网络

siki 微信：devsiki QQ:804632564

WebClient概述

从MSDN中我们可以得知，WebClient的作用就是“Provides common methods for sending data to and receiving data from a resource identified by a URI.”也就是说我们可以通过这个类去访问与获取网络上的资源文件。

WebClient类不能被继承,我们可以通过WebRequest和WebResponse这两个类来处理向URI标示的资源和获取数据了。这两个类功能挺强大的，但不足之处的是利用WebRequest和WebResponse时设置过于复杂，使用起来颇为费劲。而WebClient可以理解为对WebRequest和WebResponse等协作的封装。它使人们使用起来更加简单方便，然后它也有先天不足的地方。那就是缺少对cookies/session的支持,用户无法对是否自动url转向的控制，还有就是缺少对代理服务器的支持等等，不过我们可以通过重写WebClient的一些方法来实现这些功能。

WebClient的函数与基本用法

WebClient提供四种将数据上载到资源的方法：

1. OpenWrite 返回一个用于将数据发送到资源的 Stream。
2. UploadData 将字节数组发送到资源并返回包含任何响应的字节数组。
3. UploadFile 将本地文件发送到资源并返回包含任何响应的字节数组。
4. UploadValues 将 NameValueCollection 发送到资源并返回包含任何响应的字节数组。

另外WebClient还提供三种从资源下载数据的方法：

1. DownloadData 从资源下载数据并返回字节数组。
2. DownloadFile 从资源将数据下载到本地文件。
3. OpenRead 从资源以 Stream 的形式返回数据。

WebClient与其他网络相关类的区别

WebClient和HttpWebRequst是用来获取数据的2种方式，一般而言，WebClient更倾向于“按需下载”，事实上掌握它也是相对容易的，而HttpWebRequst则允许你设置请求头或者对内容需要更多的控制，后者有点类似于form中的submit。虽然两者都是异步请求事件，但是WebClient是基于事件的异步，而HttpWebRequst是基于代理的异步编程。

WebClient使用范例

```
private void button1_Click(object sender, RoutedEventArgs e)
        {
            //通过WebClient方式去获取资源文件
            Uri uri = new Uri("http://localhost:2052/Images/cnblogs.png", UriKind.Absolute);
            WebClient webClient = new WebClient();
            webClient.OpenReadAsync(uri);
            webClient.OpenReadCompleted += new OpenReadCompletedEventHandler(client_OpenReadCompleted);
        }

        void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
        {
            Stream stream = e.Result;
            BitmapImage bitmap = new BitmapImage();
            bitmap.SetSource(stream);
            this.image1.Source = bitmap;
        }
```

WebRequest和WebResponse入门案例

```
public Form1()
{
WebRequest wrq = WebRequest.Create("");
WebResponse wrs = wrq.GetResponse();
Stream strm = wrs.GetResponseStream();
StreamReader sr = new StreamReader(strm);
while ( (line = sr.ReadLine()) != null)
{
Console.WriteLine(line);
}
strm.Close();
}
```

案例分析

WebRequest类是支持不同网络协议的类层次结构的一部分，为了给请求类型接收一个对正确对象的引用，需要一个工厂(factory)机制。WebRequest.Create()方法会为给定的协议创建合适的对象。WebRequest类代表要给某个URI发送信息的请求，URI作为参数传送给Create()方法。WebResponse类代表从服务器获取的数据。调用WebRequest.GetResponse()方法，实际上是把请求发送给Web服务器，创建一个Response对象，检查返回的数据。

HTTP标题信息

HTTP协议的一个重要方面就是能够利用请求和响应数据流发送扩展的标题信息。标题信息可以包括cookies、以及发送请求的特定浏览器(用户代理)的一些详细信息。

WebRequest类和WebResponse类提供了读取标题信息的一些支持。而两个派生的类HttpWebRequest类和HttpWebResponse类提供了其他HTTP特定的信息。用HTTP URI创建WebRequest会生成一个HttpWebRequest对象实例。因为HttpWebRequest派生自WebRequest，可以在需要WebRequest的任何地方使用新实例。

