Visual C#.Net 网络程序

（一）Visual C#.Net 网络程序开发 Socket 篇

Microsoft.Net Framework 为应用程序访问 Internet 提供了分层的、可扩展的以及受管辖的网络服务，其名字空间 System.Net 和 System.Net.Sockets 包含丰富的类可以开发多种网络应用程序。.Net 类采用的分层结构允许应用程序在不同的控制级别上访问网络，开发人员可以根据需要选择针对不同的级别编制程序，这些级别几乎囊括了 Internet 的所有需要--从 socket 套接字到普通的请求/响应，更重要的是，这种分层是可以扩展的，能够适应 Internet 不断扩展的需要。

抛开 ISO/OSI 模型的 7 层构架，单从 TCP/IP 模型上的逻辑层面上看，.Net 类可以视为包含 3 个层次：请求/响应层、应用协议层、传输层。WebReqeust 和 WebResponse 代表了请求/响应层，支持 Http、Tcp 和 Udp 的类组成了应用协议层，而 Socket 类处于传输层。可以如下示意：

可见，传输层位于这个结构的最底层，当其上面的应用协议层和请求/响应层不能满足应用程序的特殊需要时，就需要使用这一层进行 Socket 套接字编程。

而在.Net 中，System.Net.Sockets 命名空间为需要严密控制网络访问的开发人员提供了 Windows Sockets (Winsock) 接口的托管实现。System.Net 命名空间中的所有其他网络访问类都建立在该套接字 Socket 实现之上，如 TCPClient、TCPListener 和 UDPClient 类封装有关创建到 Internet 的 TCP 和 UDP 连接的详细信息；NetworkStream 类则提供用于网络访问的基础数据流等，常见的许多 Internet 服务都可以见到 Socket 的踪影，如 Telnet、Http、Email、Echo 等，这些服务尽管通讯协议 Protocol 的定义不同，但是其基础的传输都是采用的 Socket。

其实，Socket 可以象流 Stream 一样被视为一个数据通道，这个通道架设在应用程序端（客户端）和远程服务器端之间，而后，数据的读取（接收）和写入（发送）均针对这个通道来进行。

可见，在应用程序端或者服务器端创建了 Socket 对象之后，就可以使用 Send/SentTo 方法将数据发送到连接的 Socket,或者使用 Receive/ReceiveFrom 方法接收来自连接 Socket 的数据；

针对 Socket 编程，.NET 框架的 Socket 类是 Winsock32 API 提供的套接字服务的托管代码版本。其中为实现网络编程提供了大量的方法，大多数情况下，Socket 类方法只是将数据封送到它们的本机 Win32 副本中并处理任何必要的安全检查。如果你熟悉 Winsock API 函数，那么用 Socket 类编写网络程序会非常容易，当然，如果你不曾接触过，也不会太困难，跟随下面的解说，你会发觉使用 Socket 类开发 windows 网络应用程序原来有规可寻，它们在大多数情况下遵循大致相同的步骤。

在使用之前，你需要首先创建 Socket 对象的实例，这可以通过 Socket 类的构造方法来实现：

```
public Socket(AddressFamily addressFamily,SocketType socketType,ProtocolType protocolType);
```

其中，addressFamily 参数指定 Socket 使用的寻址方案，socketType 参数指定 Socket 的类型，protocolType 参数指定 Socket 使用的协议。

下面的示例语句创建一个 Socket，它可用于在基于 TCP/IP 的网络（如 Internet）上通讯。

```
Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

若要使用 UDP 而不是 TCP，需要更改协议类型，如下面的示例所示：

