' method is called.
    MsgBox Description
    sockClient.Close
    cmdConnect.Enabled = True
End Sub

Private Sub sockServer_Close(index As Integer)
    Dim itemx As Object
    ' Close the given Winsock control
```

```
    '
    sockServer(index).Close

    Set itemx = lstStates.ListItems.Item(index + 2)
    lstStates.ListItems.Item(index + 2).Text = "---.---.---.---"
    itemx.SubItems(2) = "-1"

End Sub

Private Sub sockServer_ConnectionRequest(index As Integer, _
        ByVal requestID As Long)
    Dim i As Long, place As Long, freeSock As Long, itemx As Object

    ' Search through the array to see wether there is a closed
    ' control that we can reuse
    freeSock = 0
    For i = 1 To ServerIndex
        If sockServer(i).State = sckClosed Then
            freeSock = i
            Exit For
        End If
    Next i
    ' If freeSock is still 0, there are no free controls
    ' so load a new one
    '
    If freeSock = 0 Then
        ServerIndex = ServerIndex + 1
        Load sockServer(ServerIndex)

        sockServer(ServerIndex).Accept requestID
        place = ServerIndex
    Else
        sockServer(freeSock).Accept requestID
        place = freeSock
    End If
    '  If no free controls were found, we added one above.
    '  Create an entry in the ListView control for the new
    '  control.  In either case set the state of the new
    '  connection to sckConnected.
    '
    If freeSock = 0 Then
        Set itemx = lstStates.ListItems.Add(, , _
            sockServer(ServerIndex).RemoteHostIP)
    Else
        Set itemx = lstStates.ListItems.Item(freeSock + 2)
        lstStates.ListItems.Item(freeSock + 2).Text = _
            sockServer(freeSock).RemoteHostIP
    End If
    itemx.SubItems(2) = sockServer(place).RemotePort

End Sub

Private Sub sockServer_DataArrival(index As Integer, _
        ByVal bytesTotal As Long)
    Dim data As String, entry As String

    ' Allocate a large enough string buffer and get the
    ' data
    '
    data = String(bytesTotal + 2, Chr$(0))
    sockServer(index).GetData data, vbString, bytesTotal
    ' Add the client's IP address to the beginning of the
```

```
    ' message and add the message to the list box
    '
    entry = sockServer(index).RemoteHostIP & ": " & data
    lstMessages.AddItem entry
End Sub

Private Sub sockServer_Error(index As Integer, _
        ByVal Number As Integer, Description As String, _
        ByVal Scode As Long, ByVal Source As String, _
        ByVal HelpFile As String, ByVal HelpContext As Long, _
        CancelDisplay As Boolean)
    ' Print the error message and close the specified control.
    ' An error puts the control in the sckError state, which
    ' is cleared only when the Close method is called.
    MsgBox Description
    sockServer(index).Close
End Sub

Private Sub Timer1_Timer()
    Dim i As Long, index As Long, itemx As Object

    ' Set the state of the local client Winsock control
    '
    Set itemx = lstStates.ListItems.Item(1)
    Select Case sockClient.State
        Case sckClosed
            itemx.SubItems(1) = "sckClosed"
        Case sckOpen
            itemx.SubItems(1) = "sckOpen"
        Case sckListening
            itemx.SubItems(1) = "sckListening"
        Case sckConnectionPending
            itemx.SubItems(1) = "sckConnectionPending"
        Case sckResolvingHost
            itemx.SubItems(1) = "sckResolvingHost"
        Case sckHostResolved
            itemx.SubItems(1) = "sckHostResolved"
        Case sckConnecting
            itemx.SubItems(1) = "sckConnecting"
        Case sckConnected
            itemx.SubItems(1) = "sckConnected"
        Case sckClosing
            itemx.SubItems(1) = "sckClosing"
        Case sckError
            itemx.SubItems(1) = "sckError"
        Case Else
            itemx.SubItems(1) = "unknown: " & sockClient.State
    End Select
        ' Now set the states for the listening server control as
        ' well as any connected clients
        '
        index = 0
        For i = 2 To ServerIndex + 2
            Set itemx = lstStates.ListItems.Item(i)

