C# 温故而知新：Stream篇（七）NetworkStream

1.NetworkStream的作用

和先前的流有所不同，NetworkStream 的特殊性可以在它的命名空间中得以了解（System.Net.Sockets）,聪明的你马上会反应过来：既然是在网络中传输的流，那必然有某种协议或者规则约束它，不错，这种协议便是Tcp/IP协议，这个是什么东东？别急，我先让大家了解下NetworkStream的作用：如果服务器和客户端之间基于TCP连接的，他们之间能够依靠一个稳定的字节流进行相互传输信息，这也是NetworkStream的最关键的作用，有了这个神奇的协议，NetWorkStream便能向其他流一样在网络中（进行点对点的传输），这种传输的效率和速度是非常高的（UDP也很快，稍后再介绍）如果大家对这个概念还不是很清晰的话，别怕，后文中我会更详细的说明

  这里有5点大家先理解就行

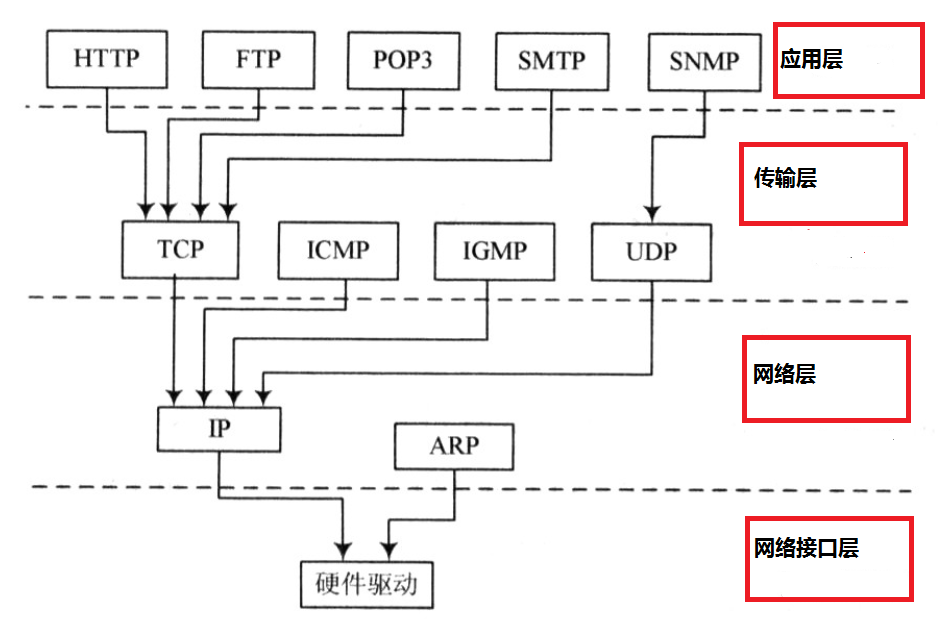
1. NetworkStream只能用在具有Tcp/IP协议之中，如果用在UDP中编译不报错，会报异常
2. NetworkStream 是面向连接的
3. 在网络中利用流的形式传递信息
4. 必须借助Socket (也称之为流式socket)，或使用一些返回的返回值，例如TcpClient类的GetStream方法
5. 用法和普通流方法几乎一模一样，但具有特殊性

2.简单介绍下TCP/IP 协议和相关层次

提到协议相信许多初学者或者没搞过这块的朋友会一头雾水，不过别怕，协议也是人定的，肯定能搞懂：其实协议可以这么理解，是人为定制的为某个活动定义的一些列规则和约束，就好比足球赛上的红黄牌，这是由世界足联定制的协议或者规范，一旦不按照这个协议足球赛肯定会一片混乱

进入正题：

TCP/IP全称：Transmission Control Protocol/Internet Protocol （传输控制协议/因特网互联协议，又名网络通讯协议）这个便是互联网中的最基本的协议，Tcp/IP 定义了电子设备如何进入到互联网，以及数据如何在网络中传递。既然有了协议但是空头支票还是不行地，就好比足联定制了这些规则，但是没有裁判在球场上来实施这些规则一样，Tcp/IP协议也有它自己的层次结构，关于它的层次结构，大家看图就能明白



 发送数据：大家不用刻板的去理解这个协议，我还是用我们最普通的浏览网页来让大家理解下，首先打开浏览器输入一个url,这时候应用层会判断这个要求是否是http的，然后http会将请求信息交给传输层来执行，传输层主要负责信息流的格式化并且提供一个可靠地传输，这时候，TCP和UDP这两个协议在这里起作用了，