```
Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
```

一旦创建 Socket，在客户端，你将可以通过 Connect 方法连接到指定的服务器，并通过 Send/SendTo 方法向远程服务器发送数据，而后可以通过 Receive/ReceiveFrom 从服务端接收数据；而在服务器端，你需要使用 Bind 方法绑定所指定的接口使 Socket 与一个本地终结点相联，并通过 Listen 方法侦听该接口上的请求，当侦听到用户端的连接时，调用 Accept 完成连接的操作，创建新的 Socket 以处理传入的连接请求。使用完 Socket 后，记住使用 Shutdown 方法禁用 Socket，并使用 Close 方法关闭 Socket。其间用到的方法/函数有：

Socket.Connect 方法:建立到远程设备的连接
public void Connect(EndPoint remoteEP)（有重载方法）
Socket.Send 方法:从数据中的指示位置开始将数据发送到连接的 Socket。
public int Send(byte[], int, SocketFlags);(有重载方法)
Socket.SendTo 方法 将数据发送到特定终结点。
public int SendTo(byte[], EndPoint);（有重载方法）
Socket.Receive 方法:将数据从连接的 Socket 接收到接收缓冲区的特定位置。
public int Receive(byte[],int,SocketFlags);
Socket.ReceiveFrom 方法：接收数据缓冲区中特定位置的数据并存储终结点。
public int ReceiveFrom(byte[], int, SocketFlags, ref EndPoint);
Socket.Bind 方法：使 Socket 与一个本地终结点相关联：
public void Bind(EndPoint localEP);
Socket.Listen 方法：将 Socket 置于侦听状态。
public void Listen(int backlog);
Socket.Accept 方法:创建新的 Socket 以处理传入的连接请求。
public Socket Accept();
Socket.Shutdown 方法:禁用某 Socket 上的发送和接收
public void Shutdown(SocketShutdown how);
Socket.Close 方法:强制 Socket 连接关闭
public void Close();

可以看出，以上许多方法包含 EndPoint 类型的参数，在 Internet 中，TCP/IP 使用一个网络地址和一个服务端口号来唯一标识设备。网络地址标识网络上的特定设备；端口号标识要连接到的该设备上的特定服务。网络地址和服务端口的组合称为终结点，在 .NET 框架中正是由 EndPoint 类表示这个终结点，它提供表示网络资源或服务的抽象，用以标志网络地址等信息。.Net 同时也为每个受支持的地址族定义了 EndPoint 的子代；对于 IP 地址族，该类为 IPEndPoint。IPEndPoint 类包含应用程序连接到主机上的服务所需的

主机和端口信息，通过组合服务的主机 IP 地址和端口号，IPEndPoint 类形成到服务的连接点。

用到 IPEndPoint 类的时候就不可避免地涉及到计算机 IP 地址，.Net 中有两种类可以得到 IP 地址实例：

IPAddress 类：IPAddress 类包含计算机在 IP 网络上的地址。其 Parse 方法可将 IP 地址字符串转换为 IPAddress 实例。下面的语句创建一个 IPAddress 实例：

IPAddress myIP = IPAddress.Parse("192.168.1.2");

Dns 类：向使用 TCP/IP Internet 服务的应用程序提供域名服务。其 Resolve 方法查询 DNS 服务器以将用户友好的域名（如"host.contoso.com"）映射到数字形式的 Internet 地址（如 192.168.1.1）。Resolve 方法 返回一个 IPHostEnty 实例，该实例包含所请求名称的地址和别名的列表。大多数情况下，可以使用 AddressList 数组中返回的第一个地址。下面的代码获取一个 IPAddress 实例，该实例包含服务器 host.contoso.com 的 IP 地址。

IPHostEntry ipHostInfo = Dns.Resolve("host.contoso.com");
IPAddress ipAddress = ipHostInfo.AddressList[0];

你也可以使用 GetHostName 方法得到 IPHostEntry 实例：

IPHosntEntry hostInfo=Dns.GetHostByName("host.contoso.com")

在使用以上方法时，你将可能需要处理以下几种异常：

SocketException 异常：访问 Socket 时操作系统发生错误引发

ArgumentNullException 异常：参数为空引用引发

ObjectDisposedException 异常：Socket 已经关闭引发

在掌握上面得知识后，下面的代码将该服务器主机（ host.contoso.com 的 IP 地址与端口号组合，以便为连接创建远程终结点：

IPEndPoint ipe = new IPEndPoint(ipAddress,11000);

确定了远程设备的地址并选择了用于连接的端口后，应用程序可以尝试建立与远程设备的连接。下面的示例使用现有的 IPEndPoint 实例与远程设备连接，并捕获可能引发的异常：

try {
s.Connect(ipe);//尝试连接

