Lập trình mạng trong .NET

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Lập trình mạng trong .NET

Mạng cục bộ LAN(Local Area Network) là mạng được thiết lập để liên kết các máy tính trong một khu vực như trong một tòa nhà, một khu nhà

Microsoft .NET Framework bao gồm một tập các lớp dùng để lập trình mạng thuộc hai không gian tên: System.Net và System.Net.Sockets. Các lớp này hỗ trợ mọi thứ, từ lập trình dựa trên-socket với TCP/IP cho đến download file và trang

HTML từ web thông qua HTTP. Hai không gian tên này cung là nền tảng cho hai nền networking cấp cao hơn - Remoting và dịch vụ Web XML.

Giới thiệu về Socket

SocKet là một giao diện lập trình (API – Application Program Interface) ứng dụng mạng thông qua giao diện này có thể lập trình điều khiển việc truyền thông giữa 2 máy sử dụng các giao thức mức thấp như TCP,UDP… , Socket là một sự trừu tượng hóa ở mức cao, có thể tưởng tượng nó như một thiết bị truyền thông 2 chiều tương tự như tệp tin, chúng ta gửi/nhận dữ liệu giữa 2 máy, tương tự như việc đọc ghi trên tệp tin

Để liên lạc thông qua socket, ta cần tiến hành các thao tác sau:

- Tạo hay mở một socket

+ Gắn một socket với một địa chỉ, địa chỉ này chính là địa chỉ của máy cần liên lạc.

+ Thực hiện việc liên lạc tuỳ thuộc vào chế độ kết nối:

* Liên lạc theo chế độ không kết nối (UDP):

Hai tiến trình liên lạc với nhau không kết nối trực tiếp, mỗi thông điệp gửi đi phải kèm theo địa chỉ người nhận,

Hình thức liên lạc này có đặc điểm:

- Người gửi không chắc chắn thông điệp của họ có đến tay người nhận hay không?
- Một thông điệp có thể gửi nhiều lần.
- Thông điệp gửi sau có thể đến đích trước thông điệp gửi trước đó.

* Liên lạc trong chế độ kết nối (TCP):

Có một đường kết nối ảo được thành lập giữa 2 tiến trình, trước khi một kết nối được thành lập thì một trong 2 tiến trình đó phải đợi tiến trình kia yêu cầu kết nối, có thể sử dụng theo mô hình kết nối Client/Server. Trong đó mô hình server sử dụng lời kêu gọi listen và accept để lắng nghe và chấp nhận một yêu cầu kết nối.

Trong phần này chúng ta học cách thao tác kết nối socket bằng .NET Compact Framework.

Lập trình Socket với .NET Compact Framework

Lớp System.Net.Sockets.Socket. Thủ tục để nhận một lớp Socket kết nối với máy ở xa phụ thuộc vào máy tính đó, tuy nhiên quá trình xử lý để đọc và ghi dữ liệu là giống nhau. Để sử dụng các lớp xử lý mạng trong .NET Compact Framework, chúng ta phải khai báo không gian tên System.Net. Ví dụ: using System.Net.

Tạo kết nối từ máy khách tới máy chủ (client)

Để tạo một kết nối thành công, trước tiên chúng ta phải tìm hiểu lớp System.Net.EndPoint. Để lưu giữ thông tin về điểm cuối nơi mà kết nối đến: địa chỉ IP của máy chủ và số hiệu cổng mong muốn. Để thiết lập đúng điểm cuối và sử dụng nó để kết nối socket tới máy chủ, chúng ta làm theo các bước sau:

Bước 1: Khai báo biến điểm cuối (EndPoint) và biến Socket.

Bước 2: Điểm cuối gồm thông tin địa chỉ và số hiệu cổng. Có hai cách để làm điều này, phụ thuộc vào địa chỉ của máy chủ, giống như là: 192.168.4.3, hoặc tên DSN của máy chủ, như là www.mycomputer.net.