TCP协议规定：接收端必须发回确认，并且假如分组丢失，必须重新发送，接着网络层得到了这些需要发送的数据，（网络中的IP协议非常重要，不仅是IP协议，还有ARP协议（查找远程主机MAC地址）），这时候网络层会命令网络接口层去发送这些信息（IP层主要负责的是在节点之间（End to End）的数据包传送，这里的节点是一台网络设备，比如计算机，大家便可理解为网络接口层的设备），最终将请求数据发送至远程网站主机后等待远程主机发送来信息

接收数据： 好了，远程网站主机会根据请求信息（Ip,数据报等等）发送一些列的网页数据通过网线或者无线路由，回到网络接口层，然后逐级上报，通过网络层的ip然后通过传输层的一些列格式化，最终通过http返回至浏览器显示网页了基于篇幅的关系，还有其他的协议大家可以自行去学习了解学习。喜欢足球的朋友的朋友也许会反应过来：这不是2-4-5阵型么？其实不然，很多协议我还没画上去，其实大致含义就是每个层次上的协议（足球队员有他各自的职责），这些才能构成计算机与计算机之间的传输信息的桥梁。

3.简单说明下 TCP和UDP的区别

TCP:

1 TCP是面向连接的通信协议，通过三次握手建立连接

2 TCP提供的是一种可靠的数据流服务，采用“带重传的肯定确认”技术来实现传输的可靠性

 UDP:

1 UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送

2 UDP通讯时不需要接收方确认，属于不可靠的传输，可能会出丢包现象，实际应用中要求在程序员编程验证

3 由于上述2点的关系，UDP传输速度更快，但是安全性比较差，很容易发生未知的错误，所以本章的NetworkStream无法使用在UDP的功能上

4.简单介绍下套接字（Socket）的概念

关于Socket的概念和功能可能可以写很长一篇博文来介绍，这里大家把Socket理解Tcp/IP协议的抽象，并且能够实现Tcp/IP协议栈的工具就行，换句话说，我们可以利用Socket实现客户端和服务端双向通信，同样，对于Socket最关键的理解还没到位，很多新人或者不常用的朋友会问：Socket到底功能是什么？怎么工作的？再次举个例子，女友打电话给我，我可以选择连接，或者拒绝，如果我接了她的电话，也就是说，我和她通过电话连接（Connect），那电话就是“Socket”，女友和我都可以是客户端或服务端，只要点对点就行，我们的声音通过电话传递，但是具体传输内容不归Socket管辖范围，Socket的直接任务可以归纳为以下几点：

|  |
| --- |
| 1. 创建客户端或服务端 2. 服务端或客户端监听是否有服务端或客户端传来的连接信息(Listening) 3. 创建点对点的连接(Connect) 4. 发送accept 信息给对方，表示两者已经建立连接，并且可以互相传递信息了(Send) 5. 具体发送什么信息内容不是Socket管辖的范围，但是必须是Socket进行发送的动作 6. 同理可以通过Socket去接受对方发来的信息，并加以处理 |

 简单的Socket示例代码：

 附件1

5.简单介绍下TcpClient,TcpListener,IPEndPoint类的作用

1: TcpClient：此类是微软基于Tcp封装类，用于简化Tcp客户端的开发，主要通过构造带入主机地址或者IPEndPonint对象，然后调用Connect进行和服务器点对点的连接，连接成功后通过GetStream方法返回NetworkStream对象

2: TcpListener：此类也是微软基于Tcp封装类，用于监听服务端或客户端的连接请求，一旦有连接请求信息，立刻交给TcpClient的AcceptTcpClient方法捕获，Start方法用于开始监听

3: IPEndPonint：处理IP地址和端口的封装类

4:IPAddress：提供包含计算机在 IP 网络上的地址的工具类

6．使用NetworkStream的注意事项和局限性

抱歉到目前为止才开始介绍NetworkStream,我相信大家到这里在回过头去看第一节的作用时能够更多的领悟。前五节意在说明下NetworkStream背后那个必须掌握的知识点，这样才能在实际编程过程中很快上手，毕竟NetworkStream的工作环境和其他流有着很大的差别，再回到第一节关于NetworkStream的知识点，在使用时有几点必须注意

首先

1 再次强调NetworkStream是稳定的，面向连接的，所以它只适合TCP协议的环境下工作。所以一旦在UDP环境中，虽然编译不会报错，但是会跳出异常

2 我们可以通过NetworkStream简化Socket开发

3 如果要建立NetworkStream一个新的实例，则必须使用已经连接的Socket

4 NetworkStream 使用后不会自动关闭提供的socket，必须使用NetworkStream构造函数时指定Socket所有权（NetworkStream 的构造函数中设置）。

6 NetworkStream支持异步读写操作

NetworkStream的局限性

1. 可惜的是NetworkStream基于安全上的考虑不支持 Posion属性或Seek方法，寻找或改变流的位置，如果试图强行使用会报出NotSupport的异常
2. 支持传递数据的种类没有直接使用Socket来的多