```
}
//处理参数为空引用异常
catch(ArgumentNullException ae) {
Console.WriteLine("ArgumentNullException : {0}", ae.ToString());
}
//处理操作系统异常
catch(SocketException se) {
Console.WriteLine("SocketException : {0}", se.ToString());
}
catch(Exception e) {
Console.WriteLine("Unexpected exception : {0}", e.ToString());
}
```

需要知道的是：Socket 类支持两种基本模式：同步和异步。其区别在于：在同步模式中，对执行网络操作的函数（如 Send 和 Receive）的调用一直等到操作完成后才将控制返回给调用程序。在异步模式中，这些调用立即返回。

另外，很多时候，Socket 编程视情况不同需要在客户端和服务器端分别予以实现，在客户端编制应用程序向服务端指定端口发送请求，同时编制服务端应用程序处理该请求，这个过程在上面的阐述中已经提及；当然，并非所有的 Socket 编程都需要你严格编写这两端程序；视应用情况不同，你可以在客户端构造出请求字符串，服务器相应端口捕获这个请求，交由其公用服务程序进行处理。以下事例语句中的字符串就向远程主机提出页面请求：

```
string Get = "GET / HTTP/1.1\r\nHost: " + server + "\r\nConnection: Close\r\n\r\n";
```

远程主机指定端口接受到这一请求后，就可利用其公用服务程序进行处理而不需要另行编制服务器端应用程序。

综合运用以上阐述的使用 Visual C#进行 Socket 网络程序开发的知识，下面的程序段完整地实现了 Web 页面下载功能。用户只需在窗体上输入远程主机名（Dns 主机名或以点分隔的四部分表示法格式的 IP 地址）和预保存的本地文件名，并利用专门提供 Http 服务的 80 端口，就可以获取远程主机页面并保存在本地机指定文件中。如果保存格式是.htm 格式，你就可以在 Internet 浏览器中打开该页面。适当添加代码，你甚至可以实现一个简单的浏览器程序。

实现此功能的主要源代码如下：

```
//"开始"按钮事件
private void button1_Click(object sender, System.EventArgs e) {
//取得预保存的文件名
string fileName=textBox3.Text.Trim();
//远程主机
```

```
string hostName=textBox1.Text.Trim();
//端口
int port=Int32.Parse(textBox2.Text.Trim());
//得到主机信息
IPHostEntry ipInfo=Dns.GetHostByName(hostName);
//取得 IPAddress[]
IPAddress[] ipAddr=ipInfo.AddressList;
//得到 ip
IPAddress ip=ipAddr[0];
//组合出远程终结点
IPEndPoint hostEP=new IPEndPoint(ip,port);
//创建 Socket 实例
Socket socket=new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
try
{
//尝试连接
socket.Connect(hostEP);
}
catch(Exception se)
{
MessageBox.Show("连接错误"+se.Message,"提示信息
,MessageBoxButtons.RetryCancel,MessageBoxIcon.Information);
}
//发送给远程主机的请求内容串
string sendStr="GET / HTTP/1.1\r\nHost: " + hostName +
"\r\nConnection: Close\r\n\r\n";
//创建 bytes 字节数组以转换发送串
byte[] bytesSendStr=new byte[1024];
//将发送内容字符串转换成字节 byte 数组
bytesSendStr=Encoding.ASCII.GetBytes(sendStr);
try
{
//向主机发送请求
socket.Send(bytesSendStr,bytesSendStr.Length,0);
}
catch(Exception ce)
{
MessageBox.Show("发送错误:"+ce.Message,"提示信息
,MessageBoxButtons.RetryCancel,MessageBoxIcon.Information);
}
//声明接收返回内容的字符串
string recvStr="";
//声明字节数组，一次接收数据的长度为 1024 字节
byte[] recvBytes=new byte[1024];
```