Tìm địa chỉ IP của một máy chủ:

Nếu chúng ta biết địa chỉ IP của máy chủ, sử dụng IPAddress trong cấu trúc. Ví dụ sau mô tả khởi tạo một điểm cuối, máy chủ có địa chỉ IP là 192.168.4.3, và cổng 2000:

```
EndPoint l_EndPoint = new IPEndPoint( IPAddress.Parse("192.168.4.3"), Convert.ToInt16(2000));
```

Nếu chúng ta không biết địa chỉ IP, chúng ta phải dùng DSN để tìm địa chỉ IP của máy chủ thông qua tên. DSN tìm kiếm trả lại địa chỉ IP tương ứng với tên. Đoạn mã sau là một trường hợp:

```
IPHostEntry l_IPHostEntry = Dns.Resolve("www.mycomputer.net");

EndPoint l_EndPoint = new IPEndpoint(l_IPHostEntry.AddressList[0], 9981);
```

Bước 3: Sử dụng điểm cuối (EndPoint) để thử kết nối socket tới máy chủ. Chúng ta phải sử dụng mệnh đề try/catch ở đây, bởi vì thử kết nối sẽ đưa ra một ngoại lệ nếu có vấn đề, như máy chủ từ chối không chấp nhận kết nối hoặc máy chủ không tồn tại,...

Ví dụ sau mô tả ba bước ở trên:

```
try

{

Socketl_Socket =new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

l_Socket.Connect(l_EndPoint);

if (l_Socket.Connected){

// l_Socket bầy giờ có thể gửi và nhận dữ liệu

}

}

catch (SocketException e)

{ /* Đưa ra thông báo lỗi,… */ }
```

Tạo kết nối từ máy chủ lằng nghe từ máy khách (Host)

Chúng ta có thể thu được một kết nối socket từ máy tính ở xa bằng cách đảm nhiệm như là máy chủ. Khi một thiết bị như máy chủ, nó đợi nhận kết nối từ các máy khách. Để tạo kết nối để thiết bị của chúng ta như là máy chủ, chúng ta phải thiết lập một socket lắng nghe trên một cổng đến khi một ai đó gửi một yêu cầu kết nối đến thiết bị của chúng ta. Sau đây là các bước tạo socket lắng nghe trên một cổng để cho máy khác kết nối tới:

Bước 1: Tạo một socket để lắng nghe kết nối.

Bước 2: Ràng buộc socket lắng nghe trên một cổng. Nó chỉ lắng nghe kết nối trên một cổng.

Bước 3: Gọi Accept() trên socket lắng nghe nhận được từ socket khác khi một ai đó kết nối tới. Đoạn mã có thể đọc và ghi socket nhận được, và socket tiếp tục đợi kết nối mới.

Ví dụ sau mô tả ba bước ở trên:

```
m_listenSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

m_listenSocket.Bind(new IPEndPoint(IPAddress.Any, 8758));

m_listenSocket.Listen((int)SocketOptionName.MaxConnections);

m_connectedSocket = m_listenSocket.Accept();

if (m_connectedSocket != null)

{

if (m_connectedSocket.Connected)

{

// Someone has connected to us.

}

}
```

Gửi và nhận trên Socket đã kết nối

Một socket được kết nối tới máy tính ở xa. Nó có thể sử dụng gửi và nhận dữ liệu. Cách đơn giản nhất để làm việc này là gọi Socket.Send() để gửi dữ liệu và Socket.Receive() nhận dữ liệu.

<u>Gửi dữ liệu vào một Socket cùng với Socket.Send</u>

Socket.Send() có bốn thành phần nạp chồng, mỗi thành phần là một mức khác nhau của điều khiển thông qua cái được gửi:

- Send(Byte[] buffer): Gửi tất cả mội thứ bên trong mảng byte buffer.

- Send(Byte[] buffer, SocketFlags socketFlags) Gửi tất cả mọi thứ trong buffer cùng với sự hạn chế riêng thông qua cách dữ liệu đi như thế nào.

- Send(Byte[] buffer, Int32 size, SocketFlags socketFlags): Gửi tất cả dữ liệu trong buffer tuỳ theo kích cỡ size. Nếu chúng ta muốn gửi chỉ một phần của một buffer, sau đó có thể chỉ rõ SocketFlags.None sử dụng mặc định hành vi gửi. Ví dụ, để gửi 16 byte đầu tiền của mảng, chúng ta có thể sử dụng l_Socket.Send(l_buffer, 16, SocketFlags.None).

- Send(Byte[]buffer,Int32 offset Int32 size, SocketFlags socketFlags): Giống như thành phần trên chỉ khác là chúng ta có thể chỉ rõ chỉ số bắt đầu của mảng. Ví dụ, để gửi từ byte tứ 3 đến bute thứ 7 của mảng, chúng ta có thể sử dụng như sau: l_Socket.Send(l_buffer, 2, 6, SocketFlags.None);

Phương thức Send trả về số byte gửi thành công. Vấn đề này cùng với phương thức send() dường như giống nhau rất nhiều việc biến đổi tất cả mọi cái chúng ta muốn gửi vào mảng các byte để gửi thông qua socket. .NET Compact Framework hỗ trợ hai lớp rất hữu ích, System.Text.Encoding và System.Convert, hai lớp này giúp chuyển đổi kiểu dữ liệu cơ bản thành mảng các byte để có thể gửi qua socket.