还可以把实例的类型强制转换为HttpWebRequest引用，访问HTTP协议特定的属性。同样，在使用HTTP时，GetResponse()方法调用会返回WebResponse引用，也可以进行一个简单的强制转换，以访问HTTP特定的特性。

```
WebRequest wrq = WebRequest.Create("");
HttpWebRequest hwrq = (HttpWebRequest)wrq;
Console.WriteLine("Request Timeout (ms) = " + wrq.Timeout);
Console.WriteLine("Request Keep Alive = " + hwrq.KeepAlive);
Console.WriteLine("Request AllowAutoRedirect = " + hwrq.AllowAutoRedirect);
```

HTTP标题信息

imeout属性的单位是毫秒，其默认值是100 000。可以设置这个属性，以控制WebRequest对象在产生WebException之前要在响应中等待多长时间。可以检查属性WebException.Status，看看产生异常的原因。这个枚举类型包括超时的状态码、连接失败、协议错误等。

KeepAlive属性是对HTTP协议的特定扩展，所以可以通过HttpWebRequest引用访问这个属性。该属性允许多个请求使用同一个连接，在后续的请求中节省关闭和重新打开连接的时间。其默认值为true。

AllowAutoRedirect属性也是专用于HttpWebRequest类的，使用这个属性可以控制Web请求是否应自动跟随Web服务器上的重定向响应。其默认值也是true。如果只允许有限的重定向，可以把HttpWebRequest的MaximumAutomaticRedirections属性设置为想要的数值。

HTTP标题信息

请求和响应类把大多数重要的标题显示为属性，也可以使用Headers属性本身显示标题的总集合。在GetResponse()方法调用的后面添加如下代码

```
WebRequest wrq = WebRequest.Create("");
WebResponse wrs = wrq.GetResponse();
WebHeaderCollection whc = wrs.Headers;
for(int i = 0; i < whc.Count; i++)
{
Console.WriteLine("Header " + whc.GetKey(i) + " : " + whc[i]);
}
```

异步页面请求

WebRequest类的另一个特性就是可以异步请求页面。由于在给主机发送请求到接收响应之间有很长的延迟，因此，异步请求页面就显得比较重要。像WebClient.DownloadData()和WebRequest.GetResponse()等方法，在响应没有从服务器回来之前，是不会返回的。

如果不希望在那段时间中应用程序处于等待状态，可以使用BeginGetResponse() 方法和EndGetResponse()方法，BeginGetResponse()方法可以异步工作，并立即返回。在底层，运行库会异步管理一个后台线程，从服务器上接收响应。

BeginGetResponse() 方法不返回WebResponse对象，而是返回一个执行IAsyncResult接口的对象。使用这个接口可以选择或等待可用的响应，然后调用EndGetResponse()搜集结果。

也可以把一个回调委托发送给BeginGetResponse()方法。该回调委托的目的地是一个返回类型为void并把IAsyncResult引用作为参数的方法，当工作线程完成了搜集响应的任务后，运行库就调用该回调委托，通知用户工作已完成。如下面的代码所示，在回调方法中调用EndGetResponse()可以接收WebResponse对象：

```
public Form1()
{
InitializeComponent();
WebRequest wrq = WebRequest.Create("");
wrq.BeginGetResponse(new AsyncCallback(OnResponse), wrq);
}
```

实用工具类-URI

Uri和UriBuilder是System命名空间下的两个类。UriBuilder可以通过给定的字符串，从而构建一个URI，而Uri类可以分析，组合和比较URI。

1,创建Uri类

```
Uri uri = new Uri("http://www.microsoft.com/somefolder/somefile.htm?order=true");
```