            Select Case sockServer(index).State
                Case sckClosed
                    itemx.SubItems(1) = "sckClosed"
                Case sckOpen
                    itemx.SubItems(1) = "sckOpen"
                Case sckListening
```

```
                itemx.SubItems(1) = "sckListening"
            Case sckConnectionPending
                itemx.SubItems(1) = "sckConnectionPending"
            Case sckResolvingHost
                itemx.SubItems(1) = "sckResolvingHost"
            Case sckHostResolved
                itemx.SubItems(1) = "sckHostResolved"
            Case sckConnecting
                itemx.SubItems(1) = "sckConnecting"
            Case sckConnected
                itemx.SubItems(1) = "sckConnected"
            Case sckClosing
                itemx.SubItems(1) = "sckClosing"
            Case sckError
                itemx.SubItems(1) = "sckError"
            Case Else
                itemx.SubItems(1) = "unknown"
        End Select
        index = index + 1
    Next i
End Sub
```

15.5.1 TCP服务器

从现在开始，我们打算为大家讲解窗体后面的代码。先来看看程序清单 15-2显示的Form_Load例程。头两条语句的作用很简单，将两个标签分别设为本地机器的主机名和IP地址。这些标签显示于Winsock信息分组框内，它们的作用和UDP示例中的信息框完全相同。接下来，我们将服务器控件sockServer初始化成TCP协议。Winsock控件数组的元素0肯定对应于监听套接字。在这以后，进程会先将 Close Listen（关闭监听）按钮禁用。稍后，等服务器开始监听客户机连接时，该按钮又会启用。这个进程的最后部分负责 ListView控件（lstStates）的设置。该控件用于显示当前使用的每一个 Winsock控件的最新状态。代码会为客户机和服务器控件添加对应的条目，令其分别成为数组元素 1和2。以后假如动态载入了其他任何 Winsock控件，都会添加到这两个元素之后。与服务器控件对应的条目名为“ Local Server ”（本地服务器）。和在UDP例子中一样，进程也会设置一个计时器，对套接字状态的更新频率进行管理。在默认情况下，计时器每隔半秒钟，便刷新一道显示。

现在，让我们来看看服务器使用的两个按钮。第一个是 Listen（监听）按钮，它的用法非常简单。Listen按钮的控制模块 cmdListen负责将LocalPort（本地端口）属性设为用户在txtServerPort文本框内输入的值。对监听套接字来说，本地端口是最重要的一项设定。所有客户机都会通过这个端口尝试建立与服务器的连接。设好 LocalPort属性后，代码下一步要做的唯一的一件事情是调用 Listen方法。Listen按钮的控制模块将sockServer控件置于监听模式后，程序会开始等待在我们的sockServer控件上发生ConnectionRequest事件 —— 表明有一个客户机请求连接。用户可点击另一个按钮（Close Listen），以关闭sockServer控件。Close Listen按钮的控制模块会针对sockServer(0)调用Close方法，防止以后再进行任何客户机连接。

对TCP服务器来说，最重要的事件是 ConnectionRequest，它负责对进入的客户机请求的控制。若客户机请求建立一个连接，那么可选择两种方法来处理这一请求。第一种方法，可用服务器套接字本身来控制客户机。但这样做也有缺点，因为它会关闭监听套接字，并禁止

再为其他任何连接提供服务。要想采用这种方法，只需针对拥有特定 requestID的服务器控件调用Accept方法。那个requestID早已传递到事件控制模块内。控制客户机连接请求的另一个方法是将连接传递给某个单独的控件，这正是我们的示范 SockTCP程序所采用的。要记住的是，我们有一个由Winsock控件构成的数组，而且元素 0对应于监听套接字。要做的第一件事是搜索整个数组，在其中寻找状态为“ 关闭 ”的控件。举个例子来说，可搜索值为 sckClosed的State属性。当然，在第一次循环中，不可能找到空闲的控件，因为它们尚未载入。在这种情况下，大家会发现第一次循环根本没有执行。同时，变量 freeSock仍然为0，表明没有找到空闲的控件。接下来的步骤是动态载入一个 Winsock控件。具体做法是递增 ServerIndex计数器的值（对应于载入控件的数组位置），然后执行下述语句：