Cách dễ nhất để tìm hiểu cách sử dụng lớp Encoding và Convert là xem ví dụ. Sau đây

là ví dụ socket có tên là l_Socket đã tồn tại và đã được kết nối:

Gửi một chuỗi sử dụng mã hoá ASCII :

```
l_Socket.Send(Encoding.ASCII.GetBytes("Send me")
```

Gửi một chuỗi sử dụng mã hoá Unicode:

```
l_Socket.Send(Encoding.Unicode.GetBytes("Send me")
```

Gửi một số nguyên có giá trị 2003:

```
l_Socket.Send(Encoding.ASCII.GetBytes(Convert.ToString(2003))
```

Gửi một số thực có giá trị 2.7:

```
l_Socket.Send(Encoding.ASCII.GetBytes(Convert.ToString(2.7))
```

<u>Nhận dữ liệu từ từ socket bằng</u> <u>Socket.Receive</u>

Nhận dữ liệu từ một socket thông qua phương thức Socket.Receive. Receive có bốn thành phần nạp chồng, giống như thành phần nạp chồng của Socket.Send. Mỗi thành phần nạp chồng trả về số byte đọc thành công:

- Receive (Byte[] buffer): Thành phần này nhận dữ liệu trong bộ đệm.

- Receive (Byte[] buffer, SocketFlags socketFlags) Thành phần này nhận dữ liệu trong bộ đệm bằng cách sử dụng cờ để chỉ ra dữ liệu được lấy như thế nào.

- Receive (Byte[] buffer, Int32 size, SocketFlags socketFlags) Thành phần này nhận tuỳ theo kích cỡ của dữ liệu trong bộ đệm. Nếu dữ liệu nhiều hơn dữ liệu sẵn sàng, nó được bỏ qua. Chúng ta có thể nhận dữ liệu còn lại bằng cách gọi lại Receive. Nếu chúng ta chỉ muốn nhận những byte mà chúng ta không nhận được, sau đó chúng ta có thể chỉ SocketFlags.None để sử dụng mặc định cho hành động gửi. Ví dụ để nhận 16 byte đầu tiên của dữ liệu sẵn sàng, sử dụng l_Socket.Receive(l_buffer, 16, SocketFlags.None)

- Receive (Byte[]buffer, Int32 offset Int32 size, SocketFlags socketFlags) Thành phần này giống như thành phần trước, chỉ khác là chúng ta có thể chỉ ra chỉ số trong mảng để sử dụng bắt đầu ghi dữ liệu vào mảng. Ví dụ, để nhận 7 byte dữ liệu trong bộ đệm bắt đầu từ vị trí thứ 3 trong bộ đệm, sử dụng đoạn mã sau:

```
l_Socket.Receive(l_buffer, 2, 6, SocketFlags.None);
```

Có kỹ thuật cho phép chuyển đổi dữ liệu để gửi từ socket ra mảng, kỹ thuật đơn giản nhấtlà chuyển đổi mảng byte trong kiểu dữ liệu cơ bản. Như phần trước, lớp Encoding và Convert cung cấp phương tiện cho chuyển đổi, và chúng ta sẽ xem trong ví dụ. Đầy là ví dụ thừa nhận dữ liệu đã được nhận trong mảng Byte có tên là l_Buffer:

Chuyển đổi các byte nhận được trong một chuỗi ASCII :

```
string l_ASCII = Encoding.ASCII.GetString(l_Buffer);
```

Chuyển đổi các nhận được trong một chuỗi Unicode:

```
string l_ASCII = Encoding.Unicode.GetString(l_Buffer);
```

Chuyển đổi các byte nhận được, cái đó là mã ASCII text integer:

```
int l_Integer = Convert.ToInt32(Encoding.ASCII.GetString(l_Buffer));
```

Chuyển đổi các byte nhận được, cái đó là mã ASCII text integer, into a Double:

```
Double l_Double = Convert.ToInt32(Encoding.ASCII.GetString(l_Double));
```