2，分析Uri取得一些属性

```
string query = uri.Query;// ?order=true 得到参数部分
string absPath = uri.AbsolutePath; //   /somefoler/somefile.htm 得到路径部分
string scheme = uri.Scheme; // http 得到协议
int port = uri.Port ; // 80 访问端口
string host = uri.Host;     // www.microsoft.com
bool isDefaultPort = uri.IsDefaultPort;
```

实用工具类-UriBuilder

```
UriBuilder builder = new UriBuilder("http","www.microsoft.com",80,"somefolder/somefile.htm");
```

也可以单独给每个属性部分赋值

```
builder.Scheme = "http";
builder.Host="www.microsoft.com";
builder.Port = 80;
builder.Path="somefolder/somefile.htm";

//通过builder构建uri
Uri completeUri = builder.Uri;
```

实用工具类-IPAddress

创建IP地址

```
IPAddress ipAddress = IPAddress.Parse("234.34.5.3");
byte[] address = ipAddress.GetAddressBytes();//得到四个位置上的具体值
string ipString = ipAddress.ToString();//得到ip的字符串
```

实用工具类-Dns和IPHostEntry

```
IPHostEntry ipentry = Dns.Resolve("www.baidu.com");
IPHostEntry ipentry = Dns.GetHostByAddress("23.34.3.43");
```

通过域名或者ip地址获取主机信息

Socket(套接字)编程（Tcp）

1.基于Tcp协议的Socket通讯类似于B/S架构，面向连接，但不同的是服务器端可以向客户端主动推送消息。

使用Tcp协议通讯需要具备以下几个条件：

(1).建立一个套接字(Socket)