```
Load sockServer(ServerIndex)
```

现在，一个新的Winsock控件已经载入了。接下来，进程可随同指定的请求ID（requestID），调用Accept方法。剩下的语句会在lstStates这个ListView（列表视图）控件中加入一个新条目，使程序能够显示出新Winsock控件的最新状态。

假如已经存在一个加载的 Winsock控件，其状态已为“ 关闭 ”，那么进程只需简单地重新使用那个控件。具体说来，就是在它上面调用 Accept方法。经常加载和卸载控件并不是一种好的编程习惯，因为它会对性能产生一定的影响。某些时候，加载和卸载操作会让你付出惨重代价。在Winsock控件卸载时，还存在一个“ 内存渗漏 ”的问题。本章稍后将就此进行详述。

剩下的服务器函数非常简单。只要有客户机在自己那一端调用 Close方法，就会触发sockServer_Close事件。此时，服务器要做的事情只是关闭自己这一端的套接字，同时清除ListView控件内的 IP地址条目（将其设为“ ---.---.---.--- ”，并将条目的端口设为- 1）。sockServer_DataArrival函数会分配一个缓冲区，以便接收数据。随后，调用 GetData方法，来实际执行读操作。之后，消息会添加到 lstMessages列表框内。最后一个服务器函数是 Error事件控制模块。一旦发生错误，该控制模块就会显示出文本消息，随后关闭控件。

15.5.2 TCP客户机

目前为止，大家已知道了怎样实现服务器。接下来，看看客户机的情况。在 Form_Load进程中，客户机执行的唯一初始化操作是将 sockClient的协议设为TCP。除初始化代码之外，还有三个从属于客户机的命令按钮控制模块，以及几个事件控制模块。其中，第一个按钮是Connect（连接），它的控制模块名为cmdConnect_Click。LocalPort设为0，因为无论本地系统分配哪一个端口编号，都不重要。 RemoteHost和RemotePort则分别根据txtServerName和txtPort字段中的值来设定。这些便是与服务器建立 TCP连接所需的所有信息。万事俱备，剩下的事便是调用 Connect方法。调用这个方法后，控件状态便是“ 正在解析名字 ”或“ 连接 ”（控件的状态是sckResolvingHost、sckResolved或sckConnecting）。终于建立连接之后，状态变成“ sckConnected ”（已连接），并触发Connect（连接）事件。下一小节将全面讲解各种连接状态以及各状态之间的转换。

连接一旦建立，便调用 sockClient_Connect句柄。这个句柄只是启用了“ SendData ”（发送数据）和“ Disconnect ”（取消连接）两个按钮。除此以外，本地机器上建立了连接的那个端口编号已针对lstStates ListView控件中的“ 本地客户机条目 ”而得以更新。现在便可收发数

据。另外还有两个事件句柄：sockClient_Close和sockClient_Error。sockClient_Close事件句柄只是关闭客户机Winsock控件，而sockClient_Error事件句柄则显示一个具有错误说明的消息框，之后关闭这个控件。

关于客户机，最后需要提到的是这两个命令按钮： Send Data（发送数据）和Disconnect（取消连接）。子例程cmdSendData_Click对Send Data（发送数据）按钮进行控制。如果已连接了Winsock控件，这个例程便随txtSendData文本框内的字串一起，调用SendData方法。最后，Disconnect（取消连接）按钮由cmdDisconnect_Click控制。这个句柄只关闭客户机控件，重新把一系列按钮设置成初始状态，并更新lstStates ListView控件中的Local Client（本地客户机）条目。