```
//返回实际接收内容的字节数
int bytes=0;
//循环读取，直到接收完所有数据
while(true)
{
bytes=socket.Receive(recvBytes,recvBytes.Length,0);
//读取完成后退出循环
if(bytes<=0)
break;
//将读取的字节数转换为字符串
recvStr+=Encoding.ASCII.GetString(recvBytes,0,bytes);
}
//将所读取的字符串转换为字节数组
byte[] content=Encoding.ASCII.GetBytes(recvStr);
try
{
//创建文件流对象实例
FileStream fs=new FileStream(fileName,FileMode.OpenOrCreate,FileAccess.ReadWrite);
//写入文件
fs.Write(content,0,content.Length);
}
catch(Exception fe)
{
MessageBox.Show("文件创建/写入错误:"+fe.Message,"提示信息
",MessageBoxButtons.RetryCancel,MessageBoxIcon.Information);
}
//禁用 Socket
socket.Shutdown(SocketShutdown.Both);
//关闭 Socket
socket.Close();
}
）
```

程序在 WindowsXP 中文版、.Net Frameworkd 中文正式版、Visual Studio.Net 中文正式版下调试通过.

（二）Visual C#.Net 网络程序开发-Tcp 篇

前一篇《Visual C#.Net 网络程序开发-Socket 篇》中说到：支持 Http、Tcp 和 Udp 的类组成了 TCP/IP 三层模型(请求响应层、应用协议层、传输层)的中间层-应用协议层，该层的类比位于最底层的 Socket 类提供了更高层次的抽象，它们封装 TCP 和 UDP 套接字的创建，不需要处理连接的细节，这使得我们在编写套接字级别的协议时，可以更多地尝试使用 TCPClient 、 UDPClient 和 TcpListener，而不是直接向 Socket 中写。它们之间的这种层次关系示意如下：

可见， TcpClient 类基于 Socket 类构建，这是它能够以更高的抽象程度提供 TCP 服务的基础。正因为这样，许多应用层上的通讯协议，比如 FTP(File Transfers Protocol)文件传输协议、HTTP(Hypertext Transfers Protocol)超文本传输协议等都直接创建在 TcpClient 等类之上。

TCPClient 类使用 TCP 从 Internet 资源请求数据。TCP 协议建立与远程终结点的连接，然后使用此连接发送和接收数据包。TCP 负责确保将数据包发送到终结点并在数据包到达时以正确的顺序对其进行组合。

从名字上就可以看出，TcpClient 类专为客户端设计，它为 TCP 网络服务提供客户端连接。TcpClient 提供了通过网络连接、发送和接收数据的简单方法。

若要建立 TCP 连接，必须知道承载所需服务的网络设备的地址(IPAddress)以及该服务用于通讯的 TCP 端口 (Port)。Internet 分配号码机构 (Internet Assigned Numbers Authority, IANA) 定义公共服务的端口号（你可以访问 http://www.iana.org/assignments/port-numbers 获得这方面更详细的资料）。IANA 列表中所没有的服务可使用 1,024 到 65,535 这一范围中的端口号。要创建这种连接，你可以选用 TcpClient 类的三种构造函数之一：

1、public TcpClient()当使用这种不带任何参数的构造函数时，将使用本机默认的 ip 地址并将使用默认的通信端口号 0。这样情况下，如果本机不止一个 ip 地址，将无法选择使用。以下语句示例了如何使用默认构造函数来创建新的 TcpClient：

```
TcpClient tcpClientC = new TcpClient();
```

2、public TcpClient(IPEndPoint)使用本机 IPEndPoint 创建 TcpClient 的实例对象。上一篇介绍过了，IPEndPoint 将网络端点表示为 IP 地址和端口号，在这里它用于指定在建立远程主机连接时所使用的本地网络接口（IP 地址）和端口号，这个构造方法为使用本机 IPAddress 和 Port 提供了选择余地。下面的语句示例了如何使用本地终结点创建 TcpClient 类的实例：

```
IPHostEntry ipInfo=Dns.GetHostByName("www.tuha.net");//主机信息
IPAddressList[] ipList=ipInfo.AddressList;//IP 地址数组
IPAddress ip=ipList[0];//多 IP 地址时一般用第一个
IPEndPoint ipEP=new IPEndPoint(ip,4088);//得到网络终结点
try{
TcpClient tcpClientA = new TcpClient(ipLocalEndPoint);
}
catch (Exception e ) {
```