(2).绑定服务器端IP地址及端口号--服务器端

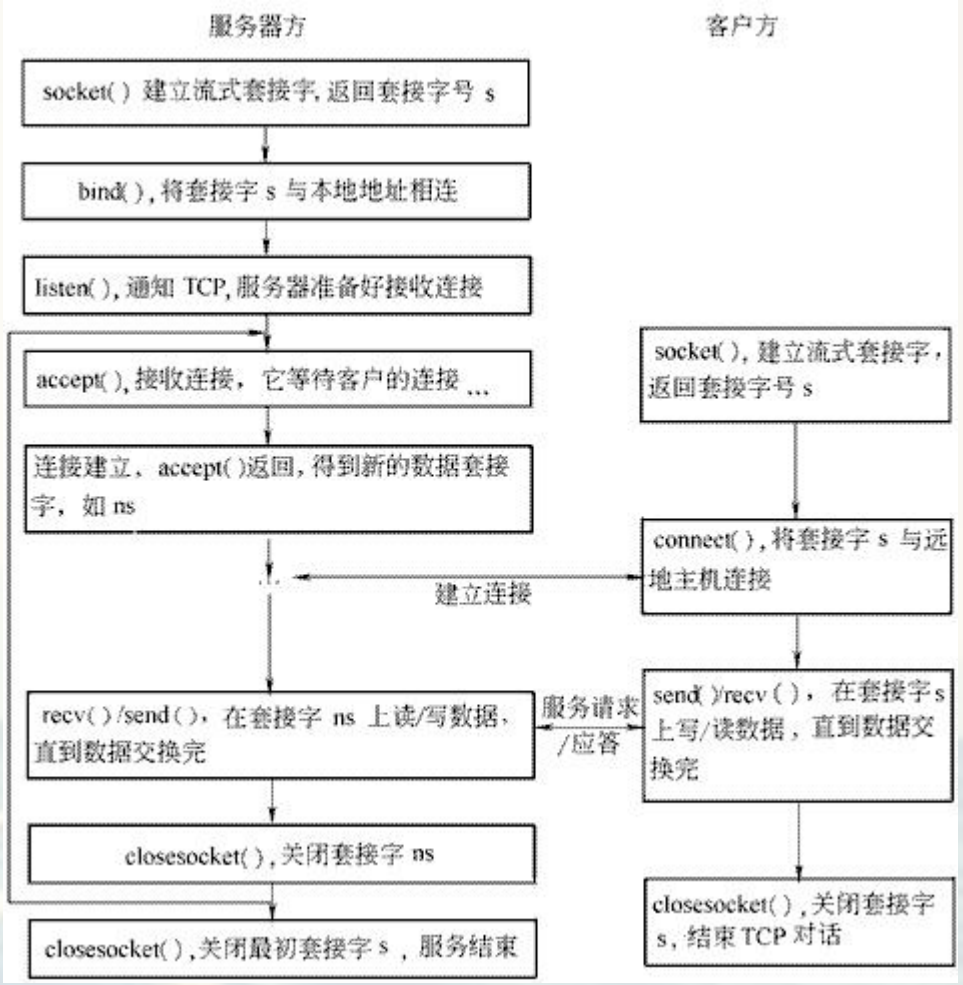
(3).利用Listen()方法开启监听--服务器端

(4).利用Accept()方法尝试与客户端建立一个连接--服务器端

(5).利用Connect()方法与服务器建立连接--客户端

(5).利用Send()方法向建立连接的主机发送消息

(6).利用Recive()方法接受来自建立连接的主机的消息(可靠连接)



TcpServer

```
public static void TcpServer(IPEndPoint serverIP)
        {
            Console.WriteLine("客户端Tcp连接模式");
            Socket tcpServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            tcpServer.Bind(serverIP);
            tcpServer.Listen(100);
            Console.WriteLine("开启监听...");
            new Thread(() =>
            {
                while (true)
                {
                    try
                    {
                        TcpRecive(tcpServer.Accept());
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(string.Format("出现异常：{0}", ex.Message));
                        break;
                    }
                }
            }).Start();
            Console.WriteLine("\n\n输入\"Q\"键退出。");
            ConsoleKey key;
```

TcpRecive

```
public static void TcpRecive(Socket tcpClient)
        {
            new Thread(() =>
            {
                while (true)
                {
                    byte[] data = new byte[1024];
                    try
                    {
                        int length = tcpClient.Receive(data);
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(string.Format("出现异常：{0}", ex.Message));
                        break;
                    }
                    Console.WriteLine(string.Format("收到消息：{0}", Encoding.UTF8.GetString(data)));
                    string sendMsg = "收到消息！";
                    tcpClient.Send(Encoding.UTF8.GetBytes(sendMsg));
                }
            }).Start();
        }
```

TcpClient

```
public static void TcpServer(IPEndPoint serverIP)
      {
          Console.WriteLine("客户端Tcp连接模式");
          Socket tcpClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
          try
          {
              tcpClient.Connect(serverIP);
          }
          catch (SocketException e)
          {
              Console.WriteLine(string.Format("连接出错：{0}", e.Message));
              Console.WriteLine("点击任何键退出！");
              Console.ReadKey();
              return;
          }
          Console.WriteLine("客户端：client-->server");
          string message = "我上线了...";
          tcpClient.Send(Encoding.UTF8.GetBytes(message));
          Console.WriteLine(string.Format("发送消息：{0}", message));
          new Thread(() =>
          {
              while (true)
              {
                  byte[] data = new byte[1024];
```