15.5.3 获取Winsock信息

TCP示例的最后部分是Winsock信息。我们在以前曾对此进行了解释，但为了使大家心里更清楚，还需要简明扼要地讲一讲。在UDP示例中，对所有已加载Winsock控件来说，当前套接字状态的更新是计时器触发的。默认刷新时间是500毫秒。加载之后的lstStates ListView控件中增加了两个条目。第一个条目是Local Client（本地客户机），它对应于客户机Winsock控件sockClient。第二个条目是LocalServer（本地服务器），它引用正在监听的套接字。只要建立了新的客户机连接，就会动态加载一个新的Winsock控件，sckStates控件中也会增加一个新条目；条目名就是该客户机的IP地址。客户机取消连接时，把IP地址设为“ ---.---.---.--- ”，端口号设为-1，便将条目设为默认状态。当然，如果有另一个客户机连接时，便重新使用服务器数组中没有用过的控件。同时显示本地机器的IP地址和主机名。

15.5.4 运行TCP示例

运行TCP示例非常简单。启动三个应用程序实例，一台机器一个。利用TCP时，机器是否是多址机这一点无关紧要，因为路由表会判断哪个接口更适合这个具体的 TCP连接。其中一个TCP示例中，单击Listen（监听）按钮，启动监听套接字。大家将注意到 State Information ListView（状态信息列表视图）中的Local Server（本地服务器）条目从sckClosed变成了sckListening，列出的端口号是5150。现在起，服务器准备接受客户机连接。

其中一个客户机上，先把Server Name（服务器名）字段设为运行第一个应用程序实例的那台计算机（监听服务器）的名字，再单击Connect（连接）按钮。客户机应用程序上， State Information（状态信息列表）中的Local Client（本地客户机）条目现在处于sckConnected状态，建立连接的本地端口之端口号更新为一个“非负值”。另外，服务器这一端，状态信息列表中增加了一个条目，该列表名为刚连接的客户机的IP地址。新增条目的状态是sckConnected，其中还包含了服务器这一端上已建立连接的那个端口的编号。现在，可以向客户机上的Message（消息）文本域内输入文字，并单击几次Send Data（发送数据）按钮。大家将看到出现在服务器端的Message（消息）文本域内的消息。接下来单击 Disconnect按钮取消这个客户机连接。在客户机端， State Information（状态信息）列表框内的Local Client（本地客户机）条目设为原来的sckClosed，端口号设为-1。对服务器来说，与客户机对应的那个条目没有被删除；只用这样的方式来表示未用：一个名字设为虚线 IP地址，状态设为sckClosed，端口号值设为-1。

第三台机器上，在Server Name（服务器名）文本框内输入监听套接字的名字，建立一个客户机连接。其结果与第一个客户机一样，只不过这个服务器用同样的 Winsock控件，像第一台机器上那样，对这个客户机进行处理。如果 Winsock控件处于已关闭状态，就可用于接受任何一个接入连接。最后一步是利用服务器应用程序上的客户机建立本地连接。连接建立之后，Socket Information（套接字信息）列表中便增加一个新条目，这和前面的示例一样。唯一的区别是：这时列出的 IP和服务器的 IP地址一样。实战多客户机和一个服务器的确能深入了解它们之间是如何交互的，以其相关命令会触发什么样的结果。