```
Console.WriteLine(e.ToString());
}
```

到这里，你可能会感到困惑，客户端要和服务端创建连接，所指定的 IP 地址及通信端口号应该是远程服务器的呀！事实上的确如此，使用以上两种构造函数，你所实现的只是 TcpClient 实例对象与 IP 地址和 Port 端口的绑定，要完成连接，你还需要显式指定与远程主机的连接，这可以通过 TcpClient 类的 Connect 方法来实现， Connet 方法使用指定的主机名和端口号将客户端连接到 远程主机：

1)、public void Connect(IPEndPoint); 使用指定的远程网络终结点将客户端连接到远程 TCP 主机。

public void Connect(IPAddress, int); 使用指定的 IP 地址和端口号将客户端连接到 TCP 主机。

public void Connect(string, int); 将客户端连接到指定主机上的指定端口。

需要指出的是，Connect 方法的所有重载形式中的参数 IPEndPoint 网络终

结点、IPAddress 以及表现为 string 的 Dns 主机名和 int 指出的 Port 端口均指的是远程服务器。

以下示例语句使用主机默认 IP 和 Port 端口号 0 与远程主机建立连接：

```
TcpClient tcpClient = new TcpClient();//创建 TcpClient 对象实例
try{
tcpClient.Connect("www.contoso.com",11002);//建立连接
}
catch (Exception e ){
Console.WriteLine(e.ToString());
}
```

3、public TcpClient(string, int);初始化 TcpClient 类的新实例并连接到指定主机上的指定端口。与前两个构造函数不一样，这个构造函数将自动建立连接，你不再需要额外调用 Connect 方法，其中 string 类型的参数表示远程主机的 Dns 名，如：www.tuha.net。

以下示例语句调用这一方法实现与指定主机名和端口号的主机相连：

```
try{
TcpClient tcpClientB = new TcpClient("www.tuha.net", 4088);
}
catch (Exception e ) {
Console.WriteLine(e.ToString());
}
```

前面我们说,TcpClient 类创建在 Socket 之上，在 Tcp 服务方面提供了更高层次的抽象,体现在网络数据的发

送和接受方面,是 TcpClient 使用标准的 Stream 流处理技术，使得它读写数据更加方便直观，同时，.Net 框架负责提供更丰富的结构来处理流，贯穿于整个.Net 框架中的流具有更广泛的兼容性，构建在更一般化的流操作上的通用方法使我们不再需要困惑于文件的实际内容（HTML、XML 或其他任何内容），应用程序都将使用一致的方法（Stream.Write、Stream.Read） 发送和接收数据。另外，流在数据从 Internet 下载的过程中提供对数据的即时访问，可以在部分数据到达时立即开始处理，而不需要等待应用程序下载完整个数据集。.Net 中通过 NetworkStream 类实现了这些处理技术。

NetworkStream 类包含在.Net 框架的 System.Net.Sockets 命名空间里，该类专门提供用于网络访问的基础数据流。NetworkStream 实现通过网络套接字发送和接收数据的标准.Net 框架流机制。NetworkStream 支持对网络数据流的同步和异步访问。NetworkStream 从 Stream 继承，后者提供了一组丰富的用于方便网络通讯的方法和属性。

同其它继承自抽象基类 Stream 的所有流一样，NetworkStream 网络流也可以被视为一个数据通道，架设在数据来源端(客户 Client)和接收端（服务 Server）之间，而后的数据读取及写入均针对这个通道来进行。

.Net 框架中，NetworkStream 流支持两方面的操作：

1、 写入流。写入是从数据结构到流的数据传输。

示 意 图

2、读取流。读取是从流到数据结构（如字节数组）的数据传输。

示 意 图

与普通流 Stream 不同的是，网络流没有当前位置的统一概念，因此不支持查找和对数据流的随机访问。相应属性 CanSeek 始终返回 false，而 Seek 和 Position 方法也将引发 NotSupportedException。

基于 Socket 上的应用协议方面，你可以通过以下两种方式获取 NetworkStream 网络数据流：

1、使用 NetworkStream 构造函数：public NetworkStream(Socket, FileAccess, bool);（有重载方法）,它用指定的访问权限和指定的 Socket 所属权为指定的 Socket 创建 NetworkStream 类的新实例，使用前你需要创建 Socket 对象实例，并通过 Socket.Connect 方法建立与远程服务端的连接，而后才可以使用该方法得到网络传输流。示例如下：