Socket(套接字)编程（Udp）

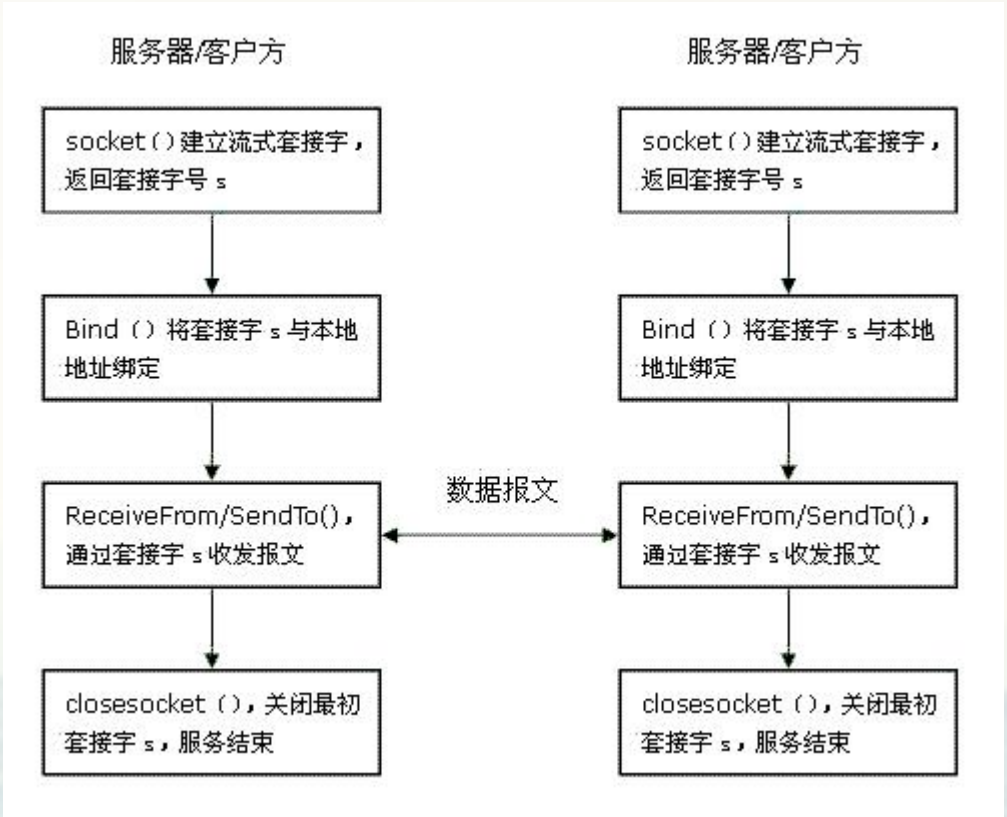
基于Udp协议是无连接模式通讯，占用资源少，响应速度快，延时低。至于可靠性，可通过应用层的控制来满足。(不可靠连接)

(1).建立一个套接字(Socket)

(2).绑定服务器端IP地址及端口号--服务器端

(3).通过SendTo()方法向指定主机发送消息(需提供主机IP地址及端口)

(4).通过ReciveFrom()方法接收指定主机发送的消息(需提供主机IP地址及端口)



UdpServer

```
public static void UdpServer(IPEndPoint serverIP)
        {
            Console.WriteLine("客户端Udp模式");
            Socket udpServer = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            udpServer.Bind(serverIP);
            IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 0);
            EndPoint Remote = (EndPoint)ipep;
            new Thread(() =>
            {
                while (true)
                {
                    byte[] data = new byte[1024];
                    try
                    {
                        int length = udpServer.ReceiveFrom(data, ref Remote);//接受来自服务器的数据
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(string.Format("出现异常：{0}", ex.Message));
                        break;
                    }
                    Console.WriteLine(string.Format("{0} 收到消息：{1}", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),
Encoding.UTF8.GetString(data)));
                    string sendMsg = "收到消息！";
```

UdpClient

```
public static void UdpClient(IPEndPoint serverIP)
        {
            Console.WriteLine("客户端Udp模式");
            Socket udpClient = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            string message = "我上线了...";
            udpClient.SendTo(Encoding.UTF8.GetBytes(message), SocketFlags.None, serverIP);
            Console.WriteLine(string.Format("发送消息：{0}", message));
            IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
            EndPoint Remote = (EndPoint)sender;
            new Thread(() =>
            {
                while (true)
                {
                    byte[] data = new byte[1024];
                    try
                    {
                        int length = udpClient.ReceiveFrom(data, ref Remote);//接受来自服务器的数据
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(string.Format("出现异常：{0}", ex.Message));
                        break;
                    }
                    Console.WriteLine(string.Format("{0} 收到消息：{1}", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),
```