15.5.5 TCP状态

使用TCP协议的Windows控件比UDP套接字复杂得多，因为TCP涉及到的套接字状态更多。图15-4是一个TCP套接字的状态图。默认的起始状态是 sckClosed。各个状态之间的转换非常直接，无需过多解释，但 sckClosing状态除外。由于TCP的半关闭特性，在使用 SendData方法时，便有两条转换途径。 TCP连接的一端执行一个 Close（关闭）方法，不能再发送数据了。连接的另一端收到了这个 Close事件，置入 sckClosing状态，但仍然可以接收数据。这就是SendData方法的sckClosing状态转换有两条途径的原因。如果执行 Close方法的这端试图调用SendData，就会产生错误，并且状态转为 sckError。收到Close事件的一端可自由发送数据和接收其余的数据。

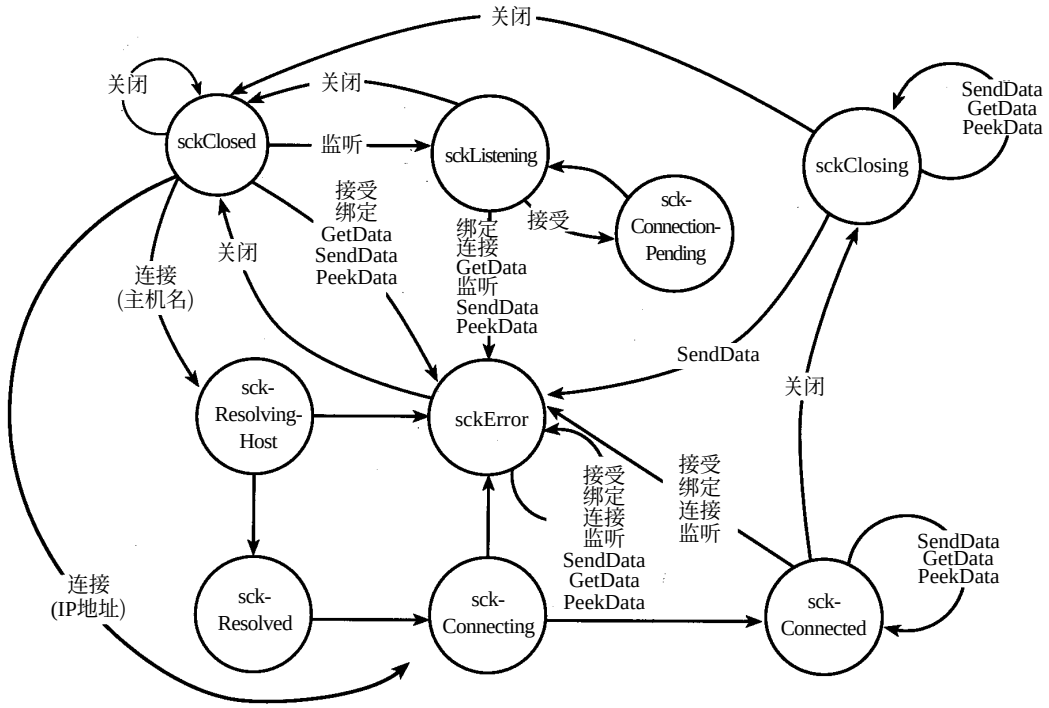

图15-4 TCP状态

15.6 存在的局限

Winsock控件用处颇为广泛，而且易于使用；但不幸的是，还存在几个错误，令控件不能

使用执行关键任务的应用程序。本小节中讨论的错误出现在最新版本的 Visual Basic 5.0控件，这个控件是在SP 2基础上更新的。

第一个错误问题不大，涉及到动态加载和卸载控件。卸载一个以前加载的控件时，会导致内存渗漏。这就是我们在服务器示例中进行客户机连接和取消连接时，没有加载和卸载控件的原因。控件一旦载入内存，我们会尽可能地不卸载它，留给另外的客户机使用。

第二个错误是在将队列中的所有数据发送进入网络之前，关闭套接字连接造成的。某些情况下，若在SendData事件之后调用Close方法（Close的处理先于SendData时），会导致数据的丢失，至少从接收者的角度来看时如此。通过捕捉 SendComplete事件（SendData完全将数据送入网络之后触发的）的方式，便可避免这个问题。另外一个解决办法是，可通过对发送／接收事务处理的特殊安排，使接收者在收到自己希望的所有数据时，能够先一步执行Close命令。随后便会触发发送者上的Close事件，后者随之通知自己发出的所有数据均已被对方收到。现在，可以非常安全地将连接关闭。

最后一个、同时也是最严重的错误是：提交一个大缓冲区时造成数据的丢失。假如一个足够大的数据块正在队列之中，等候进入网络传输，那么控件的内部缓冲区有可能变得一团糟，造成部分数据的丢失。不幸的是，没有一种完美的方案可有效解决这个问题。最好的办法是每次提交不超过1000字节的数据块。提交了一个缓冲区之后，便等待 SendComplete（发送完成）事件的发生，再提交下一个缓冲区。尽管这样做十分不便，但确实是提高控件可靠性的最佳方式。

随Visual Basic 6.0发布的最新Winsock控件修正了上述除第二个之外的所有问题。如在调用SendData后马上执行一个Close命令，那么套接字会立即关闭，根本不会等待全部数据都发送完毕。当然，如果能将所有问题都解决，便显得更加完美了。但是，剩下的这个问题是三个问题中最不严重的，而且最容易解决。

15.7 常见错误

通过第6到第14章的学习，大家已经知道，一个应用程序可能会经历大量 Winsock错误。在此，我们不打算将所有问题都总结一遍。在接下去的两个小节中，我们只准备介绍使用Winsock控件的时候，最常遇到的两个错误是“ 本地地址正在使用 ”以及“ 当前状态下的无效操作 ”。

1. 本地地址正在使用