Kiểu dữ liệu đầu vào được chấp nhận

Phương thức	**Tên của các kiểu dữ liệu đầu vào được chấp nhận**
ToBoolean	object, bool, sbyte, char, byte, short, ushort, int, uint, long, String, float, double, decimal
ToChar	object, char, sbyte, byte, short, ushort, int, uint, long, ulong, String, float, double, decimal
ToSByte	object, bool, sbyte, char, byte, short, ushort, int, uint, long, ulong, float, double, decimal, String
ToByte	object, bool, sbyte, char, byte, short, ushort, int, uint, long, ulong, float, double, decimal, String
ToInt16	object, bool, char, sbyte, byte, ushort, int, uint, short, long, ulong, float, double, decimal, String
ToUInt16	object, bool, char, sbyte, byte, ushort, int, uint, short, long, ulong, float, double, decimal, String
ToInt32	object, bool, char, sbyte, byte, short, ushort, uint, int, long, ulong, float, double, decimal, String
ToUInt32	object, bool, char, sbyte, byte, short, ushort, uint, int, long, ulong, float, double, decimal, String
ToInt64	object, bool, char, sbyte, byte, short, ushort, int, uint, ulong, long, float, double, decimal, String
ToUInt64	object, bool, char, sbyte, byte, short, ushort, int, uint, ulong, long, float, double, decimal, String
ToSingle	object, sbyte, byte, char, short, ushort, int, uint, long, ulong, float, double, decimal, String, bool
ToDouble	object, sbyte, byte, short, char, ushort, int, uint, long, ulong, float, double, decimal, String, bool
ToDecimal	object, sbyte, byte, char, short, ushort, int, uint, long, ulong, float, double, String, decimal, bool, DateTime
ToDateTime	object, String
ToString	Object, bool, char, sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal, Decimal, DateTime
ToBase64String	byte[]
Byte[]FromBase64String	String

ToBase64CharArray	byte[]
Byte[]FromInt64CharArray	char[]

Tuần tự hóa đối tượng để truyền qua Socket

Phiển bản desktop của .NET Framework cho phép tuần tự hóa hầu hết kiểu đối tượng thành mảng các byte, để có thể gửi qua socket. Các đối tượng phức tạp, người phát triển thực thi giao diện ISerializable, cùng với mã tuần tự (serialize) và hồi phục (deserialize) đối tượng dữ liệu.

.NET Compact Framework không hỗ trợ những chức năng này. Lớp DataSet là lớp duy nhất có thể tự tuần tự hóa. Thông thường lớp DataSet được sử dụng như là một cơ sở dữ liệu quan hệ trong bộ nhớ. Nó là một ý tưởng cho bộ đệm dữ liệu nhỏ dựa vào máy chủ ở xa trong khi duy trì cấu trúc quan hệ của dữ liệu. DataSet có thể lưu trữ tất cả các kiểu dữ liệu cơ bản trên .NET Compact Framework.

Sử dụng gói UDP và gói TCP

Như đã đề cập, có hai kiểu gói tin thường được sử dụng để truyền tin trên mạng. Kiểu chung nhất, gói TCP phải chọn cho gần như tất cả các trường hợp bởi vì nó đảm bảo rằng dữ liệu đến không bị hư hỏng hoặc ngược lại trả lại tín hiệu lỗi nếu có một vấn đề gì mà không thể sửa chữa. Gói tin UDP rất hữu ích cho các ứng dụng dòng thời gian thực. Gói tin UDP khác gói tin TCP trong cách mà chúng kết nối, giao thức TCP là giao thức hướng kết nối, điều này có nghĩa là chúng ta cần kết nối với một socket trên máy tính ở xa trước khi chúng ta có thể gửi hoặc nhận dữ liệu bằng socket. Giao thức kết nối không yêu cầu bất kỳ kết nối nào được thiết lập trước khi có gắn gửi hoặc nhận dữ liệu. Nếu không có một lắng nghe rên địa chỉ IP và cổng nơi mà gói UDP được gửi, sau đó gói tin đơn giản là bị mất.

Cách đơn giản nhất đển làm việc với gói UDP là sử dụng lớp UdpClient, lớp này được .NET Compact Framework hỗ trợ. Lớp UdpClient cho phép các lập trình viên gửi các byte tới nhóm ở xa. UdpClient cho phép người phát triển nhận byte từ nhóm ở xa hoặc từ bất kỳ người nào cố gắng gửi dữ liệu tới cổng mà UdpClient lắng nghe. Quan tâm đến các cấu trúc và phương thức được UdpClient sử dụng sau:

- void Connect(String hostname, Int32 port) Thiết lập kết nối tới một máy tính có địa chỉ IP tương ứng được chỉ ra bởi tên máy chủ (hostname) và số hiệu cổng (port). Sau đó sử dụng phương thức Send(Byte[] dgram, Int32 bytes) sẽ gửi dữ liệu đến vị trí được chỉ ra trong phân kết nối. Phương thức này trả về kiểu void bởi vì không có khái niệm kết nối thành công khi sử dụng gói UDP. Phương thức này chỉ đơn thuần là tạo để gửi dữ liệu tới một địa chỉ IP và số hiệu cổng.