TCP和UDP的区别

TCP协议和UDP协议连接过程的区别

1.基于连接与无连接；

2.对系统资源的要求（TCP较多，UDP少）；

3.UDP程序结构较简单；

4.流模式与数据报模式 ；

5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

socket - TcpClient,TcpListener,UdpClient

应用程序可以通过 TCPClient、TCPListener 和 UDPClient 类使用传输控制协议 (TCP) 和用户数据文报协议 (UDP) 服务。这些协议类建立在 System.Net.Sockets.Socket 类的基础之上，负责数据传送的细节。(也就是说TCPClient、TCPListener 和 UDPClient 类是用来简化Socket)

TcpClient 和 TcpListener 使用 NetworkStream 类表示网络。使用 GetStream 方法返回网络流，然后调用该流的 Read 和 Write 方法。NetworkStream 不拥有协议类的基础套接字，因此关闭它并不影响套接字。

UdpClient 类使用字节数组保存 UDP 数据文报。使用 Send 方法向网络发送数据，使用 Receive 方法接收传入的数据文报。

TcpListener

TcpListener 类提供一些简单方法，用于在阻止同步模式下侦听和接受传入连接请求。可使用 TcpClient 或 Socket 来连接 TcpListener。可使用 IPEndPoint、本地 IP 地址及端口号或者仅使用端口号，来创建 TcpListener。可以将本地 IP 地址指定为Any，将本地端口号指定为 0（如果希望基础服务提供程序为您分配这些值）。如果您选择这样做，可在连接套接字后使用LocalEndpoint 属性来标识已指定的信息。

Start 方法用来开始侦听传入的连接请求。Start 将对传入连接进行排队，直至您调用 Stop 方法或它已经完成 MaxConnections 排队为止。可使用 AcceptSocket 或 AcceptTcpClient 从传入连接请求队列提取连接。这两种方法将阻止。如果要避免阻止，可首先使用 Pending 方法来确定队列中是否有可用的连接请求。

调用 Stop 方法来关闭 TcpListener。

TcpClient

TcpClient 类提供了一些简单的方法，用于在同步阻止模式下通过网络来连接、发送和接收流数据。为使 TcpClient 连接并交换数据，使用 TCP ProtocolType 创建的 TcpListener 或 Socket 必须侦听是否有传入的连接请求。可以使用下面两种方法之一连接到该侦听器：

（1）创建一个 TcpClient，并调用三个可用的 Connect 方法之一。

（2）使用远程主机的主机名和端口号创建 TcpClient。此构造函数将自动尝试一个连接。

给继承者的说明要发送和接收数据，请使用 GetStream 方法来获取一个 NetworkStream。调用 NetworkStream 的 Write 和Read 方法与远程主机之间发送和接收数据。使用 Close 方法释放与 TcpClient 关联的所有资源。

UdpClient

UdpClient 类提供了一些简单的方法，用于在阻止同步模式下发送和接收无连接 UDP 数据报。因为 UDP 是无连接传输协议，所以不需要在发送和接收数据前建立远程主机连接。但您可以选择使用下面两种方法之一来建立默认远程主机：

使用远程主机名和端口号作为参数创建 UdpClient 类的实例。

创建 UdpClient 类的实例，然后调用 Connect 方法。

可以使用在 UdpClient 中提供的任何一种发送方法将数据发送到远程设备。使用 Receive 方法可以从远程主机接收数据。

UdpClient 方法还允许发送和接收多路广播数据报。使用 JoinMulticastGroup 方法可以将 UdpClient 预订给多路广播组。使用 DropMulticastGroup 方法可以从多路广播组中取消对 UdpClient 的预订。

关注公众号

发布最新的视频，文章和教学资源

THANK

@siki 微信：devsiki QQ:804632564