“ 本地地址正在使用 ”错误是在同一个本地端口绑定时发生的。要想同本地端口建立绑定关系，要么可以使用 Bind方法，要么可以使用 Connect方法。它表明要绑定的那个端口正在使用。这种现象通常见于 TCP服务器中，那些服务器一直同一个本地端口绑定在一起，使客户机能够随时定位（找到）服务。假定在应用程序使用一个套接字之前，那个套接字被不正确地关闭，它就会短时间内处于TIME_WAIT状态，以确保所有数据都在那个端口上完成了收发。在这期间，若试图建立同那个端口的绑定，便会产生所谓的“ 本地地址正在使用 ”错误。另外，在客户机那一端，一个常邮的误操作也会造成这样的错误。假若将 LocalPort属性的值设为0，然后建立了一个连接，那么 LocalPort会自动更新为用于建立本地客户机连接的那个端口号。若计划重新使用相同的控件，来进行后续的连接，那么一定要记住重设 LocalPort属性，将其变成0。否则的话，一旦以前的连接没有正确关闭，便会遇到这样的错误。

2. 当前状态下的无效操作

“当前状态下的无效操作”（Invalid operation at current state）可能是最常见的一种错误。若调用一个Winsock控件方法，但取决于控件的当前状态，却要求禁止那样的操作，便会产生这样的错误。大家可参考一下图15-2和图15-4，观察UDP和TCP套接字的结构。要想写出真正“健壮”的代码，在调用一个方法之前，一定记住先检查一下套接字的状态。

Winsock错误会通过Error事件生成。这些错误等同于直接用Winsock编程时遇到的那些错误。要想了解Winsock错误的详情，请参考第7章，其中讲述了最常见的一系列错误；亦可参考附录C，那里列出了可能出现的所有Winsock错误代码。

15.8 Windows CE的Winsock控件

在Visual Basic Toolkit for Windows CE（VBCE）中，包含了一个Winsock控件，提供了与普通Visual Basic Winsock控件差不多的功能。两者最主要的差别在于，尽管UDP未获支持，但Windows CE的Winsock控件提供了IrDA（红外线通信联盟）协议。此外，由于两者存在的一些细微差异，促使我们在编程的时候，同非Windows CE的Winsock控件编程相比，需要作出一些变化。

通过第7章的学习，大家知道Windows CE没有提供异步Winsock模型。在此，Windows CE的Winsock控件也不例外。就编程来说，最主要的区别在于Connect方法工作于“锁定”模式。此时不再有Connect事件。一旦调用Connect，试图建立一个连接，调用便会被暂时挂起，直至建好连接，或者返回一个错误。

除此以外，VBCE 1.0没有提供对控件数组的支持。换言之，我们必须更改在程序清单15-2中显示的服务器设计。因此，要想同时控制多个连接，唯一的方法便是在窗体中置入大量Windows CE Winsock控件。在实际应用中，这样做便限制了能够同时建立的客户机连接的数量，造成整个应用程序的扩展能力大打折扣。

最后，ConnectionRequest事件不再有RequestID参数，这让人感觉似乎非常奇怪。为此，我们必须在控件上调用Accept方法。触发ConnectionRequest事件的连接请求是由负责接收连接请求的控件加以满足的。

15.8.1 Windows CE Winsock示例

本节介绍一个示范应用程序，它运用了Windows CEWinsock控件。除前面提到的区别之外，桌面机Winsock控件的设计原理同样适用于Windows CE控件的设计。在程序清单15-3中，我们展示了Windows CE Winsock控件背后的一系列代码。

程序清单15-3 Windows CE Winsock示例