```
Socket s=new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);//创建客户端 Socket 对象实例
try{
s.Connect("www.tuha.net",4088);//建立与远程主机的连接
}
catch(Exception e){
MessageBox.show("连接错误：" +e.Message);
}
```

```
try{
NetworkStream stream=new NetworkStream(s,FileAccess.ReadWrite,false);//取得网络传输流
}
```

2、通过 TcpClient.GetStream 方法：public NetworkStream etStream();它返回用于发送和接收数据的基础网络流 NetworkStream。GetStream 通过将基础 Socket 用作它的构造函数参数来创建 NetworkStream 类的实例。使用前你需要先创 TcpClient 对象实例并建立与远程主机的连接，示例如下：

```
TcpClient tcpClient = new TcpClient();//创建 TcpClient 对象实例
Try{
tcpClient.Connect("www.tuha.net",4088);//尝试与远程主机相连
}
catch(Exception e){
MessageBox.Show("连接错误:"+e.Message);
}
try{
NetworkStream stream=tcpClient.GetStream();//获取网络传输流
}
catch(Exception e)
{
MessageBox.Show("TcpClient 错误："+e.Message);
}
```

通过以上方法得到 NetworkStream 网络流之后,你就可以使用标准流读写方法 Write 和 Read 来发送和接受数据了。

以上是.Net 下使用 TcpClient 类实现客户端编程的技术资料，为了向客户端提供这些服务，我们还需要编制相应的服务端程序，前一篇《Visual C#.Net 网络程序开发-Socket 篇》上曾经提到， Socket 作为其他网络协议的基础，既可以面向客户端开发，也可以面向服务端开发，在传输层面上使用较多，而在应用协议层面上，客户端我们采用构建于 Socket 类之上的 TcpClient 取代 Socket；相应地，构建于 Socket 之上的 TcpListener 提供了更高理念级别的 TCP 服务，使得我们能更方便地编写服务端应用程序。正是因为这样的原因，像 FTP 和 HTTP 这样的应用层协议都是在 TcpListener 类的基础上建立的。

.Net 中的 TCPListener 用于监视 TCP 端口上的传入请求，通过绑定本机 IP 地址和相应端口（这两者应与客户端的请求一致）创建 TcpListener 对象实例,并由 Start 方法启动侦听；当 TcpListener 侦听到用户端的连接后，视客户端的不同请求方式，通过 AcceptTcpClient 方法接受传入的连接请求并创建 TcpClient 以处理请求，或者通过 AcceptSocket 方法接受传入的连接请求并创建 Socket 以处理请求。最后，你需要使用 Stop 关闭用于侦听传入连接的 Socket，你必须也关闭从 AcceptSocket 或 AcceptTcpClient 返回的任何实例。这个过程详细解说如下：

首先，创建 TcpListener 对象实例，这通过 TcpListener 类的构造方法来实现：

```
public TcpListener(port);//指定本机端口
public TcpListener(IPEndPoint)//指定本机终结点
public TcpListener(IPAddress,port)//指定本机 IP 地址及端口
```

以上方法中的参数在前面多次提到，这里不再细述，唯一需要提醒的是，这些参数均针对服务端主机。下面的示例演示创建 TcpListener 类的实例：

```
IPHostEntry ipInfo=Dns.Resolve("127.0.0.1");//主机信息
IPAddressList[] ipList=ipInfo.IPAddressList;//IP 数组
IPAddress ip=ipList[0];//IP
try{
TcpListener tcpListener = new TcpListener(ipAddress, 4088);//创建 TcpListener 对象实例以侦听用户端连接
}
catch ( Exception e){
MessageBox.Show("TcpListener 错误："+e.Message);
}
```

随后，你需要调用 Start 方法启动侦听：