7．NetworkStream的构造

1.NetworkStream (Socket)  为指定的 Socket 创建 NetworkStream 类的新实例

2.NetworkStream (Socket, Boolean ownsSocket)  用指定的 Socket 所属权为指定的 Socket，ownsSocket表示指示NetworkStream是否拥有该Socket

3.NetworkStream (Socket, FileAccess)  用指定的访问权限为指定的 Socket 创建[FileAccess](http://msdn.microsoft.com/zh-cn/library/4z36sx0f(v=vs.80)) 值的按位组合，这些值指定授予所提供的 Socket 上的 NetworkStream 的访问类型

4.NetworkStream (Socket, FileAccess, Boolean ownsSocket) 。

|  |
| --- |
| 对于NetworkStream构造函数的理解相信大家经过前文的解释也能够掌握了，但是有几点必须强调下  1如果用构造产生NetworkStream的实例，则必须使用连接的Socket  2 如果该NetworkStream拥有对Socket的所有权，则在使用NetworkStream的Close方法时会同时关闭Socket,否则关闭NetworkStream时不会关闭Socket  3, 能够创建对指定Socket带有读写权限的NetworkStream |

8．NetworkStream的属性

1. CanSeek :用于指示流是否支持查找，它的值始终为 false

2. DataAvailable 指示在要读取的 NetworkStream 上是否有可用的数据。一般来说通过判断这个属性来判断NetworkStream中是否有数据

3. Length:NetworkStream不支持使用Length属性，强行使用会发生NotSupportedException异常

4.Position: NetworkStream不支持使用Position属性，强行使用会发生NotSupportedException异常

9．NetworkStream的方法

同样，NetworkStream的方法大致重写或继承了Stream的方法

但是以下方法必须注意：

1 int Read(byte[] buffer,int offset,int size)：该方法将数据读入 buffer 参数并返回成功读取的字节数。如果没有可以读取的数据，则 Read 方法返回 0。Read 操作将读取尽可能多的可用数据，直至达到由 size 参数指定的字节数为止。如果远程主机关闭了连接并且已接收到所有可用数据，Read 方法将立即完成并返回零字节。

2 long Seek(long offset, SeekOrigin origin)：将流的当前位置设置为给定值。此方法当前不受支持，总是引发 NotSupportedException。

3  void Write(byte[] buffer, int offset,int size)：Write方法在指定的 offset 处启动，并将 buffer 内容中的 size 字节发送到网络。Write 方法将一直处于阻止状态(可以用异步解决)，直到发送了请求的字节数或引发 SocketException 为止。如果收到 SocketException，可以使用 SocketException.ErrorCode 属性获取特定的错误代码。

10．NetworkStream的简单示例

创建一个客户端向服务端传输图片的小示例，服务端一直监听客户端传来的图片信息

/// <summary>服务端监听客户端信息，一旦有发送过来的信息，便立即处理</summary>

class Program {

//全局TcpClient

static TcpClient client;

//文件流建立到磁盘上的读写流

static FileStream fs = new FileStream("E:\\abc.jpg", FileMode.Create);

//buffer

static int bufferlength = 200;

static byte[] buffer = new byte[bufferlength];

//网络流

static NetworkStream ns;

static void Main(string[] args){

ConnectAndListen();

}

static void ConnectAndListen() {

//服务端监听任何IP 但是端口号是80的连接

TcpListener listener = new TcpListener(IPAddress.Any,80);

//监听对象开始监听 listener.Start();

while(true){

Console.WriteLine("等待连接");

//线程会挂在这里，直到客户端发来连接请求

client = listener.AcceptTcpClient();

Console.WriteLine("已经连接");

//得到从客户端传来的网络流

ns = client.GetStream();

//如果网络流中有数据

if (ns.DataAvailable) {

//同步读取网络流中的byte信息

// do

// {

// ns.Read(buffer, 0, bufferlength);

//} while (readLength > 0);

//异步读取网络流中的byte信息

ns.BeginRead(buffer, 0, bufferlength, ReadAsyncCallBack, null);

}

}

}

/// <summary>异步读取</summary>

static void ReadAsyncCallBack(IAsyncResult result) {

int readCount;

//获得每次异步读取数量

readCount = client.GetStream().EndRead(result);

//如果全部读完退出，垃圾回收

if (readCount < 1) {

client.Close();

ns.Dispose();

fs.Dispose();

return;