```
Option Explicit

' This global variable is used to retain the current value
' of the radio buttons.  0 corresponds to TCP, while 2 means
' IrDA (infrared).  Note that UDP is not supported by the
' control currently.
Public SocketType

Private Sub cmdCloseListen_Click()
' Close the listening socket, and set the other buttons
```

```
' back to the start state
    WinSock1.Close

    cmdConnect.Enabled = True
    cmdListen.Enabled = True
    cmdDisconnect.Enabled = False
    cmdSendData.Enabled = False
    cmdCloseListen.Enabled = False
End Sub

Private Sub cmdConnect_Click()
' Check which type of socket type was chosen, and initiate
' the given connection

    If SocketType = 0 Then
        ' Set the protocol and the remote host name and port
        '
        WinSock1.Protocol = 0
        WinSock1.RemoteHost = txtServerName.Text
        WinSock1.RemotePort = CInt(txtPort.Text)
        WinSock1.LocalPort = 0

        WinSock1.Connect
    ElseIf SocketType = 2 Then
        ' Set the protocol to IrDA, and set the service name
        '
        WinSock1.Protocol = 2
        'WinSock1.LocalPort = 0
        'WinSock1.ServiceName = txtServerName.Text
        WinSock1.RemoteHost = txtServerName.Text

        WinSock1.Connect
    End If
    ' Make sure the connection was successful; if so,
    ' enable/disable some commands
    '
    MsgBox WinSock1.State
    If (WinSock1.State = 7) Then
        cmdConnect.Enabled = False
        cmdListen.Enabled = False
        cmdDisconnect.Enabled = True
        cmdSendData.Enabled = True
    Else
        MsgBox "Connect failed"
        WinSock1.Close
    End If
End Sub

Private Sub cmdDisconnect_Click()
' Close the current client connection, and reset the
' buttons to the start state
    WinSock1.Close

    cmdConnect.Enabled = True
    cmdListen.Enabled = True
```

```
    cmdDisconnect.Enabled = False
    cmdSendData.Enabled = False
    cmdCloseListen.Enabled = False
End Sub

Private Sub cmdListen_Click()
' Set the socket to listening mode for the given protocol
' type
'
    If SocketType = 0 Then
        WinSock1.Protocol = 0
        WinSock1.LocalPort = CInt(txtLocalPort.Text)
        WinSock1.Listen
    ElseIf SocketType = 2 Then
        WinSock1.Protocol = 2
        WinSock1.ServiceName = txtServerName.Text
        WinSock1.Listen
    End If
    ' If we're not in listening mode now, something
    ' went wrong
    '
    If (WinSock1.State = 2) Then
        cmdConnect.Enabled = False
        cmdListen.Enabled = False
        cmdCloseListen.Enabled = True
    Else
        MsgBox "Unable to listen!"
    End If
End Sub

Private Sub cmdSendData_Click()
' Send the data in the box on the current connection
'
    WinSock1.SendData txtSendData.Text
End Sub

Private Sub Form_Load()
' Set the initial values for the buttons, the timer, etc.
'
    optTCP.Value = True
    SocketType = 0

    Timer1.Interval = 750
    Timer1.Enabled = True

    cmdConnect.Enabled = True
    cmdListen.Enabled = True

    cmdDisconnect.Enabled = False
    cmdSendData.Enabled = False
    cmdCloseListen.Enabled = False

    lblLocalIP.Caption = WinSock1.LocalIP
End Sub
```

```
Private Sub optIRDA_Click()
' Set the socket type to IrDA
'
    optIRDA.Value = True
    SocketType = 2
End Sub

Private Sub optTCP_Click()
' Set the socket type to TCP
'
    optTCP.Value = True
    SocketType = 0
    cmdConnect.Caption = "Connect"
End Sub