- void Connect(IPAddress addr, Int32 port) Giống như phương thức trước, ngoại trừ cho phép bạn chỉ ra máy tính ở xa bằng IPAddress và port. Đoạn mã ví dụ:

```
l_UdpClient.Connect(IPAddress.Parse("172.68.25.34"), 9999)
```

- void Connect(IPEndpoint endPoint) Kết nối với máy xa bằng cách chỉ ra endPoint.

- Int32 Send(Byte[] dgram, Int32 bytes, IPEndPoint endPoint) Gửi tất cả bytes của bộ đệm dgram tới máy tính có địa chỉ IP và cổng được chỉ ra trong endPoint.

- Send(Byte[] dgram, Int32 bytes, String hostname, Int32 port) Gửi tất cả các bytes của bộ đệm dgram tới máy tính có địa chỉ IP tương ứng với hostname và cổng, như trong đoạn mã ví dụ sau:

```
Send(aBuffer, aBuffer.Length, "www.mycomputer.net", 9999)
```

Phương thức trên trả về số byte gửi.

- Send(Byte[] dgram, Int32 bytes) Gửi tổng số byte của bộ đệm tới máy chủ ở xa đươc chỉ ra trong phương thức kết nối. Để sử dụng thành phần nạp chồng, chúng ta trước tiền phải gọi Connect, vì vậy UdpClient biết nơi gửi gói UDP. Phương thức này trả về số byte gửi được.

- Receive(ref IPEndPoint remoteEP) Đợi để nhận dữ liệu từ EndPoint. Chúng ta có thể tạo một EndPoint tham chiếu đến một địa chỉ IP và cổng, hoặc chúng ta có thể thiết lập EndPoint để nhận dữ liệu từ bất kỳ địa chỉ IP và port. EndPoint được cập nhật sau khi dữ liệu được nhận cho biết nơi dữ liệu đến.

Viết mã cho UdpClient

Đoạn mã này mô tả cách thiết lập một UdpClient, sau đó gửi gói tin UDP tới máy tính có địa chỉ IP là 192.168.0.200, cổng 8888. Chú ý là thông qua gọi phương thức UdpClient.Connect(). UdpClient biết nơi gửi gói tin UDP khi UdpClient.Send() được gọi, nhưng không có kết nối liên tục.

```
IPEndPoint senderIP = new IPEndPoint(IPAddress.Parse("192.168.0.200"), Convert.ToInt32(8888));

UdpClient l_UdpClient = new UdpClient();

l_UdpClient.Connect(senderIP);

for (int i = 0; i < 20; i++)
```

```
{

l_UdpClient.Send(Encoding.ASCII.GetBytes("Hello_UDP_1"),

Encoding.ASCII.GetBytes("Hello_UDP_1").Length);

System.Threading.Thread.Sleep(1000);

}

l_UdpClient.Close();
```

Sau đây đoạn mã tạo một vòng lặp. Mỗi lần lặp của khối lặp và lắng nghe trên cổng 8888 đến khi nó nhận một gói tin từ bất kỳ địa chỉ IP nào.

```
IPEndPoint listenerIP = new IPEndPoint(IPAddress.Any, 8888);

UdpClient listener = new UdpClient(listenerIP);

for (int i = 0; i < Convert.ToInt16(this.txtMaxPackets.Text); i++)

{

// Now receive the three datagrams from the listener

IPEndPoint receivedIPInfo = new IPEndPoint(IPAddress.Any, 0);

byte[] data = listener.Receive(ref receivedIPInfo);

this.textBox1.Text += ("GOT: " +

Encoding.ASCII.GetString(data, 0,

data.Length) + " FROM: " + receivedIPInfo.ToString());

}
```

Mục đích của lớp UDPClient ở trên là dùng cho lập trình với giao thức UDP, với giao thức này thì hai bên không cần phải thiết lập kết nối trước khi gửi do vậy mức độ tin cậy không cao. Để đảm bảo độ tin cậy trong các ứng dụng mạng, người ta còn dùng một giao thức khác, gọi là giao thức có kết nối : TCP (Transport Control Protocol). Trên Internet chủ yếu là dùng loại giao thức này, ví dụ như Telnet, HTTP, SMTP, POP3… Để lập trình theo giao thức TCP, MS.NET cung cấp hai lớp có tên là TCPClient và TCPListener.