}

//将网络流中的图片数据片段顺序写入本地

fs.Write(buffer, 0, 200);

//再次异步读取

ns.BeginRead(buffer, 0, 200, ReadAsyncCallBack, null);

}

}

 客户端先连接上服务端后在发送图片，注意如果是双向通信的话最好将客户端和服务端的项目设置为多个启动项便于调试

class Program {

/// <summary> 客户端</summary>

static void Main(string[] args) {

SendImageToServer("xxx.jpg");

}

static void SendImageToServer(string imgURl) {

if (!File.Exists(imgURl)) return;

//创建一个文件流打开图片

FileStream fs = File.Open(imgURl, FileMode.Open);

//声明一个byte数组接受图片byte信息

byte[] fileBytes = new byte[fs.Length];

using (fs){

//将图片byte信息读入byte数组中

fs.Read(fileBytes, 0, fileBytes.Length);

fs.Close();

}

//找到服务器的IP地址

IPAddress address = IPAddress.Parse("127.0.0.1");

//创建TcpClient对象实现与服务器的连接

TcpClient client = new TcpClient();

//连接服务器

client.Connect(address, 80);

using (client){

//连接完服务器后便在客户端和服务端之间产生一个流的通道

NetworkStream ns = client.GetStream();

using (ns) {

//通过此通道将图片数据写入网络流，传向服务器端接收

ns.Write(fileBytes, 0, fileBytes.Length);

}

}

}

}  
附件： 关于Socket的一个简单示例

服务器端建立服务并且循环监听

const int PORT = 80;

static Socket clientSocket;

static Socket client;

static void Main(string[] args) {

Thread thread=new Thread(new ThreadStart(SetUpBlockServer));

thread.Start();

}

/// <summary> 服务器端建立服务并且循环监听</summary>

static void SetUpBlockServer() {

//同样建立TcpListener 监听对象监听客户端传来的信息

TcpListener lis = new TcpListener(IPAddress.Any, PORT);

Console.WriteLine("正在监听任何端口号为80的任意IP的连接");

//启动监听 lis.Start();

while (true) {

//进程会挂起，知道客户端的socket发送连接请求

clientSocket = lis.AcceptSocket();

Console.WriteLine("{0}时刻接收到客户端的连接请求",DateTime.Now.ToString("G"));

//连接成功后发送给客户端信息

string testMessage = "连接成功";

clientSocket.Send(Encoding.Default.GetBytes(testMessage));

}

}

 客户端连接服务端的请求，和循环监听服务端传来的信息

class Program {

//端口

const int PORT = 80;

static void Main(string[] args){

Connect("127.0.0.1");

}

/// <summary>建立与服务器端的异步连接</summary>

static void Connect(string server) {

//建立一个socket用来和服务的Socket进行通信

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

//获得服务器IP地址

IPAddress ipAddress = IPAddress.Parse(server);

//获得服务器端口

IPEndPoint point = new IPEndPoint(ipAddress, PORT);

//开始异步连接，注意将socket放入异步方法的参数中，提供给回调方法使用

socket.BeginConnect(point, new AsyncCallback(ConnectCallBack), socket);

Thread.Sleep(10000000);

}

/// <summary>异步连接后的callback事件 </summary>

static void ConnectCallBack(IAsyncResult result) {

try{

//建立一个接受信息的byte数组

byte[] receiveBuffer = new byte[4098];

//从回调参数中获取上面Conntect方法中的socket对象

Socket socket = result.AsyncState as Socket;

//判断是否和服务端的socket建立连接

if (socket.Connected) {

//开始 异步接收服务端传来的信息，同样将socket传入回调方法的参数中

socket.BeginReceive(receiveBuffer, 0, receiveBuffer.Length, SocketFlags.None, new AsyncCallback(ReceiveCallBack), socket);

}else { ConnectCallBack(result); }

}

catch {

Console.WriteLine("连接出错");

}

}

/// <summary> 一旦服务器发送信息，则会触发回调方法</summary>

static void ReceiveCallBack(IAsyncResult result) {

byte[] receiveBuffer = new byte[4098];

Socket socket = result.AsyncState as Socket;

//读取从服务器端传来的数据，EndReceive是关闭异步接收方法，同时读取数据

int count = socket.EndReceive(result);

if (count > 0) {

try{

//接受完服务端的数据后的逻辑

}

catch { }

}

// 递归监听服务器端是否发来信息，一旦服务器再次发送信息，客户端仍然可以接收到

socket.BeginReceive(receiveBuffer, 0, receiveBuffer.Length, SocketFlags.None, ReceiveCallBack,socket);

}

}