Private Sub Timer1_Timer()
' This is the event that gets called each time the
' timer expires. Update the socket state label.
'
    Select Case WinSock1.State
        Case 0
            lblState.Caption = "sckClosed"
        Case 1
            lblState.Caption = "sckOpen"
        Case 2
            lblState.Caption = "sckListening"
        Case 3
            lblState.Caption = "sckConnectionPending"
        Case 4
            lblState.Caption = "sckResolvingHost"
        Case 5
            lblState.Caption = "sckHostResolved"
        Case 6
            lblState.Caption = "sckConnecting"
        Case 7
            lblState.Caption = "sckConnected"
        Case 8
            lblState.Caption = "sckClosing"
        Case 9
            lblState.Caption = "sckError"
    End Select
End Sub

Private Sub WinSock1_Close()
' The other side initiated a close, so we'll close our end.
' Reset the buttons to their initial state.
'
    WinSock1.Close

    cmdConnect.Enabled = True
    cmdListen.Enabled = True
    cmdDisconnect.Enabled = False
    cmdSendData.Enabled = False
    cmdCloseListen.Enabled = False
End Sub
```

```
Private Sub WinSock1_ConnectionRequest()
' We got a client connection; accept it on the listening
' socket
'
    WinSock1.Accept
End Sub

Private Sub WinSock1_DataArrival(ByVal bytesTotal)
' This is the event for data arrival.  Get the data, and
' add it to the list box.
'
    Dim rdata

    WinSock1.GetData rdata
    List1.AddItem rdata
End Sub

Private Sub WinSock1_Error(ByVal number, ByVal description)
' An error occurred; display the message, and close the socket
'
    MsgBox description
    Call WinSock1_Close
End Sub
```

在此，我们不打算就程序清单 15-3这个示范程序的规格进行详细说明，因为它其实和程序清单15-2展示的那个SockTCP示例是类似的。唯一的区别在于前一节提到的那些限制。在此要注意的是，Windows CE Winsock控件只是一种比较“初级”的控件。换句话说，它的设计并不像桌面版本那么完善。类型库并没有完全实现：必须通过一个整数，对协议类型加以区分；而不能对类型进行枚举。除此以外，就像下一节“已知的问题”要讲述的那样，套接字状态枚举类型也存在着问题。

红外线连接的控制其实并不难，和 TCP连接差不多。但在通过红外线端口建立一个监听套接字时，却会出现一个例外。对一个红外线服务器来说，我们用“服务名”对其进行引用，后者已在第6章的IrDA定址小节进行了详细解释。Windows CE Winsock控件新增了一个属性，名为ServiceName。我们可将该属性设为客户机试图连接的下一个服务名文本字串。举个例子来说，下述代码片断可将名为 CeWinsock的Windows CE Winsock控件置入监听模式，并赋予“MyServer”的名字：

```
CeWinsock.Protocol = 2              ' Protocol 2 is IrSock
CeWinsock.ServiceName = "MyServer"
CeWinsock.Listen
```

要想通过红外线套接字发布一项服务，除此以外便无其他特殊要求，只需指定服务名即可。

15.8.2 已知的问题

使用VBCE Winsock控件时，我们在为Winsock状态使用列举的值时，发现了一个奇怪的问题。不知由于什么原因，这些值是在开发环境中定义的，但每次在代码中引用一个 sck枚举值时，在远程设备上，都会弹出下述错误提示消息：“运行这个程序时遇到错误”（An error was encountered while running this program）。若将枚举换成各自对应的常数值，这种错误便会消失。我们认为这是VBCE的一个错误，并希望在未来版本的工具包中，能得以纠正。

15.9 小结

Visual Basic Winsock控件非常适用于简单的、不重要的网络应用程序。 Winsock控件的Visual Basic 5.0版本中存在着几个问题，造成难以成功地进行控件编程，但主要问题都在最新发布的Visual Basic版本中得到了解决。有了这个控件，便可在自己的 Visual Basic应用程序中自由增添网络通信能力。当然，它并不是万能的，应用程序如果需要与 Winsock进行大量沟通，便应考虑从Winsock DLL中手工导入必要的函数和常数。就像我们在前面提到的那样，在本书第二部分提供的各个Winsock Visual Basic示例中，实际都从Ws2_32.dll中导入了Winsock函数。例如，大家可参考本书配套光盘 Chapter07\VB目录下的SimpleTCP与SimpleUDP程序，以及它们对应的Winsock.bas文件。