Các thành viên của lớp TCPClient

Thành viên của lớp TCPClient

Constructor Method	
Name	Description
TcpClient ()	Tạo một đối tượng TcpClient. Chưa đặt thông số gỡ.
TcpClient (IPEndPoint)	Tạo một TcpClient và gắn cho nú một EndPoint cục bộ. (Gán địa chỉ máy cục bộ và số hiệu cổng để sử dụng trao đổi thông tin về sau)
TcpClient (RemoteHost: String, RemotePort: Int32)	Tạo một đối tượng TcpClient và kết nối đến một máy có địa chỉ và số hiệu cổng được truyền vào.. RemoteHost có thể là địa chỉ IP chuẩn hoặc tên máy.

Public Properties (see also Protected Properties)

Các phương thức của lớp TCPClient

	Name	Description
	Available	Cho biết số byte đó nhận về từ mạng và cú sẵn để đọc.
	Client	Trả về Socket ứng với TCPClient hiện hành.
	Connected	Trạng thái cho biết đó kết nối được đến Server hay chưa ?

Public Methods (see also Protected Methods)

Các phương thức của lớp TCPClient

	Name	Description
	Close	Giải phóng đối tượng TcpClient nhưng không đóng kết nối.
	Connect (RemoteHost, RemotePort)	Kết nối đến một máy TCP khác có Tên và số hiệu cổng.
	GetStream	Trả về NetworkStream để từ đó giúp ta gửi hay nhận dữ liệu. (Thường làm tham số khi tạo StreamReader và StreamWriter để gửi và nhận dữ liệu dưới dạng xâu ký tự) .Khi đó gắn vào StreamReader và StreamWriter rồi thỡ ta cú thể gửi và nhận dữ

		liệu thụng qua cỏc phương thức Readline, writeline tương ứng của các lớp này.

Từ các thành viên của lớp TCPClient ở trên ta thấy rằng, việc kết nối và thực hiện gửi nhận rất đơn giản. Theo các trình tự sau:

B1: Tạo một đối tượng TCPClient

B2: Kết nối đến máy chủ (Server) dùng phương thức Connect

B3: Tạo 2 đối tượng StreamReader (Receive)và StreamWriter (Send) và "nối" với GetStream của TCPClient

B4: - Dùng đối tượng StreamWriter.Writeline/write vừa tạo ở trên để gửi dữ liệu đi.

- Dùng đối tượng StreamReader.Readline/Read vừa tạo ở trên để đọc dữ liệu về.

B5: Đóng kết nối.

*** Nếu muốn gửi/nhận dữ liệu ở mức byte (nhị phân) thì sử dụng NetworkStream. (truyền GetStream cho NetworkStream)

TCPListerner là một lớp cho phép người lập trình có thể xây dựng các ứng dụng Server (Ví dụ như SMTP Server, FTP Server, DNS Server, POP3 Server hay server tự định nghĩa ….). Ứng dụng server khác với ứng dụng Client ở chỗ nó luôn luôn thực hiện lắng nghe và chấp nhận các kết nối đến từ Client.

Các thành viên của lớp TCPListerner

Các thành viên của lớp TCPListerner

Constructor method	
Name	Description
TcpListener (Port: Int32)	Tạo một TcpListener và lắng nghe tại cổng chỉ định.
TcpListener (IPEndPoint)	Tạo một TcpListener với giỏ trị Endpoint truyền vào.
TcpListener (IPAddress, Port: Int32)	Tạo một TcpListener và lắng nghe các kết nối đến tại địa chỉ IP và cổng chỉ định.

Public Methods (see also Protected Methods)

Các phương thức của lớp TCPListerner

	Name	Description
	AcceptSocket	Chấp nhận một yêu cầu kết nối đang chờ.
	AcceptTcpClient	Chấp nhận một yêu cầu kết nối đang chờ. (Ứng dụng sẽ dừng tại lệnh này cho đến khi nào có một kết nối đến – "Blocking")
	Pending	Cho biết liệu có kết nối nào đang chờ đợi không ? (True = có).
	Start	Bắt đầu lắng nghe các yêu cầu kết nối.
	Stop	Dừng việc nghe.