```
public void Start();
```

其次，当侦听到有用户端连接时，需要接受挂起的连接请求，这通过调用以下两方法之一来完成连接：

```
public Socket AcceptSocket();
public TcpClient AcceptTcpClient();
```

前一个方法返回代表客户端的 Socket 对象，随后可以通过 Socket 类的 Send 和 Receive 方法与远程计算机通讯；后一个方法返回代表客户端的 TcpClient 对象，随后使用上面介绍的 TcpClient.GetStream 方法获取 TcpClient 的基础网络流 NetworkStream，并使用流读写 Read/Write 方法与远程计算机通讯。

最后，请记住关闭侦听器：public void Stop();

同时关闭其他连接实例：public void Close();

下面的示例完整体现了上面的过程：

```
bool done = false;
TcpListener listener = new TcpListener(13);// 创建 TcpListener 对象实例(13 号端口提供时间服务)
listener.Start();//启动侦听
```

```
while (!done) {//进入无限循环以侦听用户连接
TcpClient client = listener.AcceptTcpClient();//侦听到连接后创建客户端连接 TcpClient
NetworkStream ns = client.GetStream();//得到网络传输流
byte[] byteTime = Encoding.ASCII.GetBytes(DateTime.Now.ToString());//预发送的内容(此为服务端时间)转换为字节数组以便写入流
try {
ns.Write(byteTime, 0, byteTime.Length);//写入流
ns.Close();//关闭流
client.Close();//关闭客户端连接
}
catch (Exception e) {
MessageBox.Show("流错误:"+e.Message)
}
```

综合运用上面的知识，下面的实例实现了简单的网络通讯-双机互连，针对客户端和服务端分别编制了应用程序。客户端创建到服务端的连接，向远程主机发送连接请求连接信号，并发送交谈内容；远程主机端接收来自客户的连接，向客户端发回确认连接的信号，同时接收并显示客户端的交谈内容。在这个基础上，发挥你的创造力，你完全可以开发出一个基于程序语言(C#)级的聊天室！

客户端主要源代码：

```
public void SendMeg()//发送信息
{
try
{


int port=Int32.Parse(textBox3.Text.ToString());//远程主机端口
try
{
tcpClient=new TcpClient(textBox1.Text,port);//创建 TcpClient 对象实例 }
catch(Exception le)
{
MessageBox.Show("TcpClient Error:"+le.Message);
}
string strDateLine=DateTime.Now.ToShortDateString()+" "+DateTime.Now.ToLongTimeString();//得到发送时客户端时间
netStream=tcpClient.GetStream();//得到网络流
sw=new StreamWriter(netStream);//创建 TextWriter,向流中写字符
string words=textBox4.Text;//待发送的话
string content=strDateLine+words;//待发送内容
sw.Write(content);//写入流
sw.Close();//关闭流写入器
netStream.Close();//关闭网络流
tcpClient.Close();//关闭客户端连接
```

```
}
catch(Exception ex)
{
MessageBox.Show("Sending Message Failed!"+ex.Message);
}
textBox4.Text="";//清空
}
```

服务器端主要源代码：

```
public void StartListen()//侦听特定端口的用户请求
{
//ReceiveMeg();
isLinked=false; //连接标志
try
{
int port=Int32.Parse(textBox1.Text.ToString());//本地待侦听端口
serverListener=new TcpListener(port);//创建 TcpListener 对象实例
serverListener.Start(); //启动侦听
}
catch(Exception ex)
{
MessageBox.Show("Can't Start Server"+ex.Message);
return;
}
isLinked=true;
while(true)//进入无限循环等待用户端连接
{
try
{
tcpClient=serverListener.AcceptTcpClient();//创建客户端连接对象
netStream=tcpClient.GetStream();//得到网络流
sr=new StreamReader(netStream);//流读写器
}
catch(Exception re)
{
MessageBox.Show(re.Message);
}
string buffer="";
string received="";
received+=sr.ReadLine();//读流中一行
while(received.Length!=0)
{
```

```
buffer+=received;
buffer+="\r\n";
//received="";
received=sr.ReadLine();
}
listBox1.Items.Add(buffer);//显示
//关闭
sr.Close();
netStream.Close();
tcpClient.Close();
}
}
```